UNIVERSITY OF CALIFORNIA

Los Angeles

Insights from the Intersection of Logic and Probabilistic Reasoning

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Tal Friedman

2021

ABSTRACT OF THE DISSERTATION

Insights from the Intersection of Logic and Probabilistic Reasoning

by

Tal Friedman
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 2021
Professor Guy Van den Broeck, Chair

Logic and uncertainty form two of the primary pillars of modern artificial intelligence. Seeking to draw insights about what can be gained by understanding both, we explore different contexts where both are crucial. First, we use logical circuits as our underlying machinery, developing a hybrid circuit sampling technique for approximate probabilistic inference, as well as an axiomatized method for enforcing logical constraints when learning a deep neural network. We then explore modifying probabilistic databases, incorporating schema level constraints to overcome lack of data, as well as demonstrating how and why to directly incorporate them with relational machine learning techniques for free efficient inference on well tuned models.

The dissertation of Tal Friedman is approved.

Quanquan Gu

Yizhou Sun

Todd Millstein

Guy Van den Broeck, Committee Chair

University of California, Los Angeles

2021

For Ima and Aba

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This has been a challenging journey, and I am fortunate to have so many people in my life who helped me through it.

I am extremely grateful to have had Guy as my advisor. You managed to give me creative freedom, constructive feedback, and a gentle shove precisely when I needed each. You were also an endless source of interesting ideas and research discussions, and were always a model for doing what is right in our community.

The members of the StarAI lab both past and present made coming in to the office fun: Kareem, Pasha, Zhe, Antonio, Honghua, Anton, Meihua, Paolo, and Anji. I am especially grateful to YooJung, Steven, Yitao, and Jason, who were there from the start and with whom I have shared every up and down of the past 5 years. I am also indebted to Arthur, who was the first person to welcome me to UCLA, and without whom I might still be hacking away at my first project to this day.

Getting completely wrapped up in the PhD is easy, and I am fortunate to have had the overthonkers discord: a wonderful group of friends who made sure that never happened to me. Jon, Freddy, Axel, Sebo, Idan, Kevin, and Jay: thanks for the countless late nights and memories, I am still yet to find anything a few games and laughs with you all couldn't fix.

Christina: your love and support has been the foundation for everything I did here. Four years apart was a long time, but I have learned to appreciate you so much more. I would also be remiss not to mention Waffles, who has come into our lives and made colours somehow look brighter.

Finally, I am eternally grateful to Ima, Aba, Nir, and Saar. You taught me to be curious, ambitious, and kind, and gave me the unconditional love and encouragement I needed.

VITA

2012-2016 B.Sc. (Computer Science & Mathematics) University of Toronto

PUBLICATIONS

Tal Friedman and Guy Van den Broeck. Symbolic Querying of Vector Spaces: Probabilistic Databases Meets Relational Embeddings. *In Proceedings of the 36th Conference on Uncertainty in Aritifical Intelligence (UAI), 2020.*

Tal Friedman and Guy Van den Broeck. On Constrained Open-World Probabilistic Databases. *In Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI), 2019.*

Tal Friedman and Guy Van den Broeck. Approximate Knowledge Compilation by Online Collapsed Importance Sampling. *In Advances in Neural Information Processing Systems 31 (NeurIPS), 2018.*

Jingyi Xu, Zilu Zhang, **Tal Friedman**, Yitao Liang and Guy Van den Broeck. A Semantic Loss Function for Deep Learning with Symbolic Knowledge. *In Proceedings of the 35th International Conference on Machine Learning (ICML), 2018.*

CHAPTER 1

Introduction

Logic is among the most the most fundamental and critical tools available to us in computer science. Modern artificial intelligence appears to be playing a progressively larger role in our lives, but has in large part shifted away from heavy use of logic [Darwiche, 2020]. This work explores what insights can be drawn by developing an understanding of both, and using them in concert.

A frequently touted benefit of symbolic AI is that explicitly well defined models lead to a better understanding from the researcher. So what is the upshot of this in terms of solving problems? An understanding of how to build and manipulate the logical structures used for probabilistic inference means we will craft samplers built on advanced sampling techniques. Understanding the semantics and algorithms behind models used for uncertain relational data means we will make progress in data challenges without ever touching the data. Through this all is the background thread of machine learning: how can logic shore up weaknesses of machine learning? Sometimes there are things we know but the data does not: we will show logic provides us the tools to inform our learning algorithms. For relational data, machine learning excels at finding effective representations to make accurate predictions about unknown data. Weaving an understanding of both these representations and those provided by the databases community will allow us to reap the rewards of both.

Throughout this work, the focus is interleaving logic with other tools towards a goal of probabilistic inference. We do this in a few different settings, focusing on two sets of machinery: circuits and databases. First, in Chapter 2 we explore using logic as form of background information for training neural networks. Circuits are used to define and compute a loss function on the end of the neural network describing how close an output is to the logical constraint - this allows us to tackle

more difficult combinatorial problems, as well as achieve good performance in a semi supervised setting. Then, in Chapter 3 we integrate circuits and importance sampling to overcome a classic challenge in circuits: their size. We build an algorithm whose samples are themselves smaller circuits, allowing us to make use of them even in contexts where computational cost would normally be prohibitive.

The second half of this work focuses on incorporating logic and probability towards modelling uncertainty in databases, using the machinery of probabilistic databases [Suciu et al., 2011]. We explore ways to overcome the need for a complete and well calibrated set of probabilities to perform meaningful inference on probabilistic databases, relying on these foundations in logic and probability. In Chapter 4 we augment existing open-world probabilistic database semantics from Ceylan et al. [2016a], allowing us to reason about unknown entries using summary statistics. We formally develop the model, providing new algorithms for query evaluation and studying resulting changes to the complexity landscape. Then, in Chapter 5 we integrate techniques from relational machine learning with probabilistic databases, leading to a learned database with clear querying semantics that admits efficient inference for a larger class of queries.

1.1 Contribution Details

Chapter 2 of this document represents work from Xu et al. [2018] done in collaboration with Jingyi Xu, Zilu Zhang, Yitao Liang and Guy Van den Broeck. I conducted experiments for and wrote Section 5, and contributed to writing the rest of the paper. Chapter 3 of this document represents work from Friedman and Van den Broeck [2018]. I did all the work and wrote the paper, with guidance and direction from Guy. The same is true for Chapter 4 and Friedman and Van den Broeck [2019], as well as Chapter 5 and Friedman and Van den Broeck [2020].

CHAPTER 2

A Semantic Loss Function for Deep Learning with Symbolic Knowledge

2.1 Introduction

The widespread success of representation learning raises the question of which AI tasks are amenable to deep learning, which tasks require classical model-based symbolic reasoning, and whether we can benefit from a tighter integration of both approaches. In recent years, significant effort has gone towards various ways of using representation learning to solve tasks that were previously tackled by symbolic methods. Such efforts include neural computers or differentiable programming [Weston et al., 2014, Reed and De Freitas, 2015, Graves et al., 2016, Riedel et al., 2016], relational embeddings or deep learning for graph data [Yang et al., 2014b, Lin et al., 2015, Bordes et al., 2013, Neelakantan et al., 2015, Duvenaud et al., 2015, Niepert et al., 2016], neural theorem proving, and learning with constraints [Hu et al., 2016, Stewart and Ermon, 2017, Minervini et al., 2017, Wang et al., 2017].

This paper considers learning in domains where we have symbolic knowledge connecting the different outputs of a neural network. This knowledge takes the form of a constraint (or sentence) in Boolean logic. It can be as simple as an exactly-one constraint for one-hot output encodings, or as complex as a structured output prediction constraint for intricate combinatorial objects such as rankings, subgraphs, or paths. Our goal is to augment neural networks with the ability to learn how to make predictions subject to these constraints, and use the symbolic knowledge to improve the learning performance.

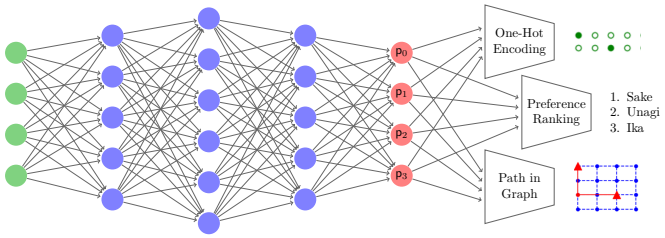


Figure 2.1: Outputs of a neural network feed into semantic loss functions for constraints representing a one-hot encoding, a total ranking of preferences, and paths in a grid graph.

Most neuro-symbolic approaches aim to simulate or learn symbolic reasoning in an end-to-end deep neural network, or capture symbolic knowledge in a vector-space embedding. This choice is partly motivated by the need for smooth *differentiable* models; adding symbolic reasoning code (e.g., SAT solvers) to a deep learning pipeline destroys this property. Unfortunately, while making reasoning differentiable, the precise logical meaning of the knowledge is often lost. In this paper, we take a distinctly unique approach, and tackle the problem of differentiable but sound logical reasoning from first principles. Starting from a set of intuitive axioms, we derive the differentiable *semantic loss* which captures how well the outputs of a neural network match a given constraint. This function precisely captures the *meaning* of the constraint, and is independent of its *syntax*.

Next, we show how semantic loss gives significant *practical improvements* in semi-supervised classification. In this setting, semantic loss for the exactly-one constraint permits us to obtain a learning signal from vast amounts of unlabeled data. The key idea is that semantic loss helps us improve how confidently we are able to classify the unlabeled data. This simple addition to the loss function of standard deep learning architectures yields (near-)state-of-the-art performance in semi-supervised classification on MNIST, FASHION, and CIFAR-10 datasets.

Our final set of experiments study the benefits of semantic loss for learning tasks with highly structured output, such as preference learning and path prediction in a graph [Daumé et al., 2009, Chang et al., 2013, Choi et al., 2015, Graves et al., 2016]. In these scenarios, the task is two-fold: learn both the structure of the output space, and the actual classification function within that space. By capturing the structure of the output space with logical constraints, and minimizing semantic

loss for this constraint during learning, we are able to learn networks that are much more likely to correctly predict structured objects.

2.2 Background and Notation

To formally define semantic loss, we make use of concepts in propositional logic. We write uppercase letters (X,Y) for Boolean variables and lowercase letters (x,y) for their instantiation ($X = 0$ or $X = 1$). Sets of variables are written in bold uppercase ($\mathbf{X}$,$\mathbf{Y}$), and their joint instantiation in bold lowercase ($\mathbf{x}$,$\mathbf{y}$). A literal is a variable (X) or its negation ($\neg X$). A logical sentence (α or β) is constructed in the usual way, from variables and logical connectives ($\wedge$, $\vee$, etc.), and is also called a formula or constraint. A state or world $\mathbf{x}$ is an instantiation to all variables $\mathbf{X}$. A state $\mathbf{x}$ satisfies a sentence α, denoted $\mathbf{x} \models \alpha$, if the sentence evaluates to be true in that world, as defined in the usual way. A sentence α entails another sentence β, denoted $\alpha \models \beta$ if all worlds that satisfy α also satisfy β. A sentence α is logically equivalent to sentence β, denoted $\alpha \equiv \beta$, if both $\alpha \models \beta$ and $\beta \models \alpha$.

The output row vector of a neural net is denoted p. Each value in p represents the probability of an output and falls in $[0, 1]$. We use both softmax and sigmoid units for our output activation functions. The notation for states $\mathbf{x}$ is used to refer the an assignment, the logical sentence enforcing that assignment, or the binary output vector capturing that same assignment, as these are all equivalent notions.

Figure 2.1 illustrates the three different concrete output constraints of varying difficulty that are studied in our experiments. First, we examine the exactly-one or *one-hot constraint* capturing the encoding used in multi-class classification. It states that for a set of indicators $\mathbf{X} = \{X_1, \ldots, X_n\}$, one and exactly one of those indicators must be true, with the rest being false. This is enforced through a logical constraint α by conjoining sentences of the form $\neg X_1 \vee \neg X_2$ for all pairs of variables (at most one variable is true), and a single sentence $X_1 \vee \cdots \vee X_n$ (at least one variable is true). Our experiments further examine the *valid simple path constraint*. It states for a given source-destination pair and edge indicators that the edge indicators set to true must form a valid

simple path from source to destination. Finally, we explore the *ordering constraint*, which requires that a set of n^2 indicator variables represent a total ordering over n variables, effectively encoding a permutation matrix. For a full description of the path and ordering constraints, we refer to Section 2.5.

2.3 Semantic Loss

In this section, we formally introduce semantic loss. We begin by giving the definition and our intuition behind it. This definition itself provides all of the necessary mechanics for enforcing constraints, and is sufficient for the understanding of our experiments in Sections 4 and 5. We also show that semantic loss is not just an arbitrary definition, but rather is defined uniquely by a set of intuitive assumptions. After stating the assumptions formally, we then provide an axiomatic proof of the uniqueness of semantic loss in satisfying these assumptions.

2.3.1 Definition

The semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$ is a function of a sentence α in propositional logic, defined over variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, and a vector of probabilities p for the same variables $\mathbf{X}$. Element p_i denotes the predicted probability of variable X_i, and corresponds to a single output of the neural net. For example, the semantic loss between the one-hot constraint from the previous section, and a neural net output vector p, is intended to capture how close the prediction p is to having exactly one output set to true (i.e. 1), and all others set to false (i.e. 0), regardless of which output is correct. The formal definition of this is as follows:

Definition 1 (Semantic Loss)**.** *Let* p *be a vector of probabilities, one for each variable in* $\mathbf{X}$*, and let* α *be a sentence over* $\mathbf{X}$*. The semantic loss between* α *and* p *is*

$$\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \propto -\log \sum_{\mathbf{x} \models \alpha} \prod_{i : \mathbf{x} \models X_i} \mathsf{p}_i \prod_{i : \mathbf{x} \models \neg X_i} (1 - \mathsf{p}_i).$$

Intuitively, the semantic loss is proportional to a negative logarithm of the probability of generating a state that satisfies the constraint, when sampling values according to p. Hence, it is the self-information (or "surprise") of obtaining an assignment that satisfies the constraint [Jones, 1979].

2.3.2 Derivation from First Principles

In this section, we begin with a theorem stating the uniqueness of semantic loss, as fixed by a series of axioms. The full set of axioms and the derivation of the precise semantic loss function is described in Appendix A.1[1].

Theorem 1 (Uniqueness)**.** *The semantic loss function in Definition 1 satisfies all axioms in Appendix A.1 and is the only function that does so, up to a multiplicative constant.*

In the remainder of this section, we provide a selection of the most intuitive axioms from Appendix A.1, as well as some key properties.

First, to retain logical meaning, we postulate that semantic loss is monotone in the order of implication.

Axiom 1 (Monotonicity)**.** If $\alpha \models \beta$, then the semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{p})$ for any vector p.

Intuitively, as we add stricter requirements to the logical constraint, going from β to α and making it harder to satisfy, the semantic loss cannot decrease. For example, when β enforces the output of an neural network to encode a subtree of a graph, and we tighten that requirement in α to be a path, the semantic loss cannot decrease. Every path is also a tree and any solution to α is a solution to β.

A direct consequence following the monotonicity axiom is that logically equivalent sentences must incur an identical semantic loss for the same probability vector p. Hence, the semantic loss is indeed a semantic property of the logical sentence, and *does not depend on its syntax*.

[1] Appendices are included in the supplementary material.

Proposition 2 (Semantic Equivalence)**.** *If* $\alpha \equiv \beta$*, then the semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{p})$ *for any vector* p*.*

Another consequence is that semantic loss must be non-negative if we want the loss to be 0 for a true sentence.

Next, we state axioms establishing a correspondence between logical constraints and data. A state $\mathbf{x}$ can be equivalently represented as both a binary data vector, as well as a logical constraint that enforces a value for every variable in $\mathbf{X}$. When both the constraint and the predicted vector represent the same state (for example, $X_1 \wedge \neg X_2 \wedge X_3$ vs. $[1\,0\,1]$), there should be no semantic loss.

Axiom 2 (Identity)**.** For any state $\mathbf{x}$, there is zero semantic loss between its representation as a sentence, and its representation as a deterministic vector: $\forall \mathbf{x}, \mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) = 0$.

The axiom above together with the monotonicity axiom imply that any vector satisfying the constraint must incur zero loss. For example, when our constraint α requires that the output vector encodes an arbitrary total ranking, and the vector $\mathbf{x}$ correctly represents a single specific total ranking, there is no semantic loss.

Proposition 3 (Satisfaction)**.** *If* $\mathbf{x} \models \alpha$*, then the semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x}) = 0$*.*

As a special case, logical literals (X or $\neg X$) constrain a single variable to take on a value, and thus play a role similar to the labels used in supervised learning. Such constraints require an even tighter correspondence: the semantic loss must act like a classical loss function (i.e., cross entropy).

Axiom 3 (Label-Literal Correspondence)**.** The semantic loss of a single literal is proportionate to the cross-entropy loss for the equivalent data label: $\mathrm{L}^{\mathrm{s}}(X, p) \propto -\log(p)$ and $\mathrm{L}^{\mathrm{s}}(\neg X, p) \propto -\log(1-p)$.

Appendix A.1 states additional axioms that allow us to prove the following form of the semantic loss for a state $\mathbf{x}$.

Lemma 4. *For state* $\mathbf{x}$ *and vector* p*, we have* $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathsf{p}) \propto -\sum_{i:\mathbf{x}\models X_i} \log \mathsf{p}_i - \sum_{i:\mathbf{x}\models \neg X_i} \log(1 - \mathsf{p}_i)$.

Lemma 4 falls short as a full definition of semantic loss for arbitrary sentences. One can define additional axioms to pin down L^{s}. For example, the following axiom is satisfied by Definition 1,

and is highly desirable for learning.

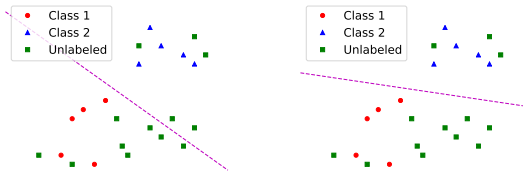
Axiom 4 (Differentiability)**.** For any fixed α, the semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$ is monotone in each probability in p, continuous and differentiable.

Appendix A.1 makes the notion of semantic loss precise by stating one additional axiom. It is based on the observation that the state loss of Lemma 4 is proportionate to a log-probability. In particular, it corresponds to the probability of obtaining state $\mathbf{x}$ after independently sampling each X_i with probability p_i. We have now derived the semantic loss function from first principles, and arrived at Definition 1. Moreover, we can show that Theorem 1 holds - that it is the only choice of such a loss function.

2.4 Semi-Supervised Classification

The most straightforward constraint that is ubiquitous in classification is mutual exclusion over one-hot-encoded outputs. That is, for a given example, exactly one class and therefore exactly one binary indicator must be true. The machine learning community has made great strides on this task, due to the invention of assorted deep learning representations and their associated regularization terms [Krizhevsky et al., 2012, He et al., 2016]. Many of these models take large amounts of labeled data for granted, and big data is indispensable for discovering accurate representations [Hastie et al., 2009]. To sustain this progress, and alleviate the need for more labeled data, there is a growing interest into utilizing unlabeled data to augment the predictive power of classifiers [Stewart and Ermon, 2017, Bilenko et al., 2004]. This section shows why semantic loss naturally qualifies for this task.

Illustrative Example To illustrate the benefit of semantic loss in the semi-supervised setting, we begin our discussion with a small toy example. Consider a binary classification task; see Figure 2.2. Ignoring the unlabeled examples, a simple linear classifier learns to distinguish the two classes by separating the labeled examples (Figure 2.2a). However, the unlabeled examples are also



(a) Trained w/o semantic loss (b) Trained with semantic loss

Figure 2.2: Binary classification toy example: a linear classifier without and with semantic loss.

informative, as they must carry some properties that give them a particular label. This is the crux of semantic loss for semi-supervised learning: a model must confidently assign a consistent class even to unlabeled data. Encouraging the model to do so results in a more accurate decision boundary (Figure 2.2b).

2.4.1 Method

Our proposed method intends to be generally applicable and compatible with any feedforward neural net. Semantic loss is simply another regularization term that can directly be plugged into an existing loss function. More specifically, with some weight w, the new overall loss becomes

$$\text{existing loss} + w \cdot \text{semantic loss}.$$

When the constraint over the output space is simple (for example, there is a small number of solutions $\mathbf{x} \models \alpha$), semantic loss can be directly computed using Definition 1. Concretely, for the exactly-one constraint used in n-class classification, semantic loss reduces to

$$\mathrm{L}^{\mathrm{s}}(\text{exactly-one}, \mathsf{p}) \propto -\log \sum_{i=1}^{n} \mathsf{p}_i \prod_{j=1, j\neq i}^{n} (1 - \mathsf{p}_j),$$

where values p_i denote the probability of class i as predicted by the neural net. Semantic loss for the exactly-one constraint is efficient and causes no noticeable computational overhead in our

experiments.

In general, for arbitrary constraints α, semantic loss is not efficient to compute using Definition 1, and more advanced automated reasoning is required. Section 2.5 discusses this issue in more detail. For example, using automated reasoning can reduce the time complexity to compute semantic loss for the exactly-one constraint from $O(n^2)$ (as shown above), to $O(n)$.

2.4.2 Experimental Evaluation

In this section, we evaluate semantic loss in the semi-supervised setting by comparing it with several competitive models.[2] As most semi-supervised learners build on a supervised learner, changing the underlying model significantly affects the semi-supervised learner's performance. For comparison, we add semantic loss to the same base models used in ladder nets [Rasmus et al., 2015], which currently achieves state-of-the-art results on semi-supervised MNIST and CIFAR-10 [Krizhevsky, 2009]. Specifically, the MNIST base model is a fully-connected multilayer perceptron (MLP), with layers of size 784-1000-500-250-250-250-10. On CIFAR-10, it is a 10-layer convolutional neural network (CNN) with 3-by-3 padded filters. After every 3 layers, features are subject to a 2-by-2 max-pool layer with strides of 2. Furthermore, we use ReLu [Nair and Hinton, 2010], batch normalization [Ioffe and Szegedy, 2015], and Adam optimization [Kingma and Ba, 2015] with a learning rate of 0.002. We refer to Appendix A.2 and A.3 for a specification of the CNN model and additional details about hyper-parameter tuning.

