Querying Factorized Probabilistic Triple Databases

Denis Krompaß[1], Maximilian Nickel[2], and Volker Tresp[1,3]

[1] Ludwig Maximilian University, 80538 Munich, Germany
Denis.Krompass@campus.lmu.de
[2] Massachusetts Institute of Technology, Cambridge, MA
and Istituto Italiano di Tecnologia, Genova, Italy
mnick@mit.edu
[3] Siemens AG, Corporate Technology, Munich, Germany
Volker.Tresp@siemens.com

Abstract. An increasing amount of data is becoming available in the form of large triple stores, with the Semantic Web's linked open data cloud (LOD) as one of the most prominent examples. Data quality and completeness are key issues in many community-generated data stores, like LOD, which motivates probabilistic and statistical approaches to data representation, reasoning and querying. In this paper we address the issue from the perspective of probabilistic databases, which account for uncertainty in the data via a probability distribution over all database instances. We obtain a highly compressed representation using the recently developed RESCAL approach and demonstrate experimentally that efficient querying can be obtained by exploiting inherent features of RESCAL via sub-query approximations of deterministic views.

Keywords: Probabilistic Databases, Tensor Factorization, RESCAL, Querying, Extensional Query Evaluation

1 Introduction

The rapidly growing Web of Data, e.g., as presented by the Semantic Web's linked open data cloud (LOD), is providing an increasing amount of data in form of large triple databases, also known as triple stores. However, the LOD cloud includes many sources with varying reliability and to correctly account for data veracity remains a big challenge. To address this issue, reasoning with inconsistent and uncertain ontologies has recently emerged as a research field of its own [6, 31, 4, 9, 3, 15]. In this paper we approach the veracity issue from the perspective of probabilistic databases (PDB), which consider multiple possible occurrences of a database via a possible worlds semantics and account for uncertainty in the data by assigning a probability distribution over all database instances [27]. As such, querying PDBs has a clear interpretation as generalizations of deterministic relational database queries.

When applying PDBs to large triple stores various key challenges need to be addressed. First, consider storage requirements. A common assumption in PDBs

is tuple independence, or in our case triple independence, which requires a representation of each triple's uncertainty. Unless default values are used, representing the individual uncertainty levels can lead to huge storage requirements.

Second, there is the question of how the triple uncertainty is quantified and where it is specified. In PDBs one typically assumes that the data uncertainty is specified by the application, e.g., one assumes given measurements of uncertainty. However, in the Web of Data such information is typically not available. Rather the Web of Data is incomplete and contains incorrect information, as triples are missing and existing triples can be false. The third issue concerns probabilistically correct and efficient querying. Although individual triples are assumed to be independent, complex queries introduce dependencies such that correct query answering becomes, in the worst case, intractable.

We address all three issues by exploiting the recently developed RESCAL approach [19], which computes a memory-efficient low-rank tensor representation of a triple store from which probabilities for individual triples can be derived easily without materializing the whole probabilistic representation of the triple store. These probabilities are meaningful in as much as the underlying data generation process (which we explore through the factorization) is sensible for the particular triple store; experimentally it has been shown that this is the case, e.g., for significant parts of the LOD [20]. While the RESCAL model alone allows to query the probability of individual ground triples, more complex queries, which generally might contain complex dependencies between ground triples throughout the complete PDB, remain a major challenge. When restricting the allowed queries to so-called safe queries, extensional query evaluation can provide answers with polynomial time complexity [27]. Unfortunately, polynomial time complexity does not necessarily imply acceptable query time. During query evaluation it might be necessary to materialize sub-queries, which often produce large (dense) views and as such have high computational and high memory complexity. Here, we propose a new method for extensional query evaluation (after factorization) that avoids the expensive materialization of dense database views by exploiting the factorized low-rank representation of a triple store computed by RESCAL. The idea is to first materialize the sub-query in the initial deterministic representation of the triple store (that can be efficiently constructed and is generally sparse) and to then derive a low-rank approximation of that view using fast RESCAL updates, which then produces the probabilistic view.

The paper is organized as follows: In the next section we discuss related work. Section 3 reviews PDBs and Section 4 describes the RESCAL approach. Section 5 contains the main contribution of the paper and addresses the querying of a factorized probabilistic triple databases. Section 6 describes our experimental results and Section 7 contains our conclusions.

2 Related Work

Probabilistic databases (PDB) have gained much interest in recent years and an overview over the state of the art is provided by [27]. Important ongoing

research projects include the MayBMS project [10] at Cornell University, the MystiQ project [2] at the University of Washington, the Orion project [26] at the Purdue University, and the Trio project [16] at the Stanford University. The idea of materialized views on PDBs has been developed in [5], where it was proposed to materialize probabilistic views to be used for query answering at runtime.

Uncertainty in Semantic Web ontologies has been addressed in BayesOWL [6] and OntoBayes [31]. Furthermore, PR-OWL [4] is a Bayesian Ontology Language for the Semantic Web. The link between PDBs, Description Logic and Semantic Web data structures has been explored by [9, 3, 15]. In contrast to these approaches, we start with a deterministic triple store and then derive probabilities via the factorization. Common to all these approaches is the challenge of efficient querying.

