Neeraj Vashistha (190573735)

ECS759P: ARTIFICIAL INTELLIGENCE

2019/20 – Semester 2 - Coursework 1

Question 1.Let's Travel!

Instead of priority queue, the logic has been implemented using list. And the pseudo code is as follows:

```
function ucs(start_location,destination,graph)
      frontier ← node(start_location), a list of nodes
      explored ← a set of visited nodes
      loop do
            if frontier is empty then return failure
            node ← frontier[0],
            frontier.remove(node)
            if node_state is not visited
            then
                  child_node_list[node] ← graph[node]  # depth 1 evaluation
                  if node_state is not destination
                  then
                        child_nodes ← evaluate(child_node_list)
                        frontier ← frontier + child_nodes
                        frontier ← sort(frontier , path_cost)
                  explored ← explored + node
                  if node_state is destination
                  then
                        previous_nodes_list ← get_previous_node(node)
                        cummulative_cost ← get_cummlative_cost(previous_nodes_list)
                        return previous_nodes_list, cummulative_cost
```

Q1.a.

```
ucf(graph,'london','aberdeen')

frontier =  [('london', '0.0', ['london'])]
Frontier selected & Child nodes of london are
Exploring all frontiers =  [('london', '0.0', [('birmingham', 1, 110.0), ('brighton', 1, 52.0), ('bristol', 1, 116.0),
('cambridge', 1, 54.0), ('cardiff', 1, 161.0), ('carlisle', 1, 302.0), ('dover', 1, 71.0), ('exeter', 1, 172.0),
('glasgow', 1, 396.0), ('hull', 1, 172.0), ('leeds', 1, 198.0), ('liverpool', 1, 198.0), ('oxford', 1, 57.0)])]
Sorting frontier based on cost  =  [('london', '0.0', [('brighton', 1, 52.0), ('cambridge', 1, 54.0), ('oxford', 1, 57.0),
('dover', 1, 71.0), ('birmingham', 1, 110.0), ('bristol', 1, 116.0), ('cardiff', 1, 161.0), ('exeter', 1, 172.0), ('hull',
1, 172.0), ('leeds', 1, 198.0), ('liverpool', 1, 198.0), ('carlisle', 1, 302.0), ('glasgow', 1, 396.0)])]
explored = {'london'}
destination = aberdeen
*************************
frontier =  [('brighton', '52.0', ['london'])]
Frontier selected & Child nodes of brighton are
Exploring all frontiers =  [('brighton', '52.0', [('cambridge', 1, 54.0), ('oxford', 1, 57.0), ('dover', 1, 71.0),
('birmingham', 1, 110.0), ('bristol', 1, 116.0), ('cardiff', 1, 161.0), ('exeter', 1, 172.0), ('hull', 1, 172.0), ('leeds',
1, 198.0), ('liverpool', 1, 198.0), ('carlisle', 1, 302.0), ('glasgow', 1, 396.0), ('aberystwyth', 2, 301.0),
('birmingham', 2, 211.0), ('bristol', 2, 188.0), ('cambridge', 2, 158.0), ('cardiff', 2, 235.0), ('carlisle', 2, 406.0),
('dover', 2, 133.0), ('nottingham', 2, 230.0), ('oxford', 2, 148.0), ('penzance', 2, 329.0), ('portsmouth', 2, 101.0),
('sheffield', 2, 268.0), ('swansea', 2, 288.0), ('york', 2, 302.0), ('london', 2, 104.0)])]