For all semi-supervised experiments, we use the standard 10,000 held-out test examples provided in the original datasets and randomly pick 10,000 from the standard 60,000 training examples (50,000 for CIFAR-10) as validation set. For values of N that depend on the experiment, we retain N randomly chosen labeled examples from the training set, and remove labels from the rest. We balance classes in the labeled samples to ensure no particular class is over-represented. Images are preprocessed for standardization and Gaussian noise is added to every pixel ($\sigma = 0.3$).

[2]The code to reproduce all the experiments in this paper can be found at https://github.com/UCLA-StarAI/Semantic-Loss/.

Table 2.1: MNIST. Previously reported test accuracies followed by baselines and semantic loss results (± stddev)

Accuracy % with # of used labels	100	1000	ALL
AtlasRBF [Pitelis et al., 2014]	91.9 (±0.95)	96.32 (±0.12)	98.69
Deep Generative [Kingma et al., 2014]	96.67(±0.14)	97.60 (±0.02)	99.04
Virtual Adversarial [Miyato et al., 2016]	97.67	98.64	99.36
Ladder Net [Rasmus et al., 2015]	**98.94** (±0.37)	**99.16** (±0.08)	99.43 (±0.02)
Baseline: MLP, Gaussian Noise	78.46 (±1.94)	94.26 (±0.31)	99.34 (±0.08)
Baseline: Self-Training	72.55 (±4.21)	87.43 (±3.07)	
Baseline: MLP with Entropy Regularizer	96.27 (±0.64)	98.32 (±0.34)	99.37 (±0.12)
MLP with Semantic Loss	98.38 (±0.51)	98.78 (±0.17)	99.36 (±0.02)

Table 2.2: FASHION. Test accuracy comparison between MLP with semantic loss and ladder nets.

Accuracy % with # of used labels	100	500	1000	ALL
Ladder Net [Rasmus et al., 2015]	81.46 (±0.64)	85.18 (±0.27)	86.48 (±0.15)	90.46
Baseline: MLP, Gaussian Noise	69.45 (±2.03)	78.12 (±1.41)	80.94 (±0.84)	89.87
MLP with Semantic Loss	**86.74** (±0.71)	**89.49** (±0.24)	**89.67** (±0.09)	89.81

MNIST The permutation invariant MNIST classification task is commonly used as a test-bed for general semi-supervised learning algorithms. This setting does not use any prior information about the spatial arrangement of the input pixels. Therefore, it excludes many data augmentation techniques that involve geometric distortion of images, as well as convolutional neural networks.

When evaluating on MNIST, we run experiments for 20 epochs, with a batch size of 10. Experiments are repeated 10 times with different random seeds. Table 2.1 compares semantic loss to three baselines and state-of-the-art results from the literature. The first baseline is a purely supervised MLP, which makes no use of unlabeled data. The second is the classic self-training method for semi-supervised learning, which operates as follows. After every 1000 iterations, the unlabeled examples that are predicted by the MLP to have more than 95% probability of belonging to a single class, are assigned a pseudo-label and become labeled data.

Additionally, we constructed a third baseline by replacing the semantic loss term with the entropy regularizor described in Grandvalet and Bengio [2005] as a direct comparison for semantic

loss. With the same amount of parameter tuning, we found that using entropy achieves an accuracy of 96.27% with 100 labeled examples, and 98.32% with 1000 labelled examples, both are slightly worse than the accuracies reached by semantic loss. Furthermore, to our best knowledge, there is no straightforward method to generalize entropy loss to the settings of complex constraints, where semantic loss is clearly defined and can be easily deployed. We will discuss this more in Section 2.5.

Lastly, We attempted to create a fourth baseline by constructing a constraint-sensitive loss term in the style of Hu et al. [2016], using a simple extension of Probabilistic Soft Logic (PSL) [Kimmig et al., 2012]. PSL translates logic into continuous domains by using soft truth values, and defines functions in the real domain corresponding to each Boolean function. This is normally done for Horn clauses, but since they are not sufficiently expressive for our constraints, we apply fuzzy operators to arbitrary sentences instead. We are forced to deal with a key difference between semantic loss and PSL: encodings in fuzzy logic are highly sensitive to the syntax used for the constraint (and therefore violate Proposition 2). We selected two reasonable encodings detailed in Appendix A.5. The first encoding results in a constant value of 1, and thus could not be used for semi-supervised learning. The second encoding empirically deviates from 1 by < 0.01, and since we add Gaussian noise to the pixels, no amount of tuning was able to extract meaningful supervision. Thus, we do not report these results.

When given 100 labeled examples ($N = 100$), MLP with semantic loss gains around 20% improvement over the purely supervised baseline. The improvement is even larger (25%) compared to self-training. Considering *the only change is an additional loss term*, this result is very encouraging. Comparing to the state of the art, ladder nets slightly outperform semantic loss by 0.5% accuracy. This difference may be an artifact of the excessive tuning of architectures, hyper-parameters and learning rates that the MNIST dataset has been subject to. In the coming experiments, we extend our work to more challenging datasets, in order to provide a clearer comparison with ladder nets. Before that, we want to share a few more thoughts on how semantic loss works. A classical softmax layer interprets its output as representing a categorical distribution. Hence, by normalizing its outputs,

Table 2.3: CIFAR. Test accuracy comparison between CNN with Semantic Loss and ladder nets.

Accuracy % with # of used labels	4000	ALL
CNN Baseline in Ladder Net	76.67 ($\pm$0.61)	90.73
Ladder Net [Rasmus et al., 2015]	79.60 ($\pm$0.47)	
Baseline: CNN, Whitening, Cropping	77.13	90.96
CNN with Semantic Loss	**81.79**	90.92

softmax enforces the same mutual exclusion constraint enforced in our semantic loss function. However, there does not exist a natural way to extend softmax loss to unlabeled samples. In contrast, semantic loss does provide a learning signal on unlabeled samples, by forcing the underlying classifier to make an decision and construct a confident hypothesis for all data. However, for the fully supervised case ($N =$ all), semantic loss does not significantly affect accuracy. Because the MLP has enough capacity to almost perfectly fit the training data, where the constraint is always satisfied, semantic loss is almost always zero. This is a direct consequence of Proposition 3.

FASHION The FASHION [Xiao et al., 2017] dataset consists of Zalando's article images, aiming to serve as a more challenging drop-in replacement for MNIST. Arguably, it has not been overused and requires more advanced techniques to achieve good performance. As in the previous experiment, we run our method for 20 epochs, whereas ladder nets need 100 epochs to converge. Again, experiments are repeated 10 times and Table 2.2 reports the classification accuracy and its standard deviation (except for $N =$ all where it is close to 0 and omitted for space).

Experiments show that utilizing semantic loss results in a very large 17% improvement over the baseline when only 100 labels are provided. Moreover, our method compares favorably to ladder nets, except when the setting degrades to be fully supervised. Note that our method already nearly reaches its maximum accuracy with 500 labeled examples, which is only 1% of the training dataset.

CIFAR-10 To show the general applicability of semantic loss, we evaluate it on CIFAR-10. This dataset consisting of 32-by-32 RGB images in 10 classes. A simple MLP would not have enough representation power to capture the huge variance across objects within the same class. To cope

with this spike in difficulty, we switch our underlying model to a 10-layer CNN as described earlier. We use a batch size of 100 samples of which half are unlabeled. Experiments are run for 100 epochs. However, due to our limited computational resources, we report on a single trial. Note that we make slight modifications to the underlying model used in ladder nets to reproduce similar baseline performance. Please refer to Appendix A.2 for further details.

As shown in Table 2.3, our method compares favorably to ladder nets. However, due to the slight difference in performance between the supervised base models, a direct comparison would be methodologically flawed. Instead, we compare the net improvements over baselines. In terms of this measure, our method scores a gain of 4.66% whereas ladder nets gain 2.93%.

2.4.3 Discussion

The experiments so far have demonstrated the competitiveness and general applicability of our proposed method on semi-supervised learning tasks. It surpassed the previous state of the art (ladder nets) on FASHION and CIFAR-10, while being close on MNIST. Considering the simplicity of our method, such results are encouraging. Indeed, a key advantage of semantic loss is that it only requires a simple additional loss term, and thus incurs almost no computational overhead. Conversely, this property makes our method sensitive to the underlying model's performance.

Without the underlying predictive power of a strong supervised learning model, we do not expect to see the same benefits we observe here. Recently, we became aware that Miyato et al. [2016] extended their work to CIFAR-10 and achieved state-of-the-art results [Miyato et al., 2017], surpassing our performance by 5%. In future work, we plan to investigate whether applying semantic loss on their architecture would yield an even stronger performance.

Figure A.1 in the appendix illustrates the effect of semantic loss on FASHION pictures whose correct label was hidden from the learner. Pictures A.1a and A.1b are correctly classified by the supervised base model, and on the first set it is confident about this prediction ($\mathrm{p}_i > 0.8$). Semantic loss rarely diverts the model from these initially correct labels. However, it bootstraps

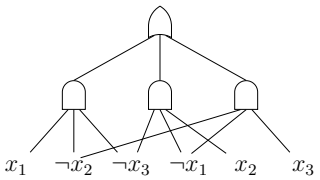


Figure 2.3: A compiled decomposable and deterministic circuit for the exactly-one constraint with 3 variables.

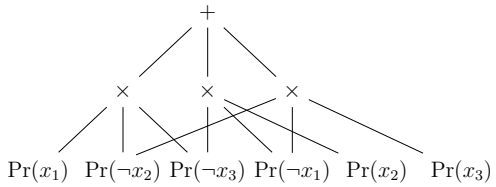


Figure 2.4: The corresponding arithmetic circuit for the exactly-one constraint with 3 variables.

these unlabeled examples to achieve higher confidence in the learned concepts. With this additional learning signal, the model changes its beliefs about Pictures A.1c, which it was previously uncertain about. Finally, even on confidently misclassified Pictures A.1d, semantic loss is able to remedy the mistakes of the base model.

2.5 Learning with Complex Constraints

While much of current machine learning research is focused on problems such as multi-class classification, there remain a multitude of difficult problems involving highly constrained output domains. As mentioned in the previous section, semantic loss has little effect on the fully-supervised exactly-one classification problem. This leads us to seek out more difficult problems to illustrate that semantic loss can also be highly informative in the supervised case, provided the output domain is a sufficiently complex space. Because semantic loss is defined by a Boolean formula, it can be used on any output domain that can be fully described in this manner. Here, we develop a framework for making semantic loss tractable on highly complex constraints, and evaluate it on some difficult examples.

2.5.1 Tractability of Semantic Loss

Our goal here is to develop a general method for computing both semantic loss and its gradient in a tractable manner. Examining Definition 1 of semantic loss, we see that the right-hand side is a well-known automated reasoning task called weighted model counting (WMC) [Chavira and Darwiche, 2008, Sang et al., 2005].

Furthermore, we know of circuit languages that compute WMCs, and that are amenable to backpropagation [Darwiche, 2003]. We use the circuit compilation techniques in Darwiche [2011b] to build a Boolean circuit representing semantic loss. We refer to the literature for details of this compilation approach. Due to certain properties of this circuit form, we can use it to compute both the values and the gradients of semantic loss in time linear in the size of the circuit [Darwiche and Marquis, 2002]. Once constructed, we can add it to our standard loss function as described in Section 2.4.1.

Figure 2.3 shows an example Boolean circuit for the exactly-one constraint with 3 variables. We begin with the standard logical encoding for the exactly-one constraint $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \wedge \neg x_3) \wedge (\neg x_2 \wedge \neg x_3)$, and then compile it into a circuit that can perform WMC efficiently [Chavira and Darwiche, 2008]. The cost of this step depends on the type of the constraint: for bounded-treewidth constraints it can be done efficiently, and for some constraints exact compilation is theoretically hard. In that case, we have to rely on advanced knowledge compilation algorithms to still perform this step efficiently in practice. Our semantic loss framework can be applied regardless of how the circuit gets compiled. On our example, following the circuit bottom up, the logical function can be read as $(x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_1 \wedge \neg x_2 \wedge x_3)$. Once this Boolean circuit is built, we can convert it to an arithmetic circuit, by simply changing `AND` gates into $*$, and `OR` gates into $+$, as shown in Figure 2.4. Now, by pushing the probabilities up through the arithmetic circuit, evaluating the root gives the probability of the logical formula described by the Boolean circuit – this is precisely the exponentiated semantic loss. Notice that this computation was not possible with the Boolean formula we began with: it is a direct result of our circuit having

two key properties called determinism and decomposability. Finally, we can similarly do another pass down on the circuit to compute partial derivatives [Darwiche and Marquis, 2002].

2.5.2 Experimental Evaluation

Our ambition when evaluating semantic loss' performance on complex constraints is not to achieve state-of-the-art performance on any particular problem, but rather to highlight its effect. To this end, we evaluate our method on problems with a difficult output space, where the model could no longer be fit directly from data, and purposefully use simple MLPs for evaluation. We want to emphasize that the constraints used in this evaluation are intentionally designed to be very difficult; much more so than the simple implications that are usually studied (e.g., Hu et al. [2016]). Hyper-parameter tuning details are again in Appendix A.3.

Grids We begin with a classic algorithmic problem: finding the shortest path in a graph. Specifically, we use a 4-by-4 grid $G = (V, E)$ with uniform edge weights. We randomly remove edges for each example to increase difficulty. Formally, our input is a binary vector of length $|V| + |E|$, with the first $|V|$ variables indicating sources and destinations, and the next $|E|$ which edges are removed. Similarly, each label is a binary vector of length $|E|$ indicating which edges are in the shortest path. Finally, we require through our constraint α that the output form a valid simple path between the desired source and destination. To compile this constraint, we use the method of Nishino et al. [2017] to encode pairwise simple paths, and enforce the correct source and destination. For more details on the constraint and data generation process, see Appendix A.4.

To evaluate, we use a dataset of 1600 examples, with a 60/20/20 train/validation/test split. Table 2.4 compares test accuracy between a 5-layer MLP baseline, and the same model augmented with semantic loss. We report three different accuracies that illustrate the effect of semantic loss: "Coherent" indicates the percentage of examples for which the classifier gets the entire configuration right, while "Incoherent" measures the percentage of individually correct binary labels, which as a whole may not constitute a valid path at all. Finally, "Constraint" describes the percentage of

Table 2.4: Grid shortest path test results: coherent, incoherent and constraint accuracy.

Test accuracy %	Coherent	Incoherent	Constraint
5-layer MLP	5.62	**85.91**	6.99
+ Semantic loss	**28.51**	83.14	**69.89**

Table 2.5: Preference prediction test results: coherent, incoherent and constraint accuracy.

Test accuracy %	Coherent	Incoherent	Constraint
3-layer MLP	1.01	**75.78**	2.72
+ Semantic loss	**13.59**	72.43	**55.28**

predictions given by the model that satisfy the constraint associated with the problem. In the case of incoherent accuracy, semantic loss has little effect, and in fact slightly reduces the accuracy as it combats the standard sigmoid cross entropy. In regard to coherent accuracy however, semantic loss has a very large effect in guiding the network to jointly learn true paths, rather than optimizing each binary output individually. We further see this by observing the large increase in the percentage of predictions that really are paths between the desired nodes in the graph.

Preference Learning The next problem is that of predicting a complete order of preferences. That is, for a given set of user features, we want to predict how the user ranks their preference over a fixed set of items. We encode a preference ordering over n items as a flattened binary matrix $\{X_{ij}\}$, where for each $i, j \in \{1, \ldots, n\}$, X_{ij} denotes that item i is at position j [Choi et al., 2015]. Clearly, not all configurations of outputs correspond to a valid ordering, so our constraint allows only for those that are.

We use preference ranking data over 10 types of sushi for 5000 individuals, taken from PREFLIB [Mattei and Walsh, 2013]. We take the ordering over 6 types of sushi as input features to predict the ordering over the remaining 4 types, with splits identical to those in Shen et al. [2017]. We again split the data 60/20/20 into train/test/split, and employ a 3-layer MLP as our baseline. Table 2.5 compares the baseline to the same MLP augmented with semantic loss for valid total orderings. Again, we see that semantic loss has a marginal effect on incoherent accuracy, but significantly

improves the network's ability to predict valid, correct orderings. Remarkably, without semantic loss, the network is only able to output a valid ordering on 1% of examples.

2.6 Related Work

Incorporating symbolic background knowledge into machine learning is a long-standing challenge [Srinivasan et al., 1995]. It has received considerable attention for structured prediction in natural language processing, in both supervised and semi-supervised settings. For example, *constrained conditional models* extend linear models with constraints that are enforced through integer linear programming [Chang et al., 2008, 2013]. Constraints have also been studied in the context of probabilistic graphical models [Mateescu and Dechter, 2008, Ganchev et al., 2010]. Kisa et al. [2014] utilize a circuit language called the *probabilistic sentential decision diagram* to induce distributions over arbitrary logical formulas. They learn generative models that satisfy preference and path constraints [Choi et al., 2015, 2016], which we study in a discriminative setting.

Various deep learning techniques have been proposed to enforce either arithmetic constraints [Pathak et al., 2015, Márquez-Neila et al., 2017] or logical constraints [Rocktäschel et al., 2015, Hu et al., 2016, Demeester et al., 2016, Stewart and Ermon, 2017, Minervini et al., 2017, Diligenti et al., 2017, Donadello et al., 2017] on the output of a neural network. The common approach is to reduce logical constraints into differentiable arithmetic objectives by replacing logical operators with their fuzzy t-norms and logical implications with simple inequalities. A downside of this fuzzy relaxation is that the logical sentences lose their precise meaning. The learning objective becomes a function of the syntax rather than the semantics (see Section 2.4). Moreover, these relaxations are often only applied to Horn clauses. One alternative is to encode the logic into a factor graph and perform loopy belief propagation to compute a loss function [Naradowsky and Riedel, 2017], which is known to have issues in the presence of complex logical constraints [Smith and Gogate, 2014].

Several specialized techniques have been proposed to exploit the rich structure of real-world labels. Deng et al. [2014] propose hierarchy and exclusion graphs that jointly model hierarchical

categories. It is a method invented to address examples whose labels are not provided at the most specific level. Finally, the objective of semantic loss to increase the confidence of predictions on unlabeled data is related to information-theoretic approaches to semi-supervised learning [Grandvalet and Bengio, 2005, Erkan and Altun, 2010], and approaches that increase robustness to output perturbation [Miyato et al., 2016]. A key difference between semantic loss and these information-theoretic losses is that semantic loss generalizes to arbitrary logical output constraints that are much more complex.

2.7 Conclusions & Future Work

Both reasoning and semi-supervised learning are often identified as key challenges for deep learning going forward. In this paper, we developed a principled way of combining automated reasoning for propositional logic with existing deep learning architectures. Moreover, we showed that semantic loss provides significant benefits during semi-supervised classification, as well as deep structured prediction for highly complex output spaces.

An interesting direction for future work is to come up with effective approximations of semantic loss, for settings where even the methods we have described are not sufficient. There are several potential ways to proceed with this, including hierarchical abstractions, relaxations of the constraints, or projections on random subsets of variables.

CHAPTER 3

Approximate Knowledge Compilation by Online Collapsed Importance Sampling

3.1 Introduction

Modern probabilistic inference algorithms for discrete graphical models are designed to exploit key properties of the distribution. In addition to classical conditional independence, they exploit local structure in the individual factors, determinism coming from logical constraints [Darwiche, 2009], and the context-specific independencies that arise in such distributions [Boutilier et al., 1996]. The *knowledge compilation* approach in particular forms the basis for state-of-the-art probabilistic inference algorithms in a wide range of models, including Bayesian networks [Chavira and Darwiche, 2008], factor graphs [Choi et al., 2013], statistical relational models [Chavira et al., 2006, Van den Broeck, 2013], probabilistic programs [Fierens et al., 2015], probabilistic databases [Van den Broeck and Suciu, 2017], and dynamic Bayesian networks [Vlasselaer et al., 2016]. Based on logical reasoning techniques, knowledge compilation algorithms construct an *arithmetic circuit* representation of the distribution on which inference is guaranteed to be efficient [Darwiche, 2003]. The inference algorithms listed above have one common limitation: they perform exact inference by compiling a worst-case exponentially-sized arithmetic circuit representation. Our goal in this paper is to upgrade these techniques to allow for *approximate probabilistic inference*, while still naturally exploiting the structure in the distribution. We aim to open up a new direction towards scaling up knowledge compilation to larger distributions.

When knowledge compilation produces circuits that are too large, a natural solution is to sample

some random variables and do exact compilation on the smaller distribution over the remaining variables. This collapsed sampling approach suffers from two problems. First, collapsed sampling assumes that one can determine a priori which variables need to be sampled to make the distribution amenable to exact inference. When dealing with large amounts of context-specific independence, it is difficult to find such a set, because the independencies are a function of the particular values that variables get instantiated to. Second, collapsed sampling assumes that one has access to a proposal distribution that determines how to sample each variable, and the success of inference largely depends on the quality of this proposal. In practice, the user often needs to specify the proposal distribution manually, and it is difficult to automatically construct one that is general purpose.

As our first contribution, Section 3.2 introduces *online collapsed importance sampling*, where the sampler chooses which variable to sample next based on the values sampled for previous variables. This algorithm is a solution to the first problem identified above: based on the context of each individual sample, it allows the sampler to determine which subset of the variables is amenable to exact inference. We show that the sampler corresponds to a classical collapsed importance sampler on an augmented graphical model and prove conditions for it to be asymptotically unbiased.

