

**ARTIFICIAL INTELLIGENCE ASSIGNMENT 2**  
**SANKET RAJENDRA GUJAR ([srgujar@wpi.edu](mailto:srgujar@wpi.edu))**

Question 3.8:

**a. Suppose that actions can have arbitrarily large negative costs; explain why this possibility would force any optimal algorithm to explore the entire state space.**

Answer: If the actions can have arbitrarily large negative cost then any of the path leading to the goal state can have large reward, thus the algorithm has to explore the entire state because any one of them can lead it to the optimal path.

**B. Does it help if we insist that step costs must be greater than or equal to some negative constant  $c$ ? Consider both trees and graphs.**

Answer:

For Tree, Yes it will help,

Let the step cost be  $> 'c'$  (as defined in problem), let the depth remaining to solution be ' $d$ '.

So best next path available is  $c*d$ .

so we can neglect any path, which when added  $c*d$  (the best possible next path) to its cost, is worse than best path cost, as it will not produce the optimal result.

For Graphs, No, it will not help as it will go in loop trying to get the lowest cost ( $c$ ).

**C. Suppose that a set of actions forms a loop in the state space such that executing the set in some order results in no net change to the state. If all of these actions have negative cost, what does this imply about the optimal behavior for an agent in such an environment?**

Answer :

As there is no change in state and have negative cost, the agent should iterate the loop infinitely, until it finds a better optimal path.

**D. One can easily imagine actions with high negative cost, even in domains such as route finding. For example, some stretches of road might have such beautiful scenery as to far outweigh the normal costs in terms of time and fuel. Explain, in precise terms, within the context of state-space search, why humans do not drive around scenic loops indefinitely, and explain how to define the state space and actions for route finding so that artificial agents can also avoid looping.**

Answer:

Let me list some reason why humans avoid drive around scenic loops indefinitely:

1. More weight given to the goal state. (e.g Important Meeting to reach, Work etc)
2. Bounding cost (time) for actions (scenery viewing while travelling). (e.g will stop only for 1 hour in the green fields.)
3. Visiting frequently the same state space ( e.g. Going through the same road everyday for work will decrease the interest independent of the beauty).

For Agents to avoid scenic beauty loop:

1. Giving a high reward to the goal state.
2. Bounding the negative cost, to not outweigh the normal cost.
3. The state should have the knowledge of the previous locations visited, so it will avoid going back to the previous location it had visited.

**E. Can you think of a real domain in which step costs are such as to cause looping?**

Answer:

The Life of a Software developer is a common example as they say:

**Eat Sleep Code Repeat**

The agent eats, which makes him sleepy so the agent sleeps, after waking up he codes because he it is his job, this makes him hungry and tired, then he eats, and thus this loop continues.

**3.9 The missionaries and cannibals problem is usually stated as follows. Three missionaries and three cannibals are on one side of a river, along with a boat that can hold one or two people. Find a way to get everyone to the other side without ever leaving a group of missionaries in one place outnumbered by the cannibals in that place. This problem is famous in AI because it was the subject of the first paper that approached problem formulation from an analytical viewpoint (Amarel, 1968).**

**a. Formulate the problem precisely, making only those distinctions necessary to ensure a valid solution. Draw a diagram of the complete state space.**

Let,

M denote missionary

C denote Cannibal

Right and Left denote the boat side of river.

The State space can be denoted by size 5 tuple where,

1st position specify the number of missionary on left side of the river

2nd position specify the number of cannibals on left side of the river

3rd position specify the number of missionary on right side of the river

4th position specify the number of cannibals on right side of the river

5th position specify the side of river the boat is on.

