Symbolic Querying of Vector Spaces: Probabilistic Databases Meets Relational Embeddings

Tal Friedman
Department of Computer Science
University of California, Los Angeles
tal@cs.ucla.edu

Guy Van den Broeck
Department of Computer Science
University of California, Los Angeles
guyvdb@cs.ucla.edu

Abstract

We propose unifying techniques from probabilistic databases and relational embedding models with the goal of performing complex queries on incomplete and uncertain data. We formalize a probabilistic database model with respect to which all queries are done. This allows us to leverage the rich literature of theory and algorithms from probabilistic databases for solving problems. While this formalization can be used with any relational embedding model, the lack of a well-defined joint probability distribution causes simple query problems to become provably hard. With this in mind, we introduce TRACTOR, a relational embedding model designed to be a tractable probabilistic database, by exploiting typical embedding assumptions within the probabilistic framework. Using a principled, efficient inference algorithm that can be derived from its definition, we empirically demonstrate that TRACTOR is an effective and general model for these querying tasks.

1 INTRODUCTION

Relational database systems are ubiquitous tools for data management due to their ability to answer a wide variety of queries. In particular, languages such as SQL allow one to take advantage of the relational structure of the data to ask complicated question to learn, analyse, and draw conclusions from data. However, traditional database systems are poorly equipped to deal with uncertainty and incompleteness in data. Meanwhile, techniques from the machine learning community can successfully make predictions and infer new facts. In this work we marry ideas from both machine learning and databases to provide a framework for answering such queries while dealing with uncertain and incomplete data.

Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI), PMLR volume 124, 2020.

The first key question we need an answer for when dealing with uncertain relational data is how to handle the fact that our data is invariably incomplete. That is, there will always be facts that we do not explicitly see, but would like to be able to infer. In the machine learning community, this problem is known as *link prediction*, a task which has garnered a lot of attention in recent years [31, 30, 24, 37] using a variety of techniques [4, 15]. Recently, the most common techniques for this problem are relational embedding models, which embed relations and entities as vectors and then use a scoring function to predict whether or not facts are true. While these techniques are popular and have proven effective for link prediction, they lack a consistent underlying probabilistic semantics, which makes their beliefs about the world unclear. As a result, investigations into them have rarely gone beyond link prediction [20, 26].

On the other hand, the databases community has produced a rich body of work for handling uncertainty via probabilistic databases (PDBs). In contrast to relational embedding models which are fundamentally predictive models, PDBs [34, 39] are defined by a probabilistic semantics, with strong and clearly specified independence assumptions. With these semantics, PDBs provide us with a wealth of theoretical and algorithmic research into complex queries, including tractability results [11, 12, 13, 16] and approximations [14, 18]. Recently there has even been work in finding explanations for queries [9, 19], and querying subject to constraints [6, 3, 17]. Where PDBs fall short is in two major areas. Firstly, populating PDBs with meaningful data in an efficient way remains a major challenge, due to their brittleness to incomplete data, and due to their disconnect from the statistical models that can provide these databases with probability values. Secondly, while querying is well understood, certain types of desirable queries are provably hard under standard assumptions [13].

In this work, our goal will be to unify the predictive capability of relational embedding models with the sound un-

derlying probabilistic semantics of probabilistic databases. The central question then becomes how should we do this unification such that we maintain as many of the benefits of each as possible, while finding ways to overcome their limitations. As we will discover in Section 3, this is not a question with an obvious answer. The straightforward option is to simply convert the relational embedding model's prediction into probabilities, and then use these to populate a probabilistic database. While this does give us a meaningful way to populate a PDB, the resulting model is making some clearly problematic independence assumptions, and moreover still struggles with making certain queries tractable.

At its core, the reason this straightforward solution is ineffective is as follows: while both PDBs and relational embedding models make simplifying assumptions, these assumptions are not being taken into account *jointly*. Each is treating the other as a black box. To overcome this, we incorporate the factorization assumption made by many relational embedding models [41, 30] directly into our probabilistic database. The resulting model, which we call TRACTOR, thus takes advantages of the benefits of both: it can efficiently and accurately predict missing facts, but it also provides a probabilistic semantics which we can use for complex probabilistic reasoning. Due to its factorization properties, TRACTOR can even provide efficient reasoning where it was previously difficult in a standard PDB.

The rest of the paper is organized as follows. Section 2 provides the required technical background on PDBs and their associated queries. In Section 3 we discuss using (tuple-independent) PDBs as the technical framework for relational embedding models, as well as giving a brief formalization and discussion of challenges. Then, in Section 4 we introduce TRACTOR, a relational embedding model designed around PDBs to allow for a large range of efficient queries. Section 5 provides an empirical evaluation of TRACTOR . Finally, Section 6 gives a broad discussion on related work along with ties to future work.

2 PROBABILISTIC DATABASES

We now provide the necessary technical background on probabilistic databases, which will serve as the foundation for our probabilistic semantics and formalism for queries, as well as the underlying inspiration for TRACTOR.

2.1 RELATIONAL LOGIC AND DATABASES

We begin with necessary background from *function-free finite-domain* first-order logic. An atom $R(x_1, x_2, ..., x_n)$ consists of a predicate R of arity n, together with n arguments. These arguments can either be *constants* or *variables*. A *ground atom* is an atom that contains no variables. A *formula* is a series of atoms combined with conjunctions ($\wedge$) or disjunctions ($\vee$), and with quantifiers $\forall, \exists$. A *substitution* $\mathsf{Q}[x/t]$ replaces all occurrences of x by t in a formula Q.

