SC4003 Intelligent Agents Assignment 1

Zhang Xinyang

23 March 2025

Part 1

1. Value Iteration

1.1 Description of implemented solution

Environment

The environment is created by first defining a 6×6 grid (even though the comment suggests a 5×5 grid, the array dimensions are 6×6) and setting the agent's starting position to (3, 2). The grid is stored in the form of a 2-dimensional NumPy ndarray.

Every cell in the grid is initially assigned the default negative reward of -0.05. Specific cells are then updated with different rewards: cells at positions (0, 0), (0, 2), and (0, 5) receive a positive reward of +1; cells at (1, 1), (1, 5), (2, 2), and (3, 3) receive a negative reward of -1; and additional positive rewards are placed at (1, 3), (2, 4), (3, 5), and (4, 4). Next, a list of wall positions is defined, which represent impassable states that the agent cannot enter, including positions like (0, 1), (1, 4), and (4, 1), among others.

The environment also defines four basic actions corresponding to moving up, down, left, or right, with each action represented by its respective vector (for instance, 'U' moves the agent one cell up). A probabilistic transition model is established so that when an action is chosen, there is an 80% chance of moving in the intended direction and a 10% chance each of moving in one of the two perpendicular directions.

Value Iteration

The value iteration function implements backward induction to compute the optimal utility function for each state in a grid world.

Initially, it creates a utility grid V with all values set to zero and then iteratively updates each cell's utility until convergence is achieved (i.e., when the maximum change across all states falls below a specified tolerance). For every iteration, the algorithm loops through each cell in the grid. If a cell is designated as a wall, it is skipped. For non-wall cells, the algorithm considers all possible actions (up, down, left, right) and, for each action, calculates the expected value by summing the immediate reward in that cell with the discounted utility of the next state, considering the probability distribution of possible movement outcomes as given by the transition model, as shown in the following equation.

$$V_{i+1}(s) = \max_a \left\{ \sum_{s'} P(s'|s,a)[R(s) + \gamma V_i(s')] \right\}$$
$$= R(s) + \max_a \left\{ \sum_{s'} P(s'|s,a)\gamma V_i(s') \right\}$$

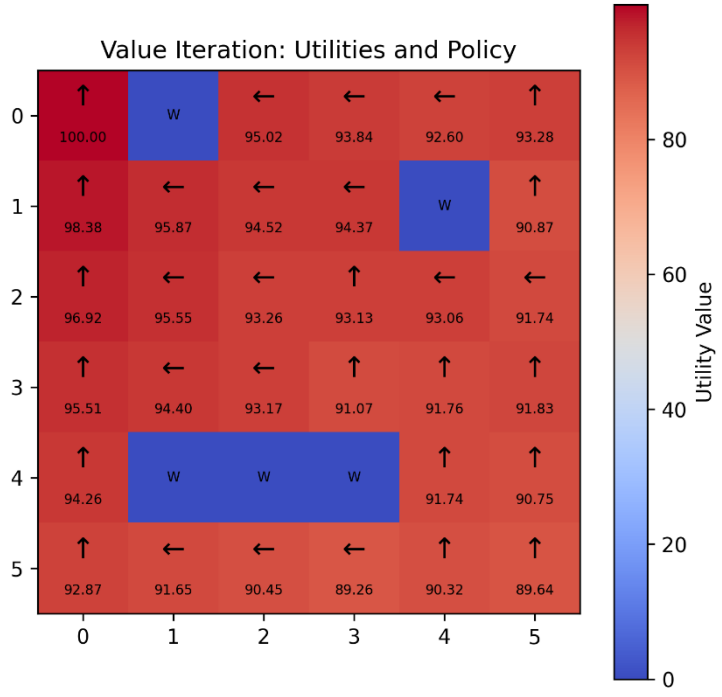
At iteration $i+1$, the utility $V_{i+1}(s)$ for each state s is determined by selecting the utility of an action a that maximizes the expected utility. The expected return for a given action is computed by summing, over all possible successor states s', the immediate reward plus the discounted utility of s' using utility values from the previous iteration i, multiplied by the probability of transitioning to s' from s.

If an intended move would lead to an out-of-bound cell or a wall, the model assumes the agent remains in the current cell. The maximum expected value across all actions is then chosen as the new utility for that cell.

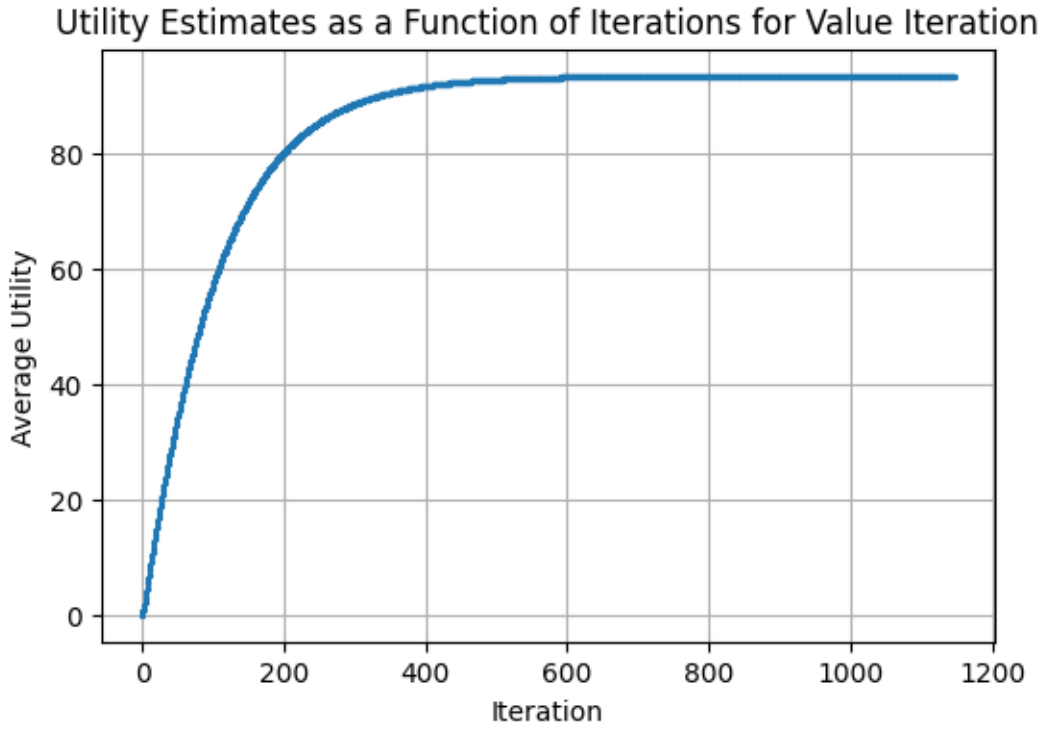
Along with updating the utilities, the function tracks the maximum change (delta) in utilities per iteration. The iterative process terminates once the maximum change is below the tolerance threshold, which is defined as 10^{-5}.