Section 3.3 describes our second contribution: a *collapsed compilation* algorithm that maintains a partially-compiled arithmetic circuit during online collapsed importance sampling. This circuit provides a solution to the second problem identified above: it serves as a highly-effective proposal distribution at each step of the algorithm. Moreover, by setting a limit on the circuit size as we compile more factors into the model, we are able to sample exactly as many variables as needed to fit the arithmetic circuit into memory. This allows us to maximize the amount of exact inference performed by the algorithm. Crucially, through online collapsing, the set of collapsed variables changes with every sample, exploiting different independencies in each sample's arithmetic circuit. We provide an open-source Scala implementation of this collapsed compilation algorithm.[1]

[1]The code is available at https://github.com/UCLA-StarAI/Collapsed-Compilation. It uses the SDD library for knowledge compilation [Darwiche, 2011a] and the Scala interface by Bekker et al. [2015].

Finally, we experimentally validate the performance of collapsed compilation on standard benchmarks. We begin by empirically examining properties of collapsed compilation, to show the value of the proposal distribution and pick apart where performance improvements are coming from. Then, in a setting where the amount of exact inference is fixed, we find that collapsed compilation is competitive with state-of-the-art approximate inference algorithms, outperforming them on several benchmarks.

3.2 Online Collapsed Importance Sampling

We begin with a brief review of collapsed importance sampling, before motivating the need for dynamically selecting which variables to sample. We then demonstrate that we can select variables in an online fashion while maintaining the desired unbiasedness property of the sampler, using an algorithm we call online collapsed importance sampling.

We denote random variables with uppercase letters (X), and their instantiation with lowercase letters (x). Bold letters denote sets of variables ($\mathbf{X}$) and their instantiations ($\mathbf{x}$). We refer to Koller and Friedman [2009] for notation and formulae related to (collapsed) importance sampling.

3.2.1 Collapsed Importance Sampling

The basic principle behind collapsed sampling is that we can reduce the variance of an estimator by making part of the inference exact. That is, suppose we partition our variables into two sets: $\mathbf{X_p}$, and $\mathbf{X_d}$. In collapsed importance sampling, the distribution of variables in $\mathbf{X_p}$ will be estimated via importance sampling, while those in $\mathbf{X_d}$ will be estimated by computing exactly $P(\mathbf{X_d}|\mathbf{x_p})$ for each sample $\mathbf{x_p}$. In particular, suppose we have some function $f(\mathbf{x})$ where $\mathbf{x}$ is a complete instantiation of $\mathbf{X_p} \cup \mathbf{X_d}$, and a proposal distribution Q over $\mathbf{X_p}$. Then we estimate the expectation of f by

$$\hat{\mathbb{E}}(f) = \frac{\sum_{m=1}^{M} w[m](\mathbb{E}_{P(\mathbf{X_d}|\mathbf{x_p}[m])}[f(\mathbf{x_p}[m], \mathbf{X_d})])}{\sum_{m=1}^{M} w[m]} \tag{3.1}$$

on samples $\{\mathbf{x_p}[m]\}_{m=1}^{M}$ drawn from a proposal distribution Q. For each sample, we analytically compute the importance weights $w[m] = \frac{\hat{P}(\mathbf{x_p}[m])}{Q(\mathbf{x_p}[m])}$, and the exact expectation of f conditioned on the sample, that is, $\mathbb{E}_{P(\mathbf{X_d}|\mathbf{x_p}[m])}[f(\mathbf{x_p}[m], \mathbf{X_d})]$. Due to the properties of importance samplers, the estimator given by (3.1) is asymptotically unbiased. Moreover, if we compute $P(\mathbf{x_p}[m])$ exactly rather than the unnormalized $\hat{P}(\mathbf{x_p}[m])$, then the estimator is unbiased [Tokdar and Kass, 2010].

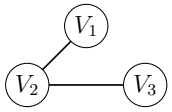
3.2.2 Motivation

A critical decision that needs to be made when doing collapsed sampling is selecting a partition – which variables go in $\mathbf{X_p}$ and which go in $\mathbf{X_d}$. The choice of partition can have a large effect on the quality of the resulting estimator, and the process of choosing such a partition requires expert knowledge. Furthermore, selecting a partition a priori that works well is not always possible, as we will show in the following example. All of this raises the question whether it is possible to choose the partition on the fly for each sample, which we will discuss in Section 3.2.3.

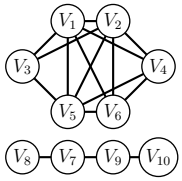
Suppose we have a group of n people, denoted $1, ..., n$. For every pair of people $(i, j), i < j$, there is a binary variable F_{ij} indicating whether i and j are friends. Additionally, we have features V_i for each person i, and $F_{ij} = 1$ (that is i, j are friends) implies that V_i and V_j are correlated. Suppose we are performing collapsed sampling on the joint distribution over $\mathbf{F}$ and $\mathbf{V}$, and that we have already decided to place all friendship indicators F_{ij} in $\mathbf{X_p}$ to be sampled. Next, we need to decide which variables in $\mathbf{V}$ to include in $\mathbf{X_p}$ for the remaining inference problem over $\mathbf{X_d}$ to become tractable. Observe that given a sampled $\mathbf{F}$, due to the independence properties of $\mathbf{V}$ relying on $\mathbf{F}$, a graphical model G is induced over $\mathbf{V}$ (see Figures 3.1a,3.1b). Moreover, this graphical model can vary greatly between different samples of $\mathbf{F}$. For example, G_1 in Figure 3.1c densely connects $\{V_1, ..., V_6\}$ making it difficult to perform exact inference. Thus, we will need to sample some variables from this set. However, exact inference over $\{V_7, ..., V_{10}\}$ is easy. Conversely, G_2 in Figure 3.1d depicts the opposite scenario: $\{V_1, ..., V_5\}$ forms a tree, which is easy for inference, whereas $\{V_6, ..., V_{10}\}$ is now intractable. It is clearly impossible to choose a small subset of $\mathbf{V}$ to sample that fits all cases, thus demonstrating a need for an online variable selection during collapsed

F_{12}	1
F_{13}	0
F_{23}	1

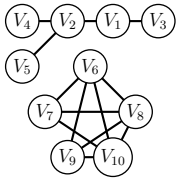
(a) Sampled Friendships



(b) Induced Dependencies



(c) Induced Network G_1



(d) Induced Network G_2

Figure 3.1: Different samples for $\mathbf{F}$ can have a large effect on the resulting dependencies between $\mathbf{V}$.

Algorithm 1 Online Collapsed IS

Input: $\mathbf{X}$: The set of all variables, π: Variable selection policy, $Q_{X_i|\mathbf{x_p}}$: Proposal distributions
Output: A sample $\left(\mathbf{X_d^m}, \mathbf{x_p^m}, w[m]\right)$

$\mathbf{x_p} \leftarrow \{\}$; $\mathbf{X_d} \leftarrow \mathbf{X}$
while π does not stop **do**
  $X_i \sim \pi\left(\mathbf{x_p}\right)$
  $x_i \sim Q_{X_i|\mathbf{x_p}}(X_i|\mathbf{x_p})$
  $\mathbf{x_p} \leftarrow \mathbf{x_p} \cup \{x_i\}$
  $\mathbf{X_d} \leftarrow \mathbf{X_d} \setminus \{X_i\}$
return $\left(\mathbf{X_d}, \mathbf{x_p}, \frac{\hat{P}(\mathbf{x_p})}{Q(\mathbf{x_p})}\right)$

sampling.

3.2.3 Algorithm

We now introduce our online collapsed importance sampling algorithm. It decides at sampling time which variables to sample and which to do exact inference on.

To gain an intuition, suppose we are in the standard collapsed importance sampling setting. Rather than sampling an instantiation $\mathbf{x_p}$ jointly from Q, we can instead first sample $x_{p_1} \sim Q(X_{p_1})$, then $x_{p_2} \sim Q(X_{p_2}|x_{p_1})$, and so on using the chain rule of probability. In *online* collapsed importance sampling, rather than deciding $X_{p_1}, X_{p_2}, X_{p_3}, \ldots$ a priori, we select which variable will be X_{p_2} based on the previous sampled value x_{p_1}, we select which will be X_{p_3} based on x_{p_1} and x_{p_2}, and so on.

Definition 2. *Let* $\mathbf{y}$ *be an instantiation of* $\mathbf{Y} \subset \mathbf{X}$. *A* variable selection policy π *takes* $\mathbf{y}$ *and either stops sampling or returns a distribution over which variable in* $\mathbf{X} \setminus \mathbf{Y}$ *should be sampled next.*

For example, a naive policy could be to select a remaining variable uniformly at random. Once the policy π stops sampling, we are left with an instantiation $\mathbf{x_p}$ and a set of remaining variables $\mathbf{X_d}$, where both are specific to the choices made for that particular sample.

Algorithm 1 shows more precisely how online collapsed importance sampling generates a single sample, given a full set of variables $\mathbf{X}$, a variable selection policy π, and proposal distributions $Q_{X_i|\mathbf{x_p}}$ for any choice of X_i and $\mathbf{x_p}$. This sample consists of a set of variables $\mathbf{X_d^m}$ to do exact inference for, an instantiation of the sampled variables $\mathbf{x_p^m}$, and the corresponding importance weights $w[m]$, all indexed by the sample number m. Note that $\mathbf{x_p}$ is a set of variables together with their instantiations, while $\mathbf{X_d}$ is just a set of variables. The global joint proposal $Q(\mathbf{x_p})$, denoting the probability that Algorithm 1 returns $\mathbf{x_p}$, is left abstract for now (see Section 3.2.4.2 for a concrete instance). In general, it is induced by variable selection policy π and the individual local proposals $Q_{X_i|\mathbf{x_p}}$.

Definition 3. *Given M samples $\left\{\mathbf{X_d^m}, \mathbf{x_p^m}, w[m]\right\}_{m=1}^{M}$ produced by online collapsed importance sampling, the* online collapsed importance sampling estimator *of f is*

$$\hat{\mathbb{E}}(f) = \frac{\sum_{m=1}^{M} w[m](\mathbb{E}_{P(\mathbf{X_d^m}|\mathbf{x_p^m})}[f(\mathbf{x_p^m}, \mathbf{X_d^m})])}{\sum_{m=1}^{M} w[m]}. \tag{3.2}$$

Note that the only difference compared to Equation 3.1 is that sets $\mathbf{X_p^m}$ and $\mathbf{X_d^m}$ vary with each sample.

3.2.4 Analysis

Our algorithm for online collapsed importance sampling raises two questions: does Equation 3.2 yield unbiased estimates, and how does one compute the proposal $Q(\mathbf{x_p})$? We study both questions next.

3.2.4.1 Unbiasedness of Estimator

If we let π be a policy that always returns the same variables in the same order, then we recover classical offline collapsed importance sampling - and thus retain all of its properties. In order to make a similar statement for any arbitrary policy π, we will use the augmented factor graph construction presented in Figure 3.2. Our goal is to reduce online collapsed importance sampling on F to a problem of doing offline collapsed importance sampling on F_A.

(a) Original factor graph F

(b) Augmented factor graph F_A

Figure 3.2: Online collapsed sampling corresponds to collapsed sampling on an augmented graph

Intuitively, we add variable X_i^* to the factor graph, representing a copy variable of X_i. We design our offline collapsed sampler on augmented graph F_A such that we are always sampling X_i^* and computing X_i exactly. To make this possible without actually inferring the entire distribution exactly, we add variable S_i to the model (also always to be sampled). Each S_i acts as an indicator for whether X_i^* and X_i are constrained to be equal. S_i can also be thought of as indicating whether or not we are sampling X_i in our original factor graph F when doing online collapsed importance sampling. These dependencies are captured in the new factor f_{au}. We are now ready to state the following results.

Theorem 5. *For any factor graph F and its augmented graph F_A, we have $\forall \mathbf{x}$, $P_F(\mathbf{x}) = P_{F_A}(\mathbf{x})$.*

Theorem 6. *Let F be a factor graph and let F_A be its augmented factor graph. The collapsed importance sampling estimator (Eq. 3.1) with $\mathbf{X_p} = \mathbf{X}^* \cup \mathbf{S}$ and $\mathbf{X_d} = \mathbf{X}$ on F_A is equivalent to the online collapsed importance sampling estimator (Eq. 3.2) on F.*

Corollary 1. *The estimator given by Eq. 3.2 is asymptotically unbiased.*

Proofs and the details of this construction can be found in Appendix B.1.

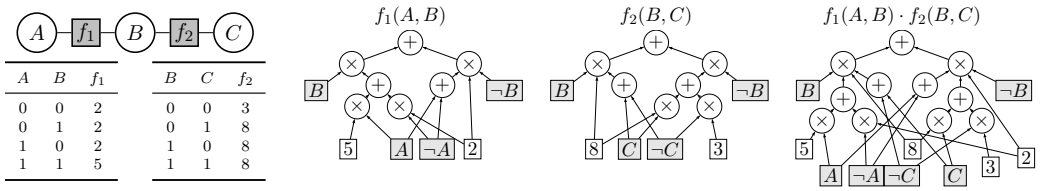


Figure 3.3: Multiplying Arithmetic Circuits: Factor graph and ACs for individual factors which multiply into a single AC for the joint distribution. Given an AC, inference is tractable by propagating inputs.

3.2.4.2 Computing the Proposal Distribution

Our next question is how to compute the global joint proposal distribution $Q(\mathbf{x_p})$, given that we have variable selection policy π and each local proposal distribution $Q_{X_i|\mathbf{x}_p}$. Notice that since these $Q_{X_i|\mathbf{x}_p}$ are unconstrained and unrelated distributions, the computation is not easy in general. In particular, considering $|\mathbf{X_p}| = n$ and our previous example of a uniformly random policy π, then for any given instantiation $\mathbf{x_p}$, there are $n!$ different ways $\mathbf{x_p}$ could be sampled by Algorithm 1 – one for each ordering that arrives at $\mathbf{x_p}$. In this case, computing $Q(\mathbf{x_p})$ requires summing over exponentially many terms, which is undesirable. Instead, we restrict the variable selection policies we use to the following class.

Definition 4. *A* deterministic *variable selection policy $\pi(\mathbf{x_p})$ is a function with a range of $\mathbf{X} \setminus \mathbf{X_p}$.*

Theorem 7. *For any sample $\mathbf{x_p}$ and deterministic variable selection policy $\pi(\mathbf{x_p})$, there is exactly one order $X_{p_1}, X_{p_2}, \ldots, X_{p_{|\mathbf{X_p}|}}$ in which the variables $\mathbf{X_p}$ could have been sampled. Therefore, the joint proposal distribution is given by $Q(\mathbf{x_p}) = \prod_{i=1}^{|\mathbf{X_p}|} Q_{X_{p_i}|\mathbf{x}_{p_{1:i-1}}}(x_{p_i}|\mathbf{x}_{p_{1:i-1}})$.*

Hence, computing the joint proposal $Q(\mathbf{x_p})$ becomes easy given a deterministic selection policy π.

3.3 Collapsed Compilation

Online collapsed importance sampling presents us with a powerful technique for adapting to problems traditional collapsed importance sampling may struggle with. However, it also demands

we solve several difficult tasks: one needs a good proposal distribution over any subset of variables, an efficient way of exactly computing an expectation given a sample, and an efficient way of finding the true probability of sampled variables. In this section, we introduce collapsed compilation, which tackles all three of these problems at once using techniques from knowledge compilation.

3.3.1 Knowledge Compilation Background

We begin with a short review of how to perform exact inference on a probabilistic graphical model using knowledge compilation to arithmetic circuits (ACs).

Suppose we have a factor graph [Koller and Friedman, 2009] consisting of three binary variables A, B and C, and factors f_1, f_2 as depicted in Figure 3.3. Each of these factors, as well as their product can be represented as an arithmetic circuit. These circuits have inputs corresponding to variable assignments (e.g., A and $\neg A$) or constants (e.g., 5). Internal nodes are sums or products. We can encode a complete instantiation of the random variables by setting the corresponding variable assignments to 1 and the opposing assignments to 0. Then, the root of the circuit for a factor evaluates to the value of the factor for that instantiation. However, ACs can also represent products of factors. In that case, the AC's root evaluates to a weight that is the product of factor values. Under factor graph semantics, this weight represents the unnormalized probability of a possible world.

The use of ACs for probabilistic inference stems from two important properties. Product nodes are *decomposable*, meaning that their inputs are disjoint, having no variable inputs in common. Sum nodes are *deterministic*, meaning that for any given complete input assignment to the circuit, at most one of the sum's inputs evaluates to a non-zero value. Because of decomposability, we are able to perform marginal inference on ACs: by setting both assignments for the same variable to 1, we effectively marginalize out that variable. For example, by setting all inputs to 1, the arithmetic circuit evaluates to the sum of weights of all worlds, which is the partition function of the graphical model. We refer to Darwiche [2009] for further details on how to reason with arithmetic circuits.

In practice, arithmetic circuits are often compiled from graphical models by encoding graphical

model inference into a logical task called weighted model counting, followed by using Boolean circuit compilation techniques on the weighted model counting problem. We refer to Choi et al. [2013] and Chavira and Darwiche [2008] for details. As our Boolean circuit compilation target, we will use the sentential decision diagram (SDD) [Darwiche, 2011a]. Given any two SDDs representing factors f_1, f_2, we can efficiently compute the SDD representing the factor multiplication of f_1 and f_2, as well as the result of conditioning the factor graph on any instantiation x. We call such operations APPLY, and they are the key to using knowledge compilation for doing online collapsed importance sampling. An example of multiplying two arithmetic circuits is depicted in Figure 3.3.

As a result of SDDs supporting the APPLY operations, we can directly compile graphical models to circuits in a bottom-up manner. Concretely, we start out by compiling each factor into a corresponding SDD representation using the encoding of Choi et al. [2013]. Next, these SDDs are multiplied in order to obtain a representation for the entire model. As shown by Choi et al. [2013], this straightforward approach can be used to achieve state-of-the-art exact inference on probabilistic graphical models.

3.3.2 Algorithm

Now that we have proposed online collapsed importance sampling and given background on knowledge compilation, we are ready to introduce collapsed compilation, an algorithm that uses knowledge compilation to do online collapsed importance sampling.

Collapsed compilation begins by multiplying factors represented as SDDs. When the resulting SDD becomes too large, we invoke online collapsed importance sampling to instantiate one of the variables. On the arithmetic circuit representation, sampling a variable replaces one input by 1 and the other by 0. This conditioning operation allows us to simplify the SDD until it is sufficiently small again. At the end, the sampled variables form $\mathbf{x_p}$, and the variables remaining in the SDD form $\mathbf{X_d}$.

Concretely, collapsed compilation repeatedly performs a few simple steps, following Algorithm 1:

1. Choose an order, and begin multiplying compiled factors into the current SDD until the size limit is reached.
2. Select a variable X using the given policy π.
3. Sample X according to its marginal probability in the current SDD, corresponding to the partially compiled factor graph conditioned on prior instantiations.
4. Condition the SDD on the sampled value for X.

We are taking advantage of knowledge compilation in a few subtle ways. First, to obtain the importance weights, we compute the partition function on the final resulting circuit, which corresponds to the unnormalized probability of all sampled variables, that is, $\hat{P}(\mathbf{x_p})$ in Algorithm 1. Second, Step 3 presents a non-trivial and effective proposal distribution, which due to the properties of SDDs is efficient to compute in the size of the circuit. Third, all APPLY operations on SDDs can be performed tractably [Van den Broeck and Darwiche, 2015], which allows us to multiply factors and condition SDDs on sampled instantiations.

The full technical description and implementation details can be found in Appendix B.2 and B.3.

3.4 Experimental Evaluation

Data & Evaluation Criteria To empirically investigate collapsed compilation, we evaluate the performance of estimating a single marginal on a series of commonly used graphical models. Each model is followed in parentheses by its number of random variable nodes and factors.

From the 2014 UAI inference competition, we evaluate on linkage(1077,1077), Grids(100,300), DBN(40, 440), and Segmentation(228,845) problem instances. From the 2008 UAI inference competition, we use two semi-deterministic grid instances, 50-20(400, 400) and 75-26(676, 676).

Here the first number indicates the percentage of factor entries that are deterministic, and the second indicates the size of the grid. Finally, we generated a randomized frustrated Ising model on a 16x16 grid, frust16(256, 480). Beyond these seven benchmarks, we experimented on ten additional standard benchmarks. Because those were either too easy (showing no difference between collapsed compilation and the baselines), or similar to other benchmarks, we do not report on them here.

For evaluation, we run all sampling-based methods 5 times for 1 hour each. We report the median Hellinger distance across all runs, which for discrete distributions P and Q is given by

$$H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{i=1}^{k}(\sqrt{p_i} - \sqrt{q_i})^2}.$$

Compilation Order Once we have compiled an SDD for each factor in the graphical model, collapsed compilation requires us to choose in which order to multiply these SDDs. We look at two orders: `BFS` and `revBFS`. The first begins from the marginal query variable, and compiles outwards in a breadth-first order. The second does the same, but in exactly the opposite order arriving at the query variable last.

Variable Selection Policies We evaluate three variable selection policies:

The first policy `RBVar` explores the idea of picking the variable that least increases the Rao-Blackwell variance of the query [Darwiche, 2009]. For a given query α, to select our next variable from $\mathbf{X}$, we use $\operatorname{argmin}_{X \in \mathbf{X}} \sum_x P(\alpha|X)^2 P(X)$. This quantity can be computed in time linear in the size of the current SDD.

The next policy we look at is `MinEnt`, which selects the variable with the smallest entropy. Intuitively, this is selecting the variable for which sampling assumes the least amount of unknown information.

Finally, we examine a graph-based policy `FD` (FrontierDistance). At any given point in our compilation we have some frontier $\mathcal{F}$, which is the set of variables that have some but not all factors in-

cluded in the current SDD. Then we select the variable in our current SDD that is, on the graph of our model, closest to the "center" induced by $\mathcal{F}$. That is, we use $\text{argmin}_{X \in \mathbf{X}} \max_{F \in \mathcal{F}} \text{ distance}(X, F)$.

In our experiments, policy `RBVar` is used with the compilation order `BFS`, while policies `MinEnt` and `FrontierDist` are used with order `RevBFS`.

3.4.1 Understanding Collapsed Compilation

We begin our evaluation with experiments designed to shed some light on different components involved in collapsed compilation. First, we evaluate our choice in proposal distribution by comparison to marginal-based proposals. Then, we examine the effects of setting different size thresholds for compilation on the overall performance, as well as the sample count and quality.