Sorting frontier based on cost  =  [('brighton', '52.0', [('cambridge', 1, 54.0), ('oxford', 1, 57.0), ('dover', 1, 71.0),
('portsmouth', 2, 101.0), ('london', 2, 104.0), ('birmingham', 1, 110.0), ('bristol', 1, 116.0), ('dover', 2, 133.0),
('oxford', 2, 148.0), ('cambridge', 2, 158.0), ('cardiff', 1, 161.0), ('exeter', 1, 172.0), ('hull', 1, 172.0), ('bristol',
2, 188.0), ('leeds', 1, 198.0), ('liverpool', 1, 198.0), ('birmingham', 2, 211.0), ('nottingham', 2, 230.0), ('cardiff', 2,
235.0), ('sheffield', 2, 268.0), ('swansea', 2, 288.0), ('aberystwyth', 2, 301.0), ('carlisle', 1, 302.0), ('york', 2,
302.0), ('penzance', 2, 329.0), ('glasgow', 1, 396.0), ('carlisle', 2, 406.0)])]
explored = {'brighton', 'london'}
destination = aberdeen
*************************
frontier =  [('cambridge', '54.0', ['london', 'brighton'])]
Frontier selected & Child nodes of cambridge are
Exploring all frontiers =  [('cambridge', '54.0', [('oxford', 1, 57.0), ('dover', 1, 71.0), ('portsmouth', 2, 101.0),
('london', 2, 104.0), ('birmingham', 1, 110.0), ('bristol', 1, 116.0), ('dover', 2, 133.0), ('oxford', 2, 148.0),
('cambridge', 2, 158.0), ('cardiff', 1, 161.0), ('exeter', 1, 172.0), ('hull', 1, 172.0), ('bristol', 2, 188.0), ('leeds',
1, 198.0), ('liverpool', 1, 198.0), ('birmingham', 2, 211.0), ('nottingham', 2, 230.0), ('cardiff', 2, 235.0),
('sheffield', 2, 268.0), ('swansea', 2, 288.0), ('aberystwyth', 2, 301.0), ('carlisle', 1, 302.0), ('york', 2, 302.0),
('penzance', 2, 329.0), ('glasgow', 1, 396.0), ('carlisle', 2, 406.0), ('birmingham', 2, 151.0), ('brighton', 2, 160.0),
('bristol', 2, 205.0), ('cardiff', 2, 231.0), ('carlisle', 2, 314.0), ('dover', 2, 179.0), ('exeter', 2, 270.0),
('glasgow', 2, 408.0), ('hull', 2, 178.0), ('portsmouth', 2, 180.0), ('sheffield', 2, 170.0), ('swansea', 2, 270.0),
```

```
('york', 2, 203.0), ('london', 2, 108.0)])]
Sorting frontier based on cost  =  [('cambridge', '54.0', [('oxford', 1, 57.0), ('dover', 1, 71.0), ('portsmouth', 2, 101.0), ('london', 2, 104.0), ('london', 2, 108.0), ('birmingham', 1, 110.0), ('bristol', 1, 116.0), ('dover', 2, 133.0), ('oxford', 2, 148.0), ('birmingham', 2, 151.0), ('cambridge', 2, 158.0), ('brighton', 2, 160.0), ('cardiff', 1, 161.0), ('sheffield', 2, 170.0), ('exeter', 1, 172.0), ('hull', 1, 172.0), ('hull', 2, 178.0), ('dover', 2, 179.0), ('portsmouth', 2, 180.0), ('bristol', 2, 188.0), ('leeds', 1, 198.0), ('liverpool', 1, 198.0), ('york', 2, 203.0), ('bristol', 2, 205.0), ('birmingham', 2, 211.0), ('nottingham', 2, 230.0), ('cardiff', 2, 231.0), ('cardiff', 2, 235.0), ('sheffield', 2, 268.0), ('exeter', 2, 270.0), ('swansea', 2, 270.0), ('swansea', 2, 288.0), ('aberystwyth', 2, 301.0), ('carlisle', 1, 302.0), ('york', 2, 302.0), ('carlisle', 2, 314.0), ('penzance', 2, 329.0), ('glasgow', 1, 396.0), ('carlisle', 2, 406.0), ('glasgow', 2, 408.0)])]
explored = {'cambridge', 'brighton', 'london'}
destination = aberdeen
*************************
```

