CS F407 - Artificial Intelligence

Assignment 2

Parmesh Mathur **2018A7PS0133G**

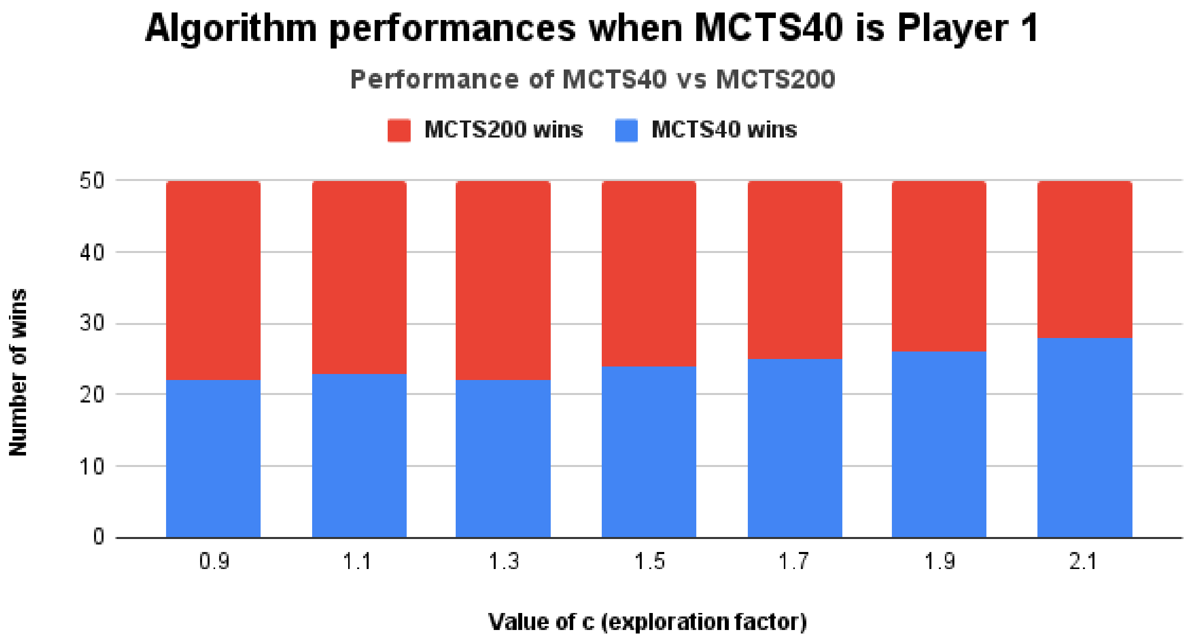
The Connect-4 game was to be played using the Monte Carlo Tree Search algorithm (or MCTS for short) and the Q-learning algorithm. These two algorithms were eventually made to play against each other.

In the following report, the general convention of $MCTS_n$ signifies that the MCTS algorithm performed n playouts to reach its judgement of the best move at every step.

