# Probabilistic Database System

Zeel Doshi: zeeldoshi@cs.ucla.edu (UID: 905220399)
Vishwa Karia: vishwakaria2@cs.ucla.edu (UID: 705223110)
Vidhu Malik: vidhu26@ucla.edu (UID:305225272)
Vaishnavi Pendse: vaishnavipendse@cs.ucla.edu (UID:005226683)

December 2018

## 1 Introduction

In this report, we will discuss our implementation approach for developing a probabilistic database system. The system takes as input a fully quantified query in first-order logic and efficiently computes the probability of the input query. Our probabilistic database system is capable of evaluating both safe queries as well as hard queries. The evaluation of hard queries, that is, those queries which cannot be solved in PTIME, is a part of our project's extension module.

Upon reading the input query from a file, the system first tries to apply the lifted inference rules [1] on it. If the lifted inference algorithm throws an exception or returns saying that the query is unliftable, then the input query is #-P hard and we execute different approximation algorithms to evaluate the hard query and compare their results. Otherwise, the algorithm evaluates the safe query and gives a probability value as the output.

## 2 Implementation Specifics

Python v3.6 is our preferred programming language throughout the project.

### 2.1 Design Decisions and Challenges

The following sub-sections discuss the challenges that we faced while implementing the problem and the design decisions that we made to handle these challenges.

#### 2.1.1 Data Structures

Some of the most challenging design decisions that were required to be made were with respect to the data structure that is used to represent the query. Our initial data structure for representing the CNF was as follows:

1. List of Clauses
2. Each Clause is represented by a python dictionary where each key-value pair represents relations within the Clause
3. The keys are names of relations and the value is represented with another dictionary that has two kinds of attributes:
   (a) List of variables
   (b) Boolean variable indicating whether the relation is negated or not

The above data structure seemed to work well for simple queries. However, we found several shortcomings when it came to complex queries. There was a possibility that multiple relations have the same relation name but different variables within a single clause (as given in Example 2 of the project description). Our initial data structure could not handle this because a clause is represented with a

dictionary and a dictionary can have only unique keys, hence only unique relation names could be supported. Secondly, another attribute was required within the dictionary that represents a single variable. This new attribute signified whether all the variables in a relation have been substituted with a constant value or not. This was required to correctly detect when the code has reached the base of recursion step, where to find the probability of a relation, all variables need to have a constant value assigned to their variables. To accommodate the above changes, our final data structure was designed with an extra nesting level to represent the CNF as follows:

1. List of Clauses
2. Each Clause is represented by a list of relations within the Clause.
3. Each relation is also represented by a list where the first element of the list is a relation name and second element is a dictionary defining its attributes.
4. The attributes for every relation are present as follows:
   (a) List of variables
   (b) Boolean variable indicating whether the relation is negated or not
   (c) Boolean variable indicating whether all the variables are replaced with constants or not

Example of a CNF:

[ [ ['R', 'var': ['x1'], 'negation': True, 'const': False], ['S', 'var': ['x1', 'y1'], 'negation': True, 'const': False], ['S', 'var': ['x2', 'y2'], 'negation': True, 'const': False], ['T', 'var': ['x2'], 'negation': True, 'const': False] ] ]

A UCNF has been represented with a list where each item in the list is the above defined data structure. Example of a UCNF:

[ [ [ ['R', 'var': ['x1'], 'negation': True, 'const': True], ['S', 'var': ['x1', 'y1'], 'negation': True, 'const': True] ], [ ['S', 'var': ['x2', 'y2'], 'negation': True, 'const': True], ['T', 'var': ['x2'], 'negation': True, 'const': True] ] ] ]

Finally, the tables representing each of the relations and probabilities are represented as a dictionary with key being table name and value being a list of rows. Each row is represented as a list where the first element is is a list of values for variables and second element is the probability. Example:
{'S': [[[0, 0], 0.7], [[0, 1], 0.4], [[1, 0], 0.3]], 'R': [[[0], 0.7], [[1], 0.4], [[2], 0.9]],'T':[[[0],0.2],[[1],0.4],[[2],0.7]]}

#### 2.1.2 Quantification

The second important design decision that was required was the handling of quantifiers. Since the input query was existentially quantified and the query that the lifted inference algorithm handles is universally quantified, the conversion was required to be made. We initially started designing a conversion function and the our query data structure represented a DNF which was to be converted to a CNF represented by another data structure. But, we soon realized that this was significantly increasing the complexity of the problem. A better approach to the problem was used. Instead of maintaining a DNF and a CNF, the query was always directly parsed to a CNF. This was implemented by always negating every variable and representing every disjunction with a conjunction and a conjunction with a disjunction. Finally, the probability that was returned by lifted inference was subtracted from 1, and this was returned as the final answer. The intuition for

the above changes was drawn from simple DeMorgan Laws.

