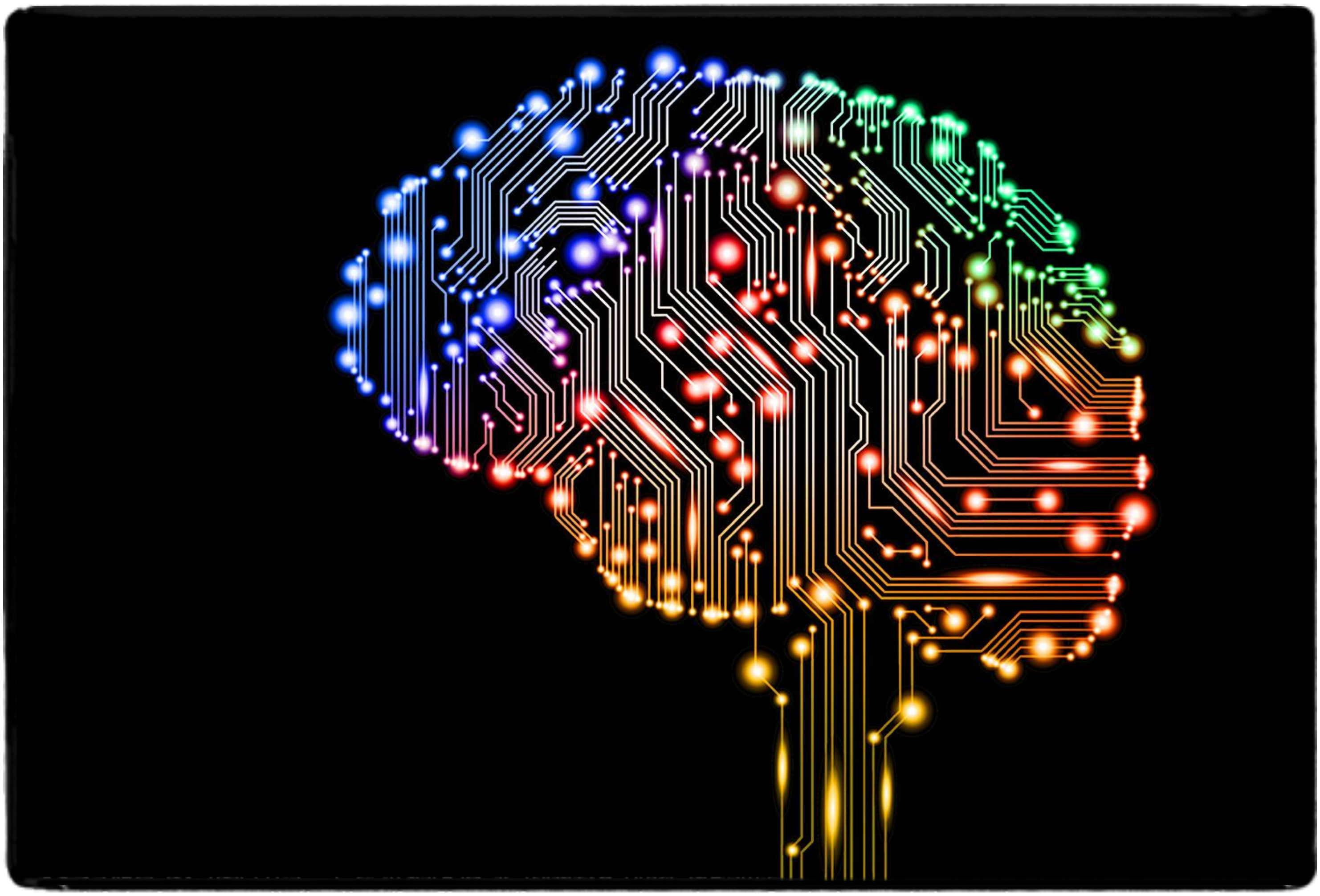
Artificial Intelligence
Programming HW#2



Mehdi Safaee

prelude :

I've used python3 for this problems.

for each of the questions , I've provided 5 pieces of code :
- node.py : to simplify making graphs and tree's (python doesnt have a built in data structure for them)
- problem.py : where I've defined each problem
- algorithm.py : where I've provided the algorithms
- main.py : the main file of code which I've gathered all others together

To run each code , simply run the following command in your terminal :

python3 main.py

1st Question :