The policy can then be extracted by choosing, for each state, the action that maximizes the expected return, based on the transition probabilities and the final utility values.

1.2 Plot of optimal policy and utilities of each state



1.3 Plot of utility estimates as a function of the number of iterations



2. Policy Iteration

2.1 Description of implemented solution

Environment
See 1.1.

Policy Iteration
The policy iteration function alternates between evaluating a policy and improving it to converge on the optimal policy for a grid world. Initially, it assigns a random policy to every non-wall cell, before entering a loop consisting of 2 main steps. For this assignment, all states are assigned an initially policy of facing right.

First, in the policy evaluation step, it repeatedly updates the utility of each state under the current policy until the values converge below a certain tolerance. This evaluation uses the Bellman equation for policy evaluation, summing the immediate reward in each cell with the discounted utility of the next state, weighted by the probabilities in the transition model, as shown in the following equation.

$$V_{i+1}(s) = \sum_{s'} P(s'|s, \pi_i(s))[R(s) + \gamma V_i(s')]$$

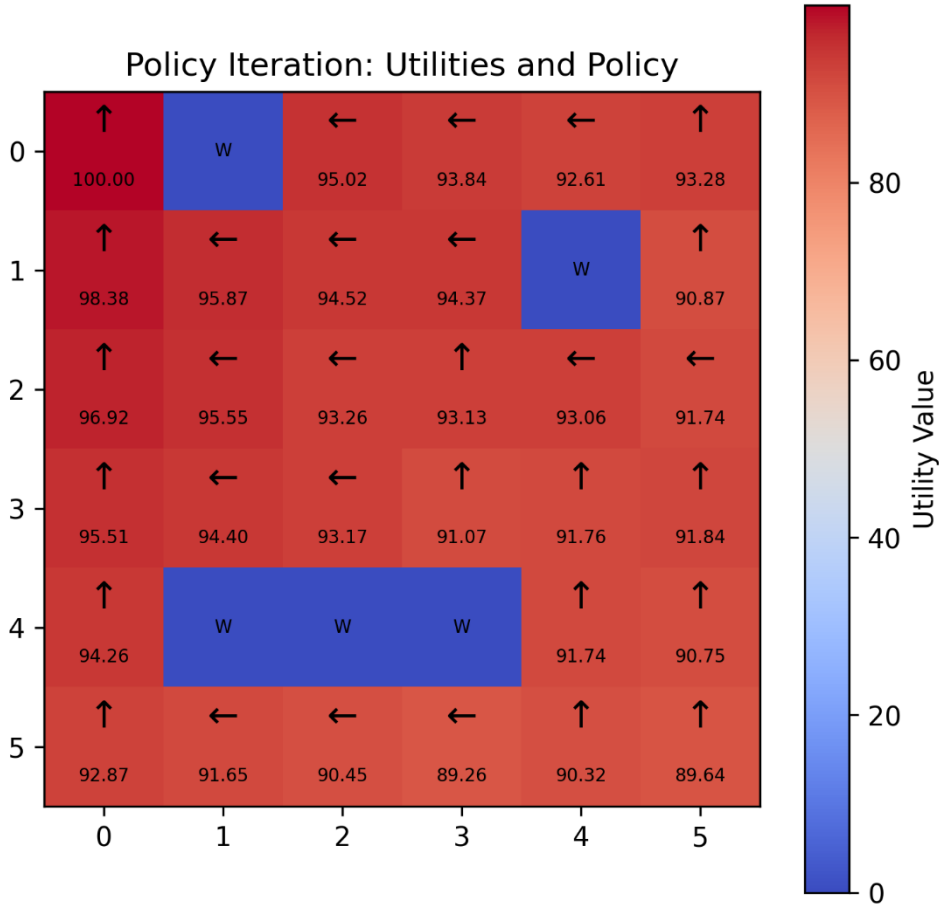
If an intended move would lead to an out-of-bound cell or a wall, the agent remains in the current cell. Once delta (i.e., the change in value) is sufficiently small, the algorithm moves onto the next step.

In the policy improvement step, the algorithm looks at each state and chooses the action that maximizes the expected return based on the newly evaluated utilities.