In our work we perform a probabilistic ranking of candidate query answers as done by the top-k querying approaches [7, 21, 25], however without pruning of low-confidence answers. In these top-k querying approaches the computation of exact probabilities for the potential top-k answers are avoided by using lower and upper bounds for the corresponding marginal probabilities. Unfortunately, none of these approaches, apart from [7], can avoid extensive materializations in finding answer candidates [28].

Tensors have been applied to Web analysis in [12] and to ranking predictions in the Semantic Web in [8]. [23] applied tensor models to rating predictions. Using factorization approaches to predict ground atoms was pioneered in [29]; [19], [30], [1], and [11] applied tensor models for this task, where [19] introduced the RESCAL model. [24] applied matrix factorization for relation extraction in universal schemas.

In [20] the YAGO ontology was factorized and meaningful predictions of simple triples in selected subgroups of YAGO were derived through the reconstruction from the low rank representation of the triple store. However, in these papers querying was limited to the evaluation of independent ground triples. Here, we study the significantly more difficult problem of complex queries on predicted triple confidence values (including existentially quantified variables). Thereby we realize extensional query evaluation on safe queries, which can induce complex dependencies between individual predicted triple probabilities throughout the database. In particular, we are concerned with how these queries can be executed efficiently, without the need for extensive materialization of probability tables, by exploiting the factorized representation during querying.

3 Probabilistic Databases

3.1 Semantics

Probabilistic databases (PDB) have been developed to extend database technologies to handle uncertain data. A general overview is provided by [27]. PDBs build on the concept of incomplete databases, i.e. databases that are allowed to

be in one of multiple states (worlds): Given an active domain $ADom$ of constants (resources in RDF) each world contains a subset of all possible ground atoms (triples), formed by the elements of $ADom$ and the predicates in the database. A PDB then assigns a probability distribution to all possible worlds $W \in \mathbf{W}$, where $\mathbf{W}$ is the set of all worlds under consideration. More precisely, given an active domain $ADom$ of constants or resources in terms of a RDF framework, each world contains a subset of all possible ground atoms (triples), formed by the elements of $ADom$ and the predicates. A PDB is an incomplete database where, furthermore, a probability distribution is assigned to the possible worlds. In the following, we adopt the common assumption of PDBs that the probabilities for all ground atoms are mutually independent (i.e., tuple independence).

3.2 Querying on Probabilistic Databases

Query evaluation in PDBs remains a major challenge. In particular, scalability to large domains is significantly harder to achieve when compared to deterministic databases.

Of interest here is the *possible answer semantics* which calculates the probability that a given tuple t is a possible answer to a query Q in a world $W \in \mathbf{W}$ of a PDB $\mathbf{D}$. For the marginal probability over all possible worlds, we get

$$P(t \in Q) = \sum_{W \in \mathbf{W}: t \in Q_W} P(W)$$

where Q_W is the query with respect to one possible world of $\mathbf{D}$.

An important concept in query evaluation is the lineage $\Phi_Q^{\mathbf{D}}$ of a possible answer tuple t to Q with respect to $\mathbf{D}$. The lineage of a possible output tuple to a query is a propositional formula over tuple states in the database, which says which input tuples must be present in order for the query to return the particular output. The concept of lineage allows a reduction of the query evaluation problem to the evaluation of propositional formulas. For queries involving many variables the lineage can become very complex but can still be derived in polynomial time (with the size of the database).[4]

In *intensional query evaluation*, probabilistic inference is performed over the lineage of all possible answers to a query Q. Every relational query can be evaluated this way, but the data complexity depends dramatically on the query being evaluated, and can be hard for #P. Thus it is in general advantageous to avoid the evaluation of the lineage.

In contrast, *extensional query evaluation* is concerned with queries where the entire probabilistic inference can be computed in a database engine and thus, can be processed as effectively as the evaluation of standard SQL queries. Relational queries that can be evaluated this way are called *safe queries*. If the extensional query plan does compute the output probabilities correctly for any input database, then it is called a safe query plan.

[4] For a more detailed explanation on the concept of lineage we refer to [27, Chp.2].

In this paper we focus on queries that allow an extensional query evaluation, as discussed next.

3.3 Extensional Query Evaluation

Extensional query evaluation is only dependent on the query expression itself and the lineage does not need to be computed. During the evaluation process, several rules are applied to the query in order to divide it into smaller and easier sub-queries until it reduces to ground tuples with elementary probability values. Queries that can be completely simplified to ground tuples with extensional evaluation rules are *safe*, in the sense mentioned above. Extensional query evaluation can be implemented via the application of six rules [27]. In the following we will briefly describe three of those rules relevant throughout this paper.

