Learning probabilistic relational models with (partially structured) graph databases

Marwa El Abri, Philippe Leray, Nadia Essoussi

Learning Probabilistic Relational Models with (partially structured) Graph Databases

Marwa El abri*†, Philippe Leray†, Nadia Essoussi*
*LARODEC Laboratory, ISG, Université de Tunis, Tunisia
Email: marwa.el-abri@etu.univ-nantes.fr, nadia.essoussi@isg.rnu.tn
†LS2N UMR 6004, DUKe research group, University of Nantes, France
Email: philippe.leray@univ-nantes.fr

***Abstract*—Probabilistic Relational Models (PRMs) such as Directed Acyclic Probabilistic Entity Relationship (DAPER) models are probabilistic models dealing with knowledge representation and relational data. Existing literature dealing with PRM and DAPER relies on well structured relational databases. In contrast, a large portion of real-world data is stored in Nosql databases specially graph databases that do not depend on a rigid schema. This paper builds on the recent work on DAPER models, and describes how to learn them from partially structured graph databases. Our contribution is twofold. First, we present how to extract the underlying ER model from a partially structured graph database. Then, we describe a method to compute sufficient statistics based on graph traversal techniques. Our objective is also twofold: we want to learn DAPERs with less structured data, and we want to accelerate the learning process by querying graph databases. Our experiments show that both objectives are completed, transforming the structure learning process into a more feasible task even when data are less structured than an usual relational database.**

***Keywords*—Probabilistic Relational Models, Entity Relationship Models, DAPER models, graph database, structure learning.**

I. Introduction

With the rise of the Internet, data increase both in volume and interconnections. This fact produces collection of large and complex data sets which are difficult to process using traditional database management tools and applications. With the emergence of Big data, need for more flexible databases is evident. According to [1], Big data consists of the major four V's namely "Volume" which means simply lots of data gathered by a company. "Variety" refers to the type of data that Big Data can comprise, data can be structured as well as unstructured. "Velocity" refers to speed of data generation and the time in which Data Stream can be processed. "Veracity" deals with the uncertainty of data.

Probabilistic Graphical Models (PGM) [2] are one of the most used models to handle uncertainty in real world applications. PGMs such as Bayesian networks are restricted to deal with flat data with homogeneous set of attributes stored in a single table. Extensions have been proposed in literature to deal with structured relational data, such as Probabilistic Relational Models (PRM) [3], [4], Directed Acyclic Probabilistic Entity-Relationships models (DAPER) [5] or Markov Logic Networks (MLN) [6].

PRMs deal with the problem of learning and inference from well structured relational data using SQL language for data querying. The learning part usually consists in generalizing algorithms proposed for Bayesian networks, by adding another dimension in the search space, i.e. traversing the relational schema in order to discover a probabilistic dependency between one attribute and another one from another class. This step requires that the underlying relational schema is already identified, which is simple with a relational database.

A fundamental operation required for PRM learning is counting sufficient statistics for probability estimation and probabilistic dependencies identification. Existing methods [4] compute sufficient statistics using only SQL joins from relational database. However, SQL joins have an exponential cost in term of time consuming specially with complex and highly connected data. This is explained by the fact that large increase of connected data led to a consequent growth of the number joins which impacted negatively on performances of the relational databases.

This reason expedited the movement NOSQL(not only SQL) which are often used for non relational databases as a category of data storage and processing which provide more scalability [7]. NoSQL includes four major types of databases: key/value, oriented document, oriented column, and graph databases. Recently there has been an increasing interest in graph databases to model objects and interactions. In contrast to relational databases, where join-intensive query performance deteriorates as the dataset gets bigger, a graph database depends less on a rigid schema and its performance tends to remain relatively constant, even as the dataset grows. This is because queries are localized to a portion of the graph.

Therefore, we propose in this paper to learn PRM such as Directed Acyclic Probabilistic Entity-Relationships models (DAPER) from partially structured Graph Databases.

Our contribution is twofold. We first present how to extract

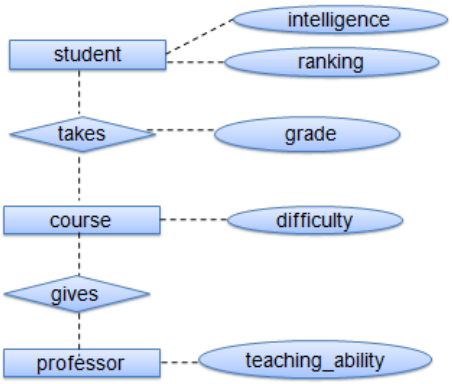

Fig. 1. Example of an ER diagram (inspired from [8]).

the underlying ER model from a partially structured graph database. We then describe a method to compute sufficient statistics based on graph traversal techniques.

Our objective is also twofold: we want to learn DAPERs with less structured data, and we want to accelerate the learning process by querying graph databases. Our experiments will show that both objectives are completed.