Q1 b-1 Provide the optimal path found by the algorithm (both the path and its length);

```
Path =
london -> cambridge , 54.0  units
cambridge -> york , 149.0  units
york -> aberdeen , 299.0  units
Total Cost = 502.0  units
```

Q1 b-2

```
*************************
frontier =  [('glasgow', '396.0', ['london', 'brighton', 'cambridge', 'oxford', 'dover', 'portsmouth', 'birmingham', 'bristol', 'nottingham', 'cardiff', 'sheffield', 'exeter', 'hull', 'leeds', 'liverpool', 'manchester', 'swansea', 'york', 'aberystwyth', 'newcastle', 'carlisle', 'penzance', 'edinburgh'])]
Frontier selected & Child nodes of glasgow are
Exploring all frontiers =  [('glasgow', '396.0', [('london', 2, 396.0), ('london', 2, 396.0), ('glasgow', 2, 396.0), ('sheffield', 3, 398.0), ('cardiff', 2, 399.0), ('hull', 4, 400.0), ('york', 3, 403.0), ('dover', 3, 405.0), ('carlisle', 2, 406.0), ('sheffield', 4, 406.0), ('cardiff', 2, 407.0), ('glasgow', 2, 408.0), ('manchester', 2, 409.0), ('glasgow', 2, 409.0), ('edinburgh', 2, 409.0), ('glasgow', 2, 410.0), ('liverpool', 2, 411.0), ('glasgow', 2, 412.0), ('edinburgh', 3, 413.0), ('edinburgh', 3, 413.0), ('manchester', 4, 413.0), ('glasgow', 3, 414.0), ('bristol', 3, 414.0), ('york', 2, 414.0), ('dover', 2, 415.0), ('glasgow', 2, 415.0), ('dover', 2, 418.0), ('cambridge', 3, 418.0), ('portsmouth', 3, 421.0), ('manchester', 2, 422.0), ('swansea', 2, 427.0), ('leeds', 3, 431.0), ('glasgow', 4, 431.0), ('glasgow', 4, 431.0), ('newcastle', 3, 432.0), ('carlisle', 3, 432.0), ('cardiff', 3, 435.0), ('glasgow', 3, 436.0), ('brighton', 2, 656.0), ('dover', 2, 675.0), ('nottingham', 2, 677.0), ('aberdeen', 3, 704.0), ('york', 3, 705.0), ('cambridge', 2, 750.0), ('penzance', 4, 751.0), ('oxford', 2, 751.0), ('bristol', 4, 756.0), ('cardiff', 4, 757.0), ('bristol', 2, 766.0), ('cardiff', 2, 767.0), ('newcastle', 3, 775.0), ('london', 2, 792.0), ('exeter', 4, 832.0), ('exeter', 2, 842.0), ('dover', 2, 863.0), ('aberdeen', 3, 970.0)])]
explored = {'swansea', 'glasgow', 'manchester', 'carlisle', 'leeds', 'liverpool', 'aberystwyth', 'cambridge', 'edinburgh', 'newcastle', 'brighton', 'birmingham', 'nottingham', 'bristol', 'dover', 'oxford', 'exeter', 'london', 'sheffield', 'portsmouth', 'hull', 'york', 'cardiff', 'penzance'}
destination = aberdeen
*************************
frontier =  [('aberdeen', '502.0', ['london', 'brighton', 'cambridge', 'oxford', 'dover', 'portsmouth', 'birmingham', 'bristol', 'nottingham', 'cardiff', 'sheffield', 'exeter', 'hull', 'leeds', 'liverpool', 'manchester', 'swansea', 'york', 'aberystwyth', 'newcastle', 'carlisle', 'penzance', 'edinburgh', 'glasgow'])]
explored = {'swansea', 'aberdeen', 'glasgow', 'manchester', 'carlisle', 'leeds', 'liverpool', 'aberystwyth', 'cambridge', 'edinburgh', 'newcastle', 'brighton', 'birmingham', 'nottingham', 'bristol', 'dover', 'oxford', 'exeter', 'london', 'sheffield', 'portsmouth', 'hull', 'york', 'cardiff', 'penzance'}
destination = aberdeen
************************
```

Q1 c

The approach is described below:

`cost = time + pollution_factor` -- (1)

`time = distance/speed`

`pollution_factor` = $10^{-5} * v^2$

We need to find the best speed to find the optimal cost and path

Differentiating the eq. (1) w.r.t speed v, we get *v = 316.22*

```
Path =
london -> cambridge , 0.3415259874011764  units
cambridge -> york , 0.942358743014357  units
york -> aberdeen , 1.8910420413509583  units
Total Cost = 3.1749267717664917  units
```

Q1 The approach is described below:

Let `speed_limit` is equal to `distance/cost` of reaching from A to B

Fine factor is defined as a value 0 to 1 of how likely a fine is imposed

```
fine_factor = 0 if speed < speed_limit else 1 - math.exp(-(speed-speed_limit))
```

Fine is calculated as 1000 or 0 if fine_factor is 1 or 0.

```
fine =  1000 * (1 if fine_factor > 0 else 0)
```

Car rental is dependent upon time thus

```
car_rental = (distance/speed)*100
```

Total cost is given by

```
cost = car_rental + fine
```

```
Path =
london -> glasgow , 125.22927076086268  units
glasgow -> exeter , 141.0410473720827  units
exeter -> newcastle , 113.84479160078422  units
newcastle -> swansea , 100.56289924735938  units
swansea -> aberdeen , 158.7502371766492  units
Total Cost = 639.4282461577382  units
```

Question 2: It's on the tip of my tongue! (Genetic Algorithm)

Q2 a) For my id 190573735 passwords are

CHUKN088IS

1_L1K3TH4T

Q2 b) State encoding → I have used the list as the data structure and encoded each probable password as a list of char of length 10 the char are A-Z 0-9 and _

Selection → In order to understand the performance I have implemented Ranking, mixed ranking and tournament selection procedure to select the parents

In the *ranking selection* method, I have picked top (descending order according to fitness score) N num of parents which seemed viable as their fitness score was high.

