Report for < Probabilistic Datalog System>

Prepared by group 3

Chenwei Song 40024460

Zhongxu Huang 40052560

Luguang Liu 40040721

1. Introduction

In this project, our team aims to design, implement, and test a running prototype, called ProbLog, for probabilistic datalog system. The system could support both naïve and semi-naïve evaluation methods, in order to allow measure and compare the efficiency of the naïve and semi-naïve methods. The syntax of a ProbLog rule and fact are same as syntax in standard datalog, which is no negation, no built-in predicates, and no function symbols. Moreover, each rule and fact are associated with a number value which range is (0, 1]. The value means the probability of rule and fact.

1.1 Purpose

The purpose of this document is to introduce how the system works, record and compare the efficiency of the naïve and semi-naïve method and report all new findings such as run time improvement, Fibonacci sequence. In our program, which have two main functions: parser and evaluation engine. Specifically, evaluation engine has two method of evaluation: naïve method and semi-naïve method. The performance of Program from five respects: error handling, correctness, scalability, run-time and space time.

1.2 Document Conventions

When writing this report for probabilistic datalog system, the following terminologies are used:

- File.pl: a database to storage all rules and facts as input content
- File.cf: a configuration file which stores default functions and certainty value
- File.log: a result for naïve or semi-naïve method
- p, q: predicate
- X, Y, Z: variable
- v_1, v_2: probability of rule and fact
- $a_1, a_2 \dots a_m$: constant
- f_d, f_p, f_c: disjunction function, propagation function, conjunction function
- $b_1, b_2 \dots, b_k$: set of probability of fact or rule

2. Overall Description

2.1 Database and syntax

The database for this project is a certain number of rule and facts.
The Syntax of this system rule/ fact is basically the same as in the standard datalog (that is no negation, no built-in predicates, and no function symbols), except that each rule/fact is associated with a real number value v in the range (0, 1], which indicates the probability of the rule/fact. That is, a rule is an expression of the form:

$$p(X_1, \ldots, X_n) :- q_1(Y_1, \ldots, Y_i), \ldots., q_k(Z_1, .., Z_j): v_1.$$

and a fact is an expression of the form:

$$q(a_1, \ldots a_m): v_2.$$

In the above rule and fact, the values v_1 and v_2 are probabilities of the given rule and fact. The value v_1 for instance can be thought of as saying that the probability that "the rule body implies the rule head" is v_1. For the fact, the value v_2 simply indicates the probability of the given fact, that is, the amount of truth in the fact. The values v_1 and v_2 are in the range (0, 1].

2.2 Database and Semantics

The semantics of this program includes “independent”, “maximum”, “minimum”, “product”. Each rule contains three functions: disjunction, propagation, and conjunction, expressed as follows: the triple $< f_d, f_p, f_c >$. Independent and maximum semantic is applied in disjunction function. First of all, the rules and facts are assumed to be "independent", in probability sense. That is, if we obtain different derivations of the same fact "p" with probabilities p : v_1 and p : v_2, then we use the probability disjunction function to combine these derivations into p : v_3 , where v_3 = $v_1+v_2-v_1*v_2$. If all the rules and facts are assumed to be “maximum”, we use the probability disjunction function to combine these derivations into p : v_3, where v_3 = max(v_1,v_2). The semantic for propagation function is “minimum” or “product”. “minimum” means that the fact "p" with probabilities p : v_1 where v_1 is the value after conjunction function, and the rule value v, min(v_1, v). “product” means that $v_1 * v$. The semantic for conjunction is “minimum” or “product” as well. If the probability of $q_1(\ldots)$ is $b_1, \ldots$, and probability of $q_k(\ldots)$ is b_k, then min$\{b_1, \ldots, b_k\}$. Minimum in conjunction function is min$\{b_1, \ldots, b_k\}$. “product” for conjunction function is $\{b_1 * , \ldots ,* b_k\}$.

In our program, you can choose those semantics before running the code, even customized define the semantics of those three functions.

2.3 Internal database

In the system, there are mainly 5 databases:

List ***IDB***

The database which store all derived facts.

List ***lastIDB***

Using in semi-naïve, the database which store all derived facts from last iteration.

List ***multiSet***

Using in semi-naïve, the database which store all derived facts from each derivation without disjunction.

List ***EDB***

The database which store facts in the input file.

List ***Rules***

The database which store all the rules from the input file.

2.4 Data Structure

We mainly design three class to store literal, rule and fact as shown below.