#### 2.1.3 Query cancellations

Under certain scenarios of the query, when applying inclusion-exclusion, we might obtain queries that are of the form:
$[R(t) \wedge T(t)] \vee [\forall y_1 S(t, y_1) \wedge R(t)] \vee [\forall y_1 S(t, y_1) \wedge T(t)] \vee [\forall y_1 \forall y_2 S(t, y_1) \wedge S(t, y_2)]$
These queries need to be reduced to the form:
$[R(t) \wedge T(t)] \vee [\forall y S(t, y)]$

These cancellations are performed under the following conditions: if the length of the UCNF returned = 4, then we first look for terms that have the same table in every clause like the one in $[\forall y_1 \forall y_2 S(t, y_1) \wedge S(t, y_2)]$. This is reduced by removing one of the clauses and unifying it.

Secondly we keep track of the table name that was unified. In this case it is S. We further go ahead and scan the UCNF and ensure that all CNF's in the UCNF that have this relation are removed.

This kind of cancellation is limited and will only work on queries of the above form (or in other words, queries that need to be reudced from 4CNFs to 2CNFs). Following the cancellation, the usual lifted inference algorithm is applied.

#### 2.1.4 Conversion to UCQ

One of the most challenging tasks of this project was the correct conversion to UCQ for all types of queries. The UCQ data structure that we were using was nested till 5 levels and hence made it quite difficult for us to manage it and correctly handle the representation of the UCQ. This part took up a very significant amount of time in the development of the project. Finally, after the optimization of the data structure and designing of multiple utility functions, this task was completed.

### 2.2 Validation

To confirm the correctness of our code, we also validated it by cross-checking the values against the values computed manually.

The queries that were verified are as follows:

1. $Pr(R(x, y) \wedge P(x)) =$ Liftable
2. $Pr(R(x, y) \wedge P(z)) =$ Liftable
3. $Pr(R(x, y) \wedge P(x) \wedge Q(z) \wedge R(z, f)) =$ Liftable
4. $Pr(T(x, z, z) \wedge R(x, z) \wedge P(z)) =$ Liftable
5. $Pr((P(x) \wedge R(x, y)) \vee (Q(z) \wedge R(x, z))) =$ Unliftable
6. $Pr(R(x, y) \wedge P(y) \wedge R(z, y) \wedge Q(y)) =$ Unliftable
7. $Pr(T(x, y, z) \wedge S(x, y, w)) =$ Cannot be Handled because the system cannot obtain the value for $\neg Pr(T(x, y, z) \wedge \neg S(x, y, w)) =$ when applying inclusion-exclusion
8. $Pr(P(x1) \vee S(x1, y2) \vee S(x2, y2) \vee Q(x2) \vee P(x3) \vee S(x2, y1)) =$ Cannot be handled because the system does not support UCNF of size greater than 2 when performing inclusion-exclusion
9. $[Pr(P(x) \wedge R(x, y) \wedge S(x, z)] \vee [P(x) \wedge T(y, x, z)] =$ Cannot be handled because the system doesn't support inclusion-exclusion and cancellation when the initial UCNF is of length greater than 2.

## 3 Extension

### 3.1 Problems Addressed

In the first part of our project, we have implemented the Lifted Inference Algorithm for the evaluation of queries on Probabilistic Databases. However, this algorithm cannot process queries which are non-hierarchical. This is because the problem of computing the probability of a non-hierarchical query is #P-hard in the size of the database. Therefore, as an extension to the first part, we implemented different approximation schemes like Monte Carlo sampling and Markov Chain Monte Carlo (MCMC) methods such as Metropolis-Hastings and Gibbs Sampling for evaluating hard queries and compared their results.

