CS533-25Winter

Aikeremu Daniyaer

ID: 934559396

Mini Project 1 Report

Part A:

1. Environment setup
   (1) State: Each state is represented as (x, y, water, fire) where water and fire are binary values (0 indicating absence, 1 indicating presence). The start state is (0,0,0,0) and the goal state is (3,3,0,0). The fire cells are at (0,1) and (0,2) (represented as (0, 1, 0, 1) and (0, 2, 0, 1)), and the cells with water are at (2,1) and (2,2) (represented as ((2, 1, 1, 0) and (2, 2, 1, 0)). All other cells are considered normal (e.g., (x, y, 0, 0))
   (2) Action: The agent can move in four directions: up, down, left, and right (0: up, 1: down, 2: left, 3: right). The intended move succeeds with a probability of 0.8, and there is a 0.1 probability to "slide" to each of the two perpendicular directions. If a move is invalid (e.g., moving off the grid), the agent remains in the same state.
   (3) Reward: Reaching the goal state (3,3,0,0)(3,3,0,0): +100. Entering a fire state: -10. Entering a water state: -5. All other transitions incur a movement cost of -1.
   (1) Gamma = 0.95 and theta = 1e-6

2. Value iteration

   I implemented a value_iteration function that initializes the state-value function V(s) to 0 for all states and iteratively updates it using the Bellman optimality equation:

   value = sum(prob * (get_rewards(s) + mdp["gamma"] * V[s])

   Compare the output from gamma = 0.3 and gamma = 0.3, it clearly shows that when the discount factor is set to gamma = 0.3, future rewards are heavily discounted, so the agent primarily focuses on immediate costs and benefits. As a result, the computed state values remain very low (or even negative) – for instance, the start state (0,0,0,0) has a value of -1.42, and even hazardous states such as the fire cells show low values.

   In contrast, with a high discount factor of gamma = 0.95, future rewards receive much more weight, leading to significantly higher state values overall. For example, the start state (0,0,0,0) attains a value of 63.49, and states along the optimal path exhibit values in the range of 80 to 97. This demonstrates that the agent, by considering longterm rewards, is inclined to choose safer paths that avoid hazards, even if the immediate cost is slightly higher.

Output:

```
Final state value function V from Value Iteration (gamma=0.3):
  State (0, 0, 0, 0): Value -1.42
  State (0, 1, 0, 1): Value -2.35
  State (0, 2, 0, 1): Value -1.17
  State (0, 3, 0, 0): Value 2.82
  State (1, 0, 0, 0): Value -1.18
  State (1, 1, 0, 0): Value -1.54
  State (1, 2, 0, 0): Value 2.98
  State (1, 3, 0, 0): Value 19.48
  State (2, 0, 0, 0): Value -0.41
  State (2, 1, 1, 0): Value 3.81
  State (2, 2, 1, 0): Value 21.37
  State (2, 3, 0, 0): Value 82.52
  State (3, 0, 0, 0): Value 3.68
  State (3, 1, 0, 0): Value 19.09
  State (3, 2, 0, 0): Value 82.52
  State (3, 3, 0, 0): Value 0.00
Objective Value (gamma=0.3): -1.42

Final state value function V from Value Iteration (gamma=0.95):
  State (0, 0, 0, 0): Value 63.49
  State (0, 1, 0, 1): Value 66.83
  State (0, 2, 0, 1): Value 73.89
  State (0, 3, 0, 0): Value 80.37
  State (1, 0, 0, 0): Value 69.75
  State (1, 1, 0, 0): Value 73.26
  State (1, 2, 0, 0): Value 80.87
  State (1, 3, 0, 0): Value 88.97
  State (2, 0, 0, 0): Value 75.21
  State (2, 1, 1, 0): Value 81.62
  State (2, 2, 1, 0): Value 89.75
  State (2, 3, 0, 0): Value 97.16
  State (3, 0, 0, 0): Value 81.20
  State (3, 1, 0, 0): Value 88.61
  State (3, 2, 0, 0): Value 97.16
  State (3, 3, 0, 0): Value 0.00
```

```
Objective Value (gamma=0.95): 63.49
```

3. Policy iteration

The policy_iteration function was implemented in two steps:

(1) Policy evaluation:

Start by setting every state's value to 0. Then, update each state's value by adding the immediate reward to the discounted sum of the values of possible next states, following the Bellman equation. Repeat these updates until the maximum change in any state value is below a small threshold, ensuring the value function is accurate.