I've made the following graph :

```python
class Problem(object):
    def __init__(self,colors):
        nodes = []
        self.population = 12
        self.colors = colors
        for i in range(0,self.population) :
            x = Node(str(i))
            nodes.append(x)
        Node.connect(nodes[0] , nodes[1])
        Node.connect(nodes[0] , nodes[2])
        Node.connect(nodes[1] , nodes[2])
        Node.connect(nodes[1] , nodes[3])
        Node.connect(nodes[1] , nodes[5])
        Node.connect(nodes[2] , nodes[4])
        Node.connect(nodes[3] , nodes[4])
        Node.connect(nodes[3] , nodes[8])
        Node.connect(nodes[4] , nodes[7])
        Node.connect(nodes[5] , nodes[6])
        Node.connect(nodes[6] , nodes[8])
        Node.connect(nodes[6] , nodes[9])
        Node.connect(nodes[6] , nodes[11])
        Node.connect(nodes[7] , nodes[8])
        Node.connect(nodes[8] , nodes[10])
        Node.connect(nodes[9] , nodes[11])
        Node.connect(nodes[10] , nodes[11])
        self.state = nodes
```

gave it 4 colors to colorize it , using 3 different hill climbing algorithms.
gave it runtime of 10 seconds (so it wont continue after that)

to run the codes , simply uncomment each needed :

```
p = Problem(colors=4)
s = algorithm(p)
s.stochasticHillClimbingSearch(runningTime=10)
# s.firstBestHillClimbingSearch(runningTime=10)
# s.randomRestartHillClimbingSearch(runningTime=10)
```

here are my results for each one :

stochastic :

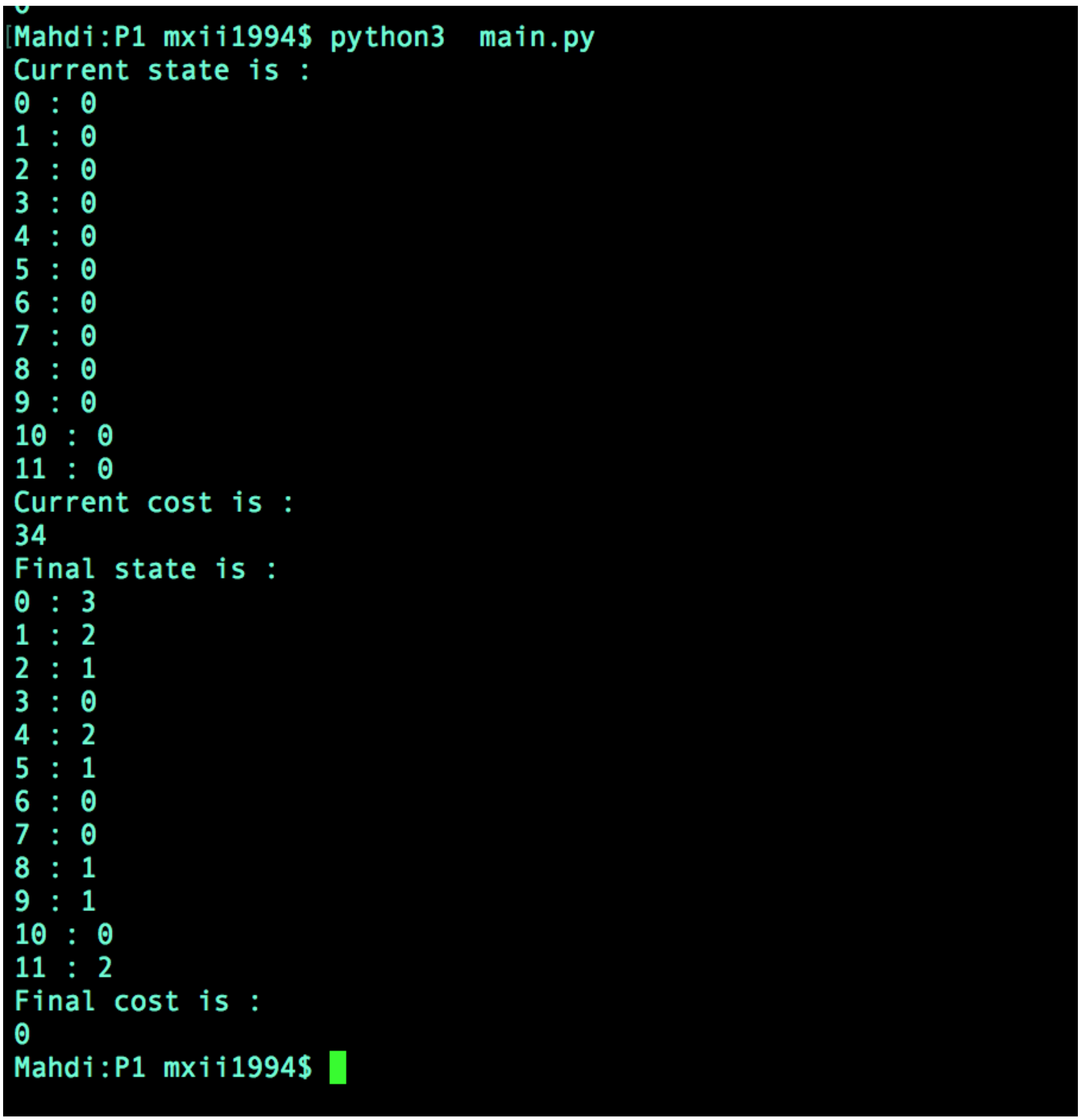
here are my results for each one :