Consider a query that can be written as a conjunction of two simpler queries

$$Q = Q_1 \wedge Q_2.$$

If one can guarantee that the sub-queries are independent probabilistic events, one can write

$$P(Q_1 \wedge Q_2) = P(Q_1) \cdot P(Q_2). \qquad \text{(independent-join rule)}$$

More formally one needs the condition of a *syntactical independence* between Q_1 and Q_2: Two queries Q_1 and Q_2 are syntactically independent if no two relational atoms unify, which means that it is not possible that exactly the same ground tuples, under any assignments of constants, can occur in both sub-queries. With syntactical independence we also get

$$P(Q_1 \vee Q_2) = 1 - (1 - P(Q_1)) \cdot (1 - P(Q_2)). \qquad \text{(independent-union rule)}$$

Further for queries of the form $\exists x.Q$ we can use the independent-project rule

$$P(\exists x.Q) = 1 - \prod_{b \in ADom} (1 - P(Q[b/x])). \tag{1}$$

This rule can always be applied if x is a separator variable.[5] As an example, consider that we want to know the probability that Jack is born in Rome and that he likes someone who is a child of Albert Einstein (AE). This query can be written as a conjunction of two sub-queries Q_1 and Q_2,

$$Q_1(x,t) :- \mathit{bornIn}(x,t)$$
$$Q_2(x,z) :- \exists y.(\mathit{likes}(x,y), \mathit{childOf}(y,z))$$

with $x = \mathsf{Jack}$, $t = \mathsf{Rome}$ and $z = \mathsf{AE}$. Q_1 asks if Jack is born in Rome and Q_2 if Jack likes somebody who is a child of Albert Einstein. By exploiting the

[5] x is a separator variable if it occurs in all atoms in Q and if in the case that atoms unify, x occurs at a common position. Two atoms unify if they can be made identical by an assignment of constants.

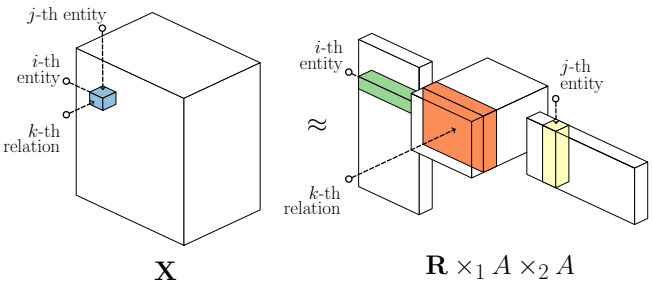


Fig. 1. RESCAL model for binary relations

independent join rule on Q_1 and Q_2 and the independent project rule on Q_2 this query can be calculated as,

$$P(Q(x = \mathsf{Jack}, t = \mathsf{Rome}, z = \mathsf{AE}) = P(\mathit{bornIn}(\mathsf{Jack}, \mathsf{Rome})) \times \left(1 - \prod_{b \in A\mathit{Dom}} [1 - P(\mathit{likes}(\mathsf{Jack}, b)) \cdot P((\mathit{childOf}(b, \mathsf{AE}))]\right).$$

Note that since a person can like many other persons the *likes* atoms in Q_2 are not mutually exclusive what induces a complex expression that is not simply the sum or the product or probabilities.

4 RESCAL

In PDBs one typically assumes that the data uncertainty is specified by the application. Although there are efforts towards uncertainty management for the Web of Data, currently such information not widely available. Rather the Web of Data is incomplete, i.e., many triples are missing, and contains incorrect information, i.e., existing triples are false. To overcome this problem, we employ a probabilistic model of the triple database – i.e. the recently developed RESCAL approach – and derive data uncertainty from the agreement of the data with this model. RESCAL relies on a specific factorization of a third-order adjacency tensor from which triple probabilities can be derived. In addition, RESCAL computes a highly compressed representation of the PDB, reducing the memory footprint dramatically. The latter point is of particular interest, since PDBs can be dense.

4.1 Notation

To describe RESCAL, we need to introduce some notation. We consider a database as a set of M relations $\{r_k\}_{k=1}^{M}$. A relation is a table with attributes as columns and subject-object entity pairs $E_z = (e_i, e_j)$ as rows. If r_k is a relation, then $r_k(\cdot)$ is the corresponding predicate. A predicate is a function that

maps a tuple to true (or one), if the tuple is part of the relation, and to false (or zero), otherwise. A predicate applied to a particular tuple $r_k(E_z)$ is called a ground atom. A mapping of all possible tuples for all predicates to true or false, i.e., the assignment of true or false to all ground atoms, defines a world W (see Section 3). We now assume the following parameterization for a possible world,

$$P(W|\Theta) = \prod_{k,z} P(r_k(E_z)|\theta_{k,E_z}(\Theta)). \tag{2}$$

Here, Θ is a set of model parameters and $\theta_{k,E_z}(\Theta)$ is a scalar that is a function of the parameters. As it is common in probabilistic databases, Equation (2) assumes that all ground atoms are independent given the model parameters.

4.2 Parameterization

RESCAL [19] is a latent variable method for learning from multi-relational data where Equation (2) is parametrized as

$$\theta_{k,(e_i,e_j)} = \sum_{l_1=1}^{r} \sum_{l_2=1}^{r} R_{k_{l_1,l_2}} a_{i,l_1} a_{j,l_2} = \boldsymbol{a}_i^T R_k \boldsymbol{a}_j. \tag{3}$$

Thus in RESCAL each entity e_i is assigned a vector of r latent variables $\boldsymbol{a}_i$ and the matrix R_k describes how these latent variables interact for a particular relation.

Note that the parameters $\Theta = \{\{\boldsymbol{a}_i\}_{i=1}^n, \{R_k\}_{k=1}^m\}$ are coupled via the unique representation of the entities, what permits the global exchange of information across different relations and across subject/object-occurrences of entities. Here, n is the number of entities and m is the number of relation types. This property is also referred to as collective learning.

