CS267A: Probabilistic Programming and Relational Learning
Fall 2018
Project Final Report: Probabilistic Database System

Xufan Wang *xufanwang@cs.ucla.edu*
Jiachen Zhong *julightzhong10@cs.ucla.edu*
Zixiang Liu *zliu46@g.ucla.edu*
Weijia Yuan *weijia@cs.ucla.edu*

December 14, 2018

1 Introduction

In this project, we implemented a probabilistic database system as described in the project specification. A probabilistic database is a database representing a world in first logic, while each ground atom is assigned to a number in $[0, 1]$ indicating its probability to be true. Under closed-world assumption, we assume that any ground atom that is not in the database has probability 0 to be true. The database system takes a probabilistic database and a UCQ as input. It then stores the probabilistic database in an MySQL database and evaluates the probability of the query over the given database using lifted inference [Van den Broeck et al., 2017], Gibbs sampling or brute force algorithm.
While lifted algorithm is required for the project, MySQL storage, Gibbs sampling and brute force algorithm are three extensions we chose to implement.

2 Usage

The environmental requirements for this project are Python 3.7, MySQL 8.0, and MySQL Connector/Python. First, unzip package and go to the unzipped directory. Before running, please first open config.py and change the user name, password and host name for your local MySQL database system, and change dbname to a database for the program to put its data. The program will change a database setting to enable file read/write, so please ensure that your account is authorized to do so. Alternatively, you can change the database setting by yourself, and delete the line in database.py that change the setting:

```
self.cursor.execute(
    "SET GLOBAL local_infile = true")
```

After configuration, please run the program using the following command:

```
python ./run.py --query <query-file>
--table <table-file-1>
[--table <table-file-2>
--table <table-file-3> ...]
```

Where the format of query file and table files are the same as those described in the project description. For more detail instructions, please find in the *README.txt* file.

2.1 Example Run

```
$ python ./run.py
--query ./query_test/test0.txt
--table ./tP.txt --table ./tQ.txt
--table ./tR.txt
  Initializing MySQL Database...
  Creating table from ./tP.txt
```

```
Creating table from ./tQ.txt
Creating table from ./tR.txt
The query is liftable.
result: [[('Q', ['x'])]] : 0.8950
What do you what to do next?
 [0]Exit   [1]Gibbs Sampling
 [2]Brute Force
1
result: [[('Q', ['x'])]] : 0.8760
```

3 Implementation

The database system stores the input tables in a MySQL database, and then fetches data from it.A parser is applied to convert the input UCQ into a list of CNFs. The system then attempts to use lifted inference algorithm. If the query is unliftable, then the system will offer options to use brute force or Gibbs sampling to evaluate the query. Those solutions are sub-optimal because brute force is expensive to perform on a large database, and Gibbs sampling gives an approximation rather than an exact solution.

3.1 Parser

For one input query, view each '||', parenthesis, comma and concatenation of characters without '||', parenthesis or comma in between as a token, we tokenize given query into a list of tokens. Then we loop through the list to construct a syntax tree. The UCQ is stored as a list, within which is each element is another list, representing each CNF clause. The second level CNF list stores a list of tuples, each represents a predicate. In the tuple, the first element is the predicate name in string and the second element is a list of all variables in string. Table name is not limited to a single capital letter and variable name can start with either upper- or lower-case letter. This parser handles a more general language than the given one.

3.2 Lifted Inference

The lifted inference algorithm first takes input from the parser and converts the input query to a suitable format. For every parsed atom in the query, it appends a list of hashtags in the end of the atom. The number of hashtags is equal to the number of variables used in the atom. Doing so helps us distinguish grounded atoms.
The algorithm lifts a query with 9 steps:

1. Eliminate repeated or redundant atoms in the query, where repeated atoms are atoms that looks exactly the same, and redundant atoms are atoms that represent the same relation on different variables. For example, in $Q(x1), Q(x2)$, the two atoms are redundant.
2. Return probability if the query contains only one grounded atom.
3. If the input is a decomposible disjunction of UCQs, then decompose it and use lift rules to calculate probabilities for each part. Put the probabilities together and return.
4. If input is UCQ and has separator(s), then instantiate one separator with different values and lift the resulting queries. Put the probabilities for those queries together and return.
5. If the input is UCQ and it was not lifted in the previous steps, then every clause in the UCQ are dependent. Apply exclusion rule on the whole UCQ.
6. From now on, the input query can only be CNF because we have already solved the UCQ cases. If the input CNF is decomposible, then decompose it and return.
7. If the CNF has separator(s), then instantiate one separator with different possible values, lift the resulting queries, put the probabilities together and return.
8. If the input CNF is hierarchical, then divide the CNF into groups of CNFs where every two distinct groups have disjoint set of variables. Apply inclusion rules on those groups. In this step, we need to calculate the probabilities of every possible disjunction of those groups. We will try

to simplify the combinations using first-order-logic rules. If there is one combination that we fail to simplify, then we fail the algorithm. Otherwise, return the result.