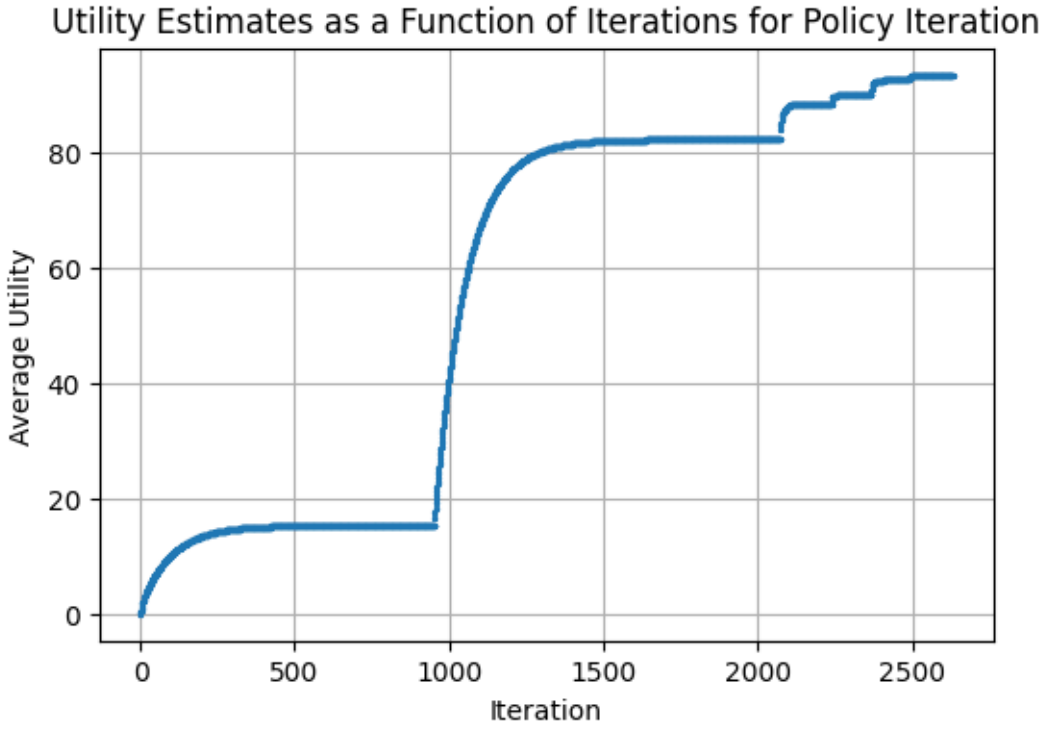
$$\pi_{i+1}(s) = \underset{a}{\operatorname{argmax}} \sum_{s'} P(s'|s, a) V^{\pi_i}(s')$$

If no changes occur to the policy during improvement, the algorithm has converged, indicating that the current policy is optimal. Otherwise, the algorithm repeats the evaluation and improvement steps until the policy stabilizes.

2.2 Plot of optimal policy and utilities of each state



2.3 Plot of utility estimates as a function of the number of iterations



Part 2

Environment

For this section, an updated random environment generator was created.

It takes in grid size, complexity and feature ratios as parameters to generate a new environment to test the algorithm. Grid size refers to the length of the grid environment, while complexity refers to the ratio of features to the size of the environment. A higher complexity indicates more walls and reward and penalty squares. In this experiment, feature ratios were implemented but not tested. This parameter would have controlled the number of walls, rewards and penalties generated relative to each other.

Experiment

The experiment was conducted on grid sizes of 4, 6, 8, 10, 12, 14, 16 and 18, as well as complexities of 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0. For complexity of 1.0, 1 square was left out for the starting square. Each combination of parameters was run 10 times with a different environment configuration. This was except for complexity 0.0 as there is only 1 possible configuration for that parameter (i.e., an empty board). In total, 808 different runs were conducted.

Results

1. How does the number of states affect convergence?

   The following graphs show how the number of states affect convergence. As grid size increases, there exists more individual boxes within the grids. Hence the number of states increases. Therefore, for this question, grid size increases will be synonymous with increases in number of states.

   For value iteration, all runs shared converged after 1100 to 1150 iterations, with the notable exception of when no features are present as seen in Figure 1. Since 0 complexity can be considered an edge case of limited significance, they were excluded in Figure 2. From Figure 2, we can observe that there was a very limited upward trend in the number of iterations taken to converge as the number of states increased. Furthermore, considering that the number of states increases quadratically relative to grid length in the graph, it is likely that the number of states does not significantly affect the rate at which the algorithm converges.

   Without further testing on significantly larger environments, it is hard to draw a persuasive conclusion on the effect of number of states on convergence.