4.3 Cost Functions

In RESCAL, one uses a least-squares penalty on the parameters (implying a Gaussian prior distribution) and uses the cost function

$$\sum_{k,E_z} \text{loss}_{k,E_z} + \lambda_A \|A\|_F^2 + \lambda_R \sum_k \|R_k\|_F^2$$

where A is the matrix of latent factors with $(A)_{i,j} = a_{i,j}$ and $\lambda_A \geq 0$ and $\lambda_R \geq 0$ are hyperparameters for the regularization. $\|\cdot\|_F$ is the Frobenius norm. For the loss function there are several options. An appropriate choice for the conditional probability in Equation (2) would be a Bernoulli distribution and we would obtain $\text{loss}_{k,E_z} = -r_k(E_z) \log \theta_{k,E_z} - (1 - r_k(E_z))(1 - \theta_{k,E_z})$. After training, we can interpret $\theta_{k,E_z} \approx P(r_k(E_z) = 1|\theta_{k,E_z})$. Alternatively, one can use a least-squared loss

$$\text{loss}_{k,E_z} = (r_k(E_z) - \theta_{k,E_z})^2$$

with the same interpretation of θ_{k,E_z}. A drawback is that we cannot guarantee that predicted values are nonnegative and upper bounded by one. To overcome this issue, we employ a post-processing step of the form,

$$\hat{\theta}_z^B = \text{sig}_\epsilon(\theta_z^G) \tag{4}$$

where θ_z^G is the parameter derived from the Gaussian model and $\hat{\theta}_z^B$ would be an estimate for the corresponding Bernoulli parameter. The precise definition of $\text{sig}_\epsilon(\theta_z^G)$ can be found in the Appendix. Alternatively, a form of Platt scaling can be used to define $\text{sig}_\epsilon(\theta_z^G)$ [22].

In our work we employ the least squares cost function since then the highly efficient alternating least-squares (ALS) updates can be employed that exploit data sparsity [19]. RESCAL has been scaled up to work with several million entities and close to 100 relation types. In contrast, with the Bernoulli cost function it would not be possible to exploit the sparsity of the data efficiently: gradient-based methods for the Bernoulli cost function would require to compute the dense tensor AR_kA^T explicitly, what is both slow and impractical [18] and pattern-based approaches (such as stochastic gradient descent) have not proven to be effective with a Bernoulli cost function.

4.4 Tensor Factorization

Note, that in Equation (3) we need to optimize the latent entity representations $\boldsymbol{a}_i$ and the relation-specific R_k matrices. This can be done by first factorizing an adjacency tensor $\mathbf{X} \in \{0,1\}^{n \times n \times m}$ whose entries x_{ijk} correspond to all possible ground atoms over n entities and m different relation types. In particular, the entries of $\mathbf{X}$ are set to one, if the ground atom $r_k(E_z)$ exists and to zero otherwise. Figure 1 illustrates the tensor factorization.

5 Querying Factorized Probabilistic Databases

We will describe now how complex queries on PDBs can be answered efficiently via the RESCAL model when the queries are restricted to $\exists x.Q$-type safe queries.

A naive approach would be to use the triple probabilities as calculated by the RESCAL model in the extensional query evaluation rules,

$$P(\mathit{likes}(\mathsf{Jack}, \mathsf{Jane})) = \text{sig}_\epsilon(a_{\mathsf{Jack}}^T R_{\mathit{likes}} a_{\mathsf{Jane}})$$

where sig_ϵ is defined in Equation (4). However, this would not remove the computational complexity of the evaluation and would computationally be demanding for any reasonably sized triple store. A probabilistic database is not sparse in general: e.g., the evaluation of a query the examples used in Section 3.3 requires on the order of $|ADom|$ evaluations of the RESCAL model. The complexity is mainly introduced by the existentially quantified query variables and their product-aggregation in the independent-project rule (Equation (1)). Assuming further that we are not only interested in the probability with $x = \mathsf{Jack}$ but

in a probabilistic ranking of all persons in the database, the total costs become already quadratic in $|ADom|$.

The key idea is to *approximate* safe sub-queries of the type $\exists x.Q$ by employing the RESCAL tensor factorization model. Through this approach, we avoid costly evaluations of sub-queries and additionally construct materialized views that have a memory efficient factorized representation. To illustrate the proposed approach, consider the query example from Section 3.3. This query can be subdivided into two parts, $Q(x,t,z) = Q_1(x,t) \wedge Q_2(x,z)$ with

$$Q_1(x,t) :- \mathit{bornIn}(x,t)$$
$$Q_2(x,z) :- \exists y.(\mathit{likes}(x,y),\ \mathit{childOf}(y,z)).$$