9. Fail.

3.3 Grounding

The grounding of each query is used in both brutal force algorithm and Gibbs sampling. Because each CNF clause in UCQ always use existential qualifier for all variables, grounded UCQ will be all CNFs with different assignments unioned together. For simplicity, output form of grounding remains the same as the syntax tree data structure. The first step is then to acquire the domain of each variables. The intersection of the variables domain in each list represents the domains of variables, which is stored as a dictionary with key as the variable name in string and value as a list of all possible values of it. With domain, we can easily generate all possible combinations of assignments. There are assignments in this combination that are not valid in predicate tables with multiple variables, that is why the next step is to loop through all assignments and remove ones not recorded in tables. Finally, we append CNFs with each assignment to a list to form the return value.

3.4 Gibbs Sampling

Gibbs Sampling acquires one sample at first, and generate the next sample iteratively using the currently variables. However, each predicate after grounding is independent of any other predicates, which means Gibbs Sampling in this case is simply Direct Sampling. Direct Sampling, in each iteration, takes a random value of each predicate and uses a SAT solver to test satisfaction. Sum all success samples and divide by total samples to get the possibility.

3.5 Brute Force

The brute force algorithm is a backup solution when the query is not liftable. In brute force, we use model counting to calculate the solution to a propositional logic formula. The logic of brute force includes query grounding, DPLL recursive search algorithm and DNF-SAT algorithm. The main function of brute force will first read tables, and then turn them into the right format that can be recognized by the following algorithm. Separated table will be merged into one large table which takes table name as keys and a little dictionary contains probability as values. Then it will get grounding from the Grounding class and send grounded query and table dictionary to the DPLL algorithm.
The DPLL algorithm is a backtracking algorithm runs by recursively choosing different literals and assigning different value to them. Every time when a literal is assigned, DPLL will check if the query is still satisfiable. If it's satisfiable in the end, it will calculate the probability of this query and add this value to an accumulator. DPLL function will return this accumulator as the final result of brute force.
Because the parser and grounding modules will turn all possible grounding into the form of DNF, this module does not take care of inputs in other forms.

3.6 Database

The database module connects MySQL database system to our program. On initialization, it cleans the target database in MySQL system. Tables in the given format can be added to the database by the createTable function. It also keeps track of all table names in memory.
According to the methods for data access used in this project, this module offers three ways to fetch data from the database. The first way is to get every record from one table. The second way is to find out all possible groundings for a specific variable in a query. The last is to get all the probability(s) related to a constant from a table.

4 Difficulties

We started our implementation early, so When implementing lifted inference rules, we decided to take a different approach than what is described in later posted instructions. This approach resembles the one we used when calculating lifted rules by hand. Thus we needed to spend more time testing different cases to ensure its correctness.
We found that redundant relations will break the lifted inference algorithm, because in these cases we cannot distinguish hierarchical queries using the common method. We solved this problem by eliminating redundancy before applying other rules.

5 Results

5.1 Correctness

To test the correctness of the system, we constructed more than 50 test queries and compared the results between Lifted Inference, Gibbs Sampling and Brute Force. The result of running these test queries is in the table in Appendix B.
In all liftable test cases, lifted inference gives the same result as brute force. In addition, Gibbs sampling with 10000 iterations always has less than 1 percent error compared to results from brute force. Since it is less likely that the three modules produces the same wrong results, we conclude that our program are correct.
Since Gibbs Sampling is an approximation, we are interested in its error rate. On a database with 100 variables and random probability for each row, we changed the size of iterations from 10 to 1000 and measured the MAE of the sampling algorithm on 5 simple queries. As shown in Figure 1 in Appendix A, though the results are fluctuating, there is a tendency that the more iterations we use, the less error we get. Also, the error rate is always less than 5 percent.