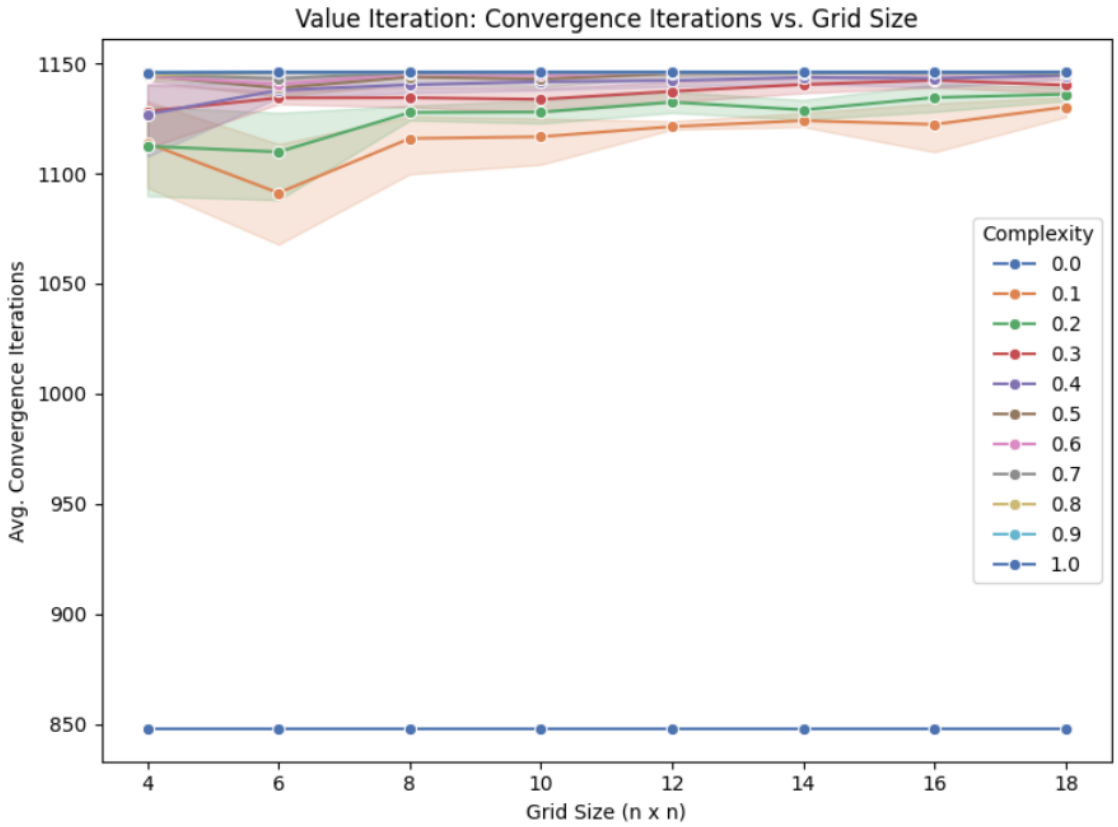


Figure 1: Average Iterations to Converge against Grid Size for each complexity using Value Iteration Algorithm

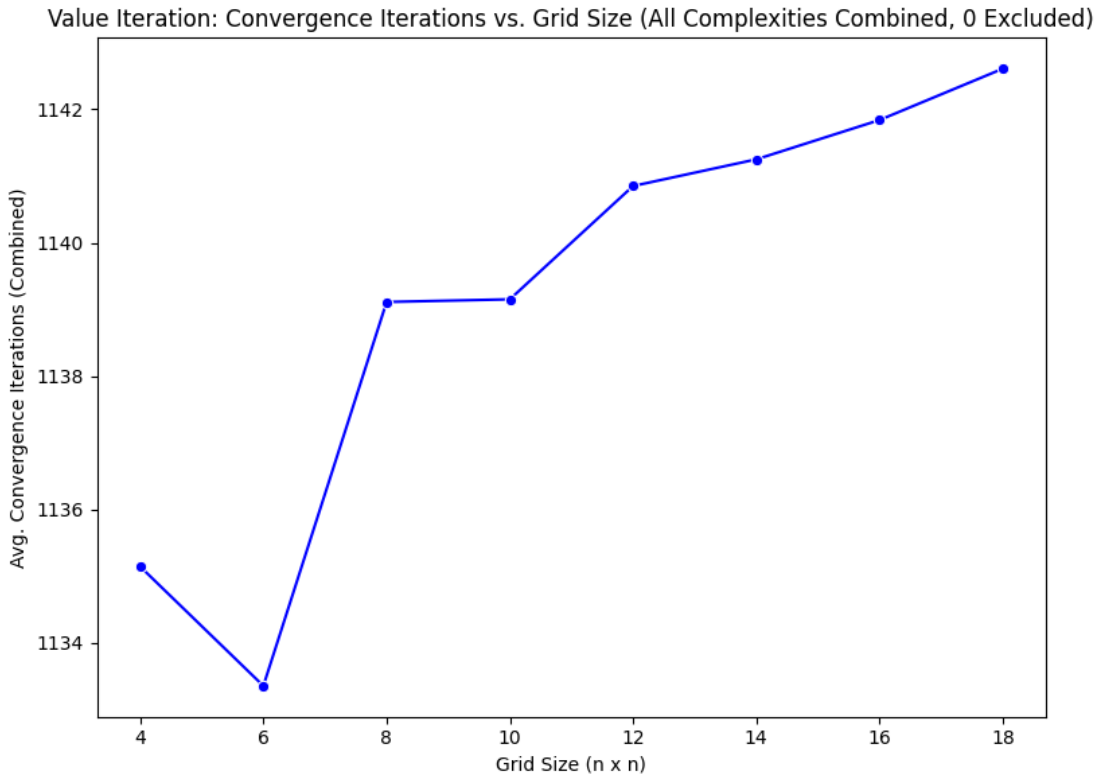


Figure 2: Average Iterations to Converge against Grid Size (excluding 0.0 complexity) for Policy Iteration Algorithm

For policy iteration, runs demonstrated a consistent increase in iterations to converge as the number of states increased, with the notable exception of when no features are present, staying at 2 iterations to converge as seen in Figure 3. Since 0 complexity can be considered an edge case of limited significance, they were excluded in Figure 4. From Figure 4, we can observe that a positive relationship in the number of iterations taken to converge as the number of states increased. It is unclear whether the relationship is linear or logarithmic. An exact mathematical representation of the relationship requires more data and experiments to derive.

Note that the number of iterations here refers to the number of policy improvements made (i.e., number of times step 2 is executed), rather than the number of utility updates performed in step 1. If the utility update threshold in step 1 was to be replaced with a constant update every k steps, then the number of policy updates may be more constant and similar to the observation for value iteration.