The organization of this paper is as follows: some recalls about ER models are presented in Sec. II-A. Sect. II-B presents DAPER models (where data come from well structured relational databases). Sect. II-C describes graph database where we consider that the data is partially structured). We will show in Sect. III-A how to identify an ER model for a partially structured graph database. Then, given this ER model and the graph database, we will focus in Sect. III-B on the computation of the sufficient statistics by querying this graph database. Sect. IV presents experiments that we have performed to evaluate the efficiency of the proposed method. Finally, Sect. V presents conclusion and future work.

II. Background

A. ER model

The Entity-Relationship (ER) model is a higher-level conceptual model developed by [9] used as an abstract representation of database structures. An ER model is defined by three basic concepts: entity classes, relationship classes, and attribute classes. Formally, the **ER model** is defined as $ER = \{\mathcal{E}, \mathcal{R}, \mathcal{A}\}$ where $\mathcal{E} = \{E_1, ..., E_m\}$ is a set of entity classes, $\mathcal{R} = \{R_1, ..., R_n\}$ is a set of relationship classes which correspond to specific interactions between entity classes, and $\mathcal{A}(X)$ are attribute classes for each $X \in (\mathcal{E} \cup R)$ which correspond to variables describing some properties of an entity or relationship. Each attribute $x.A$ has a domain $Dom(x.A)$.

The basic definition of an ER model can be extended by adding other properties such as relationship types or cardinality. Relationship type can be one-to-one, one-to many, many-to-many. Relationship cardinality can be binary, ternary or even more. However, it is often possible to replace higher-order relationships by a collection of binary relationships linking pairs of the original entity types.

The visualization of an ER model is made via an ER diagram, graphical representation of entities and their relationships to each other. In an ER diagram, the entities classes are shown as rectangular nodes, the relationship classes are shown as diamond-shaped nodes, and the attribute classes are represented using ellipses.

Figure 1 illustrates an example of ER diagram designed for a university model (inspired from [8]) which holds information relating to students,courses that they take, and professors that give these courses. In this example we distinguish between :
- $\mathcal{E} = \{student, course, professor\}$,
- $\mathcal{R} = \{takes, gives\}$,
- $\mathcal{A}(student) = \{student.intelligence, student.ranking\}$,
- $\mathcal{A}(course) = \{course.difficulty\}$,
- $\mathcal{A}(professor) = \{professor.teaching\_ability\}$,
- $\mathcal{A}(takes) = \{takes.grade\}$.

$\sigma_{\mathcal{E}}$ (resp. $\sigma_{\mathcal{R}}$) is a specification of the entity classes $\mathcal{E}$ (resp. relationship classes $\mathcal{R}$) with a particular database. A **skeleton** $\sigma_{\mathcal{ER}}$ is defined by $\sigma_{\mathcal{E}} \cup \sigma_{\mathcal{R}}$. An **instance of an ER model**, $\mathcal{I}_{\mathcal{ERA}}$, is defined by a skeleton $\sigma_{\mathcal{ER}}$ and an assignment to a valid value in $Dom(X.A)$ for every attribute $x.A$ where $x \in \sigma_{\mathcal{ER}}$ and A $\in \mathcal{A}(X)$.

B. DAPER model

A DAPER [5] is a probabilistic extension of an ER model where all attributes $\mathcal{A}(X)$ (with $X \in (\mathcal{E} \cup \mathcal{R})$) are random variables that can depend on each other. Generally, the probabilistic graphical structure is defined by a set of parents $pa(X.A)$ for each attribute object $X.A$, associated to a local distribution corresponding to this set of variables.

In [10], these local distributions are defined for each attribute $X.A \in \mathcal{A}(\mathcal{E} \cup \mathcal{R})$ by the conditional probability distribution denoted $P(X.A|pa(X.A))$. Figure 2 shows a DAPER model related to the ER model in Figure 1 where a student's grade for one course depends both on the student's intelligence and on the difficulty of the course. In this figure, the probabilistic dependencies between attributes are represented by red solid arcs.

Parents $pa(X.A)$ of a given attribute can be defined by using paths (also named slot chains) in the ER model [11], [4] or more general constraints [5]. Formally, a slot chain is a set of slots $\rho_1, \rho_2, ..., \rho_n$ where ρ is a (inverse) reference slot which relates objects of a class $X \in (\mathcal{E} \cup \mathcal{R})$ to objects of a class Y. $takes.student$ is the student associated to the corresponding registration. $student.student^{-1}$ is an inverse reference slot corresponding to all the registrations of a given student. As an example of slot chain, $student.student^{-1}.course$ will correspond to all the courses taken by a particular student.

Another important concept in DAPER is the notion of aggregator that comes into play when there is dependency

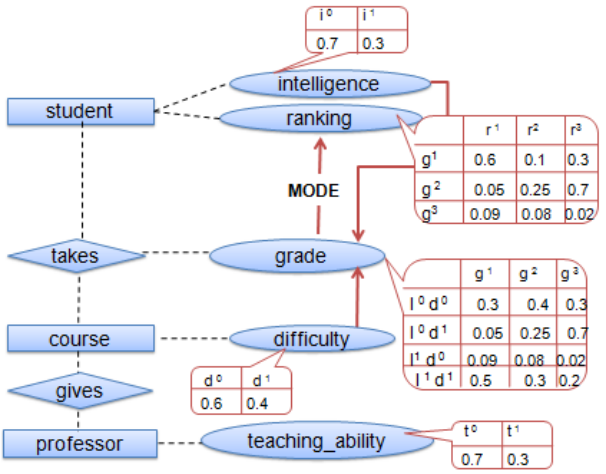

Fig. 2. Example of a DAPER model (inspired from [8]).