5.2 Performance

We measured the run time of evaluating 5 simple queries for the lifted inference algorithm and the Gibbs sampling algorithm on different database sizes with randomly generated records. As shown in Figure 2 in Appendix A, the run time seems to be linearly related to database size for both algorithms. In addition thee number of iterations is linearly related to its run time.

6 Conclusion

In conclusion, we meet the project requirement by handling liftable cases by the lifted inference rules. In addition, our project can handle every sentence in UCQ using brute force or Gibbs sampling. We also investigated several special queries to ensure that our project can handle as many cases as possible. We also observed that a negative correlation between number of iterations and MAE for Gibbs sampling, and a positive correlation between database size and run time for lifted inference and Gibbs sampling.
One problem for this project is that including MySQL into our implementation does not seem to improve our efficiency. The reason is that our implementation depends heavily on memory storage and fails to use the interfaces provided by MySQL. If more time is provided, we may be able to refactor our code to better incorporate MySQL and get better performance.

7 Feedback

It was a great project. Our team worked together and produced some meaningful results. However, it could be better if the project specification could have been posted before we have done some unnecessary works, such as designing our own grammar and building database interfaces.

References

[Van den Broeck et al., 2017] Van den Broeck, G., Suciu, D., et al. (2017). Query processing on probabilistic data: A survey. *Foundations and Trends® in Databases*, 7(3-4):197–341.

A Appendix: Figures

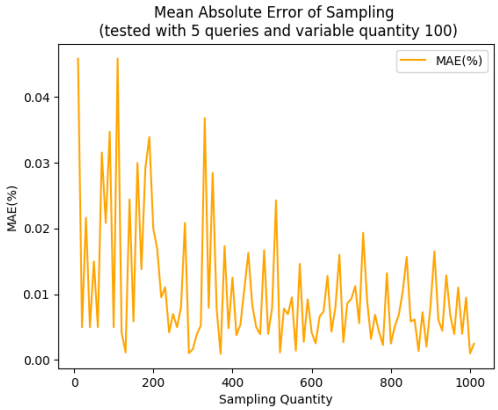

Figure 1: Sampling MAE vs. number of iterations of sampling.

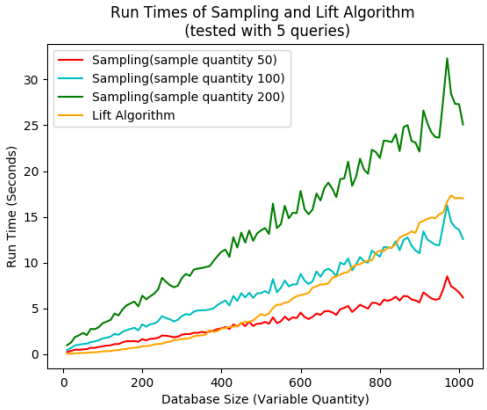

Figure 2: Run time vs. Database size for lifted inference and sampling algorithms.

B Appendix: Test Queries

Query	Lift	Sampling	BF
Q(x)	0.8950	0.8924	0.8950
P(x)	0.9760	0.9771	0.9760
R(x,x)	0.9800	0.9794	0.9800
R(x,y)	0.9976	0.9974	0.9976
R(x,y), Q(x)	0.8458	0.8456	0.8458
R(x,y), Q(y)	0.8025	0.7981	0.8025
R(x,x), Q(x)	0.7580	0.7618	0.7580
R(x,y), P(x)	0.9182	0.9181	0.9182
R(x,y), P(y)	0.8767	0.8701	0.8767
R(x,x), P(x)	0.7976	0.7917	0.7976
R(x,x), P(x) , Q(x)	0.5562	0.5667	0.5562
R(y,x), P(y) , Q(y)	0.6629	0.6637	0.6629
R(y,x), P(x) , Q(x)	0.6120	0.6138	0.6120
R(y,x), P(x) , Q(z)	0.7847	0.7807	0.7847
R(y,x), P(y) , Q(z)	0.8218	0.8182	0.8218
R(y,x), P(z) , Q(y)	0.8255	0.8169	0.8255
R(x,x), P(y) , Q(z)	0.8560	0.8531	0.8560
R(x,x), P(x) ||Q(y)	0.9787	0.9798	0.9787
R(x,z), P(z) ||Q(y)	0.9871	0.9867	0.9871

