In what follows, I will briefly discuss things I did to implement BM for solving TSP problems.

1. Optimization Objective Function:

I used Consensus function and defined a maximization problem to solve TSP. I used the following equation to calculate consensus value.

$$C = \sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_i x_j - \sum_{i=1}^{n} \theta_i x_i$$

2. Asynchronous Update:

I followed asynchronous update that is at a given iteration, a state is randomly picked and probabilistically updated based on the following acceptance criteria:

$$prob = \frac{1}{1+e^{\frac{-deltaC_i}{T}}}$$

$$rnd = np.random.random()$$

$$if\ rnd \leq prob\ then\ x_i = 1 - x_i$$

where T is current temperature and $deltaC_i$ is the change in the consensus if the state update happens. $deltaC_i$ is given by the following expression:
for a randomly picked node x_i:

$$s_i = \sum_{j\neq i}^{n} w_{ij}x_j + \theta_i$$
$$deltaC_i = (1 - 2 * x_i) * s_i$$

3. Defining Weights:

The weights are defined as follows in order to capture the constraints in the TSP problem

Distance between the city:

$$W_{(c_{in},c_{jm})} = -d_{ij}\ \ where\ i \neq j$$
$$,n \neq m\ and\ d_{ij}\ is\ the\ distnce\ between\ city\ i\ and\ j$$

Penalties:

$$W_{(c_{in},c_{jm})} = -p\ \ where\ i = j\ or\ n = m\ and\ p > 0$$

Biases:

$$\theta_i = \mu_i\ for\ city\ i\ and\ \mu_i\ emperically\ determined\ as\ explained\ below$$

I experimented number of times by assigning different value for p and μ_i. Generally, I obtained better result whenever p > μ_i and μ_i > the average of all possible distance that connects city i to other cities. Finally, after many experimentation and statistical analysis, I found that:

$$p = 150\ and$$
$$Biases = \{'A': 95, 'B': 105, 'C': 113, 'D': 140, 'E': 115\}$$

Gave me much better result.

4. **The Sixth Epoch:**

   To enforce the requirement that the sales man should return to the same city he started from, I implemented hard constraint where I forced the sixth epoch to agree with first epoch in terms of the name of the city.

   Initially, I tried to enforce this constraint softly by assigning negative weight whenever the sixth city is different from the first city. However, it required me very high number of iterations to get fairly good result. Hence, I resorted to enforcing it forcibly as explained above.

5. **Cooling schedule:**

   I have used a simple cooling strategy in which I started from very high temperature T = 1000 and reduce it by 5% (T =).95*T) between each steps. At each step (temperature), I carried out Boltzmann probabilistic update 10,000 times.

6. **Result Trace File:**

   The actual output trace file from the program execution is too big. I am presenting here only part of it for three different experiments. I will be showing the initial state, what happens at early stage, what happens at the later stage of iteration (as T -> 0).

   Experiment 1:

```
**********Initilized**********
['A1', 'D2', 'C3', 'C4', 'C5', 'A6']
********** after 10000 random updates at 1000.000000 temp **********
['B1', 'E1', 'C1', 'E2', 'D2', 'B3', 'A3', 'C3', 'B4', 'E5', 'D5', 'C6', 'B6', 'E6'] : 249
********** after 10000 random updates at 950.000000 temp **********
['E1', 'C1', 'C2', 'D2', 'A3', 'D3', 'B4', 'C5', 'C6', 'E6'] : 114

********** after 10000 random updates at 9.888365 temp **********
['A1', 'E2', 'B3', 'C4', 'D5', 'A6'] : 73
********** after 10000 random updates at 9.393946 temp **********
['A1', 'B3', 'E4', 'D5', 'A6'] : 60
********** after 10000 random updates at 8.924249 temp **********
['A1', 'D2', 'B3', 'C4', 'E5', 'A6'] : 86
********** after 10000 random updates at 8.478037 temp **********
['A1', 'D2', 'B3', 'C4', 'E5', 'A6'] : 86
********** after 10000 random updates at 8.054135 temp **********
['A1', 'D2', 'B3', 'C4', 'E5', 'A6'] : 86
********** after 10000 random updates at 7.651428 temp **********
['A1', 'D2', 'B3', 'C4', 'E5', 'A6'] : 86
********** after 10000 random updates at 7.268857 temp **********
['A1', 'D2', 'E3', 'B4', 'A6'] : 60
********** after 10000 random updates at 6.905414 temp **********
['A1', 'D2', 'C3', 'B4', 'E5', 'A6'] : 73
********** after 10000 random updates at 6.560143 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 6.232136 temp **********
```