- Literal:

class Literal (example: edge (X, Y))
- Predicate : edge
- Variables [] : [0]:X, [1]:Y

Rule:

- Rule:

class Rule (example: reachable (X, Y) := edge(X, Y))
- Head: reachable (X, Y) //Literal
- Body: edge (X, Y) //List of literals

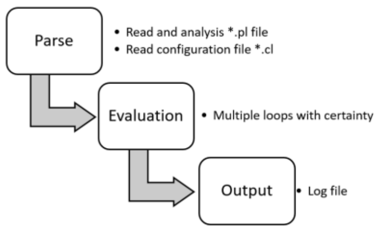
- Fact:

class Fact (example: edge (0, 1))
- Predicate: edge
- Constant []: [0]: 0, [1]:1
- Value: default value of certainty 0.5
- Path: List<Fact> //record the derived path
- Length: Integer

3. System working process

In order to evaluate the efficiency of naïve evaluation method and semi-naïve method, we developed a system prototype in java which can evaluate a program with certainty. The user can choose function for three semantics $< f_d, f_p, f_c >$ by modify the related configuration file.

In order to achieve those functions by java, we divide our program into three processes shown as below. There are three main parts of the system: parse, evaluation and output.


Parse
• Read and analysis *.pl file
• Read configuration file *.cl
Evaluation
• Multiple loops with certainty
Output
• Log file

4. Naïve and Semi-naïve

4.1 Naïve evaluation method

Minimum fixed-point semantics of ***Datalog*** yields a simple bottom-up evaluation algorithm: evaluations can be performed from an iteration of the underlying data instance containing only EDB. In each iteration, all rules are evaluated, and tuples that satisfy the rule body are derived (by the direct consequence operator TP); when no new tuples can be exported, the iterative evaluation stops. This method of evaluation is named a naive approach. Generally, naïve algorithm is a simple bottom-up approach as well as more time-wasting method to evaluate a Datalog program P. Specifically, in a standard datalog program without certainty, starting by assuming ***IDB*** is empty and then in each round of evaluation, it will start with exist facts and try to match with every rule, then derive new facts. In other word, it means repeatedly evaluate the rules using the EDB and the previous ***IDB***, to get new facts. The evaluation will terminate when there is no new fact derived, then the evaluation is finished. The main issue of naïve algorithm is that the same fact is derived repletely in each round, which will cost much unnecessary time and resource.

When it comes to ***Problog***, we combine the certainty with standard naïve algorithm, we need to calculate the certainty by using conjunction, propagation and disjunction functions for facts in each round. The condition of termination is also changed as follows, the evaluation will terminate at any round in which the certainty of no fact has increased, compared to its value in last round. In order to be more clearly, there is a special case to show the whole process of naïve method. If the case looks like as follow:

r1: $\mathrm{reachable(X,Y)\!:-edge(X,Y).\,v_1}$
r2: $\mathrm{reachable(X,Y)\!:-edge(X,Y),reachable(Y,Z).\,v_2}$
$\mathrm{edge(0{,}1)}.\,p_1$
$\mathrm{edge(1{,}2)}.\,p_2$
$\mathrm{edge(2{,}3)}.\,p_3$

Initially, we have 3 fact in ***EDB***, which is $\mathrm{edge(0{,}1).}$ $\mathrm{edge(1{,}2).}$ and $\mathrm{edge(2{,}3).}$ Assuming the probability of each fact is 0.5, 0.5 and 0.5 respectively. At the same time there are two rules:
$\mathrm{reachable(X,Y)\!:-edge(X,Y).}$
$\mathrm{reachable(X,Y)\!:-edge(X,Y),reachable(Y,Z).}$
Similarly, assuming the probability of each rule is 0.5 and 0.5 respectively. Assuming in this case, the conjunction function is ***ind*** function, propagation is min function and conjunction is product function.

So, in first iteration, the ***IDB*** is empty, and we read fact from ***EDB***:

Iteration 1: EDB	
edge (0, 1):	0.5
edge (1, 2):	0.5
edge (2, 3):	0.5

In iteration 2, we use ***EDB*** and rule r1 to derive new facts which stored in ***IDB***:

Iteration 2: ***r1***	
reachable (0, 1):	0.5
reachable (1, 2):	0.5
reachable (2, 3):	0.5

In iteration 3, we use ***EDB*** and ***IDB*** to derive all facts:

Iteration 3: ***r1, r2***	
reachable (0, 1):	0.5
reachable (1, 2):	0.5
reachable (2, 3):	0.5
reachable (0, 2):	0.25
reachable (1, 3):	0.25

In iteration 4, we use facts of last iteration to replace with previous ***IDB*** and use ***EDB*** and ***IDB*** to derive all facts.