As discussed before, the calculation of Q_2 can become very expensive. To overcome this problem, we approximate the probabilistic answer to Q_2 by a two-step process. First, we create a newly formed compound relation *likesChildOf*, which represents the database view generated by Q_2

$$Q_2(x,z) : -\mathit{likesChildOf}(x,z).$$

To create the compound relation we use the *deterministic* and *sparse* representation of the affected relations from Q_2 and join them into a single relation. This avoids the expensive calculation of the probabilities for each instance of the active domain of y and can be computed efficiently as the deterministic join is not expensive if the (deterministic) domain is sparse. However, the representation of Q_2 would only be based on the available deterministic information so far and would not utilize the probabilistic model of the triple store computed by RESCAL. Hence in a second step we now need to derive probabilities for the triples in the newly formed relation. Fortunately, all that is needed to derive these probabilities under the RESCAL model is to compute a latent representation of the newly created compound relation *likesChildOf*, which can be done very efficiently. In the following, let $X_{(*)}$ denote the newly created compound relation. Furthermore, assume that a meaningful latent representation of the entities has been explored via the factorization of the deterministic triple store. Since the RESCAL model uses a unique representation of entities over *all* relations, all that is needed to derive probabilities for a new relation is to compute its latent representation $R_{(*)}$. The big advantage of the proposed method is that $R_{(*)}$ can now be derived by simply projecting $X_{(*)}$ into the latent space that is spanned by the RESCAL factor matrix A. This can be done very efficiently: Consider the ALS updates derived in [17]. Essentially what is needed is to calculate the latent matrix for the materialized view, i.e., $R_{(*)}$ as

$$R_{(*)} = (Z^T Z + \lambda I)^{-1} Z^T X_{(*)}$$
$$\text{with } Z = A \otimes A.$$

$R_{(*)}$ can be calculated more efficiently by using the following property of the singular value decomposition regarding the Kronecker product [13]

$$A \otimes A = (U \otimes U)(\Sigma \otimes \Sigma)(V^T \otimes V^T)$$

where $A = U\Sigma V^T$. From this property the following update for $R_{(*)}$ can be derived [17],

$$R_{(*)} = V(S \circledast U^T X_{(*)} U)V^T$$

where $\circledast$ represents the Hadamard (element-wise) matrix product and S is defined as

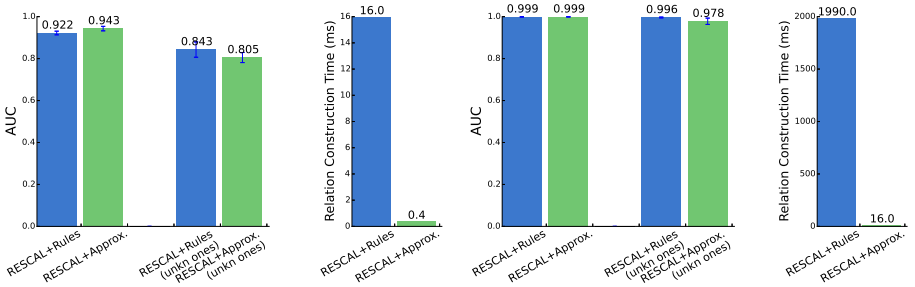
$$[S]_{ij} = \frac{\sigma_i \sigma_j}{\sigma_i^2 \sigma_j^2 + \lambda}$$

where σ_i is the i-th singular value of A. The calculation of $R_{(*)}$ for the newly created relation can now be done in $O(r^3 + nr + pr)$, where p represents the number of nonzero entries in $X_{(*)}$, r represents the rank of the factorization, and where $n \gg r$ represents all entities in the triple database. Please note that the computation of $R_{(*)}$ is now *linear* in all parameters regarding the size of the triple store and cubic only in the number of latent components r that are used to factorize the triple store. As such it can be computed efficiently even for very large triple stores. Furthermore, for each query of the same type the same relational representation can be used, i.e. $R_{(*)}$ only needs to be computed once.

6 Experiments

6.1 Evaluating the Quality of the Approximation

First we conducted experiments on various smaller benchmark datasets. Here, the materialized views (compound relations) can still be computed (and stored) and we can compare the standard approach using materialized views constructed by extensional query evaluation (Section 3.3) with our proposed method that approximates these views. The constructed views that we consider in these experiments range over the join of two relational atoms of the type $\exists y.S(x, y), T(y, z)$. For evaluation, we removed ground tuples only from the deterministic relations S and T and factorized the truncated triple store with RESCAL. From the resulting factorization, we construct the compound relation of S and T by extensional query evaluation or approximated it with our proposed method. For both approaches, we measure the quality of the ranking via the Area Under the Receiver Operating Characteristic Curve (AUC) in two settings: First, we compared the ranking to the full ground truth, i.e. against the ranking constructed from the full triple store(called *all* tuples in the following) including training and test data (evaluation setting all). Second, we compared the ranking to that part of the ground truth that could not have been inferred deterministically from the truncated (training) data, therefore evaluating only the discovery of new tuples (called *unknown* tuples in the following)(evaluation setting unknown). We report the average scores after 10-fold cross-validation. Additionally, we also compared the runtime of both techniques. All experiments were conducted with an Intel(R) Core(TM) i5-3320M CPU @ 2.60GHz.



(a) *negativeBehaviorTo.* (b) Runtime (c) *associatedAndResultOf* (d) Runtime

Fig. 2. Results for the two materialized views *negativeBehaviorToAlliedNationOf* (Nations) and *associatedToResultOf* (UMLS): *RESCAL+Rules*(blue) represents the construction solely with the independent-project rule (Section 3.3). *RESCAL+Approx.*(green) represents the approximation of these views with the method proposed in this work (Section 5). (a,c) The left two bars show the performance (AUC) on all tuples of the deterministic version of the corresponding materialized view (evaluation setting all). The right two bars show the performance (AUC) on the tuples that were unknown at factorization and querying time (evaluation setting unknown).(b,d) Show the runtime for constructing the views for each technique (independent-project rule or approximation).

