University of Queensland

AI Assignment 2 Report

COMP 7702

Yifei Wang & Guan Qin

45114917 & 44679129

1. Define MDP problem:

State:

For state description, each state consists of following attributes:

{Terrain Type (NT), Driver Type (DT), Tire Type, Tire Pressure, Fueling level.}

- Terrain type that the state current is.

Car type is a string which decides the different probability on action A1.

- Driver name is a string which decides the different probability on action A1.
- Tire type decides the different probability on action A1.
- Tire pressure decide the slip possibility and fuel consumption.
- Fuel level describes how much fuel is left in the tank

Furthermore, there are more detailed attributes can be assigned to the MDP state to help solving the problem:

There are {cells moved, steps, slip condition, breakdown condition and so on}.

Action:

There are totally eight main actions can be made for each step:

- A1: Move
- A2: Change car type
- A3: Change driver
- A4: Change existing cars' tire(s)
- A5: Add fuel
- A6: Change pressure
- A7: Combination of A2 and A3
- A8: Combination of A4, A5 and A6

The available actions are varying on the problem level, for level 1, only A1-A4 are available, for level 2 and level 3, actions A1-A6 are possible, for level 4, A1-A7 are available for car to choose. For level 5, all eight main actions are available.

However, for each main action the car can be changed into any different available options specifically. So, the total available actions for level 5 are:

$$A1 + A2 * \text{CarTypeNum} + A3 * \text{DriverNum} + A4 * \text{TireNum} + A5 * \text{FuelUnit} + A6 * \text{PressureLevels} + A7 * \text{CarTypeNum} * \text{DriverNum} + A8 * \text{TireNum} * \text{PressureLevel}$$

It can be significantly reduced to increase the efficient of our solve method, which will be further introduced in the question 3 part.

Transition (Probability):

Basically, this part is described in the input file where we can get probability to calculate transition matrix to implement S_{old} x A->S_{new}.

With a given state, we can use the possibilities by its car type, driver type, tire

type, tire pressure and terrain type to calculate the final possibilities array by using modified Bayesian equation. The equation is given as:

$$\mathrm{P}(\mathrm{D}=\mathrm{d}|\mathrm{B}=\mathrm{b},\mathrm{C}=\mathrm{c}) = \frac{P(B=b|D=d)P(C=c|D=d)P(D=d)}{\sum_{all\ possible\ d} P(B=b|D=d)P(C=c|D=d)P(D=d)}$$

Reward Function:

Because an online method is used to solve this MDP problem, so a reward function is given at end of each simulation. For this case, the reward function is highly depending on three parameters: cells moved, steps used and goal condition.

Firstly, we determine the goal condition by checking the cell position at end of each roll out process. If it reaches the goal, the reward function is:

$$\mathrm{R} = \mathrm{MaxT}/\mathrm{UsedT} * \mathrm{Level}$$

Here is the further explanation, the maxT is the maximum steps it allowed to solve, usedT is the steps that car have used to reach the goal. So, the sooner the car reach to goal, the higher reward will be received. Level is the problem level to solve, which is a weight parameter added to both reward equation. The reason of this setting is because we found that it is more difficult to reach the goal for higher level game, so this weight helps ensure a bigger weight will assigned when problem level become higher.

For the condition that it is not get to the goal when the max steps reached. A different reward function is given:

$$\mathrm{R} = \mathrm{cellsMoved}/(\mathrm{cellsNeed} * \mathrm{Level})$$

Although in most of the case the car never reaches to the goal for the roll out simulation, however, the effective of series of decision can still be judged by using this equation function. The cellsMoved represent the cells that car has moved when max steps reached. The cellsNeed is the number of cell that the car required to moved to reach the goal from initial position. Opposite to the reward function above, the higher game level is, the lower reward will be received when it not reaches to the goal. A zero reward is assigned if the car's position doesn't change as the initial.