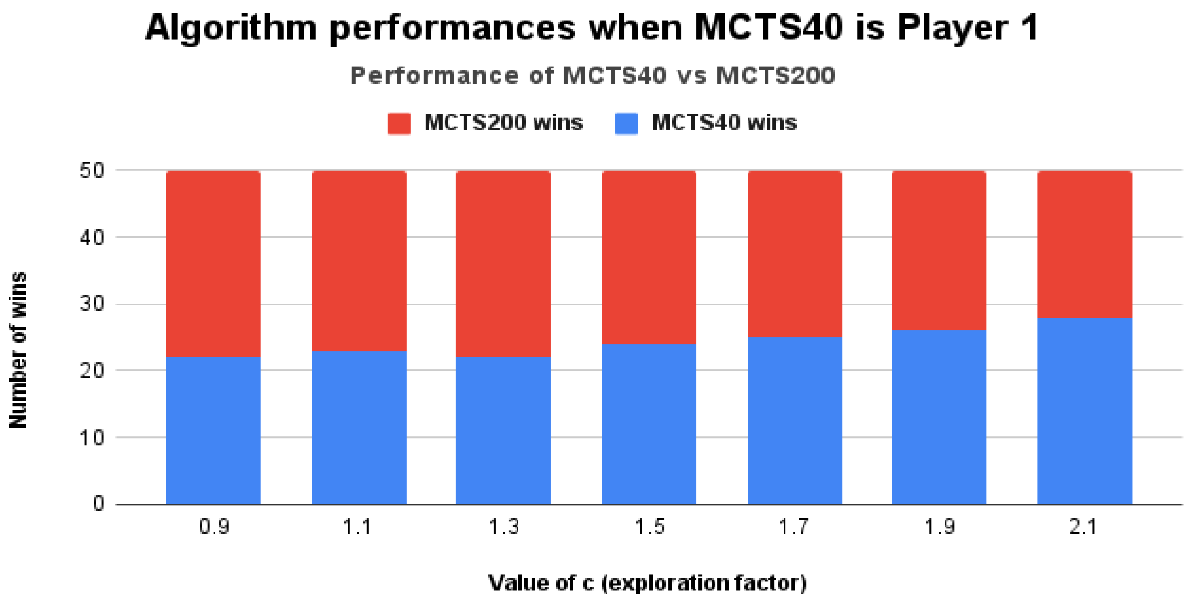
a. MCTS40 vs MCTS100

In this section, the relation of the performance of MCTS algorithm (with different number of playouts) with the exploration factor (c), is observed and remarked upon.

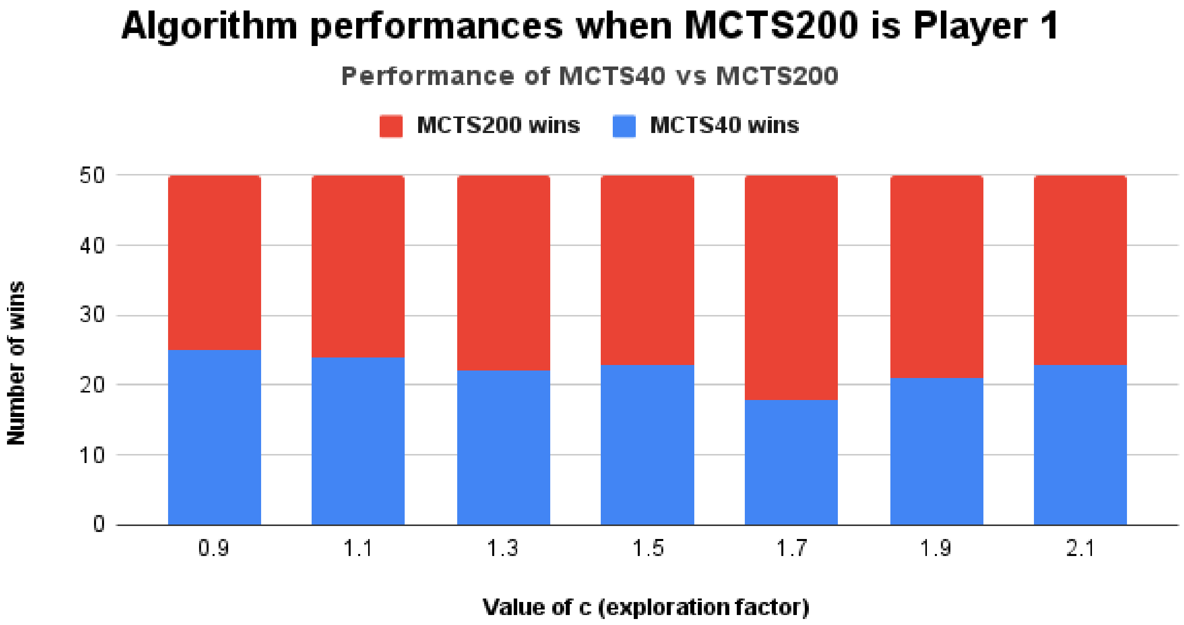
- The exploration factor used in the policy to choose the next state was varied from 0.9 to 2.1 at constant steps of 0.2 each.