In the experiments, we used the Nations and the UMLS datasets:
Nations $14 \times 14 \times 56$ multi-relational data that consist of relations between nations (treaties, immigration, etc).
UMLS $135 \times 135 \times 49$ multi-relational data that consist of biomedical relationships between categorized concepts of the Unified Medical Language System.

For the nations dataset we explored the view : "Does x show negative behavior towards an ally of z?", leading to the query

$$\mathit{negativeBehaviorToAlliedNationOf}(x,z) : -\exists y.\mathit{negativeBehaviorTo}(x,y), \mathit{alliedNationOf}(y,z).$$

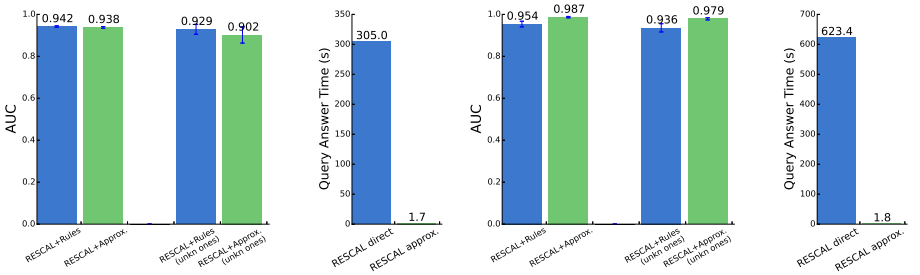
For the UMLS dataset we materialized the view *associatedToAndResultOf* as

$$\mathit{associatedToAndResultOf}(x,z) : -\exists y.\mathit{associatedTo}(x,y), \mathit{resultOf}(y,z).$$

The results of the experiments are shown in Figure 2. Generally, the results in Figure 2.a and Figure 2.b show that both techniques do a good job when constructing the compound relations. Regarding the ranking of all tuples in these views, very high AUC values could be achieved. In case of the UMLS dataset, the score is almost perfect (0.999). Also the discovery of unknown tuples seems to work quite well (right bars in plots a and c). As would have been expected, in both views the materializations based on the independent-project rule seem to work a little bit better than the ones approximated by our proposed method, but the performance is comparable (0.843/0.805 and 0.996/0.978).

When we are looking at the runtime of the materialization of the views, it can be clearly seen that the approximated compound relations are constructed significantly faster than the ones constructed through the independent-project rule. For the compound relation *negativeBehaviorTo* the rules based approach takes 124 times longer (40 times longer for *negativeBehaviorToAlliedNationOf*). This result was expected since the complexity of the approximation is mostly dependent on the rank of the factorization, where the construction through independent-project is cubic in $|ADom|$.[6]

6.2 Evaluating Queries



(a) $Q_1(x)$ (b) Runtime (c) $Q_2(x)$ (d) Runtime

Fig. 3. Results for the queries (a) $Q_1(x)$ and $Q_2(x)$: *RESCAL+Rules*(blue) represents the answering with extensional query evaluation rules (Section 3.3). *RESCAL+Approx.*(green) represents the query answer of a combination of extensional query evaluation rules and the approximation of joined $\exists y.Q$ type relational atoms. (Section 5). (a,c) The left two bars show the quality of the probabilistic ranking (AUC) when compared to all answer tuples returned from the complete deterministic triple store with respect to the corresponding query (evaluation setting all). The right two bars show the ranking quality (AUC) with respect to unknown answer tuples (evaluation setting unknown).(b,d) Shows the runtime for answering the query with each technique (Extensional query evaluation with or without sub-query approximation).

In the second set of experiments, we used a larger triple store, i.e., a sample of the DBpedia dataset[7] [14] covering the musical domain and the task was to answer predefined queries. The queries are answered by using both techniques, rules and the sub-query approximation proposed in this work. We measure the quality of the probabilistic ranking of answer tuples to the different predefined queries in AUC. For evaluation we removed ground tuples from those relations that take part in constructing parts of the query where our method is supposed to

[6] $|ADom|$ products have to be computed for each tuple in the compound relation matrix when applying the independent-project rule.

[7] http://wiki.dbpedia.org/Downloads35?v=pb8

approximate the sub-query. The resulting truncated triple store is then factorized by RESCAL. We compare the query answers inferred solely through extensional query evaluation rules against our method, where sub-queries (compound relations) were approximated instead of applying the independent-project rule. As in the first set of experiments, we compare the ranking of possible answer tuples to the full ground truth, i.e. against the answer tuples inferred from the complete deterministic triple store, including training and test data (called *all* tuples in the following)(evaluation setting all). In addition we compared the ranking to those answer tuples that could not have been inferred deterministically from the truncated (training) data, therefore evaluating only the discovery of new tuples (called *unknown* tuples in the following)(evaluation setting unknown). For the query evaluation process the runtime of both approaches was also compared.