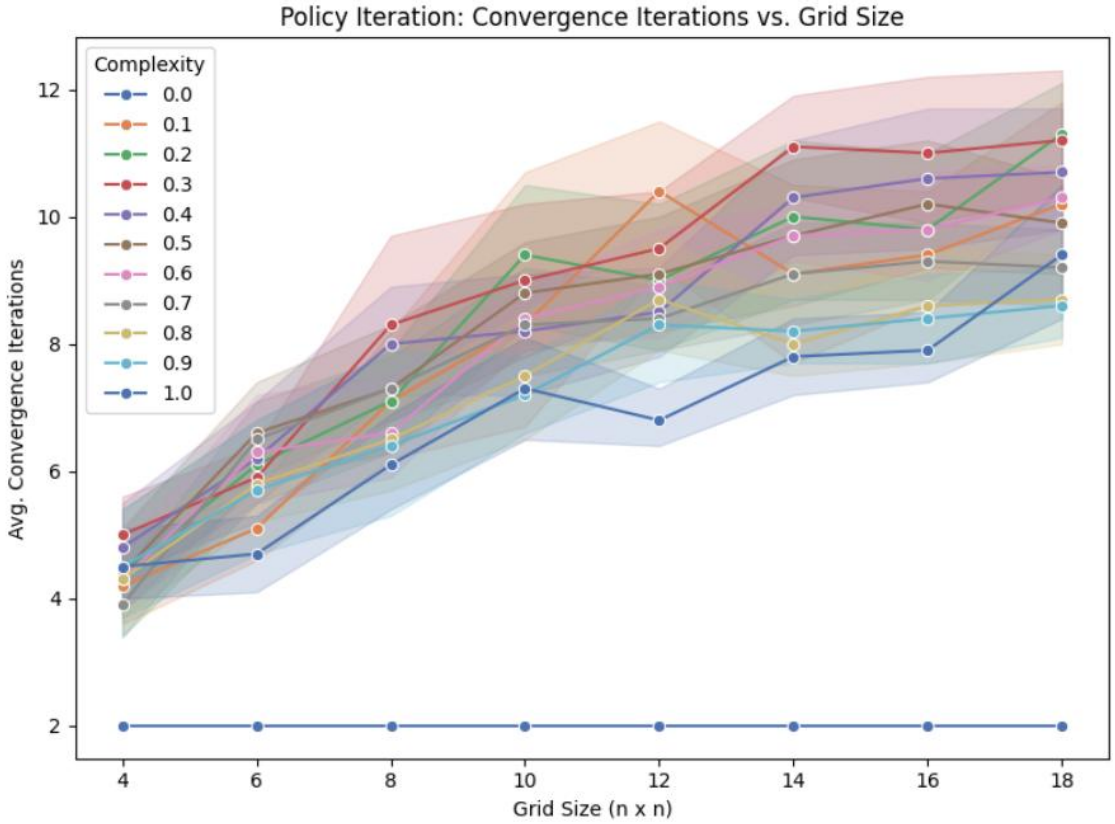


Figure 3: Average Iterations to Converge against Grid Size for each complexity using Policy Iteration Algorithm

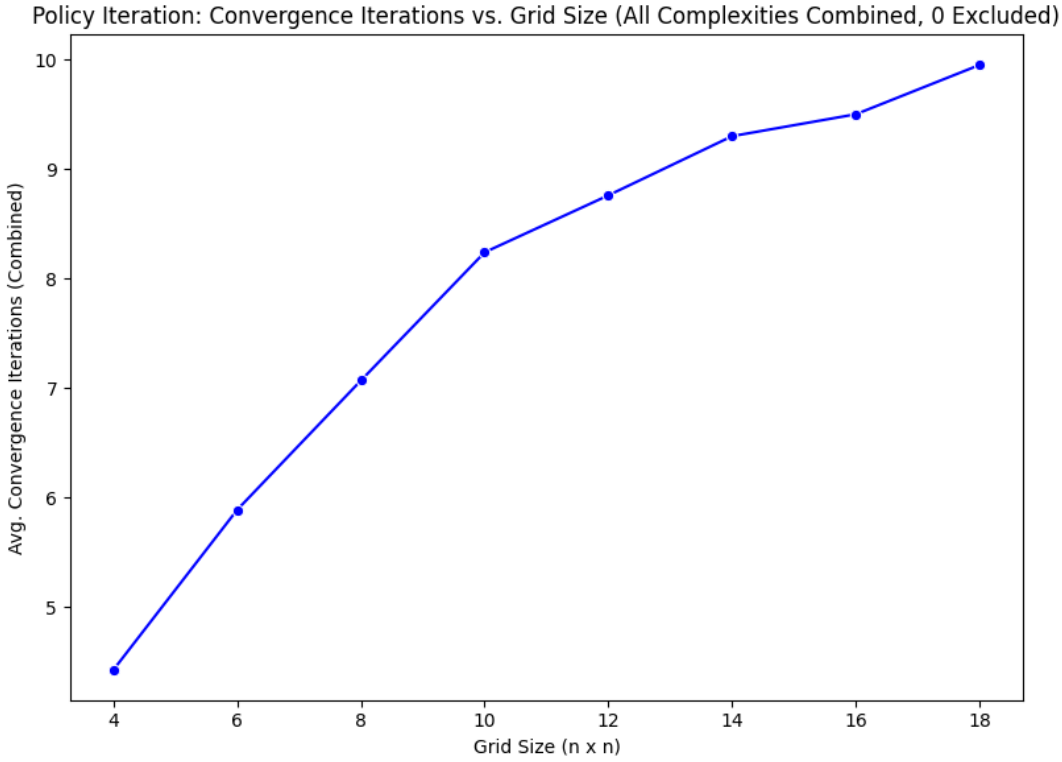


Figure 4: Average Iterations to Converge against Grid Size (excluding 0.0 complexity) for Policy Iteration Algorithm

2. How does the complexity of the environment affect convergence?

   For value iteration, value iteration requires relatively few iterations to converge when the environment is completely empty of features (complexity = 0). Once features are introduced (complexity > 0), the number of iterations needed to converge jumps noticeably. After this initial jump, further increases in complexity (from 0.1 up to 1.0) do not substantially change the number of iterations—most lines flatten out near the same upper bound.

   In other words, value iteration is very fast to converge in a trivial environment with no special features, but as soon as there are multiple walls or reward cells, the algorithm needs more iterations to distinguish among different states and compute their utilities accurately. Beyond a certain point, additional walls or reward cells do not significantly increase the required iterations, likely because the environment is already sufficiently complex such that value iteration must make many updates to resolve the differences among states.

