Write-up

Zhizhou Zhang

University of Rochester

CSC242 Intro to AI

Project 3: Uncertain Inference

2016/04/14

1. Input and output

The input of the inference algorithm is basically the same: the .bif or .xml file which contains the Bayesian network, the variable to query and evidence variables with their value. For the approximate inference in part 2, there is one more argument standing for the number of trails in approximation. To deal with unfixed-length evidence variables and the corresponding values, I use args argument in the main function.

The output of the program is the distribution of the query variable: each possible value for it and the possibility corresponded to it.

2. Part I Exact Inference

The algorithm of this part is stated explicitly in the AIMA section 14.4 as “inference by enumerating”. I use Depth First Search (DFS) to implement the enumerating part. However some work has to be finished before the enumerating the variables. First, I have to let the parser, which is provided by Professor Ferguson, parse the Bayesian Network according to their file type, so that I can get the whole network and know the total number the variables. Then I have to deal with the query variable and evidence variables, since they are either fixed or need to be searched and cannot be assigned

valued during the DFS.

The DFS is very typical: I use an auxiliary variable to record the current depth (it begins with the numbers of fixed variable which is all evidence variable plus query variable), when it reaches the total number of the variables, which is the maximum depth, query the possibility of current assignment and return. Or using loop to exhaustively assign the value of the next unassigned variable, increase the current depth and go deeper (the whole process of DFS should go in topological order of the variables).

By assign each possible value to query variable and use DFS, I can get its distribution.

3. Part II Approximate Inference

In this part, I implement both rejection sampling and likelihood weighting. The input and output of the program is roughly the same as part I except it has a new variable indicates the number of total trials.

a) Rejection-sampling

The pseudocode of this part is showed in the Figure 14.14 in AIMA. The main idea of this part to generate exact numbers of trail in prior-sampling. Then reject all the trails that are not consistent with evidence variables. Count the remaining trails according to the value of query variables.

When dealing with the every simple or prior-sample, I use a special trick to simplify the question. As the every entry in distribution of probability is between 0 and 1, at the same time random function in Java as Math.random() also generate a

pseudo-random number in 0 and 1. So every time I get a random number from the function, then sum the possibility from the current variable until it exceeds. By doing so, I can get the value of variable easily. For example, considering a variable a with value of a0, a1 and a2. The distribution of probability is {0.2, 0.3, 0.5}. If I get a random value 0.75 from the trial, then first sum the first entry in distribution: 0 + 0.2 = 0.2 < 0.75, so go next. 0.2 + 0.3 = 0.5 < 0.75, go further. When it comes to a2, 0.5 + 0.5 = 1.0 > 0.75, so break the loop and set a to a2 in this assignment. The rest of the code is straightforward.

b) Likelihood weighting

The sampling-rejection inference simply reject too many examples, in order to deal it, Likelihood weighting use special technique. When it comes to evidence variable, check the possibility from table directly and does not trail. Otherwise generate examples as prior-sampling procedure. By multiplying the weight, the outcome is guaranteed to be consistent with the evidence variables. There isn't much to say about the implementation of this algorithm as it use the same trick above.

4. Result and analyze

```
C:\Users\mayer\Desktop>java -jar haha.jar aima-alarm.xml B J true M true
true 0.2841718353643929
false 0.7158281646356071
```

I use the alarm to test the correctness of my exact inference algorithm, it shows the same result from textbook. For other examples, the three algorithm generate the same or similar result if I set numbers of the approximate sampling large enough. I didn't record the time of exact inference, but it is really not good, when I test big networks, it

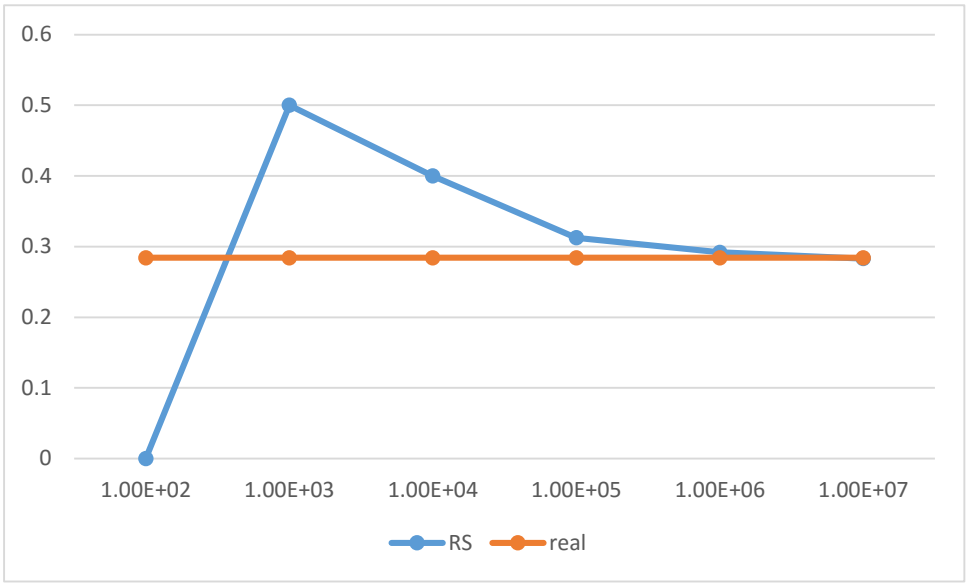
becomes very slow. It is consistent with the theory form textbook that it is #P-hard.

