SC4003 Assignment 1

Koh Ming En

March 9, 2025

1 Introduction

This report will discuss Value Iteration and Policy Iteration, which are dynamic programs that compute optimal values for solving Markov Decision Processes (MDPs). To demonstrate the capabilities of both Value Iteration and Policy Iteration, we will test both algorithms in a Maze environment as shown in Figure 1. Section 2 will cover the overview of the algorithms, the resulting optimum policy and the estimated utilities of each state.

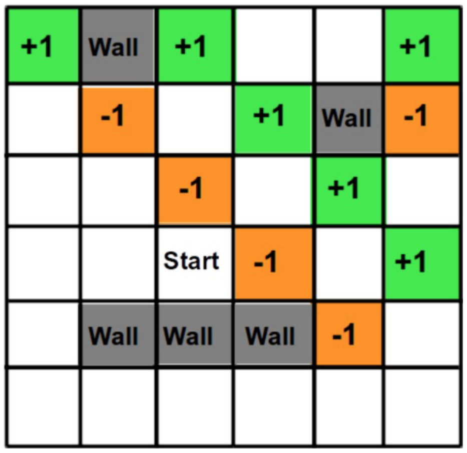


Figure 1: Maze used in Section 2

In section 3, we will analyse the time complexity of the two algorithms to estimate an upper bound on the complexity of the environment for these algorithms to terminate. We will also test the algorithms on increasingly complex maze environments to provide empirical evidence for the time complexity. W

This project was coded in Python, with the `Matplotlib` library used for plotting graphs and `tkinter` library used for visualizing the estimated utilities and optimum policies.

2 Solving the Maze

2.1 Value Iteration

2.1.1 Description

Value Iteration is a dynamic-programming method for finding the optimal value function V^* by solving the Bellman equations iteratively. It uses the concept of dynamic programming to maintain a value function V that approximates the optimal value function V^*, iteratively improving V until it converges to V^*.

The implementation of the algorithm in this project uses the idea of Q-values, as given by the pseudocode below:

Algorithm 1 Value Iteration Algorithm

```
**Input:** *mdp*, an MDP with states $\mathcal{S}$ , actions $A(s)$, transition model $P(s'|s,a)$,
rewards $R(s)$, discount $\gamma$, $\epsilon$, the maximum error allowed in the utility of any
state
Initialize $U(s) = 0$, for all $s \in \mathcal{S}$
**repeat**
    $\delta \leftarrow 0, U \leftarrow U'$
    **for each** $s \in \mathcal{S}$ **do**
        $U'(s) \leftarrow R(s) + \gamma \max_{a \in \mathcal{A}} \sum_{s' \in S} P(s'|s,a)U(s')$
        $\delta \leftarrow \max(\delta, |U'(s) - U(s)|)$
    **end for**
**until** $\delta < \epsilon(1-\gamma)/\gamma$
**Return** $U$
```

Value iteration converges asymptotically to the optimal policy as iterations continue: $V \to V^*$ as $i \to \infty$, where i is the number of iterations. However, in practice we terminate the algorithm when the largest change in values between iterations δ reaches some pre-determined threshold.

2.1.2 Plot of Optimal Policy

For the given map and $\epsilon = 1e^{-4}$, the algorithm terminates at iteration **1375**. Figure 2 shows the optimum policy discovered by this algorithm.