A relational *vocabulary* σ is composed of a set of predicates $\mathcal{R}$ and a domain $\mathcal{D}$. Using the *Herbrand semantics* [21], the *Herbrand base* of σ is the set of all ground atoms possible given $\mathcal{R}$ and $\mathcal{D}$. A σ-interpretation ω is then an assignment of truth values to every element of the Herbrand base of σ. We say that ω is a *model* of a formula Q whenever ω satisfies Q. This is denoted by $\omega \models \mathsf{Q}$.

Under the standard model-theoretic view [1], a relational database for a vocabulary σ is a σ-interpretation ω. In words: a relational database is a series of relations, each of which corresponds to a predicate. These are made up by a series of rows, also called *tuples*, each of which corresponds to a ground atom being true. Any atom not appearing as a row in the relation is considered to be *false*, following the closed-world assumption [32]. Figure 1 shows an example database.

2.2 PROBABILISTIC DATABASES

To incorporate uncertainty into relational databases, *probabilistic databases* assign each tuple a probability [34, 39].

Definition 1. A *(tuple-independent) probabilistic database* (PDB) $\mathcal{P}$ for a vocabulary σ is a finite set of tuples of the form $\langle t : p \rangle$ where t is a σ-atom and $p \in [0, 1]$. Furthermore, each t can appear at most once.

Given such a collection of tuples and their probabilities, we are now going to define a *distribution* over relational databases. The semantics of this distribution are given by treating each tuple as an independent random variable.

Definition 2. A PDB $\mathcal{P}$ for vocabulary σ induces a probability distribution over σ-interpretations ω:

$$P_{\mathcal{P}}(\omega) = \prod_{t \in \omega} P_{\mathcal{P}}(t) \prod_{t \notin \omega} (1 - P_{\mathcal{P}}(t))$$

$$\text{where} \quad P_{\mathcal{P}}(t) = \begin{cases} p & \text{if } \langle t : p \rangle \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

Each tuple is treated as an independent Bernoulli random variable, so the probability of a relational database instance is given as a simple product, based on which tuples are or are not included in the instance.

2.3 PROBABILISTIC QUERIES

Much as in relational databases, in probabilistic databases we are interested in answering queries – the difference

Scientist
Einstein
Erdős
von Neumann

CoAuthor	
Einstein	Erdős
Erdős	von Neumann

Figure 1: Example relational database. Notice that the first row of the right table corresponds to the atom CoAuthor(Einstein, Erdős).

Scientist	Pr
Einstein	0.8
Erdős	0.8
von Neumann	0.9
Shakespeare	0.2

CoAuthor		Pr
Einstein	Erdős	0.8
Erdős	von Neumann	0.9
von Neumann	Einstein	0.5

Figure 2: Example probabilistic database. Tuples are now of the form $\langle t : p \rangle$ where p is the probability of the tuple t being present. These tuples are assumed to be independent, so the probability both Einstein and Erdős are scientists is $0.8 \cdot 0.8 = 0.64$.

being that we are now interested in probabilities over queries. In particular, we study the theory of queries that are fully quantified and with no free variables or cosntants, also known as fully quantified *Boolean queries* – we will see later how other queries can be reduced to this form. On a relational database, this corresponds to a fully quantified query that has an answer of True or False.

For example, on the database given in Figure 1, we might ask if there is a scientist who is a coauthor:

$$\mathsf{Q}_1 = \exists x.\exists y.S(x) \wedge CoA(x, y)$$

Which there clearly is, by taking x to be Einstein and y to be Erdős. If we instead asked this query of the PDB in Figure 2, we would be computing the probability by summing over the worlds in which the query is true:

$$P_{\mathcal{P}}(\mathsf{Q}_1) = \sum_{\omega \models \mathsf{Q}_1} P_{\mathcal{P}}(\omega)$$

Queries of this form that are a conjunction of atoms are called *conjunctive queries*. They are commonly shortened as:

$$\mathsf{Q}_1 = S(x), CoA(x, y).$$

A disjunction of conjunctive queries is known as a *union of conjunctive queries* (UCQ). While they capture a rather complex set of queries, the algorithmic landscape of UCQs is remarkably well understood.

Theorem 1. *Dalvi and Suciu [13] Let* Q *be a UCQ and* $\mathcal{P}$ *be a tuple-independent probabilistic database. Then the query* Q *is either:*

- Safe*:* $P_{\mathcal{P}}(\mathsf{Q})$ *can be computed in time polynomial in* $|\mathcal{P}|$ *for all probabilistic databases* $\mathcal{P}$ *using the standard lifted inference algorithm (see Section 2.3.2);*
- Unsafe*: Computing* $P_{\mathcal{P}}(\mathsf{Q})$ *is a* $\#P$*-hard problem.*

Furthermore, we can efficiently determine whether Q *is safe or unsafe.*

In much of the literature of probabilistic databases [34, 13], as well as throughout this paper, UCQs (and consequently conjunctive queries) are the primary query object studied.

