# RL_HW1_gridworld

ShengHsun Chang

September 29, 2025

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

## 1.1 Summary

I've tried three asynchronous DP methods mentioned in class: In-place DP, Prioritized Sweeping, and Real-Time DP. The implementation details and performance are discussed in the following subsections.

## 1.2 Method1: In-place DP

I implement In-place DP base on value iteration method. The difference is that I update the value function $V(s)$ right after computing the new value, instead of waiting until the end of the episode. It is better than the original value iteration because it uses the up-to-date value. However, it still waste steps in states with little residual. The pseudo code is shown below.

---

**Algorithm 1:** In-place Dynamic Programming

---

**Input :** GridWorld $\mathcal{G}$, discount $\gamma$; threshold $\theta$
**Output :** Updated $V(\cdot)$ and greedy $\pi(\cdot)$

**Initialization:**;
$V(s) \leftarrow 0$ for all $s \in S$ ; `// initial value`
$\Delta \leftarrow 2\,\theta$ ; `// ensure at least one sweep`
**while** $\Delta > \theta$ **do**
  $\Delta_{max} \leftarrow 0$;
  **foreach** $s \in S$ *in a fixed order* **do**
    $v_{\text{old}} \leftarrow V(s)$;
    $q_{\max} \leftarrow -\infty$;
    **foreach** $a \in \{0, \ldots, |A|-1\}$ **do**
      $(s', r, \texttt{done}) \leftarrow \texttt{step}(s, a)$;
      **if** *done* **then** $q \leftarrow r$ ;
      **else** $q \leftarrow r + \gamma \cdot V(s')$ ;
      $q_{\max} \leftarrow \max(q_{\max}, q)$;
    $V(s) \leftarrow q_{\max}$ ; `// in-place overwrite`
    $\Delta_{max} \leftarrow \max\big(\Delta_{max},\, |V(s) - v_{\text{old}}|\big)$;
**foreach** $s \in S$ **do**
  $\pi(s) \leftarrow \arg\max_a \{r(s,a) + \gamma \cdot V(s')\}$;

---

## 1.3 Method2: Prioritized Sweeping

Prioritized Sweeping improves In-place DP by choosing the state with the largest residual to update. Right after updating a state, we push its predecessors back into an array(priority queue) In my implementtation, I use a simple list instead of a priority queue for:

- Simplicity: Python's built-in list is easy to use and understand.
- Duplication: The same state would be pushed multiple times, and the checking is too annoying.
- Efficiency: The performance in the test case is still too small.

It is worth noting that I have to initialize the value function $V(s)$ to $-\infty$ instead of 0. I think it is because the first residual can thus be quite large, and the update can propagate faster.

**Algorithm 2:** Prioritized Sweeping

**Input :** GridWorld $\mathcal{G}$, discount $\gamma$; threshold $\theta$
**Output :** Updated $V(\cdot)$ and greedy $\pi(\cdot)$

**Initialization:**;
$V(s) \leftarrow -\infty$ for all $s \in S$;
Pred$[s] \leftarrow \emptyset$ for all $s \in S$;
Compute initial residual$[s]$ for all $s$;
**while** *some residual*$[s] \geq \theta$ **do**
  $s^\star \leftarrow \arg\max_s$ residual$[s]$;
  Recompute $V(s^\star)$ by one-step backup:;
    $V(s^\star) \leftarrow \max_a\{r(s,a) + \gamma \cdot V(s')\}$;
  residual$[s^\star] \leftarrow 0$;
  **foreach** $p \in$ *Pred*$[s^\star]$ **do**
    Recompute backup value for $p$;
    residual$[p] \leftarrow |\,\text{backup}(p) - V(p)\,|$;
**foreach** $s \in S$ **do**
  $\pi(s) \leftarrow \arg\max_a\{r(s,a) + \gamma \cdot V(s')\}$;

## 1.4 Method3: Real-Time DP

Real-Time DP, different from the previous two methods, updates value function only along the trajectory generated by the current policy.
Therefore, to ensure all state policy are updated, I choose all states as the start state in a random order in each episode.
Despite these modifications, the performance is still bad compared to Prioritized Sweeping.