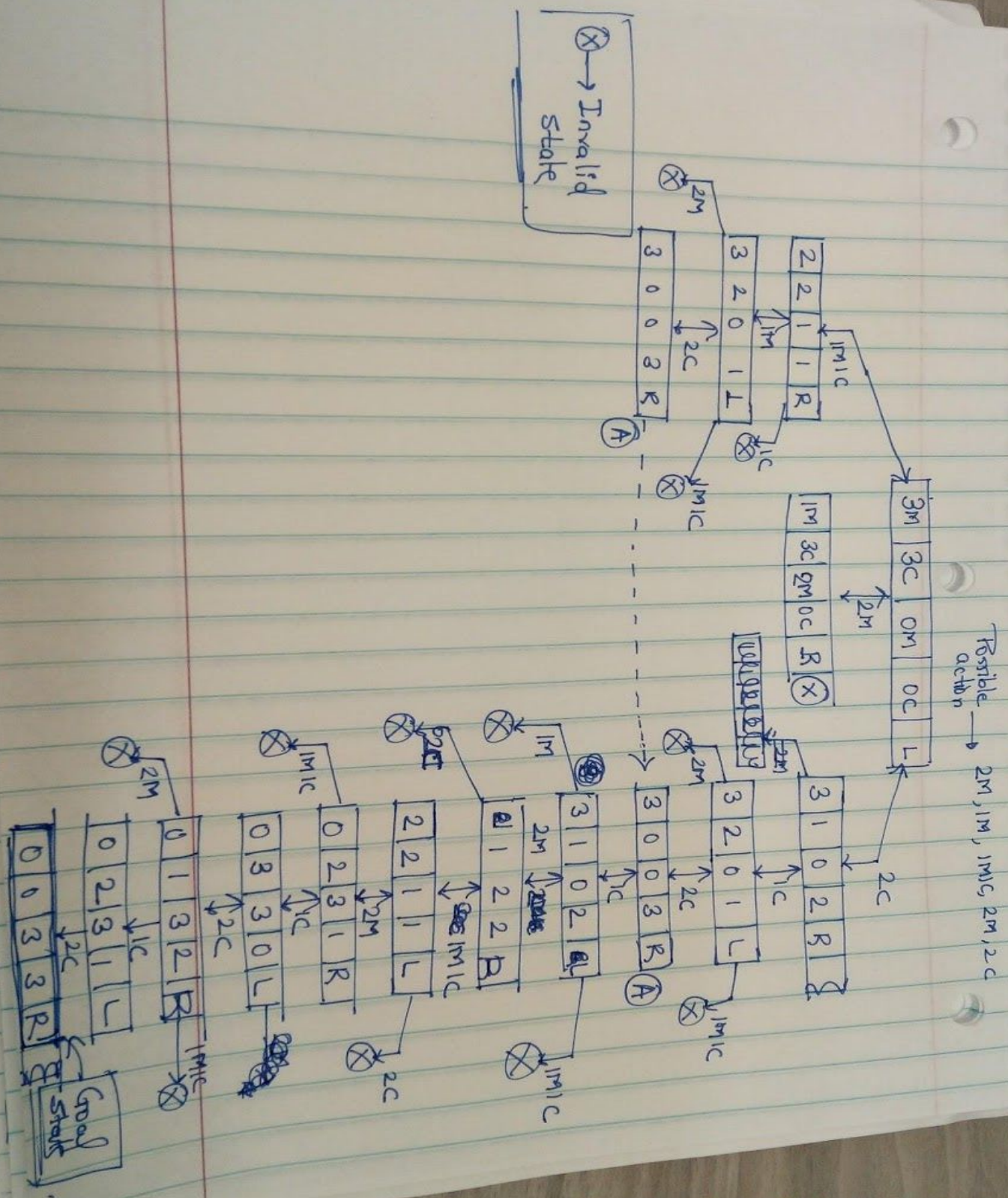
So, the initial state will be :

3M	3C	0M	0C	LEFT
----	----	----	----	------

Goal state will be:

0M	0C	3M	3C	RIGHT
----	----	----	----	-------

\*Left in boat position is not possible in goal state.



**b. Implement and solve the problem optimally using an appropriate search algorithm. Is it a good idea to check for repeated states?**

The problem was implemented using BFS.

Please find the code in Missionary\_and\_Cannibals.py

The output of the code is as shown below, where it prints the state visited until it reaches the goal state.

```
sanket@sanket-HP-Pavilion-15-Notebook-PC:/media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python Missionary_and_Cannibals.py
[3, 3, 0, 0, 0]
[3, 1, 0, 2, 1]
[2, 2, 1, 1, 1]
[3, 2, 0, 1, 1]
[3, 3, 0, 0]
[3, 2, 0, 1]
[3, 2, 0, 1, 0]
[3, 3, 0, 0, 0]
[3, 0, 0, 3, 1]
[3, 1, 0, 2, 1]
[2, 2, 1, 1, 1]
[3, 2, 0, 1, 0]
[3, 1, 0, 2, 0]
[1, 1, 2, 2, 1]
[3, 0, 0, 3, 1]
[3, 1, 0, 2, 0]
[2, 2, 1, 1, 0]
[0, 2, 3, 1, 1]
[1, 1, 2, 2, 1]
[2, 2, 1, 1, 0]
[0, 3, 3, 0, 0]
[0, 1, 3, 2, 1]
[0, 2, 3, 1, 1]
[0, 3, 3, 0, 0]
[0, 2, 3, 1, 0]
[1, 1, 2, 2, 0]
[0, 0, 3, 3, 1]
[0, 1, 3, 2, 1]
[0, 0, 3, 3, 1]
[0, 1, 3, 2, 1]
Goal reached [0, 0, 3, 3, 1]
sanket@sanket-HP-Pavilion-15-Notebook-PC:/media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ 
sanket@sanket-HP-Pavilion-15-Notebook-PC:/media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$
```

**c. Why do you think people have a hard time solving this puzzle, given that the state space is so simple?**

The states are easily repeatable, people mostly end up with going in a loop of same states with partially realizing it, as well as it has very unclear branching, resulting in not sufficient estimation of upcoming states. It's easy when you draw the state diagram and don't repeat state and can clearly estimate the upcoming states.