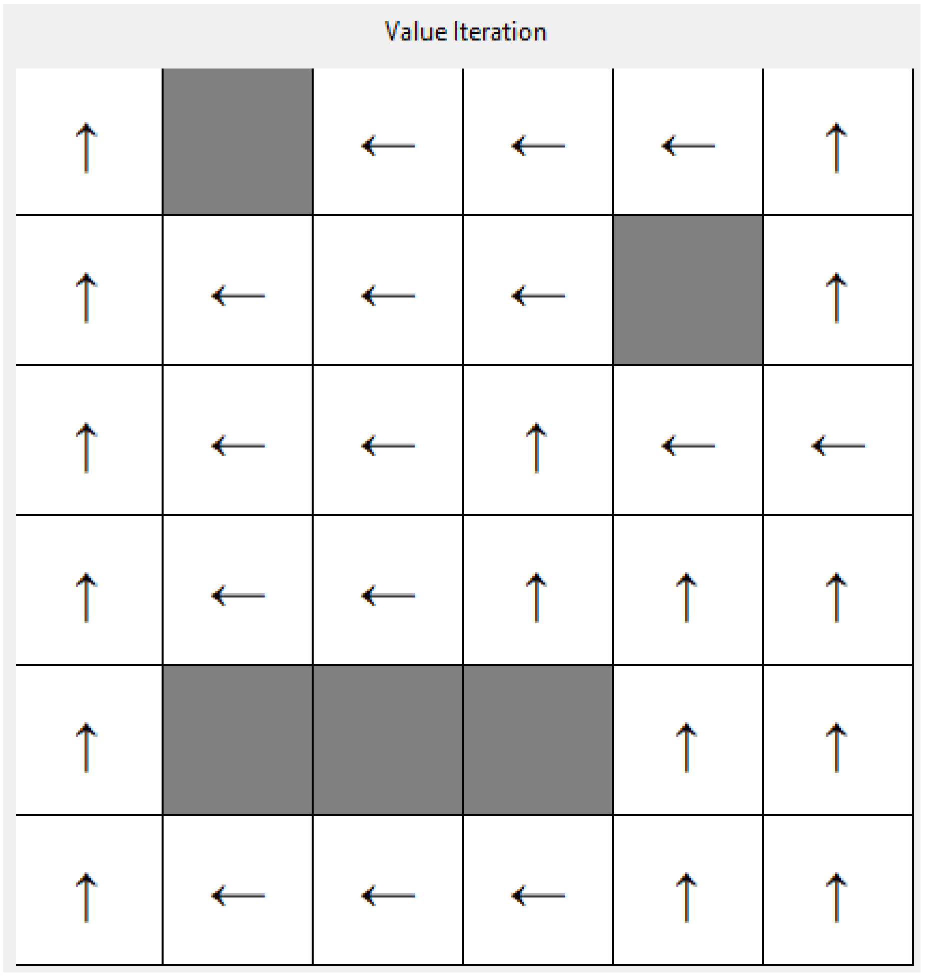
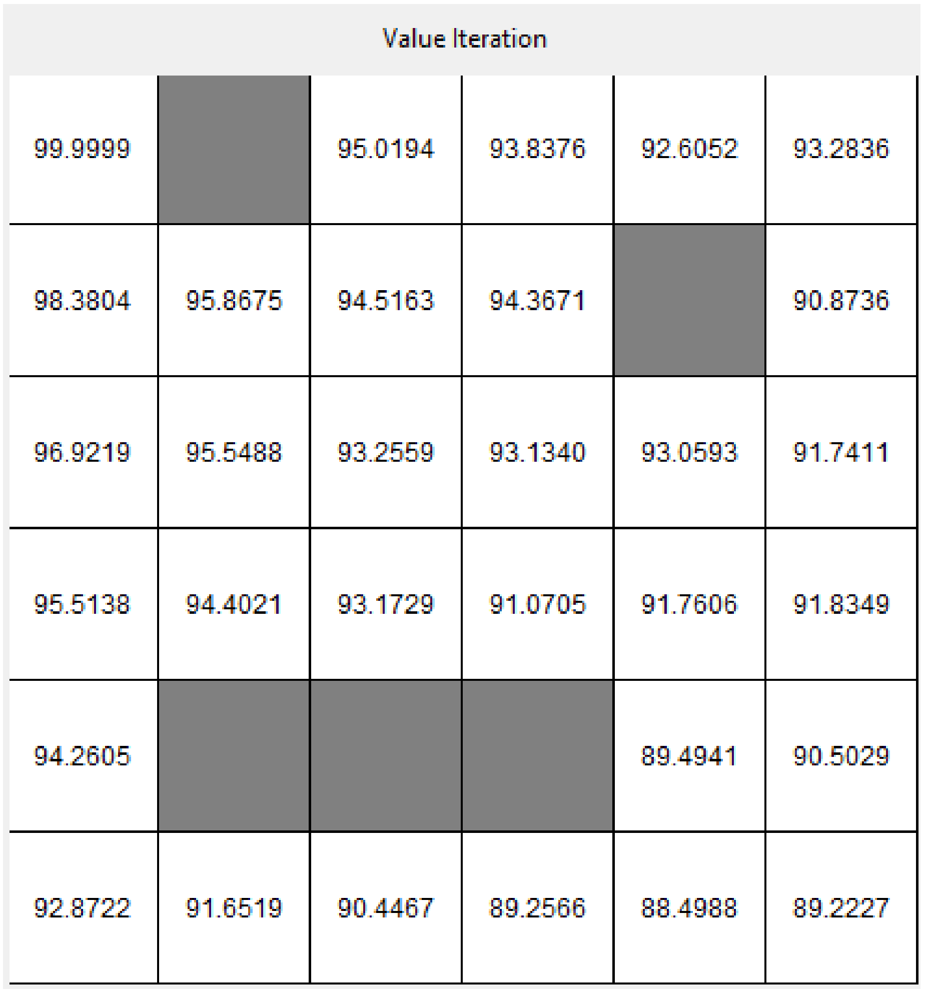


Figure 2: Optimum Policy found by Value Iteration

2.1.3 Utilities of States

Figure 3 shows the utilities of each state $U(S)$ after the algorithm has terminated.


Value Iteration

99.9999	▨	95.0194	93.8376	92.6052	93.2836
98.3804	95.8675	94.5163	94.3671	▨	90.8736
96.9219	95.5488	93.2559	93.1340	93.0593	91.7411
95.5138	94.4021	93.1729	91.0705	91.7606	91.8349
94.2605	▨	▨	▨	89.4941	90.5029
92.8722	91.6519	90.4467	89.2566	88.4988	89.2227

Figure 3: Estimated utilities of each state found by Value Iteration

2.1.4 Plot of estimated utilities

Figure 4 shows the plot of the estimated utilities as a function of the number of iterations. The estimated utilities of each state $U(s)$ approaches the optimum utility $U^*(s)$ asymptotically as the number of iterations approaches infinity.