Query	Lift	Sampling	BF	**Query**	Lift	Sampling	BF
R(x,z), P(x) ||Q(y)	0.9914	0.9911	0.9914	R(x,x), P(x) || Q(y), P(z)	0.9573	0.9541	0.9573
Q(y) ||R(x,z), P(x)	0.9914	0.9925	0.9914				
R(x,x), Q(x) ||P(y)	0.9942	0.9943	0.9942	R(x1, y1), P(x1), Q(x2), R(x2, y2), Q(x3), R(x3, y3)	0.7835	0.7823	0.7835
R(x,z), Q(z) ||P(y)	0.9953	0.9956	0.9953				
R(x,z), Q(x) ||P(y)	0.9963	0.9958	0.9963	R(x1, y1), Q(x1), P(x2), R(x2, y2), Q(x3), R(x3, y3)	0.7835	0.7838	0.7835
P(y) ||R(x,z), Q(x)	0.9963	0.9968	0.9963				
R(x,x) ||Q(y), P(z)	0.9975	0.9977	0.9975	Q(x1), R(x1, y1), P(x2), Q(x3), R(x2, y2), R(x3, y3)	0.7835	0.7915	0.7835
R(x,a) ||Q(y), P(z)	0.9997	0.9998	0.9997				
R(x,a) ||Q(y), P(y)	0.9993	0.9997	0.9993	Q(x1),R(x1, y1), P(x2), Q(x3), R(x2, y2),R(x3, y3)	0.7835	0.7783	0.7835
Q(y), P(y) ||R(x,a)	0.9993	0.9995	0.9993				
Q(z)||P(y) ||R(x,x)	0.9999	1.0000	0.9999	R(x1, y1), P(x1), R(x2, y2),R(x6, y6), Q(x2), Q(x4), Q(x5)	0.7835	0.7924	0.7835
Q(z)||P(y) ||R(x,a)	1.0000	0.9999	1.0000				
P(y)||Q(z) ||R(x,a)	1.0000	1.0000	1.0000	R(x1, y1), P(x1)|| R(x2, y2),P(x2)|| R(x3, y3), Q(x3)	0.9804	0.9807	0.9804
R(x,a)||P(y)||Q(z)	1.0000	0.9999	1.0000				
R(x,y), Q(x), P(y)	unliftable	0.6841	0.6867	R(x1, y1), P(x1)|| R(x2, y2),P(x2)|| R(x3, y3), P(x3)	0.9182	0.9150	0.9182
R(x,y),R(a,b)	0.9976	0.9974	0.9976				
R(x,y) ||R(a,b)	0.9976	0.9977	0.9976	R(x1, y1), P(x1)|| R(x2, y2),Q(x2)|| R(x3, y3), Q(x3)	0.9804	0.9815	0.9804
R(x1, y1), P(x1), Q(x2), R(x2, y2)	0.7835	0.7858	0.7835	R(x1, y1), Q(x1)|| R(x2, y2),P(x2)|| R(x3, y3), Q(x3)	0.9804	0.9815	0.9804
R(x1, y1), P(x1)|| Q(x2), R(x2, y2)	0.9804	0.9798	0.9804				
R(x, y), P(x), Q(x), R(x, y)	0.6629	0.6623	0.6629	R(x1, y1), Q(x1)|| R(x2, y2),Q(x2)|| R(x3, y3), P(x3)	0.9804	0.9823	0.9804

Query	Lift	Sampling	BF
Q(x1),R(x1, y1)|| R(x2, y2),Q(x2)|| R(x3, y3), P(x3)	0.9804	0.9825	0.9804
Q(x1), R(x1, y1)|| Q(x2), R(x2, y2)|| R(x3, y3), P(x3)	0.9804	0.9777	0.9804
Q(x1),R(x1, y1)|| Q(x2), R(x2, y2)|| P(x3),R(x3, y3)	0.9804	0.9825	0.9804
Q(x1),R(x1, y1)|| Q(x1), R(x1, y1)|| P(x1),R(x1, y1)	0.9804	0.9801	0.9804
R(x1,y1), Q(x1)|| R(x2,y2), P(y2)	unliftable	0.9714	0.9741
R(x,y), Q(x)|| R(x,y), P(y)	unliftable	0.9764	0.9741
R(x1, y1), P(x1)|| R(x2, y2), R(x6, y6)|| Q(x2), Q(x4), Q(x5)	0.9997	0.9997	0.9997
R(x1,y1), Q(x1), R(x2,y2), P(y2)	unliftable	0.7466	0.7483