**Algorithm 3:** Real-Time Dynamic Programming

```
Input  : GridWorld 𝒢, discount γ; threshold θ
Output : Updated V(·) and greedy π(·)

Initialization:;
V(s) ← 0 for all s ∈ S ;                // initial optimistic values
LegalActions[s] ← all actions for each s;
Unsolved ← S ;                          // all states initially unsolved
while not converged do
    Choose a random permutation of unsolved states;
    foreach s ∈ permutation do
        if s is terminal then
            continue
        steps ← 0, max_steps ← 4|S|;
        while steps < max_steps do
            v_new ← max_a{r(s, a) + γ · V(s′)};
            diff ← |v_new − V(s)|;
            V(s) ← v_new;
            if diff < θ then
                remove s from Unsolved
            a* ← arg max_a{r(s, a) + γ · V(s′)};
            π(s) ← a*;
            (s′, r, done) ← step(s, a*);
            if done then
                break
            if s′ = s then
                remove a* from LegalActions[s]; break
            s ← s′, steps ← steps + 1;
```

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

## 2.1 Concept

In my asynchronous DP implementation, I find out that Prioritized Sweeping is the best method.
So, I implement an optimization based on Prioritized Sweeping.
In the current maze, if an action leads to the same state(a.k.a hit the wall), then this action is useless.
Therefore, I modify the Bellman equation to ignore these actions.

**Algorithm 4:** Optimized Prioritized Sweeping (Async DP)

**Input** : GridWorld $\mathcal{G}$, discount $\gamma$; threshold $\theta$
**Output** : Updated $V(\cdot)$ and greedy $\pi(\cdot)$

**Initialization:**;
$V(s) \leftarrow -\infty$ for all $s \in S$;
$\text{Pred}[s] \leftarrow \emptyset$ for all $s \in S$;
$\text{LegalActions}[s] \leftarrow$ all actions for each $s$;

**foreach** $s \in S$ **do**
  $q_{\max} \leftarrow -\infty$;
  **foreach** $a \in A$ **do**
    $(s', r, done) \leftarrow$ `step`$(s, a)$;
    **if** $s' = s$ **then** *remove* $a$ from LegalActions$[s]$ ;
    **if** *done* **then** $q \leftarrow r$ ;
    **else** $q \leftarrow r + \gamma \cdot V(s')$ ;
    $q_{\max} \leftarrow \max(q_{\max}, q)$;
    **if** $s' \neq s$ **then**
      add $s$ to Pred$[s']$
  $\text{residual}[s] \leftarrow |\, q_{\max} - V(s)\,|$;
  $\text{new\_values}[s] \leftarrow q_{\max}$;

**while** *true* **do**
  $s^\star \leftarrow \arg\max_s \text{residual}[s]$;
  **if** *residual*$[s^\star] < \theta$ **then**
    **break**
  $v_{\text{new}} \leftarrow$ `get_state_value`$(s^\star)$;
  $err \leftarrow |\, v_{\text{new}} - V(s^\star)\,|$;
  **if** $err < \theta$ **then**
    $\text{residual}[s^\star] \leftarrow 0$; **continue**
  $V(s^\star) \leftarrow v_{\text{new}}$;
  $\text{residual}[s^\star] \leftarrow 0$;
  **foreach** $p \in Pred[s^\star]$ **do**
    $v_{\text{pred}} \leftarrow$ `get_state_value`$(p)$;
    $e_p \leftarrow |\, v_{\text{pred}} - V(p)\,|$;
    **if** $e_p > \theta$ **then**
      $\text{new\_values}[p] \leftarrow v_{\text{pred}}$;
      $\text{residual}[p] \leftarrow e_p$;

**foreach** $s \in S$ **do**
  **if** *LegalActions*$[s] = \emptyset$ **then**
    **continue**
  $\pi(s) \leftarrow \arg\max_{a \in \text{LegalActions}[s]} \{r(s, a) + \gamma \cdot V(s')\}$;

# 3 Experiment Results & Discussion

## 3.1 Experiment Setup

I choose the sample map, and three gridworld map(10x10, 15x15, 20x20) generated by chat-GPT with different sizes. the maps are shown below.