2.3.1 Reduction to Fully Quantified Boolean Queries

In general, one is not always interested in computing fully quantified queries. For example, in Section 5 one of the queries we are interested in computing will be of the form

$$\exists x, y.R(A, x) \wedge S(x, y) \wedge T(y, B) \tag{1}$$

For relations R, S, T and constants A, B. To convert this query to a fully quantified one, we need to *shatter* the query [39]. In this case, we replace the binary relation $R(A, x)$ by the unary query $R_A(x)$, where $\forall x.R_A(x) = R(A, x)$. A similar procedure for T gives us the following query:

$$H_0 = \exists x, y.R_A(x) \wedge S(x, y) \wedge T_B(y) \tag{2}$$

This is now a fully quantified query, and is also a simple example of an unsafe query. That is, for an arbitrary probabilistic database $\mathcal{P}$ we cannot compute $P_{\mathcal{P}}(Q)$ in time polynomial in $|\mathcal{P}|$ given our current independence and complexity assumptions.

2.3.2 Efficient Query Evaluation

In addition to providing an underlying probabilistic semantics, one of the motivations for exploring probabilistic databases as the formalism for relational embedding models was to be able to evaluate complex queries efficiently. Algorithm 1 does this in polynomial time for all safe queries. We now explain the steps in further detail.

We begin with the assumption that Q has been processed to not contain any constant symbols, and that all variables appear in the same order in repeated predicate occurrences in Q. This can be done efficiently [13].

Step 0 covers the base case where Q is simply a tuple, so it looks it up in $\mathcal{P}$. *Step 1* attempts to rewrite the UCQ into a conjunction of UCQs to find decomposable parts. For example, the UCQ $(R(x) \wedge S(y, z)) \vee (S(x, y) \wedge T(x))$ can

Algorithm 1 $\textbf{Lift}^{\mathbf{R}}(\mathsf{Q}, \mathcal{P})$, abbreviated by $\mathbf{L}(\mathsf{Q})$

Require: UCQ Q , prob. database $\mathcal{P}$ with constants T.
Ensure: The probability $P_{\mathcal{P}}(\mathsf{Q})$

Step 0 *Base of Recursion*
  if Q is a single ground atom t
    if $\langle t : p \rangle \in \mathcal{P}$ **return** p **else return** 0
Step 1 *Rewriting of Query*
  Convert Q to conjunction of UCQ: $\mathsf{Q}_\wedge = \mathsf{Q}_1 \wedge \cdots \wedge \mathsf{Q}_m$
Step 2 *Decomposable Conjunction*
  if $m > 1$ and $\mathsf{Q}_\wedge = \mathsf{Q}_1 \wedge \mathsf{Q}_2$ where $\mathsf{Q}_1 \perp \mathsf{Q}_2$
    return $\mathbf{L}(\mathsf{Q}_1) \cdot \mathbf{L}(\mathsf{Q}_2)$
Step 3 *Inclusion-Exclusion*
  if $m > 1$ but $\mathsf{Q}_\wedge$ has no independent Q_i
    (Do Cancellations First)
    return $\sum_{s \subseteq [m]} (-1)^{|s|+1} \cdot \mathbf{L}\left(\bigvee_{i \in s} \mathsf{Q}_i\right)$
Step 4 *Decomposable Disjunction*
  if $\mathsf{Q} = \mathsf{Q}_1 \vee \mathsf{Q}_2$ where $\mathsf{Q}_1 \perp \mathsf{Q}_2$
    return $1 - (1 - \mathbf{L}(\mathsf{Q}_1)) \cdot (1 - \mathbf{L}(\mathsf{Q}_2))$
Step 5 *Decomposable Existential Quantifier*
  if Q has a *separator variable* x
    return $1 - \prod_{c \in T} (1 - \mathbf{L}(\mathsf{Q}[x/c]))$
Step 6 *Fail* (the query is #P-hard)

be written as the conjunction of $(R(x)) \vee (S(x,y) \wedge T(x))$ and $(S(y,z)) \vee (S(x,y) \wedge T(x))$. When multiple conjuncts are found this way, there are two options. If they are symbolically independent (share no symbols, denoted $\perp$), then *Step 2* applies independence and recurses. Otherwise, *Step 3* recurses using the inclusion-exclusion principle, performing cancellations first to maintain efficiency [13]. If there is only a single UCQ after rewriting, *Step 4* tries to split it into independent parts, applying independence and recursing if anything is found.

Next, *Step 5* searches for a *separator* variable, one which appears in every atom in Q. If x is a separator variable for Q, and a, b are different constants in the domain of x, this means that $\mathsf{Q}[x/a]$ and $\mathsf{Q}[x/b]$ are independent. This independence is again recursively exploited. Finally, if *Step 6* is reached, then the algorithm has failed and the query provably cannot be computed efficiently [13], under standard complexity assumptions.

3 RELATIONAL EMBEDDINGS AS PROBABILISTIC DATABASES

We now tackle the primary goal of this work: to use probabilistic databases as the formalism for doing probabilistic reasoning with relational embeddings. We begin with some background.

$R(x,y)$	Score
A B	-0.6
B C	0.2
A C	2.3

$\Longrightarrow$

$R(x,y)$	Pr
A B	0.35
B C	0.55
A C	0.91

Figure 3: An example of mapping a relational embedding to a probabilistic database using the sigmoid function.