Iteration 4: ***r1, r2***	
reachable (0, 1):	0.5
reachable (1, 2):	0.5
reachable (2, 3):	0.5
reachable (0, 2):	0.25
reachable (1, 3):	0.25
reachable (0, 3):	0.125

The above is detailed process of naïve method. According to recursion, we can obtain minimum fixpoint of datalog problem with naïve method finally.

Based on the basic logic, below is the pseudocode of our implementation about naïve algorithm:

```
NaïveEvaluation
InputFacts ← IDB
Iterate all rules (Matching)
    Each rule matches all the facts $p_1, \dots, p_n$
        if all the atoms of body matched
            do $f_c, f_p$ , generate derived fact and add to newIDB
Iterate each fact in newIDB (Disjunction)
    If there is one more fact with same predicate and constants,
        do $f_d$, Disjunction
Compare each fact in newIDB and IDB (Updating)
    If every fact appears in newIDB and IDB with same certainty
        Evaluation terminated.
    If it is a new fact
        Add to IDB
    If it is a exist fact with higher certainty
        Update the higher certainty
Clear newIDB
```

4.2 Semi-naïve evaluation method

Semi-naïve evaluation method is a booster of naïve algorithm in some cases, which should always derive same result but maybe cost less time or resource. The main purpose is to avoid or minimize repeated application of rules and facts at each iteration. To optimize the performance, we consider two aspects of optimization about semi-naïve evaluation based on the rule's type.
In general, there are two ways for optimization:
1. Avoid redundant rules evaluation;
2. Shrink size of input facts for each iteration.
Consider the naïve evaluation of the above program, and the case looks like as follow:

$\mathrm{r1{:}\ reachable(X,Y){:}-edge(X,Y).\,v_1}$
$\mathrm{r2{:}\ reachable(X,Y){:}-edge(X,Y),reachable(Y,Z).\,v_2}$
$\mathrm{edge(0{,}1)}.\,p_1$
$\mathrm{edge(1{,}2)}.\,p_2$
$\mathrm{edge(2{,}3)}.\,p_3$

By using naïve algorithm, in each iteration, we can find that for the rule ***r1*** is always produce the same facts with same certainties. Since the atom in the body is from ***EDB***, as we know the fact in ***EDB*** will never be changed in each iteration, because it cannot appear in the head of any rules. Accordingly, we try to define a theorem:
Theorem 1: If all the atoms in the body in a rule are from ***EDB***, this rule will always produce same facts with same certainties in each iteration, if ***EDB*** is not changed during the evaluation.
Optimization 1:
For the rule which accords with Theorem 1, we can only evaluate the rule with the facts in ***EDB*** in the first iteration and store the derived facts in a dataset ***multiSet***. For the following iterations, we skip to evaluate such rules. Instead, we compare every derived fact with ***multiSet***, if we find a fact in ***multiSet*** which has the same predicate and same fact but different derivation, then we will do the disjunction and add to ***newIDB***. In this way, we can avoid unnecessary evaluation for some rules.
Optimization 2:
For ***r2***, the atoms in the body are not only from ***EDB***, it cannot apply to theorem 1, but we can also reduce redundant times of evaluation. For example, in the following facts reachable(0, 2),

$$\mathrm{reachable(0{,}2){:}-edge(0{,}1),reachabe(1{,}2).}$$

We cannot derive reachable(0, 2) from this rule before we have reachable(1 ,2).
After we get reachable(0, 2) from this rule, the certainty of reachable(0, 2) will changed only reachable(1, 2) changed, since the certainty of edge(0, 1) will never change.
For example, here are two general cases which our semi-naïve algorithm is dealing with:
1.

Iteration 2: ***r2***	Iteration 3: ***r2***
~~reachable(1, 2)~~	reachable(1, 2) : 0.25
edge(0, 1) : 0.5	edge(0, 1) : 0.5
Cannot derive any fact.	↓
	reachable(0, 2): 0.125

2.