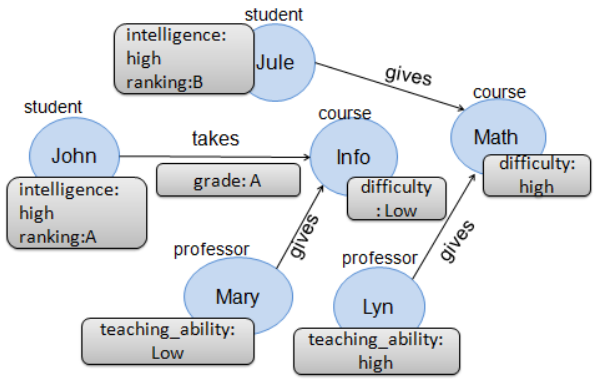

Fig. 3. Example of a graph database.

between the objects that have one-to-many or many-to-many relations. In the figure 2, $student.ranking$ depends on $student.student^{-1}.grade$. As a student can take more than one course, $student.ranking$ will depend on grades of more than one $takes$ object and this number will not be the same for all students. So, in order to get a summary of such dependencies, aggregators (such as MODE, MEDIAN, MIN, ...) are introduced.

The structure learning task for DAPER models aims at identifying the probabilistic dependencies between attributes (and the corresponding conditional probability distributions) given an ER model and its instantiation $\mathcal{I}_{\mathcal{ER}\mathcal{A}}$. The algorithms proposed in order to solve this task are inspired from the classical approaches used to learn Bayesian Networks. RGS (Relational Greedy Search) [11], [4] is the main score-based approximate approach proposed to learn Probabilistic Relational Models structure. [12] adapts an exact learning algorithm based on A^* algorithm. Constraint-based approaches (based on statistical tests) have been developed in [13], [14] and hybrid approaches have been proposed by [15], [16]. When the implementations of these algorithms exist, they all rely on a relational database in order to obtain (1) the relational schema or the ER model, and (2) the instance (i.e. the data) needed in order to estimate the sufficient statistics $N(X.A, pa(X.A))$ mandatory to estimate some scoring functions or independence measurements.

C. Graph database

Graph databases are becoming increasingly popular as an alternative to relational databases for managing complex, increasingly inter-connected, unstructured data [17]. Where relational database management systems use the relational data model, with relations, attributes, and tuples, graph databases use a graph data model, in its simplest form, with the notion of vertices and edges, familiar from graph theory. Semantically, the graph data model is richer than the relational data model, because in a relational model relationships between entities have to be inferred through out primary and foreign keys, while they are modelled explicitly in a graph database.

A graph database is represented by a set of nodes $\mathcal{V} = \{v_i\}$ and a set of edges (relationships) $\mathcal{E}_v = \{e_k =< v_i, v_j >\}$ which connect nodes of $\mathcal{V}$.

As described in Figure 3 with a simple example of a graph database, each node v_i (resp. each edge e_k) can also contain a list of properties called also attributes $\mathcal{A}(v_i)$ (resp. $\mathcal{A}(e_k)$). Each attribute is stored in the form of a {key:value} pair.

We also consider that each node v_i (resp. each edge e_k) is typed, i.e. associated to one label $type(v_i)$ (resp. $type(e_k)$). Finally, we define the signature of one edge as the triplet defined by the types of the two vertices connected by this edge and the type of the edge: $signature(e_k) =< type(v_i), type(e_k), type(v_j) >$ with $e_k =< v_i, v_j >$.

[17] and [7] illustrated the limitations of relational database when modeling interconnected data especially many-to-many relationships in large data sets because it requires multiple joins. In the opposite, with graph databases there are no tables and columns nor "select" and "join" commands, but only graph traversing operations. This flexibility makes graph databases best suited for huge scale applications.

Graph databases are unstructured data stores which are possibly schema-free. Edges between nodes can have varying signatures, when it had to be constant for an ER model. For instance, in Figure 3, $Jule(student) - gives - Math(course)$ does not correspond to the ER model we would like, but can be depicted without any problem in the graph database.

III. DAPER MODEL LEARNING FROM GRAPH DATABASES

As we show in Sect. II-B the existing works about PRMs learning consider that the ER model is known (given by the relational database) and the sufficient statistics

$N(X.A, pa(X.A))$ (used by the learning algorithm) are extracted from a relational database instance.

When dealing with a graph database, we have to solve two main steps. The first one corresponds to the identification of the ER model. The second one corresponds to the computation of the sufficient statistics in this graph database.

A. ER model identification

We consider here that the data stored in a graph database is at least partially structured, i.e. there exist an underlying ER model, even if some exceptions are not derived from this model. Our objective here is the identification of this ER model.