   To better understand how convergence is affected by complexity, more testing can be performed on environments where 0 < complexity < 0.1.

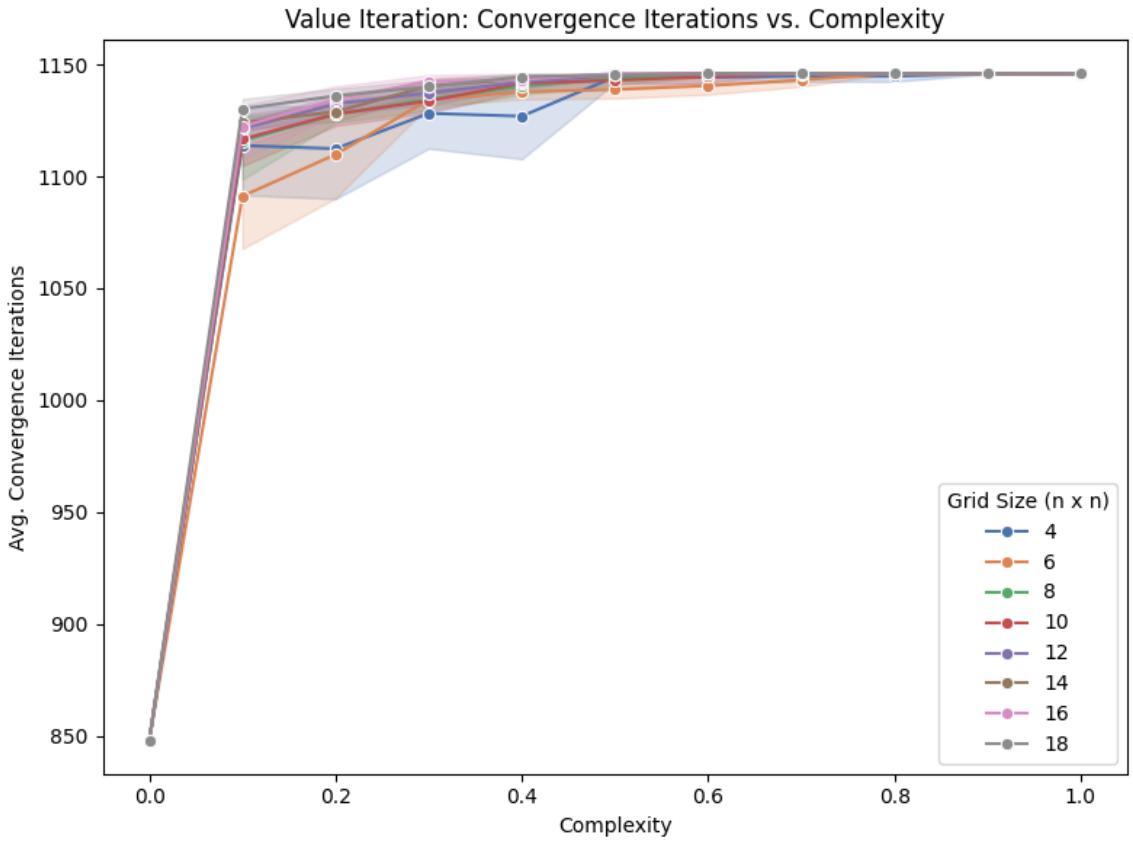


Figure 5: Average Iterations to converge against Complexity for Value Iteration Algorithm

For policy iteration, the algorithm converges in just 2 steps when the environment has no features (complexity = 0). Once features are introduced, the number of policy improvement steps needed to converge jumps noticeably. Beyond that, a slightly negative relationship between complexity and convergence iterations is observed.

This trend is made obvious in Figure 7, when all grid sizes are combined. The average number of policy improvement steps needed increases as complexity increases before peaking at a moderately complex environment (complexity = 0.3). Subsequently, the average number policy improvement steps needed decreases until the environment is full.