3.1 RELATIONAL EMBEDDING MODELS

Suppose we have a knowledge base $\mathcal{K}$ consisting of triples (h_i, R_i, t_i), denoting a head entity, relation, and tail entity (equivalently $R_i(h_i, t_i)$ in probabilistic database notation). Relational embedding models aim to learn continuous representations for both entities and relations, which together can be used to predict the presence of a triple. More formally:

Definition 3. Suppose we have a knowledge base $\mathcal{K}$ consisting of triples (h_i, R_i, t_i), with entities $\mathcal{E}$ and relations $\mathcal{R}$. Then a *relational embedding model* consists of

- Real vectors v_R, v_e for all relations $R \in \mathcal{R}$ and entities $e \in \mathcal{E}$
- A scoring function $f(v_h, v_R, v_t) \rightarrow \mathbb{R}$ which induces a ranking over triples

In general, these vectors may need to be reshaped into matrices or tensors before the scoring function can be applied. Table 1 gives some examples of models with the form their vector representations take, as well as their scoring functions.

3.2 PROBABILISTIC INTERPRETATIONS OF RELATIONAL EMBEDDINGS

Given a relational embedding model from Definition 3, if we want to give it a clear probabilistic semantics using our knowledge of probabilistic databases from Section 2, we need to find a way to interpret the model as a probability distribution.

The simplest approach is to choose some mapping function $g : \mathbb{R} \rightarrow [0, 1]$ which converts all the scores produced by the model's scoring function into probabilities. This provides us marginal probabilities, but no obvious joint distribution. Again, we can make the simplest choice and interpret these probabilities as being independent. That is, we can construct a probabilistic database where the probabilities are determined using our mapping function. Figure 3 gives an example of such a conversion, using the sigmoid function as the mapping.

Table 1: Example relational embedding scoring functions for d dimensions

Method	Entity Embedding	Relation Embedding	Triple Score
TransE [5]	$v_h, v_t \in \mathbb{R}^d$	$v_R \in \mathbb{R}^d$	$\|\|v_h + v_R - v_t\|\|$
DistMult [41]	$v_h, v_t \in \mathbb{R}^d$	$v_R \in \mathbb{R}^d$	$\langle v_h, v_R, v_t \rangle$
Rescal [30]	$v_h, v_t \in \mathbb{R}^d$	$v_R \in \mathbb{R}^{d \times d}$	$v_h^T v_R v_t$
ComplEx [37]	$v_h, v_t \in \mathbb{C}^d$	$v_R \in \mathbb{C}^d$	$\text{Re}(\langle v_h, v_R, \bar{v_t} \rangle)$

After doing this conversion, we can directly use Algorithm 1 to efficiently evaluate any safe query. This is a step in the right direction, but there are still two big issues here: firstly, as a simplifying assumption this tuple-independence presents potential issues as discussed in Meilicke et al. [28]. For example, suppose we have a relational model containing *Works-In(Alice, London)* and *Lives-In(Alice, London)*: clearly these triples should not be independent. The second issue, which is perhaps even more critical for our purposes, is that even this assumption is not sufficient for all queries to be tractable:

Theorem 2. *Suppose we have a knowledge base $\mathcal{K}$ with entities $\mathcal{E}$ and relations $\mathcal{R}$. Then, suppose we have a mapping function g and a relational embedding model represented by a scoring function f which is fully expressive. That is, for any configuration of marginal probabilities $P(R(h,t))$ over all possible triples, there is some assignment of entity and relation vectors such that $\forall R, h, t.\ g(f(v_h, v_R, v_t)) = P(R(h,t))$.*

Then for any unsafe query Q, evaluating $P(\mathsf{Q})$ is a $\#P$-hard problem.

4 TRACTOR

The main takeaway from Section 3 is that although useful, interpreting relational embedding models as providing marginals for probabilistic databases still has major challenges. While we do now have a probabilistic semantics for our relational embedding model, the fact that we used the model as a black box means that we wind up treating all triples as independent.The resulting expressiveness and tractability limitations motivate the search for a model which will not be treated as a black box by our probabilistic database semantics. Rather than simply having an arbitrary statistical model which fills in our probabilistic database, we would like to actually exploit properties of this statistical model. To put it another way: a fundamental underpinning of relational embedding models such as DistMult [41] or TransE [5] is that they make simplifying assumptions about how entity and relation vectors relate to link prediction. In Section 3, our probabilistic interpretations of these models had no way of knowing about these simplifying assumptions: now we are going to express them in the language of PDBs.

4.1 FACTORIZING IN PROBABILISTIC DATABASES

Relational embedding models such as DistMult [41] and ComplEx [37], or indeed any model derived from the canonical Polyadic decomposition [22] are built on an assumption about the way in which the tensor representing all triples factorizes. A similar idea has been used in the context of probabilistic first-order logic, where Boolean matrices representing binary relations are rewritten in terms of unary relations to make inference tractable [38]. We will now apply this technique of rewriting binary relations into unary relations as the basis for our relational embedding model.

Suppose we have a binary relation $R(x, y)$, and our model defines a single random variable $E(x)$ for each entity $x \in \mathcal{E}$ as well as a random variable $T(R)$ for relation R. Then we assume that the relation R decomposes in the following way:

$$\forall x, y. R(x, y) \iff E(x) \wedge T(R) \wedge E(y) \qquad (3)$$

We are assuming that all of the model's newly defined variables in E and T are independent random variables, so Equation 3 implies that

$$P(R(x, y)) = P(E(x)) \cdot P(T(R)) \cdot P(E(y))$$

Figure 4 gives an example of probabilities for E and T, with corresponding probabilities for R subject to Equation 3. For example, we compute $P(R(A, B))$ by:

$$\begin{aligned} P(R(A, B)) &= P(E(A)) \cdot P(T(R)) \cdot P(E(B)) \\ &= 0.04 \end{aligned}$$

To incorporate a relation S, we would define an additional $T(S)$ – no new random variable per entity is needed.