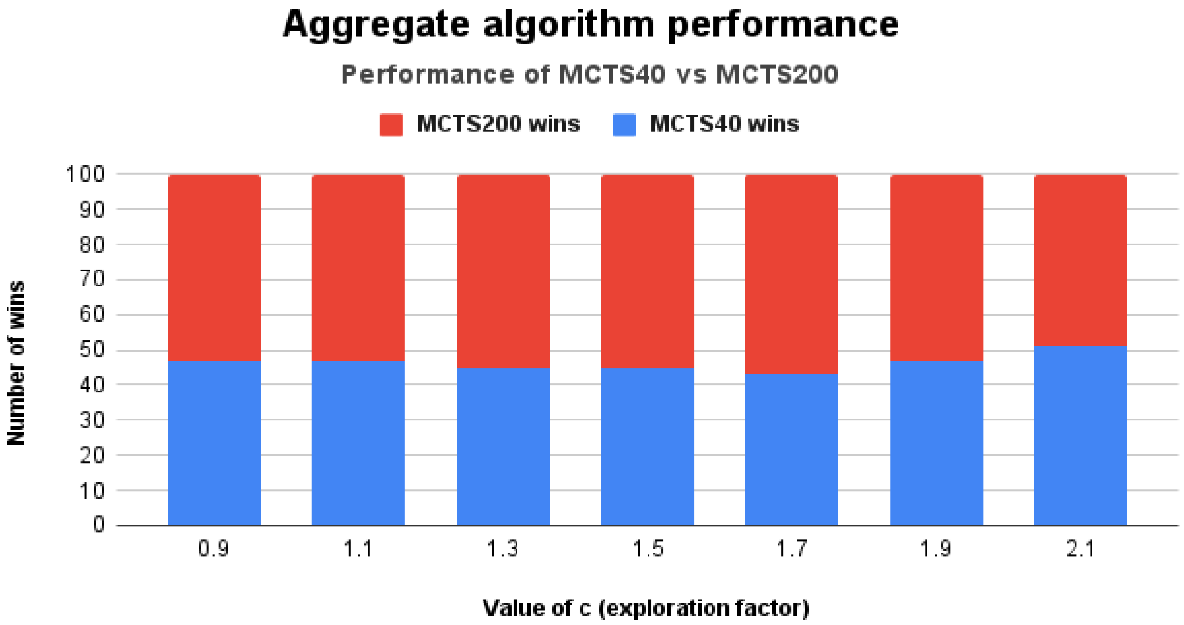
- It was noticed that when MCTS$_{40}$ played the first turn, it performed increasingly better with an increasing value of C. Whereas when MCTS$_{200}$ played first, the performance peaks near the value c = 1.7, after which it starts to deteriorate again (albeit only mildly).
-



- The performance of MCTS$_{40}$ on starting first can be explained if we think about the corresponding performance of MCTS$_{200}$ on having to play second. With a very high exploration tendency, the algorithm will keep on exploring sub optimal solutions as it has to consider more playouts for every move, which will cause it to digress, and pick the greedy move less frequently.
- Similarly, the performance of MCTS$_{200}$ on playing first can be explained. MCTS$_{200}$ has the edge at a particular value where it explores the optimal amount more than MCTS$_{40}$ does. This value has come up to be nearly 1.7, away from which will either lead MCTS$_{200}$ to explore more than required (above 1.7), or will take away what the extra playouts have to offer since there is less exploration allowed (below 1.7).



- The aggregate performance of MCTS_{200} against MCTS_{40} followed a predictable trend, where it mildly plateaus near the c = 1.6 mark (as shown in the graph below).

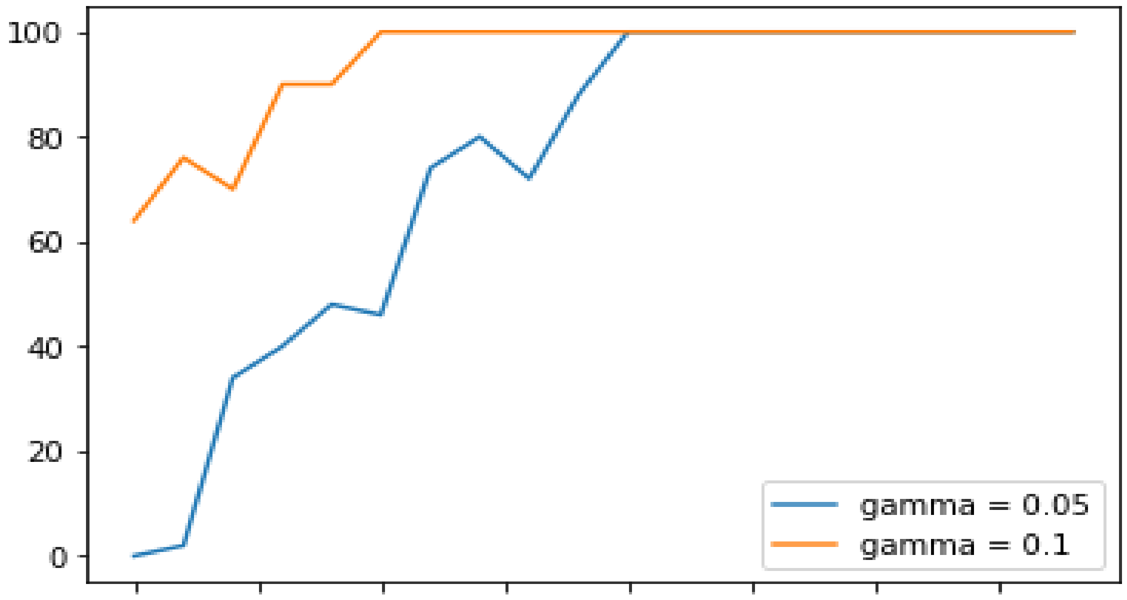


b. Q learning and its convergence

In this section, the effects of the different hyperparameters of the Q Learning algorithm are observed and discussed.