**3.21 Prove each of the following statements, or give a counterexample:**

**a. Breadth-first search is a special case of uniform-cost search.**

**Answer:**

When the step cost of all options are equal, then uniform-cost search behaves as Breadth-first search

$f(n)$  is proportional to depth

**b. Depth-first search is a special case of best-first tree search.**

**Answer:**

When the cost reduces with the depth, the best first search will work like an depth-first search.

$f(n) = -\text{depth}(n)$ .

**c. Uniform-cost search is a special case of A\* search.**

**Answer:**

When the heuristics  $h(n)=0$ , the cost of reaching the goal is zero, the decision will depend on the current path cost  $g(n)$ , which makes its Uniform-cost search

$h(n)=0$



**3.22 Compare the performance of A\* and RBFS on a set of randomly generated problems in the 8-puzzle (with Manhattan distance) and TSP (with MST—see Exercise 3.30) domains. Discuss your results. What happens to the performance of RBFS when a small random number is added to the heuristic values in the 8-puzzle domain?**

**Answer:**

```

sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket
sanket@sanket-HP-Pavilion-15-Notebook-PC: ~/FALL 1
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume
*****
The Random puzzle problem is :
[[1 0 6]
 [5 2 8]
 [7 3 4]]
Started A star search
Goal reached., the current state is
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Goal reached for A_star in : 0.349529
the nodes visited by A* 191
started RBFS search
Goal reached., the current state is
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Goal reached for RBFS in : 0.179389
the nodes visited by 353
*****
The Random puzzle problem is :
[[1 6 4]
 [3 5 0]
 [7 8 2]]
Started A star search
Goal reached., the current state is
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Goal reached for A_star in : 0.041458
the nodes visited by A* 241
started RBFS search
Goal reached., the current state is
[[0 1 2]
 [3 4 5]
 [6 7 8]]
Goal reached for RBFS in : 0.096768
the nodes visited by 489

```

For the 8 puzzle problem,  
Please find the program in '8\_puzzle\_using\_A\_star\_and\_RBFS.py'  
Random puzzle state are tested using A star and RBFS  
The Heuristics are Manhattan distance.

As seen in the screenshots,  
The Astar and RBFS both gets the solution, but as it seems A\* takes more memory as it stores every unexplored node in memory but explores less node as it not revisit node.  
Whereas the RBFS takes none memory as it only stores the best alternative, but it ends up exploring more node than A\* as it revisits the explored node.

When we add small random noise, it gets into infinite iteration as it keeps on exploring same nodes again and again.