There are a few immediate takeaways from the rewrite presented in Equation 3. Firstly, as a result of sharing dependencies in the model, we no longer have that all triples are independent of each other. For example $R(A, B)$ and $S(A, C)$ are not independent as they share a dependency on the random variable $E(A)$. Secondly, although these tuples are no longer independent (which would normally make query evaluation harder), their connection via new

$E(x)$	Pr
A	0.2
B	0.4
C	0.8

T	Pr
R	0.5

$\Longrightarrow$

$R(x,y)$	Pr
A B	0.04
B C	0.16
A C	0.08

Figure 4: Example model tables E, T_R and a few corresponding predictions for R

latent variables E, T actually helps us. By assuming the latent E, T-tuples to be tuple independent, instead of the non-latent R, S-tuples, we are no longer subject to the querying limitations described by Theorem 2. In fact, *any* UCQ can now be computed efficiently over the relations of interest. This will be proven in Section 4.4, but intuitively binary relations must be involved for Algorithm 1 to get stuck, and our rewrite allows us to avoid this.

Of course, the major drawback is that Equation 3 describes an incredibly simple and inexpressive embedding model – we can only associate a single probability with each entity and relation! We address this next.

4.2 MIXTURES & TRACTOR

In a situation such as ours where we have a simple model which is efficient for some task but not expressive, the standard machine learning approach is to employ a mixture model. For example, while tree-shaped graphical models [10] provide efficient learning and inference, they are limited in their expressive capability: so a commonly used alternative is a mixture of such models [27]. Similarly, while Gaussians are limited in their expressiveness, mixture of Gaussian models [36] have found widespread use throughout machine learning. These mixtures can typically approximate any distribution given enough components.

In our case, we will take the model described in Equation 3 as our building block, and use it to create TRACTOR.

Definition 4. TRACTOR with d dimensions is a mixture of d models each constructed from Equation 3. That is, it has tables T_i, E_i analagous to T and E above for each element i of the mixture. Then, for each element i we have

$$\forall x, y. R_i(x, y) \iff E_i(x) \wedge T_i(R) \wedge E_i(y)$$

The probability of any query is then given by TRACTOR as the average of the probabilities of the d mixture components.

Figure 5 gives an example 2-dimensional TRACTOR model, including probabilities for E_1, E_2, T_1, T_2, and corresponding probabilities for materialized relation R. For example, we compute $P(R(A, B))$ by:

$$\begin{aligned} P(R(A,B)) &= \frac{1}{2}(P(E_1(A)) \cdot P(T_1(R)) \cdot P(E_1(B)) \\ &\quad + P(E_2(A)) \cdot P(T_2(R)) \cdot P(E_2(B))) \\ &= 0.17 \end{aligned}$$

We see that the components of the mixture form what we typically think of as dimensions of the vectors of embeddings. For example, in Figure 5 the embedding of entity A is $(E_1(A), E_2(A)) = (0.2, 0.6)$.

4.3 EQUIVALENCE TO DISTMULT

The first question we need to ask about TRACTOR is how effective it is for link prediction.

Theorem 3. *Suppose we have entity embeddings $v_h, v_r \in \mathbb{R}^d$ and relation embedding $v_R \in \mathbb{R}^d$. Then* TRACTOR *and DistMult will assign identical scores (within a constant factor) to the triple (h, R, t) (equivalently $R(h, t)$).*

We already know from Yang et al. [41] that DistMult is effective for link prediction, so TRACTOR must also be.

4.3.1 Positive and Negative Weights

While we have seen that the computation used for link prediction in TRACTOR is identical to that of DistMult, there remains a key difference: TRACTOR has a probabilistic semantics, and thus all parameters must be probabilities. One option here is to indeed force all parameters to be positive, and live with any performance loss incurred. Another option is allowing for negative probabilities in E, T meaning that we can achieve exactly the same link prediction results as DistMult, whose predictive power is well documented [41]. It has been previously shown that probability theory can be consistently extended to negative probabilities [2], and their usefulness has also been documented in the context of probabilistic databases [23, 40]. Furthermore, by adding a simple disjunctive bias term, we can ensure that all fact predictions are indeed positive probabilities. In Section 5 we will explore both options.

4.4 QUERY EVALUATION

Finally, we explore query evaluation for the TRACTOR model. Suppose we have some arbitrary UCQ Q over binary and unary relations, and we would like to compute $P(\mathsf{Q})$ where all binary relations are given by a TRACTOR model. First, we substitute each binary relation according to Equation 3 using TRACTOR tables E and T. What remains is a query Q' which contains only unary relations.

Theorem 4. *Suppose that Q' is a UCQ consisting only of unary relations. Then Q' is safe.*

$E_1(x)$	Pr
A	0.2
B	0.4
C	0.8

T_1	Pr
R	0.5

+

$E_2(x)$	Pr
A	0.6
B	0.5
C	0.2

T_2	Pr
R	1

$\Longrightarrow$

$R(x,y)$		Pr
A	B	0.17
B	C	0.13
A	C	0.10

Figure 5: Example TRACTOR model tables E_1, E_2, T_1, T_2 and a few corresponding predictions for R

Proof. We prove this by showing that Algorithm 1 never fails on Q'. Consider if Q' cannot be rewritten as a conjunction of UCQs. Then each CQ must contain only a single quantified variable, or else that CQ would contain 2 separate connected components (due to all relations unary). Thus, if we ever reach Step 5 of Algorithm 1, each CQ must have a separator. So Q' is safe.