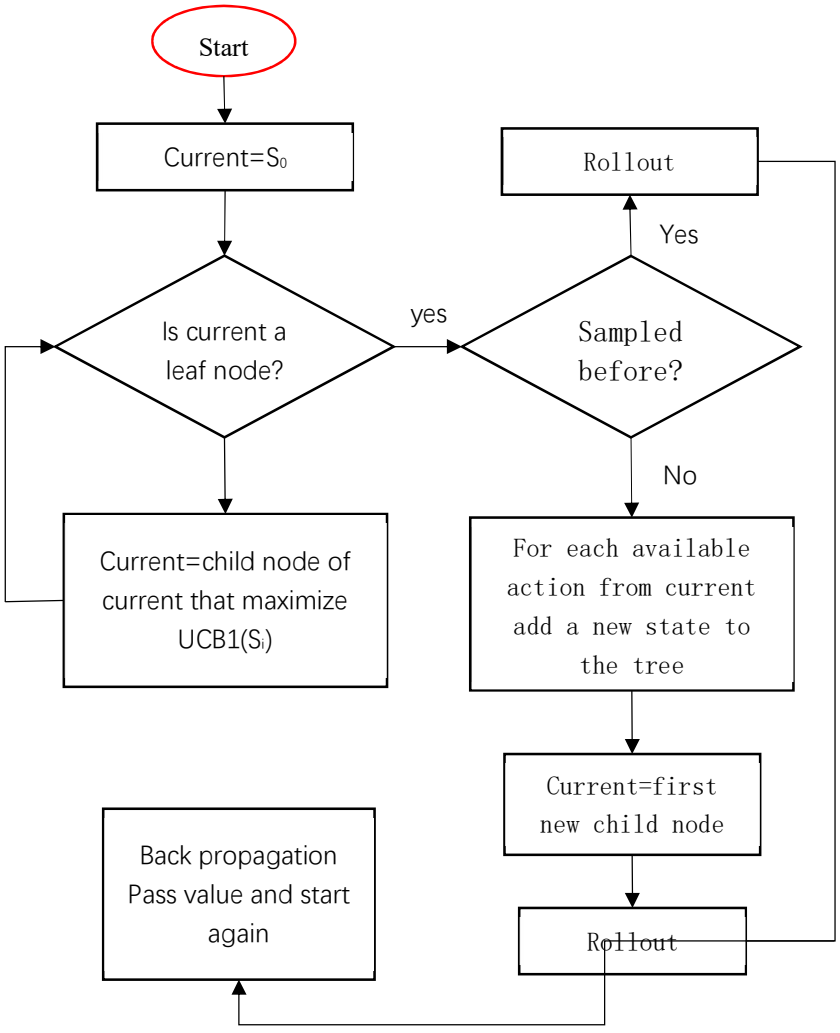
For comparison, we have compared our result with this reward function to simply assigned 1 to win and 0 to lose reward function on same test input. For level 1,2,3 the success rate has increased 5~10%, and 70~80%for level 4 and level 5.

2. Solve method used for MDP problem:

Pseudocode and Data Structure

We used a varied MCTS (Monte Carlo Tree Search) to solve the problem, and the data structure is for MCTS is a tree structure. For each of the tree node, it stores all the state attributions and the previous action which used to reaches to the state. There are more information stores in the tree node such as number of visit and the value received by reward function which helps to decide the best children tree node. The tree node contains the parent node and children node's information to construct the Monte Carlo Tree and used in the method like roll out.

The main pseudocode is described as the flow chart below, and more detailed modification is described later in part 3.



UCB function as well as the parameters are notified as below:

$$\mathrm{UCB1(v)} = \frac{Q(v')}{N(v')} + \sqrt{\frac{2\ln N(v)}{N(v')}}$$

f: S x A->S : state transition function
N(v): how many times the node was visited;
Q(v): the vector of rewards from simulated playouts;
s(v): associated state;
a(v): incoming action;
△(v; p): the function that associates the reward from Q(v)

