MDP Report – Ji Hoon Oh s43789991

1. Defining My MDP Problem

- **State: State Space**
  - In my particular implementation, the each state would represent a its position, (String) driver, (String) car type, (Tire) tire model, (TirePressure) pressure of the tire, (int) fuel level, as well as whether the car is currently in a slip state or a breakdown state.
- **Action: Action Space**
  - In this particular implementation, an action would be dependent on the level of the problem. Generally, it would be modifying one of the variables in each state. For example, an action in the action space would be 'change driver' mapping this particular action to the state would result in a next state with a different driver.
  - An actual list of all actions would be:
    1. Continue Moving -> Retains all characteristics of previous state except for position, possible slip position as well as breakdown. On harder levels, it would affect fuel and tire pressure.
    2. Change Car -> Retains all characteristics of previous state except for the type of car, which can affect move (speed) at next action, fuel efficiency in harder levels, as well as resetting all tires to 100% capacity.
    3. Change Driver -> Retains all characteristics of previous state except the driver for the next state. Different drivers will affect next state's move speed.
    4. Change Tires -> (4 Types of tires). Retains all characteristics except tire model. Mapping this action to the state will result in resetting tire pressure to max as well as affect how far the car will move in a single step.
    5. Add fuel -> Mapping this action to the state will reset fuel integer or add 0-50 integers to initial fuel level. Will affect steps (essentially slowing down) based on fuel amount.
    6. Change Tire Pressure -> Adding or reducing tire pressure of current State moving onto the next state. Mapping this action to the state will affect fuel consumption and probability of slip.
    7. Combo of 2-3 and Combo of 4-6.
- **Transition: Transition Function**
  - T(S, A, S`)
  - Transition is the mapping of actions to states or states to actions that results in the next state. In this particular problem, the goal is to reach the goal cell position = 10 or 30 (based on level) within the boundaries provided in the problem spec (maximum steps). Transition would be the mapping of current state with the most optimal action and the next state that is closer to reaching the goal position.
- **Reward: Reward function**
  - Reward function implementation would 'reward' an agent per every step (by action) that is closer to the goal step. Reaching the goal cell would result in a large sum (e.g., 1).
  - In my online (MCTS) implementation, a leaf node that is not in the terminal state would be simulated for random plays until it reaches a terminal state. Upon reaching a terminal state, it would be backpropagated with a reward of 1 and appropriate node visit count.

2. My Methods

- My implementation of the solution to the MDP problem is the utilization of the (online) MCTS algorithm. To simplify, it is a probabilistic search / decision making algorithm that is efficient in traversing a significant amount of possibilities or possible states. One of the limitations of implementing the online MCTS algorithm to solve an MDP problem is that games or environments with enormous amounts (millions) of different possibilities such as (e.g., Go) would result in time and memory complexity issues. Although explained in detail in part 3, large states would result in the time complexity of O(m*k*I/C) where m is the number of children to consider in a search, k being the number of parallel searches, and I being the number of iterations as well as C being the number of available nodes.
- It is also important to note that in my implementation of all available actions fail to switch drivers, tires, and car types to a different type if the state is already being occupied by a type. However, I am confident that a better implementation of adding all available actions into an array of actions will be a solution that doesn't require any changes to the MCTS methods themselves. Given a correct list of actions, I am confident that the algorithm should work.

Main Pseudo-Code:

Loop:

While(!node.isLeaf() && terminal state)

selectNode(currentNode) <- selects the best node (child) based on a calculated UCT value

expand(currentNode) <- for all available actions (per level) add to the current Node the states that is the result of an action

simulate(currentNode) <- simulate from this node (until goal is reached) and yield a result/reward based on the result of the simulation

backpropagate(currentNode, simulateValue) <- given the simulation results/rewards, update the current Node values and visit count

- As these loop, the value of the appropriate child Nodes and parent Nodes update (their UCT Value). Selecting Node then will pick the node with the highest UCT value which is correlated with the best action (action that leads to the optimal state).
- Simulation method will eventually converge to an accurate reward value that will backpropagate the UCT value of the node and parent node to encourage optimal selection of nodes.

Extra Notes for Pseudo-Code:

1. New instances of actions (based on level) stored In Array list of actions to be randomly selected in simulation.
2. State mapped with Action yields State` as provided by supporting code
3. Node class that represents States (as described in state space Q1), State Value and State Visit Count (For UCT Calculation), as well as Action associated with the state.
   Each child node would be a State` resulted by mapping Action to a previous State.

3. Analysis

Time/Memory Complexity:

Study 1:
In level one cases, the environment is laid out as a 1D N=10 grid with each cell that can be represented as the position of the agent. According to a Stanford study found via shortened url: https://stanford.io/2RSeeSW , with 4 different actions that can be mapped to the state, there, per cell or position, can be S (10) * A (4) different possible iterations. This brings us to an almost O(m*k*I/C) big O notation where m is the number of children to consider in a search, k being the number of parallel searches, and I being the number of iterations as well as C being the number of available nodes. Assuming my implementation is of standard MCTS format, it is likely that the implementation matches the upper bound of O(m*k*I/C).

Study 2:
There has been another study conducted by Pierre et al. accessed via shortened url: https://bit.ly/2QExBPf , which analyses the average game run time using MCTS. While the assignment implementation isn't decided on wins or losses, the expansion of nodes until a terminal state is reached is similar. The study considers a board game of 20x20 different cells to be visited. In comparison to the assignment, while it is a 1D environment, cells can move left and right (negative position/position position) with actions ranging from 4 different possibilities to 8 (or 10+ if you consider the last two actions being a combination of multiple actions). Regardless, the number of nodes to expand is in par or close to equal in comparison, leaving a lower bound (calculated for level1) of $O(N^2/\log_2 N)$, ignoring constants.

Argument:
My implementation (or attempt at implementing) is a single-threaded implementation with parallelized algorithm that relies on no recursive methods. The implementation is a simple simulation where four methods are called:

1. Node Selection: Which is ultimately dependent on the level of the tree after fully expansion. Ultimate traversal after all the node values have been assigned can be done at constant time of O(n), with N being the level of the tree (ignoring other child nodes). Selection within the simulation can also be done at O(n) time according to my implementation.
2. Node Expansion: Node expansion theoretically can be done at constant time as a pointer to the node to be expanded is already established. Iterating all available actions in the available actions list can be done at or worse case O(n) depending on the size of the list. Adding onto a node given the action can be done at constant time.
3. Node Simulation: Simulation is conducted via randomly mapping available actions onto a node (current State) until a goal is reached. This, I would argue can be accomplished in O(n) time depending on the luck or timing of reaching the goal state.
4. Backpropagation: Assigning node values all the way up to the parent node or the root node can be argued to be done at O(n) time as a pointer to the initial node is already established as well as to the parents (in my implementation). Assigning values is

constant but traversing up the tree (back the other way) would depend on N level of the tree.

5. Entire Operation: Entire operation, if my argument is correct, would be accomplished via O(n) by removing low order operations in Big O notations. However, based on the studies of other variations of implementing MCTS, unless my selection, expansion, and simulation are implemented flawlessly, could be argued to take longer, as exemplified in Study 1.