Towards Privacy-Preserving Relational Data Synthesis via Probabilistic Relational Models

Malte Luttermann[1], Ralf Möller[2], and Mattis Hartwig[1,3]

[1] German Research Center for Artificial Intelligence (DFKI), Lübeck
{malte.luttermann,mattis.hartwig}@dfki.de
[2] Institute for Humanities-Centered Artificial Intelligence, University of Hamburg
ralf.moeller@uni-hamburg.de
[3] singularIT GmbH, Leipzig

Abstract. Probabilistic relational models provide a well-established formalism to combine first-order logic and probabilistic models, thereby allowing to represent relationships between objects in a relational domain. At the same time, the field of artificial intelligence requires increasingly large amounts of relational training data for various machine learning tasks. Collecting real-world data, however, is often challenging due to privacy concerns, data protection regulations, high costs, and so on. To mitigate these challenges, the generation of synthetic data is a promising approach. In this paper, we solve the problem of generating synthetic relational data via probabilistic relational models. In particular, we propose a fully-fledged pipeline to go from relational database to probabilistic relational model, which can then be used to sample new synthetic relational data points from its underlying probability distribution. As part of our proposed pipeline, we introduce a learning algorithm to construct a probabilistic relational model from a given relational database.

Keywords: probabilistic graphical models; relational models; synthetic data.

1 Introduction

Probabilistic relational models such as parametric factor graphs (PFGs) combine first-order logic and probabilistic models and thereby provide an adequate formalism to represent relationships between objects in a relational domain. PFGs compactly encode a full joint probability distribution over a set of random variables (randvars) and hence allow to sample new relational data points from the encoded underlying full joint probability distribution. The generated synthetic relational data samples then follow the underlying full joint probability distribution and might be used for various purposes. Common applications of synthetic data include for example training machine learning models or sharing data without violating the privacy of individuals [8,27,35]. Another application could be to bootstrap PFG learning (using the given database to learn a PFG, which is then applied to generate additional synthetic relational data points to learn another PFG on the basis of a larger data set and then possibly repeating the procedure).

In this paper, we solve the problem of applying probabilistic relational models (more specifically, PFGs) to generate synthetic relational (i.e., multi-table) data from a conceptual point of view.

Previous Work. While there is, to the best of our knowledge, no previous work on the generation of synthetic relational data via probabilistic relational models, there exists previous work on learning and sampling models that combine probabilistic models and first-order logic. In particular, so-called Markov logic networks (MLNs) [30] have extensively been studied. An MLN is another formalism combining probabilistic models and first-order logic, allowing for probabilistic reasoning on a first-order level. Even though a vast amount of algorithms to learn an MLN from data has emerged (see, e.g., [4,5,16,17,18,19,22,26,31]), there is a lack of prior work on generating synthetic relational data via MLNs or other probabilistic relational models. Moreover, while sampling algorithms for MLNs have been developed [33], these sampling algorithms are used for (approximate) query answering and are not applied to synthetic data generation, where the requirements of the sampling algorithm slightly differ—that is, drawing a new data sample should yield a value for every column in the given database.

Most of the existing work on synthetic data generation focuses on the generation of single-table synthetic data [10]. While many popular approaches are based on generative adversarial networks [34], there are also approaches building on probabilistic graphical models such as Bayesian networks [13]. Both for approaches based on generative adversarial networks and probabilistic graphical models, differential privacy guarantees have been investigated [3,6,9,35]. More recently, there has also been work on the generation of synthetic relational data under differential privacy [7]. However, none of these approaches makes use of the advantages of first-order probabilistic models.

Our Contributions. In this paper, we introduce the first architecture that combines PFGs and the generation of synthetic relational data. Our main contribution is a fully-fledged pipeline from relational database to PFG, from which we can then sample new realistic synthetic data points. We further present a learning algorithm to obtain a PFG (that is, both the graph structure and the parameters of the PFG) from a relational database. Our proposed architecture exploits the advantages of PFGs, including that (i) PFGs are able to effectively encode the relationships between objects in a relational domain, (ii) PFGs provide an explainable model that can also be used for, e.g., probabilistic inference [32] and causal inference [24] on a first-order level, and (iii) PFGs naturally abstract from individuals by grouping indistinguishable objects, which yields a promising foundation for differential privacy guarantees.

Structure of this Paper. The remainder of this paper is structured as follows. We begin by introducing necessary background information and notations. Afterwards, we propose an architecture to first learn a PFG from a relational database and then employ the PFG to generate new synthetic relational data points, which might be used for arbitrary applications. We then go through the

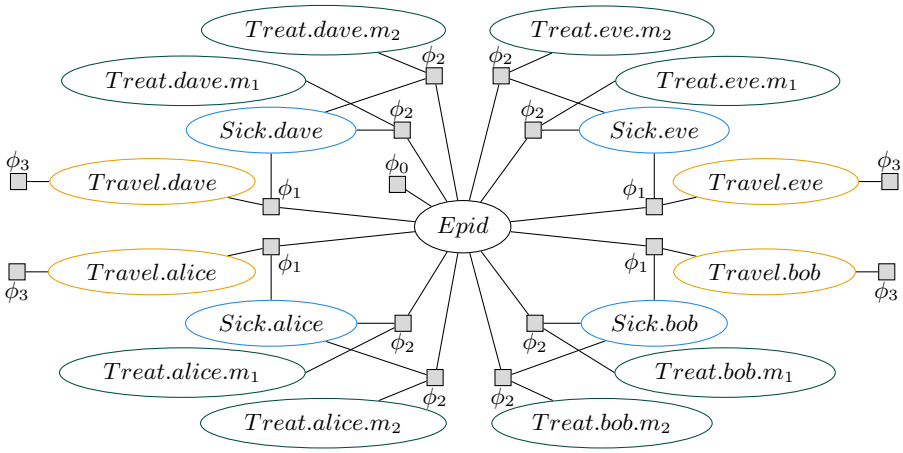


Fig. 1: An FG encoding a full joint probability distribution for an epidemic example [14]. We omit the input-output pairs of the factors for brevity.

individual steps of the proposed architecture in detail, using a small comprehensive example for guidance, before we conclude.