3. Algorithm analysis

Original action sizes and reduced action sizes

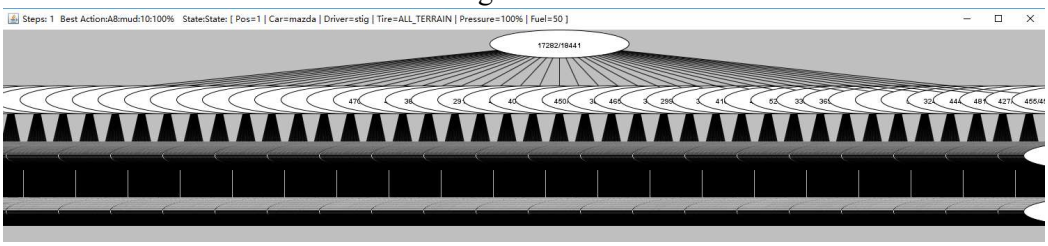
By the equation of action number introduced in answer of question 1 above, it is apparent that the number of actions increase dramatically when problem level is higher. For the given input test cases, the action number for level one is less that 10, but the number increased to around 60 for level 5 question. Since online MCTS method is applied and a tree structure is used, that suggests the width of three size is dominated by the action sizes when the tree depth grow. In other words, if a goal can be reached at expected tree depth, the more actions available means it will be more difficult of the search tree to reach to that depth.

To reduce the action sizes, we first defined some limitation when randomly choose an action. Firstly, it's forbidden for the car to swap into itself for any one action, as well as swap into itself after several actions without move. To done that, a method is defined to return a tree node after scan through all tree node which stored for a user defined memory length.

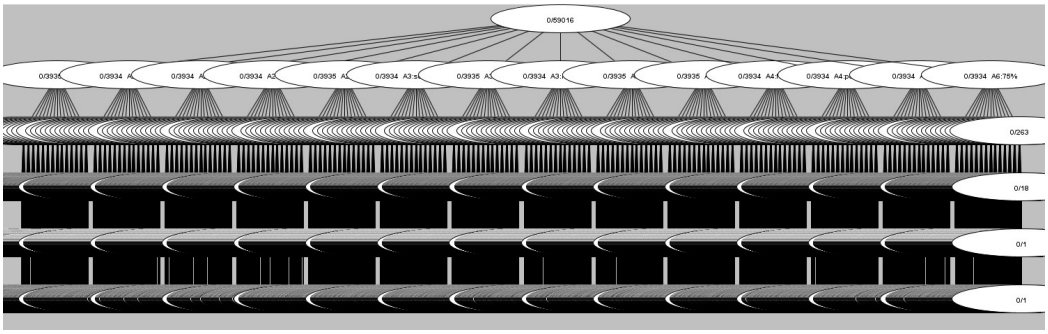
After the limitation, the actions are still too much for MCTS to have an efficient search. For this reason, we spited actions into two categories, single action and combined action, which separately represent action A1, A2, A3, A4, A5, A6 and A7, A8. The core idea is that whenever a combined action is valid, the designed logic make it always consider to use combined action first instead of multiples single actions. It's needs to mention that for A5(fueling) is not considered in the end because we found that it can be efficient replaced by switch similar car or A8.

We are able to draw the tree for same time steps length for both cases with original action number and with reduced action number as below. Note that since the tree node produced during roll out is not produced so the tree is only shown partially

Case with original action sizes



Case with reduced action sizes



For the result comparison, the pass rates for level 1 and 2 are not really impacted, however, for level 3,4,5 the pass rates increased roughly 40~60%

Analysis on tree node size and memory usage

The tree node that stored in the memory for each time steps is not as high as imaged because only the tree node that generated during expansion and exploration processes are stored. The new tree node in the roll out stage is never stored except the value returned at end of each roll out process. The two diagrams in last question also proved it.

Because we are not completely choosing a random action, instead we choose randomly best from stored list (for a length of 20 in code), the calculation time for each simulation is increased. However, the list is discarded every time when a best action is choosing so the memory cost can be ignored compare to the memory usage to search tree.

After each required time steps passed, a new tree will be generated with next state as the new tree node. This is suggesting that the previous search tree will not be stored so it will not cost further burden to the memory usage.

Overall analysis

We have run our MCTS algorithm on the test set that released on 11/10th for 15 times for all five game levels with three second time steps between each move. The success rates are 100% for all tests except there was one failure on level one. For most times, the average steps cost for the car to reach goal is around 30%~60%. This statistic suggested that the MCTS algorithm we designed is very efficient to solve this game changing technology problem.