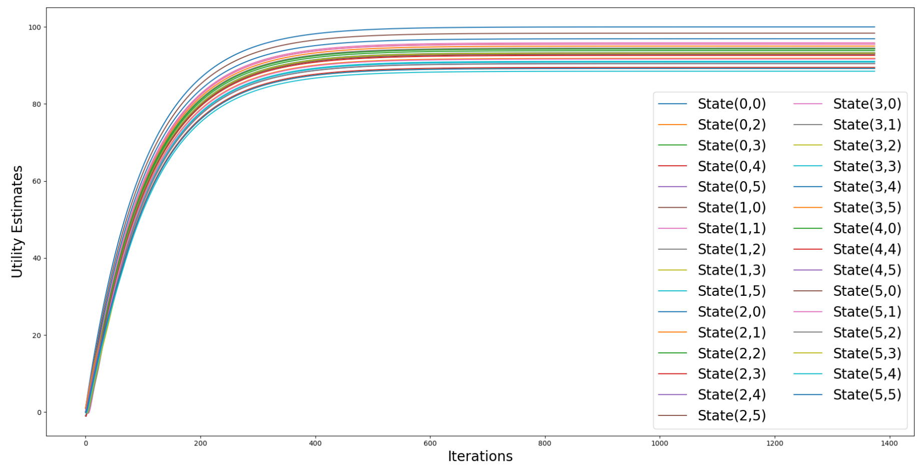


Figure 4: Plot of estimated utilities

2.2 Policy Iteration

2.2.1 Description

While Value Iteration iterates over values, Policy Iteration iterates over policies themselves, creating a strictly improved policy in each iteration (except if the iterated policy is already optimal). Policy Iteration first starts with some (non-optimal) policy, such as a random policy, and then calculates the value of each state of the MDP given that policy — this step is called policy evaluation. At the ith iteration, the utility of a state s (under the policy π_i) is related to its neighbors as such:

$$U_i(s) = R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi_i(s))U_i(s')$$

It then updates the policy itself for every state by calculating the expected reward of each action applicable from that state - this step is called policy evaluation. The implementation of Policy Evaluation used in this project is given by the pseudocode below:

Algorithm 2 Policy Iteration Algorithm

```
Input: mdp, an MDP with states $\mathcal{S}$ , actions $A(s)$, transition model $P(s'|s,a)$,
rewards $R(s)$, discount $\gamma$, $\epsilon$, the maximum error allowed in the utility of any
state
Initialize $U(s) = 0$, for all $s \in \mathcal{S}$
Initialize policy $\pi$ where $\pi(s) =$ Left, for all $s \in \mathcal{S}$
repeat
    $U \leftarrow$ POLICY-EVALUATION($\pi$,$U$,$mdp$)
    unchanged $\leftarrow$ true
    for each $s \in \mathcal{S}$ do
        if $\max_{a \in \mathcal{A}} \sum_{s' \in S} P(s'|s,a)U(s') > \sum_{s' \in S} P(s'|s,\pi(s))U(s')$ then
            $\pi(s) \leftarrow \text{argmax}_{a \in \mathcal{A}} P(s'|s,a)U(s')$
            unchanged $\leftarrow$ false
        end if
    end for
until unchanged
Return $\pi$
```

2.2.2 Plot of Optimal Policy

For the given map, the algorithm terminates at iteration **5**. Figure 5 shows the optimum policy discovered by this algorithm.