We propose to define $\mathcal{E}$, set of entity classes in the ER model, as the set of unique types of the vertices in the graph databases. $\mathcal{E} = unique(\{type(v_i), \forall v_i \in \mathcal{V}\})$

The definition of the relationship classes is more complex. In an usual ER model, a relationship class is defined for one unique pair of entity classes. In our graph database, a same typed relationship could be used to connect pairs of vertices with different pairs of classes. For instance, for 10 instances of the relation $takes$, our graph database could contain 9 instances with signature $student - takes - courses$ and 1 instance with signature $professor - takes - course$.

By inspiring ourselves from simple application of Inductive Logic Programming principles [18], we propose to define a relationship class in $\mathcal{R}$ for each relationship label and each corresponding signature s that is enough represented for this label in the database, i.e. $N(type(v_i) - type(e_k) - type(v_j)) > \lambda N(type(e_k))$ where $N(.)$ is the number of occurrences in the graph database, and $\lambda \in [0, 1]$ is a user-defined threshold.

For instance, with $N(takes) = 10$, $N(student - takes - course) = 9$, $N(professor - takes - course) = 1$ and $\lambda = .3$, we create one only $takes$ relationship in $\mathcal{R}$. With $N(takes) = 10$, $N(student - takes - course) = 6$, $N(professor - takes - course) = 4$ and $\lambda = .3$, we create two relationships $takes_{sc}$ and $takes_{pc}$ in $\mathcal{R}$, one for each signature.

Finally, $\mathcal{A}(X)$, attribute classes for each entity (or relationship) class, will be defined as the set of all the different attributes we met for each node (resp. edge) of the corresponding class.

Each step in this process can be solved by an unique graph traversal.

B. Sufficient statistics computation

All the algorithms proposed for DAPER structure learning relies on scoring function or independence measurement based on the estimation of sufficient statistics (or contingency tables) $N(X.A, pa(X.A))$, where each parent in $pa(X.A)$ have general form of $\gamma(X.K.B)$ when referencing an attribute B related to the starting class X with a slot chain K, and a possible aggregation function γ.

As an example, estimating the conditional probability distribution $P(takes.grade \mid takes.student.intelligence, takes.course.difficulty)$ requires the computation of the contingency table $N(takes.grade, takes.student.intelligence, takes.course.difficulty)$.

For a relational database, the computation of this table can be done by executing SQL joins of the database tables [4].

```
SELECT grade, intelligence, difficulty, count(*)
FROM takes, student, course
JOIN takes ON student.id = takes.student-id
JOIN course ON takes.course-id=course.id
GROUP BY grade, intelligence, difficulty
```

However, a table join is not feasible for a long length of slot chains and large collection of relationship classes. In fact, queries become more complex because we have to join large number of entities to specify the mapping between a foreign key in one table and the associated primary key.

For a graph database, the same computation can be seen as an exploration of the graph database, where we count occurences of subgraphs patterns corresponding to the variables we want to count, without performing complex grouping operations on the entire data set. In Neo4J graph database and using Cypher language the previous query will be written simpler as follow:

```
MATCH (a:student)-[b:takes] --> (c:course)
RETURN b.grade, a.intelligence, c.difficulty,
count(*)
```

As another more complex example, estimating the conditional probability distribution $P(student.ranking \mid MODE(student.student^{-1}.grade), student.student^{-1}.course.course^{-1}.professor.teach\_ability)$ requires the computation of the contingency table $N(student.ranking, MODE(student.student^{-1}.grade), student.student^{-1}.course.course^{-1}.professor.teach\_ability)$.
For Neo4j graph database, the computation of this table can be done by executing the following Cypher query:

```
MATCH (a:student)-[b:takes] --> (c:course)
OPTIONAL MATCH (a:student)-[d:takes] --> (e:course)
OPTIONAL MATCH (e:course)<--[f:gives]-(g:professor)
RETURN  a.ranking, MODE(b.grade),g.teaching_ability
count(*)
```

However for relational database, this same query is done by executing the following SQL joins:

```
SELECT ranking, MODE(grade), teaching_ability, count(*)
FROM takes, student, course, gives, professor
JOIN takes ON student.id = takes.student-id
JOIN course ON takes.course-id=course.id
JOIN gives ON gives.course-id=course.id
JOIN professor ON professor.id=gives.professor.id
GROUP BY ranking, MODE(grade), teaching_ability
```

IV. Experiments and results

A. Networks and Datasets

Unlike standard Bayesian networks, where several benchmarks or golden networks are available to perform exper-

	$\mid \mathcal{E} \mid$	$\mid \mathcal{R} \mid$	$\mid \mathcal{A} \mid$	$\mid \mathcal{V} \mid$	$\mid \mathcal{K} \mid$
DAPER1	2	1	8 (2-3)	2-3	1 : 2-1
DAPER2	5	4	16(2-5)	2-3	1 : 9-4
DAPER3	8	7	32 (1-5)	2-4	4 : 0-4-12-5-6

TABLE I. CHARACTERISTICS OF DAPER NETWORKS USED FOR DATASET GENERATION. VALUES IN PARENTHESIS ARE MIN/MAX VALUES PER CLASS. VALUES IN THE LAST COLUMN CORRESPOND TO THE NUMBER OF DEPENDENCIES FOR EACH SLOT CHAIN LENGTH (FROM 0 TO k_{max}).