□

5 EMPIRICAL EVALUATION

We will now empirically investigate the effectiveness of TRACTOR as a relational embedding model. As discussed in Section 4.3, for the purposes of link prediction TRACTOR actually turns out to be equivalent to DistMult. While it does have certain limitations regarding asymmetric relations, the overall effectiveness of DistMult for link prediction has been well documented [41], so we will not be evaluating TRACTOR on link prediction. Instead, we will focus on evaluating TRACTOR's performance when computing more advanced queries.[1] While training the models we evaluated, we confirmed that training TRACTOR and DistMult produced the same embeddings and link prediction performance.

5.1 QUERIES & COMPARISON TARGET

As our comparison for evaluation, we will use the graph query embeddings (GQE) [20] framework and evaluation scheme. Fundamentally, GQE differs from TRACTOR in its approach to query prediction. Where TRACTOR is a distribution representing beliefs about the world which can then be queried to produce predictions, GQE treats queries as their own separate prediction task and defines vector operations to specifically be used for conjuctive query prediction. The consequence of this is that where TRACTOR has a single correct way to answer any query (the answer induced by the probability distribution), a method in the style of GQE needs to find a new set of linear algebra tools for each type of query.

In particular, GQE uses geometric transformations as representations for conjunction and existential quantifiers, allowing it to do query prediction via repeated application of these geometric transformations. Hamilton et al. [20] detail further exactly which queries are supported, but put simply it is any conjunctive query that can be represented as a directed acyclic graph with a single sink.

[1] Code at https://github.com/ucla-starai/pdbmeetskge

Table 2: Example CQs and UCQs

$\mathsf{Q}_1(t) =$	$R(A,t)$
$\mathsf{Q}_2(t) =$	$\exists x.R(A,x)$
$\mathsf{Q}_3(t) =$	$\exists x.R(A,x) \wedge S(x,t)$
$\mathsf{Q}_4(t) =$	$\exists x,y.R(A,x) \wedge S(x,y) \wedge T(y,t)$
$\mathsf{Q}_5(t) =$	$R(A,t) \wedge S(B,t)$
$\mathsf{Q}_6(t) =$	$R(A,t) \wedge S(B,t) \wedge T(C,t)$
$\mathsf{Q}_7(t) =$	$\exists x.R(A,x) \wedge S(x,t)$
	$\vee \exists y.R(A,y) \wedge T(y,t)$
$\mathsf{Q}_8(t) =$	$\exists x.R(A,x) \wedge S(x,t) \wedge T(B,t)$
$\mathsf{Q}_9(t) =$	$\exists x.R(A,x) \wedge S(B,x) \wedge T(x,t)$
$\mathsf{Q}_{10}(t) =$	$\exists x_1,y_1.R(A,x_1) \wedge S(x_1,y_1)$
	$\vee \exists x_2,y_2.S(x_2,y_2) \wedge T(y_2,t)$
$\mathsf{Q}_{11}(t) =$	$\exists x,y,z.R(A,x) \wedge S(x,y) \wedge T(y,z)$

To evaluate these models, the first question is which queries should be tested. We describe a query template as follows: R, S, T are placeholder relations, A, B, C placeholder constants, x, y, z quantified variables, and t is the parameterized variable. That is, the goal of the query is to find the entity t which best satisfies the query (in our framework gives the highest probability). Table 5.1 gives a series of example template CQs and UCQs. In Figure 6, we categorize each of these query templates based on their hardness under standard probabilistic database semantics, as well as their compatibility with GQE. Notice that TRACTOR can compute all queries in Figure 6 in time linear in the domain size, including queries $\mathsf{Q}_4, \mathsf{Q}_{11}, \mathsf{Q}_{10}$ which would be $\#P$-hard in a standard tuple-independent probabilistic database. For the sake of comparison, we perform our empirical evaluation using the queries that are also supported by GQE.

5.2 DATASET

For our dataset, we use the same choice in relational data as Hamilton et al. [20]. In that work, two datsets were evaluated on, which were termed bio and reddit respectively. Bio is a dataset consisting of knowledge

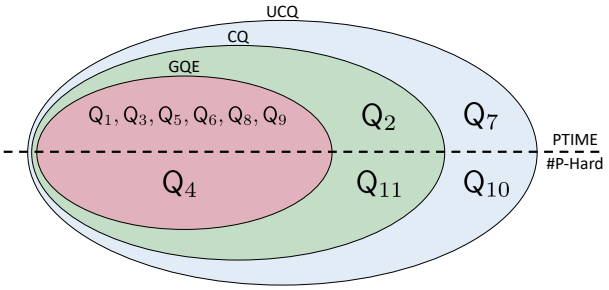

Figure 6: Categorizing different queries based on safeness and compatibility with GQE [20]. TRACTOR efficiently supports all queries in the diagram.

from public biomedical databases, consisting of nodes which correspond to drugs, diseases, proteins, side effects, and biological processes. It includes 42 different relations, and the graph in total has over 8 million edges between 97,000 entities. The reddit dataset was not made publicly available so we were unable to use it for evaluation.