### 3.2 Description

The first technique we used for approximation was the Monte Carlo Algorithm. It takes a Boolean Formula $F$ and a tuple-independent probabilistic database $D$. The algorithm repeatedly chooses a random assignment $\theta$, with probability $\Pr(\theta)$ for each tuple, thereby creating a concrete database for each trial. The final probability is equal to the fraction of trials where $\theta(F) = True$ [2].

We then implemented two MCMC methods-Metropolis-Hastings and Gibbs Sampling to draw samples from the probabilistic database. Both the methods proceed in a similar manner except for a few differences. Metropolis-Hastings proceeds by randomly attempting to move about the sample space based on a Jumping Distribution, sometimes accepting the moves and sometimes remaining in place. For Gibbs Sampling, the next sample is based on the distribution of that component conditioned on all other components sampled so far [3][4].

### 3.3 Technical Details

We ran all the three algorithms on the input query to evaluate and compare the results for different number of samples. Experiments suggested that the probability value stabilizes after approximately 20,000 samples each, fewer samples sometimes provided unstable results for some types of queries. We have used a Gaussian Distribution as the probability density function for the MCMC Methods, that suggests a candidate for the next sample value $x$, given the previous sample value $y$.

### 3.4 Validation

In order to validate the results we obtained on hard queries, we executed our algorithms for safe queries whose results could be correctly obtained from the Lifted Inference Algorithm. For these safe queries, we plotted the error rate against the number of samples. Fig. 1 is the error plot for Monte Carlo sampling and Fig. 2 is that for Metropolis-Hastings sampling. One shortcoming that we identified was that in Gibbs sampling. When we compared the value

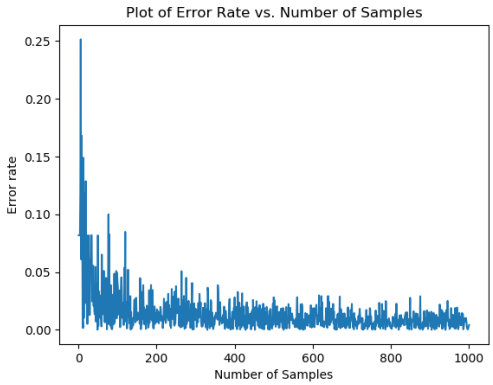


Figure 1: Plot of Error Rate vs. Number of Samples for Monte Carlo

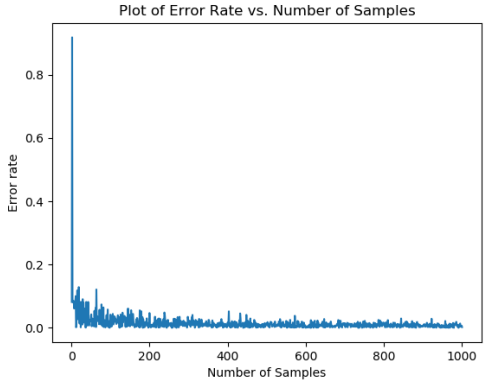


Figure 2: Plot of Error Rate vs. Number of Samples for Metropolis-Hastings

obtained from Gibbs sampling with that from lifted inference, we obtained an error rate of $\pm 0.25$, which is not very accurate. We believe that this inaccuracy resulted because we were not able to properly derive conditional probability, which is an essential parameter for Gibbs. In our future work, we intend to fix this.

## References

[1] Van den Broeck, Guy and Suciu, Dan. Query Processing on Prob-

abilistic Data: A Survey. 2017. http://web.cs.ucla.edu/guyvdb/papers/VdB-FTDB17.pdf

[2] Vudepu, Vamshi Krishna. Monte Carlo Method. 2012. http://community.wvu.edu/ krsubramani/courses/sp12/rand/lecnotes/MonteCarlo1.pdf

[3] Wick, Michael, Andrew McCallum, and Gerome Miklau. "Scalable probabilistic databases with factor graphs and MCMC." Proceedings of the VLDB Endowment 3, no. 1-2 (2010): 794-804.

[4] Breheny, Patric. MCMC Methods: Gibbs and Metropolis. https://web.as.uky.edu/statistics/users/pbreheny/701/S13/notes/2-28.pdf