2 Preliminaries

We start with the definition of factor graphs (FGs) as propositional probabilistic models and then continue to introduce PFGs, which combine first-order logic with probabilistic models. An FG is an undirected propositional probabilistic model to compactly encode a full joint probability distribution over a set of randvars [11,21]. Similar to a Bayesian network [28], an FG factorises a full joint probability distribution into a product of factors.

Definition 1 (Factor Graph). *An* FG $G = (\boldsymbol{V}, \boldsymbol{E})$ *is an undirected bipartite graph with node set* $\boldsymbol{V} = \boldsymbol{R} \cup \boldsymbol{\Phi}$ *where* $\boldsymbol{R} = \{R_1, \ldots, R_n\}$ *is a set of variable nodes (randvars) and* $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$ *is a set of factor nodes (functions). The term* $\text{range}(R_i)$ *denotes the possible values of a randvar* R_i*. There is an edge between a variable node* R_i *and a factor node* ϕ_j *in* $\boldsymbol{E} \subseteq \boldsymbol{R} \times \boldsymbol{\Phi}$ *if* R_i *appears in the argument list of* ϕ_j*. A factor is a function that maps its arguments to a positive real number, called potential. The semantics of* G *is given by*

$$P_G = \frac{1}{Z} \prod_{j=1}^{m} \phi_j(\mathcal{A}_j)$$

with Z *being the normalisation constant and* $\mathcal{A}_j$ *denoting the randvars connected to* ϕ_j *(that is, the arguments of* ϕ_j*).*

Example 1. Take a look at the FG presented in Fig. 1, which represents an epidemic example with four persons *alice*, *bob*, *dave*, and *eve* as well as two possible medications m_1 and m_2 for treatment. For each person, there are two Boolean randvars (that is, randvars having a Boolean range) $Sick$ and $Travel$, indicating whether the person is sick and travels, respectively. Moreover, there is another Boolean randvar $Treat$ for each combination of person and medication, specifying whether the person is treated with the medication. The Boolean randvar *Epid* states whether an epidemic is present.

Next, we define PFGs, first introduced by Poole [29], which combine probabilistic models and first-order logic. PFGs use logical variables (logvars) as parameters to represent sets of indistinguishable randvars. Each set of indistinguishable randvars is represented by a parameterised randvar (PRV).

Definition 2 (Parameterised Random Variable). *Let $\boldsymbol{R}$ be a set of randvar names, $\boldsymbol{L}$ a set of logvar names, and $\boldsymbol{D}$ a set of constants. All sets are finite. Each logvar L has a domain* $\mathrm{dom}(L) \subseteq \boldsymbol{D}$. *A* constraint *is a tuple $(\mathcal{X}, C_{\mathcal{X}})$ of a sequence of logvars $\mathcal{X} = (X_1, \ldots, X_n)$ and a set $C_{\mathcal{X}} \subseteq \times_{i=1}^{n} \mathrm{dom}(X_i)$. The symbol $\top$ for C marks that no restrictions apply, i.e., $C_{\mathcal{X}} = \times_{i=1}^{n} \mathrm{dom}(X_i)$. A* PRV *$R(L_1, \ldots, L_n)$, $n \geq 0$, is a syntactical construct of a randvar $R \in \boldsymbol{R}$ possibly combined with logvars $L_1, \ldots, L_n \in \boldsymbol{L}$ to represent a set of randvars. If $n = 0$, the PRV is parameterless and forms a propositional randvar. A PRV A (or logvar L) under constraint C is given by $A_{|C}$ ($L_{|C}$), respectively. We may omit $|\top$ in $A_{|\top}$ or $L_{|\top}$. The term* $\mathrm{range}(A)$ *denotes the possible values of a PRV A. An* event *$A = a$ denotes the occurrence of PRV A with range value $a \in \mathrm{range}(A)$.*

Example 2. Consider $\boldsymbol{R} = \{Epid, Travel, Sick, Treat\}$ and $\boldsymbol{L} = \{P, M\}$ with $\mathrm{dom}(P) = \{alice, bob, dave, eve\}$ (patients), $\mathrm{dom}(M) = \{m_1, m_2\}$ (medications), combined into Boolean PRVs *Epid*, $Travel(P)$, $Sick(P)$, and $Treat(P, M)$.

A parametric factor (parfactor) describes a function, mapping argument values to positive real numbers, of which at least one is non-zero.

Definition 3 (Parfactor). *Let Φ denote a set of factor names. We denote a* parfactor *g by $\phi(\mathcal{A})_{|C}$ with $\mathcal{A} = (A_1, \ldots, A_n)$ being a sequence of PRVs, ϕ: $\times_{i=1}^{n} \mathrm{range}(A_i) \mapsto \mathbb{R}^+$ being a function with name $\phi \in \Phi$ mapping argument values to a positive real number called* potential*, and C being a constraint on the logvars of $\mathcal{A}$. We may omit $|\top$ in $\phi(\mathcal{A})_{|\top}$. The term $lv(Y)$ refers to the logvars in some element Y, a PRV, a parfactor, or sets thereof. The term $gr(Y_{|C})$ denotes the set of all instances (groundings) of Y with respect to constraint C.*

A PFG is then built from a set of parfactors $\{g_1, \ldots, g_m\}$.

Definition 4 (Parametric Factor Graph). *A* PFG *$G = (\boldsymbol{V}, \boldsymbol{E})$ is a bipartite graph with node set $\boldsymbol{V} = \boldsymbol{A} \cup \boldsymbol{G}$ where $\boldsymbol{A} = \{A_1, \ldots, A_n\}$ is a set of PRVs and $\boldsymbol{G} = \{g_1, \ldots, g_m\}$ is a set of parfactors as well as edge set $\boldsymbol{E} \subseteq \boldsymbol{A} \times \boldsymbol{G}$. A PRV A_i and a parfactor g_j are connected via an edge in G (i.e., $\{A_i, g_j\} \in \boldsymbol{E}$) if A_i appears in the argument list of g_j. The semantics of G is given by grounding and*

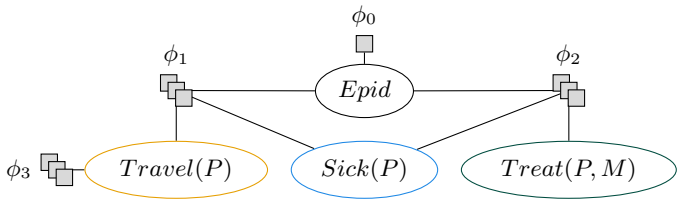


Fig. 2: A PFG encoding a full joint probability distribution for the epidemic example from Fig. 1. We omit the input-output pairs of the parfactors for brevity.