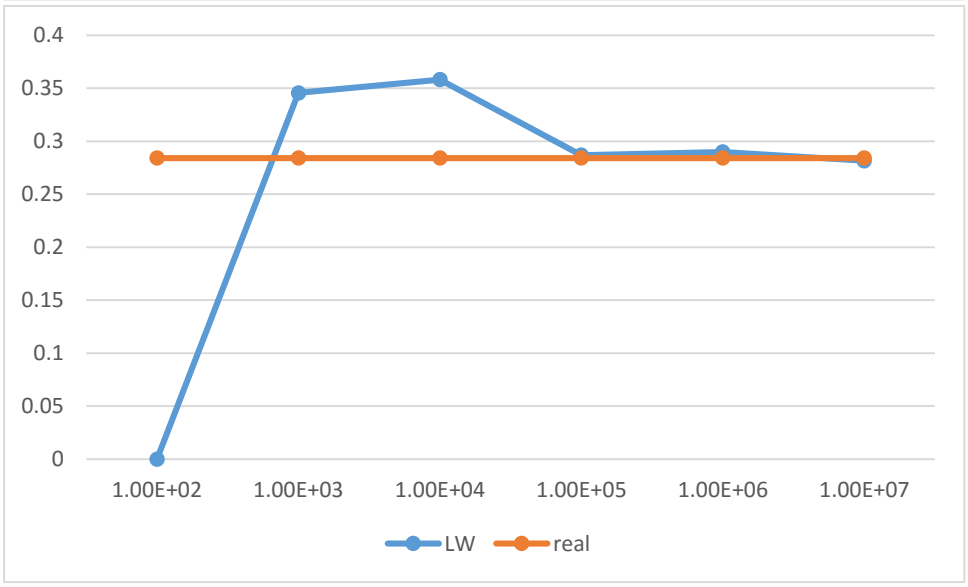
When it comes to the approximate part, I compare several different trail numbers of two algorithms. They run on the same computer with aima-alarm.xml, to query B when J and M are true. I just record the probability of true and the time (false can be calculated with 1-P(true)), RS stands for reject sampling and LW stands for likelihood weighting. Time is in millisecond. 1e3 = 1000

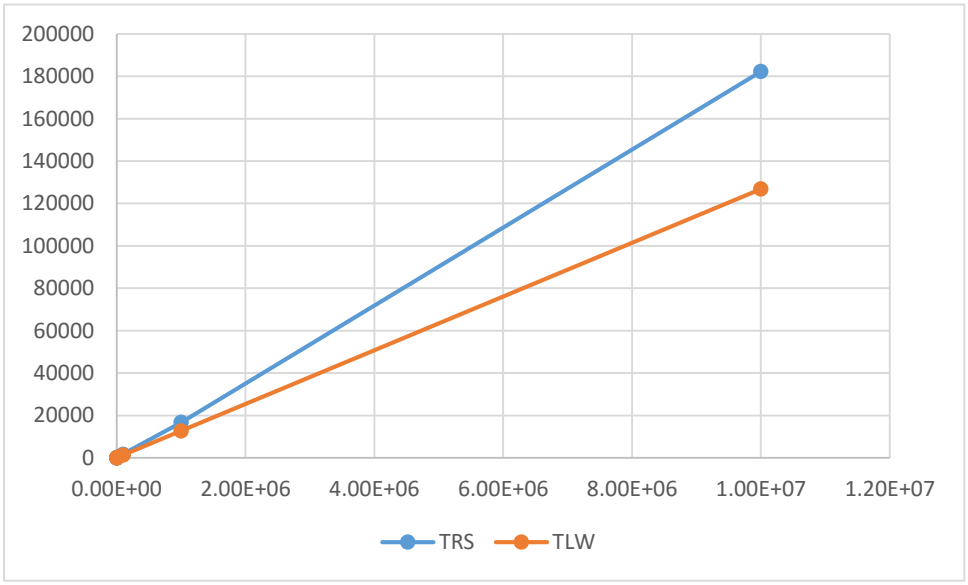
	1e2	1e3	1e4	1e5	1e6	1e7
RS	NaN	0.5	0.4	0.3125	0.2918	0.2830
T_{RS}	16	52	244	1726	16760	182251
LW	0.0	0.3456	0.3582	0.2869	0.2899	0.2816
T_{LW}	17	50	197	1352	12761	126822

Remember the exact result from part I is close to 0.2842, so we can see with the increase in trail times, the approximate result is closer to the real one. However the time is really horrible when n becomes too large. As said in the textbook, it is a trade-off between accuracy and time. The time complexity is exponential, since the x-axis is exponential and the graph is close to linear.

Generally speaking, the Likelihood Weighting is faster than rejecting-sampling and better. It’s interesting that enlarge trail number may not generate more accurate result, as shown between 1e5 and 1e6 of Likelihood Weighting. The time is longer but the result is worse. But as a trend, with the trial time grows larger, the result is closer to real one. And when trail number is very small both algorithm can’t even generate enough sample for both values in the distribution.


0.6
0.5
0.4
0.3
0.2
0.1
0
1.00E+02
1.00E+03
1.00E+04
1.00E+05
1.00E+06
1.00E+07
RS
real


0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1.00E+02
1.00E+03
1.00E+04
1.00E+05
1.00E+06
1.00E+07
LW
real


200000
180000
160000
140000
120000
100000
80000
60000
40000
20000
0
0.00E+00
2.00E+06
4.00E+06
6.00E+06
8.00E+06
1.00E+07
1.20E+07
TRS
TLW

5. Resource

I use code provided from Professor Ferguson uploaded from blackboard.