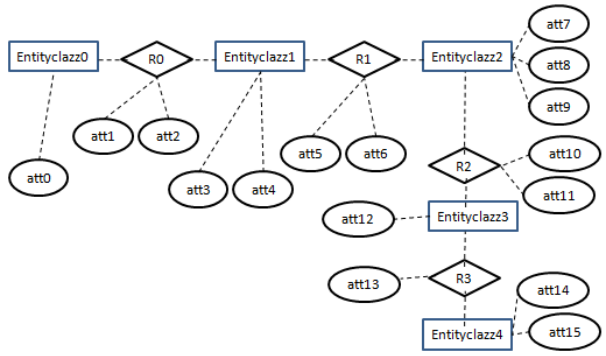

Fig. 4. The ER schema of DAPER2 network.

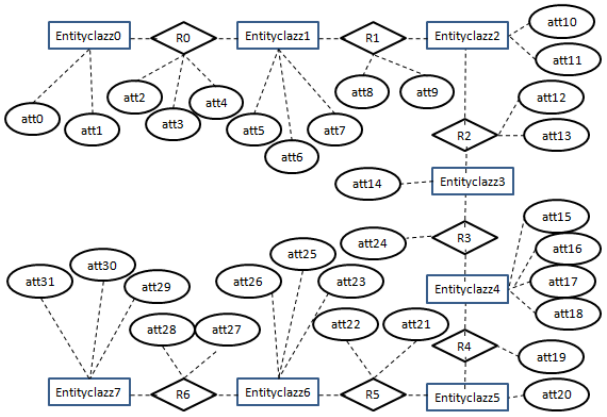

Fig. 5. The ER schema of DAPER3 network.

iments, there is no such models defined in the context of DAPER. In the other side, we have to compare DAPER learned from relational databases and DAPER learned from graph databases. Consequently, we have used the generating process defined by [19] to first generate theoretical DAPER (ER models and probabilistic dependencies) and then to sample relational database instances from these probabilistic models. We have randomly generated 3 DAPER whose characteristics are described in table I. These DAPERs contain binary relationship classes and attributes of type integer. As an example, the ER schema of DAPER2 and DAPER3 are given in Figures 4 and 5 respectively.

We have then sampled relational databases with 500, 1000, 3000 and 5000 instances. This sampling process is repeated 10 times for each DAPER. Our relational databases are managed with PostgreSQL 9.3 and transformed into graph databases managed with Neo4J 3.1.0.

DAPER3 has also be used to generate some additional graph databases. Some exceptions (as described at the end of section II-C) have been added in the 10 databases of size 3000 in order to perturb the underlying ER model. The signature of relationship instances has been replaced by another signature not conform with the underlying ER model. The percentage of exceptions α is varied as 5%, 10%, 30% and 50%. Finally, we generated one large database from DAPER3 with 500.000 instances.

B. Algorithms and implementation

We consider the problem of structure learning by applying the reference algorithm, Relational Greedy Search (RGS), already implemented in PILGRIM Relational C++ Library[1]. RGS uses as an input a relational schema (directly imported from SQL database, or identified from our graph database as described in section III-A). The parameters used for this process are :

- λ : identification threshold for ER identification, which is fixed to 10%. We have chosen a low value of λ to see the impact of exceptions by easily accepting creation of relationships.
- k_{max} : maximum possible slot chain length during greedy search (set to 1 for experiments with databases generated from DAPER1 and DAPER2, set to 1 then 4 for DAPER3). This parameter controls the complexity of the algorithm by defining "how far" (in the relational schema) can be some dependent attributes. Here, we simply choose the same horizon than the one used for generating the DAPERs, except for DAPER3, where we use a simple value (1) and an longer one (4) to illustrate the impact of this parameter on both the running time and the quality of reconstruction.

Let us denote RGS_postgres the usual RGS algorithm, working with postgreSQL databases (without exceptions), and RGS_neo4j the same algorithm, working with Neo4J graph databases (with or without exceptions) with an initial ER identification.

This C++ libray uses DTL library[2] in order to access to the PostgreSQL database, and Casablanca C++ REST SDK[3] in order to acces Neo4J via its REST API.

C. Evaluation metrics

We have compared the algorithms in term of quality of reconstruction and in term of running time.

[1] http://pilgrim.univ-nantes.fr
[2] http://dtemplatelib.sourceforge.net
[3] https://github.com/Microsoft/cpprestsdk

Data size	500	1000	3000	5000
Precision	0.22±0.11	0.28±0.1	0.35±0.11	0.49±0.09
Recall	0.29±0.13	0.38±0.1	0.40±0.05	0.56±0.07
F-score	0.27±0.11	0.30±0.08	0.37±0.13	0.51±0.07

Data size	500	1000	3000	5000
Precision	0.42±0.08	0.49±0.1	0.76±0.06	0.84±0.05
Recall	0.36±0.1	0.43±0.09	0.61±0.04	0.71±0.06
F-score	0.38±0.1	0.45±0.07	0.70±0.06	0.81±0.08

TABLE II. AVERAGE ± STANDARD DEVIATION OF RECONSTRUCTION METRICS (PRECISION, RECALL AND F-SCORE) WHEN EXECUTING RGS WITH K_{max}=1 FOR DATABASES GENERATED FROM DAPER1 (TOP) AND DAPER2 (BOTTOM).