building a full joint distribution. With Z as the normalisation constant and $\mathcal{A}_k$ denoting the randvars connected to ϕ_k, G represents the full joint distribution

$$P_G = \frac{1}{Z} \prod_{g_j \in \boldsymbol{G}} \prod_{\phi_k \in gr(g_j)} \phi_k(\mathcal{A}_k).$$

Example 3. Figure 2 shows a PFG $G = \{g_i\}_{i=0}^{3}$ with $g_0 = \phi_0(Epid)_{|\top}$, $g_1 = \phi_1(Travel(P), Sick(P), Epid)_{|\top}$, $g_2 = \phi_2(Treat(P, M), Sick(P), Epid)_{|\top}$, and $g_3 = \phi_2(Travel(P))_{|\top}$. Grounding G yields again the FG shown in Fig. 1 (assuming that the domains of the logvars are defined as in Ex. 2).

Note that the definition of PFGs also includes FGs, as every FG is a PFG containing only parameterless randvars. Compared to an FG, a PFG abstracts from individuals by grouping identically behaving objects using logvars. While the introduction of logvars increases the expressiveness of the model (e.g., to encode relationships between groups of objects), grouping identically behaving individuals can also yield significant speed-ups during probabilistic inference.

In the upcoming section, we introduce our proposed architecture to solve the problem of learning a PFG from a given relational database to then generate synthetic relational data according to its underlying probability distribution.

3 Proposed Architecture

In this section, we provide an overview of the general architecture to generate synthetic relational data from a relational database using a PFG. We first take a look at the steps involved in the synthetic data generation approach and afterwards continue to investigate each of the steps in more detail.

An overview of the architecture of our proposed architecture is depicted in Fig. 3. The whole process consists of three primary steps, which can again be decomposed into various subroutines. The three primary steps consist of (i) constructing a propositional FG, (ii) transforming the propositional FG into a PFG, and (iii) sampling from the PFG to generate new synthetic relational data. Besides the generated synthetic data, the PFG is also a valuable output of the architecture (indicated by the + sign), as it can be used for various tasks such as probabilistic inference, causal inference, or bootstrap PFG learning, for example.

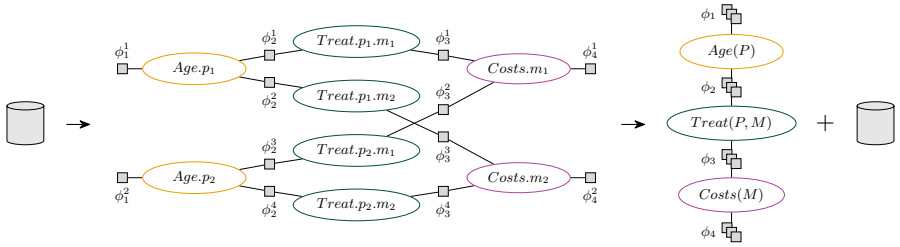


Fig. 3: Architecture overview of the general architecture to generate synthetic relational data from a given relational database using a PFG.

In the following, we illustrate each of the three steps at a small comprehensive example for guidance. Before we take a look at the steps involved in the proposed architecture, we briefly introduce the notion of an entity-relationship (ER) model, which allows us to describe a relational database $\mathcal{D}$.

Definition 5 (Entity-Relationship Model). *We define an* ER model *as a tuple* $(\mathcal{E}, \mathcal{R})$ *where* $\mathcal{E} = \{E_1, \dots, E_\ell\}$ *denotes a set of entity classes and* $\mathcal{R} = \{R_1, \dots, R_k\}$ *is a set of relationship classes. Each entity or relationship class* $B \in \mathcal{E} \cup \mathcal{R}$ *can have a set of attributes attached to it, which is denoted by* $\mathcal{A}(B)$.

Example 4. Consider the ER model depicted in Fig. 4a, which consists of the entity classes $\mathcal{C} = \{Patient, Medication\}$ and the relationship classes $\mathcal{R} = \{Treat\}$. The ER model further contains the attributes $\mathcal{A}(Patient) = \{Age\}$ and $\mathcal{A}(Medication) = \{Costs\}$.

It is generally possible to have cyclic relationships and to impose cardinality constraints on the relationships, which we omit for brevity in this paper. We next take a closer look at the individual steps involved in our proposed architecture.

3.1 Construction of a Propositional Factor Graph

While there are learning algorithms for first-order probabilistic models such as MLNs, to the best of our knowledge, there is currently no approach to directly learn a PFG from a given relational database. However, there are well-known approaches to learn an FG from the given data [1] and an FG can be transformed into a PFG by running the so-called advanced colour passing (ACP) algorithm [23]. We therefore propose to first learn a propositional model, that is, an FG G, from the given relational database and afterwards run the ACP algorithm on G to transform G into a PFG entailing equivalent semantics as G.

While such an approach seems straightforward at first glance, there are a few challenges to overcome. A major challenge is that applying a standard learning algorithm to obtain an FG from data does not fit our setting because standard learning algorithms do not include randvars and factors for individual objects into the FG. In other words, the relational structure of the data is neglected, which we illustrate in the upcoming example.

(a)

PatientId	Age
alice	≥ 18
bob	≥ 18
charlie	≥ 18
dave	< 18
eve	< 18

PatientId	MedicationId
alice	*myalept*
alice	*danyelza*
bob	*paracetamol*
charlie	*ibuprofen*
eve	*eliquis*

MedicationId	Costs
myalept	*high*
danyelza	*high*
paracetamol	*low*
ibuprofen	*low*
eliquis	*high*

(b)

Fig. 4: A small toy example for a relational database of patients and the medications they take. Figure 4a shows an ER model consisting of entities *Patient* and *Medication* with attributes *Age* and *Costs*, respectively. The entities *Patient* and *Medication* are connected by a relation *Treat*. We omit cardinalities for brevity. Figure 4b displays exemplary data for a relational database following the structure specified by the ER model from Fig. 4a.

