Problem 4.1 (a) The directed graphical model based on my intuition is shown in Figure 4.1

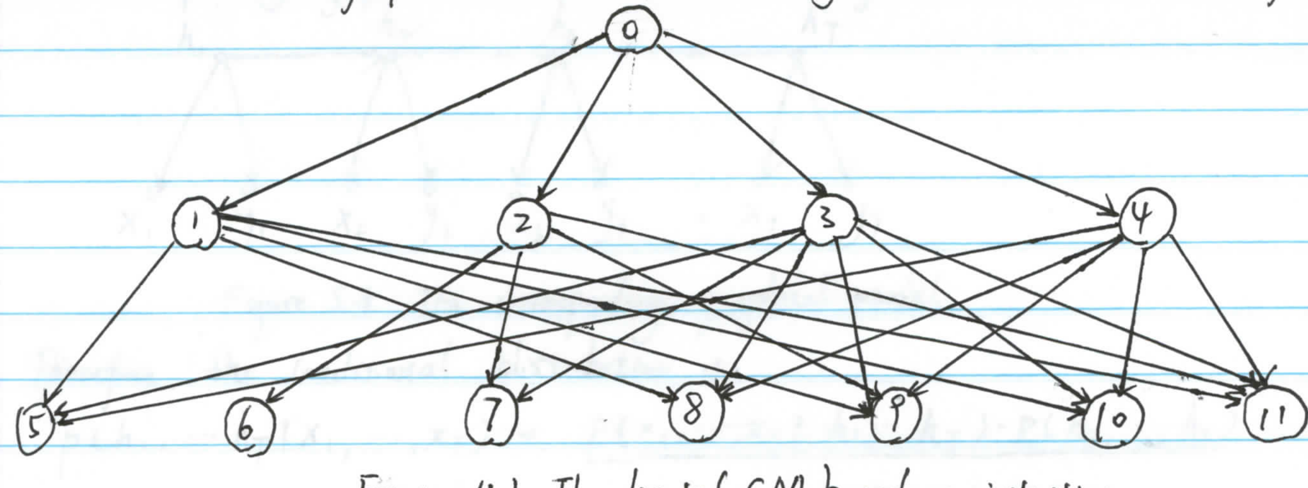


Figure 4.1 The directed GM based on intuition

I have used the graphical model in Problem 4.2 ~~and~~ for reference and setup my own model with the similar structure. But I have deleted some edges that I think are unnecessary. For example, I think flu has nothing to do with gastric problems so I removed ~~ego~~ edge $1 \to 6$.

[Implementation]

I have implemented the directed GM in Python (prob4_1.py). I have designed data structure for conditional probabilities, nodes, and directed GM. The conditional probability is stored in a 2-D array, where each row is corresponding to one assignment of the observed variables, and each ~~ro~~ column is corresponding to one value of the unknown variable. For example, the conditional probability ~~$P(X_5 | X_1, X_2, X_3, X_4)$~~ $P(X_2 | X_0)$ is represented by the 2x2 table below:

Table 4.1 The factor table for $P(X_2 | X_0)$

$X_0 \backslash X_2$	0	4
0	N_1	N_2
1	N_3	N_4

The entry of the table is the binary representation of the assignment of variables. For example, 4 has bit 2 as 1, so it represent $X_2 = 1$. This makes the indexing of the table much easier. We just need to do bitwise AND with the mask before indexing. The mask is determined by the variables we are interested in. For example, the mask of X_2 is 4, and the mask of $[X_1, X_2, X_3, X_4]$ is 30. With this data structure, we can represent any conditional probability in a at most 4096×4096 factor table, because there are 12 nodes in the GM. Besides, the numbers in the factor table could be unnormalized so we need to normalize it when output the probability or doing ~~production~~ multiplication and summation. For example, $P(X_2=0 \mid X_0=0) = \frac{N_1}{N_1+N_2}$.

We also design a class Node for the node in the GM. It stores the name and index of the variable, the pointer to its parent and the corresponding factor graph.

The class DirectedGM stores a map of nodes. When adding nodes, it also establish the directed edge between them. It also provides interface to do training, comparing, querying and sampling. This is how I have implemented the graphical model.

(b) Estimating Parameters

Based on the data structure I design, the process of estimating parameters (or training) is quite simple. First, we need to read in the sample data file. Then, for each sample data point, we need to increase the corresponding element of every factor table by 1 in the graphical model. The entry is determined by bitwise AND the sample data with the corresponding mask.