(2) Policy improvement:

Then, update the policy by evaluating every possible action for each state. For each state, calculate the expected return for all actions and select the action with the highest return as the new action for that state. If the new policy is identical to the old one for all states, the process stops and the policy is optimal; otherwise, repeat the evaluation and improvement steps.

Output:

```
Final state value function V from Policy Iteration (gamma=0.95):
  State (0, 0, 0, 0): Value 63.49
  State (0, 1, 0, 1): Value 66.83
  State (0, 2, 0, 1): Value 73.89
  State (0, 3, 0, 0): Value 80.37
  State (1, 0, 0, 0): Value 69.75
  State (1, 1, 0, 0): Value 73.26
  State (1, 2, 0, 0): Value 80.87
  State (1, 3, 0, 0): Value 88.97
  State (2, 0, 0, 0): Value 75.21
  State (2, 1, 1, 0): Value 81.62
  State (2, 2, 1, 0): Value 89.75
  State (2, 3, 0, 0): Value 97.16
  State (3, 0, 0, 0): Value 81.20
  State (3, 1, 0, 0): Value 88.61
  State (3, 2, 0, 0): Value 97.16
  State (3, 3, 0, 0): Value 0.00
```

The state values and policy from policy iteration are identical to those from value iteration, showing that both methods reach the same optimal solution.

Part B:

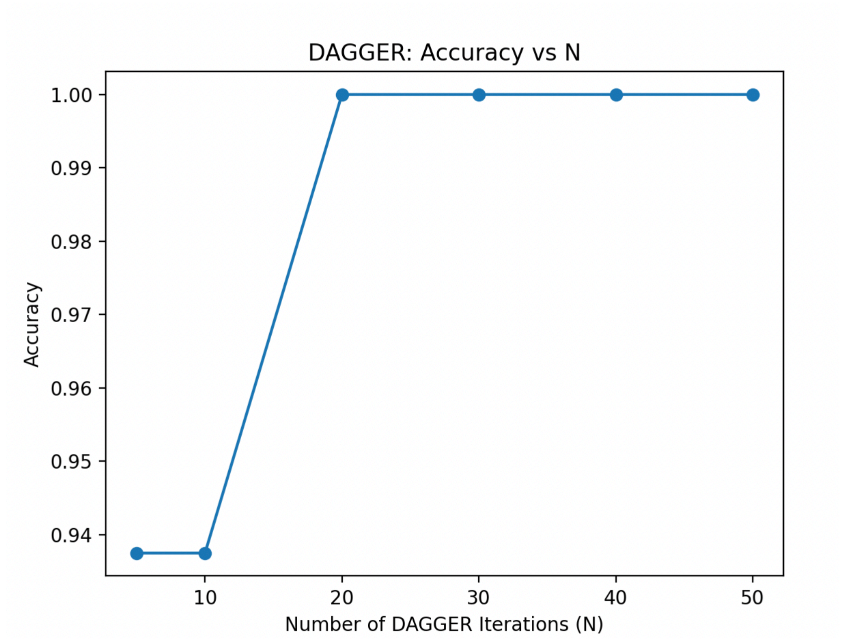
1. Policy simulation

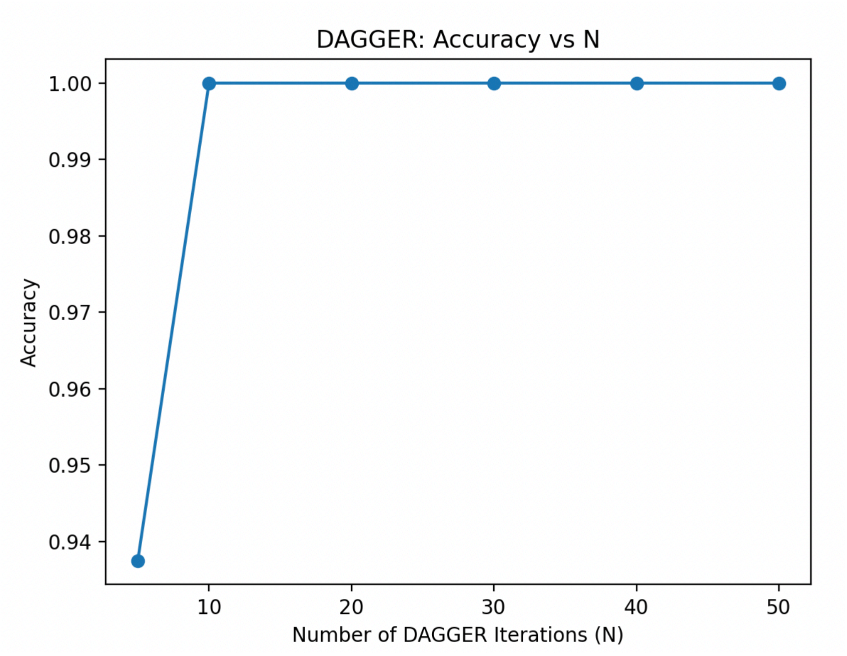
   Starting at (0, 0, 0, 0), the function repeatedly uses the current policy to select an action, then samples the next state according to the transition probabilities, and records the (state, action, reward) tuple. This loop continues until the goal state (3, 3, 0, 0) is reached (or a maximum step count is met), and then the complete trajectory is returned.

   Output:

```
Episode (using Value Iteration policy, gamma=0.95):
Step 0: State (0, 0, 0, 0), Action 3, Reward -1
Step 1: State (1, 0, 0, 0), Action 3, Reward -1
Step 2: State (1, 1, 0, 0), Action 1, Reward -1
Step 3: State (1, 2, 0, 0), Action 1, Reward -1
Step 4: State (1, 3, 0, 0), Action 3, Reward -1
Step 5: State (2, 3, 0, 0), Action 3, Reward 100
Step 6: State (3, 3, 0, 0), Action None, Reward 100
```

2. DAGGER Algorithm
   (1) According to the question, a random policy is first initialized (ensuring it is different from the expert policy) and an empty dataset D is created.
   (2) In each iteration, a complete trajectory is generated using the current learned policy. For each state in the trajectory, the expert policy (obtained from Part A with gamma = 0.95) is used to label the state with the correct action, and the state-action pair is added to D.
   (3) A Decision Tree classifier is trained on the aggregated data, mapping state features (e.g. [x, y, water, fire]) to expert actions. The trained classifier predicts actions for all states in the MDP, updating the learned policy.
   (4) The accuracy is computed as the fraction of states for which the predicted action matches the expert action. Experiments were conducted with different iteration counts (N = 5, 10, 20, 30, 40, 50), and the results (as figure below) show that with more iterations, the aggregated data becomes more comprehensive and the learned policy's accuracy improves.





The results show that as N increases, the aggregated dataset covers more of the state space, and the learned policy becomes closer to the expert policy. This improvement is reflected in the accuracy, which rises steadily and eventually reaches 100% for several iteration counts.

Reference

[1] https://arxiv.org/pdf/1011.0686

[2] CS182 : Lecture 14-15
https://www.youtube.com/watch?v=a5wkzPa4fO4&list=PL_iWQOsE6TfVmKkQHucjPAoRtIJYt8a5A&index=44