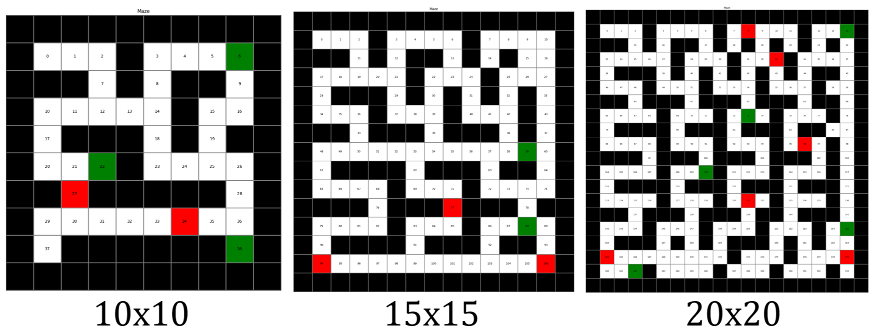


Figure 1: from left to right: 10x10 map, 15x15 map, 20x20 map

## 3.2 Experiment Results

To begin with, for the sample map, All the methods can converge to the optimal policy. like the figure below.

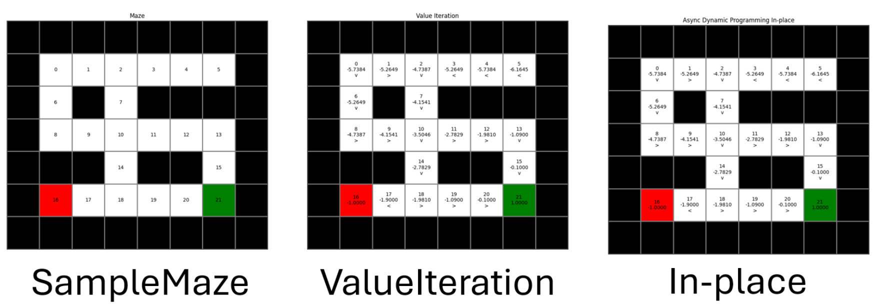


Figure 2: Optimal policy for sample map_1

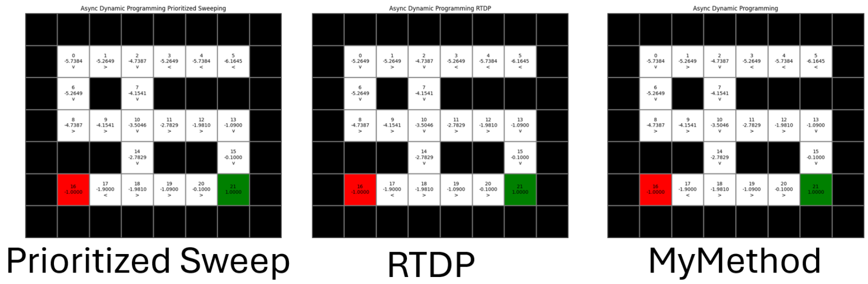


Figure 3: Optimal policy for sample map_2

| Method | Sample Map | 10x10 Map | 15x15 Map | 20x20 Map |
|---|---|---|---|---|
| Value Iteration(base line) | 1144 | 2028 | 8560 | 20952 |
| In-place DP | 1056 | 1716 | 8560 | 16296 |
| Prioritized Sweeping | 692 | 988 | 3044 | 5604 |
| Real-Time DP | 2243 | 2904 | 13386 | 26677 |
| Optimized PS | **469** | **690** | **2078** | **3850** |

Table 1: Required steps for convergence across different gridworld maps

### 3.3 Discussion

The Optimized method I propose consistently outperforms the other methods as shown in Table 1.
The reasons are as follows:

- The prioritized sweeping mechanism ensures that states with the highest residuals are updated first, accelerating the propagation of value updates through the state space.
- By ignoring useless actions, the Bellman update focuses on more promising actions, leading to faster convergence.
- The optimization reduces the number of states by 30-40% in all test cases, compared with the original Prioritized Sweeping method.