For example, for sample data $d_i = 5$, we have $d_i \;\&\; 4 = 4,\; d_i \;\&\; 1 = 1$, so we need to increase N_4 by 1. After reading in all the sample data, we need to re-normalize the conditional probabilities. One of the normalized factor table is shown in Table 4.2.

Table 4.2 Factor table of P(2|0) after training

P(X_2|X_0)	0	1
0	0.899864	0.100136
1	0.90002	0.09998

The runtime of the training process with 4,000,000 data points is about 59s.

(c) Model Accuracy

We have compared the joint distribution estimated by our graphical model with the true joint distribution by calculating the L1 distance. The result is shown in Figure 4.2.

```
Problem 4.1(c): Measure the accuracy of the model by comparing it with the true distribution
The accuracy (L1 distance) of the graphical model is 0.381109
```

Figure 4.2 The result of model accuracy comparison

The L1 distance is 0.381109. This means that the average error of probability of each assignment is

$$avg.error = \frac{L1\ distance}{Num.of\ data\ points} \approx \frac{0.381109}{4096} \approx 9.30 \times 10^{-5}$$

I think this result suggests the graphical model is relatively accurate.

(d) Querying

We have implemented the Brute-Force query method by marginalizing and normalizing the joint distribution. Again, we use bitmask to simplify the problem. For example, for the query task $P(X_1 = 1|X_8 = 1, X_{11} = 1)$, the querying variable X_1 is corresponding to the bitmask $b_q = (00000000010)_2$, and the given variables are corresponding to the bitmask $b_g = (10010000000)_2$, and the other variable are corresponding to the bitmask $b_o = \sim\left(b_q \middle| b_g\right) = (011011111101)_2$. Then we can calculate the conditional probability very efficiently:

$$P(X_1|X_8 = 1, X_{11} = 1) = \frac{\sum P(X_o, X_1 = 1, X_8 = 1, X_{11} = 1)}{\sum P(X_{qo} X_8 = 1, X_{11} = 1)}$$

For the numerator, we sum the joint distribution over all the X within [0, 4095] such that $X \,\&\, (b_q \mid b_g) = (b_q \mid b_g)$; for the denominator we sum the joint distribution over all the X within [0, 4095] such that $X \,\&\, b_g = b_g$.

The query results of the Brute-Force method and the comparisons with the result extracted from the true distribution are shown in Table 4.3.

Table 4.3 The query results using Brute-Force method and comparisons

Query	L1 distance
P(X_1|X_8, X_11)	0.000041
P(X_7, X_8, X_9, X_10, X_11|X_4)	0.179636
P(X_10, X_0)	0.000426

(e) Forward Sampling

We have designed the sample() subroutine using simple sampling method. For each variable, we sample it with the Bernoulli distribution $X_i \sim B(1, p)$, where $p = P\left(X_i \middle| Pa(X_i)\right)$, using the random function numpy.random.binomial() provided by the Numpy package. Then we store the sampled data into a file, and re-estimate the parameters with the sampled data. The changes of L1 distance after sampling and re-estimating is shown in Table 4.4.

Table 4.4 The L1 distance vs. sampling number

Sample Num.	L1 Distance
0	0.381109
10	0.381109
100	0.381111
1000	0.381113
10000	0.381092
20000	0.38109
30000	0.381071
40000	0.381071

The L1 distance is converged to 0.381071 after about 40,000 samples.

Problem 4.2 (a) A good elimination order will minimize the size of the maximal clique. For query 1, one good elimination order is [5, 6, 7, 8, 9, 10, 11, 2, 3, 4, 0], and the size of the maximal clique is 5. The variable elimination procedure is shown below.

$$P(X_1, X_8=1, X_{11}=1) = \sum_{X_0} P(X_0) \sum_{X_4} P(X_4|X_0) \sum_{X_3} P(X_3|X_0) \sum_{X_2} P(X_2|X_0) \sum_{X_{11}} P(X_{11}|X_1, X_2, X_3, X_4)$$
$$\cdot \sum_{X_{10}} P(X_{10}|X_1, X_2 X_3 X_4) \cdot \sum_{X_9} P(X_9|X_1 X_2 X_3 X_4) \cdot \sum_{X_8} P(X_8|X_1, X_2 X_3 X_4) \sum_{X_7} P(X_7|X_1 X_2 X_3 X_4) \cdot$$
$$\cdot \sum_{X_6} P(X_6|X_1, X_2, X_3, X_4) \cdot \sum_{X_5} P(X_5|X_1, X_2, X_3, X_4)$$