Iteration 2: ***r2***	Iteration 3: ***r2***	Iteration 4: ***r2***

reachable(1, 2) : 0.25 edge(0, 1) : 0.5 ↓ reachable(0, 2): 0.125	reachable(1, 2) : 0.25 edge(0, 1) : 0.5 ***No need to evaluate!***	reachable(1, 2) : 0.27 edge(0, 1) : 0.5 ↓ reachable(0, 2): 0.135

Thus, our optimization logic is to track each new derived fact in each iteration and find out all the rules which the fact can appear in the body. Then, for the next iteration, we can only evaluate the corresponding rules with derived fact from last iteration. This is a way to shrink the size of input facts in each iteration.

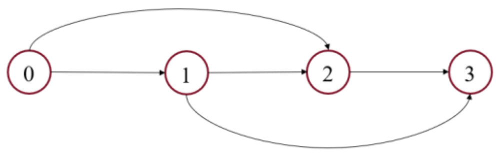
Based on the basic logic, below is the pseudocode of our implementation about semi-naïve algorithm:

```
Semi-naïveEvaluation
InputFacts ← lastIDB + relatedFacts
OPTrules ← filter redundant rules
Iterate all rules in OPTrules (Matching)
    Each rule matches all the facts in lastIDB
        if all the atoms of body matched
            do $f_c, f_p$ , generate derived fact and add to newIDB
Iterate each fact in newIDB (Disjunction)
    If there is one more fact with same predicate and constants,
        do $f_d$, Disjunction, update certainty to one fact, remove another fact.
    If the fact can do disjunction in DetailIDB
        do $f_d$, Disjunction, update certainty
Compare each fact in newIDB and IDB (Updating)
    If every fact appears in newIDB and IDB with same certainty
        Evaluation terminated.
    If it is a new fact
        Add to IDB
    If it is a exist fact with higher certainty
        Update the higher certainty
lastIDB ← newIDB
Clear newIDB
```

5. Tracking

In order to ensure the correctness as well as performance of the system, we need to keep track of derivation of each fact.

For example:



In round 2, We will get another new reachable (0, 2)

- reachable (0, 2) :- edge(0, 1), reachable(1, 2)

In IDB, there is already a fact reachable (0, 2) derived from round 1.

- reachable (0, 2) :- edge(0, 2).

How can we distinguish them?

Two approach:

1. Path
2. Length

5.1 Tracking by path

We record path for each fact. By the end of each iteration, if we get two or more facts have same predicate and same constants, but different derivations, we could record their paths to distinguish them and do disjunction.

For the above example, the ***reachable(0, 2)*** has two derivation:

1. edge(0, 2)

2. edge(0, 1), reachable(1, 2)

We record these two derivations as path for each. In naïve algorithm, we do disjunction between both and get a combined certainty of ***reachable(0, 2)***. In semi-naïve algorithm, we put both of two facts to ***multiset*** and also do disjunction between both and get a combined certainty of ***reachable (0, 2)***.

5.2 Improvement –Tracking by length

When we implement to record path for each fact, we found an interesting and meaningful conclusion, which is each fact actually has length. Assuming all of facts are edges in graph, and then we can see facts in EDB as initial edges in this graph which default value of length is 1. During the recursion of derivation of facts, we found each fact can have length belong to themselves and if new facts are derived from existing facts, which length is increasing. In other word, each length of fact is equal to sum of length of facts which derived this fact. For example, ***reachable (0, 2)← edge (0, 1), reachable (1, 2)***. The length of ***reachable (0, 2)*** is equal to length of ***edge (0, 1)*** plus length of ***reachable (1, 2)***. Based on this discover, we can implement semi-naïve method with more efficient space and time complexity than implement semi-naïve method by tracking path since we only need a integer or long datatype to store the length. In addition, when we do test about some testcases, we found if we record length of fact, we can find complete **Fibonacci Sequence**. The detailed introduction will introduce in section 7.

Another application of length tracking is the extension of ***multiset***. For semi-naïve evaluation, the standard ***multiset*** stores the facts from different derivations with the most recent certainty.

For example, assume in a round of evaluation, we get a fact ***reachable(0, 2)*** from rule ***r2*** with certainty 0.25. And in a later round of evaluation, we also get a fact ***reachable(0, 2)*** from rule ***r2*** with higher certainty 0.3. Then the ***multiset*** will only store the later ***reachable(0, 2)*** with certainty 0.3.

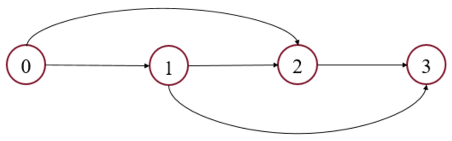
Since the ***multiset*** does not store all the derivation of facts in all rounds, we use length-tracking to design an ***extension multiset***. In the above example, we could store two ***reachable(0, 2)*** with different lengths. When we need to pick a fact from the ***extension multiset***, we could choose the fact with longest length, which is the fact with most recent certainty.