5.2.1 GENERATING QUERIES

While the bio dataset provides our entities and relations, we need to create a dataset of conjunctive queries to evaluate on. For this, we again follow the procedures from Hamilton et al. [20]. First, we sample a 90/10 train/test split for the edges in the bio data. Then, we generate evaluation queries (along with answers) using both train and test edges from the bio dataset, but sample in such a way that each test query relies on at least one edge not present in the training data. This ensures that we can not template match queries based on the training data. For each query template we sample 10,000 queries for evaluation. For further details, including queries for which some edges are adversarially chosen, see Hamilton et al. [20].

As an example, templating on Q_4 can produce:

$$D_?\exists p_1 \exists p_2 \textsc{Activates}(P_3, p_2) \wedge \textsc{Catalyzes}(p_2, p_1) \wedge \textsc{Target}(p_1, D)$$

where D is the drug we would like to find and p_1, p_2, P_3 are proteins.

5.3 EVALUATION

For each evaluation query, we ask the model being evaluated to rank the entity which answers the query in comparison to other entities which do not. We then evaluate the performance of this ranking using a ROC AUC score, as well as an average percentile rank (APR) over 1000 random negative examples.

Table 3: Overall query performance on bio dataset

Method	AUC	APR
Bilinear	79.2	78.6
DistMult	86.7	87.5
TransE	78.3	81.6
TRACTOR+	75.0	84.5
TRACTOR	82.8	86.3

5.3.1 Baselines and Model Variants

We evaluate two versions of our model: TRACTOR indicates a model where the unary predicate probabilities are allowed to be negative, and a bias term is added to ensure all triples have positive predicted probability. TRACTOR+ indicates a model where unary predicate probabilities are constrained to be positive via squaring.

As baselines, we consider model variants from Hamilton et al. [20] that do not include extra parameters that must be trained on queries, as our model contains no such parameters. These models are each built on an existing relational embedding model (Bilinear [30], DistMult [41], and TransE [5] respectively) used for link prediction and composition, as well as a mean vector operator used for queries. For example, for query Q_5, these baselines will make a prediction for t that satisfy $R(a, t)$ and $S(b, t)$ separately, and then take the mean of the resulting vectors.

5.3.2 Training

All model variants and baselines were trained using the max-margin approach with negative sampling [29] which has become standard for training relational embedding models [31]. Parameters were optimized using the Adam optimizer [25], with an embedding dimension of 128, a batch size of 256, and learning rate of 0.01.

5.3.3 Results & Discussion

Table 3 presents AUC and APR scores for all model variants and baselines on the bio dataset. TRACTOR and TRACTOR+ both perform better than TransE and Bilinear based baselines in APR, and are competitive with the DistMult baseline. Evaluating by AUC the performance is slightly worse, but TRACTOR remains better than or comparable to all baselines. These results are very encouraging as TRACTOR is competitive despite the fact that it is representing much more than just conjunctive query prediction. TRACTOR represents a complete probability distribution: effective and efficient query prediction is simply a direct consequence.

Another interesting observation to make here is the gap

between TRACTOR and TRACTOR+, where the only difference is whether the parameters are constrained to be positive. The difference in performance here essentially comes down to the difference in performance on link prediction: not being allowed to use negative values makes the model both less expressive and more difficult to train, leading to worse performance on link prediction. We did not find that increasing the number of dimensions used in the representation to make up for not having negative values helped significantly. Finding ways to improve link prediction subject to this constraint seems to be valuable for improving performance on query prediction.

6 DISCUSSION & RELATED WORK

Querying Relational Embeddings Previous work studying queries beyond link prediction in relational embedding models proposed to replace logical operators with geometric transformations [20], and learning new relations representing joins [26]. Our work differs from these in that we formalize an underlying probabilistic framework which defines algorithms for doing querying, rather than treating querying as a new learning task.

Symmetric Relations A limitation of the TRACTOR model which also appears in models like DistMult [41] and TransE [5] is that since head and tail entities are treated the same way, they can only represent symmetric relations. This is, of course, problematic as many relations we encounter in the wild are not. Solutions to this include assigning complex numbers for embeddings with an asymmetric scoring function [37], and keeping separate head and tail representations but using inverse relations to train them jointly [24]. Borrowing these techniques presents a straightforward way to extend TRACTOR to represent asymmetric relations.

Probabilistic Training One potential disconnect in TRACTOR is that while it is a probabilistic model, it is not trained in a probabilistic way. That is, it is trained in the standard fashion for relational embedding models using negative sampling and a max-margin loss. Other training methods for these models such as cross-entropy losses exist and can improve performance [33] while being more probabilistic in nature. In a similar vein, Tabacof and Costabello [35] empirically calibrates probabilities to be meaingful with respect to the data. An interesting open question is if TRACTOR can be trained directly using a likelihood derived from its PDB semantics.

Incomplete Knowledge Bases One of the main goals of this work is to overcome the common issue of *incomplete* knowledge. That is, what do we do when no probability at all is known for some fact. In this work, we directly incorporate machine learning models to overcome this. Another approach to this problem is to suppose a range of possibilities for our unknown probabilities, and reason over those. This is implemented via open-world probabilistic databases [8], with extensions to incorporate background information in the form of ontological knowledge [7] and summary statistics [17].

Increasing Model Complexity TRACTOR is a mixture of very simple models. While this makes for highly efficient querying, accuracy could potentially be improved by rolling more of the complexity into each individual model at the PDB level. The natural approach to this is to follow Van den Broeck and Darwiche [38] and replace our simple unary conjunction with a disjunction of conjunctions. This raises interesting theoretical and algorithmic questions with potential for improving query prediction.