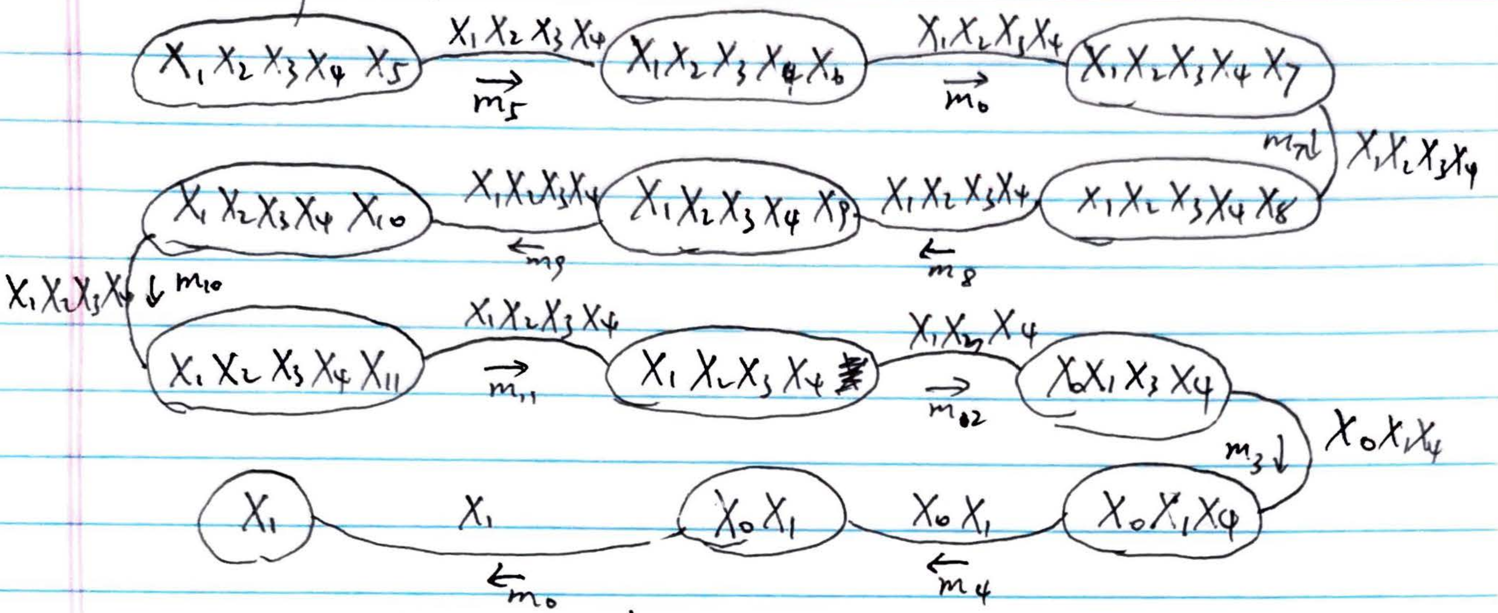
The intermediate factors are

$$m_5(X_1, X_2, X_3, X_4) = \sum_{X_5} P(X_5|X_1, X_2, X_3, X_4)$$
$$m_6(X_1, X_2, X_3, X_4) = \sum_{X_6} P(X_6|X_1, X_2, X_3, X_4) \cdot m_5(X_1, X_2, X_3, X_4)$$
$$m_7(X_1, X_2, X_3, X_4) = \sum_{X_7} P(X_7|X_1, X_2, X_3, X_4) \cdot m_6(X_1, X_2, X_3, X_4)$$
$$m_8(X_1, X_2, X_3, X_4) = \sum_{X_8} P(X_8|X_1, X_2, X_3, X_4) \cdot m_7(X_1, X_2, X_3 X_4)$$
$$m_9(X_1, X_2, X_3, X_4) = \sum_{X_9} P(X_9|X_1, X_2, X_3, X_4) \cdot m_8(X_1, X_2, X_3, X_4) \cdot \delta(X_8=1)$$
$$m_{10}(X_1, X_2 X_3, X_4) = \sum_{X_{10}} P(X_{10}|X_1, X_2 X_3, X_4) \cdot m_9(X_1, X_2, X_3 X_4)$$
$$m_{11}(X_1, X_2, X_3, X_4) = \sum_{X_{11}} P(X_{11}|X_1, X_2, X_3 X_4) \cdot m_{10}(X_1, X_2 X_3 X_4) \cdot \delta(X_{11}=1)$$
$$m_2(X_0, X_1, X_3, X_4) = \sum_{X_2} P(X_2|X_0)\, m_{11}(X_1, X_2, X_3, X_4)$$
$$m_3(X_0, X_1, X_4) = \sum_{X_3} P(X_3|X_0)\, m_2(X_0, X_1, X_3, X_4)$$
$$m_4(X_0, X_1) = \sum_{X_4} P(X_4|X_0) \cdot m_3(X_0, X_1, X_4)$$
$$m_0(X_1) = \sum_{X_0} P(X_0) \cdot m_4(X_0, X_1)$$

