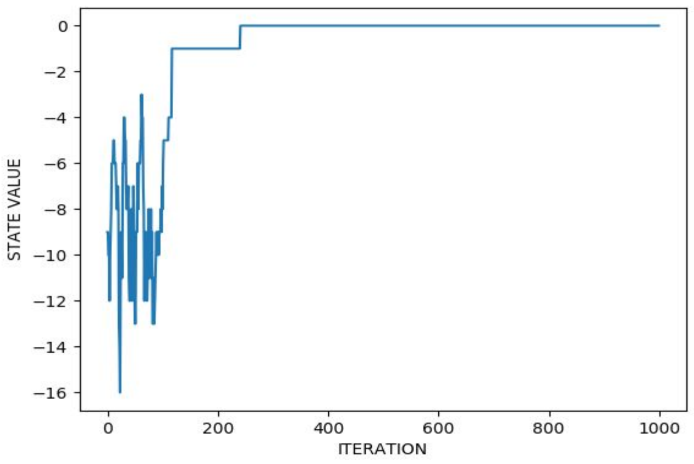
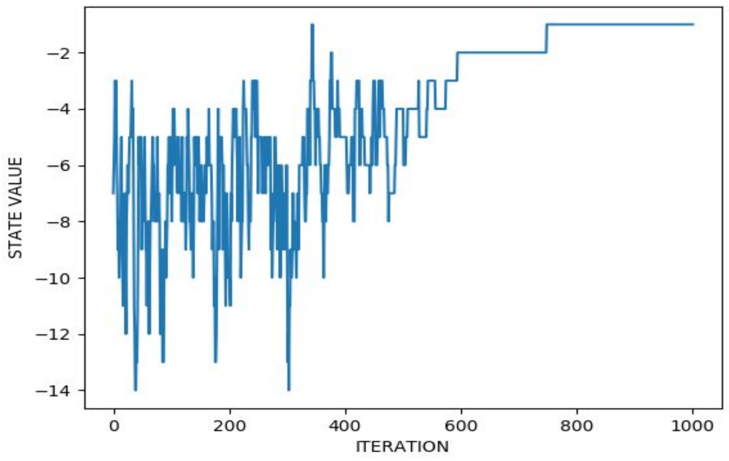
AI Assignment - 2

Local Search

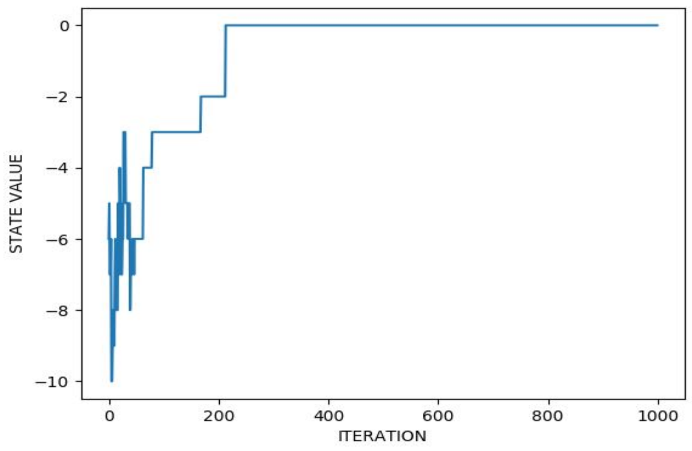
1. Simulated Annealing - In this question, I tried to 5 different temperature schedules, with 1000 as the maximum number of iterations.
   a) $T = 100e^{-0.01t} \quad if\ iteration\ <=\ 100$: In this, I observed that the agent tries to choose the best possible move (hill-climbing) after 100 iterations and it is very effective for this problem.



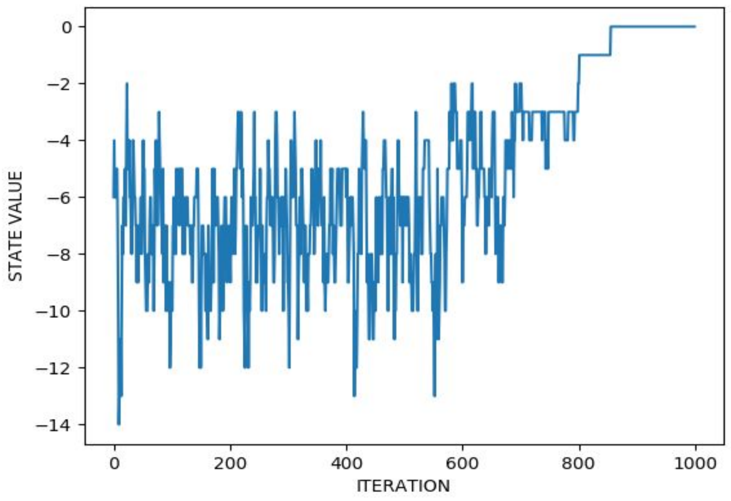
   b) $T = 100e^{-0.01t} \quad if\ iteration\ <=\ 1000$: In this, I observed the agent has a very little time to do the hill-climbing search due to which it does not reach the goal state always.