Example 5. Take a look at a simple toy example database containing information on patients and the medications they take, as depicted in Fig. 4. For simplicity, each patient only has a single attribute *Age* which can either be < 18 or ≥ 18 and each medication has a single attribute *Costs* which can either be *low* or *high*. Further, there is a relation *Treat* connecting patients with the medications they take. The specific entries of the database are depicted in Fig. 4b, where we again keep the tables simple for illustrative purposes.

A standard learning algorithm to obtain an FG from the given database would include a single randvar for each of the attributes, e.g., there would be a single randvar for the attribute *Age* instead of a separate node for the age of each patient. Analogously, there would also be a single randvar for the attribute *Costs*. When using a single randvar to model an attribute over all objects, however, we lose information about individual objects (here patients and medications) and their relationships between each other.

Therefore, we slightly adjust the learning procedure to include multiple randvars for the same attribute of different individual objects into the learned FG. However, we cannot simply include a randvar for each object per attribute because there would be no uncertainty in the resulting model. To continue Ex. 5, assume we add a randvar *Age* for each patient to the learned FG. Then, the prior probability distribution for the randvar *Age* of a specific patient would map the value found in the database for *Age* to probability one and all other values to probability zero, e.g., the probability that $Age.alice < 18$ would be set to zero and the probability that $Age.alice \geq 18$ would be set to one. Consequently, the

FG would not model any uncertainty at all. To mitigate this issue, we propose to perform an initial clustering of entities to find clusters of indistinguishable objects (here patients and medications), which allows us to naturally define the domains of the logvars when transforming the learned FG into an PFG. After the initial clustering, we then insert a randvar for each cluster per attribute into the FG. Performing an initial clustering of entities allows us to model uncertainty while keeping objects and relationships in the model.

Example 6. Consider again the example database depicted in Fig. 4. For the sake of the example, assume that the initial clustering returns two patient clusters $C_{p_1} = \{alice, eve\}$ as well as $C_{p_2} = \{bob, charlie, dave\}$ and two medication clusters $C_{m_1} = \{myalept, danyelza, eliquis\}$ and $C_{m_2} = \{paracetamol, ibuprofen\}$. Then, the resulting FG contains the randvars $Age.p_1$, $Age.p_2$, $Costs.m_1$, and $Costs.m_2$, that is, there exists one randvar for each cluster per attribute. The probability that, e.g., $Age.p_1 < 18$ is then set to 0.5 (as one out of two entries belonging to the cluster C_{p_1} has $Age < 18$) and the probability that $Age.p_2 < 18$ is set to 0.33 (as one out of three entries belonging to the cluster C_{p_2} has $Age < 18$).

To perform the clustering of the entities, an arbitrary clustering algorithm can be applied. In particular, it is also possible to apply privacy-preserving clustering algorithms, e.g., to ensure differential privacy guarantees. After clustering the entities, the variable nodes of the FG can be inserted for each clustered entity.

More specifically, the resulting FG contains a randvar for every attribute A in the database per cluster of entities occurring in A (i.e., if A is an attribute of an entity E, there is a randvar for A for each cluster of E and if A is an attribute of a relationship R, there is a randvar for R for each combination of clusters of the entities occurring in R). To account not only for the attributes but also for the relationships, there is another randvar in the FG for every relationship R in the database per combination of clusters of entities occurring in R.

In general, the randvars specifying the attributes do not have to be Boolean. The randvars for the relationships, however, are always Boolean as they indicate whether a relationship exists.

Example 7. Given the database from Fig. 4 and the clusters C_{p_1}, C_{p_2}, C_{m_1}, and C_{m_2} from Ex. 6, the resulting FG contains the randvars $Age.p_1$, $Age.p_2$, $Costs.m_1$, $Costs.m_2$, $Treat.p_1.m_1$, $Treat.p_1.m_2$, $Treat.p_2.m_1$, and $Treat.p_2.m_2$.

Note that in a standard learning algorithm for an FG, there are typically no randvars for the relationships and, in addition to that, there are also not multiple randvars for the same attribute for different entities (or clusters of entities, respectively). Therefore, an FG learned by a standard learning algorithm loses information about the relationships of individual objects (or clusters thereof).

After having identified the variable nodes in the FG, the next step is to learn the remaining graph structure, i.e., the factor nodes and the edges of the FG. The graph structure of the FG can be identified using conditional independence tests on the given data [20, Chapter 20.7]. In particular, two randvars X and Y are conditionally independent given a set of randvars Z if $P(X, Y \mid Z) = P(X \mid Z) \cdot P(Y \mid Z)$. Therefore, testing whether pairs of randvars

are conditionally independent given a set of other randvars can be done by estimating the corresponding probabilities from the given data using statistical hypothesis tests. The results of the conditional independence tests then yield the graph structure of the FG as in an FG, two randvars are conditionally independent if all paths between them are blocked by the conditioning set.

We remark that performing conditional independence tests on the individual tables does not work because the individual tables do not necessarily contain all randvars involved in the conditional independence test. The conditional independence tests are hence carried out on a join of the individual tables in the given relational database to enable the estimation of the necessary probabilities.

Example 8. Consider again the database given in Fig. 4. When checking whether the randvars $Patient$ and $Medication$ are independent, we need a joined table which contains both $Patient$ and $Medication$.

As we add randvars for the relationships as well, we augment the full join with an additional column for each relationship, indicating which relationships are present in the database and which are not. We also augment the full join with combinations of entities that do not occur in the relationships from the given database (similar to a cross join) such as the combination of patient *bob* and medication *myalept* in Fig. 4, for which we add a row with value $false$ in the column $Treat$. More details about the augmented full join are given in Appendix A.

After the entire graph structure of the FG has been learned, the next step is to learn the potentials of the factors. More specifically, at this point we know which factors (i.e., functions) are part of the model, but we do not know their input-output mappings yet. To obtain the potentials of the factors, we count the occurrences for each specific assignment of the arguments of the factors in the augmented full join of the tables. To count the occurrences for all possible assignments of factors' arguments, we also need information on combinations of entities that do not occur in the relationships from the given database, which is the reason we add this information in the augmented full join as described above. The occurrences for each specific assignment of factors arguments are then counted as illustrated in the following example.