6. Experimental Analysis

6.1 Some specific testcase

To check the correctness of our implementation of naïve method and tracking method, we test the system with some specific testcase, and check the result with our manually evaluation result.

Testcase1 (Fact with multi-paths)



<ind,*,min>

In this case, our purpose is to make sure that our method can detect different derived path for same fact in the same iteration. As the figure shown above, there are two ways to get reachable(0, 3):

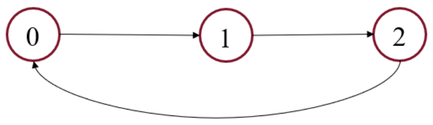
- reachable (0, 3) :- edge(0, 1), reachable(1, 3)
- reachable (0, 2) :- edge(0, 2), reachable(2, 3)

The derivation of relation ***reachable*** is shown as below:

Iteration1	Iteration2	Iteration3	Iteration4
*reachable(0,2).0.25 *reachable(1,3).0.25 *reachable(0,1).0.25 *reachable(1,2).0.25 *reachable(2,3).0.25	*reachable(0,2).0.34375 *reachable(1,3).0.34375 reachable(0,1).0.25 reachable(1,2).0.25 reachable(2,3).0.25 *reachable(0,3).0.234375	reachable(0,2).0.34375 reachable(1,3).0.34375 reachable(0,1).0.25 reachable(1,2).0.25 reachable(2,3).0.25 *reachable(0,3).0.275390625	reachable(0,2).0.34375 reachable(1,3).0.34375 reachable(0,1).0.25 reachable(1,2).0.25 reachable(2,3).0.25 reachable(0,3).0.275390625

Compared the result in each iteration manually, we find the result is correct for this testcase. we can conclude that the system can detect different derived path for same fact in the same iteration and do disjunction function. Also, the system keeps tracking of all facts in each iteration and can terminate when there is no more fact generated with increased certainty.

Testcase2 (acyclic graph)



<ind,*,min>

In this case, we are going to test our system with acyclic dataset.

As shown as above, we made a simplified version of acyclic graph coming from ***graph10.pl***, thus we can do the evaluation manually in some iterations at the beginning to compare the result derived from the system.

The derivation of relation ***reachable*** is shown as below:

Iteration1	Iteration2	Iteration3	Iteration4

*reachable(0,1).0.25 *reachable(1,2).0.25 *reachable(2,0).0.25	reachable(0,1).0.25 reachable(1,2).0.25 reachable(2,0).0.25 *reachable(0,2).0.125 *reachable(1,0).0.125 *reachable(2,1).0.125	reachable(0,1).0.25 reachable(1,2).0.25 reachable(2,0).0.25 reachable(0,2).0.125 *reachable(0,0).0.0625 reachable(1,0).0.125 *reachable(1,1).0.0625 reachable(2,1).0.125 *reachable(2,2).0.0625	*reachable(0,1).0.2734375 *reachable(1,2).0.2734375 *reachable(2,0).0.2734375 reachable(0,2).0.125 reachable(0,0).0.0625 reachable(1,0).0.125 reachable(1,1).0.0625 reachable(2,1).0.125 reachable(2,2).0.0625
Iteration5	Iteration6	Iteration7	Iteration8…
reachable(0,1).0.2734375 reachable(1,2).0.2734375 reachable(2,0).0.2734375 *reachable(0,2).0.13671875 reachable(0,0).0.0625 *reachable(1,0).0.13671875 reachable(1,1).0.0625 *reachable(2,1).0.13671875 reachable(2,2).0.0625	reachable(0,1).0.2734375 reachable(1,2).0.2734375 reachable(2,0).0.2734375 reachable(0,2).0.13671875 *reachable(0,0).0.068359375 reachable(1,0).0.13671875 *reachable(1,1).0.068359375 reachable(2,1).0.13671875 *reachable(2,2).0.068359375	*reachable(0,1).0.275634765625 *reachable(1,2).0.275634765625 *reachable(2,0).0.275634765625 reachable(0,2).0.13671875 reachable(0,0).0.068359375 reachable(1,0).0.13671875 reachable(1,1).0.068359375 reachable(2,1).0.13671875 reachable(2,2).0.068359375	reachable(0,1).0.275634765625 reachable(1,2).0.275634765625 reachable(2,0).0.275634765625 *reachable(0,2).0.1378173828125 reachable(0,0).0.068359375 *reachable(1,0).0.1378173828125 reachable(1,1).0.068359375 *reachable(2,1).0.1378173828125 reachable(2,2).0.068359375

Above are the first 8 iterations of evaluation. By the tracking *, we can see in the first three iterations, we can derive new facts in each iteration. And in following iterations, since there is a cycle in the dataset, we should always keep track of the fact which the certainty is changed, then take the fact in the following iteration to derived corresponding fact with new certainty if possible. Based on the functions of conjunction, propagation and disjunction in this case, the certainty of existed fact is increased.