The time complexity depends on the ~~length~~ number of nodes and the time to calculate the factor of the max clique. The number of nodes is $L=12$, the max clique size is $M=5$, so the time complexity is $O(L \cdot 2^M) = 12 \times 2^5 t_0$, t_0 is the time of a single addition or multiplication operation.

The space complexity is the size to store the largest clique factor, which is $2^5 = 32$.

(b) (i) The clique tree corresponding to the order [5, 6, 7, 8, 9, 10, 11, 2, 3, 4, 0] is shown in Figure 4.3



The messages here are the same as the $m_5, \cdots, m_0$ in (a).
The time complexity is also the same, $O(L\cdot 2^M) = 12\times 2^5$ to.
The space complexity is also 32, determined by the size of the largest clique.

(ii) The factor here in the factor graph corresponding to the conditional probability in the original graphical model.
For example, the message from variables to factors are

$\mu_{X_5\to f(X_1X_2X_3X_4X_5)} = 1$ (extremal variable) $\mu_{X_6\to f(X_1,X_2X_3X_4,X_6)} = 1$

$\mu_{X_1\to f(X_0,X_1)} = \prod_{f_i\in ne(f)} \mu_{f_i\to v}(X_1) = \prod_{i\in\{5,\dots,11\}} \mu_{f(X_1,X_2X_3X_4,X_i)\to X_1}(X_1)$

The message from factors to variables are

$\mu_{f(X_1,X_2X_3X_4,X_5)\to X_1}(X_1) = \sum_{X_1X_2X_3X_4} P(X_5|X_1X_2X_3X_4)$ $\mu_{f(X_1X_2X_3X_4X_6)}(X_1) = \sum_{X_1X_2X_3X_4X_6} P(X_6|X_1X_2X_3X_4)$

$\mu_{f(X_1X_2X_3X_4X_7)\to X_1}(X_1) = \sum_{X_1X_2X_3X_4X_7} P(X_7|X_1X_2X_3X_4)$ (extremal)

The time complexity is $O(L\, 2^M) = 12\times 2^5$ and the space complexity is 12×32
Because we need to store all the factors.

(c) Implement Variable Elimination and Message Passing

I have implemented both variable elimination and message passing on clique tree in prob4_2.py.

The variable elimination algorithm is implemented based on the factor table data structure we have described in Problem 4. The intermediate factors are represented by 1-D factor tables. And the sum operation is implemented by sum the columns of these tables; the multiplication (product) is implemented by the inner product of the factor tables. For the observed variables, we multiply the message by a delta function, which is equal to 1 when the variable has the observed value and equal to 0 when the variable does not have the observed value. Based on these basic operations, we can eliminate the variables, generate the intermediate factors, and then normalize the final factor table to get the query results.

The procedures of message passing is similar to variable elimination. We have designed a new class Clique to represent the cliques in the clique tree. It stores a list of variables, the parent clique pointer, the children list, the separator list, and the message factor. Given the variable elimination order, the tree structure is already determined. Therefore, we have not implemented the moralizing, triangulating, or the max-weight spanning tree algorithm to construct the tree. Instead, we construct the tree according to the variable elimination order, and conduct the similar message passing procedures.

The intermediate messages of query 1 is shown in Figure 4.4.