Example 9. Take a look at the FG shown in Fig. 5. The FG results from the learning procedure described above applied to the database illustrated in Fig. 4. The potentials for, e.g., $\phi_1^2(Age.p_2)$ are obtained by counting the occurrences of $Age.p_2 < 18$ and $Age.p_2 \geq 18$ in the augmented full join shown in Fig. 7 (Appendix A). Recall that $C_{p_2} = \{bob, charlie, dave\}$. Thus, in this particular example, it holds that $\phi_1^2(Age.p_2 < 18) = 5$ and $\phi_1^2(Age.p_2 \geq 18) = 10$.

Note that the absolute number of a potential value is not pivotal as the semantics of the FG includes a normalisation constant. Hence, the semantics would remain unchanged if we had counted the occurrences of $Age.p_2 < 18$ and $Age.p_2 \geq 18$ in the original table. In the original table, we have one occurrence of $Age.p_2 < 18$ and two occurrences of $Age.p_2 \geq 18$ for the cluster $C_{p_2} = \{bob, charlie, dave\}$, i.e., the ratio is exactly the same as in the augmented full join (five occurrences

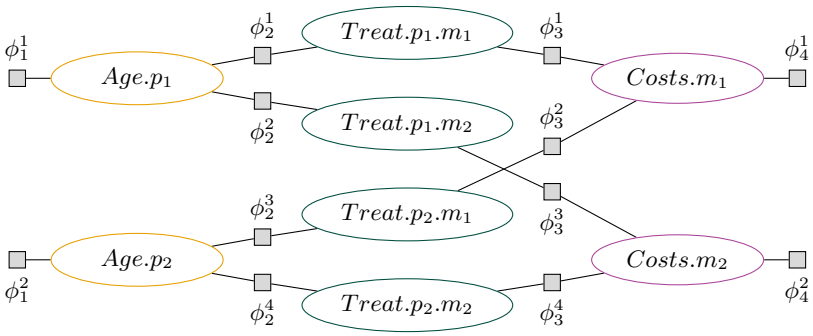


Fig. 5: The learned FG from the given database depicted in Fig. 4.

and ten occurrences). However, we use the augmented full join for counting as we cannot count occurrences of randvars appearing in different tables separately in factors with multiple arguments such as in $\phi_2^3(Age.p_2, Treat.p_2.m_1)$ because we have to incorporate the relationship between those tables. For example, in $\phi_2^3(Age.p_2, Treat.p_2.m_1)$, we need to count the occurrences of $Age.p_2 < 18$ *and* $Treat.p_2.m_1 = true$ and such a combination is not present in the original tables.

A summary of our proposed learning algorithm is provided in Alg. 1. So far, we discussed the steps to learn a propositional FG from a given relational database (Line 1 to Line 14 in Alg. 1). We next describe the procedure to transform the learned FG into a PFG (Line 15 in Alg. 1).

3.2 Transforming the Factor Graph into a Parametric Factor Graph

To obtain a PFG from a given FG, we need to find groups of identically behaving randvars and factors in the FG. Then, PRVs with logvars represent such groups of indistinguishable randvars and parfactors represent groups of identical factors. Replacing indistinguishable randvars by PRVs with logvars further abstracts from individuals and thus yields a promising foundation for privacy guarantees [12]. The ACP algorithm (which is a generalisation of the colour passing algorithm [2,15]) is able to construct a PFG from a given propositional FG [23]. The idea behind ACP is to exploit symmetries in a propositional FG and then group together symmetric subgraphs. ACP looks for symmetries based on potentials of factors, on ranges and evidence of randvars, as well as on the graph structure by passing around colours. A formal description as well as an example run of the ACP algorithm can be found in Appendix B. Figure 6 shows the PFG resulting from calling ACP on the FG depicted in Fig. 5 under the assumption that all potentials of the factors ϕ_i, $i \in \{1, \dots, 4\}$, are considered identical. Note that the assumption of identical factors is just for the sake of the example as in general, not all potentials are identical (and hence, not all of the factors ϕ_i are grouped into a single group).

We remark that in its original form, ACP requires potentials of factors to identically match in order to group factors together. When learning an FG from

Algorithm 1: LearnPFG

Input : A relational database $\mathcal{D} = (\mathcal{E}, \mathcal{R})$ with corresponding data samples.
Output: A PFG $G' = (\boldsymbol{V}, \boldsymbol{E})$ representing the full joint probability distribution of the given database.

$G \leftarrow$ Empty FG;
$F \leftarrow$ Augmented full join over $\mathcal{D}$;
$C_{E_1}, \ldots, C_{E_\ell} \leftarrow$ ClusterEntities($\mathcal{D}$);
foreach *entity or relationship* $B \in \mathcal{E} \cup \mathcal{R}$ **do**
  `// Let` $E_1, \ldots, E_j$ `denote all entities occurring in` B
  foreach *combination of clusters* $(c_1, \ldots, c_j) \in \times_{i=1}^{j} C_{E_i}$ **do**
    foreach *attribute* $A \in \mathcal{A}(B)$ **do**
      Add a randvar $A.c_1.\cdots.c_j$ to G;
foreach *relationship* R **do**
  `// Let` $E_1, \ldots, E_j$ `denote all entities occurring in` R
  foreach *combination of clusters* $(c_1, \ldots, c_j) \in \times_{i=1}^{j} C_{E_i}$ **do**
    Add a randvar $R.c_1.\cdots.c_j$ to G;
Add edges and factors to G by running conditional independence tests on F;
foreach *factor* $\phi(R_1, \ldots, R_k)$ *in* G **do**
  `// Let` $c_1, \ldots, c_j$ `denote all clusters occurring in` $R_1, \ldots, R_k$
  foreach *assignment* $(r_1, \ldots, r_k) \in \times_{i=1}^{k} \mathrm{range}(R_i)$ **do**
    Set $\phi(R_1, \ldots, R_k)$ to the number of rows in F belonging to clusters c_1 to c_j that contain the values r_1 to r_k in the columns of R_1 to R_k;
$G' \leftarrow$ Call ACP on G with evidence $E = \emptyset$;
return G'

data, however, there might be small deviations in the potentials, that is, counting the occurrences of a specific combination of values might differ for two similarly behaving clusters, e.g., by one. Therefore, we adapt the condition for two factors to be considered identical by allowing for a difference in their potentials depending on a user-defined parameter ε. Two factors ϕ and ϕ' are now considered identical (and thus are assigned the same colour in ACP) if for every assignment of their arguments the corresponding potentials, say φ and φ', differ by at most $\varphi \cdot \epsilon$ (or $\varphi' \cdot \epsilon$, respectively). Factors with slightly deviating potentials that are grouped are then all assigned the mean of the potentials in the group.