During our experiment, we find that the certainty of each derived fact is increasing, but the growth is getting smaller and smaller. For example, in iteration 4, the certainty of reachable(0, 1) is increased from 0.25 to 0.2734375. And in iteration 7, the changing is from 0.2734375 to 0.27563… Since our system is running on Java platform, we find there is a precision limited about data type **double** in Java.

For the above testcase, we find the system terminates at iteration 52, since the certainties of all facts are not changed.

In iteration 52, by our three functions f_d, f_p, f_c, the certainties of reachable(2, 0), reachable(0, 1) and reachable(1, 2) should be increased in iteration 52, since the certainties of reachable(0, 0), reachable(1, 1) and reachable(2, 2) were increased in iteration 51 compared with iteration 50. But why in the system, the certainty is not changed?

Iteration 50	Iteration 51	Iteration 52
reachable(0,1).0.27586206896551724 reachable(1,2).0.27586206896551724 reachable(2,0).0.27586206896551724 *reachable(0,2).0.13793103448275862 reachable(0,0).0.0689655172413793 *reachable(1,0).0.13793103448275862 reachable(1,1).0.0689655172413793 *reachable(2,1).0.13793103448275862 reachable(2,2).0.0689655172413793	reachable(0,1).0.27586206896551724 reachable(1,2).0.27586206896551724 reachable(2,0).0.27586206896551724 reachable(0,2).0.13793103448275862 *reachable(0,0).0.06896551724137931 reachable(1,0).0.13793103448275862 *reachable(1,1).0.06896551724137931 reachable(2,1).0.13793103448275862 *reachable(2,2).0.06896551724137931	reachable(0,1).0.27586206896551724 reachable(1,2).0.27586206896551724 reachable(2,0).0.27586206896551724 reachable(0,2).0.13793103448275862 reachable(0,0).0.06896551724137931 reachable(1,0).0.13793103448275862 reachable(1,1).0.06896551724137931 reachable(2,1).0.13793103448275862 reachable(2,2).0.06896551724137931

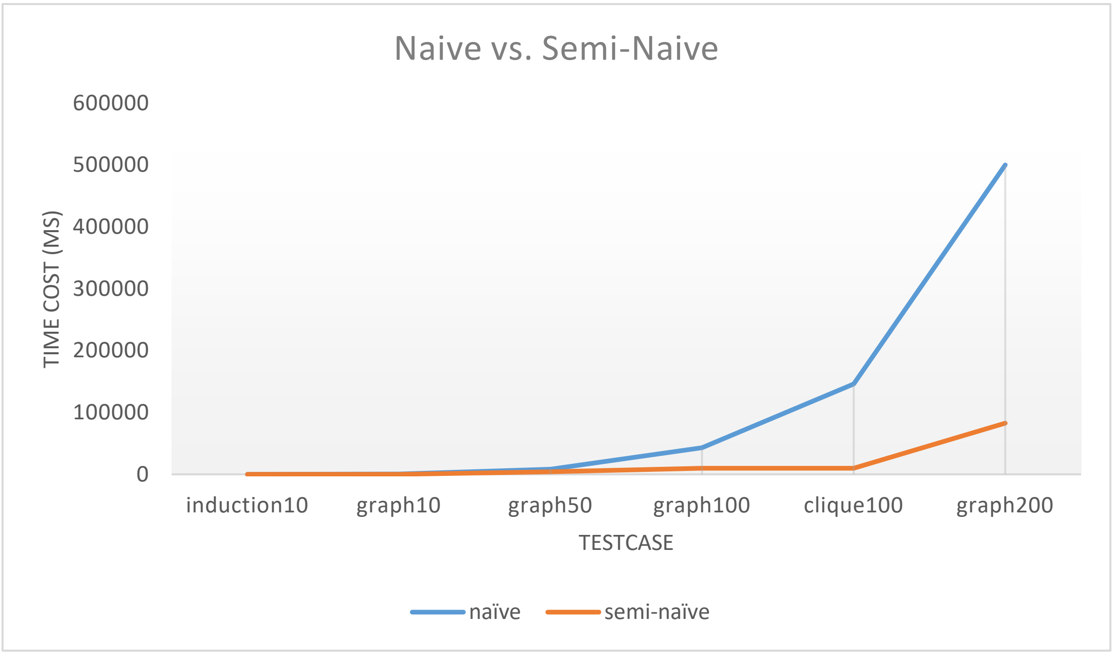
After some research about the similar issues of computation, we find there is an issue about double datatype in Java called loss precision, which means when doing math calculation with some extremely small numbers, the precision of lowest bit may be loss. Back to our testcase, if we look at the certainty of reachable(0, 0) in iteration 50 and 51, the certainty is only increased by 0.00000000000000001. Thus, when we derive reachable(2, 2) through edge(2, 0) and reachable(0, 0), the 0.00000000000000001 will be loss, that's why the certainty is not changed in iteration 52.