Evaluating the Proposal Distribution Selecting an effective proposal distribution is key to successfully using importance sampling estimation [Tokdar and Kass, 2010]. As discussed in Section 3.3, one requirement of online collapsed importance sampling is that we must provide a proposal distribution over any subset of variables, which in general is challenging.

To evaluate the quality of collapsed compilation's proposal distribution, we compare it to using marginal-based proposals, and highlight the problem with such proposals. First, we compare to a dummy uniform proposal. Second, we compare to a proposal that uses the true marginals for each variable. Experiments on the 50-20 benchmark are shown in Table 3.1a. Note that these experiments were run for 3 hours rather than 1 hour, so the numbers can not be compared exactly to other tables.

Particularly with policies `FrontierDist` and `MinEnt`, the results underline the effectiveness of collapsed compilation's proposal distribution over baselines. This is the effect of conditioning – even sampling from the true posterior marginals does not work very well, due to the missed correlation between variables. Since we are already conditioning for our partial exact inference, collapsed compilation's proposal distribution is providing this improvement for very little added cost.

Table 3.1: Internal comparisons for collapsed compilation. Values represent Hellinger distances.

(a) Comparison of proposal distributions

Policy	Dummy	True	SDD
FD	2.37e−4	1.77e−4	3.72e−7
MinEnt	3.29e−4	1.31e−3	2.10e−8
RBVar	5.81e−3	5.71e−3	7.34e−3

(b) Comparison of size thresholds

Policy	10k	100k	1m
FD	7.33e−5	9.77e−6	7.53e−6
MinEnt	1.44e−3	1.50e−5	8.07e−4
RBVar	2.96e−2	2.66e−2	8.81e−3

(c) Comparison of size thresholds (50 samples)

Policy	10k	100k	1m
FD	1.63e−3	5.08e−7	1.27e−6
MinEnt	1.69e−2	1.84e−6	7.24e−6
RBVar	1.94e−2	1.52e−1	3.07e−2

(d) Number of samples taken in 1 hour by size

Size Threshold	10k	100k	1m
Number of Samples	561.3	33.5	4.7

Choosing a Size Threshold A second requirement for collapsed compilation is to set a size threshold for the circuit being maintained. Setting the threshold to be infinity leaves us with exact inference which is in general intractable, while setting the threshold to zero leaves us with importance sampling using what is likely a poor proposal distribution (since we can only consider one factor at a time). Clearly, the optimal choice finds a trade-off between these two considerations.

Using benchmark 50-20 again, we compare the performance on three different settings for the circuit size threshold: 10,000, 100,000, and 1,000,000. Table 3.1b shows that generally, 100k gives the best performance, but the results are often similar. To further investigate this, Table 3.1c and Table 3.1d show performance with exactly 50 samples for each size, and number of samples per hour respectively. This is more informative as to why 100k gave the best performance - there is a massive difference in performance for a fixed number of samples between 10k and 100k or 1m. The gap between 100k and 1m is quite small, so as a result the increased number of samples for 100k leads to better performance. Intuitively, this is due to the nature of exact circuit compilation, where at a certain size point of compilation you enter an exponential regime. Ideally, we would like to stop compiling right before we reach that point. Thus, we proceed with 100k as our size-threshold setting for further experiments.

3.4.2 Memory-Constrained Comparison

In this section, we compare collapsed compilation to two related state-of-the-art methods: edge-deletion belief propagation (EDBP) [Choi and Darwiche, 2006], and IJGP-Samplesearch (SS) [Gogate and Dechter, 2011]. Generally, for example in past UAI probabilistic inference competitions, comparing methods in this space involves a fixed amount of time and memory being given to each tool. The results are then directly compared to determine the empirically best performing algorithm. While this is certainly a useful metric, it is highly dependent on efficiency of implementation, and moreover does not provide as good of an understanding of the effects of being allowed to do more or less exact inference. To give more informative results, in addition to a time limit, we restrict our comparison at the algorithmic level, by controlling for the level of exact inference being performed.

Edge-Deletion Belief Propagation EDBP performs approximate inference by increasingly running more exact junction tree inference, and approximating the rest via belief propagation [Choi and Darwiche, 2006, Choi et al., 2005]. To constrain EDBP, we limit the corresponding circuit size for the junction tree used. In our experiments we set these limits at 100,000 and 1,000,000.

IJGP-Samplesearch IJGP-Samplesearch (SS) is an importance sampler augmented with constraint satisfaction search [Gogate and Dechter, 2011, 2007]. It uses iterative join-graph propagation [Dechter et al., 2002] together with w-cutset sampling [Bidyuk and Dechter, 2007] to form a proposal, and then uses search to ensure that no samples are rejected. To constrain SS, we limit treewidth w at either 15, 12, or 10. For reference, a circuit of size 100,000 corresponds to a treewidth between 10 and 12.

Appendix B.4 describes both baselines as well as the experimental setup in further detail.

Table 3.2: Hellinger distances across methods with internal treewidth and size bounds

Method	50-20	75-26	DBN	Grids	Segment	linkage	frust
EDBP-100k	2.19e−3	3.17e−5	6.39e−1	1.24e−3	1.63e−6	6.54e−8	4.73e−3
EDBP-1m	7.40e−7	2.21e−4	6.39e−1	1.98e−7	1.93e−7	5.98e−8	4.73e−3
SS-10	2.51e−2	2.22e−3	6.37e−1	3.10e−1	3.11e−7	4.93e−2	1.05e−2
SS-12	6.96e−3	1.02e−3	6.27e−1	2.48e−1	3.11e−7	1.10e−3	5.27e−4
SS-15	9.09e−6	1.09e−4	(Exact)	8.74e−4	3.11e−7	4.06e−6	6.23e−3
FD	9.77e−6	1.87e−3	1.24e−1	1.98e−4	6.00e−8	5.99e−6	5.96e−6
MinEnt	1.50e−5	3.29e−2	1.83e−2	3.61e−3	3.40e−7	6.16e−5	3.10e−2
RBVar	2.66e−2	4.39e−1	6.27e−3	1.20e−1	3.01e−7	2.02e−2	2.30e−3

3.4.2.1 Discussion

Table 3.2 shows the experimental results for this setting. Overall, we have found that when restricting all methods to only do a fixed amount of exact inference, collapsed compilation has similar performance to both Samplesearch and EDBP. Furthermore, given a good choice of variable selection policy, it can often perform better. In particular, we highlight DBN, where we see that collapsed compilation with the `RBVar` or `MinEnt` policies is the only method that manages to achieve reasonable approximate inference. This follows the intuition discussed in Section 3.2.2: a good choice of a few variables in a densely connected model can lead to relatively easy exact inference for a large chunk of the model.

Another factor differentiating collapsed compilation from both EDBP and Samplesearch is the lack of reliance on some type of belief propagation algorithm. Loopy belief propagation is a cornerstone of approximate inference in graphical models, but it is known to have problems converging to a good approximation on certain classes of models [Murphy et al., 1999]. The problem instance frust16 is one such example – it is an Ising model with spins set up such that potentials can form loops, and the performance of both EDBP and Samplesearch highlights these issues.

3.4.3 Probabilistic Program Inference

As an additional point of comparison, we introduce a new type of benchmark. We use the probabilistic logic programming language ProbLog [De Raedt and Kimmig, 2015] to model a graph with probabilistic edges, and then query for the probability of two nodes being connected. This problem presents a unique challenge, as every non-unary factor is deterministic.

Method	Prob12
EDBP-1m	3.18e−1
SS-15	3.87e−3
FD	1.50e−3

Table 3.3: Hellinger distances for ProbLog benchmark

Table 3.3 shows the results for this benchmark, with the underlying graph being a 12x12 grid. We see that EDBP struggles here due to the large number of deterministic factors, which stop belief propagation from converging in the allowed number of iterations. Samplesearch and collapsed compilation show similarly decent results, but interestingly they are not happening for the same reason. To contextualize this discussion, consider the stability of each method. Collapsed compilation draws far fewer samples than SS – some of this is made up for by how powerful collapsing is as a variance reduction technique, but it is indeed less stable than SS. For this particular instance, we found that while different runs for collapsed compilation tended to give different marginals fairly near the true value, SS consistently gave the same incorrect marginal. This suggests that if we ran each algorithm until convergence, collapsed compilation would tend toward the correct solution, while SS would not, and appears to have a bias on this benchmark.

3.5 Related Work and Conclusions

We have presented online collapsed importance sampling, an asymptotically unbiased estimator that allows for doing collapsed importance sampling without choosing which variables to collapse a priori. Using techniques from knowledge compilation, we developed collapsed compilation, an implementation of online collapsed importance sampling that draws its proposal distribution from partial compilations of the distribution, and naturally exploits structure in the distribution.

In related work, Lowd and Domingos [2010] study arithmetic circuits as a variational approxi-

mation of graphical models. Approximate compilation has been used for inference in probabilistic (logic) programs [Vlasselaer et al., 2015]. Other approximate inference algorithms that exploit local structure include samplesearch and the family of universal hashing algorithms [Ermon et al., 2013, Chakraborty et al., 2014]. Finally, collapsed compilation can be viewed as an approximate knowledge compilation method: each drawn sample presents a partial knowledge base along with the corresponding correction weight. This means that it can be used to approximate any query which can be performed efficiently on an SDD – for example the most probable explanation (MPE) query [Chan and Darwiche, 2006, Choi and Darwiche, 2017]. We leave this as an interesting direction for future work.

CHAPTER 4

On Constrained Open-World Probabilistic Databases

4.1 Introduction

An ubiquitous pursuit in the study of knowledge base representation is the search for a model that can represent uncertainty while simultaneously answering interesting queries efficiently. The key underlying challenge is that these goals are at odds with each other. Modelling *uncertainty* requires additional model complexity. At the same time, the ability to answer *meaningful queries* usually demands restrictive model assumptions. Both of these properties are at odds with the key limiting factor of *tractability*: success in the first two goals is not nearly as impactful if it is not achieved efficiently. Unfortunately, probabilistic reasoning is often computationally hard, even on databases [Roth, 1996, Dalvi and Suciu, 2012].

One approach towards achieving this goal is to begin with a simple model such a probabilistic database (PDB) [Suciu et al., 2011, Van den Broeck and Suciu, 2017]. A PDB models uncertainty, but is inherently simple and makes very strong independence assumptions and closed-world assumptions allowing for tractability on a very large class of queries [Dalvi and Suciu, 2007, 2012]. However, PDBs can fall short under non-ideal circumstances, as their semantics are brittle to incomplete knowledge bases [Ceylan et al., 2016a].

To bring PDBs closer to the desired goal, Ceylan et al. [2016a] propose open-world probabilistic databases (OpenPDB), wherein the semantics of a PDB are strengthened to relax the closed-world assumption. While OpenPDBs maintain a large class of tractable queries, their semantics are so relaxed these queries lose their precision: they model further uncertainty, but in exchange give less

useful query answers.

In this work, we aim to overcome these querying challenges, while simultaneously maintaining the degree of uncertainty modeled by OpenPDBs. To achieve this, we propose further strengthening the semantics of OpenPDBs by constraining the mean probability allowed for a relation. These constraints work at the *schematic* level, meaning no additional per-item information is required. They are practically motivated by knowledge of summary statistics, of how many tuples we expect to be true. A theoretical analysis shows that, despite their simplicity, such constraints fundamentally change the difficulty landscape of queries, leading us to propose a general-purpose approximation scheme.

The rest of the paper is organized as follows: Section 4.2 provides necessary background on relational logic and PDBs, as well as an introduction to OpenPDBs. Section 4.3 motivates and introduces our construction for constraining OpenPDBs. Section 4.4 analyses exact solutions subject to these constraints, providing a class of tractable queries along with an algorithm. It also shows that the problem is in general hard, even in some cases where standard PDB queries are tractable. Section 4.5 investigates an efficient and provably bounded approximation scheme. Section 4.6 discusses our findings, and summarizes interesting directions that we leave as open problems.

4.2 Background

This section provides background and motivation for probabilistic databases and their open-world counterparts. Notation and definitions are adapted from Ceylan et al. [2016a].

4.2.1 Relational Logic and Databases

We now describe 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*

Scientist
Einstein
Erdős
von Neumann

CoAuthor	
Einstein	Erdős
Erdős	von Neumann

Figure 4.1: Example relational database. Notice that the first row of the right table corresponds to the atom CoAuthor(Einstein, Erdős).

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 occurences of x by t in a formula Q.

A relational *vocabulary* σ is comprised of a set of predicates $\mathcal{R}$ and a domain $\mathcal{D}$. Using the *Herbrand semantics* [Hinrichs and Genesereth, 2006], 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 *models* a formula Q whenever ω satisfies Q. This is denoted by $\omega \models \mathsf{Q}$.

A reasonable starting point for the target knowledge base to construct would be to use a traditional *relational database*. Using the standard model-theoretic view [Abiteboul et al., 1995], a relational database for a vocabulary σ is a σ-interpretation ω. Less formally, a relational database consists of a series of relations, each of which corresponds to a predicate. Each relation consists of a series of rows, also called *tuples*, each of which corresponds to an atom of the predicate being true. Any atom not appearing as a row in the relation is considered to be *false*, following the closed-world assumption [Reiter, 1981]. Figure 4.1 shows an example database.

4.2.2 Probabilistic Databases

Despite the success of relational databases, their deterministic nature leads to a few shortcomings. A common way to gather a large knowledge base is to apply some sort of statistical model Carlson et al. [2010], Suchanek et al. [2007], Peters et al. [2014], Dong et al. [2014] which returns a probability value for potential tuples. Adapting the output of such a model to a relational database involves thresholding on the probability value, discarding valuable information along the way. A

Scientist	p
Einstein	0.8
Erdős	0.8
von Neumann	0.9
Shakespeare	0.2

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

Figure 4.2: Example probabilistic database. Tuples are now of the form $\langle t : p \rangle$ where p is the probability of the tuple t being present.

probabilistic database (PDB) circumvents this problem by assigning each tuple a probability.

Definition 5. *A* (tuple-independent) probabilistic database $\mathcal{P}$ *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 6. *A probabilistic database* $\mathcal{P}$ *for vocabulary* σ *induces a probability distribution over* σ*-interpretations* ω*:*

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

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

Notice this last statement is again making the closed-world assumption: any tuple that we have no information about is assigned probability zero. Figure 4.2 shows an example PDB.

4.2.2.1 Probabilistic Queries

In relational databases, the fundamental task we are interested in solving is how to answer queries. The same is true for probabilistic databases, with the only difference being that we are now interested

in probabilities over queries. In particular, we are interested in queries that are fully quantified - also known as *Boolean queries*. On a relational database, this corresponds to a query that has an answer of True or False.

For example, on the database given in Figure 4.1, we might ask if there is a scientist who is a coauthor:

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

If we instead asked this query of the probabilistic database in Figure 4.2, we would be computing the probability by summing over the worlds in which the query is true:

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

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). UCQs have been shown to live in a dichotomy of efficient evaluation [Dalvi and Suciu, 2012]: computing the probability of a UCQ is either polynomial in the size of the database, or it is $\#P$-hard. This property can be checked through the syntax of a query, and we say that a UCQ is *safe* if it admits efficient evaluation. In the literature of probabilistic databases [Suciu et al., 2011, Dalvi and Suciu, 2012], as well as throughout the rest of this paper, UCQs are the primary query object studied.

4.2.2.2 Efficient Query Evaluation

For probabilistic databases to be useful, we need to be able to efficiently compute the probabilities of queries: we now describe how to do this. Algorithm 2 does this in polynomial time for all queries that can be computed efficiently (known as *safe* queries). We now explain the steps in further detail.

Algorithm 2 $\textbf{Lift}^{\mathbf{R}}(\mathsf{Q}, \mathcal{P})$, abbreviated by $\mathbf{L}(\mathsf{Q})$

Input: UCQ Q , prob. database $\mathcal{P}$ with constants T.
Output: 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)

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 occurences in Q. These preprocessing steps are known as *shattering* and *ranking* respectively, and can be done efficiently [Dalvi and Suciu, 2012].

Step 0 covers the base case where Q is simple a tuple, so it looks it up in $\mathcal{P}$. *Step 1* attemps 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 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 recurse. Otherwise, *Step 3* recurses using the inclusion-exclusion principle,

performing cancellations first to maintain efficiency [Dalvi and Suciu, 2012]. 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 cannot be computed efficiently [Dalvi and Suciu, 2012].

4.2.3 Open-World Probabilistic Databases

In the context of automatically constructing a knowledge base, as is done in for example NELL [Carlson et al., 2010] or Google's Knowledge Vault [Dong et al., 2014], making the closed-world assumption is conceptually unreasonable. Conversely, it is also not feasible to include all possible tuples and their probabilities in the knowledge base. The resulting difficulty is that there are an enormous number of probabilistic facts that can be scraped from the internet, and by definition these tools will keep only those with the very highest probability. As a result, knowledge bases like NELL [Carlson et al., 2010], PaleoDeepDive [Peters et al., 2014], and YAGO [Suchanek et al., 2007] consist almost entirely of probabilities above 0.95.

This tells us that the knowledge base we are looking at is fundamentally *incomplete*. In response to this problem, Ceylan et al. [2016a] propose the notion of a *completion* for a probabilistic database.

Definition 7. *A* λ-completion *of a probabilistic database* $\mathcal{P}$ *is another probabilistic database obtained as follows. For each atom* t *that does not appear in* $\mathcal{P}$*, we add tuple* $\langle t : p \rangle$ *to* $\mathcal{P}$ *for some* $p \in [0, \lambda]$.

Then, we can define the open world of possible databases in terms of the set of distributions induced by all completions.

Definition 8. *An* open-world probabilistic database *(OpenPDB) is a pair* $\mathcal{G} = (\mathcal{P}, \lambda)$*, where* $\mathcal{P}$ *is a*

probabilistic database and $\lambda \in [0, 1]$. $\mathcal{G}$ induces a set of probability distributions $K_{\mathcal{G}}$ such that a distribution P belongs to $K_{\mathcal{G}}$ iff P is induced by some λ-completion of probabilistic database $\mathcal{P}$.

4.2.3.1 Open-World Queries

OpenPDBs specify a set of probability distributions rather than a single one, meaning that a given query produces a set of possible probabilities rather than a single one. We focus on computing the minimum and maximum possible probability values that can be achieved by completing the database.

Definition 9. *The* probability interval of a Boolean query Q *in OpenPDB $\mathcal{G}$ is the pair* $K_{\mathcal{G}}(\mathsf{Q}) = [\underline{P}_{\mathcal{G}}(\mathsf{Q}), \overline{P}_{\mathcal{G}}(\mathsf{Q})]$*, where*

$$\underline{P}_{\mathcal{G}}(\mathsf{Q}) = \min_{P \in K_{\mathcal{G}}} P(\mathsf{Q}) \qquad\qquad \overline{P}_{\mathcal{G}}(\mathsf{Q}) = \max_{P \in K_{\mathcal{G}}} P(\mathsf{Q})$$

In general, computing the probability interval for some first-order Q is not tractable. As observed in Ceylan et al. [2016a], however, the situation is different for UCQ queries, because they are monotone (they contain no negations). For UCQs, the upper and lower bounds are given respectively by the full completion (where all unknown probabilities are λ), and the closed world database. This is a direct result of the fact that OpenPDBs form a credal set: a closed convex set of probability measures, meaning that probability bounds always come from extreme points [Cozman, 2000].

Furthermore, Ceylan et al. [2016a] also provide an algorithm for efficiently computing this upper bound corresponding to a full completion, and show that it works whenever the UCQ is safe.

4.3 Mean-Constrained Completions

This section motivates the need to strengthen the OpenPDB semantics, and introduces our novel probabilistic data model.

4.3.1 Motivation

The ability to perform efficient query evaluation provides an appealing case for OpenPDBs. They give a more reasonable semantics, better matching their use, and for a large class of queries they come at no extra cost in comparison to traditional PDBs. However, in practice computing an upper bound in this way tends to give results very close to 1. Intuitively, this makes sense: our upper bound comes from simultaneously assuming that *every* possible missing atom has some reasonable probability. While such a bound is easy to compute, it is too strong of a relaxation of the closed-world assumption.

Recall the motivation for the initial OpenPDB semantics: statistical knowledge base construction (KBC) tools store only the most likely extracted tuples [Ceylan et al., 2016a]. The λ parameter in OpenPDBs is designed to account for this, representing an upper bound on the probability of unobserved tuples. However, this discards other information potentially collected by the KBC system: for example, suppose that a table in our database describes whether or not a person is a scientist. The OpenPDB model will account for the fact that many of the people we discard have a non-zero chance of being a scientist, but it will not take into account the fact that our KBC system observes that fewer than 1% of the population are scientists.

In order to consider a restricted subset of completions representing reasonable situations, we propose directly incorporating these summary statistics. Specifically, we place constraints on the overall probability of a relation across the entire population. In the scientist example, our model only considers completions in which the probability mass of people being scientists totals less than 1%. This allows us to include more information at the domain level, without having more information about each individual.

Example. To illustrate the effect this has, consider a schema in which we have 3 relations: $LiLA(x)$ denoting whether one lives in Los Angeles, $LiSpr(x)$ denoting whether one lives in Springfield, and $S(x)$ denoting whether one is a scientist. Using a vocabulary of 500 people where each person is present in at most one relation, Table 4.1 shows the resulting upper probability bound

Query	CW	OW	CoOW
$LiLA(x), S(x)$	0	$1 - 10^{-290}$	$1 - 10^{-15}$
$LiSpr(x), S(x)$	0	$1 - 10^{-191}$	0.96

Table 4.1: Comparison of upper bounds for the same query and database with different model assumptions: Closed-World (CW), Open-World (OW), and Constrained Open-World (CoOW).

under different model assumptions, where the constrained open-world restricts at most 50% of mass on $LiLA$, 5% on S, and 0.5% on $LiSpr$. In particular, notice how extreme the difference is in upper bound with and without constraints being imposed. The closed-world probability of both of these queries is always 0, as each person in our database only has a known probability for at most one relation. It is clear that of these three options, the constrained open-world is the most reasonable – the rest of this section formalizes this idea and investigates the resulting properties.