For the TSP problem:

```
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket
sanket@sanket-HP-Pavilion-15-Notebook-PC: ~/FALL 17/AI_CS
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI 162x41
sanket@sanket-HP-Pavilion-15-Notebook-PC:/media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_Astar_and_RBFS.py

*****Defining problem on new cities*****
The coordinate of location of the cities are : [(3, 10), (5, 30), (10, 35), (2, 9), (55, 25), (30, 20), (30, 25), (20, 35)]
In order [0,1,2,3,4,5,6,7]
Starting A star search .....
Goal reached., the path is [4, 6, 5, 3, 0, 1, 2, 7] and the cost is : 2111
Starting RBFS search .....
Goal reached., the path is [4, 6, 5, 3, 0, 1, 2, 7] and the cost is : 2111

*****Defining problem on new cities*****
The coordinate of location of the cities are : [(1, 10), (5, 30), (2, 9), (10, 35), (30, 20), (55, 25), (20, 35), (30, 25)]
In order [0,1,2,3,4,5,6,7]
Starting A star search .....
Goal reached., the path is [5, 4, 2, 0, 1, 3, 6, 7] and the cost is : 2323
Starting RBFS search .....
Goal reached., the path is [5, 4, 2, 0, 1, 3, 6, 7] and the cost is : 2323

*****Defining problem on new cities*****
The coordinate of location of the cities are : [(80, 40), (5, 15), (10, 45), (15, 55), (4, 14), (45, 30), (45, 40), (30, 52)]
In order [0,1,2,3,4,5,6,7]
Starting A star search .....
Goal reached., the path is [0, 6, 5, 4, 1, 2, 3, 7] and the cost is : 4548
Starting RBFS search .....
Goal reached., the path is [0, 6, 5, 4, 1, 2, 3, 7] and the cost is : 4548

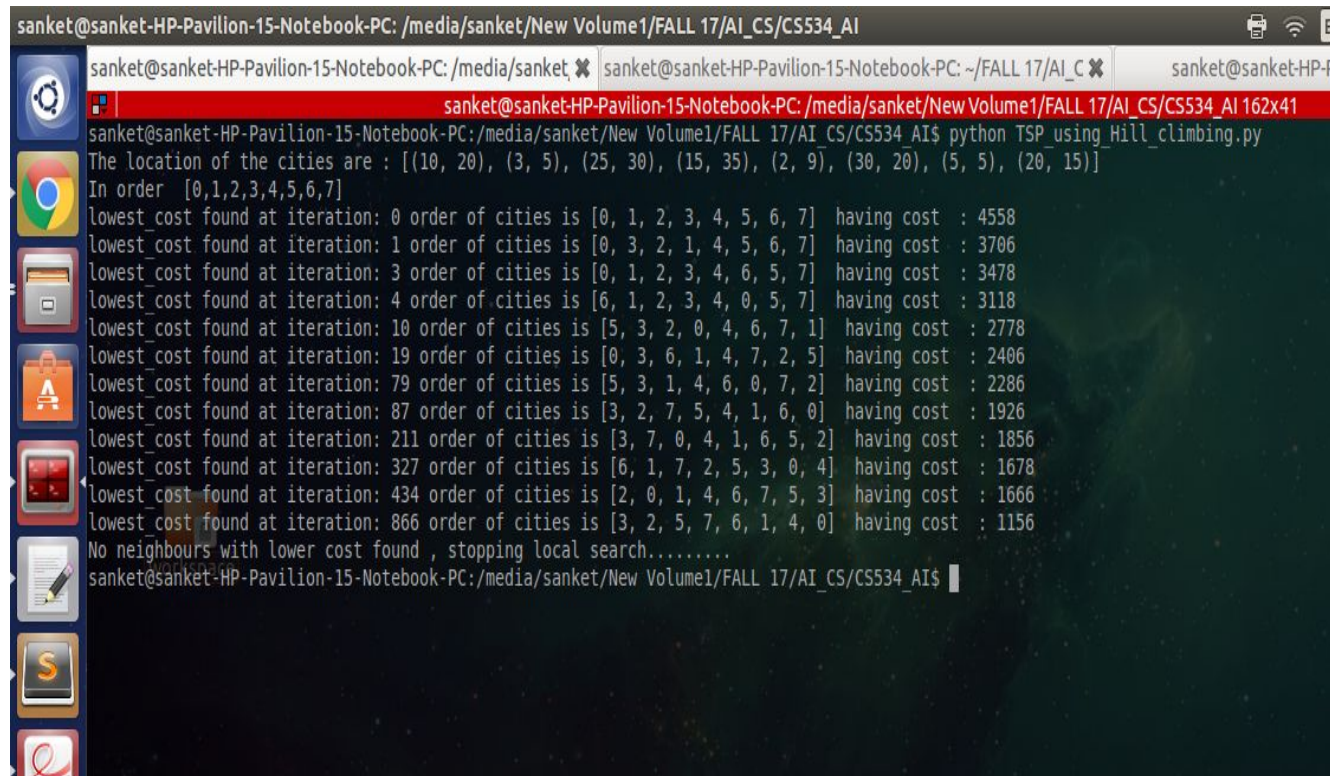
*****Defining problem on new cities*****
The coordinate of location of the cities are : [(2, 7), (4, 22), (8, 26), (1, 7), (22, 15), (41, 18), (15, 24), (22, 18)]
In order [0,1,2,3,4,5,6,7]
Starting A star search .....
Goal reached., the path is [5, 4, 3, 0, 1, 2, 6, 7] and the cost is : 1275
Starting RBFS search .....
Goal reached., the path is [5, 4, 3, 0, 1, 2, 6, 7] and the cost is : 1275
sanket@sanket-HP-Pavilion-15-Notebook-PC:/media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$
```

Please find the code in 'TSP\_using\_Astar\_and\_RBFS.py'  
The Heuristics is Minimum Spanning Tree.