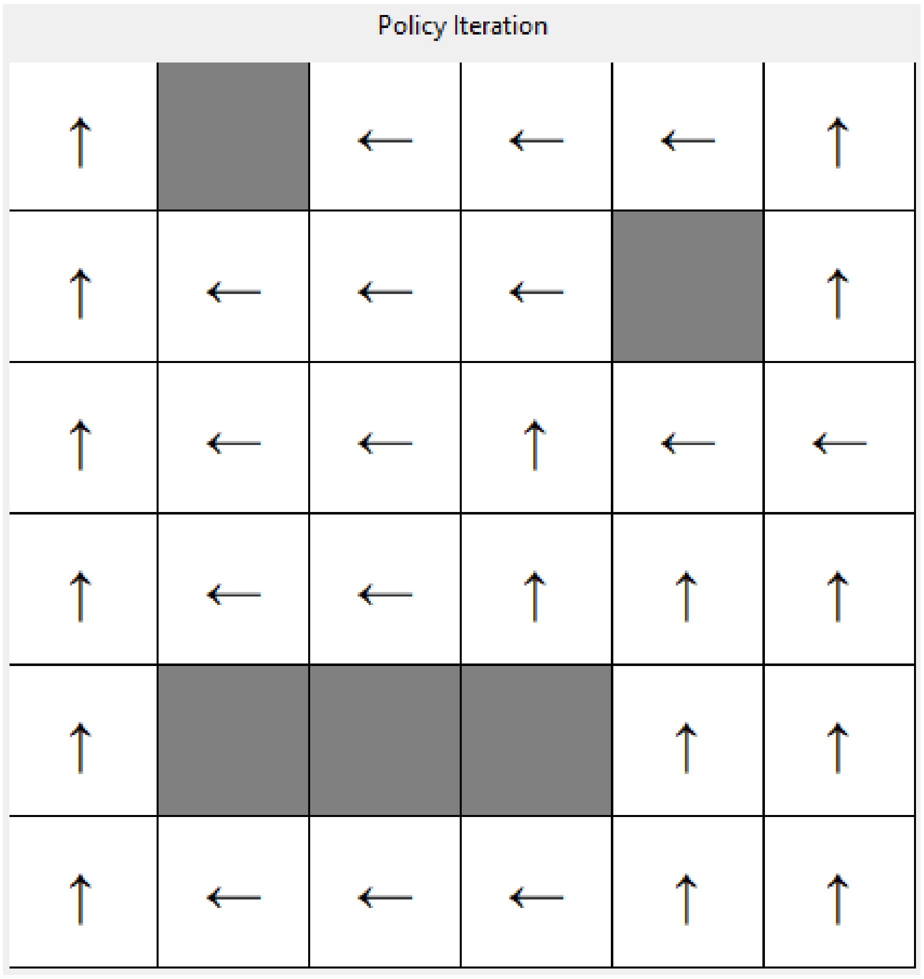
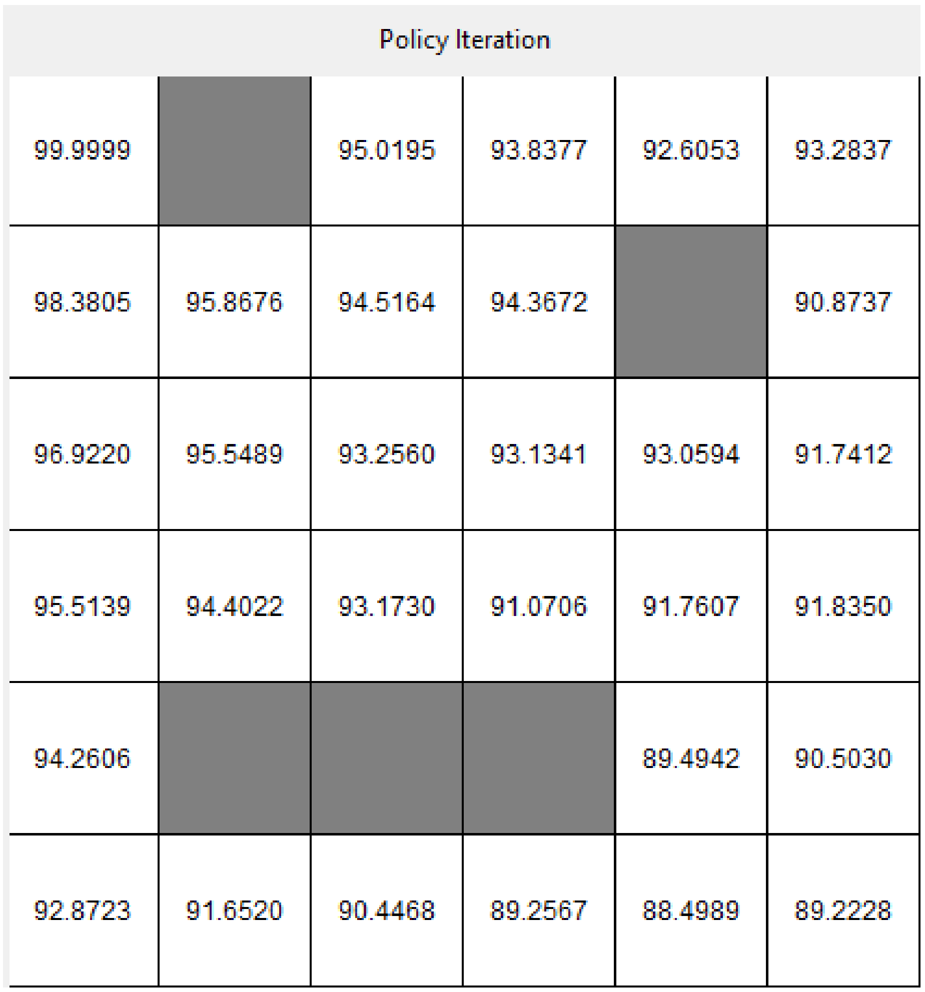


Figure 5: Optimum Policy found by Policy Iteration

The optimum policy found by Policy Iteration is the same as Value Iteration, which is to be expected as there is only one policy that is optimal. However, we note that policy iteration converges after much less iterations, since by fully evaluating the current policy in each iteration, the algorithm can make significant improvements in the policy.

2.2.3 Utilities of States

Figure 6 shows the utilities at each state $U(S)$ after the algorithm has terminated.



Policy Iteration

99.9999	▨	95.0195	93.8377	92.6053	93.2837
98.3805	95.8676	94.5164	94.3672	▨	90.8737
96.9220	95.5489	93.2560	93.1341	93.0594	91.7412
95.5139	94.4022	93.1730	91.0706	91.7607	91.8350
94.2606	▨	▨	▨	89.4942	90.5030
92.8723	91.6520	90.4468	89.2567	88.4989	89.2228

Figure 6: Estimated utilities of each state found by Policy Iteration

The estimated utilities of each state found by Policy Iteration is identical to the estimated utilities of each state found by Value Iteration.

