Statistical Theories for Brain and Parallel Computing

Assignment No.6

17M18819

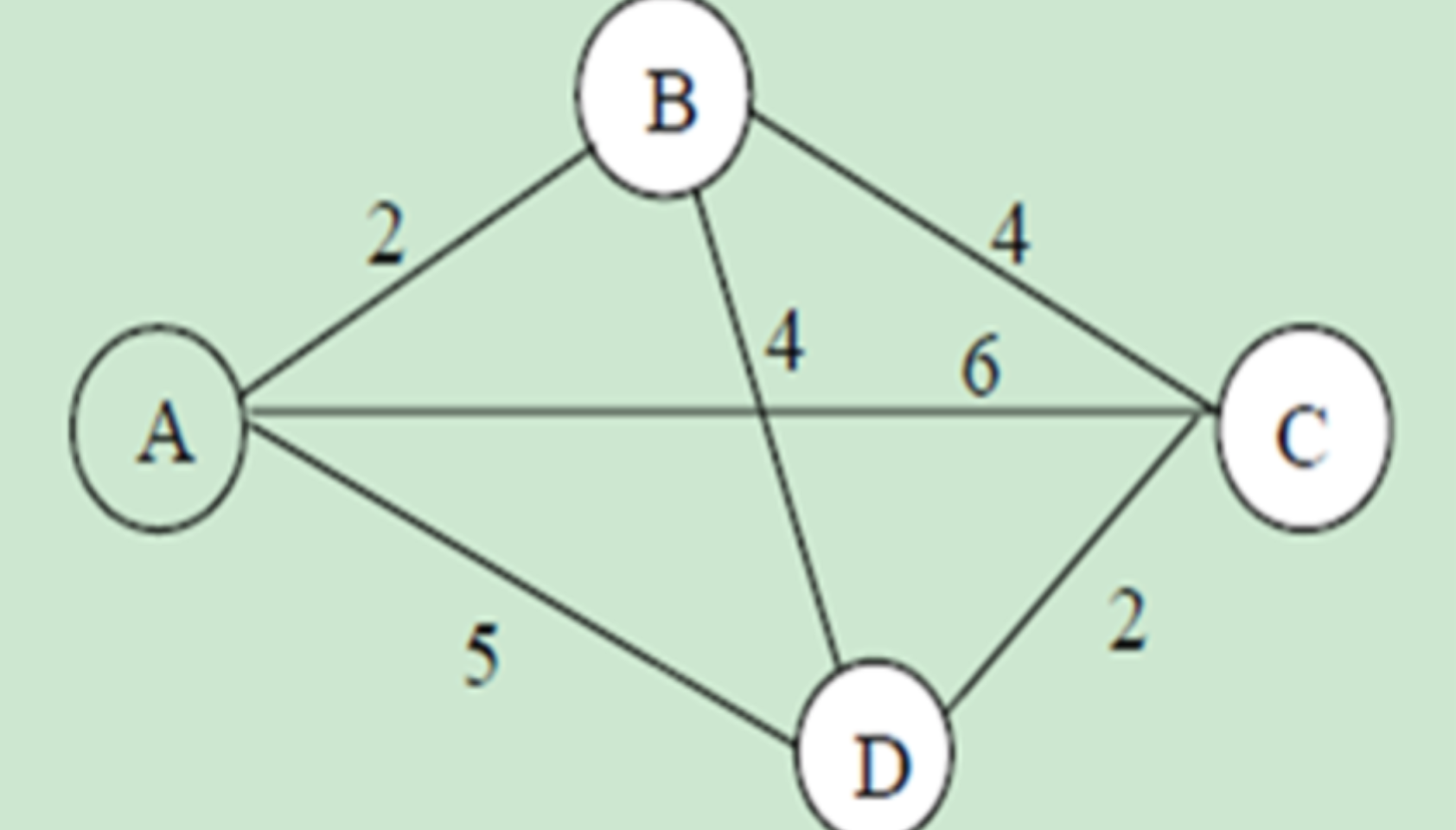
ZHU LICHENG

## Description of Assignment

In assignment 6, I need to construct a probabilistic and binary model to solve the salesman traveling problem, which is a very classic problem in computer optimization. We have many ad-hoc solutions to it like dynamic programming. However, with the increasing of number of cities, the computational cost becomes extremely large. Therefore, I would like to apply RNN to solve this problem. The detailed description of TSP is as follows:

*Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city.*

I applied my algorithm to solve this problem and the distances between each pair of cities is in the following picture:



## Solution

The most important part to solve this problem is to construct a proper RNN network by figuring out the weights and thresholds of this network.