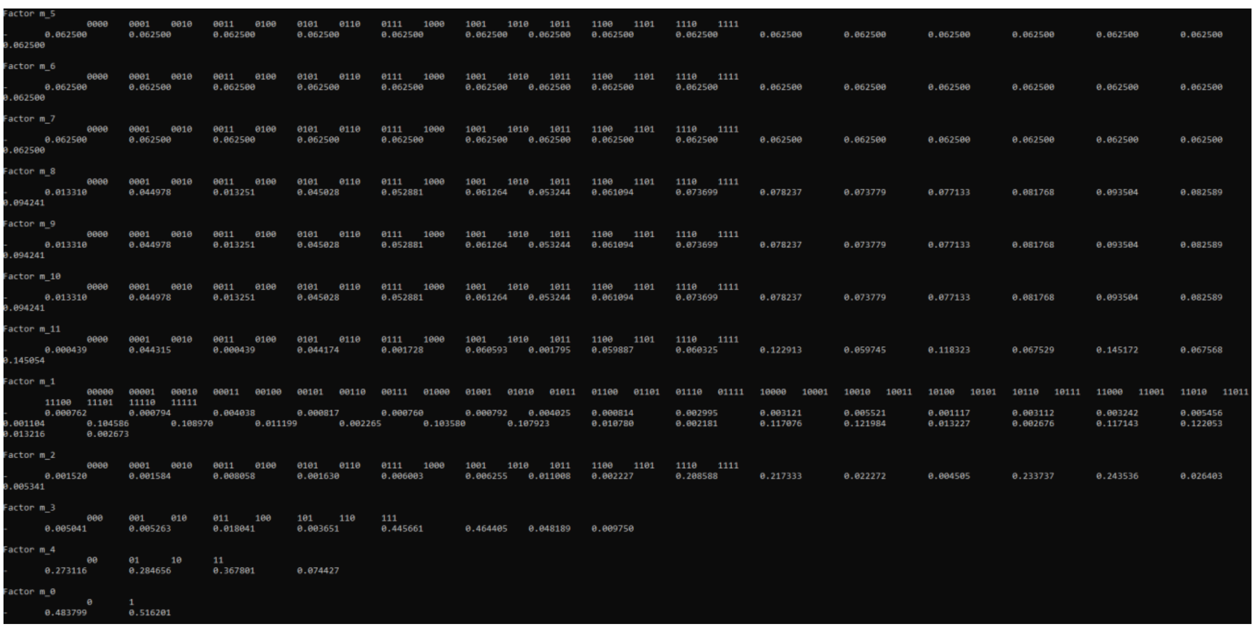


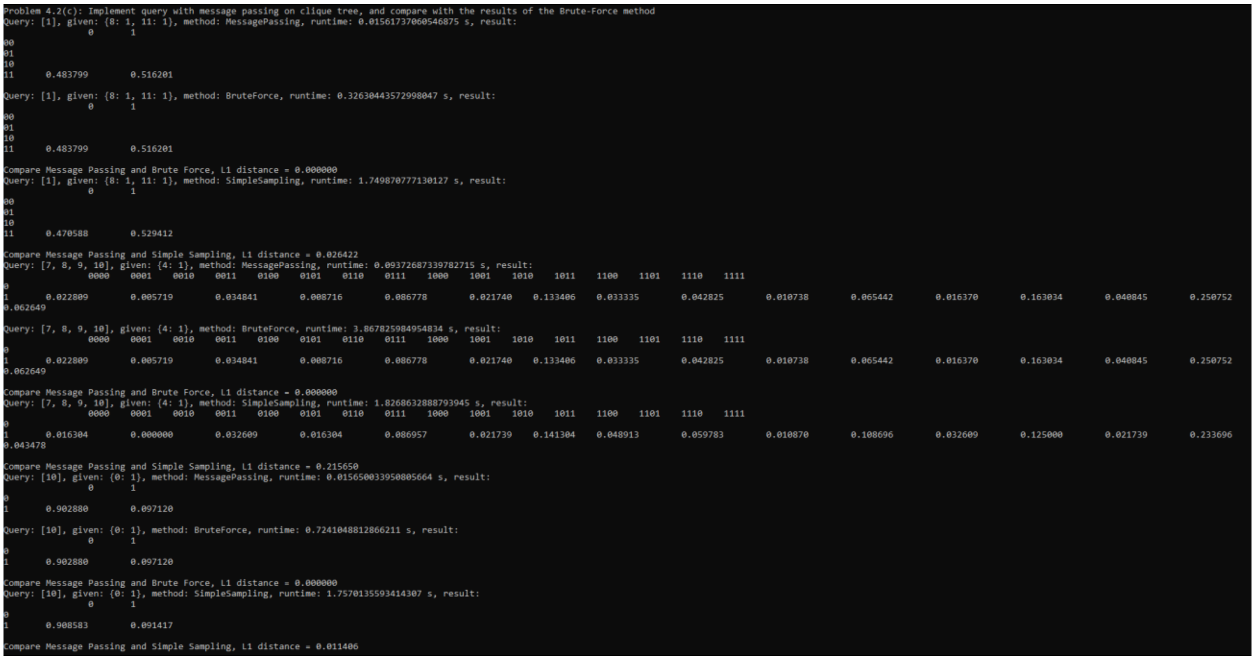
Figure 4.4 The intermediate messages of query 1