2.2.4 Plot of estimated utilities

Figure 7 shows the plot of the estimated utilities as a function of the number of iterations. The biggest increase in total estimated utilities was made during iteration 3, and the increases in the remaining iterations were small, suggesting the policy was close to optimum from iteration 3 onwards.

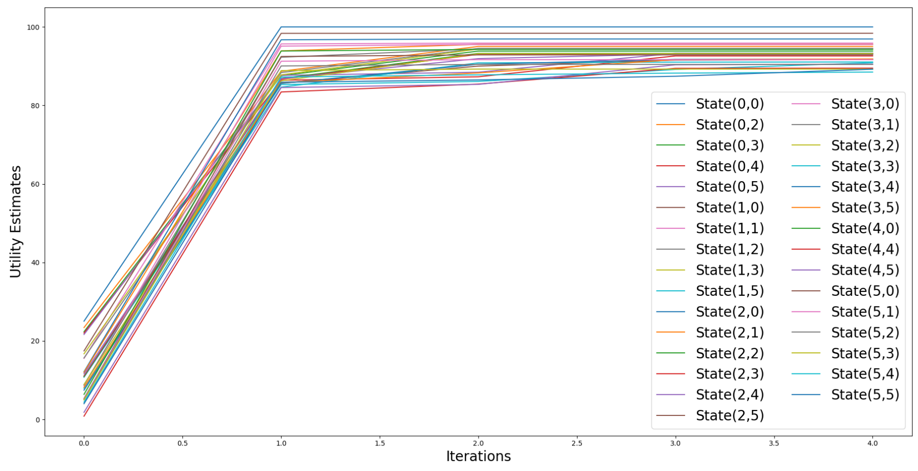


Figure 7: Plot of estimated utilities

2.3 Modified Policy Iteration

The standard policy evaluation involves solving a linear system with n equations and n unknowns using standard linear algebra methods, which has a time complexity of $O(n^3)$. For small state spaces, policy evaluation using exact solution methods is often the most efficient approach. For large state spaces, $O(n^3)$ time might be prohibitive. Therefore, we can perform some number of simplified value iteration steps to give a reasonably good approximation of the utilities. The simplified Bellman update for this process is:

$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s' \in S} P(s'|s, \pi_i(s)) U_i(s')$$

and we repeat this update k times to produce the next utility estimate. This is called modified policy iteration and is often much more efficient than standard policy iteration or value iteration.

2.3.1 Plot of Optimal Policy

For the given map and $k = 50$, Modified Policy Iteration terminates at iteration **5**. Figure 8 shows the optimum policy discovered by this algorithm.