1. Changing the learning rate (alpha)

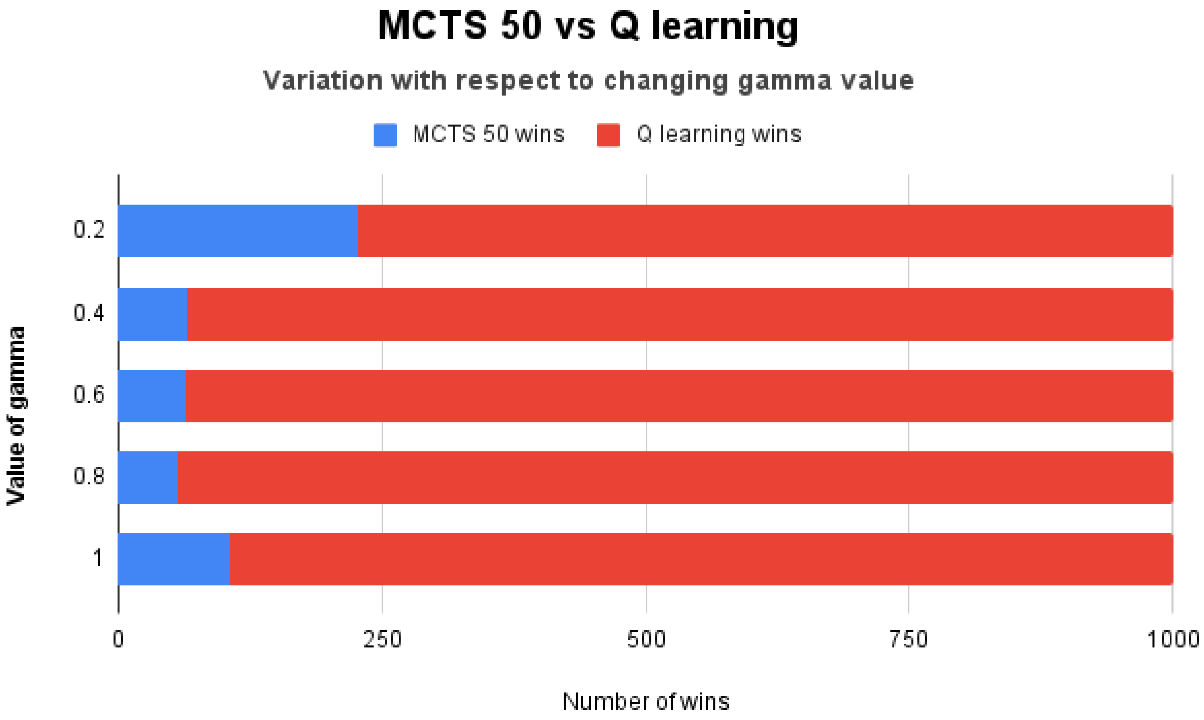
- Two learning rates of 0.05 and 0.1 were tried. It was found that a learning rate of 0.1 converges faster than 0.05. This is a fairly intuitive conclusion, as a higher learning rate will converge to the optimal values quicker than a lower learning rate.
- But, the limitations of a high learning rate can be seen if it is taken too high (to 0.5 or 0.8). The algorithm changes the values of the q table by more than what would be ideal, and hence the winning rate is compromised.



- In any case, it was noticed that after some point of time Q Learning has an Q table so well adapted that it wins in every game (refer to graph above).