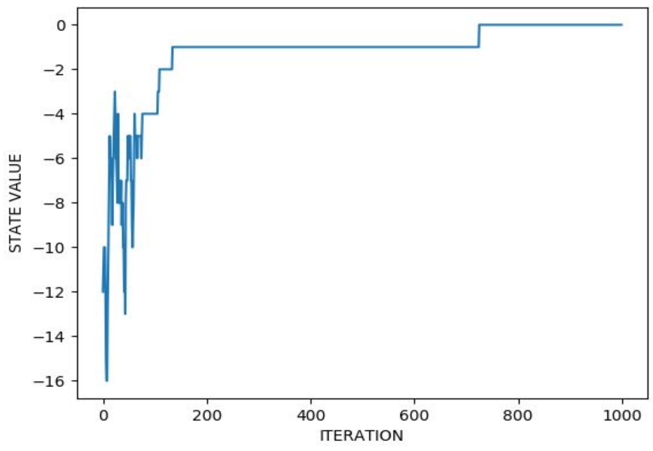
c) $T = 100e^{-0.1t} \quad if\ iteration\ <=\ 100$: In this, the Temperature falls very rapidly due to which it has a little more time to do the hill-climbing search as compared to part b)



d) $T = 1000e^{-0.01t} \quad if\ iteration\ <=\ 1000$: In this, as the initial temperature is really high the probability of choosing a worse action is very high due to which it oscillates a lot.

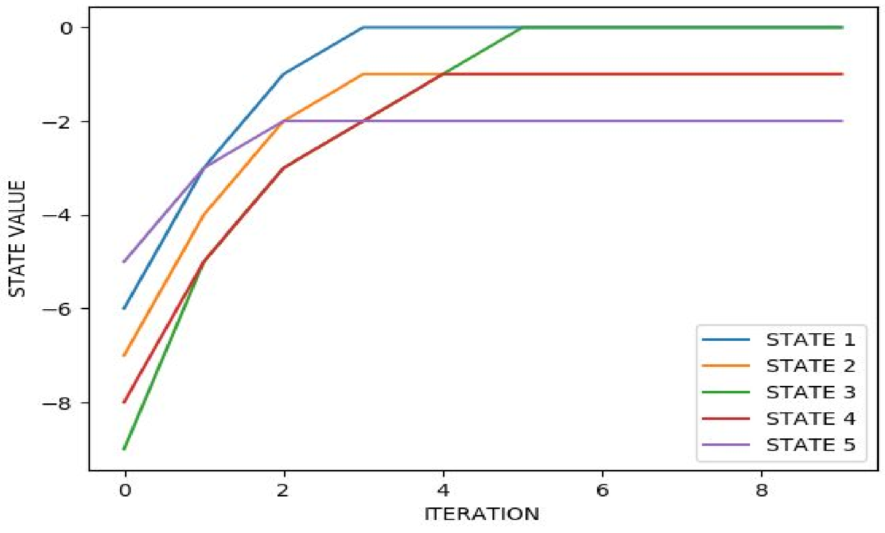


e) $T = 1000e^{-0.1t} \quad if\ iteration\ <=\ 100$: In this, due to high temperature there are many oscillations but as the exponent falls very quickly the probability of choosing a bad action falls very quickly.