4.3.2 Formalization

We begin here by defining mean-based constraints, before examining some immediate observations about the structure of the resulting constrained database.

Definition 10. *Suppose we have a PDB $\mathcal{P}$, and let $Tup(R) \subseteq \mathcal{P}$ be the set of probabilistic tuples in relation R. Let $\bar{p}$ be a probability threshold. Then a* mean tuple probability constraint *(MTP constraint) φ is a linear constraint of the form*

$$\bar{p} \geq \frac{1}{|\,Tup(R)|} \sum_{\langle t:p\rangle \in Tup(R)} p$$

Definition 11. *We say that a λ-completion is φ-*constrained *if the λ-completed database satisfies MTP φ. If it satisfies all of $\Phi = (\varphi_1, \varphi_2, ..., \varphi_n)$, then we say it is Φ-constrained.*

Being φ-constrained is not a property of OpenPDBs, but of their PDB completions. Hence, we are interested in the subset of completions that satisfy this property.

Definition 12. *An OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$ together with MTP constraints Φ induces a set of proba-*

bility distributions $K^{\Phi}_{\mathcal{G}}$, where distribution P belongs to $K^{\Phi}_{\mathcal{G}}$ iff P is induced by some Φ-constrained λ-completion of $\mathcal{P}$.

Much like with standard OpenPDBs, for a Boolean query Q we are interested in computing bounds on $P(\mathsf{Q})$.

Definition 13. *The probability interval of a Boolean query Q in OpenPDB $\mathcal{G}$ with MTP constraints Φ is $K^{\Phi}_{\mathcal{G}}(\mathsf{Q}) = [\underline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q}), \overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q})]$, where*

$$\underline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q}) = \min_{P \in K^{\Phi}_{\mathcal{G}}} P(\mathsf{Q}); \qquad \overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q}) = \max_{P \in K^{\Phi}_{\mathcal{G}}} P(\mathsf{Q}).$$

4.3.3 Completion Properties

A necessary property of OpenPDBs for efficient query evaluation is that they are credal – this is what allows us to consider only a finite subset of possible completions. MTP-constrained OpenPDBs maintain this property.[1]

Proposition 8. *Suppose we have an OpenPDB $\mathcal{G}$ together with MTP constraints Φ. Then the induced set of probability distributions $K^{\Phi}_{\mathcal{G}}$ is credal.*

This property allows us to examine only a finite subset of configurations when looking at potential completions, since query probability bounds of a credal set are always achieved at points of extrema [Cozman, 2000]. Next, we would like to characterize these points of extrema, by showing that the number of tuples not on their own individual boundaries (that is, 0 or λ) is given by the number of MTP constraints.

Theorem 9. *Suppose we have an OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$ with MTP constraints Φ, and a UCQ Q. Then there exists a Φ-constrained λ-completion $\mathcal{P}'$ for which $P_{\mathcal{P}'}(\mathsf{Q}) = \overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q})$ and that contains*

[1] Proofs of all theorems and lemmas are available in appendix of the full version of the paper at http://starai.cs.ucla.edu/papers/FriedmanIJCAI19.pdf

some $\mathcal{T} \subseteq \mathcal{P}' \setminus \mathcal{P}$ such that $|\mathcal{T}| \leq |\Phi|$, and

$$\forall \langle t : p \rangle \in \mathcal{T} : \quad p \in [0, \lambda], \text{ and}$$
$$\forall \langle t : p \rangle \in (\mathcal{P}' \setminus \mathcal{P}) \setminus \mathcal{T} : \quad p \in \{0, \lambda\}.$$

That is, our upper bound is given by a completion that has at most $|\Phi|$ added tuples with probability not exactly 0 or λ. Intuitively, each MTP constraint contributes a single non-boundary tuple, which can be thought of as the "leftover" probability mass once the rest has been assigned in full.

This insight allows us to treat MTP query evaluation as a combinatorial optimization problem for the rest of this paper. Thus, we only consider the case where achieving the mean tuple probability exactly leaves us with every individual tuple at its boundary. To see that we can do this, we observe that Theorem 9 leaves a single tuple per MTP constraint not necessarily on the boundary. But this tuple can always be forced to be on the boundary by very slightly increasing the mean $\bar{p}$ of the constraint, as follows.

Corollary 2. *Suppose we have an OpenPDB $\mathcal{G} = (\mathcal{P}, \lambda)$ with MTP constraints Φ, and a UCQ Q. Suppose further that each relation in $\mathcal{G}$ has at most 1 constraint in Φ, and that each constraint allows adding open-world probability mass exactly divisible by λ. Then there exists a Φ-constrained λ-completion $\mathcal{P}'$, where $K^{\Phi}_{\mathcal{G}}(\mathsf{Q}) = [P_{\mathcal{P}}(\mathsf{Q}), P_{\mathcal{P}'}(\mathsf{Q})]$, and*

$$\forall \langle t : p \rangle \in \mathcal{P}' \setminus \mathcal{P} : \quad p \in \{0, \lambda\}.$$

Our investigation into the algorithmic properties of MTP query evaluation will be focused on constraining a single relation, subject to a single combinatorial budget constraint.

4.4 Exact MTP Query Evaluation

With Section 4.3 formalizing MTP constraints and showing that computing upper bounds subject to MTP constraints is a combinatorial problem of choosing which λ-probability tuples to add in the completion, we now investigate exact solutions. With this now being a combinatorial problem, we slightly change our terminology: "adding" an open-world tuple t to a relation means we consider only completions where $P(t) = \lambda$, and a "budget" b for a relation means we can add up to b tuples while still satisfying the MTP constraint.

4.4.1 An Algorithm for Inversion-Free Queries

We begin by describing a class of queries that admits poly-time evaluation subject to an MTP constraint. We first need to define some syntactic properties of queries.

Definition 14. *Let* Q *be a conjunctive query, and let* $at(x)$ *denote the set of relations containing variable* x*. We say that* Q *is* hierarchical *if for any* x, y*, we have either* $at(x) \subseteq at(y)$*,* $at(y) \subseteq at(x)$*, or* $at(x) \cap at(y) = \emptyset$*.*

Intuitively, a conjunctive query being hierarchical indicates that it can either be separated into independent parts (the $at(x) \cap at(y) = \emptyset$ case), or there is some variable that appears in every atom. This simple syntactic property is the basis for determining whether query evaluation on a conjunctive query can be done in polynomial time [Dalvi and Suciu, 2007]. We can further expand on this definition in the context of UCQs.

Definition 15. *A UCQ* Q *is* inversion-free *if each of its disjuncts is hierarchical, and they all share the same hierarchy.*[2] *If* Q *is not inversion-free, we say that it has an inversion.*

Inversion-free queries represent an especially tractable class of queries for general inference. Since they are hierarchical, they are also safe, meaning query evaluation is efficient. Moreover, they precisely characterize the class of queries that support compilation to a tractable form for

[2] See Jha and Suciu [2011] for a more detailed definition.

performing more complex queries, such as computing any joint distribution [Jha and Suciu, 2011]. This query class remains tractable under MTP constraints.

Theorem 10. *For any inversion-free query* Q*, evaluating the probability* $\overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q})$ *subject to an MTP constraint is in PTIME.*

In order to prove Theorem 10, we provide a polytime algorithm for MTP query evaluation on inversion-free queries. As with OpenPDBs, our algorithm depends on Algorithm 2, the standard lifted inference algorithm for PDBs that was discussed in Section 4.2.

We now present an algorithm for doing exact MTP query evaluation on inversion-free queries. For brevity, we present the case of a binary relation; the general case follows similarly and can be found in appendix. Suppose that we have a probabilistic database $\mathcal{P}$, a domain T of constants denoted c, a query Q, and an MTP constraint on relation $R(x, y)$ allowing us to add exactly b tuples with probability λ. Suppose that Q immediately reaches Step 5 of Algorithm 2 (other steps will be discussed later), implying that x and y are unique variables in the query. We let $A(c_x, c_y, b)$ denote the upper query probability of $\mathsf{Q}(x/c_x, y/c_y)$ subject to an MTP constraint allowing budget b on R restricted to $x = c_x, y = c_y$. That is, A tells us the highest probability we can achieve for a partial assignment given a fixed budget. Observe that we can compute all entries of A using a slight modification of Algorithm 2 where we compute probabilities with and without each added tuple. This will take time polynomial in $|T|$.

Next, we impose an ordering $c_1, \ldots, c_{|T|}$ on the domain. Then we let $D(j, c_y, b)$ denote the upper query probability of

$$\bigvee_{c \in \{c_1, \ldots c_j\}} \mathsf{Q}(x/c, y/c_y)$$

with a budget of b on the relevant portions of R. Then $D(|T|, c_y, b)$ considers all possible substitutions in our first index, meaning we have effectively removed a variable. Doing this repeatedly would allow us to perform exact MTP query evaluation. However, D is non-trivial to compute, and

cannot be done by simply modifying Algorithm 2. Instead, we observe the following recurrence:

$$D(j+1, y/c_y, b) = \max_{k \in \{1,\ldots,b\}} 1 - (1 - D(j, y/c_y, b-k)) \cdot (1 - A(x/c_{j+1}, y/c_y, k))$$

Intuitively, this recurrence says that since the tuples from each fixed constant are all independent, we do not need to store which budget configuration on the first j constants got us our optimal solution. Thus, when we add the $j+1$th constant, we just need to check each possible value we could assign to our new constant, and see which gives the overall highest probability. This recurrence can be implemented efficiently, yielding a dynamic programming algorithm that runs in time polynomial in the domain size and budget.

Finally, we would like to generalize this algorithm beyond the assumption that Q immediately reaches Step 5 of Algorithm 2. Looking at other cases, we see that Steps 0 and 1 have no effect on this recurrence, and that Steps 2 and 4 correspond to multiplicative factors. For a query that reaches Step 3 (inclusion-exclusion), we need to construct such A and D for each term in the inclusion-exclusion sum, and follow the analogous recurrence.

Notice that the modified algorithm would only work in the case where we can always pick a common variable for all sub-queries to do dynamic programming on – that is, when the query is inversion-free, as was our assumption. If the sub-calls generated by inclusion-exclusion do not share a common variable hierarchy, and thus an order for using our dynamic programming algorithm, we suffer an exponential blowup.

4.4.2 Queries with Inversion

We now show that allowing for inversions in safe queries can cause MTP query evaluation to become NP-hard. Interestingly, this means that MTP constraints fundamentally change the difficulty

landscape of query evaluation.

To show this, we investigate the following UCQ query.

$$\begin{aligned} M_0 = \exists x \exists y \exists z \, (R(x,y,z) \wedge U(x)) \vee (R(x,y,z) \wedge V(y)) \\ \vee (R(x,y,z) \wedge W(z)) \vee (U(x) \wedge V(y)) \\ \vee (U(x) \wedge W(z)) \vee (V(y) \wedge W(z)) \end{aligned}$$

A key observation here is that the query M_0 is a *safe* UCQ. That is, if we ignore constraints and evaluate it subject to the closed- or open-world semantics, computing the probability of the query would be polynomial in the size of the database. We now show that this is *not* the case for open-world query evaluation subject to a single MTP constraint on R.

Theorem 11. *Evaluating the upper query probability bound $\overline{P}^{\Phi}_{\mathcal{G}}(M_0)$ subject to an MTP constraint Φ on R is NP-hard.*

The full proof of Theorem 11 can be found in appendix, showing a reduction from the NP-complete 3-dimensional matching problem to computing $\overline{P}^{\Phi}_{\mathcal{G}}(M_0)$ with an MTP constraint on R. It uses the following intuitive correspondence.

Definition 16. *Let X, Y, Z be finite disjoint sets representing nodes, and let $T \subseteq X \times Y \times Z$ be the set of available hyperedges. Then $M \subseteq T$ is a* matching *if for any distinct triples $(x_1, y_1, z_1) \in M, (x_2, y_2, z_2) \in M$, we have that $x_1 \neq x_2, y_1 \neq y_2, z_1 \neq z_2$. The* 3-dimensional matching *decision problem is to determine for a given X, Y, Z, T and positive integer k if there exists a matching M with $|M| \geq k$.*

The set of available tuples for R will correspond to all edges in T. The MTP constraint on R forces a decision on which subset of T to add to the database.

However, if we simply queried to maximize $P(R(x,y,z))$, this completion need not correspond to a matching. Instead, we have the disjunct $R(x,y,z) \wedge U(x)$ which is maximized when each tuple chosen from R has a different x value. Similar disjuncts for y and z ensure that the query is

maximized when using distinct y and z values. Putting all of these together ensures that the query probability is maximized when the subset of tuples chosen to complete R form a matching.

Finally, the last part of the query $(U(x) \wedge V(y)) \vee (U(x) \wedge W(z)) \vee (V(y) \wedge W(z))$ ensures that inference on M_0 is tractable, but it is unaffected by the choice of tuples in R.

4.5 Approximate MTP Query Evaluation

With Section 4.4.2 answering definitively that a general-purpose algorithm for evaluating MTP query bounds is unlikely to exist, even when restricted to safe queries, an approximation is the logical next step. We now restrict our discussion to situations where we constrain a single relation, and dig deeper into the properties of MTP constraints to show their submodular structure. We then exploit this property to achieve efficient bounds with guarantees.

4.5.1 On the Submodularity of Adding Tuples

To formally define and prove the submodular structure of the problem, we analyze query evaluation as a set function on adding tuples. We begin with a few relevant definitions.

Definition 17. *Suppose that we have an OpenPDB $\mathcal{G}$, with an MTP constraint φ on a single relation R, and we let $\mathbf{O}$ be the set of possible tuples we can add to R. Then the* set query probability *function $S_{\mathcal{P},\mathsf{Q}} : 2^{\mathbf{O}} \to [0, 1]$ is defined as*

$$S_{\mathcal{P},\mathsf{Q}}(X) = P_{\mathcal{P} \cup \{\langle t:\lambda \rangle | t \in X\}}(\mathsf{Q}).$$

Intuitively, this function describes the probability of the query as a function of which open tuples have been added. It provides a way to reason about the combinatorial properties of this optimization problem. Observe that $S_{\mathcal{P},\mathsf{Q}}(\emptyset)$ is the closed-world probability of the query, while $S_{\mathcal{P},\mathsf{Q}}(\mathbf{O})$ is the open-world probability.

We want to show that $S_{\mathcal{P},\mathsf{Q}}$ is a submodular set function.

Definition 18. *A* submodular set function *is a function $f : 2^{\Omega} \to \mathbb{R}$ such that for every $X \subseteq Y \subseteq \Omega$, and every $x \in \Omega \setminus Y$, we have that*

$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y).$$

Theorem 12. *The set query probability function $S_{\mathcal{P},\mathsf{Q}}$ is submodular for any tuple independent probabilistic database $\mathcal{P}$ and UCQ query Q without self-joins.*

This gives us the desired submodularity property, which we can exploit to build efficient approximation algorithms.

4.5.2 From Submodularity to Approximation

Given the knowledge that the probability of a safe query without self-joins is submodular in the completion of a single relation, we are now tasked with using this to construct an efficient approximation. Since we further know the probability is also monotone as we have restricted our language to UCQs, Nemhauser et al. [1978] tell us that we can get a $1 - \frac{1}{e}$ approximation using a simple greedy algorithm. The final requirement to achieve the approximation described in Nemhauser et al. [1978] is that our set function must have the property that $f(\emptyset) = 0$. This can be achieved in a straightforward manner as follows.

Definition 19. *In the context of the set query probability function of Definition 17, the* normalized *set query probability function $S'_{\mathcal{P},\mathsf{Q}} : 2^{\mathbf{O}} \to [0, 1]$ is defined as*

$$S'_{\mathcal{P},\mathsf{Q}}(X) = P_{\mathcal{P} \cup \{\langle t:\lambda \rangle | t \in X\}}(\mathsf{Q}) - P_{\mathcal{P}}(\mathsf{Q}).$$

Proposition 13. *Any normalized set query probability function $S'_{\mathcal{P},\mathsf{Q}}$ is monotone, submodular, and satifies $S'_{\mathcal{P},\mathsf{Q}}(\emptyset) = 0$.*

By simply normalizing the set query probability function, we can now directly apply the greedy

approximation described in Nemhauser et al. [1978]. We slightly modify Algorithm 2 to efficiently compute the next best tuple to add based on the current database, and add it. This is repeated until adding another tuple would violate the MTP constraint. Finally, we say that $P_{Greedy}(\mathsf{Q})$ is the approximation given by this greedy algorithm and recall that the true upper bound is $\overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q})$. We observe that $P_{Greedy}(\mathsf{Q}) \leq \overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q})$. Furthermore, Nemhauser et al. [1978] tells us the following:

$$P_{Greedy}(\mathsf{Q}) - P_{\mathcal{P}}(\mathsf{Q}) \geq (1 - \frac{1}{e})(\overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q}) - P_{\mathcal{P}}(\mathsf{Q}))$$

Combining these and multiplying through gives us the following upper and lower bound on the desired probability.

$$P_{Greedy}(\mathsf{Q}) \leq \overline{P}^{\Phi}_{\mathcal{G}}(\mathsf{Q}) \leq \frac{e \cdot P_{Greedy}(\mathsf{Q}) - P_{\mathcal{P}}(\mathsf{Q})}{e - 1} \tag{4.1}$$

It should be noted that depending on the query and database, it is possible for this upper bound to exceed 1.

4.6 Discussion, Future & Related Work

We propose the novel problem of constraining open-world probabilistic databases at the schema level, without having any additional ground information over individuals. We introduced a formal mechanism for doing this, by limiting the *mean tuple probability* allowed in any given completion, and then sought to compute bounds subject to these constraints. We now discuss remaining open problems and related work.

Section 4.4 showed that there exists a query that is NP-hard to compute exactly, and also presented a tractable algorithm for a class of inversion-free queries. The question remains how hard the other queries are – in particular, is the algorithm presented complete. Is there a complexity dichotomy, that is, a set of syntactic properties that determine the hardness of a query subject to MTP constraints. Questions of this form are a central object of study in probabilistic databases. It has been explored for conjunctive queries [Dalvi and Suciu, 2007], UCQs [Dalvi and Suciu, 2012],

and a more general class of queries with negation [Fink and Olteanu, 2016].

The central goal of our work is to find stronger semantics based on OpenPDBs, while still maintaining their desirable tractability. This notion of achieving a powerful semantics while maintaining tractability is a common topic of study. De Raedt and Kimmig [2015] study this problem by using a probabilistic interpretation of logic programs to define a model, leading to powerful semantics but a more limited scope of tractability [Fierens et al., 2015]. Description logics [Nardi et al., 2003] are a knowledge representation formalism that can be used as the basis for a semantics. This is implemented in a probabilistic setting in, for example, probabilistic ontologies [Riguzzi et al., 2012, 2015], probabilistic description logics [Heinsohn, 1994], probabilistic description logic programs [Lukasiewicz, 2005], or the bayesian description logics [Ceylan and Peñaloza, 2014].

Probabilistic databases in particular are of interest due to their simplicity and practicality. Foundational work defines a few types of probabilistic semantics, and provides efficient algorithms as well as when they can be applied Dalvi and Suciu [2004, 2007, 2012]. These algorithms along with practical improvements are implemented as industrial level systems such as MystiQ [Ré and Suciu, 2008], SPROUT [Olteanu et al., 2009], MayBMS [Huang et al., 2009], and Trio which implements the closely related Uncertainty-Lineage Databases [Benjelloun et al., 2007].

Problems outside of simple query evaluation are also points of interest for PDBs, for example the most probable database problem [Gribkoff et al., 2014], or the problem of ranking the top-k results [Ré et al., 2007]. In the context of OpenPDBs in particular, Grohe and Lindner [2018] study the notion of an infinite open world, using techniques from analysis to explore when this is feasible. Borgwardt et al. [2017] study an orthogonal way to introduce constraints on OpenPDBs to make the probability bounds realistic, by adding logical constraints based on an ontology.

Finally, probabilistic databases are closely related to other statistical relational models such as Markov logic networks [Richardson and Domingos, 2006] and probabilistic soft logic [Kimmig et al., 2012]. These models implicitly support the open-world assumption, although inference will not be efficient in general given the large number of random variables induced by the open world.

CHAPTER 5

Symbolic Querying of Vector Spaces: Probabilistic Databases Meets Relational Embeddings

5.1 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.

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 [Nickel et al., 2015, 2011, Kazemi and Poole, 2018, Trouillon et al., 2016] using a variety of techniques [Blockeel and De Raedt, 1997, Dumancic et al., 2019]. 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 [Hamilton et al., 2018, Krompass et al., 2014].

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 [Suciu et al., 2011, Van den Broeck and Suciu, 2017] 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 [Dalvi and Suciu, 2004, 2007, 2012, Fink and Olteanu, 2016] and approximations [den Heuvel et al., 2019, Gribkoff and Suciu, 2016]. Recently there has even been work in finding explanations for queries [Ceylan et al., 2017, Gribkoff et al., 2014], and querying subject to constraints [Borgwardt et al., 2017, Bienvenu, 2016, Friedman and Van den Broeck, 2019]. 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 [Dalvi and Suciu, 2012].

In this work, our goal will be to unify the predictive capability of relational embedding models with the sound underlying 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 5.3, 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 [Yang et al., 2014a, Nickel et al., 2011] 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 5.2 provides the required technical background on PDBs and their associated queries. In Section 5.3 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 5.4 we introduce TRACTOR, a relational embedding model designed around PDBs to allow for a large range of efficient queries. Section 5.5 provides an empirical evaluation of TRACTOR . Finally, Section 5.6 gives a broad discussion on related work along with ties to future work.

5.2 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.

5.2.1 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* [Hinrichs and Genesereth, 2006], 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 [Abiteboul et al., 1995], 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 [Reiter, 1981]. Figure 5.1 shows an example database.

5.2.2 Probabilistic Databases

To incorporate uncertainty into relational databases, *probabilistic databases* assign each tuple a probability [Suciu et al., 2011, Van den Broeck and Suciu, 2017].