3.3 Sampling from the Parametric Factor Graph

Every PFG compactly encodes a full joint probability distribution from which we can draw new samples. Note that, in contrast to previous sampling approaches (such as, e.g., [33]) that apply sampling for (approximate) query answering, we aim to generate new data samples that comply with the given ER model (that is, we wish to draw new data samples that contain a value for every attribute and every relationship in the ER model). As the PFG is inherently encoding a relational structure, we are able to synthesise relational data. More specifically, the semantics of a PFG is defined on a ground level, that is, sampling from

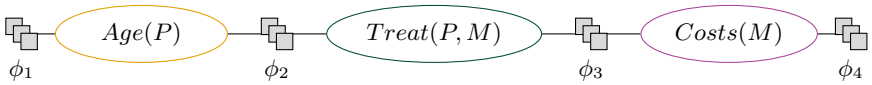


Fig. 6: The PFG resulting from calling ACP on the FG depicted in Fig. 5 assuming for the sake of the example that all clusters behave identically. Note that in our specific example from Ex. 5, not all clusters behave identically and thus, not all randvars representing the same attribute or relationship are grouped.

the underlying probability distribution yields new data samples for multiple clustered entities. As the ground FG consists of randvars for various clustered entities, we draw samples for multiple clusters at the same time.

Example 10. Assume we want to sample the PFG which yields the FG shown in Fig. 5 when grounding the model. Sampling thus yields a value for every randvar in the FG, thereby allowing to synthesise new objects and relationships between them following the full joint probability distribution encoded by the model. An exemplary data sample could look like this: $Age.p_1 < 18$, $Age.p_2 \geq 18$, $Treat.p_1.m_1 = false$, $Treat.p_1.m_2 = false$, $Treat.p_2.m_1 = true$, $Treat.p_2.m_2 = false$, $Costs.m_1 = low$, $Costs.m_2 = high$ (values are chosen arbitrarily for the sake of the example).

The generated synthetic data samples might be used for arbitrary applications and thus, it is necessary to define application-specific quality criteria to assess the quality of the generated data.

4 Conclusion

We introduce a fully-fledged pipeline to deploy probabilistic relational models, in particular PFGs, for the generation of synthetic relational (i.e., multi-table) data. To construct a PFG from a given relational database, we propose a learning algorithm that learns both the graph structure as well as the parameters of a PFG from the relational database. We further elaborate on how the learned PFG can be applied to generate new samples of synthetic relational data. By ensuring certain privacy guarantees (e.g., differential privacy) during the construction process of the PFG, PFG provide a promising model to generate synthetic relational data in a privacy-preserving manner such that generated synthetic data can be publicly shared without leaking sensitive data of individuals.

There are three primary directions for future work. First, privacy guarantees for PFG learning need to be theoretically investigated. Second, the scalability of our proposed architecture should be assessed and improved to allow an efficient handling of large-scale relational databases, and finally, the practicality of the entire framework has to be tested empirically on real-world data sets.

Acknowledgements

This work is funded by the BMBF project AnoMed 16KISA057. This preprint has not undergone peer review or any post-submission improvements or corrections. The Version of Record of this contribution is published in *Lecture Notes in Computer Science, Volume 14992*, and is available online at https://doi.org/10.1007/978-3-031-70893-0_13.

References

1. Abbeel, P., Koller, D., Ng, A.Y.: Learning Factor Graphs in Polynomial Time and Sample Complexity. Journal of Machine Learning Research **7**, 1743–1788 (2006)
2. Ahmadi, B., Kersting, K., Mladenov, M., Natarajan, S.: Exploiting Symmetries for Scaling Loopy Belief Propagation and Relational Training. Machine Learning **92**, 91–132 (2013)
3. Bao, E., Xiao, X., Zhao, J., Zhang, D., Ding, B.: Synthetic Data Generation with Differential Privacy via Bayesian Networks. Journal of Privacy and Confidentiality **11** (2021)
4. Biba, M., Ferilli, M., Esposito, F.: Structure Learning of Markov Logic Networks through Iterated Local Search. In: Proceedings of the Eighteenth European Conference on Artificial Intelligence (ECAI-08). pp. 361–365. IOS Press (2008)
5. Biba, M., Ferilli, S., Esposito, F.: Discriminative Structure Learning of Markov Logic Networks. In: Proceedings of the Eighteenth International Conference on Inductive Logic Programming (ILP-08). pp. 59–76. Springer (2008)
6. Cai, K., Lei, X., Wei, J., Xiao, X.: Data Synthesis via Differentially Private Markov Random Fields. Proceedings of the VLDB Endowment **14**, 2190–2202 (2021)
7. Cai, K., Xiao, X., Cormode, G.: PrivLava: Synthesizing Relational Data with Foreign Keys under Differential Privacy. Proceedings of the ACM on Management of Data **1**, 1–25 (2023)
8. Chen, R.J., Lu, M.Y., Chen, T.Y., Williamson, D.F., Mahmood, F.: Synthetic Data in Machine Learning for Medicine and Healthcare. Nature Biomedical Engineering **5**, 493–497 (2021)
9. Fang, M.L., Dhami, D.S., Kersting, K.: DP-CTGAN: Differentially Private Medical Data Generation Using CTGANs. In: Proceedings of the Twentieth International Conference on Artificial Intelligence in Medicine (AIME-22). pp. 178–188. Springer (2022)
10. Figueira, A., Vaz, B.: Survey on Synthetic Data Generation, Evaluation Methods and GANs. Mathematics **10**, 2733–2773 (2022)
11. Frey, B.J., Kschischang, F.R., Loeliger, H.A., Wiberg, N.: Factor Graphs and Algorithms. In: Proceedings of the Thirty-Fifth Annual Allerton Conference on Communication, Control, and Computing. pp. 666–680. Allerton House (1997)
12. Gehrke, M., Liebenow, J., Mohammadi, E., Braun, T.: Lifting in Support of Privacy-Preserving Probabilistic Inference. German Journal of Artificial Intelligence (2024)
13. Gogoshin, G., Branciamore, S., Rodin, A.S.: Synthetic Data Generation with Probabilistic Bayesian Networks. Mathematical Biosciences and Engineering **18**, 8603–8621 (2021)