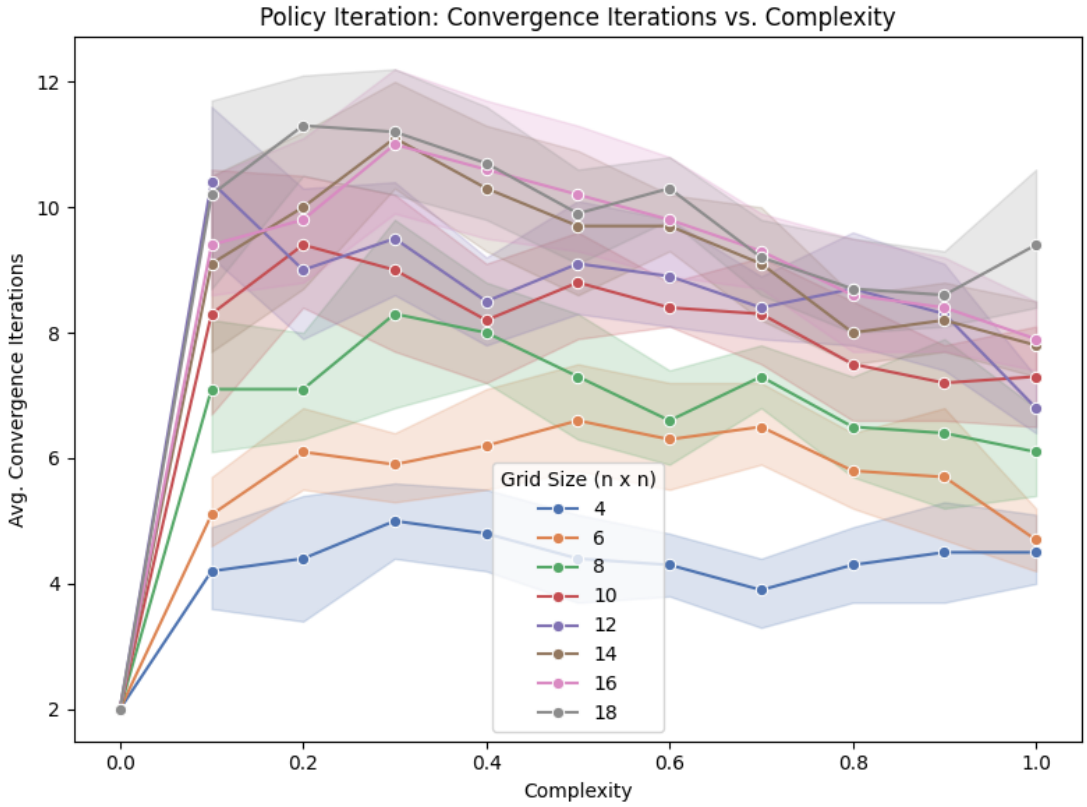


Figure 6: Average Iterations to converge against Complexity for Policy Iteration Algorithm

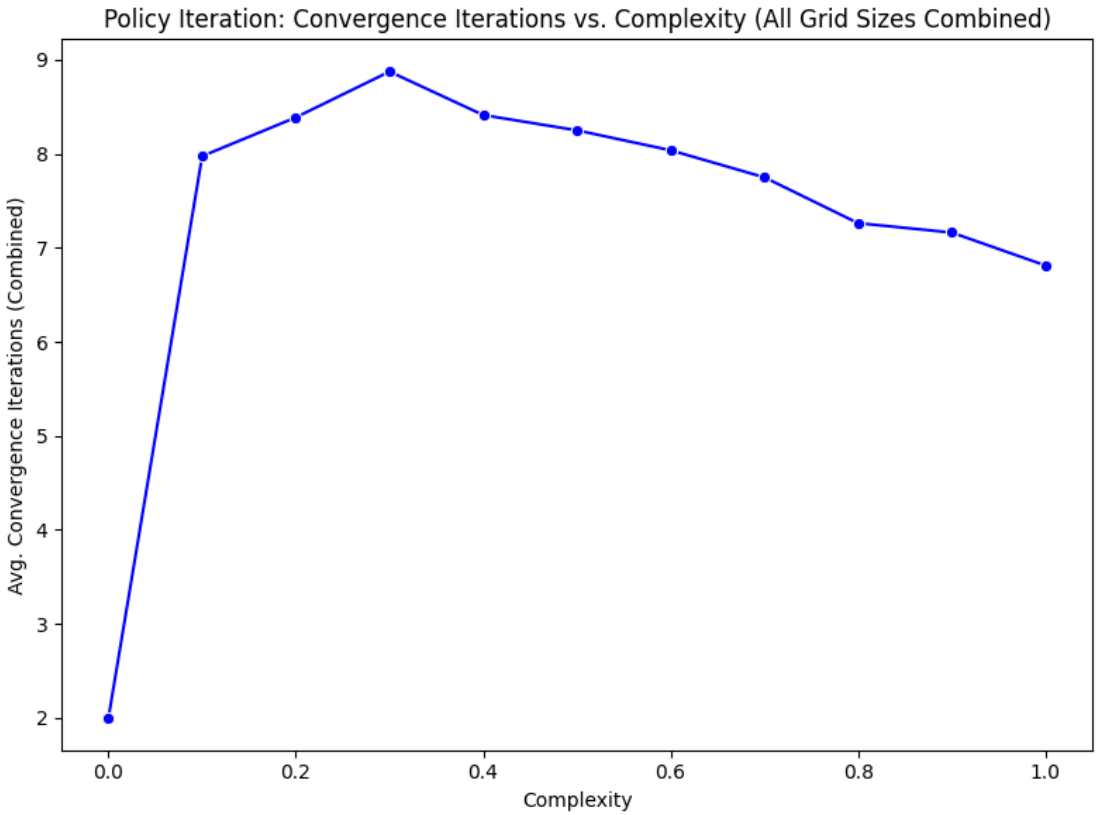


Figure 7: Average Iterations to converge against Complexity for Policy Iteration Algorithm (Combined grid sizes)

In summary, value iteration converges very quickly when the environment is empty of features (complexity = 0) but experiences a sharp increase in the number of iterations required as soon as any features (walls or reward cells) are introduced. After that initial jump (around complexity 0.1), further increases in complexity do not cause a substantial rise in iterations—likely because the environment is already "complex enough" that the algorithm must perform many updates to differentiate among states.

By contrast, policy iteration converges in only two steps in the trivial (zero-complexity) environment. As soon as features appear, the number of policy improvement steps required rises noticeably. Once the environment becomes very cluttered (high complexity), policy iteration tends to require fewer steps again, suggesting that an overly complex environment can sometimes introduce too many constraints that instead serve to stabilize the policy more quickly. Note that however that is the policy is updated irregularly, and utility updates may follow a trend more like value iteration.

Nevertheless, these conclusions have certain limitations.

First, the current definition of complexity is based solely on the total number of features (walls and reward/penalty cells). As illustrated in Figure 7, simply adding more features

does not necessarily yield a proportionately more difficult environment. This indicates that a more nuanced measure of complexity may be required.

Second, the code maintains an approximate 1:1:1 ratio of walls, positive rewards, and negative rewards. Other ratios or distributions of these features could lead to environments that require more iterations to converge or exhibit different convergence patterns altogether.

3. How complex can you make the environment and still be able to learn the right policy?

To answer this question, we will need to define some key terms. We will define complex to refer to both the number of states as well as the number of features.

To define the right policy is a much more nuanced matter. We will rely on heuristics to assess whether a policy meets certain criteria. One such heuristic involves examining whether the policy leads to so-called global pseudo-terminal solutions. Since the environment does not terminate, the agent will then seek to end up in a state that maximises the utility of its subsequent states. However, this is complicated by the presence of the state transition model which adds uncertainty to the next state.