Definition 20. *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 21. *A PDB* $\mathcal{P}$ *for vocabulary* σ *induces a probability distribution over* σ*-interpretations*

Scientist
Einstein
Erdős
von Neumann

CoAuthor	
Einstein	Erdős
Erdős	von Neumann

Figure 5.1: 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 5.2: 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 $0.8 \cdot 0.8 = 0.64$.

ω:

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

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

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.

5.2.3 Probabilistic Queries

Much as in relational databases, in probabilistic databases we are interested in answering queries – the difference 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 5.1, 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 5.2, 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 14. *Dalvi and Suciu [2012] 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 5.2.3.2);*
- 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 [Suciu et al., 2011, Dalvi and Suciu, 2012], as well as throughout this paper, UCQs (and consequently conjunctive queries) are the primary query object studied.

5.2.3.1 Reduction to Fully Quantified Boolean Queries

In general, one is not always interested in computing fully quantified queries. For example, in Section 5.5 one of the queries we are interested in computing will be of the form

$$\exists x, y.R(A, x) \wedge S(x, y) \wedge T(y, B) \tag{5.1}$$

For relations R, S, T and constants A, B. To convert this query to a fully quantified one, we need to *shatter* the query [Van den Broeck and Suciu, 2017]. 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:

$$H_0 = \exists x, y.R_A(x) \wedge S(x, y) \wedge T_B(y) \tag{5.2}$$

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.

5.2.3.2 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 3 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 [Dalvi and Suciu, 2012].

Step 0 covers the base case where Q is simply a tuple, so it looks it up in $\mathcal{P}$. *Step 1* attempts to

Algorithm 3 $\textbf{Lift}^{\mathbf{R}}(\mathsf{Q}, \mathcal{P})$, abbreviated by $\mathbf{L}(\mathsf{Q})$

Input: UCQ Q , prob. database $\mathcal{P}$ with constants T.
Output: 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)

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 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 [Dalvi and Suciu, 2012]. 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 [Dalvi and Suciu, 2012], under standard complexity assumptions.

5.3 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.

5.3.1 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 22. *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 $f(v_h, v_R, v_t) \to \mathbb{R}$ 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 5.1 gives some examples of models with the form their vector representations take, as well as their scoring functions.

5.3.2 Probabilistic Interpretations of Relational Embeddings

Given a relational embedding model from Definition 22, if we want to give it a clear probabilistic semantics using our knowledge of probabilistic databases from Section 5.2, we need to find a way

Table 5.1: Example relational embedding scoring functions for d dimensions

Method	Entity Embedding	Relation Embedding	Triple Score
TransE [Bordes et al., 2013]	$v_h, v_t \in \mathbb{R}^d$	$v_R \in \mathbb{R}^d$	$\|\|v_h + v_R - v_t\|\|$
DistMult [Yang et al., 2014a]	$v_h, v_t \in \mathbb{R}^d$	$v_R \in \mathbb{R}^d$	$\langle v_h, v_R, v_t \rangle$
Rescal [Nickel et al., 2011]	$v_h, v_t \in \mathbb{R}^d$	$v_R \in \mathbb{R}^{d \times d}$	$v_h^T v_R v_t$
ComplEx [Trouillon et al., 2016]	$v_h, v_t \in \mathbb{C}^d$	$v_R \in \mathbb{C}^d$	Re($\langle v_h, v_R, \bar{v}_t \rangle$)

to interpret the model as a probability distribution.

The simplest approach is to choose some mapping function $g : \mathbb{R} \to [0, 1]$ 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 5.3 gives an example of such a conversion, using the sigmoid function as the mapping.

After doing this conversion, we can directly use Algorithm 3 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 triple-independence presents potential issues as discussed in Meilicke et al. [2019]. 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 15. *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*

$R(x, y)$	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 5.3: An example of mapping a relational embedding to a probabilistic database using the sigmoid function.

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.

5.4 TRACTOR

The main takeaway from Section 5.3 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 [Yang et al., 2014a] or TransE [Bordes et al., 2013] is that they make simplifying assumptions about how entity and relation vectors relate to link prediction. In Section 5.3, 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.

5.4.1 Factorizing in Probabilistic Databases

Relational embedding models such as DistMult [Yang et al., 2014a] and ComplEx [Trouillon et al., 2016], or indeed any model derived from the canonical Polyadic decomposition [Hitchcock, 1927] 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 [Van den Broeck and Darwiche, 2013]. 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:

$$\forall x, y.R(x, y) \iff E(x) \wedge T(R) \wedge E(y) \tag{5.3}$$

We are assuming that all of the model's newly defined variables in E and T are independent random variables, so Equation 5.3 implies that

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

Figure 5.4 gives an example of probabilities for E and T, with corresponding probabilities for R subject to Equation 5.3. 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 5.3. 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 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 15. In fact, *any* UCQ can now be computed efficiently over the relations of interest. This will be proven in

$E(x)$	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 5.4: Example model tables E, T_R and a few corresponding predictions for R

Section 5.4.4, but intuitively binary relations must be involved for Algorithm 3 to get stuck, and our rewrite allows us to avoid this.

Of course, the major drawback is that Equation 5.3 describes an incredibly simple and inexpressive embedding model – we can only associate a single probability with each entity and relation! We address this next.

5.4.2 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 [Chow and Liu, 1968] provide efficient learning and inference, they are limited in their expressive capability: so a commonly used alternative is a mixture of such models [Meila and Jordan, 1998]. Similarly, while Gaussians are limited in their expressiveness, mixture of Gaussian models [Titterington et al., 1985] 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 5.3 as our building block, and use it to create TRACTOR.

Definition 23. TRACTOR *with d dimensions is a mixture of d models each constructed from Equation 5.3. 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*

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

$E_1(x)$	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 5.5: Example TRACTOR model tables E_1, E_2, T_1, T_2 and a few corresponding predictions for R

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

Figure 5.5 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 5.5 the embedding of entity A is $(E_1(A), E_2(A)) = (0.2, 0.6)$.

5.4.3 Equivalence to Distmult

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

Theorem 16. *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. [2014a] that DistMult is effective for link prediction, so TRACTOR must also be.

5.4.3.1 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 [Yang et al., 2014a]. It has been previously shown that probability theory can be consistently extended to negative probabilities [Bartlett, 1945], and their usefulness has also been documented in the context of probabilistic databases [Jha and Suciu, 2012, Van den Broeck et al., 2014]. Furthermore, by adding a simple disjunctive bias term, we can ensure that all fact predictions are indeed positive probabilities. In Section 5.5 we will explore both options.

5.4.4 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 5.3 using TRACTOR tables E and T. What remains is a query Q' which contains only unary relations.

Theorem 17. *Suppose that* Q' *is a UCQ consisting only of unary relations. Then* Q' *is safe.*

Proof. We prove this by showing that Algorithm 3 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 3, each CQ must have a separator. So Q' is safe.

□

5.5 Empirical Evaluation

We will now empirically investigate the effectiveness of TRACTOR as a relational embedding model. As discussed in Section 5.4.3, 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 [Yang et al., 2014a], 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.

5.5.1 Queries & Comparison Target

As our comparison for evaluation, we will use the graph query embeddings (GQE) [Hamilton et al., 2018] 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. [2018] 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 is available at https://github.com/ucla-starai/pdbmeetskge

Table 5.2: 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 5.2 gives a series of example template CQs and UCQs. In Figure 5.6, 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 5.6 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.

5.5.2 Dataset

For our dataset, we use the same choice in relational data as Hamilton et al. [2018]. In that work, two datsets were evaluated on, which were termed bio and reddit respectively. Bio is a dataset

consisting of knowledge 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.

5.5.2.1 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. [2018]. 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. [2018].

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.

5.5.3 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.

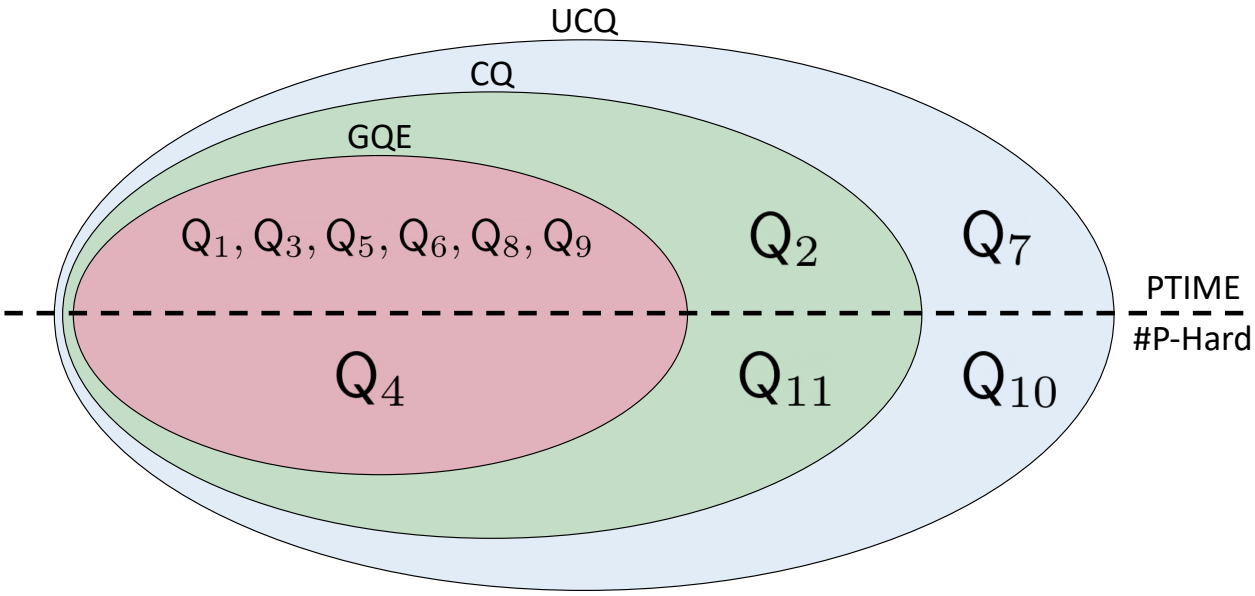


Figure 5.6: Categorizing different queries based on safeness and compatibility with GQE [Hamilton et al., 2018]. TRACTOR efficiently supports all queries in the diagram.

5.5.3.1 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. [2018] 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 [Nickel et al., 2011], DistMult [Yang et al., 2014a], and TransE [Bordes et al., 2013] 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.

Table 5.3: 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

5.5.3.2 Training

All model variants and baselines were trained using the max-margin approach with negative sampling [Mikolov et al., 2013] which has become standard for training relational embedding models [Nickel et al., 2015]. Parameters were optimized using the Adam optimizer [Kingma and Ba, 2015], with an embedding dimension of 128, a batch size of 256, and learning rate of 0.01.

5.5.3.3 Results & Discussion

Table 5.3 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.

5.6 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 [Hamilton et al., 2018], and learning new relations representing joins [Krompass et al., 2014]. 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 [Yang et al., 2014a] and TransE [Bordes et al., 2013] 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 [Trouillon et al., 2016], and keeping separate head and tail representations but using inverse relations to train them jointly [Kazemi and Poole, 2018]. 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 [Ruffinelli et al., 2020] while being more probabilistic in nature. In a similar vein, Tabacof and Costabello [2020] 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 [Ceylan et al., 2016b], with extensions to incorporate background information in the form of ontological knowledge [Borgwardt et al., 2019] and summary statistics [Friedman and Van den Broeck, 2019].

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 [2013] 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. [2014] 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. [2017] 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.

Appendix A

Semantic Loss

A.1 Axiomatization of Semantic Loss: Details

This appendix provides further details on our axiomatization of semantic loss. We detail here a complete axiomatization of semantic loss, which will involve restating some axioms and propositions from the main paper.

The first axiom says that there is no loss when the logical constraint α is always true (it is a logical tautology), independent of the predicted probabilities p.

Axiom 5 (Truth)**.** The semantic loss of a true sentence is zero: $\forall \mathsf{p}, \mathrm{L}^{\mathrm{s}}(\mathit{true}, \mathsf{p}) = 0$.

Next, when enforcing two constraints on disjoint sets of variables, we want the ability to compute semantic loss for the two constraints separately, and sum the results for their joint semantic loss.

Axiom 6 (Additive Independence)**.** Let α be a sentence over $\mathbf{X}$ with probabilities p. Let β be a sentence over $\mathbf{Y}$ disjoint from $\mathbf{X}$ with probabilities q. The semantic loss between sentence $\alpha \wedge \beta$ and the joint probability vector $[\mathsf{p}\,\mathsf{q}]$ decomposes additively: $\mathrm{L}^{\mathrm{s}}(\alpha \wedge \beta, [\mathsf{p}\,\mathsf{q}]) = \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) + \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{q})$.

It directly follows from Axioms 5 and 6 that the probabilities of variables that are not used on the constraint do not affect the semantic loss.

Proposition 18 formalizes this intuition.

Proposition 18 (Locality)**.** *Let α be a sentence over $\mathbf{X}$ with probabilities p. For any $\mathbf{Y}$ disjoint from $\mathbf{X}$ with probabilities q, the semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, [\mathsf{p}\,\mathsf{q}]) = \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$.*

Proof. Follows from the additive independence and truth axioms. Set $\beta = \mathit{true}$ in the additive independence axiom, and observe that this sets $\mathrm{L}^{\mathrm{s}}(\beta, \mathsf{q}) = 0$ because of the truth axiom. □

To maintain logical meaning, we postulate that semantic loss is monotone in the order of implication.

Axiom 7 (Monotonicity)**.** If $\alpha \models \beta$, then the semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{p})$ for any vector p.

Intuitively, as we add stricter requirements to the logical constraint, going from β to α and making it harder to satisfy, semantic loss cannot decrease. For example, when β enforces the output of an neural network to encode a subtree of a graph, and we tighten that requirement in α to be a path, semantic loss cannot decrease. Every path is also a tree and any solution to α is a solution to β.

A first consequence following the monotonicity axiom is that logically equivalent sentences must incur an identical semantic loss for the same probability vector p. Hence, the semantic loss is indeed a semantic property of the logical sentence, and *does not depend on the syntax* of the sentence.

Proposition 19. *If $\alpha \equiv \beta$, then the semantic loss $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\beta, \mathsf{p})$ for any vector p.*

A second consequence is that semantic loss must be non-negative.

Proposition 20 (Non-Negativity)**.** *Semantic loss is non-negative.*

Proof. Because $\alpha \models \mathit{true}$ for all α, the monotonicity axiom implies that $\forall \mathsf{p}, \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\mathit{true}, \mathsf{p})$. By the truth axiom, $\mathrm{L}^{\mathrm{s}}(\mathit{true}, \mathsf{p}) = 0$, and therefore $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) \geq 0$ for all choices of α and p. □

A state $\mathbf{x}$ is equivalently represented as a data vector, as well as a logical constraint that enforces a value for every variable in $\mathbf{X}$. When both the constraint and the predicted vector represent the same state (for example, $X_1 \wedge \neg X_2 \wedge X_3$ vs. $[1\,0\,1]$), there should be no semantic loss.

Axiom 8 (Identity)**.** For any state $\mathbf{x}$, there is zero semantic loss between its representation as a sentence, and its representation as a deterministic vector: $\forall \mathbf{x}, \mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) = 0$.

The axioms above together imply that any vector satisfying the constraint must incur zero loss. For example, when our constraint α requires that the output vector encodes an arbitrary total ranking, and the vector $\mathbf{x}$ correctly represents a single specific total ranking, there is no semantic loss.

Proposition 21 (Satisfaction)**.** *If* $\mathbf{x} \models \alpha$*, then the semantic loss* $\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x}) = 0$.

Proof of Proposition 21. The monotonicity axiom specializes to say that if $\mathbf{x} \models \alpha$, we have that $\forall \mathsf{p}, \mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathsf{p}) \geq \mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})$. By choosing p to be $\mathbf{x}$, this implies $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) \geq \mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x})$. From the identity axiom, $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathbf{x}) = 0$, and therefore $0 \geq \mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x})$. Proposition 20 bounds the loss from below as $\mathrm{L}^{\mathrm{s}}(\alpha, \mathbf{x}) \geq 0$. □

As a special case, logical literals (x or $\neg x$) constrain a single variable to take on a single value, and thus play a role similar to the labels used in supervised learning. Such constraints require an even tighter correspondence: semantic loss must act like a classical loss function (i.e., cross entropy).

Axiom 9 (Label-Literal Correspondence)**.** The semantic loss of a single literal is proportionate to the cross-entropy loss for the equivalent data label: $\mathrm{L}^{\mathrm{s}}(x, p) \propto -\log(p)$ and $\mathrm{L}^{\mathrm{s}}(\neg x, p) \propto -\log(1-p)$.

Next, we have the symmetry axioms.

Axiom 10 (Value Symmetry)**.** For all p and α, we have that $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\bar{\alpha}, 1 - \mathsf{p})$ where $\bar{\alpha}$ replaces every variable in α by its negation.

Axiom 11 (Variable Symmetry)**.** Let α be a sentence over $\mathbf{X}$ with probabilities p. Let π be a permutation of the variables $\mathbf{X}$, let $\pi(\alpha)$ be the sentence obtained by replacing variables x by $\pi(x)$, and let $\pi(\mathsf{p})$ be the corresponding permuted vector of probabilities. Then, $\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\pi(\alpha), \pi(\mathsf{p}))$.

The value and variable symmetry axioms together imply the equality of the multiplicative constants in the label-literal duality axiom for all literals.

Lemma 22. *There exists a single constant* K *such that* $\mathrm{L}^{\mathrm{s}}(X, p) = -K \log(p)$ *and* $\mathrm{L}^{\mathrm{s}}(\neg X, p) = -K \log(1-p)$ *for any literal* x.

Proof. Value symmetry implies that $\mathrm{L}^{\mathrm{s}}(X_i, \mathsf{p}) = \mathrm{L}^{\mathrm{s}}(\neg X_i, 1-\mathsf{p})$. Using label-literal correspondence, this implies $K_1 \log(p_i) = K_2 \log(1-(1-p_i))$ for the multiplicative constants K_1 and K_2 that are left unspecified by that axiom. This implies that the constants are identical. A similar argument based on variable symmetry proves equality between the multiplicative constants for different i. □

Finally, this allows us to prove the following form of semantic loss for a state $\mathbf{x}$.

Lemma 23. *For state* $\mathbf{x}$ *and vector* p*, we have* $\mathrm{L}^{\mathrm{s}}(\mathbf{x}, \mathsf{p}) \propto -\sum_{i:\mathbf{x}\models X_i} \log \mathsf{p}_i - \sum_{i:\mathbf{x}\models \neg X_i} \log(1-\mathsf{p}_i)$.

Proof of Lemma 23. A state $\mathbf{x}$ is a conjunction of independent literals, and therefore subject to the additive independence axiom. Each literal's loss in this sum is defined by Lemma 22. □

The following and final axiom requires that semantic loss is proportionate to the logarithm of a function that is additive for mutually exclusive sentences.

Axiom 12 (Exponential Additivity)**.** Let α and β be mutually exclusive sentences (i.e., $\alpha \wedge \beta$ is unsatisfiable), and let $f^s(K, \alpha, \mathsf{p}) = K^{-\mathrm{L}^{\mathrm{s}}(\alpha, \mathsf{p})}$. Then, there exists a positive constant K such that $f^s(K, \alpha \vee \beta, \mathsf{p}) = f^s(K, \alpha, \mathsf{p}) + f^s(K, \beta, \mathsf{p})$.

We are now able to state and prove the main uniqueness theorem.

Theorem 24 (Uniqueness)**.** *The semantic loss function in Definition 1 satisfies all axioms in Appendix A.1 and is the only function that does so, up to a multiplicative constant.*

Proof of Theorem 24. The truth axiom states that $\forall \mathsf{p}, f^s(K, true, \mathsf{p}) = 1$ for all positive constants K. This is the first Kolmogorov axiom of probability. The second Kolmogorov axiom for $f^s(K, ., \mathsf{p})$ follows from the additive independence axiom of semantic loss. The third Kolmogorov axiom (for the finite discrete case) is given by the exponential additivity axiom of semantic loss. Hence, $f^s(K, ., \mathsf{p})$ is a probability distribution for some choice of K, which implies the definition up to a multiplicative constant. □


(a) Confidently Correct


(b) Unconfidently Correct


(c) Unconfidently Incorrect


(d) Confidently Incorrect

Figure A.1: FASHION pictures grouped by how confidently the supervised base model classifies them correctly. With semantic loss, the final semi-supervised model predicts all correctly and confidently.

A.2 Specification of the Convolutional Neural Network Model

Table A.1 shows the slight architectural difference between the CNN used in ladder nets and ours. The major difference lies in the choice of ReLu. Note we add standard padded cropping to preprocess images and an additional fully connected layer at the end of the model, neither is used in ladder nets. We only make those slight modification so that the baseline performance reported by Rasmus et al. [2015] can be reproduced.

A.3 Hyper-parameter Tuning Details

Validation sets are used for tuning the weight associated with semantic loss, the only hyper-parameter that causes noticeable difference in performance for our method. For our semi-supervised classification experiments, we perform a grid search over $\{0.001, 0.005, 0.01, 0.05, 0.1\}$ to find the optimal value. Empirically, 0.005 always gives the best or nearly the best results and we report its results on all experiments.

For the FASHION dataset specifically, because MNIST and FASHION share the same image size and structure, methods developed in MNIST should be able to directly perform on FASHION without heavy modifications. Because of this, we use the same hyper-parameters when evaluating our method. However, for the sake of fairness, we subject ladder nets to a small-scale parameter tuning in case its performance is more volatile.

For the grids experiment, the only hyper parameter that needed to be tuned was again the weight

Table A.1: Specifications of CNNs in Ladder Net and our proposed method.