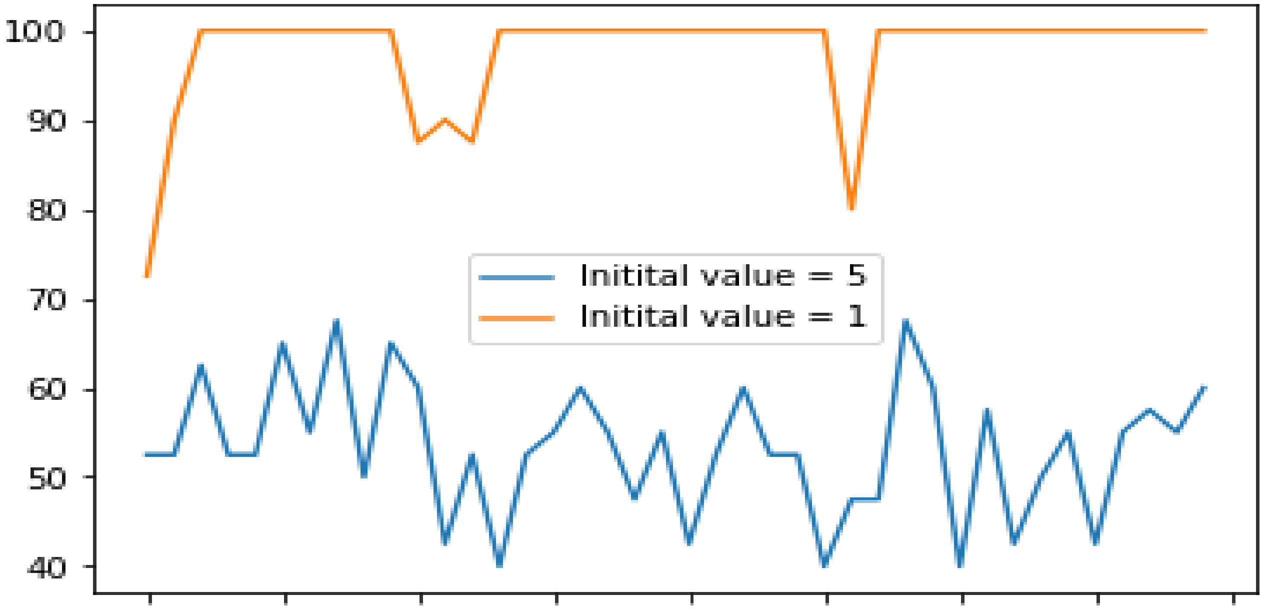
NOTE: Here the ‘optimal values’ were referred from the running percentage winning rate of the Q Learning algorithm while playing against MCTS_{50}.

2. **Changing gamma**
    - Gamma was varied with multiple values - from 0.2 to 1.0 at intervals of 0.2 each - to check the optimal result.
    - Again, it is noticed that the Q Learning algorithm very quickly learns the optimal moves and ends up winning most of the games, regardless of the gamma value (at least in the observed range).
    - It is seen that some values of gamma lead the Q Learning algorithm to perform better than others. Here, values close to 0.5 in general are performing the best; and the value of gamma = 0.8 gives the best results.
    - This value of 0.8 can be explained by the reasoning of 0.8 being a high enough value that considers future rewards in good regard.



3. **Changing the initial values of the entries in the Q-Table**
    - Changing the value of the initial entry in the Q-Table affects how much the algorithm has the freedom to explore a state multiple times, even if the first few times have led it to failure.
    - It was observed that although a higher value in the initial stages encourages exploration in the beginning stages of the algorithm, the same higher value misleads the algorithm to pick the wrong choices multiple times.
    - The values tested as the initial value of the table were: 1, 5 and 8
    - The initial value of 8 did not work well, and has hence not been recorded properly.
    - The value of 5, hence starts aggressively, but soon deviates from optimality and reaches no higher accuracy than it started. Conversely, although the value of 1 causes a slower ascent, it will eventually lead to a better and more stable state of the table.

NOTE: Here the 'optimal values' were referred from the running percentage winning rate of the Q Learning algorithm while playing against MCTS$_{50}$.



c. **MCTS vs Q learning**

The state of the board was represented by using a string of length 30, instead of as a 6 x 5 board, to reduce space and complexity of storage and hence increase efficiency.

- The Q-Table generated was saved in a .dat file and zipped into a tarball.
- A general reward of -1 was given to the algorithm on every move, which was done to discourage taking more moves to win.
- A reward of -1 was given for drawing the game, and no special reward was given for losing the game.
- A reward of 100 was given on every win, to compensate for the negative rewards given on any move leading to the victory.

NOTE:

There were 2 or 3 instances where the code went into an infinite loop while making a decision to move, even when there were places open for the particular state. This was a very rare occurrence (only twice or thrice in about 100 executions (with at least 100 iterations each)), but I was unable to find the bug and fix it.

The part ‘c’ of the code does not work as it ideally should have. I was unable to properly convert to a `.dat.gz` file and read from it, and am reading from a `.dat` file directly.