The DBpedia dataset:

DBpedia-music $44345 \times 44345 \times 7$ multi-relational data that consists of relations and entities regarding the musical domain. The relations are *Genre*, *RecordLabel* (*rL*), *associatedMusicalArtist*, *associatedBand*, *musicalArtist* (*mA*), *musicalBand*, *album*. We pre-defined the following queries for the experiments:

- What songs or albums from the Pop-Rock genre are from musical artists that have/had a contract with Atlantic Records?

$$Q_1(x) \quad : \quad -\exists y.(genre(x,z) \ \wedge \ mA(x,y) \ \wedge \ rL(y,\mathsf{Atlantic\_Records})).$$

- Which musical artists from the Hip Hop music genre have/had a contract with Shady (SD), Aftermath (AM) or Death Row (DR) records?

$$Q_2(x) : -\exists y.(genre(x,\mathsf{Hip\_hop\_music}) \wedge mA(x,y) \wedge rL(y,\{\mathsf{SD},\mathsf{AM},\mathsf{DR}\}))$$

- Which musical artists from the Hip Hop music genre are associated with musical artists that have/had a contract with Interscope Records and are involved in an album whose first letter is a "T"?

$$Q_3(x) : -\exists y.(associatedMusicalArtist(x,y) \wedge rL(y,\mathsf{Interscope\_Records})) \wedge genre(x,\mathsf{Hip\_hop\_music}) \wedge \exists z.\exists t.(mA(z,x) \wedge album(z,\{\mathsf{Album\ T.*}\}))$$

Q_1 and Q_2 are calculated as

$$Q_1(x) = \left[1 - \prod_{z \in \{\mathsf{PopRock}\}} 1 - P(genre(x,z))\right] \times \left[1 - \prod_{b \in ADom} 1 - P(mA(x,b) \cdot P(rL(b,\mathsf{Atlantic\_record}))\right].$$

$$Q_2(x) = P(genre(x,\mathsf{Hip\_hop\_music})) \times \left[1 - \prod_{b \in ADom} 1 - P(mA(x,b)\left[1 - \prod_{c \in \{\mathsf{SD},\mathsf{AM},\mathsf{DR}\}} 1 - P(rL(b,c))\right]\right].$$

The computation of Q_3 is omitted, since its structure is a combination of Q_1 and Q_2. For the factorization of the database, a rank of r=100 was used. The results of the experiments for Q_1 and Q_2 are shown in Figure 3. Similar to the previous results, both approaches perform well and comparable in answering the queries. Also the discovery of unknown answer tuples seems to work quite well (above 0.9) for both methods. Regarding the answering time (Figure 3.b,d) it can be clearly seen that the proposed method, which approximates the view on $\mathit{musicalArtist}(x, y) \wedge \mathit{recordLabel}(y, l)$, resulted in a significantly faster response time for both queries, even though the complete join of *musicalArtist* and *recordLabel* was not needed when using the independent-project rule. For Q_1, the query was answered 180 times faster and in case of Q_2 this difference is even greater, because we have a nested independent-project in the query (we ask for multiple record labels). The response time of our method is almost the same for both queries, where the runtime of the answer generated solely through the extensional rules doubles for Q_2. In case of Q_3 (results not shown) this becomes even more dramatic, because there are many potential albums that start with T. As expected, the answer generated by the proposed approach doubles its runtime to 3.6 seconds since Q_3 is a combination of Q_1 and Q_2 (AUC 0.997, 0.985 (unknown answer tuples)), but without sub-query approximation the evaluation of Q_3 did not terminate even after 6 hours runtime! In addition, we constructed a factorized representation of the approximated views which can be stored with almost no cost (100×100 entries $\approx$ 80kByte). In comparison, a materialized view constructed with rules would take 44345×44345 entries $\approx$ 15.7GB.

From the results presented in this section, it can be observed that with the technique introduced in Section 5 we are able to join sub-queries of the form $\exists x.Q$ orders of magnitudes faster and that the join is competitive to a probabilistically correct join generated solely through extensional query evaluation rules (independent-project).

7 Conclusions

In this paper we have demonstrated how a factorized model based on the RESCAL approach can lead to a very efficient representation of the probabilistic database and also can be used to derive the probabilities for the ground tuples. Most importantly, we have shown how efficient querying can be achieved within the RESCAL framework by factorizing a deterministic view at query time.

In general, the approach is not restricted to the $\exists x.Q$ type queries but can always be employed when factorized materialized deterministic views can be used to simplify the probabilistic query. Note that after we perform the deterministic joins, we might obtain views with arities larger than two. This is not a serious problem since our approach is not restricted to binary relations.