Data size	500	1000	3000	5000
Precision	0.39±0.11	0.36±0.10	0.47±0.08	0.52±0.10
Recall	0.44±0.08	0.42±0.06	0.62±0.09	0.63±0.08
F-score	0.40±0.05	0.39±0.12	0.57±0.10	0.60±0.08

Data size	500	1000	3000	5000	500000
Precision	0.45±0.05	0.47±0.07	0.56±0.10	0.66±0.08	0.68±0.05
Recall	0.48±0.11	0.53±0.07	0.64±0.07	0.68±0.06	0.71±0.04
F-score	0.46±0.12	0.50±0.10	0.58±0.06	0.66±0.09	0.69±0.05

TABLE III. AVERAGE ± STANDARD DEVIATION OF RECONSTRUCTION METRICS FOR EACH SAMPLE SIZE WHEN EXECUTING RGS WITH $K_{max} = 1$ (TOP) AND K_{max}=4 (BOTTOM) FOR DATABASES GENERATED FROM DAPER3.

When the graph databases include exceptions, we estimate the quality of the ER schema identification by computing Precision_{ER} and Recall_{ER} between the identified ER schema and the theoretical one.

$$Precision_{ER} = \frac{Relevant\ NR_{Learned}\ in\ ER_{Learned}}{NR_{Learned}\ in\ ER_{Learned}}$$

$$Recall_{ER} = \frac{Relevant\ F_{ERLearned}\ in\ ER_{Learned}}{F_{ERLearned}\ in\ ER_{True}}$$

The quality of reconstruction of the probabilistic dependencies is measured using Precision, Recall and F-score, as defined in [20].

$$Precision = \frac{N_r}{N_l}$$

$$Recall = \frac{N_r}{N_t}$$

$$F\_score = \frac{2 * Precision * Recall}{Precision + Recall}$$

where S_{True} is the dependency structure of the theoretical DAPER (and N_t the number of dependencies in S_{True}), $S_{Learned}$ is the dependency structure of the learned one (and N_l the number of dependencies in $S_{Learned}$), and N_r is the number of relevant dependencies retrieved in $S_{Learned}$.

D. Results and interpretation

We first successfully checked that RGS_postgres and RGS_neo4j converged to the same learned ER schema and probabilistic dependency structure (for databases without exceptions). This is logical because both compute the same sufficient statistics from the same data, but stored in two different ways.

Table II depicts the quality reconstruction (identical for RGS_postgres and RGS_neo4j) for our 2 first DAPERs, when executing RGS with k_{max}=1. We can observe that increasing the number of instances of datasets improves the quality of reconstruction of our structures.

Figure 6 represents running time in the same context. With a very small model such as DAPER1, the running time is almost divided by 2 when accessing the graph database. With a small model such as DAPER2, the running time is almost divided by 10.

Table III depicts the quality reconstruction (identical for RGS_postgres and RGS_neo4j) for DAPER3, when executing RGS with k_{max}=1 and with a bigger search space (k_{max}=4). Figures 7 represents running time in the same context, in log scale. We can observe that increasing the slot chain length improves the quality of reconstruction, but increases the running time. The deeper we go, the better results we get in term of Precision, Recall and F-score. By increasing the search space, we increase the number of times the data is accessed, and Figure 7 shows us again the interest of using a graph database instead of a relational one, with a running time divided by a factor greater than 10. This is even more impressive in Figure 7(right) with 500.000 instances, where the running time drops drastically. transforming the structure learning process into a feasible task.

Table IV depicts the quality reconstruction (RGS_neo4j) for DAPER3 when dealing with partially structured data. We can notice that increasing the percentage of exceptions in the relationship instances do not impact the quality reconstruction of the DAPER model with small α. The relationship signatures are not enough perturbed so the recall is perfect, and the exceptions added in the database are not sufficient to be identify as new (and false) relationships, so the precision is also perfect. In this context, precision, recall and F-score corresponding to the reconstruction of the dependency structure remain constant.

By increasing α, some existing relationships are not sufficiently represented in the database and are no more discovered and the recall decreases. The number of perturbed relationships for one given signature is yet not sufficient to generate false relationships so the precision is yet perfect. When existing relationships in the DAPER are no more discovered, some probabilistic dependencies can't be discovered because the corresponding attributes (and classes) are no more related in the ER model. For this reason the corresponding reconstruction metrics progressively decrease.