Actually this question has many similarities with 4-queen problem, which was the 5th assignment since I also constructed a matrix to save the result and every column and row cannot own duplicated neurons of value 1. The result matrix is something like

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | City1 | City2 | City3 | City4 |
| City1 | 1 | 0 | 0 | 0 |
| City2 | 0 | 1 | 0 | 0 |
| City3 | 0 | 0 | 1 | 0 |
| City4 | 0 | 0 | 0 | 1 |

In the above matrix, Xij means ith city is visited for the with the jth order. Also, there is a distance matrix from every pair of cities. In our case, the distance matrix *d* is as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | City1 | City2 | City3 | City4 |
| City1 | 0 | 2 | 6 | 5 |
| City2 | 2 | 0 | 4 | 4 |
| City3 | 6 | 4 | 0 | 2 |
| City4 | 5 | 4 | 2 | 0 |

However, the energy of this RNN is different from 4-queen problem by adding the constraint of distance between each pair of cities in the matrix. Therefore, the numeric method to calculate weights and thresholds have changed accordingly as stated below:

**When k is equal to j + 1**

*Weight(ijnm) = -a\*djk + Alpha(nm) + Alpha(ij)- Beta(ijnm) - C (where a is a coefficient we should adjust later to achieve a balance between 4-queen constraint and country distance constraint)*

***When k is not equal to j + 1***

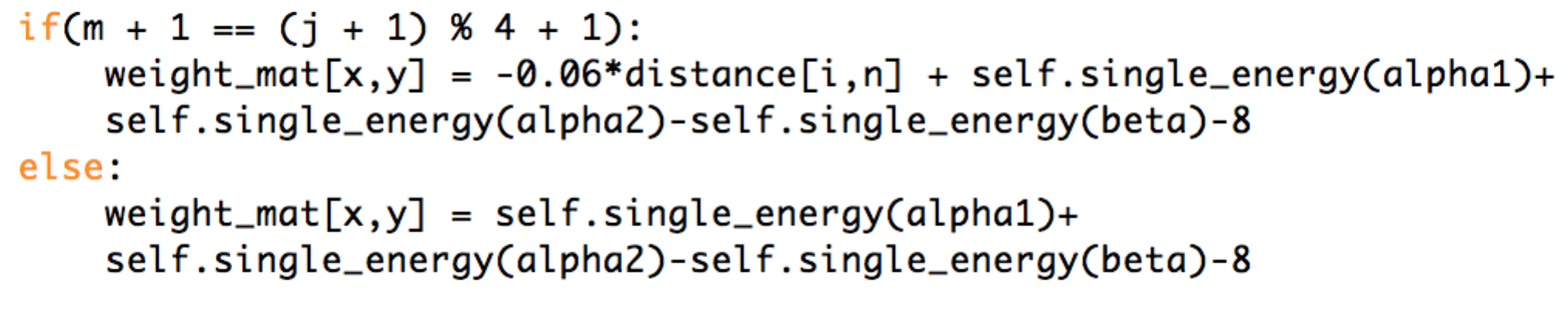
*Weight(ijnm) = Alpha(nm) + Alpha(ij)- Beta(ijnm) – C (which means if two city are not visited by neighboring order, the distance constraint should not have any effect on the energy and therefore the weight is the same as 4-queen problem)*

*Theta(ij) = E(ij = 1) – E(ij = 0)*

*Alpha(ij) = E(ij = 1)*

*Beta(ijnm) = E(ij = 1, nm = 1)*

The following is the implementation of weight calculation.