Acknowledgements

Maximilian Nickel acknowledges support by the Center for Brains, Minds and Machines (CBMM), funded by NSF STC award CCF-1231216

References

1. Antoine Bordes, Jason Weston, Ronan Collobert, and Yoshua Bengio. Learning structured embeddings of knowledge bases. In *AAAI*, 2011.
2. Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. Mystiq: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893, 2005.
3. Andrea Calì, Thomas Lukasiewicz, Livia Predoiu, and Heiner Stuckenschmidt. Tightly integrated probabilistic description logic programs for representing ontology mappings. In *FoIKS*, pages 178–198, 2008.
4. Paulo Cesar G. da Costa, Kathryn B. Laskey, and Kenneth J. Laskey. Pr-owl: A bayesian ontology language for the semantic web. In *ISWC-URSW*, 2005.
5. Nilesh N. Dalvi, Christopher Re, and Dan Suciu. Queries and materialized views on probabilistic databases. *J. Comput. Syst. Sci.*, 77(3):473–490, 2011.
6. Zhongli Ding, Yun Peng, and Rong Pan. A bayesian approach to uncertainty modelling in owl ontology. In *Proceedings of the International Conference on Advances in Intelligent Systems - Theory and Applications*, 2004.
7. Maximilian Dylla, Iris Miliaraki, and Martin Theobald. Top-k query processing in probabilistic databases with non-materialized views. Research Report MPI-I-2012-5-002, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany, June 2012.
8. Thomas Franz, Antje Schultz, Sergej Sizov, and Steffen Staab. Triplerank: Ranking semantic web data by tensor decomposition. In *International Semantic Web Conference*, pages 213–228, 2009.
9. Rosalba Giugno and Thomas Lukasiewicz. P-shoq(d): A probabilistic extension of shoq(d) for probabilistic ontologies in the semantic web. In *JELIA*, 2002.
10. Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. Maybms: a probabilistic database management system. In *SIGMOD Conference*, 2009.
11. Rodolphe Jenatton, Nicolas Le Roux, Antoine Bordes, and Guillaume Obozinski. A latent factor model for highly multi-relational data. In *NIPS*, 2012.
12. Tamara G. Kolda, Brett W. Bader, and Joseph P. Kenny. Higher-order web link analysis using multilinear algebra. In *ICDM*, pages 242–249, 2005.
13. Alan J. Laub. *Matrix analysis - for scientists and engineers*. SIAM, 2005.
14. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, 2014.
15. Thomas Lukasiewicz. Expressive probabilistic description logics. *Artif. Intell.*, 172(6-7):852–883, 2008.
16. Michi Mutsuzaki, Martin Theobald, Ander de Keijzer, Jennifer Widom, Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Raghotham Murthy, and Tomoe Sugihara. Trio-one: Layering uncertainty and lineage on a conventional dbms (demo). In *CIDR*, pages 269–274, 2007.
17. Maximilian Nickel. *Tensor factorization for relational learning*. PhDThesis, pages 48,49,74, Ludwig-Maximilian-University of Munich, August 2013.
18. Maximilian Nickel and Volker Tresp. Logistic tensor factorization for multi-relational data. In *Structured Learning: Inferring Graphs from Structured and Unstructured Inputs (ICML WS)*, 2013.
19. Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, pages 809–816, 2011.

20. Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. Factorizing yago: scalable machine learning for linked data. In *WWW*, pages 271–280, 2012.
21. Dan Olteanu and Hongkai Wen. Ranking query answers in probabilistic databases: Complexity and efficient algorithms. In *ICDE*, pages 282–293, 2012.
22. John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in large margin classifiers*, pages 61–74. MIT Press, 1999.
23. Steffen Rendle, Leandro Balby Marinho, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *KDD*, pages 727–736, 2009.
24. Sebastian Riedel, Limin Yao, Andrew McCallum, and Benjamin M. Marlin. Relation extraction with matrix factorization and universal schemas. In *HLT-NAACL*, pages 74–84, 2013.
25. Christopher R, Nilesh Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *in ICDE*, pages 886–895, 2007.
26. Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne E. Hambrusch, and Rahul Shah. Orion 2.0: native support for uncertain data. In *SIGMOD Conference*, pages 1239–1242, 2008.
27. Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.
28. Martin Theobald, Luc De Raedt, Maximilian Dylla, Angelika Kimmig, and Iris Miliaraki. 10 years of probabilistic querying - what next? In *ADBIS*, 2013.
29. Volker Tresp, Yi Huang, Markus Bundschus, and Achim Rettinger. Materializing and querying learned knowledge. In *First ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web (IRMLeS 2009)*, 2009.
30. Hendrik Wermser, Achim Rettinger, and Volker Tresp. Modeling and learning context-aware recommendation scenarios using tensor decomposition. In *ASONAM*, pages 137–144, 2011.
31. Yi Yang and Jacques Calmet. Ontobayes: An ontology-driven uncertainty model. In *CIMCA/IAWTIC*, pages 457–463, 2005.

Appendix

The sigmoidal transfer function sig_ϵ is defined as

$$\begin{aligned}
\mathrm{sig}_\epsilon(x) &= x \quad \text{if} \quad \epsilon < x < 1-\epsilon \\
\mathrm{sig}_\epsilon(x) &= \frac{\epsilon}{\exp(1)} \exp(x/\epsilon) \quad \text{if} \quad x \leq \epsilon \\
\mathrm{sig}_\epsilon(x) &= 1 - \frac{\epsilon}{\exp(1)} \exp((1-x)/\epsilon) \quad \text{if} \quad x \geq 1-\epsilon
\end{aligned}$$

and $0 \leq \epsilon \leq 1/2$.

Hereby, ϵ controls the spreading of the non-probabilistic confidence values in the interval [0,1], and which values are affected by the function. Since sig_ϵ is monotonic, the ordering is maintained. Asymptotically the Gaussian estimate converges to the correct probabilities and we can set $\epsilon \to 0$.