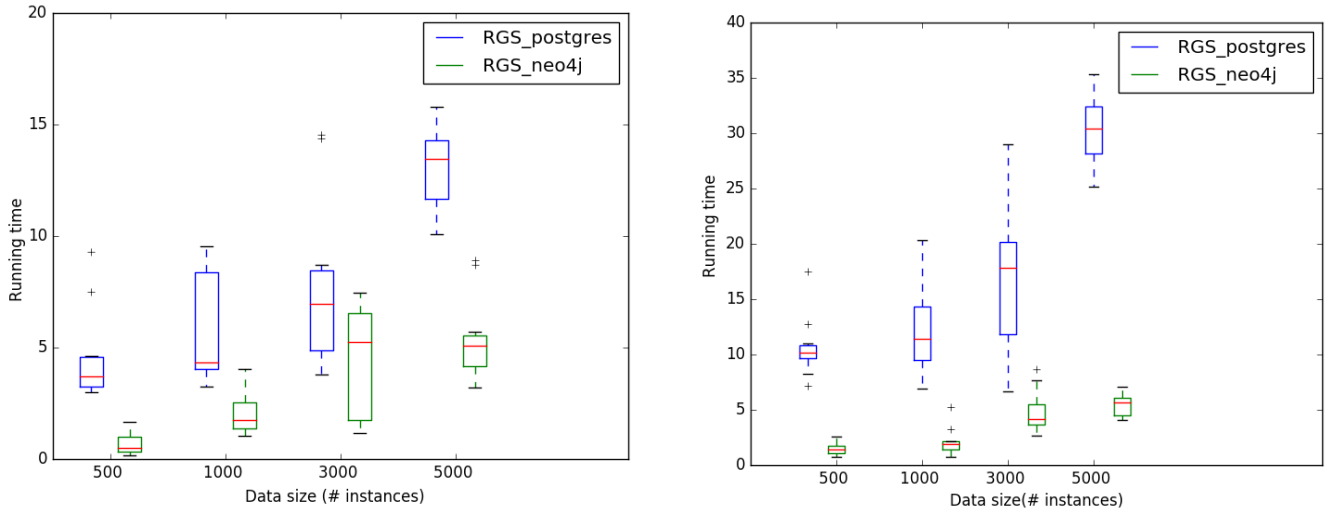

Fig. 6. Boxplot of running time (in seconds) for each sample size, when executing RGS_postgres and RGS_neo4j with k_{max}=1 and databases generated from DAPER1 (left) and DAPER2 (right).

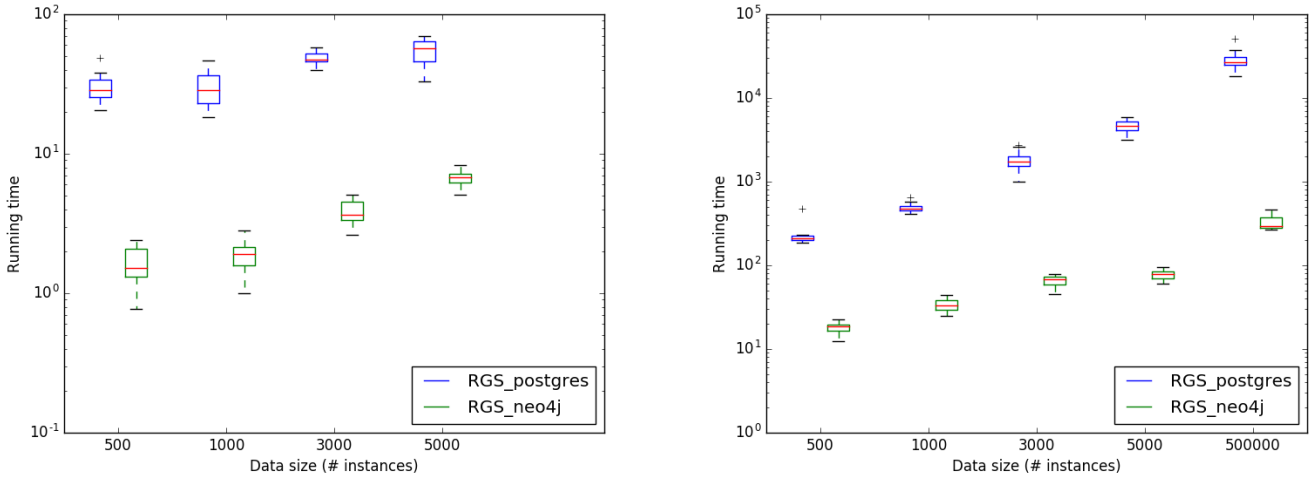

Fig. 7. Boxplot of running time (in seconds and log scale) for each sample size, when executing RGS with k_{max}=1 (left) and k_{max}=4 (right) and databases generated from DAPER3

α	0%	5%	10%	30%	50%
Prec$_{ER}$	1.00±0.00	1.00±0.00	1.00±0.00	1.00±0.00	1.00±0.00
Rec$_{ER}$	1.00±0.00	1.00±0.00	1.00±0.00	0.97±0.05	0.87±0.05
Prec	0.47±0.08	0.47±0.08	0.47±0.08	0.44±0.05	0.41±0.06
Rec	0.62±0.09	0.62±0.09	0.62±0.09	0.59±0.08	0.57±0.07
F-score	0.57±0.10	0.57±0.10	0.57±0.10	0.55±0.04	0.49±0.09

TABLE IV. AVERAGE ± STANDARD DEVIATION OF RECONSTRUCTION METRICS FOR DAPER3 FOR A PARTICULAR SIZE (3000 INSTANCES) WITH A PERCENTAGE α OF EXCEPTIONS IN THE RELATIONSHIP INSTANCES.