6.2 Experimental Results between Naïve and Semi-naïve

In this part, we are going to compare the performance of Naïve method and Semi-naïve method about different scale or structure of dataset.

EDB size	Test case	Acyclic	Naïve(ms)	Semi-naïve(ms)	Speed up
10	Induction10.pl	No	26	25	3.85%
10	Graph10.pl	Yes	419	156	62.76%
50	Graph50.pl	Yes	8126	3971	51.13%
100	Graph100.pl	Yes	42997	9751	77.32%
100	Clique100.pl	Yes	146082	10060	93.11%
200	Graph200.pl	Yes	＞15mins	82714	99.99%

$$Speed\ up = \frac{|Naive\ - SemiNaive|}{Naive} \times 100\%$$



In our experimental result, we conclude that the performance of Semi-naïve method is less time-cost compared with Naïve method, especially for acyclic dataset. Along with the growth of scale of dataset(from 10 to 200), the difference between Semi-naïve and Naïve will be more obviously. However, we think that if he scales of dataset becomes very large, the time-cost of semi-naïve method may be more than naïve method, since for each new derived fact, we need to compare with the facts in ***multiset***, which maybe takes a huge time cost. However, the space complexity of semi-naïve is obviously much more than naïve method, since it need space to store ***multiset*** while naïve method doesn't need to do that.

7. Challenge and Finding

In general, the implementation of ***Problog*** system is not an easy issue for us. Especially at the beginning, we start to learn datalog, tried to realize uncertainty and probability calculation. Through the project, we keep working and discussion, tried different testcases to ensure the correctness of the system also keep optimizing our implementation.

7.1 Challenge

7.1.1 Naïve implementation

The implementation of Naïve method for ***Problog*** is a very difficult process. In the early stages of research, we do not fully understand how naïve method is implemented. In other word, we do not clearly understand how to derive all of facts according to recursion with ***EDB*** and ***IDB***. Hence, we consider some uncorrected idea to implementation of naïve method. For example, in each iteration, we attempt to compare new derived facts with exiting ***IDB***. If there are new facts which not including in ***IDB*** or the value of fact in ***IDB*** has changed, we will do disjunction. To achieve this wrong idea, we spend lots of time on how to compare new facts in each iteration and facts in existing IDB. And we find some techniques like tracking with path of each facts and tracking with length of each facts. However, we actually do lots of disjunction and this is because we do not cleat IDB in each iteration instead of disjunction with facts in ***IDB*** contiguously. This is a very painful problem for us because we think about why the result between ***Naïve*** and ***Semi-Naïve*** are different. However, this is also a meaningful process. It helps us to better understand how ***Naïve*** method to implement and what is the drawback of naïve method. And finally, we understand correct process of naïve algorithm.

7.1.2 Semi-naïve implementation

The implementation of Semi-naïve algorithm for ***Problog*** is a heart also a hard part in this project. Difference from traditional implementation in standard datalog program, we must consider about the disjunction function to update the certainty for existed fact at the end of each iteration if applicable.