14. Hoffmann, M., Braun, T., Möller: Lifted Division for Lifted Hugin Belief Propagation. In: Proceedings of the Twenty-Fifth International Conference on Artificial Intelligence and Statistics (AISTATS-22). pp. 6501–6510. PMLR (2022)
15. Kersting, K., Ahmadi, B., Natarajan, S.: Counting Belief Propagation. In: Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09). pp. 277–284. AUAI Press (2009)
16. Khot, T., Natarajan, S., Kersting, K., Shavlik, J.: Learning Markov Logic Networks via Functional Gradient Boosting. In: Proceedings of the Eleventh IEEE International Conference on Data Mining (ICDM-11). pp. 320–329. IEEE (2011)
17. Kok, S., Domingos, P.: Learning the Structure of Markov Logic Networks. In: Proceedings of the Twenty-Second International Conference on Machine Learning (ICML-05). pp. 441–448. ACM Press (2005)
18. Kok, S., Domingos, P.: Learning Markov Logic Network Structure via Hypergraph Lifting. In: Proceedings of the Twenty-Six International Conference on Machine Learning (ICML-09). pp. 505–512. ACM Press (2009)
19. Kok, S., Domingos, P.: Learning Markov Logic Networks using Structural Motifs. In: Proceedings of the Twenty-Seventh International Conference on Machine Learning (ICML-10). pp. 551–558. Omnipress (2010)
20. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
21. Kschischang, F.R., Frey, B.J., Loeliger, H.A.: Factor Graphs and the Sum-Product Algorithm. IEEE Transactions on Information Theory **47**, 498–519 (2001)
22. Lowd, D., Domingos, P.: Efficient Weight Learning for Markov Logic Networks. In: Proceedings of the Eleventh European Conference on Principles of Data Mining and Knowledge Discovery (PKDD-07). pp. 200–211. Springer (2007)
23. Luttermann, M., Braun, T., Möller, R., Gehrke, M.: Colour Passing Revisited: Lifted Model Construction with Commutative Factors. In: Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence (AAAI-24). pp. 20500–20507. AAAI Press (2024)
24. Luttermann, M., Hartwig, M., Braun, T., Möller, R., Gehrke, M.: Lifted Causal Inference in Relational Domains. In: Proceedings of the Third Conference on Causal Learning and Reasoning (CLeaR-24). pp. 827–842. PMLR (2024)
25. Luttermann, M., Machemer, J., Gehrke, M.: Efficient Detection of Exchangeable Factors in Factor Graphs. In: Proceedings of the Thirty-Seventh International FLAIRS Conference (FLAIRS-24). Florida Online Journals (2024)
26. Mihalkova, L., Mooney, R.J.: Bottom-up Learning of Markov Logic Network Structure. In: Proceedings of the Twenty-Fourth International Conference on Machine Learning (ICML-07). pp. 625–632. ACM Press (2007)
27. Nikolenko, S.I.: Synthetic Data for Deep Learning. Springer, 1st edn. (2021)
28. Pearl, J.: Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann (1988)
29. Poole, D.: First-Order Probabilistic Inference. In: Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03). pp. 985–991. Morgan Kaufmann Publishers Inc. (2003)
30. Richardson, M., Domingos, P.M.: Markov Logic Networks. Machine Learning **62**, 107–136 (2006)
31. Singla, P., Domingos, P.: Discriminative Training of Markov Logic Networks. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05). pp. 868–873. AAAI Press (2005)

32. Taghipour, N., Fierens, D., Davis, J., Blockeel, H.: Lifted Variable Elimination: Decoupling the Operators from the Constraint Language. Journal of Artificial Intelligence Research **47**, 393–439 (2013)
33. Venugopal, D., Gogate, V.G.: Scaling-up Importance Sampling for Markov Logic Networks. In: Advances in Neural Information Processing Systems 27 (NIPS-14). pp. 2978–2986. Curran Associates, Inc. (2014)
34. Xu, L., Skoularidou, M., Cuesta-Infante, A., Veeramachaneni, K.: Modeling Tabular Data Using Conditional GAN. In: Advances in Neural Information Processing Systems 32 (NIPS-19). pp. 6167–6178. Curran Associates Inc. (2019)
35. Yoon, J., Jordon, J., van der Schaar, M.: PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees. In: Proceedings of the International Conference on Learning Representations (ICLR-19). pp. 536–545. IEEE (2019)

A Augmented Full Join

We call the full join of the tables, where an additional column for each relationship is added and missing relationships are encoded by an additional row containing the value $false$ in the corresponding relationship column, the *augmented full join*. The augmented full join can therefore be thought of as a cross join with an additional column for each relationship, which contains a Boolean value indicating which relationships are present in the database. For example, the augmented full join of the tables given in the example from Fig. 4 is illustrated in Fig. 7. In this particular example, the augmented full join contains the additional column $Treat$, which contains the value $true$ if the relationship of a given combination of $PatientId$ and $MedicationId$ in a specific row actually exists. Otherwise, the column $Treat$ contains the value $false$.

PatientId	Age	MedicationId	Treat	Costs
alice	≥ 18	*myalept*	*true*	*high*
alice	≥ 18	*danyelza*	*true*	*high*
alice	≥ 18	*paracetamol*	*false*	*low*
alice	≥ 18	*ibuprofen*	*false*	*low*
alice	≥ 18	*eliquis*	*false*	*high*
bob	≥ 18	*myalept*	*false*	*high*
bob	≥ 18	*danyelza*	*false*	*high*
bob	≥ 18	*paracetamol*	*true*	*low*
bob	≥ 18	*ibuprofen*	*false*	*low*
bob	≥ 18	*eliquis*	*false*	*high*
charlie	≥ 18	*myalept*	*false*	*high*
charlie	≥ 18	*danyelza*	*false*	*high*
charlie	≥ 18	*paracetamol*	*false*	*low*
charlie	≥ 18	*ibuprofen*	*true*	*low*
charlie	≥ 18	*eliquis*	*false*	*high*
dave	< 18	*myalept*	*false*	*high*
dave	< 18	*danyelza*	*false*	*high*
dave	< 18	*paracetamol*	*false*	*low*
dave	< 18	*ibuprofen*	*false*	*low*
dave	< 18	*eliquis*	*false*	*high*
eve	< 18	*myalept*	*false*	*high*
eve	< 18	*danyelza*	*false*	*high*
eve	< 18	*paracetamol*	*false*	*low*
eve	< 18	*ibuprofen*	*false*	*low*
eve	< 18	*eliquis*	*true*	*high*

Fig. 7: Full join of the tables from Fig. 4b, where missing entries for the relationship $Treat$ have been added by setting the value of the column $Treat$ to $false$.