In the *mixed ranking selection* method, I have sorted the population according to fitness score and selected N num of parents 60% from top, 20% from bottom and 20% from the rest of the population

In the *tournament* selection method, I have randomly picked 3, contested among them to pick the best one and performed this operation until N num of parents are not selected

Crossover → I have used N point (N=2) crossover to generate new children with N being calculated randomly. Here the gene values are captured from both the parents.

Mutation → In order to understand the performance, I have kept the mutation rate as a variable, if mutation_rate is bigger than certain threshold then the child chromosomes are altered by picking random values from all the password options.

Q2 c) For first password and on 50 iteration run and below hyper parameters

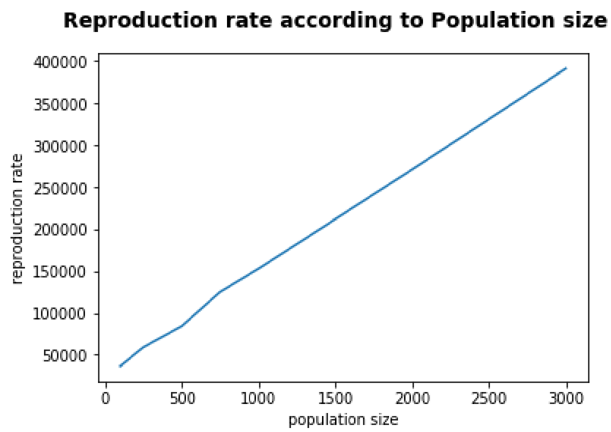
population_size = 100,

mutation_rate = 0.2

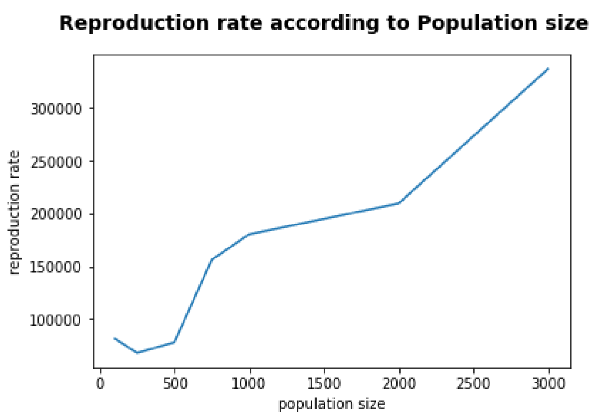
	Tournament selection	Ranking selection	Mixed Ranking selection
μ reproductions	33702	33134	78786
σ reproductions	13966.51	19159	48391.38

μ time	0.3345	0.25422	0.63165

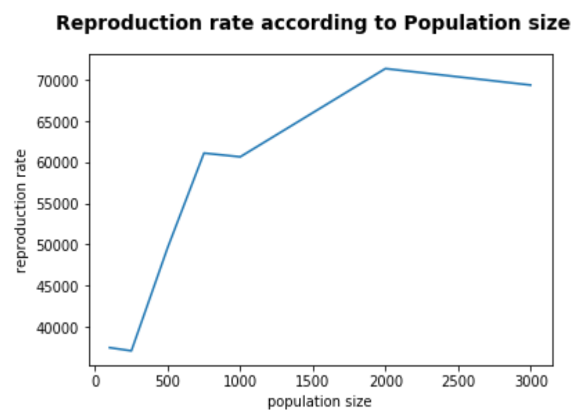
Q2 d) hyper-parameter impact Population size



