Probabilistic Database System

Zeel Doshi: zeeldoshi@cs.ucla.edu
Vishwa Karia: vishwakaria2@ucla.edu
Vidhu Malik: vidhu26@ucla.edu
Vaishnavi Pendse: vaishnavipendse@cs.ucla.edu

November 17, 2018

1 Extension Proposed

1.1 Motivation

For non-hierarchical queries, the Lifted Inference Algorithm fails and cannot be applied as the problem of model counting for these queries is #P-hard. Thus, we intend to implement an approximation scheme for hard queries using Markov Chain Monte Carlo methods as an extension to our project.

1.2 Overview

Markov chain Monte Carlo (MCMC) methods comprise a class of algorithms that are used to approximate the posterior distribution of a random variable by sampling from a probability distribution. By constructing a Markov chain that has the desired distribution as its equilibrium distribution, we can obtain a sample of the desired distribution by observing the chain after a number of steps. Monte Carlo simulations are a way of estimating a fixed parameter by repeatedly generating random numbers. By taking the random numbers generated and doing some computation on them, Monte Carlo simulations provide an approximation of a parameter where calculating it directly is impossible or prohibitively expensive. Using MCMC methods provides an efficient evaluation for any arbitrary query.

1.3 Implementation Specifics

We would make a series of modifications to the worlds. The basic idea is that MCMC can hypothesize changes to the single underlying possible worlds by proposing modifications to previous worlds. A world w1 is a result of a small modification to the original world w0. We will execute queries over hypothesized worlds and compute the marginal probabilities. Instead of running the original query over each consecutive sample, we will run the query only once on the initial world and then for each consequent sample, we will run a modified query over the difference. The probabilities obtained during sampling would eventually converge. The stopping condition would be the difference between consecutive samples being small.

2 Progress Report

2.1 Roadblocks Encountered

Before we could begin with developing the lifted inference algorithm, it was necessary to design the initial layout of the project. The first step of this design was designing the data structure that would represent the query. The data structure needed to represent the following aspects:

- The individual clauses and atoms in the query
- The atoms within a clause
- The relation that is being applied to every literal
- Quantifiers that apply to every literal

After a session of brainstorming, we have decided to represent the query as a list of clauses. Each clause

is further represented by a list of relations. Each relation is represented by the dictionary that holds data regarding the quantifiers, variables and independence within this relation.

2.2 Roadblocks Anticipated

During the initial design discussion we also realised the other potential issue we will need to work over:

- Evaluating if two relations are independent for every required pair of relations is a complex task, and will require an efficient algorithm for accomplishing this
- Detecting separator variables, their removal and consequently the removal of the universal quantifier, is yet another task that would require careful design consideration
- The biggest challenge in this task would be to find a balance between the complexity and efficiency of the problem. While the algorithm needs to be fast and needs to be optimised to achieve this, it also necessary to ensure that the algorithm is not made excessively complex in this process.

2.3 Implementation Details

2.3.1 Extraction and Analysis of Knowledge Base

We will be using NELLs (Never Ending Language Learners) knowledge base to implement our system. This is because, apart from having tuples, this knowledge base also has a confidence value associated with each tuple, which is essentially the probability of that tuple being true. Since the confidence values are on a scale of 0 to 100, we performed an initial pre-processing to bring scale them down within the range [0,1].

2.3.2 Proposed approach to parse the query

For implementation purposes, the query to the database will be stored in the file. At this point, we have finalized the process of extracting a UCQ from the file and parse it using Python. The Abstract Data Type that would best represent a UCQ $[R(x) \wedge C(x,y)] \vee \neg[S(x) \wedge H(x)]$ is as follows:

$$[[0, [R, [x]], [C, [x, y]]], [1, [S, [x]], [H[x]]]]$$

where 0 means that the clause is not negated, and 1 means that the clause is negated. Upon extraction, each clause in the query will be stored in the list. The relation between different clauses in the query will be represented in the form of a dictionary. Information regarding quantifiers (either universal or existential) and whether the query is negated or not after applying a single lifted inference rule will be stored.

2.3.3 Pseudo-code design

For computing the probabilities of the query, we will be implementing lifted inference rules in Python. So far, we have successfully designed the pseudo-codes for the following rules:

- Decomposable Disjunction
- Inclusion-Exclusion
- Decomposable Conjunction
- Decomposable Universal Quantifier

References

[1] Michael Wick, Andrew McCallum, Gerome Miklau. Probabilistic Databases with Factor Graphs and MCMC.

[2] Michael Wick, Andrew McCallum. Query-Aware MCMC.

[3] Daniel Deutch, Ohad Greenshpan, Boris Kostenko, Tova Milo. Using Markov Chain Monte Carlo to Play Trivia.

[4] Ben Shaver *https://towardsdatascience.com/a-zero-math-introduction-to-markov-chain-monte-carlo-methods-dcba889e0c50*

[5] Probabilistic Inference using Markov Chain Monte Carlo Methods. Radford M. Neal