V. CONCLUSION AND FUTURE WORKS

We are interested in learning probabilistic relational models such as DAPER with partially structured data stored in graph databases, i.e. there exist an underlying ER model, even if some exceptions are not derived from this model. We proposed one solution to identify this ER model, and a way to estimate, from this graph database, the sufficient statistics that are computed in any learning algorithm.

We have implemented these two contributions in PILGRIM, a software platform dealing with probabilistic relational models. By executing a standard DAPER structure learning algorithm (RGS) in several contexts, we have shown that even with structured data, using graph databases can effectively speed up the learning process. This is even more impressive

with a high number of instances, where the running time drops drastically. transforming the structure learning process into a feasible task. We have also demonstrated that we are able to learn the underlying ER model and the probabilistic dependency structure even with partially structured data.

We are now interested by other probabilistic and relational frameworks derived from Logic such as Markov Logic Networks [6] or ProbLog models [21]. The DAPER models we use here can only model the probabilistic dependencies between the instances that are conform with the ER model. We have the intuition than these other frameworks can help us to take also into account the exceptions that are dropped during DAPER learning. With the existing structure learning algorithms proposed for MLN or ProbLog, we also think that we could solve in the same time the ER identification step, and the probabilistic dependency structure learning.

REFERENCES

[1] J. Sun and C. K. Reddy, "Big data analytics for healthcare," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 1525–1525.

[2] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[3] D. Koller and A. Pfeffer, "Probabilistic frame-based systems," in *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, ser. AAAI '98/IAAI '98. Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998, pp. 580–587.

[4] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," in *Relational Data Mining*, S. Dzeroski and N. Lavrac, Eds. Springer-Verlag, 2001.

[5] D. Heckerman and C. Meek, "Probabilistic entity-relationship models, prms, and plate models," *ICML*, pp. 55–60, 2004.

[6] M. Richardson and P. Domingos, "Markov logic networks," *Machine Learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[7] J. Partner, A. Vukotic, and N. Watt, *Neo4j in Action*. Manning Publications Company, 2014.

[8] L. Getoor, "Learning statistical models from relational data," Ph.D. dissertation, Stanford, 2001.

[9] P. P.-S. Chen, "The entity-relationship model: Toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, Mar. 1976.

[10] F. Kaelin and D. Precup, "A study of approximate inference in probabilistic relational models," in *Proceedings of 2nd Asian Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Sugiyama and Q. Yang, Eds., vol. 13. Tokyo, Japan: PMLR, 08–10 Nov 2010, pp. 315–330.

[11] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer, "Learning probabilistic relational models," *In IJCAI*, pp. 1300–1309, 1999.

[12] N. Ettouzi, P. Leray, and M. Ben Messaoud, "An exact approach to learning probabilistic relational model," in *Proceedings of the 8th International Conference on Probabilistic Graphical Models (PGM 2016)*, Lugano, Switzerland, September 2016, pp. 171–182.

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

[14] S. Lee and V. Honavar, "On learning causal models from relational data." in *AAAI*, 2016, pp. 3263–3270.

[15] X.-L. Li and X.-D. He, "A hybrid particle swarm optimization method for structure learning of probabilistic relational models," *Information Sciences*, vol. 283, pp. 258 – 266, 2014, new Trend of Computational Intelligence in Human-Robot Interaction.

[16] M. Ben Ishak, "Probabilistic relational models: learning and evaluation," Ph.D. dissertation, Université de Nantes, Ecole Polytechnique ; Université de Tunis, Institut Supérieur de Gestion de Tunis, Jun. 2015.

[17] C. Vicknair, M. Macias, Z. Zhao, X. Nan, Y. Chen, and D. Wilkins, "A comparison of a graph database and a relational database: A data provenance perspective," *The 48th Annual Southeast Regional Conference*, pp. 421–426, 2010.

[18] S. Muggleton and L. de Raedt, "Inductive logic programming: Theory and methods," *The Journal of Logic Programming*, vol. 19, no. 0, pp. 629 – 679, 1994.

[19] M. Ben Ishak, R. Chulyadyo, and P. Leray, "Probabilistic relational model benchmark generation," University of Nantes, ISG Tunis, Tech. Rep., 2016.

[20] M. E. Maier, K. Marazopoulou, D. T. Arbour, and D. D. Jensen, "A sound and complete algorithm for learning causal models from relational data," *CoRR*, vol. abs/1309.6843, 2013.

[21] L. D. Raedt, A. Kimmig, and H. Toivonen, "Problog: A probabilistic prolog and its application in link discovery." in *IJCAI*, M. M. Veloso, Ed., 2007, pp. 2462–2467.