Further Queries Finally, there are further question one can ask of a PDB beyond the probability of a query. For example, Gribkoff et al. [19] poses the question of which world (i.e. configuration of tuple truths) is most likely given a PDB and some constraints, while Ceylan et al. [9] studies the question of which explanations are most probable for a certain PDB query being true. Extending these problems to the realm of relational embeddings poses many interesting questions.

Acknowledgements

We thank Yitao Liang, YooJung Choi, Pasha Khosravi, Dan Suciu, and Pasquale Minervini for helpful feedback and discussion. This work is partially supported by NSF grants #IIS-1943641, #IIS-1633857, #CCF-1837129, DARPA XAI grant #N66001-17-2-4032, and gifts from Intel and Facebook Research.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. Foundations of databases. 1995.

[2] M. S. Bartlett. Negative probability. *Mathematical Proceedings of the Cambridge Philosophical Society*, 41(1): 71–73, 1945.

[3] Meghyn Bienvenu. Ontology-mediated query answering: Harnessing knowledge to get more from data. In *IJCAI*, 2016.

[4] Hendrik Blockeel and Luc De Raedt. Top-down induction of logical decision trees. 1997.

[5] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NIPS*, 2013.

[6] Stefan Borgwardt, Ismail Ilkan Ceylan, and Thomas Lukasiewicz. Ontology-mediated queries for probabilistic databases. In *AAAI*, 2017.

[7] Stefan Borgwardt, Ismail Ilkan Ceylan, and Thomas Lukasiewicz. Ontology-mediated query answering over log-linear probabilistic data. In *AAAI*, 2019.

[8] Ismail Ilkan Ceylan, Adnan Darwiche, and Guy Van den Broeck. Open-world probabilistic databases. In *KR*, May 2016.

[9] Ismail Ilkan Ceylan, Stefan Borgwardt, and Thomas Lukasiewicz. Most probable explanations for probabilistic database queries. In *IJCAI*, 2017.

[10] C. K. Chow and C. N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Trans. Information Theory*, 14:462–467, 1968.

[11] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 2004.

[12] Nilesh N. Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, 2007.

[13] Nilesh N. Dalvi and Dan Suciu. The dichotomy of probabilistic inference for unions of conjunctive queries. *J. ACM*, 59:30:1–30:87, 2012.

[14] Maarten Van den Heuvel, Peter Ivanov, Wolfgang Gatterbauer, Floris Geerts, and Martin Theobald. Anytime approximation in probabilistic databases via scaled dissociations. In *SIGMOD Conference*, 2019.

[15] Sebastijan Dumancic, Alberto García-Durán, and Mathias Niepert. A comparative study of distributional and symbolic paradigms for relational learning. In *IJCAI*, 2019.

[16] Robert Fink and Dan Olteanu. Dichotomies for queries with negation in probabilistic databases. *ACM Trans. Database Syst.*, 41:4:1–4:47, 2016.

[17] Tal Friedman and Guy Van den Broeck. On constrained open-world probabilistic databases. In *IJCAI*, aug 2019.

[18] Eric Gribkoff and Dan Suciu. Slimshot: In-database probabilistic inference for knowledge bases. *PVLDB*, 9:552–563, 2016.

[19] Eric Gribkoff, Guy Van den Broeck, and Dan Suciu. The most probable database problem. *Proc. BUDA*, 2014.

[20] William L. Hamilton, Payal Bajaj, Marinka Zitnik, Daniel Jurafsky, and Jure Leskovec. Embedding logical queries on knowledge graphs. In *NeurIPS*, 2018.

[21] Timothy Hinrichs and Michael Genesereth. Herbrand logic. Technical Report LG-2006-02, Stanford University, 2006.

[22] Frank L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[23] Abhay Jha and Dan Suciu. Probabilistic databases with markoviews. *PVLDB*, 5:1160–1171, 2012.

[24] Seyed Mehran Kazemi and David Poole. Simple embedding for link prediction in knowledge graphs. In *NeurIPS*, 2018.

[25] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[26] Denis Krompass, Maximilian Nickel, and Volker Tresp. Querying factorized probabilistic triple databases. In *ISWC*, 2014.

[27] Marina Meila and Michael I. Jordan. Learning with mixtures of trees. *J. Mach. Learn. Res.*, 1:1–48, 1998.

[28] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, 2019.

[29] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[30] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, 2011.

[31] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104: 11–33, 2015.

[32] Raymond Reiter. On closed world data bases. In *Readings in artificial intelligence*, pages 119–140. Elsevier, 1981.

[33] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *ICLR*, 2020.

[34] Dan Suciu, Dan Olteanu, R. Christopher, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

[35] Pedro Tabacof and Luca Costabello. Probability calibration for knowledge graph embedding models. In *ICLR*, 2020.

[36] D.M. Titterington, A.F.M. Smith, and U.E. Makov. *Statistical Analysis of Finite Mixture Distributions*. Wiley, New York, 1985.

[37] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[38] Guy Van den Broeck and Adnan Darwiche. On the complexity and approximation of binary evidence in lifted inference. In *NIPS*, December 2013.

[39] Guy Van den Broeck and Dan Suciu. *Query Processing on Probabilistic Data: A Survey*. Foundations and Trends in Databases. Now Publishers, August 2017.

[40] Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. Skolemization for weighted first-order model counting. In *KR*, July 2014.

[41] Bishan Yang, Wen tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *CoRR*, abs/1412.6575, 2014.