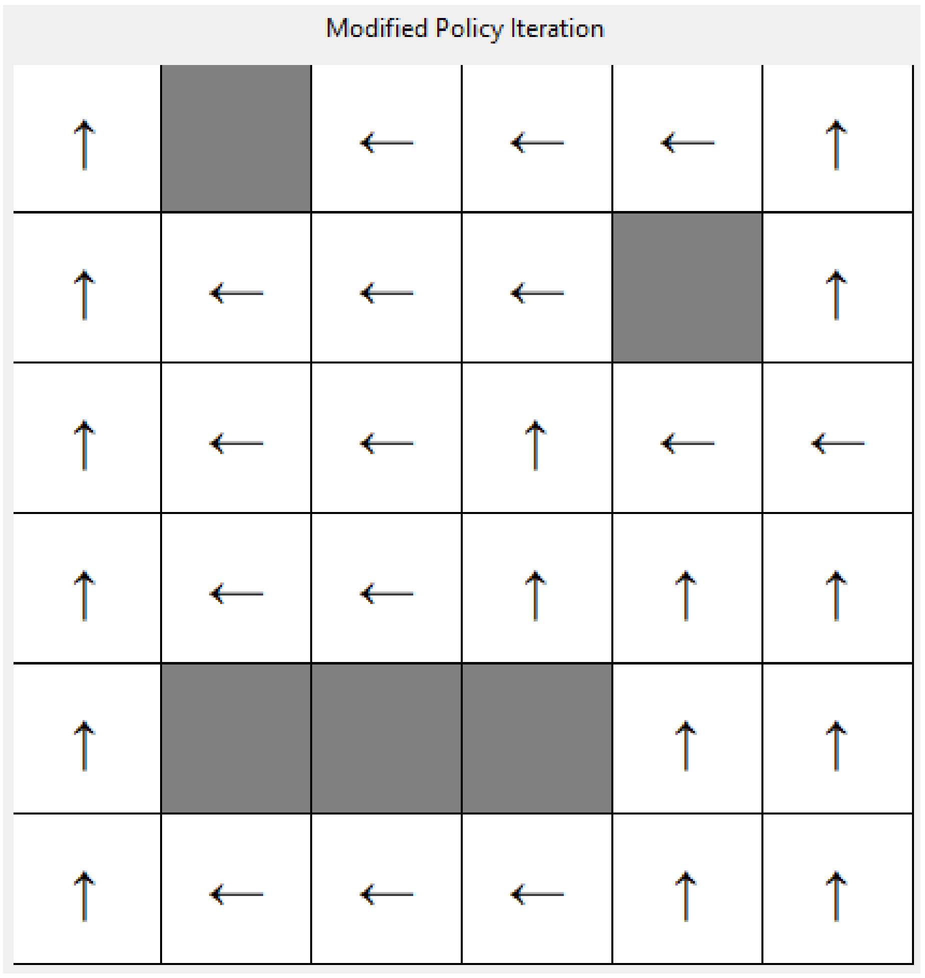
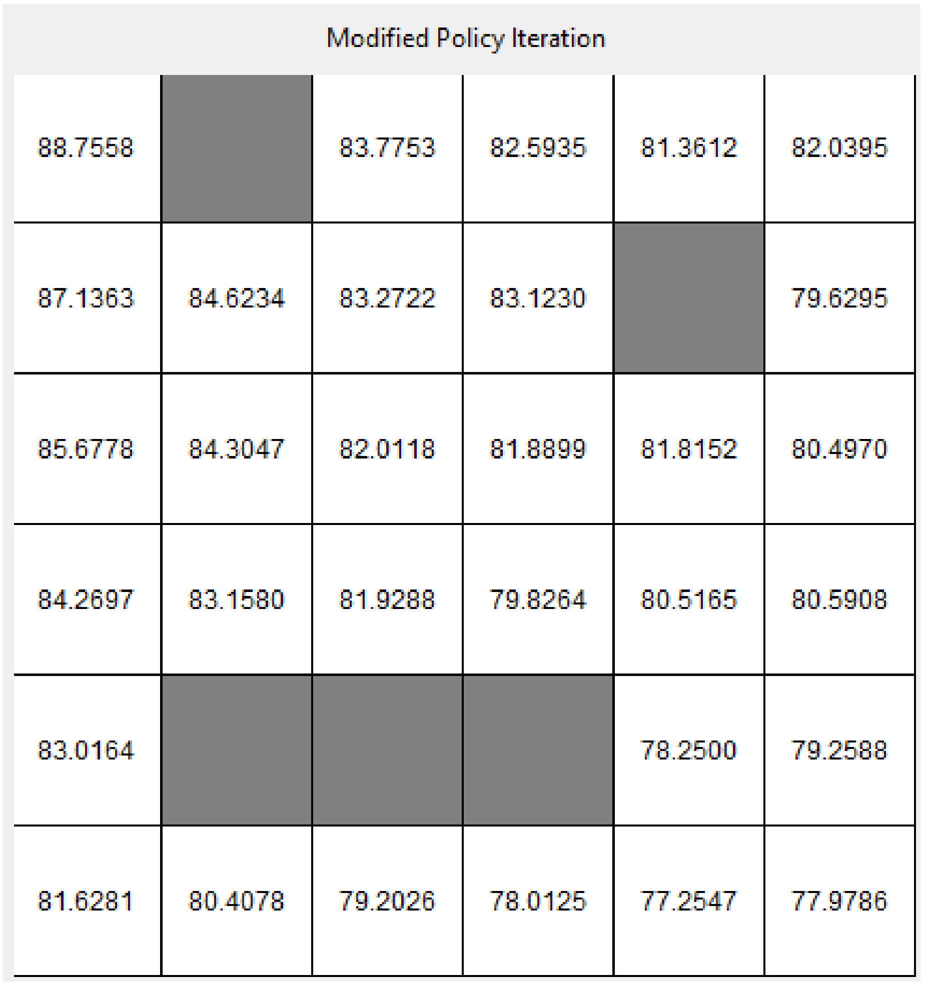


Figure 8: Optimum Policy found by Modified Policy Iteration

The optimum policy found by Modified Policy Iteration is again identical to the optimum policy found by Value Iteration and Policy Iteration.

2.3.2 Utilities of States

Figure 9 shows the utilities at each state $U(s)$ after Modified Policy Iteration has terminated.


Modified Policy Iteration

88.7558	▨	83.7753	82.5935	81.3612	82.0395
87.1363	84.6234	83.2722	83.1230	▨	79.6295
85.6778	84.3047	82.0118	81.8899	81.8152	80.4970
84.2697	83.1580	81.9288	79.8264	80.5165	80.5908
83.0164	▨	▨	▨	78.2500	79.2588
81.6281	80.4078	79.2026	78.0125	77.2547	77.9786

Figure 9: Estimated utilities of each state found by Modified Policy Iteration

The estimated utilities of each state found by Modified Policy Iteration is smaller than the estimated utilities found by both Value Iteration and Policy Iteration, as the policy update in each iteration is based on an estimate of the value function.

2.3.3 Plot of estimated utilities

Figure 10 shows the plot of the estimated utilities as a function of the number of iterations. The increase in the average estimated utilities is more gradual than that for Policy Iteration, again due to the approximation done in policy evaluation.