stochastic :

```
Mahdi:P1 mxii1994$ python3  main.py
Current state is :
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 0
9 : 0
10 : 0
11 : 0
Current cost is :
34
Final state is :
0 : 3
1 : 0
2 : 1
3 : 3
4 : 2
5 : 3
6 : 2
7 : 0
8 : 1
9 : 1
10 : 3
11 : 0
Final cost is :
0
Mahdi:P1 mxii1994$
```

here are my results for each one :

firstBest :



```
Mahdi:P1 mxii1994$ python3  main.py
Current state is :
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 0
9 : 0
10 : 0
11 : 0
Current cost is :
34
Final state is :
0 : 3
1 : 2
2 : 1
3 : 0
4 : 2
5 : 1
6 : 0
7 : 0
8 : 1
9 : 1
10 : 0
11 : 2
Final cost is :
0
Mahdi:P1 mxii1994$
```

here are my results for each one :

randomRestart :

```
Mahdi:P1 mxii1994$ python3  main.py
Current state is :
0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 0
9 : 0
10 : 0
11 : 0
Current cost is :
34
Final state is :
0 : 3
1 : 0
2 : 2
3 : 3
4 : 0
5 : 2
6 : 3
7 : 3
8 : 0
9 : 1
10 : 3
11 : 3
Final cost is :
4
number of restarts :
527
Mahdi:P1 mxii1994$
```

problem 2 :

for this problem , I've provided 3 SA functions :

```
def SA(self , mode=0 , runningTime=10 , simulatedAnnealingTrials=10000):
    trials = simulatedAnnealingTrials
    annealingFunction = []
    if(mode == 0):
        for i in range(2,trials+2):
            annealingFunction.append(1/i)
    elif(mode == 1):
        difference = 1/trials
        distance = 1
        while distance > 0:
            distance = distance - difference
            annealingFunction.append(distance)
    elif(mode == 2):
        for i in range(1,trials+1):
            annealingFunction.append(1/2*i)
```

for mode = 0 (simple hemographic function)

```
__pycache__      algorithm.py     main.py          node.py          problem.py
[Mahdi:P2 mxii1994$ python3 main.py
k
o
p

t
l
m

b
l
a

u
o
c

final cost is (number of missmatches in dictionary) : 4005
Mahdi:P2 mxii1994$
```

for mode = 1 and 2 (linear and x2 hemographic function)

```
p
t
m

b
l
a

k
c
o

l
u
o

final cost is (number of missmatches in dictionary) : 4006
[Mahdi:P2 mxii1994$ python3 main.py
c
u
t

m
l
a

l
b
p

o
k
o

final cost is (number of missmatches in dictionary) : 4009
Mahdi:P2 mxii1994$ 4
```

the simple hemographic function was most near to ideal between these functions.

Problem 3

I've Implemented this problem as the definition of genetic algorithm.

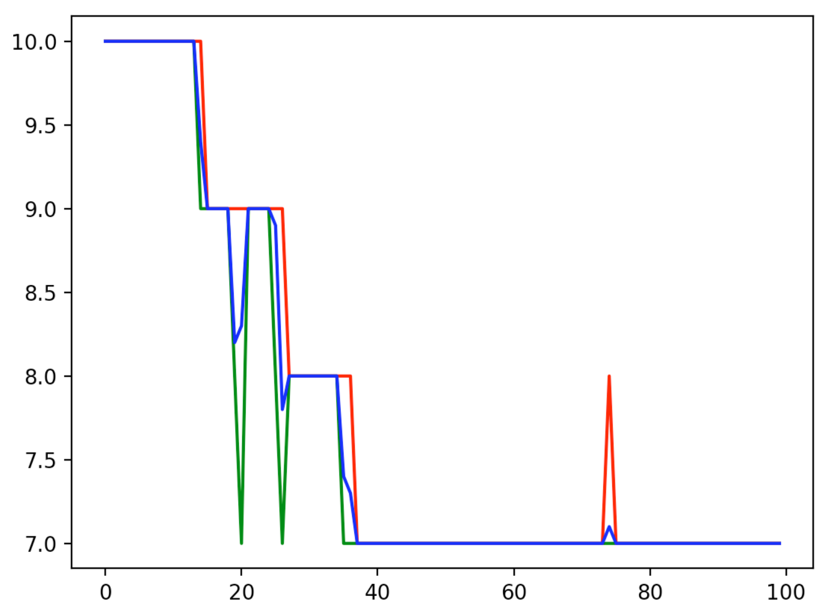
givving these as input :

```python
from problem import Problem
from algorithm import Algorithm

p = Problem(population=10)
a = Algorithm(p,crossOverRate=0.4 , mutationRate=0.1, trials=100)
chromosomes = a.genetics()
for c in chromosomes:
    print(c)

```

here is my output graph.
the cost from current population to target is reduced from 10 to 7 during 100 generations



for this problem, I've defined each chromosome to be seperated into 2 13xchar parts and defined the cost function after, according to problems definiton.
everything was simple after that.