CNN in Ladder Net	CNN in this paper
Input 32×32 RGB image	
	Resizing to 36 × 36 with padding; Cropping Back
Whitening Contrast Normalization Gaussian Noise with std. of 0.3	
3×3 conv. 96 BN LeakyReLU 3×3 conv. 96 BN LeakyReLU 3×3 conv. 96 BN LeakyReLU	3×3 conv. 96 BN ReLU 3×3 conv. 96 BN ReLU 3×3 conv. 96 BN ReLU
2×2 max-pooling stride 2 BN	
3×3 conv. 192 BN LeakyReLU 3×3 conv. 192 BN LeakyReLU 3×3 conv. 192 BN LeakyReLU	3×3 conv. 192 BN ReLU 3×3 conv. 192 BN ReLU 3×3 conv. 192 BN ReLU
2×2 max-pooling stride 2 BN	
3×3 conv. 192 BN LeakyReLU 1×1 conv. 192 BN LeakyReLU 1×1 conv. 10 BN LeakyReLU	3×3 conv. 192 BN ReLU 3×3 conv. 192 BN ReLU 1×1 conv. 10 BN ReLU
Global meanpool BN	
	Fully connected BN
10-way softmax	

given to semantic loss, which after trying $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$ was selected to be 0.5 based on validation results. For the preference learning experiment, we initially chose the semantic loss weight from $\{0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$ to be 0.1 based on validation, and then further tuned the weight to 0.25.

A.4 Complex Constraint Specification

Grids To compile our grid constraint, we first use Nishino et al. [2017] to generate a constraint for each source destination pair. Then, we conjoin each of these with indicators specifying which

source and destination pair must be used, and finally we disjoin all of these together to form our constraint.

To generate the data, we begin by randomly removing one third of edges. We then filter out connected components with fewer than 5 nodes to reduce degenerate cases, and proceed with randomly selecting pairs of nodes to create data points.

The predictive model we employ as our baseline is a 5 layer MLP with 50 hidden sigmoid units per layer. It is trained using Adam Optimizer, with full data batches [Kingma and Ba, 2015]. Early stopping with respect to validation loss is used as a regularizer.

Preference Learning We split each user's ordering into their ordering over sushis 1,2,3,5,7,8, which we use as the features, and their ordering over 4,6,9,10 which are the labels we predict. The constraint is compiled directly from logic, as this can be done in a straightforward manner for an n-item ordering.

The predictive model we use here is a 3 layer MLP with 25 hidden sigmoid units per layer. It is trained using Adam Optimizer with full data batches [Kingma and Ba, 2015]. Early stopping with respect to validation loss is used as a regularizer.

A.5 Probabilistic Soft Logic Encodings

We here give both encodings on the exactly-one constraint over three x_1, x_2, x_3. The first encoding is:

$$(\neg x_1 \wedge x_2 \wedge x_3) \vee (x_1 \wedge \neg x_2 \wedge x_3) \vee (x_1 \wedge x_2 \wedge \neg x_3)$$

The second encoding is:

$$(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_3) \wedge (\neg x_2 \vee \neg x_3)$$

Both encodings extend to cases whether the number of variables is arbitrary.

The norm functions used for these experiments are as described in Kimmig et al. [2012], with the loss for an interpretation I being defined as follows:

$$x_1 \wedge x_2 = max\{0, I(x_1) + I(x_2) - 1\}$$
$$x_1 \vee x_2 = min\{I(x_1) + I(x_2), 1\}$$
$$\neg x_1 = 1 - I(x_1)$$

Appendix B

Collapsed Compilation

B.1 Proof of Theorems

We begin with the formal definition of our augmented factor graph.

Definition 24. *Suppose we have a discrete distribution represented by a factor graph F, with variables* $\mathbf{X}$*. Then we define the corresponding augmented factor graph F_A as follows:*

- *For each X_i, we introduce variables X_i^* and S_i*
- *For each S_i, add a unary factor such that S_i is uniform*
- *For each X_i^*, add a factor such that*

$$P(X_i^*|S_i, X_i) = \begin{cases} 0.5, & \textit{if } S_i = 0 \\ 1, & \textit{if } S_i = 1, X_i = X_i^* \\ 0, & \textit{otherwise} \end{cases} \tag{B.1}$$

B.1.1 Proof of Theorem 5

Proof. Consider some variable X_i together with its corresponding augmented variables X_i^*, S_i. Then examining the resulting graphical model, we see that $X*_i$ and S_i are only connected to the rest of the model via X_i. Due to the conditional independence properties, this means that we can first sum out S_i from $P(X_i^*|S_i, X_i)$, and then simply sum X_i^*. Thus, the result of any query over $\mathbf{X}$

on F_A will be equivalent to the result on F, as we can simply sum out all additional variables, and end up at the same model. □

B.1.2 Proof of Theorem 6

NB: This section only provides the proof for the case of binary variables. When compiling graphical models with multivalued variables, it is done by assuming an indicator variable for each possible state of each variable, and then adding logical constrains such that exactly one of these indicators is true at any time. Thus, proving the binary case is sufficient.

Proof. Our goal is to estimate

$$\begin{aligned}\mathbb{E}_{P(\xi)}[f(\xi)] &= \sum_{X,X^*,S} P(X,X^*,S)f(X) \\ &= \sum_{X,X^*,S} P(X)\prod_i P(S_i)P(X_i^*|X_i,S_i)f(X)\end{aligned}$$

We begin by introducing our proposal distribution $Q(X^*,S)$, and supposing we have samples $\{X^*[m],S[m]\}_{m=1}^M$ drawn from our proposal distribution Q. Now, substituting this in and continuing our previous equations gives us:

$$\sum_{X^*[m],S[m]} \frac{1}{Q(X^*[m],S[m])}\sum_X P(X)\prod_i P(S_i[m])P(X_i^*[m])f(X).$$

We can split our sum over X up into indices where $S_i[m]=0$, and ones where $S_i[m]=1$. For the sake of notation, we will refer to these sets of indices as I^C and I respectively:

$$\begin{aligned}\sum_{m=1}^M \frac{1}{Q(X^*[m],S[m])}\sum_{X_I}\prod_{i\in I}(P(S_i[m])P(X_i^*[m]|X_i,S_i[m])) \\ \sum_{X_{I^C}} P(X)\prod_{i\in I^C} P(S_i[m])P(X_i^*[m]|X_i,S_i[m])f(X)\end{aligned}$$

$$
\begin{aligned}
&= \sum_{m=1}^{M} \frac{1}{Q(X^*[m], S[m])} \sum_{X_I} \prod_{i \in I} (P(S_i[m]) P(X_i^*[m] | X_i, S_i[m])) \\
&\qquad \sum_{X_{I^C}} P(X_I) P(X_{I^C} | X_I) \prod_{i \in I^C} (0.5) P(X_i^*[m] | X_i, S_i[m]) f(X) \\
&= \sum_{m=1}^{M} \frac{1}{Q(X^*[m], S[m])} \sum_{X_I} P(X_I) \prod_{i \in I} (P(S_i[m]) P(X_i^*[m] | X_i, S_i[m])) \\
&\qquad \sum_{X_{I^C}} P(X_{I^C} | X_I) (0.5 \cdot 0.5)^{|I^C|} f(X) \\
&= \sum_{m=1}^{M} \frac{(0.25)^{|I^C|}}{Q(X^*[m], S[m])} \sum_{X_I} P(X_I) \prod_{i \in I} (0.5) P(X_i^*[m] | X_i, S_i[m]) \sum_{X_{I^C}} P(X_{I^C} | X_I) f(X) \\
&= \sum_{m=1}^{M} \frac{(0.25)^{|I^C|}}{Q(X^*[m], S[m])} (0.5)^{|I|} \sum_{X_I} P(X_I) \prod_{i \in I} P(X_i^*[m] | X_i, S_i[m]) \mathbb{E}_{P(X_{I^C} | X_I}[f(X)]
\end{aligned}
$$

Now, observe that the term $\prod_{i \in I} P(X_i^*[m] | X_i, S_i[m]) = 1$ if and only if $\forall i$ s.t. $S_i[m] = 1$, we have that $X_i = X_i^*[m]$. Since there is only one setting of these indices that satisfies this (that is, letting $X_i = X_i^*[m]$ everywhere that $S_i[m] = 1$), this allows us to remove this sum, and obtain:

$$
\sum_{m=1}^{M} \frac{(0.25)^{I^C} (0.5)^I P_{X_I}(X_I^*[m]) \mathbb{E}_{P(X_{I^C} | X_I)}[f(X)]}{Q(X^*[m], S[m])},
$$

which is precisely what we wanted – this is the equation we would expect to use for our online collapsed importance sampler (once we adjust our proposal distribution for variables that are not actually sampled to correct for the 0.5s that are left). □

B.1.3 Proof of Theorem 7

Proof. We will proceed by contradiction. Suppose we have two paths through our variables, $X_{i_1}^*, X_{i_2}^*, \ldots, X_{i_n}^*$ and $X_{j_1}^*, X_{j_2}^*, \ldots, X_{j_n}^*$ which can produce the same assignment to all variables. Now, there are two key facts we will observe and make use of:

1. The starting point for any path is fixed, that is $X_{i_1}^* = X_{j_1}^*$. Our heuristics are deterministic,

and the memory limit remains constant, so the first variable to be sampled is always the same.

2. Once an assignment is made to the current variable being sampled, the decision of which variable to sample is deterministic – again, because our heuristics must be deterministic. To put it another way, if $x^*_{i_k} = x^*_{j_k}$, then $X^*_{i_k+1} = X^*_{j_k+1}$.

Now, since path i and path j are different, there must be some first element k where $X^*_{i_k} \neq X^*_{j_k}$. By Fact 1, $k > 1$. Also, observe that since k is the first such element, $X^*_{i_{k-1}} = X^*_{j_{k-1}}$. But since our two paths must give the same assignment to all variables, this means also that $x^*_{i_{k-1}} = x^*_{j_{k-1}}$, which means that $X^*_{i_k} = X^*_{j_k}$ by Fact 2. This is a contradiction.

□

B.2 Collapsed Compilation: Algorithm Outline

Algorithm 4 describes Collapsed Compilation in more detail. Note that to avoid confusion, we hereafter refer to the process of sampling $x \sim Q(X)$ as conditioning (since all future variables sampled are conditioned on x), and a single full run as a sample. Conditioning an SDD on instantiation x is denoted $SDD|x$.

There are a few important things to take away here. First, if we at any point interrupt bottom-up compilation, what we will get is a complete compilation of some subset of the model. This means that on Line 8, the proposal distribution P_{SDD} we are drawing from is the true distribution for X_j on some subset of graphical model M, conditioned on all previous variables.

Second, there are a few calls to a weighted model count function WMC on Line 11. Recall that for an SDD representing a probability distribution $P(\mathbf{X})$, the weighted model count subject to a function f computes $\sum_{\mathbf{x}} P(\mathbf{x}) f(\mathbf{x})$. Also, observe that the SDD we are left with when we finish compiling is representing the joint distribution $P(\mathbf{X_d}, \mathbf{X_p} = \mathbf{x_p})$. Thus, observing that $P(\mathbf{X_d}, \mathbf{X_p} = \mathbf{x_p}) = \hat{P}(\mathbf{X_d}|\mathbf{X_p} = \mathbf{x_p})$ we see that WMC_f – the weighted model count subject to f – is actually $\sum_{\mathbf{x_d}} \hat{P}(\mathbf{X_d} = \mathbf{x_d}|\mathbf{X_p} = \mathbf{x_p}) f(\mathbf{x_d}, \mathbf{x_p})$. But setting $f = 1$ allows us to compute the

Algorithm 4 A single sample of Collapsed Compilation

Input: π: A variable selection policy computed using an SDD, M: A probabilistic graphical model, f: Function to estimate expectation for

Output: A single sample $\left(\mathbf{x_p^m}[m],\ \ \mathbb{E}_{P(\mathbf{X_d^m}|\mathbf{x_p^m}[m])}[f(\mathbf{x_p^m}[m], \mathbf{X_d^m})],\ \ w[m]\right)$

$SDD \leftarrow True$
$\mathbf{x_p} \leftarrow \{\}$
$q \leftarrow 1$
while M is not compiled **do**
    $SDD \leftarrow bottomUpCompileStep(SDD, M)$
    while SDD is too large **do**
        $X \leftarrow \pi(SDD, \mathbf{x_p})$
        $x \sim P_{SDD}(X)$
        $q \leftarrow q \cdot P_{SDD}(x)$
        $\mathbf{x_p} \leftarrow \mathbf{x_p} \cup \{X = x\}$
        $SDD \leftarrow SDD|x$
return $\left(\mathbf{x_p}, \frac{WMC_f(SDD)}{WMC(SDD)}, \frac{WMC(SDD)}{q}\right)$

normalization constant, meaning that $\frac{WMC_f(SDD)}{WMC(SDD)} = \mathbb{E}_{P(\mathbf{X_d}|\mathbf{X_p}=\mathbf{x_p})}[f(\mathbf{x_p}, \mathbf{X_d})]$.

B.3 Collapsed Compilation: Algorithmic Details

There are many moving parts in this method, so in this section we will examine each in isolation.

B.3.1 Choices of Strategies

Compilation Order The main paper describes the `BFS` and `revBFS` compilation orders.

Proposal Distribution Given that we have decided to condition on a variable X_j, we decide its proposal distribution by computing the marginal probability of X_j in our currently compiled SDD. This can be done in time linear in the size of the circuit by computing the partial derivative of the weighted model count in the current circuit w.r.t. X_j [Darwiche, 2003].

Variable Selection The manner in which we select the next variable to condition on – that is, our choice of π in Algorithm 4 – has a large effect on both the tractability and efficiency of our sampler. The main paper defines three policies, the first of which depends specifically on the marginal being queried for, while the other two do not. The policies all satisfy Definition 4: they are deterministic.

B.3.2 Determinism

A desirable property for samplers – particularly when there are a large number of deterministic relationships present in the model – is to be *rejection-free*. It is clear that in the absence of deterministic factors (that is, no 0 entries in any factor), collapsed compilation will never reject samples. Here, we describe how this guarantee can be maintained in the presence of 0-valued factor entries.

Extracting a Logical Base Suppose we are given a factor over some variables $X_1, X_2..., X_n$. Then a weight of 0 given to an assignment $x_1, x_2.., x_n$ indicates a logical statement. Specifically, we can say for certain over the entire model that $\neg(x_1 \wedge x_2, ..., \wedge x_n)$. As a preprocessing step, we find all such factor entries in the graph, convert them each into the corresponding logical statement, and then take the conjunction of all of these sentences. This forms the logical base for our model.

An Oracle for Determinism Once we have obtained this base for our model, we can precompile a circuit representing it. This allows us to make queries asking whether there is a non-zero probability that $X = x$, given all previous assignments $\mathbf{x_p}$ [Darwiche, 1999]. Thus, when we sample from the marginal probability of X from our current SDD (our proposal distribution), we first renormalize this marginal to only include assignments which have a non-zero probability according to our determinism oracle. Of course, this is not always possible due to size constraints in the case where there is an enormous amount of determinism. For these cases we just run collapsed compilation as is – depending on the compilation order it will still tend to reject few samples.

Literal Entailment As a simple optimization, we can recognize any variables whose values are already deterministically chosen based on previously conditioned variables, and assign them as such in our SDD. Given a determinism oracle, deciding this can be done for all variables in the model in time linear in the size of the oracle [Darwiche, 2001].

B.4 Experimental Details

B.4.1 Edge-Deletion Belief Propagation

Edge-deletion belief propagation (EDBP) is a method for doing approximate graphical model inference by using a combination of exact inference and belief propagation [Choi and Darwiche, 2006] [Choi et al., 2005]. EDBP iteratively computes more and more of the model exactly using junction tree, at each step performing belief propagation to approximate the rest of the model. It can be viewed as the belief propagation analog of collapsed compilation, which makes it an interesting target for comparison. A major conceptual difference between the two is that while collapsed compilation is asymptotically unbiased and thus will given an accurate result given enough time, EDBP will tend to finish more quickly but has no way to improve once converged.

To capture a more direct comparison of the amount of exact inference being performed, we compare collapsed compilation to EDBP with the size of the junction tree used directly being limited, rather than the computer memory usage. In particular, we limit the size of the circuit corresponding to the junction tree to be similar to the sizes used for collapsed compilation. To this end, we use 100,000 and 1,000,000 as size limits for the junction tree, and otherwise run the algorithm as usual. Table 3.2 shows the results across all benchmarks. Keeping in mind that all policies for collapsed compilation use 100,00 as their size limit, collapsed compilation is comparable to EDBP. Both perform very well in linkage and Segment, and while collapsed compilation performs better on 50-20, EDBP does better on 75-26.

B.4.2 SampleSearch

IJGP-Samplesearch is an importance sampler augmented with constraint satisfaction search [Gogate and Dechter, 2011] [Gogate and Dechter, 2007]. It uses iterative join graph propagation [Dechter et al., 2002] together with w-cutset sampling [Bidyuk and Dechter, 2007] to form a proposal, and then uses search to ensure that no samples are rejected.

Once again, we would like to control for the amount of exact inference being done directly at the algorithmic level, rather than via computer memory. For samplesearch, we do this by limiting w, which is the largest treewidth that is allowed when using collapsing to reduce variance. We run samplesearch with three different settings, limiting w to 15, 12, and 10 respectively. Table 3.2 shows the results of running our standard set of benchmarks with all of these settings. As a reference point, empirically a circuit size limit of 100,00 generally corresponds to a treewidth somewhere between 10 and 12. The results are generally similar to constraining the memory of EDBP, but with more constrained versions of samplesearch suffering more. For example, although linkage appears to be an easy instance in general, without a large enough w-cutset, samplesearch struggles compared to other methods.

Appendix C

Constrained Open-World Probabilistic Databases

C.1 Proofs of Theorems, Lemmas, and Propositions

C.1.1 Proof of Proposition 8

Proof. To prove this, we need to show that $K_{\mathcal{G}}^{\Phi}$ is both closed and convex.

Due to the way our constraints are defined, we know that $K_{\mathcal{G}}^{\Phi} = K_{\mathcal{G}} \cap K^{\Phi}$, where K^{Φ} is the set of all completions satisfying Φ (but not necessarily having all tuple probabilities $\leq \lambda$). We already know that $K_{\mathcal{G}}$ is credal, and thus closed and convex. K^{Φ} is a half-space, which we also know is closed and convex. The intersection of closed spaces is closed, and the intersection of convex spaces is convex, so $K_{\mathcal{G}}^{\Phi}$ is credal. □

C.1.2 Proof of Theorem 9

Lemma 25. *If $S \subseteq \mathbb{R}^n$ is a set formed by the intersection of $k < n$ half-spaces, S has no points of extrema.*

Proof. Written as a set of linear equalities, the solution clearly must have at least 1 degree of freedom. This indicates that for any potential extrema point x, one can move in either direction along this degree of freedom to construct an open line intersecting x, but entirely contained in S. □

Proof. Since $K_{\mathcal{G}}^{\Phi}$ is credal, we are interested here in determining the point of extrema of $K_{\mathcal{G}}^{\Phi}$, as

this will tell us precisely which completions can represent boundaries.

Consider the construction of the set $K_{\mathcal{G}}^{\Phi}$, and suppose that there are d possible open-world tuples, meaning that $K_{\mathcal{G}}^{\Phi} \subseteq \mathbb{R}^d$. As we observed in the proof of Theorem 8, $K_{\mathcal{G}}^{\Phi} = K_{\mathcal{G}} \cap K^{\Phi}$, where K^{Φ} is the set of all completions satisfying Φ. We now make three key observations about these sets:

1. Each individual possible open-world tuple is described by the intersection of 2 half-spaces: that is, the tuple on dimension i is described by $x_i \geq 0$ and $x_i \leq \lambda$. $K_{\mathcal{G}}$ is the intersection of all $2d$ of these half-spaces.

2. For any individual open-world tuple, the boundaries of the two half-spaces that describe it cannot intersect each other.

3. An MTP constraint is a linear constraint, and thus can be described by a single half-space. So K^{Φ} is described by the intersection of these $|\Phi|$ half-spaces.

Observations 1 and 3, together with Lemma 25 tells us that any point of extrema of $K_{\mathcal{G}}^{\Phi}$ must be given by the intersection of the boundaries of at least d of the half-spaces that form $K_{\mathcal{G}}^{\Phi}$. Observation 3 tells us that at most $|\Phi|$ of these half-spaces come from MTP constraints, leaving the boundaries of at least $d - |\Phi|$ half-spaces which come from $K_{\mathcal{G}}$. Finally, observation 2 tells us that each of these $d - |\Phi|$ half-spaces is describing a different open world tuple. But this means we must have at least $d - |\Phi|$ tuples which lie on the boundary of one of their defining half-spaces: they must be either 0 or λ.

□

C.1.3 Proof of Theorem 11

Before we present the formal proof, we state and prove 2 Lemmas we will need.

Lemma 26. *Suppose we have two completions P_1 and P_2 of R, which only differ on a single triple, that is $P_1 = P_0 \cup \{x_1, y_1, z_1\}$ and $P_2 = P_0 \cup \{x_2, y_2, z_2\}$. Further suppose that $y_1 = y_2$, $z_1 = z_2$,*

and that P_0 contains no triples with x-value x_1, but does contain at least 1 triple with x-value x_2. Then $P_1(M_0) > P_2(M_0)$.

Proof. Let us directly examine Δ, the logical formula found by grounding M_0. Since M_0 is a union of conjunctive queries, Δ must be a DNF. Each disjunct either does not contain R, in which case it does not vary with the choice of completion, or it contains it exactly once. Any disjunct containing an atom of R not assigned probability by a completion is logically false.

In order to prove that $P_1(M_0) > P_2(M_0)$, let us compare the ground atoms that result from each. It is clear that the only spot on which they differ is on disjuncts involving $R(x_1, y_1, z_1)$ or $R(x_2, y_1, z_1)$. Any disjuncts involving one of these and V or W will also have an identical effect on the probability of the query, since the completions are identical over y and z.

