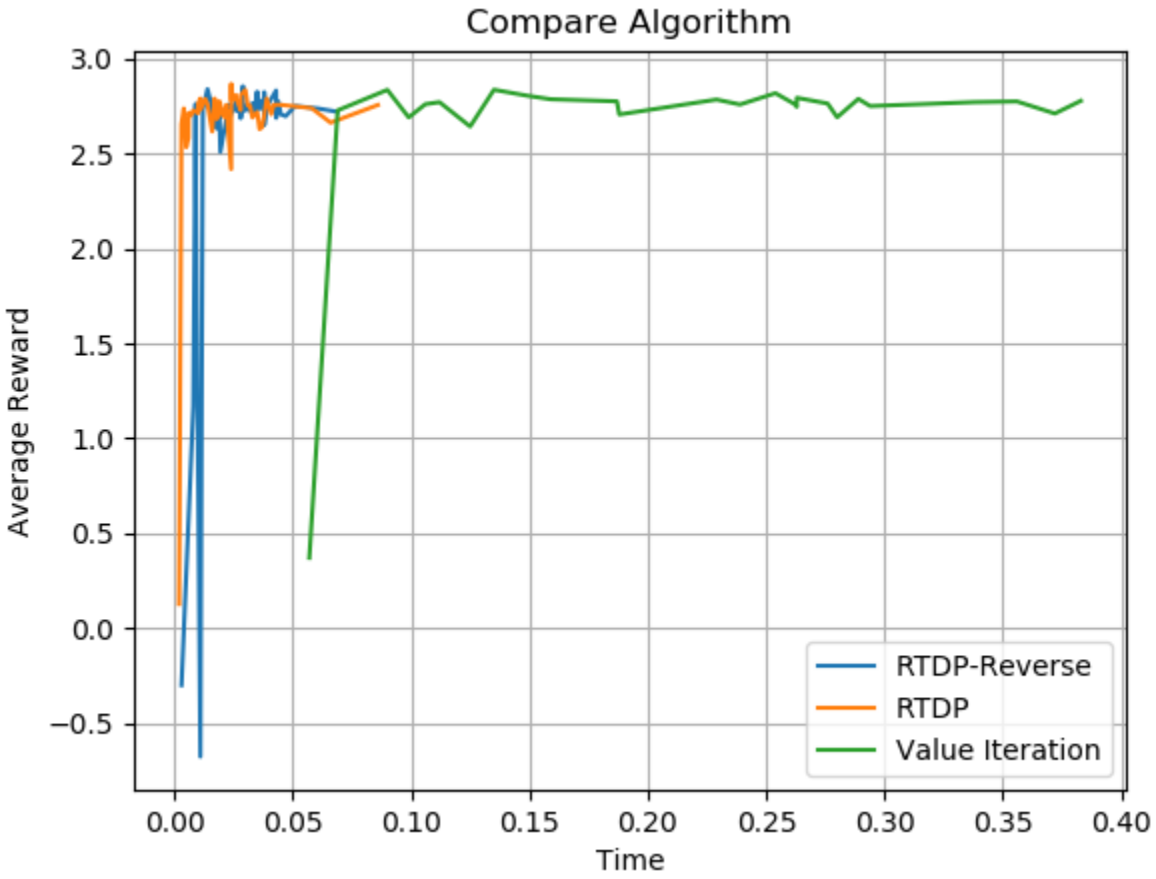
RTDP Report

1 Performance

In here, I use 100 iteration in planning, and run 1000 episodes for calculating the average time. And for **RTDP** and **RTDP-Reverse** algorithm, followed the **LRTDP** algorithm that labeled the terminal statues at beginning.

Here is the compare plot graph. For original program output please check *output.txt* file.



From the experience and this diagraph, we can find that:

- **Value Iteration**, can converge to the optimal policies but will require more time to do the planning for the better performance
    - And will not find the optimal policies when planning within 4 iterations, and this cause, during the episodes of test, agent cannot find the terminal status.
- By introducing the Heuristic function, Both **RTDP** and **RTDP-Reverse** can explore more probable state first and thus:
    - Can plaining faster to find the optimal policy
    - Can reach the terminal status even don't have trail before, like within 4 iterations of the plaining these 2 algorithms can still find the policy to reach terminal states.

- And the **RTDP-Reverse** performance worse than **RTDP** algorithm
    - I think the reason should be at first few iteration, we will still use the state value from heuristic function in other states and since the heuristic function is admissible, this will cause the agent cannot choose the optimal policies based on the heuristic value.

2 Heuristic Function

For Heuristic Function, the admissible will means that this function never overestimates the cost of reaching the goal. With same idea, since in our project, the heuristic function RTDP will use is for the reward, we should find a function that can never underestimates the reward of this state to reaching the goal.

Thus, the Heuristic Function I choose is the following:

$$h(s) = R(goalState) * \gamma^{manhattan\_distance(s,goalState)}$$

In this function it can be sure that $h(s) \geq V^*(s)$, because of the following ideas:

- Manhattan distance doesn't count in the block that cannot be moved.
- Doesn't check the noise, so will always compute the value like noise as 0, which will let value higher.
- Only use the goalState, thus ignore the sub-optimal terminal states.