B Formal Description of the Advanced Colour Passing Algorithm

The ACP algorithm [23] builds on the colour passing algorithm [2,15] and solves the problem of constructing a PFG from a given FG. Algorithm 2 provides a formal description of the ACP algorithm.

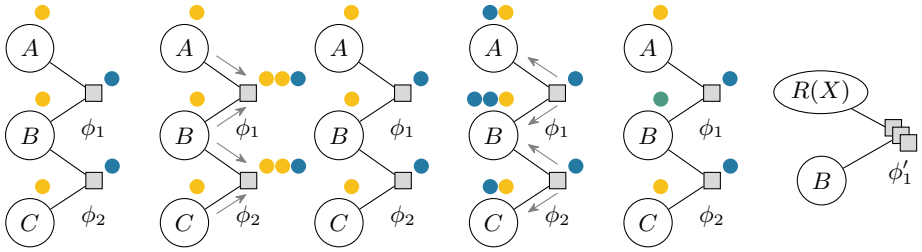
Algorithm 2: Advanced Colour Passing (as introduced in [23])

Input : An FG G with randvars $\boldsymbol{R} = \{R_1, \ldots, R_n\}$, and factors $\boldsymbol{\Phi} = \{\phi_1, \ldots, \phi_m\}$, as well as a set of evidence $\boldsymbol{E} = \{R_1 = r_1, \ldots, R_k = r_k\}$.

Output: A lifted representation G' in form of a PFG with equivalent semantics to G.

Assign each R_i a colour according to $\mathcal{R}(R_i)$ and $\boldsymbol{E}$;
Assign each ϕ_i a colour according to order-independent potentials and rearrange arguments accordingly;
repeat
  foreach *factor* $\phi \in \boldsymbol{\Phi}$ **do**
    $signature_\phi \leftarrow [\,]$;
    foreach *randvar* $R \in neighbours(G, \phi)$ **do**
      `// In order of appearance in` ϕ
      $append(signature_\phi, R.colour)$;
    $append(signature_\phi, \phi.colour)$;
  Group together all ϕs with the same signature;
  Assign each such cluster a unique colour;
  Set $\phi.colour$ correspondingly for all ϕs;
  foreach *randvar* $R \in \boldsymbol{R}$ **do**
    $signature_R \leftarrow [\,]$;
    foreach *factor* $\phi \in neighbours(G, R)$ **do**
      if ϕ *is commutative w.r.t.* $\boldsymbol{S}$ *and* $R \in \boldsymbol{S}$ **then**
        $append(signature_R, (\phi.colour, 0))$;
      else
        $append(signature_R, (\phi.colour, p(R, \phi)))$;
    Sort $signature_R$ according to colour;
    $append(signature_R, R.colour)$;
  Group together all Rs with the same signature;
  Assign each such cluster a unique colour;
  Set $R.colour$ correspondingly for all Rs;
until *grouping does not change*;
$G' \leftarrow$ construct PFG from groupings;

Figure 8 illustrates the ACP algorithm on an example FG [2]. In this example, A, B, and C are Boolean randvars with no evidence and thus, they all receive the same colour (e.g., yellow). As the potentials of ϕ_1 and ϕ_2 are iden-



A	B	$\phi_1(A,B)$
true	true	φ_1
true	false	φ_2
false	true	φ_3
false	false	φ_4

C	B	$\phi_2(C,B)$
true	true	φ_1
true	false	φ_2
false	true	φ_3
false	false	φ_4

$R(X)$	B	$\phi_1'(R(X),B)$
true	true	φ_1
true	false	φ_2
false	true	φ_3
false	false	φ_4

Fig. 8: A visualisation of the steps undertaken by the ACP algorithm on an input FG with only Boolean randvars and no evidence (left). Colours are first passed from variable nodes to factor nodes, followed by a recolouring, and then passed back from factor nodes to variable nodes, again followed by a recolouring. The procedure is iterated until convergence and the resulting PFG is depicted on the right. This figure is reprinted from [23].

tical, ϕ_1 and ϕ_2 are assigned the same colour as well (e.g., blue)[4]. The colour passing then starts from variable nodes to factor nodes, that is, A and B send their colour (yellow) to ϕ_1 and B and C send their colour (yellow) to ϕ_2. ϕ_1 and ϕ_2 are then recoloured according to the colours they received from their neighbours to reduce the communication overhead. Since ϕ_1 and ϕ_2 received identical colours (two times the colour yellow), they are assigned the same colour during recolouring. Afterwards, the colours are passed from factor nodes to variable nodes and this time not only the colours but also the position of the randvars in the argument list of the corresponding factor are shared. Consequently, ϕ_1 sends a tuple $(\text{blue}, 1)$ to A and a tuple $(\text{blue}, 2)$ to B, and ϕ_2 sends a tuple $(\text{blue}, 2)$ to B and a tuple $(\text{blue}, 1)$ to C (positions are not shown in Fig. 8). Since A and C are both at position one in the argument list of their respective neighbouring factor, they receive identical messages and are recoloured with the same colour. B is assigned a different colour during recolouring than A and C because B received different messages than A and C. The groupings do not change in further iterations and hence the algorithm terminates. The output is the PFG shown on the right in Fig. 8, where both A and C as well as ϕ_1 and ϕ_2 are grouped.

[4] The detection of exchangeable factors (DEFT) algorithm [25] can be applied to efficiently detect factors that encode identical potentials regardless of their argument orders and to rearrange the factors' arguments accordingly.