Tournament



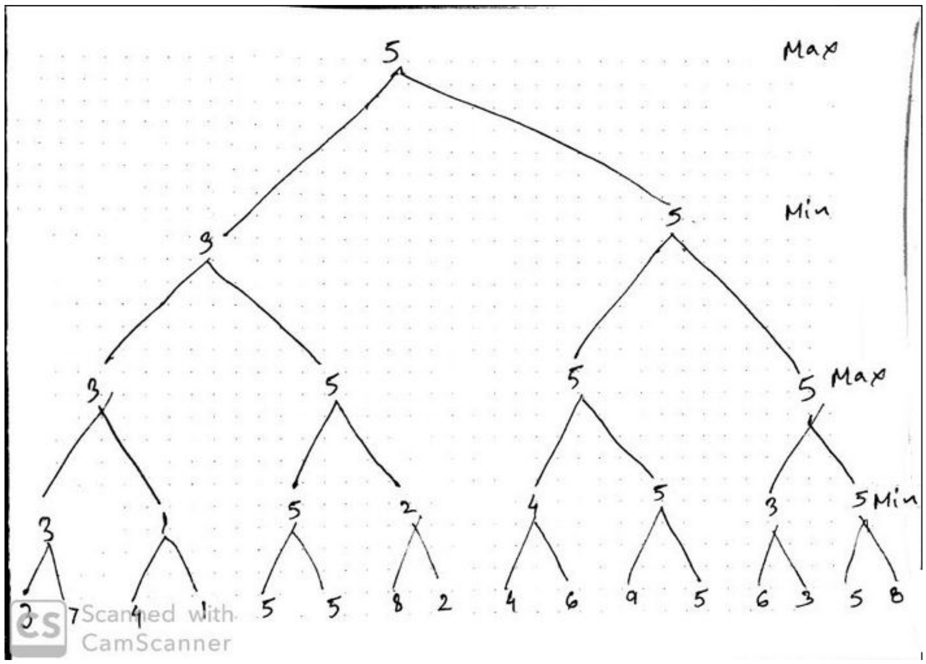
Mixed ranking



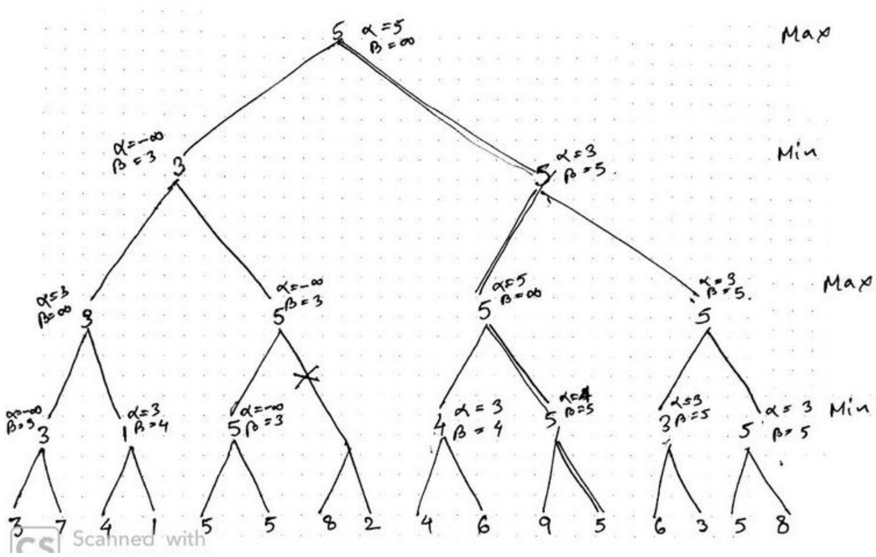
Ranking

In all the above figures we can see when we increase the population size the reproduction rate drops then it increases at a very high rate. One possible explanation is, when the population size is small, it takes a lot of randomisation and reproduce more a but as the population size becomes optimum it converges faster, but if we increase the population size to a very large number then it becomes very difficult to converge as there is lot of randomisation (lot of directions for the hill climber) in the population in each reproduction.

Q**3 a)** Play optimally (MINIMAX algorithm)



Q3 b) alpha beta pruning



The branch shown with the cross is only pruned, it is *β cut-off* as the Max player sees α ≥ β here α = max(α, v) which is 5 and β = 3, thus it gets pruned.

Q3 c) What are the ranges of values for *x* that will be still worth playing the game?

The ranges of values for *x* are less than or equal to 5 since the maximizer does not lose if he plays in that range.

Q3 d) For a fixed value of x, do you prefer to be the first player (MAXimiser) or the second player (MINimiser)? Very briefly explain your answer.

It entirely depends on what the fixed value of x is. If the value of x is greater than 5 then Minimiser would win but if the value of x is less than or equal to 5 then Maximiser would win.