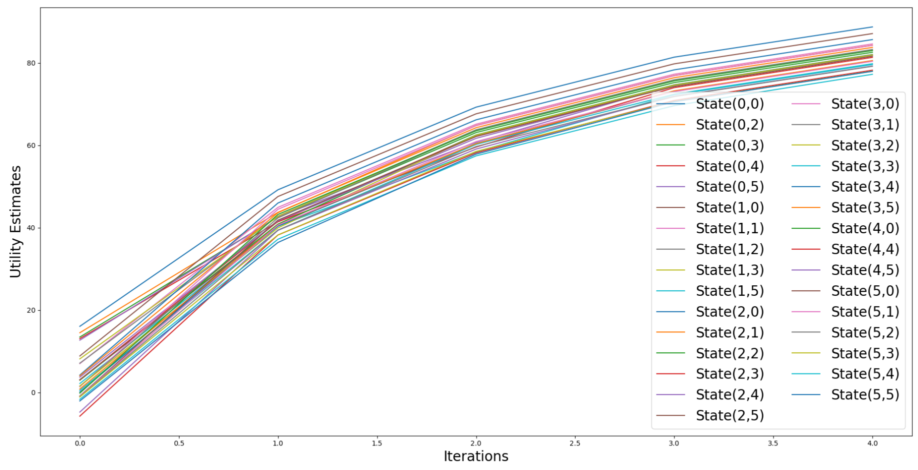


Figure 10: Plot of estimated utilities

3 Analysis of Algorithms

3.1 Time complexity of Value Iteration

First, we analyze the complexity of Value Iteration. On each iteration, we iterate in an outer loop over all states in S, and in each outer loop iteration, we need to iterate over all states ($\sum_{s \in S}$), which means $|S|^2$ iterations. Additionally, we need to calculate the value for every action to find the maximum. Hence, the complexity of each iteration is $O(|S|^2|A|)$. Additionally, the number of iterations are dependent on θ, and has no upper bound. Therefore we can conclude that Value Iteration works best with small-to-medium sized problems (where $|S|$ is manageable), and does not scale that well to larger sized problems.

3.2 Time complexity of Policy Iteration

Next, we analyze the complexity of Policy Iteration. As each state s has at most $|A|$ actions, Policy Iteration is guaranteed to terminate after at most $O(|A|^{|S|})$ iterations, unlike Value Iteration which can theoretically require infinite iterations. However, the time complexity of each iteration is $O(|S|^3 + |S|^2|A|)$, but in most situations Policy Iteration requires only a few iterations to converge. Again, we can conclude that Policy Iteration also works best with small-to-medium sized problems, and fails to scale well to larger sized problems.

3.3 Solving more complex mazes

To observe how Value and Policy Iteration scales with the size of the problems, 6 mazes of varying sizes were randomly generated using the `numpy.random.choices` function. Figure 11 shows the 6 maps that were generated.

