National Taiwan University
Department of Electrical Engineering
Name: Chen-Han, Lin
ID: R12942083

113-1 (Fall 2024)
Reinforcement Learning

Assignment 1 Report

Q1. What methods have you tried for async DP? Compare their performance.

In my implementation, I have explored several asynchronous dynamic programming (DP) methods, including In-place DP, Prioritized Sweeping combined with In-place DP, the Successive Over-Relaxation (SOR) Value Iteration method combined with In-place DP, and Wall DP combined with In-place DP and Prioritized Sweeping.

The SOR method updates the state values using the following equation:

$$V_{k+1}(s) \leftarrow V_k(s) + w(V_{new}(s) - V_k(s))$$

where w is a parameter. When $w = 1$, the SOR method reduces to regular value iteration.

In Wall DP, we track state-action pairs that cause the agent to hit a wall, avoiding unnecessary steps for these pairs during value iteration.

Performance Comparison:

async methods	In-place DP	Prioritized sweeping	SOR(w^*=1)	Wall DP
#_of_steps	1056	668	1056	**484**
#_of_updates of states	242	145	242	**145**

Table 1: Performance comparison of different async DP methods

In SOR method, we sample different w to find its optimal value:

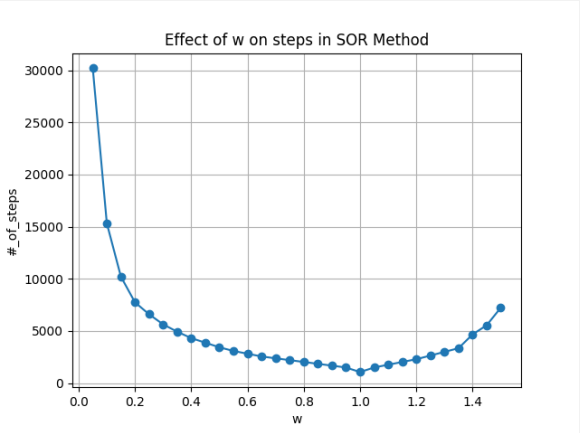

Figure 1: Effect of w on steps in SOR method. The number of steps required by the SOR method varies with different values of w, and the optimal value is found to be $w^* = 1$.

Using synchronous value iteration, which takes 1144 steps to solve the sample Markov Decision Process (MDP) as a baseline, we observe that all asynchronous DP methods outperform it in terms of efficiency.
Compared to sync value iteration, In-place DP updates the value with values which is the newest, instead of values at the last iteration, so it reduces the redundancy when updating and raises the

efficiency.

- **In-place DP**: Compared to synchronous value iteration, In-place DP updates the value of each state using the most recent information, rather than relying on the values from the last iteration. This approach reduces redundancy in the update process, leading to improved efficiency.
- Prioritized Sweeping: In this method, In-place DP is implemented alongside a mechanism that prioritizes states based on their urgency, which is determined by calculating each state's Bellman error. This allows for a more efficient propagation of information across the MDP. As a result, fewer steps are required, and the number of state updates is also reduced compared to In-place DP.
- SOR Method: Referring to fig1., for the sample MDP, we fine-tuned the parameter w and discovered that the optimal value is $w^* = 1$, effectively reducing the SOR method to In-place DP in this case.
- Wall DP: Wall DP incorporates techniques from both In-place DP and Prioritized Sweeping while also remembering state-action pairs that would cause the agent to hit a wall. These pairs are not updated during the value iteration process, as they do not contribute to solving the MDP. This memory significantly reduces the total number of steps needed to solve the MDP, but the number of state updates remains unchanged. By combining these techniques, Wall DP achieves the best performance in terms of efficiency.

Q2. What is your final method? How is it better than other methods you've tried?

As concluded in Q1, although Wall DP is the most efficient in terms of steps, I have chosen the prioritized sweeping method as my final approach because it strikes a good balance between efficiency and memory usage.
While Wall DP solves the sample maze using the fewest steps, it requires more memory compared to the prioritized sweeping method. In addition to the state values, Wall DP requires two extra arrays of size #_of_states×4: one to store the Bellman error for each state and another to track which state-action pairs cause the agent to hit a wall. Furthermore, Wall DP only works under the assumption of deterministic state transitions, limiting its applicability. Therefore, due to its higher memory usage and limited generalizability to non-deterministic MDPs, I did not select Wall DP as my final method.
On the other hand, prioritized sweeping requires only one additional array to store the Bellman error for each state. This reduces the number of steps from 1056 to 668—a substantial improvement. Moreover, since it does not retain information about state-action pairs or state transitions after actions are taken, it can handle both deterministic and non-deterministic MDPs with a finite number of states, making it more broadly applicable.
In conclusion, I have chosen the prioritized sweeping method as my final approach due to its balance between efficiency and general applicability.