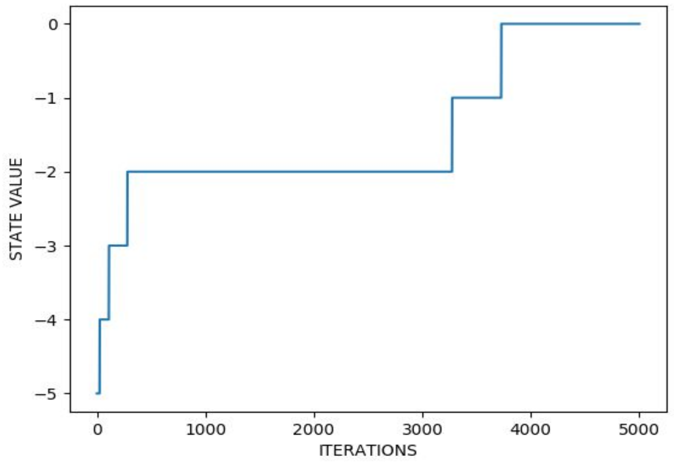
So in the simulated annealing, we had 3 parameters, 1. Limit of iteration, 2. Initial Temperature and 3. Exponent factor. We choose a bad action only till the Limit of iteration. If the initial temperature is high then the probability of choosing a bad move is very high. The Exponent factor defines the time till which we will be choosing a bad move.

2. Hill Climbing - In this search, the chances to reach the goal state is 20 percent. But is increase the number of initially chosen states to 10 then the search almost always reaches the goal state. Following is the graph of 5 different initial states.

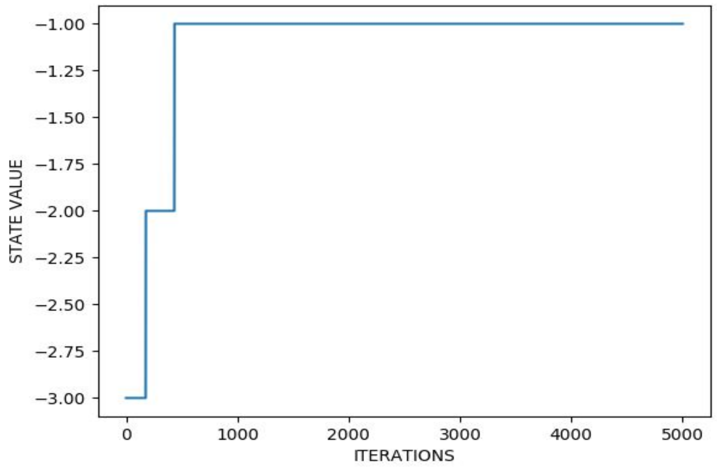


3. Genetic Algorithm - In this algorithm, there are 5 hyperparameters namely, 1. Population size, 2. Crossover probability, 3. Mutation Probability, 4. Elitism probability, and 5. Maximum Iterations. In this, we fix the maximum iteration to 5000, therefore we are left with 4 hyperparameters.

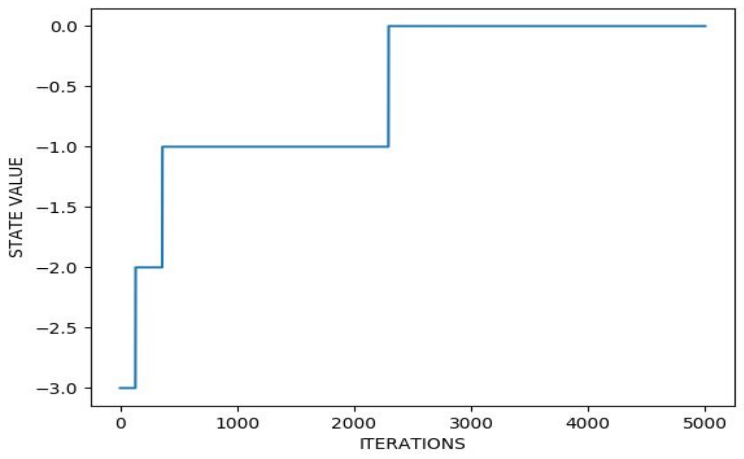
a) Population size = 50, Crossover Probability = 0.6, Mutation Probability = 0.09 and Elitism = 0.1: In this case, as we removing the bad state with probability equal to 0.9 (1 - Elitism), the algorithm takes many iterations to converge towards the goal state



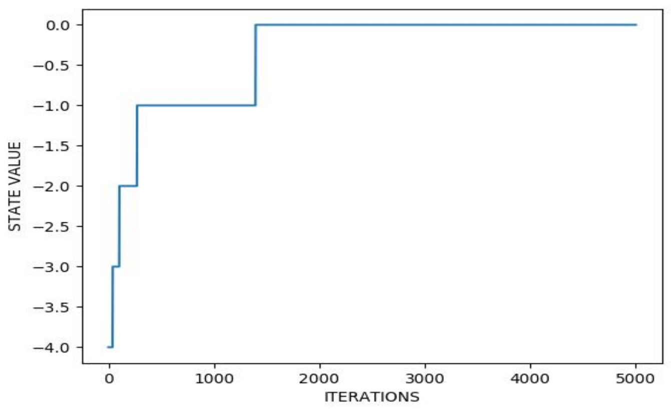
b) Population size = 50, Crossover Probability = 0.6, Mutation Probability = 0.09 and Elitism = 0.01: In this case, the algorithm converges much quicker as we are mostly choosing the best states but most of the times it sets to the local maxima.