We have compared the 3 different query methods: Message Passing (MP), Brute Force (BF), and Simple Sampling (SS). The results is shown in Table 4.5.

Table 4.5 The results of comparisons of query methods

Query	L1 Dist. between MP and BF	L1 Dist. between MP and SS	Runtime of MP (s)	Runtime of BF (s)	Runtime of SS (s)
1	0	0.026422	0.015617371	0.326304436	0.015650034
2	0	0.21565	0.093726873	3.867825985	0.724104881
3	0	0.011406	1.749870777	1.826863289	1.757013559

The results of the queries is shown in Figure 4.5.



```
Problem 4.2(c): Implement query with message passing on clique tree, and compare with the results of the Brute-Force method
Query: [1], given: {8: 1, 11: 1}, method: MessagePassing, runtime: 0.01561737060546875 s, result:
                0       1
00
01
10
11      0.483799        0.516201

Query: [1], given: {8: 1, 11: 1}, method: BruteForce, runtime: 0.32630443572998047 s, result:
                0       1
00
01
10
11      0.483799        0.516201

Compare Message Passing and Brute Force, L1 distance = 0.000000
Query: [1], given: {8: 1, 11: 1}, method: SimpleSampling, runtime: 1.749870777130127 s, result:
                0       1
00
01
10
11      0.470588        0.529412

Compare Message Passing and Simple Sampling, L1 distance = 0.026422
Query: [7, 8, 9, 10], given: {4: 1}, method: MessagePassing, runtime: 0.09372687339782715 s, result:
                0000    0001    0010    0011    0100    0101    0110    0111    1000    1001    1010    1011    1100    1101    1110    1111
0
1       0.022809        0.005719        0.034841        0.008716        0.086778        0.021740    0.133406    0.033335        0.042825        0.010738        0.065442        0.016370        0.163034        0.040845        0.250752
0.062649

Query: [7, 8, 9, 10], given: {4: 1}, method: BruteForce, runtime: 3.867825984954834 s, result:
                0000    0001    0010    0011    0100    0101    0110    0111    1000    1001    1010    1011    1100    1101    1110    1111
0
1       0.022809        0.005719        0.034841        0.008716        0.086778        0.021740    0.133406    0.033335        0.042825        0.010738        0.065442        0.016370        0.163034        0.040845        0.250752
0.062649

Compare Message Passing and Brute Force, L1 distance = 0.000000
Query: [7, 8, 9, 10], given: {4: 1}, method: SimpleSampling, runtime: 1.8268632888793945 s, result:
                0000    0001    0010    0011    0100    0101    0110    0111    1000    1001    1010    1011    1100    1101    1110    1111
0
1       0.016304        0.000000        0.032609        0.016304        0.086957        0.021739    0.141304    0.048913        0.059783        0.010870        0.108696        0.032609        0.125000        0.021739        0.233696
0.043478

Compare Message Passing and Simple Sampling, L1 distance = 0.215650
Query: [10], given: {0: 1}, method: MessagePassing, runtime: 0.015650033950805664 s, result:
                0       1
0
1       0.902880        0.097120

Query: [10], given: {0: 1}, method: BruteForce, runtime: 0.7241048812866211 s, result:
                0       1
0
1       0.902880        0.097120

Compare Message Passing and Brute Force, L1 distance = 0.000000
Query: [10], given: {0: 1}, method: SimpleSampling, runtime: 1.7570135593414307 s, result:
                0       1
0
1       0.908583        0.091417

Compare Message Passing and Simple Sampling, L1 distance = 0.011406
```

Figure 4.5 The results of the queries

Obviously, MP gives the same results as BF, but it is the fastest. The runtime of SS is a constant.