Finally this means we need to compare the terms $R(x_1, y_1, z_1), U(x_1)$ and $R(x_2, y_1, z_1), U(x_2)$. Observe that we know P_0 contains triples with x-value x_2, which means the term only contributes new probability mass when $U(x_2)$ is true *and* none of the other triples involving x_2 are true. However, P_0 does not contain any triples with x-value x_1, so the term $R(x_1, y_1, z_1), U(x_1)$ contributes the maximum probability possible. Thus, for any choice of probabilities on U such that $U(x_2) < 1$, we have that $P_1(M_0) > P_2(M_0)$. □

Lemma 27. *The upper bound $\overline{P}(M_0)$ subject to an MTP constraint on R allowing for $k \cdot \lambda$ total probability mass is maximized if and only if P is a completion formed by a matching of size k, where k is the maximum number of tuples with probability λ that can be added to R in M_0.*

Proof. Observe that if we begin with a completion given by a matching, we can repeatedly apply Lemma 26 to arrive at any completion. Thus a completion given by a matching must have higher probability than any completion not given by a matching. □

Finally, we are ready to present the proof of Theorem 11.

Proof. Suppose we are given an instance of a 3-dimensional matching problem X, Y, Z, T and an integer k. Let $U(x), V(y), W(z)$ be 0.8 wherever $x \in X, y \in Y$, or $z \in Z$ respectively, and 0

everywhere else. Additionally, let $R(x, y, z)$ be unknown for any $(x, y, z) \in T$, and 0 otherwise. Finally, we place an MTP constraint on R ensuring that at most k tuples can be added, and let $\lambda = 0.8$. Then Lemma 27 tells us that M_0 evaluated on this database will be maximized if and only if the completion used corresponds to a matching of size k. We determine this probability P_{max} using a standard probabilistic database query algorithm, and fixing R to have entries 0.8 for some disjoint set of triples. Observe that this computation is easy if we allow any tuples from R to have non-zero probability: this corresponds to a matching in the setting where all hyperedges are available, but will still give the same query probability.

Finally, we use our oracle for MTP constrained query evaluation to check $\overline{P}(M_0)$ with the database we constructed from the matching problem. We compare the upper bound given by the oracle, and if it is equal to P_{max}, Lemma 27 tells us that a matching of size k does exist. Similarly, if the upper bound given by the oracle is lower than P_{max}, Lemma 27 tells us a matching of size k does not exist. □

C.1.4 Proof of Theorem 12

Proof. Without directly computing probabilities, let us inspect Δ, the logical formula we get by grounding Q. Q is a union of conjunctive queries, and thus Δ is a DNF. Each disjunct can contain our constrained relation R at most once due to the query not having self-joins, and any one of these disjuncts containing an atom of R not assigned any probability is logically false.

Next, to show that $S_{\mathcal{P},\mathsf{Q}}$ is submodular, let $X \subseteq Y \subseteq \mathbf{O}$, and let $x \in \mathbf{O} \setminus Y$ be given. We assign names to the following subformulas of Δ

- α (β) is the disjunction of all disjuncts of Δ which are not logically false due to missing R_B tuples in X (Y)
- γ is the disjunction of all disjuncts of Δ containing the tuple x

Additionally, since $X \subseteq Y$, we also know that $\alpha \Rightarrow \beta$. Now, we make a few observations

relating these quantities with our desired values for submodularity:

- $S_{\mathcal{P},\mathsf{Q}}(X) = P(\alpha)$
- $S_{\mathcal{P},\mathsf{Q}}(Y) = P(\beta)$
- $S_{\mathcal{P},\mathsf{Q}}(X \cup \{x\}) = P(\alpha \vee \gamma)$
- $S_{\mathcal{P},\mathsf{Q}}(Y \cup \{x\}) = P(\beta \vee \gamma)$

Finally, we have the following:

$$\begin{aligned} S_{\mathcal{P},\mathsf{Q}}(X \cup \{x\}) - S_{\mathcal{P},\mathsf{Q}}(X) &= P(\alpha \vee \gamma) - P(\alpha) \\ &= P(\neg\alpha \wedge \gamma) \\ &\geq P(\neg\beta \wedge \gamma) \\ &= P(\beta \vee \gamma) - P(\beta) \\ &= S_{\mathcal{P},\mathsf{Q}}(Y \cup \{x\}) - S_{\mathcal{P},\mathsf{Q}}(Y) \end{aligned}$$

□

C.2 General Algorithm for Inversion-Free Queries

We now present an algorithm for doing exact MTP query evaluation on inversion-free queries. Suppose that we have a probabilistic database $\mathcal{P}$, a domain T of constants denoted c, a query Q, and an MTP constraint on relation $R(x_1, x_2, \ldots, x_r)$ allowing us to add B tuples. For any $I \subseteq \{1, \ldots, r\}$, we let $A(x_{i_1}/c_{i_1}, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}, b)$ denote the upper query probability of $\mathsf{Q}(x_{i_1}/c_{i_1}, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}})$ subject to an MTP constraint allowing budget b on the relevant portion of R. That is, A tells us the highest probability we can achieve for a partial assignment given a fixed budget. Observe that we can compute all entries of A using a slight modification of Algorithm 2. This will take time polynomial in $|T|$.

Next, we impose an ordering $c_1, c_2, \ldots, c_{|T|}$ on the domain. For any $I \subseteq \{1, \ldots, r\}$, we let $D(j, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}, b)$ denote the upper query probability of

$$\bigvee_{c \in \{c_1, \ldots c_j\}} \mathsf{Q}(x_{i_1}/c, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}) \tag{C.1}$$

with a budget of b on the relevant portions of R. Then $D(|T|, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}, b)$ considers all possible substitutions in our first index, meaning we no longer need to worry about it. Doing this repeatedly would allow us to perform exact MTP query evaluation. However, D is non-trivial to compute, and cannot be done by simply modifying Algorithm 2. Instead, we observe the following recurrence:

$$\begin{aligned} &D(j+1, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}, b) \\ &\quad = \max_{k \in \{1, \ldots, b\}} 1 - D(j, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}, b-k) \\ &\cdot A(x_{i_1}/c_{j+1}, x_{i_2}/c_{i_2}, \ldots, x_{i_{|I|}}/c_{i_{|I|}}, k) \end{aligned}$$

Intuitively, this recurrence is saying that since the tuples from each fixed constant are independent of each other, we can add a new constant to our vocabulary by simply considering all combinations of budget assignments. This recurrence can be implemented efficiently, yielding a dynamic programming algorithm that runs in time polynomial in the domain size and budget.

The keen reader will now observe that the above definition and recurrence only make sense if Q immediately reaches Step 5 of Algorithm 2. While this is true, we see that Steps 0 and 1 have no effect on this recurrence, and Steps 2 and 4 correspond to multiplicative factors. For a query that reaches Step 3: inclusion-exclusion, we indeed need to construct such matrices for each sub-query. Notice that the modified algorithm would only work in the case where we can always pick a common x_i for all sub-queries to do dynamic programming on - that is, when the query is inversion-free.

Appendix D

TRACTOR

D.1 Proofs of Theorems

D.1.1 Proof of Theorem 17

Theorem 28. *Suppose that* Q' *is a UCQ consisting only of unary relations. Then* Q' *is safe.*

Proof. We prove this by showing that Algorithm 2 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 2, each CQ must have a separator. So Q' is safe.

1. For each CQ in Q', each relation can only appear once. This is because the relations are all unary, and thus any repeated relations will imply each other. For example $\exists x, y.R(x) \wedge R(y)$ is equivalent to $\exists x.R(x)$.

2. If Q' can not be be rewritten as a conjunction of UCQs, each CQ must contain a single quantified variable. This is because if a CQ does contain more than one variable, observation 1 tells us there must be a way to split the CQ, which would allow us to rewrite the UCQ as a conjunction of multiple UCQs.

These two observations tell us that if we ever reach step 5 of Algorithm 2, each CQ has only has a single variable, meaning we must have a separator. Thus Algorithm 2 never fails and Q' must be safe.

□

D.1.2 Proof of Theorem 16

Theorem 29. *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 read as $R(h, t)$).*

Proof. This is just a straightforward computation. For distmult, this computation is given by $\langle v_h, v_R, v_t \rangle = \sum_i v_h^{(i)} \cdot v_R^{(i)} \cdot v_t^{(i)}$. For TRACTOR we follow Section 5.4.1 which tells us the score is given by $P(E(h)) \cdot P(T(R)) \cdot P(E(t))$. Averaging this over mixture components gives $\frac{1}{d} \sum_i P(E_i(h)) \cdot P(T_i(R)) \cdot P(E_i(t))$. So they are identical to within a constant factor. □

Bibliography

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

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

Jessa Bekker, Jesse Davis, Arthur Choi, Adnan Darwiche, and Guy Van den Broeck. Tractable learning for complex probability queries. In *Advances in Neural Information Processing Systems*, pages 2242–2250, 2015.

Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *The VLDB Journal*, 17:243–264, 2007.

Bozhena Bidyuk and Rina Dechter. Cutset sampling for Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 28:1–48, 2007.

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

Mikhail Bilenko, Sugato Basu, and Raymond J Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *ICML*, page 11. ACM, 2004.

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

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

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

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

Craig Boutilier, Nir Friedman, Moises Goldszmidt, and Daphne Koller. Context-specific independence in Bayesian networks. In *Proceedings of UAI*, pages 115–123, 1996.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka, and Tom M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

İsmail İlkan Ceylan and Rafael Peñaloza. The bayesian description logic $\mathcal{BEL}$. In Stéphane Demri, Deepak Kapur, and Christoph Weidenbach, editors, *Automated Reasoning*, pages 480–494, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08587-6.

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

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

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

Supratik Chakraborty, Daniel J Fremont, Kuldeep S Meel, Sanjit A Seshia, and Moshe Y Vardi. Distribution-aware sampling and weighted model counting for sat. In *Proceedings of AAAI*, volume 14, pages 1722–1730, 2014.

Hei Chan and Adnan Darwiche. On the Robustness of Most Probable Explanations. In *Proceedings of UAI*, pages 63–71, Arlington, Virginia, United States, 2006. AUAI Press.

Kai-Wei Chang, Rajhans Samdani, and Dan Roth. A constrained latent variable model for coreference resolution. In *EMNLP*, 2013.

Ming-Wei Chang, Lev-Arie Ratinov, Nicholas Rizzolo, and Dan Roth. Learning and inference with constraints. In *AAAI*, pages 1513–1518, 2008.

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *JAIR*, 2008.

Mark Chavira, Adnan Darwiche, and Manfred Jaeger. Compiling relational Bayesian networks for exact inference. *IJAR*, 42(1-2):4–20, 2006.

Arthur Choi and Adnan Darwiche. An edge deletion semantics for belief propagation and its practical impact on approximation quality. In *Proceedings of AAAI*, volume 21, page 1107, 2006.

Arthur Choi and Adnan Darwiche. On relaxing determinism in arithmetic circuits. In *ICML*, 2017.

Arthur Choi, Hei Chan, and Adnan Darwiche. On Bayesian network approximation by edge deletion. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 128–135, Arlington, Virginia, United States, 2005. AUAI Press. ISBN 0-9749039-1-4.

Arthur Choi, Doga Kisa, and Adnan Darwiche. Compiling Probabilistic Graphical Models Using Sentential Decision Diagrams. In *ECSQARU*, 2013.

Arthur Choi, Guy Van den Broeck, and Adnan Darwiche. Tractable learning for structured probability spaces: A case study in learning preference distributions. In *IJCAI*, 2015.

Arthur Choi, Nazgol Tavabi, and Adnan Darwiche. Structured features in naive Bayes classification. In *AAAI*, pages 3233–3240, 2016.

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

Fábio Gagliardi Cozman. Credal networks. *Artif. Intell.*, 120:199–233, 2000.

Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16:523–544, 2004.

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

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

Adnan Darwiche. Compiling Knowledge into Decomposable Negation Normal Form. In *Proceedings of IJCAI*, 1999.

Adnan Darwiche. Decomposable negation normal form. *J. ACM*, 48:608–647, 2001.

Adnan Darwiche. A Differential Approach to Inference in Bayesian Networks. *J. ACM*, 50(3): 280–305, May 2003. ISSN 0004-5411.

Adnan Darwiche. Modeling and reasoning with Bayesian networks. Cambridge University Press, 2009.

Adnan Darwiche. SDD: A New Canonical Representation of Propositional Knowledge Bases. In *Proceedings of IJCAI*, 2011a.

Adnan Darwiche. SDD: A new canonical representation of propositional knowledge bases. In *IJCAI*, 2011b.

Adnan Darwiche. Three modern roles for logic in ai. *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2020.

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *JAIR*, 17:229–264, 2002.

Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009.

Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, 100:5–47, 2015.

Rina Dechter, Kalev Kask, and Robert Mateescu. Iterative join-graph propagation. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 128–136. Morgan Kaufmann Publishers Inc., 2002.

Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Lifted rule injection for relation embeddings. In *EMNLP*, pages 1389–1399, 2016.

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.

Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *ECCV*, volume 8689, 2014.

Michelangelo Diligenti, Marco Gori, and Claudio Sacca. Semantic-based regularization for learning and inference. *JAIR*, 244:143–165, 2017.

Ivan Donadello, Luciano Serafini, and Artur d'Avila Garcez. Logic tensor networks for semantic image interpretation. In *IJCAI*, pages 1596–1602, 2017.

Xin Luna Dong, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 601–610, 2014.

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

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *NIPS*, pages 2224–2232, 2015.

Ayse Erkan and Yasemin Altun. Semi-supervised learning via generalized maximum entropy. In *AISTATS*, volume PMLR, pages 209–216, 2010.

Stefano Ermon, Carla P Gomes, Ashish Sabharwal, and Bart Selman. Embed and project: Discrete

sampling with universal hashing. In *Advances in Neural Information Processing Systems*, pages 2085–2093, 2013.

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean formulas. *TPLP*, 15(3):358–401, 2015.

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

Tal Friedman and Guy Van den Broeck. Approximate knowledge compilation by online collapsed importance sampling. In *Advances in Neural Information Processing Systems 31 (NeurIPS)*, 2018.

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

Tal Friedman and Guy Van den Broeck. Symbolic querying of vector spaces: Probabilistic databases meets relational embeddings. In *Proceedings of the 36th Conference on Uncertainty in Aritifical Intelligence (UAI)*, 2020.

Kuzman Ganchev, Jennifer Gillenwater, Ben Taskar, et al. Posterior regularization for structured latent variable models. *JMLR*, 11(Jul):2001–2049, 2010.

Vibhav Gogate and Rina Dechter. Samplesearch: A scheme that searches for consistent samples. In *Artificial Intelligence and Statistics*, pages 147–154, 2007.

Vibhav Gogate and Rina Dechter. Samplesearch: Importance sampling in presence of determinism. *Artificial Intelligence*, 175(2):694–729, 2011.

Yves Grandvalet and Yoshua Bengio. Semi-supervised learning by entropy minimization. In *NIPS*, 2005.

Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538 (7626):471–476, 2016.

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

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

Martin Grohe and Peter Lindner. Probabilistic databases with an infinite open-world assumption. *CoRR*, abs/1807.00607, 2018.

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

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Overview of supervised learning. In *The elements of statistical learning*, pages 9–41. Springer, 2009.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, June 2016.

Jochen Heinsohn. Probabilistic description logics. In *UAI*, 1994.

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

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.

Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard Hovy, and Eric Xing. Harnessing deep neural networks with logic rules. In *ACL*, 2016.

Jiewen Huang, Lyublena Antova, Christoph Koch, and Dan Olteanu. Maybms: a probabilistic database management system. In *SIGMOD Conference*, 2009.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, pages 448–456, 2015.

Abhay Jha and Dan Suciu. Knowledge compilation meets database theory: Compiling queries to decision diagrams. *Theory of Computing Systems*, 52:403–440, 2011.

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

Douglas Samuel Jones. *Elementary information theory*. Clarendon Press, 1979.

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

Angelika Kimmig, Stephen H. Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. A short introduction to probabilistic soft logic. In *NIPS (Workshop Track)*, 2012.

Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *NIPS*, pages 3581–3589. Curran Associates, Inc., 2014.

Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *KR*, 2014.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. 2009.

Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classifcation with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, pages 1097–1105, 2012.

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

Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, 2015.

Daniel Lowd and Pedro Domingos. Approximate inference by compilation to arithmetic circuits. In *NIPS*, pages 1477–1485, 2010.

Thomas Lukasiewicz. Probabilistic description logic programs. In *ECSQARU*, 2005.

Pablo Márquez-Neila, Mathieu Salzmann, and Pascal Fua. Imposing hard constraints on deep networks: Promises and limitations. *arXiv preprint arXiv:1706.02025*, 2017.

Robert Mateescu and Rina Dechter. Mixed deterministic and probabilistic networks. *Annals of mathematics and artificial intelligence*, 54(1-3):3, 2008.

Nicholas Mattei and Toby Walsh. Preflib: A library of preference data HTTP://PREFLIB.ORG. In *ADT*, 2013.

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

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

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

Pasquale Minervini, Thomas Demeester, Tim Rocktäschel, and Sebastian Riedel. Adversarial sets for regularising neural link predictors. *arXiv preprint arXiv:1707.07596*, 2017.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. In *ICLR*, 2016.

Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *ArXiv e-prints*, 2017.

Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of UAI*, pages 467–475. Morgan Kaufmann Publishers Inc., 1999.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, pages 807–814. Omnipress, 2010.

Jason Naradowsky and Sebastian Riedel. Modeling exclusion with a differentiable factor graph constraint. In *ICML (Workshop Track)*, 2017.

D. Nardi, Werner Nutt, and Francesco M. Donini. The description logic handbook: Theory, implementation, and applications. In *Description Logic Handbook*, 2003.

Arvind Neelakantan, Benjamin Roth, and Andrew McCallum. Compositional vector space models for knowledge base inference. In *ACL-IJCNLP*, pages 156–166, 2015.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—i. *Mathematical Programming*, 14(1):265–294, Dec 1978. ISSN 1436-4646. doi: 10.1007/BF01588971. URL https://doi.org/10.1007/BF01588971.

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

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.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *ICML*, pages 2014–2023, 2016.

Masaaki Nishino, Norihito Yasuda, Shin-ichi Minato, and Masaaki Nagata. Compiling graph substructures into sentential decision diagrams. In *AAAI*, 2017.

Dan Olteanu, Jiewen Huang, and Christoph Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. *2009 IEEE 25th International Conference on Data Engineering*, pages 640–651, 2009.

Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. Constrained convolutional neural networks for weakly supervised segmentation. In *ICCV*, pages 1796–1804, 2015.

Shanan E Peters, Ce Zhang, Miron Livny, and Christopher Ré. A machine reading system for assembling synthetic paleontological databases. In *PloS one*, 2014.

Nikolaos Pitelis, Chris Russell, and Lourdes Agapito. Semi-supervised learning using an unsupervised atlas. In *ECML-PKDD*, pages 565–580. Springer, 2014.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *NIPS*, 2015.

Christopher Ré and Dan Suciu. Managing probabilistic data with mystiq: The can-do, the could-do, and the can't-do. In *SUM*, 2008.

Christopher Ré, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. *2007 IEEE 23rd International Conference on Data Engineering*, pages 886–895, 2007.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

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

Matthew Richardson and Pedro M. Domingos. Markov logic networks. *Machine Learning*, 62: 107–136, 2006.

Sebastian Riedel, Matko Bosnjak, and Tim Rocktäschel. Programming with a differentiable forth interpreter. *CoRR, abs/1605.06640*, 2016.

Fabrizio Riguzzi, Elena Bellodi, Evelina Lamma, and Riccardo Zese. Epistemic and statistical probabilistic ontologies. In *URSW*, 2012.

Fabrizio Riguzzi, Elena Bellodi, Evelina Lamma, and Riccardo Zese. Reasoning with probabilistic ontologies. In *IJCAI*, 2015.

Tim Rocktäschel, Sameer Singh, and Sebastian Riedel. Injecting logical background knowledge into embeddings for relation extraction. In *HLT-NAACL*, 2015.

Dan Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

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

Tian Sang, Paul Beame, and Henry A Kautz. Performing bayesian inference by weighted model counting. In *AAAI*, volume 5, pages 475–481, 2005.

Yujia Shen, Arthur Choi, and Adnan Darwiche. A tractable probabilistic model for subset selection. In *UAI*, 2017.

David Smith and Vibhav Gogate. Loopy belief propagation in the presence of determinism. In *AISTATS*, 2014.

Ashwin Srinivasan, S Muggleton, and RD King. Comparing the use of background knowledge by inductive logic programming systems. In *ILP*, pages 199–230, 1995.

Russell Stewart and Stefano Ermon. Label-free supervision of neural networks with physics and domain knowledge. In *AAAI*, pages 2576–2582, 2017.

Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A Core of Semantic Knowledge. In *16th International Conference on the World Wide Web*, pages 697–706, 2007.

Dan Suciu, Dan Olteanu, R. Christopher, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 1st edition, 2011. ISBN 1608456803, 9781608456802.

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

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

Surya T Tokdar and Robert E Kass. Importance sampling: a review. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):54–60, 2010.

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

Guy Van den Broeck. *Lifted Inference and Learning in Statistical Relational Models*. PhD thesis, KU Leuven, January 2013.

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

Guy Van den Broeck and Adnan Darwiche. On the role of canonicity in knowledge compilation. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI)*, 2015.

Guy Van den Broeck and Dan Suciu. *Query Processing on Probabilistic Data: A Survey*. Foundations and Trends in Databases. Now Publishers, 2017. doi: 10.1561/1900000052.

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

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, and Luc De Raedt. Anytime inference in probabilistic logic programs with Tp-compilation. In *Proceedings of IJCAI*, pages 1852–1858, July 2015.

Jonas Vlasselaer, Wannes Meert, Guy Van den Broeck, and Luc De Raedt. Exploiting local and repeated structure in dynamic Bayesian networks. *Artificial Intelligence*, 232:43 – 53, March 2016. ISSN 0004-3702. doi: 10.1016/j.artint.2015.12.001.

Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. Premise selection for theorem proving by deep graph embedding. *arXiv preprint arXiv:1709.09994*, 2017.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Van den Broeck. A semantic loss function for deep learning with symbolic knowledge. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

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, 2014a.

Min-Chul Yang, Nan Duan, Ming Zhou, and Hae-Chang Rim. Joint relational embeddings for knowledge-based question answering. In *EMNLP*, 2014b.