**4.3 In this exercise, we explore the use of local search methods to solve TSPs of the type defined in Exercise 3.30.**

**a. Implement and test a hill-climbing method to solve TSPs. Compare the results with optimal solutions obtained from the A\* algorithm with the MST heuristic (Exercise 3.30).**



```
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket
sanket@sanket-HP-Pavilion-15-Notebook-PC: ~/FALL 17/AI_CS
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI 162x41
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_Hill_climbing.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
lowest_cost found at iteration: 0 order of cities is [0, 1, 2, 3, 4, 5, 6, 7] having cost : 4558
lowest_cost found at iteration: 1 order of cities is [0, 3, 2, 1, 4, 5, 6, 7] having cost : 3706
lowest_cost found at iteration: 3 order of cities is [0, 1, 2, 3, 4, 6, 5, 7] having cost : 3478
lowest_cost found at iteration: 4 order of cities is [6, 1, 2, 3, 4, 0, 5, 7] having cost : 3118
lowest_cost found at iteration: 10 order of cities is [5, 3, 2, 0, 4, 6, 7, 1] having cost : 2778
lowest_cost found at iteration: 19 order of cities is [0, 3, 6, 1, 4, 7, 2, 5] having cost : 2406
lowest_cost found at iteration: 79 order of cities is [5, 3, 1, 4, 6, 0, 7, 2] having cost : 2286
lowest_cost found at iteration: 87 order of cities is [3, 2, 7, 5, 4, 1, 6, 0] having cost : 1926
lowest_cost found at iteration: 211 order of cities is [3, 7, 0, 4, 1, 6, 5, 2] having cost : 1856
lowest_cost found at iteration: 327 order of cities is [6, 1, 7, 2, 5, 3, 0, 4] having cost : 1678
lowest_cost found at iteration: 434 order of cities is [2, 0, 1, 4, 6, 7, 5, 3] having cost : 1666
lowest_cost found at iteration: 866 order of cities is [3, 2, 5, 7, 6, 1, 4, 0] having cost : 1156
No neighbours with lower cost found , stopping local search.....
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$
```

Please find the code in TSP\_using\_Hill\_Climbing.py

The Hill Climbing algorithm was executed as follows:

1. Take a random state
2. Calculate its cost.
3. Create neighbours by swapping two cities randomly.
4. Check for a defined count if the neighbours cost decreases or not. (As here we are taking path cost it should decrease)
5. If neighbours cost don't decrease stop the search

As it a local search problem, it may or may not return the optimal solution.

As observed from the experiments

<b>A*</b>	<b>Hill Climbing</b>
Returns optimal solution	May or may not return the optimal solution
Returns optimal solution even if increase the number of cities but will require more memory	The Probability of finding optimal solution decreases drastically with increase in number of cities
Requires more memory, proportional to number of cities	Requires very low memory

**b. Repeat part (a) using a genetic algorithm instead of hill climbing. You may want to consult Larra ñaga et al. (1999) for some suggestions for representations.**

Please find the code in TSP\_using\_genetic.py

We can see the genetic algorithm execution in the screenshot below, we can observe it produces different result every time because of random selection of parent for reproduction.

We can observe that it produced near to optimal solution in some cases, as it really depends on the probability of fitness of its population.

If we can increase the number of generation it will converge to optimal solution, but it will also increase the execution time and computational cost.

```
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket
sanket@sanket-HP-Pavilion-15-Notebook-PC: ~/FALL 17/AI_CS
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI 162x41
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [6, 1, 7, 2, 5, 3, 0, 4] with cost of :1678.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [0, 7, 5, 1, 4, 6, 2, 3] with cost of :2646.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [6, 1, 4, 0, 5, 3, 2, 7] with cost of :1756.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [3, 5, 7, 6, 4, 1, 0, 2] with cost of :1666.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [4, 5, 7, 6, 1, 0, 2, 3] with cost of :2928.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [7, 2, 3, 5, 4, 6, 1, 0] with cost of :2158.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [4, 1, 7, 6, 5, 2, 3, 0] with cost of :2266.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [4, 1, 6, 0, 7, 2, 5, 3] with cost of :2066.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [4, 6, 7, 0, 2, 3, 5, 1] with cost of :2346.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$ python TSP_using_genetic.py
The location of the cities are : [(10, 20), (3, 5), (25, 30), (15, 35), (2, 9), (30, 20), (5, 5), (20, 15)]
In order [0,1,2,3,4,5,6,7]
The fittest order of the population is : [5, 2, 3, 7, 4, 6, 1, 0] with cost of :1738.0
sanket@sanket-HP-Pavilion-15-Notebook-PC: /media/sanket/New Volume1/FALL 17/AI_CS/CS534_AI$
```