['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 5.920529 temp **********
********** after 10000 random updates at 3.544840 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 3.367598 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 3.199218 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 3.039257 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 2.887294 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 2.742929 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 0.619725 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 0.588739 temp **********
['A1', 'D2', 'C3', 'E4', 'B5', 'A6'] : 66
********** after 10000 random updates at 0.559302 temp **********
['B1', 'E3', 'A4', 'B5', 'C5', 'B6'] : 68

Experiment 2:

**********Initilized**********
['E1', 'D2', 'B3', 'B4', 'E5', 'E6']

********** after 10000 random updates at 9.888365 temp **********
['E1', 'B2', 'C3', 'A5', 'E6'] : 63
********** after 10000 random updates at 9.393946 temp **********
['D1', 'B2', 'E3', 'C4', 'A5', 'D6'] : 83
********** after 10000 random updates at 8.924249 temp **********
['D1', 'B2', 'C3', 'E4', 'A5', 'D6'] : 86
********** after 10000 random updates at 8.478037 temp **********
['D1', 'B3', 'C4', 'A5', 'D6'] : 72
********** after 10000 random updates at 8.054135 temp **********
['D1', 'A2', 'E3', 'C4', 'B5', 'D6'] : 86
********** after 10000 random updates at 7.651428 temp **********
['D1', 'C2', 'E4', 'A5', 'D6'] : 64
********** after 10000 random updates at 7.268857 temp **********
['D1', 'C2', 'B3', 'A5', 'D6'] : 55
********** after 10000 random updates at 6.905414 temp **********

['D1', 'C3', 'E4', 'A5', 'D6'] : 64
********** after 10000 random updates at 6.560143 temp **********
['D1', 'B2', 'E3', 'C4', 'A5', 'D6'] : 83
********** after 10000 random updates at 4.581193 temp **********
['D1', 'B2', 'E3', 'C4', 'A5', 'D6'] : 83
********** after 10000 random updates at 4.352133 temp **********
['D1', 'E3', 'C4', 'A5', 'D6'] : 76
********** after 10000 random updates at 4.134526 temp **********
['D1', 'E3', 'C4', 'A5', 'D6'] : 76
********** after 10000 random updates at 3.927800 temp **********
['D1', 'B2', 'C4', 'A5', 'D6'] : 72
********** after 10000 random updates at 3.731410 temp **********
['D1', 'C2', 'B4', 'A5', 'D6'] : 55
********** after 10000 random updates at 2.605783 temp **********
['D1', 'C3', 'B4', 'A5', 'D6'] : 55
********** after 10000 random updates at 2.475494 temp **********
********** after 10000 random updates at 1.089531 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 1.035054 temp **********
['D1', 'C3', 'B4', 'A5', 'D6'] : 55
********** after 10000 random updates at 0.983302 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 0.934136 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 0.887430 temp **********
['D1', 'C3', 'B4', 'A5', 'D6'] : 55
********** after 10000 random updates at 0.843058 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 0.800905 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 0.652342 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 0.619725 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81
********** after 10000 random updates at 0.588739 temp **********
['D1', 'E2', 'C3', 'B4', 'A5', 'D6'] : 81

Experiment 3:

**********Initilized BM **********
randomly picked initial tour ['D1', 'E2', 'A3', 'B4', 'A5', 'D6']
********** after 10000 random updates at 1000.000000 temp **********
['C1', 'D1', 'E1', 'E2', 'D2', 'A2', 'E4', 'B4', 'B5', 'A5', 'E6', 'D6', 'C6'] : 216
********** after 10000 random updates at 950.000000 temp **********
['E1', 'B2', 'D2', 'D3', 'A3', 'A4', 'B4', 'E5', 'C5', 'E6'] : 99