After we get the weights and thetas, we are able to calculate the energy of every state. The following picture shows how to calculate energy of a certain state. However, what differentiates 4-queen problem is that I used an identity matrix as an initial value matrix, which is

**[1, 0, 0, 0**

**0, 1, 0, 0**

**0, 0, 1, 0**

**0, 0, 0, 1]**

In addition, in order to retrieve the solution faster and also in case of local minima, I do not change one neuron every time. I change the order of two cities at a time. For example, at the state of identity matrix, I would like to update the order of 2nd city to 3rd. Then X[1,1] should be changed from 1 to 0 and X[1,2] should be changed from 0 to 1. However, since at every order, we can just accept one city to be visited, we need also to change the order of city, which was originally in the 3rd order to 2nd order. In our case, we need to change X[2,2] from 1 to 0 and change X[2,1] from 0 to 1. (**Note:**  The index description of matrix X starts from 0 and ends with 3 in program so I follow the rule here to make it clearer)

This process can be simply illustrated in the transition of two matrix as follows:

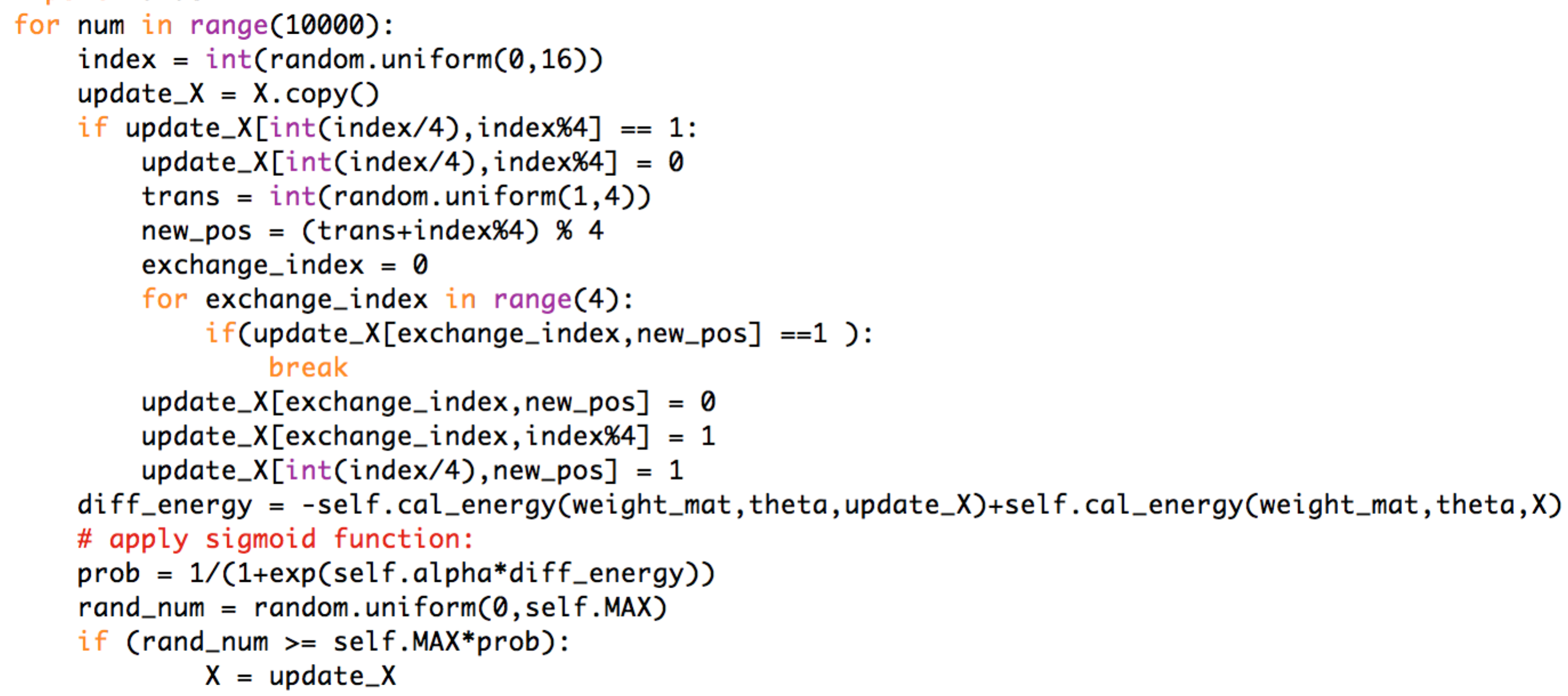
**[1, 0, 0, 0 [1, 0, 0, 0**

**0, 1, 0, 0 🡪 0, 0, 1, 0**

**0, 0, 1, 0 0, 1, 0, 0**

**0, 0, 0, 1]**  **0, 0, 1, 0]**

The following is the program implementation:

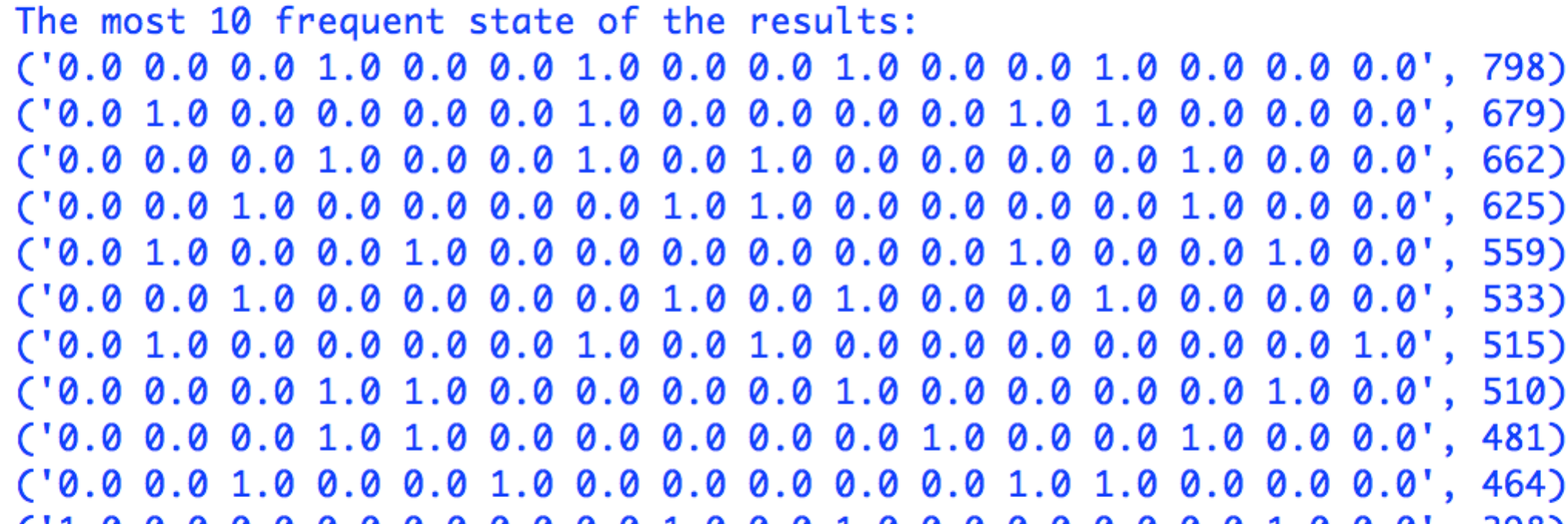


***Result***

I used python to implement this task. I created one class called **ass6.py** where we can call the function **ass6.prob\_rec()** to calculate the result of TSP problem. I ran the updating for 10000 times and showed the 10 most frequent occurring states.

I tried several times and adjusted the coefficient *a* in the weight computation equation to a reasonable value in order to balance the effect of 4-queen constraint and city distance constraint.

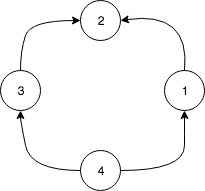
The following is the results of 10 most frequent states:



The first state should be the solution of our problem, let’s transfer it into a matrix representation:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | City1 | City2 | City3 | City4 |
| City1 | 0 | 0 | 0 | 1 |
| City2 | 0 | 0 | 1 | 0 |
| City3 | 0 | 1 | 0 | 0 |
| City4 | 1 | 0 | 0 | 0 |

Therefore, the visit order of four cities should be a circle:



We can set the starting point as 1, the visit order can be translated to 1-> 2 -> 3 -> 4. If we return to the distance matrix *d,* we can find the total distance is 2+4+2+5 = 13. This is actually the correct result of this TSP problem.

## Running command

**ass6.prob\_rec()**