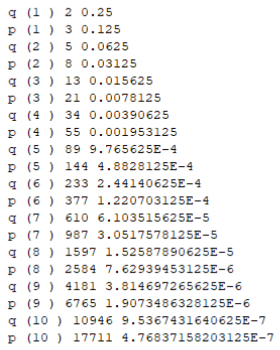
$\mathrm{r1:\ reachable(X, Y){:}-edge(X, Y).}$

$\mathrm{r2:\ reachable(X, Y){:}-edge(X, Y), reachable(Y, Z).}$

For example, as above, after the first round, Semi-naïve algorithm will only evaluate the 2nd rule.
Then the problem is how can we do the disjunction for new derived fact?
Assume there is a fact ***reachable(0, 2)*** can be derived from both of r1 (***edge(0, 2)***)and r2 (***edge(0, 1)*** and ***reachable(1, 2)***).
For naïve method, in each iteration the system will evaluate both of two rules and do disjunction to get ***reachable(0, 2)***.
For semi-naïve method, after the first evaluation with ***EDB***, it will **not** evaluate ***r1*** anymore in the following iterations.
In our first attempt, after ewe get reachable(0, 2) from ***r2***, we will do the disjunction with reachable(0, 2) in ***IDB*** with the best certainty, which is incorrect.
This fault is not easily to find out since it's also causes the increasing of the certainty of related fact, but if we focus on the result by each iteration compared with naïve method, it's obviously different. Then we find a new approach to implement semi-naïve which could derive exactly same result compared with naïve method. We use a ***multiset*** to store all derived fact before disjunction. For the above example, when we get ***reachable(0, 2)*** from ***r2***, then we will search in ***multiset***, if there is another ***reachable(0, 2)*** from different derivation, we will do the disjunction then update ***IDB***.

7.2 Finding – Fibonacci

During our experiment with the dataset ***induction10.pl***, we surprisely find a Fibonacci sequence by listing the length of each derived fact.



```
q (1 ) 2 0.25
p (1 ) 3 0.125
q (2 ) 5 0.0625
p (2 ) 8 0.03125
q (3 ) 13 0.015625
p (3 ) 21 0.0078125
q (4 ) 34 0.00390625
p (4 ) 55 0.001953125
q (5 ) 89 9.765625E-4
p (5 ) 144 4.8828125E-4
q (6 ) 233 2.44140625E-4
p (6 ) 377 1.220703125E-4
q (7 ) 610 6.103515625E-5
p (7 ) 987 3.0517578125E-5
q (8 ) 1597 1.52587890625E-5
p (8 ) 2584 7.62939453125E-6
q (9 ) 4181 3.814697265625E-6
p (9 ) 6765 1.9073486328125E-6
q (10 ) 10946 9.5367431640625E-7
p (10 ) 17711 4.76837158203125E-7
```

In the second column, we can clearly see that the value of length can produce a Fibonacci sequence. It is a special test case. If and only if the newfact is generated from facts where are generated from last iteration.

8. Conclusion

In this project, we studied evaluation of probabilistic ***datalog*** program and database. In the context of each rule and fact with probability which range is (0, 1], we achieved implementation of naïve and semi-naïve method and compare efficiency of time space of naïve and semi-naïve method. Moreover, for naïve method, we analyzed detailed process of how to derive all facts with better certainty and explanted the relation between ***IDB*** and ***EDB***. For semi-naïve method, we considered the drawback of naïve method and implement how to improve naïve method as well as guarantee the correctness of program. Specifically, in order to implement the result of semi-naïve is same as naïve method, we supposed special database (***multiSet***) which store all derived facts from each derivation without disjunction. According to this special database, we combine our tracking technique (tracking path or tracking length) to achieve result of semi-naïve method is same as naïve method. And in process of implementation of above methods, we discover the method of computing ***Fibonacci Sequence***. We are currently pursuing this Idea to further discover simple formula of computing ***Fibonacci Sequence***. Current results in this direction are encouraging.

9. Contribution

In this project, each of the member contributes more or less equally in the project in term of research, group discussion, coding, presentation, demo and report. In early stage, every member contributes same on building the construction of program.
After previous work for preparation project, we divided the program into three parts: parser, naïve and semi-naïve evaluation with tracking path, and naïve and semi-naïve evaluation with tracking

length. Luguang Liu mainly thought and completed how to achieve naïve and semi-naïve method with tracking path. Chenwei Song mainly finished and discover naïve and semi-naïve method by tracking length. Zhongxu Huang mainly achieved parser and IO of files. For presentation and demo, each person of group contributed to how and prepare slides and how-to speech together. Finally, every people in group write this report and complete the testing to the program together.

10. Reference

[1] Lakshmanan Laks V.S. and Shiri Nematollaah. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.
[2] Mingzhou Lin, https://github.com/MingzhouLin/DatalogProgramParser