********** after 10000 random updates at 540.360088 temp **********
['C1', 'A1', 'E1', 'B2', 'C2', 'C3', 'B3', 'E4', 'B4', 'C5', 'A6', 'E6', 'C6'] : 167
********** after 10000 random updates at 115.982221 temp **********
['A1', 'B2', 'D2', 'A6'] : 47
********** after 10000 random updates at 110.183110 temp **********
['B1', 'A1', 'D2', 'B4', 'D5', 'A6', 'B6'] : 94
********** after 10000 random updates at 9.888365 temp **********
['D1', 'A3', 'B4', 'C5', 'D6'] : 55
********** after 10000 random updates at 9.393946 temp **********
['D1', 'B2', 'A3', 'C4', 'E5', 'D6'] : 113
********** after 10000 random updates at 8.924249 temp **********
['D1', 'E2', 'A3', 'C4', 'D6'] : 98
********** after 10000 random updates at 6.232136 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 5.920529 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 5.624503 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 5.343278 temp **********
['D1', 'A2', 'E3', 'C4', 'B5', 'D6'] : 86
********** after 10000 random updates at 5.076114 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 4.822308 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 4.581193 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 4.352133 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 4.134526 temp **********
['D1', 'A2', 'B4', 'C5', 'D6'] : 55
********** after 10000 random updates at 3.927800 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 3.731410 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 3.544840 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 3.367598 temp **********
['D1', 'A2', 'B4', 'C5', 'D6'] : 55
********** after 10000 random updates at 3.199218 temp **********
['D1', 'A2', 'B3', 'C5', 'D6'] : 55
********** after 10000 random updates at 3.039257 temp **********
['D1', 'A2', 'B3', 'E4', 'C5', 'D6'] : 66
********** after 10000 random updates at 2.234133 temp **********
['D1', 'A2', 'B4', 'C5', 'D6'] : 55

********** after 10000 random updates at 2.122426 temp **********
['D1', 'A2', 'B4', 'C5', 'D6'] : 55
********** after 10000 random updates at 2.016305 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 1.915490 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 1.146875 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 0.800905 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 0.760860 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 0.722817 temp **********
['D1', 'A2', 'B4', 'C5', 'D6'] : 55
********** after 10000 random updates at 0.686676 temp **********
['D1', 'A2', 'B4', 'C5', 'D6'] : 55
********** after 10000 random updates at 0.652342 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 0.619725 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73
********** after 10000 random updates at 0.588739 temp **********
['D1', 'A2', 'E3', 'B4', 'C5', 'D6'] : 73

7. Appendix

Python Code:

The python code can be found below.

```
import itertools as itr
import numpy as np
import math
Dis = {}
# List of cities and epochs
city = ['A','B','C','D','E']
epoch = [1,2,3,4,5,6]

#dictionary holding the distance between cities.
Dis["AB"] = 10
Dis["BA"] = 10
Dis["AC"] = 20
Dis["CA"] = 20
Dis["AD"] = 5
Dis["DA"] = 5
Dis["AE"] = 18
Dis["EA"] = 18
Dis["BC"] = 15
Dis["CB"] = 15
Dis["BD"] = 32
Dis["DB"] = 32
Dis["BE"] = 10
Dis["EB"] = 10
Dis["CD"] = 25
Dis["DC"] = 25
Dis["CE"] = 16
Dis["EC"] = 16
Dis["DE"] = 35
Dis["ED"] = 35
# a list for creating labels for the Boltzmann machine nodes
#by cross product between city name and epochs. we will have total
# of 30 nodes

BM_node_label = list(itr.product(city,epoch))
print('******** the BM_idx details ************')
print(BM_node_label)
print(len(BM_node_label))
# create a list labeling the weights(edges between nodes) by cross
#joining BM_node_label above to itself.
#a Weight is labeled by the name of the two nodes it is connecting.
#there will be 30*30 weights at this point. The 30 self-connecting
#weights will be removed later.
BM_wt_label = list(itr.product(BM_node_label,BM_node_label))
print(len(BM_wt_label))
```

```
# the following is a dictionary holding the biases associated to each
#city.
# I will explain how I chose those values in the report

Bias = {'A':95,'B':105,'C':113,'D':140,'E':115}

#In what follows I set the weight values. I have assigned appropriate
#weight values
# to enforce the TS constraints.

BM_wt = {}
for (node1,node2) in BM_wt_label:
    if(node1 == node2): # no self connection
        pass
    #defining the weight between first epoch node and sixth epoch
node.
    #the weight is set to zero indicating that we need first and last
epoch to be the same.
    elif(node1[0] == node2[0] and ((node1[1] ==1 and node2[1] == 6) or
(node1[1] ==6 and node2[1] == 1) )):
        BM_wt[(node1,node2)] = 0
    # below weight definition enforces that the sales man should not
travel to same city more than once.
    elif(node1[0] == node2[0]):
        BM_wt[(node1,node2)] = -150
    # Below weight definition enforces that the sales man cannot be in
two cities at the same time
    elif(node1[1] == node2[1]):
        BM_wt[(node1,node2)] = -150
    # the following weight defines actual distance between the cities.
    else:
        BM_wt[(node1,node2)] = -1 * Dis[node1[0]+node2[0]]

print('************ weight *********')
#print(BM_wt)

#intialize temprature to high value
T = 1000.0
#So far we are done creating BM. Next, we will start solving the
optimization problem.
#1. The first step would be assigning initial state to BM nodes. This
is done in two steps:
# Initialize all state to zero
BM ={}
for n in BM_node_label:
    BM[n] =0
#For each epoch 1 to 5, pick one of the node and set it to 1.
#This will create a valid initial tour.
```

```
for i in range(1,6):
    rnd = np.random.randint(1,6) # generate a rondom number from 1 to
5
    #identifyinh which city to turn on based on the random number
    e = 'A' if rnd == 1 else 'B' if rnd == 2 else 'C' if rnd == 3 else
'D' if rnd == 4 else 'E'
    #the below routine is used to enforce that the city to set in
epoch 6
    #should be the same as the city set on in the first epoch
    if i == 1:
        epoch1 = e
    # turn on the randomly selected node of current epoch
    BM[(e,i)] = 1
#The following makes sure that the sales man returns to the same city
he starts.
# This can be considered as hard constraint of the optimization
problem unlike others constraints
#which are implemented softly through picking right value of weight.
BM[epoch1,6] = 1


print('************Initialized BM *************')
tour = ['%s%d' %(x,y) for (x,y) in sorted(BM,key = lambda x: x[1]) if
BM[(x,y)] == 1]
print('randomly picked initial tour %s ' % tour)

# Once the BM is initialized, now we are ready to start the annealing
process
# Below is a function that compute the probability of acceptance for a
node
# Input parameters:
# node: is the node selected for update
# BM_wt: is the weight
# T: is the temperature
def acceptance(node,BM_wt,T):
    #variable for holding activity value
    s = 0
    #compute the activity value of the node
    for (n1,n2) in BM_wt:
        if(n2 == node ):
            s = s+BM_wt[(n1,n2)]*BM[n1]
    # compute the change in Consensus based on activity value
    # and bias
    deltaC  = (1 - 2*BM[node]) *(s + Bias[node[0]])
    #compute the probability of acceptance. If deltaC/T is too big,
    # which happens when T->0, there will be OverflowError hence
    #set p = 1 when (deltaC/T) -> very large positive number
    try:
        p = 1/(1 + math.exp(-1.0*deltaC/T))
    except OverflowError:
```

```
        p =1
    return p

# Following is the routine that does the annealing process. It will
#randomly pick a node and do the probabilistic update
# again, we only consider the first 5 epochs and enforce the sixth
epoch
# to agree to the first epoch.
# get the list of nodes corresponding to epochs 1 to 5.
# there will be 25 nodes.
epoch1_5 = [n for n in BM_node_label if n[1] !=6]
# Do the annealing. For each temperature, carry out the node update
10,000 times
# then reduce T by 5% and repeat the process.
while (T > 0.5):
    for itr in range(10000):
        # randomly pick a node
        idx = np.random.randint(len(epoch1_5))
        node = epoch1_5[idx]
        #compute the acceptance probability
        p = acceptance(node,BM_wt,T)
        # do probabilistic update
        if(np.random.random() <= p ):
            BM[node] = 1 - BM[node]
            #Hard constraint: sales man should return to the same
city.
            if(node[1] == 1):
                BM[(node[0], 6)] = BM[node] # forcing 6th epoch as the
first
        else:
            pass
    # printing the final tour and tout length at each temprature
    print('*************** after 10000 random updates at %f temp
************' % T)
    tour = ['%s%d' %(x,y) for (x,y) in sorted(BM,key = lambda x: x[1])
if BM[(x,y)] == 1]
    path =[x for (x,y) in sorted(BM,key = lambda x: x[1]) if BM[(x,y)]
== 1]
    r = [path[i] + path[i+1] for i in range(len(path)) if i !=
len(path)-1]
    distance = 0
    for pr in r:
        distance += Dis.get(pr,0)
    print('%s : %d' %(str(tour), distance))
    T = 0.95*T
```