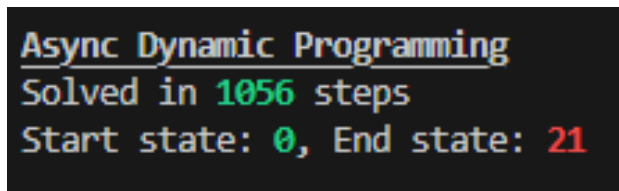


r12921a25
Chun An ,Yang

September 25, 2024

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

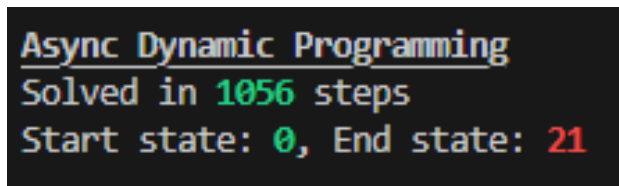
1. method 1: inplace DP
steps count : 1056 steps, start state : 0, end state : 21



```
Async Dynamic Programming  
Solved in 1056 steps  
Start state: 0, End state: 21
```

Figure 1: in-place dynamic programming result

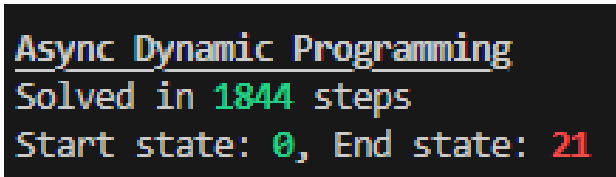
2. method 2: Prioritized sweeping
steps count : 1056 steps, start state : 0, end state : 21



```
Async Dynamic Programming  
Solved in 1056 steps  
Start state: 0, End state: 21
```

Figure 2: Prioritized sweeping result

3. method 3: Real-time DP
steps count : 1844 steps, start state : 0, end state : 21



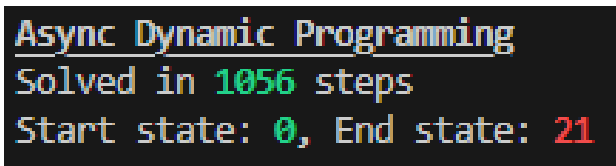
```
Async Dynamic Programming
Solved in 1844 steps
Start state: 0, End state: 21
```

Figure 3: Real-time dynamic programming result

For the reference , the result of value iteration is
1144 steps, start state: 0, end state: 21.

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

1. I choose Inplace DP as my final method.
reasonable explanation: First of all, it is really easy to implement , second, this method contains the property of normal value iteration that it will frequently and equally select every state to update, since PrioritySweeping adds priority to the update order, it is not frequently updated equally to each state and it may be affected by different scenarios of environment. Also, the RTDP (Real-Time Dynamic Programming) is updated as the real interaction to the environment , so we have to consider every state as initial state and sequentially run steps, so the update of every state will not be frequency equally.
2. novelty of new method
PrioritySweeping with consider every other state as neighbor state
steps count : 1056 steps, start state : 0, end state : 21



```
Async Dynamic Programming
Solved in 1056 steps
Start state: 0, End state: 21
```

Figure 4: Prioritized sweeping result

Since the original PrioritySweeping method requires the information of predecessor states, I revise this part to consider all the other state as neighbor state except the state itself. The reason of the improvement is to make the implementation more easier

but it may cost more steps, since this method will contain more consideration of the state's bellman error updates. However, the result is still better than value iteration.