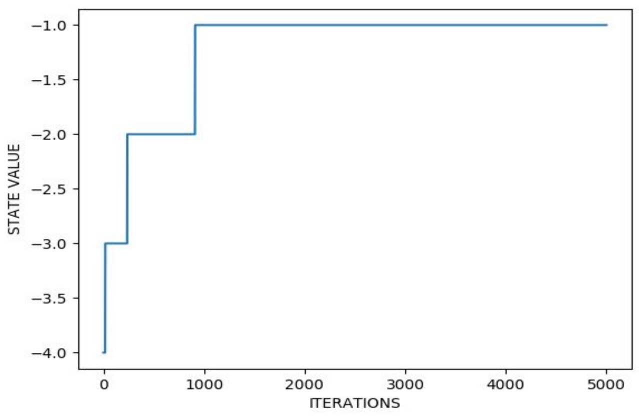
c) Population size = 500, Crossover Probability = 0.6, Mutation Probability = 0.09 and Elitism = 0.2: In this case, as the population size is very large, the algorithm finds global maxima very soon.



d) Population size = 50, Crossover Probability = 0.6, Mutation Probability = 0.4 and Elitism = 0.2: In this case there is a large probability of random walk.



e) Population size = 50, Crossover Probability = 0.9, Mutation Probability = 0.09 and Elitism = 0.2: In this case, as the crossover probability is very high, this algorithm only searches the local area and hence gets stuck in the local optimum.



Adversarial Search

In this search, we have 7 following codes :

1. TicTacToe.py - It contains the class Tic_Tac_Toe which is the basic structure of the Tic Tac Toe game.
2. Minimax_agent_vs_agent.py - In this code, two agents try to compete in a game of tic tac toe.
3. Minimax_human_vs_agent.py - In this code, a human can compete with the agent which is using the minimax algorithm
4. Minimax_agent_vs_agent_alpha_beta.py - In this code, two agents compete with each other where both are using alpha-beta pruning
5. Minimax_human_vs_agent_alpha_beta.py - In this code, a human can compete with the agent which is using the minimax algorithm with alpha-beta pruning
6. Minimax_agent_vs_agent_heuristic.py - In this code, two agents compete with each other where both are using alpha-beta pruning and a heuristic
7. Minimax_human_vs_agent_alpha_beta.py - In this code, a human can compete with the agent which is using the minimax algorithm with alpha-beta pruning and a heuristic.

Heuristic Used :
At the depth of 4 from the current node, we will get the following reward :

1. 1 ‘X’ and 2 empty spaces - 1 points
2. 2 ‘X’ and 1 empty space - 10 points
3. 3 ‘X’ - 100 points
4. Goal Node (‘X’ won) - 1000 points
5. 1 ‘X’ and 2 empty spaces: -1 points
6. 2 ‘X’ and 1 empty space: -10 points
7. 3 ‘X’ : -100 points
8. Goal Node (‘X’ won): -1000 points

In this, we observed that with even alpha-beta pruning we were not able to make an agent vs agent 4 X 4 tic tac toe game but with the heuristic mentioned we were able to implement the 4 X 4 tic tac toe.

Time taken by normal minimax algorithm to play 3 X 3 game - 91.355 seconds
Time taken minimax algorithm with alpha-beta pruning to play 3 X 3 game - 6.234 seconds
Time taken minimax algorithm with a heuristic to play 3 X 3 game - 1.179 seconds
Time taken minimax algorithm with a heuristic to play 4 X 4 game - 43.326 seconds

Branching Factor for Normal Minimax Algorithm (3 X 3, 1st move) - $986409^{(1/9)} = 4.63$
Branching Factor for Alpha-beta Minimax Algorithm (3 X 3, 1st move) - $24428^{(1/9)} = 3.07$

In agent vs agent, all the games that are played resulted in a tie.

In human vs agent, 3 people tried playing the agent using the heuristic and it was found that none of the human was able to beat the agent. Most of the games ended in a tie but some games were also won by the agent.