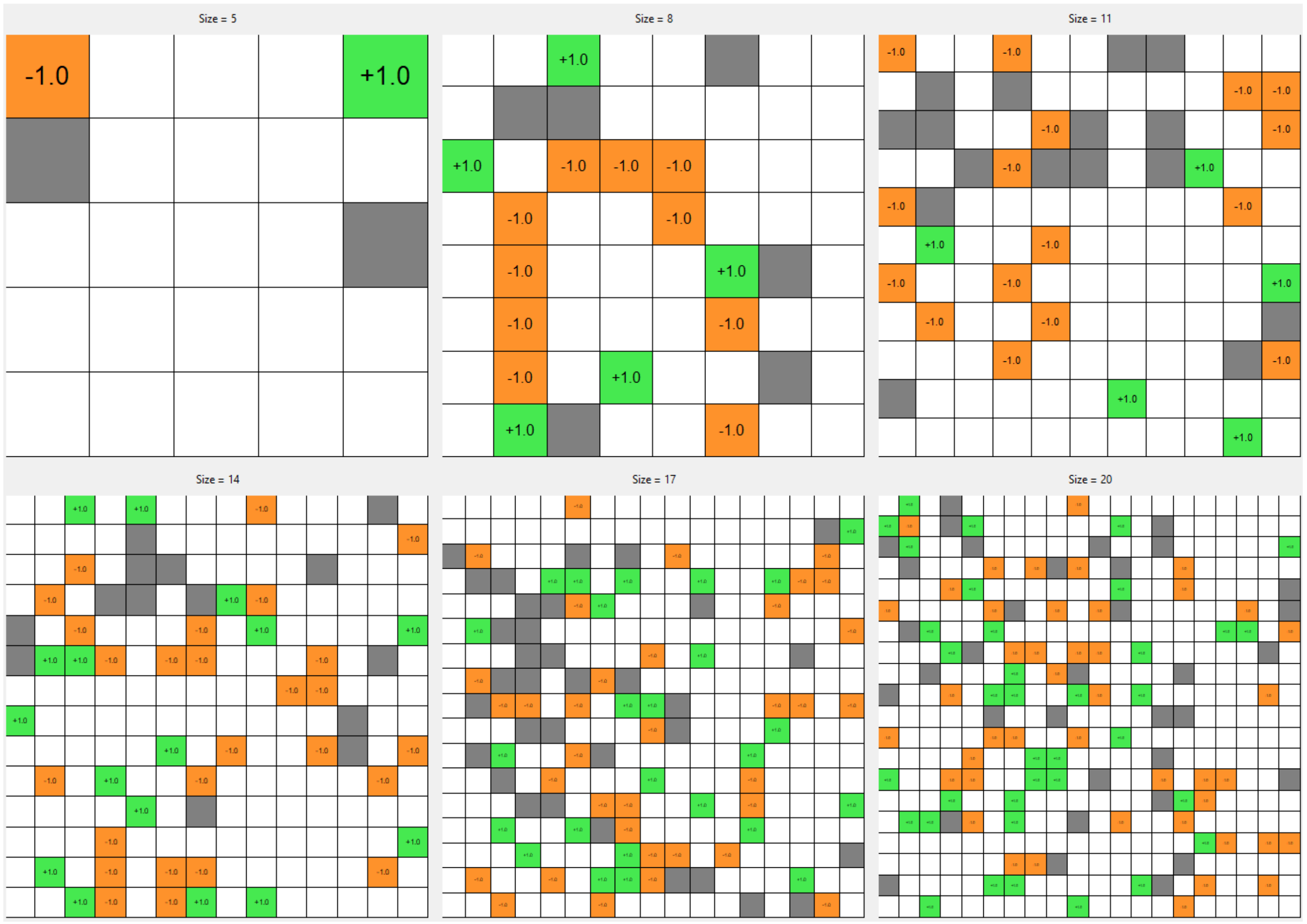


Figure 11: Mazes with sizes ranging from 5x5 to 20x20

Value Iteration, Policy Iteration and Modified Policy Iteration were then ran on the mazes. Figure 12 shows the time taken by the algorithms to find the optimum solution.

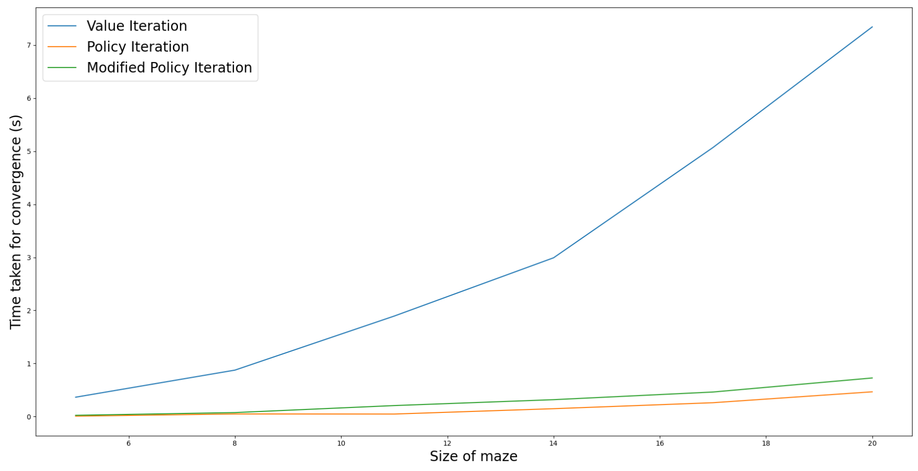


Figure 12: Time taken by the algorithms to find optimum solution

The time taken by Value Iteration to find the optimum solution scales exponentially as the size of the maze increases, while Policy Iteration and Modified Policy Iteration appear to scale better with the size of maze.

Figure 13 shows the number of iterations taken by each algorithm to find the optimum solution. The graph presents the log values to improve readability of the chart.

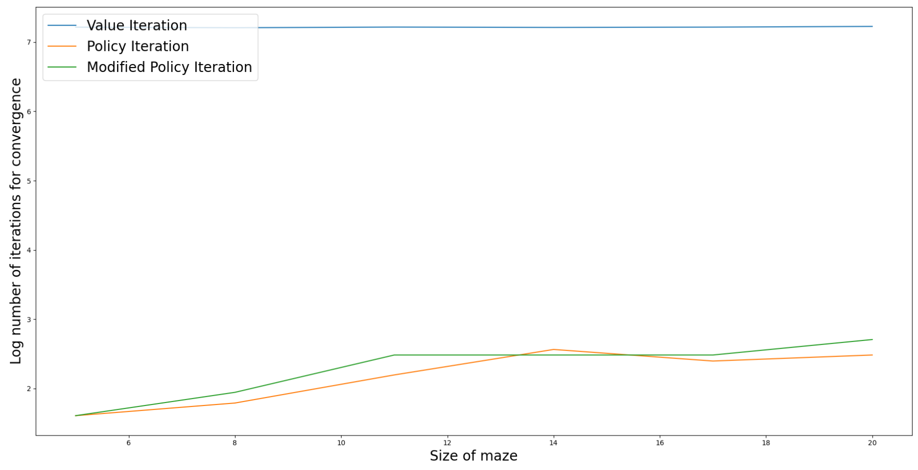


Figure 13: Number of iterations taken by the algorithms to find optimum solution

The number of iterations taken by each algorithm does not change significantly as the size of the mazes increases.

4 Conclusion

This report covers Value Iteration, Policy Iteration and Modified Policy Iteration, and demonstrate their effectiveness in solving MDPs. Unfortunately, these classical algorithms scale poorly as the size of the environment increases, making them unsuitable for larger scale problems.