Therefore, a global pseudo-terminal solution must fulfil 2 conditions:

1. The best action(s) of the state(s) in the solution must eliminate the uncertainty regarding the next state.
2. The best action(s) of the state(s) in the solution must maximise utility and reward for all subsequent states indefinitely.

Let's see an example of a global pseudo-terminal state.

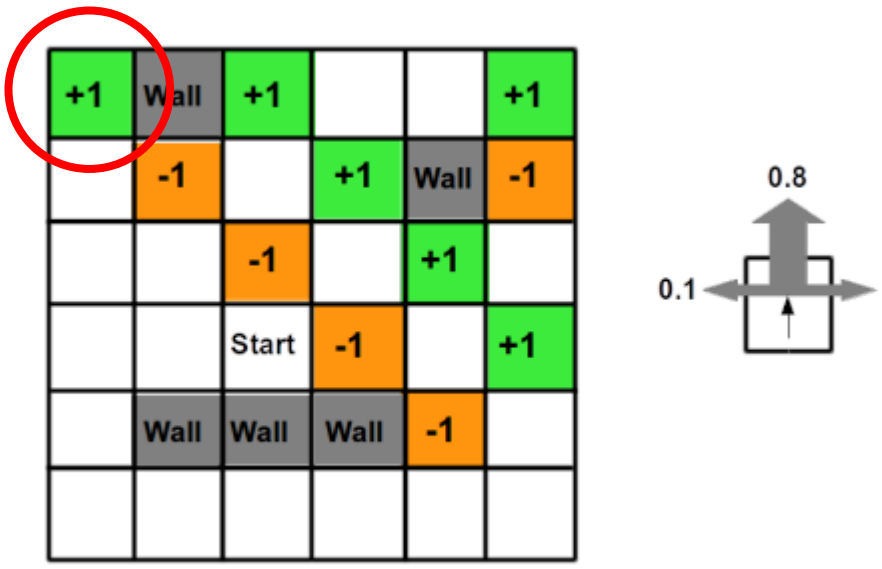


Figure 8: Default Environment given in Part 1 of Assignment

In Figure 8, the top-left corner of the grid is a global pseudo-terminal solution. The best possible action in that solution will be to go upwards in the top-left box/state, with a 10% chance of going right and 10% chance of going left. However, as the left box is the boundary and the right box is a wall, the next state will necessarily be itself. Thus, by going up, the agent will be able to collect the positive reward after every action once

it reaches the state. This must be the best possible solution because the agent is able to collect the maximum possible reward after every action when in the state area or solution.

A global pseudo-terminal solution suggests that a local pseudo-terminal solution exists.

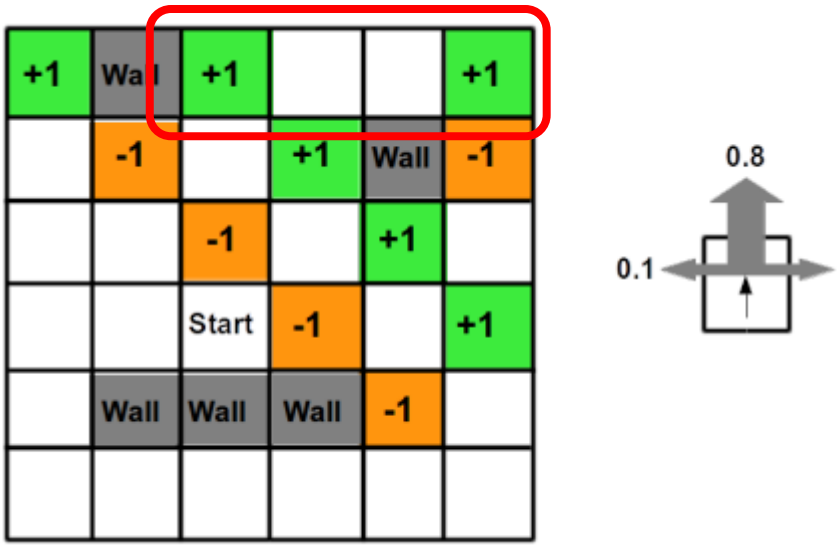


Figure 9: Default Environment given in Part 1 of Assignment

In Figure 9, a policy of Up – Left – Right – Up for the states from left to right could achieve consistent, positive rewards. By moving up in the leftmost and rightmost boxes, the agent has a 90% chance of staying in the reward square, while the middle 2 featureless squares seek to guide the agent back towards the corner reward squares by going left and right. However, this solution is not a global pseudo-terminal solution as:

1. The best action(s) of the state(s) in the solution does not eliminate the uncertainty regarding the next state. In other words, it is possible for the agent to leave the solution area, no matter how low that possibility is.
2. The best action(s) of the state(s) in the solution does not maximise utility and reward for all subsequent states indefinitely. As there are default boxes within the solution area, the agent is not deriving the maximum possible reward by staying in the solution area, unlike the top left corner.

However, since it **reduces** the uncertainty regarding the next state, and achieves an **average positive reward** when staying in the solution area, this solution can be considered a **local pseudo-terminal solution**.

Theoretically, the right policy should guide the agent towards a global pseudo-terminal solution. However, due to limitations in the discount factor, the policy may instead lead to creations of local pseudo-terminal solutions, instead of outputting a policy that guides

an agent towards a global pseudo-terminal solution. Note that an environment may not necessarily have a global pseudo-terminal solution, in which case the local pseudo-terminal solution may be the best solution (i.e., solution that maximises reward) that exists in the environment.

To examine whether the right policy is found, we will look at environments where a global pseudo-terminal solution exists **and** a local pseudo-terminal solution may exist, and whether the overall policy will guide an agent towards the global solution instead of forming a local solution.

At grid size = 6, complexity ≈ 0.44 (the default environment), the right policy was learnt. As observed in Figure 10, the learned policy for the local solution is L – L – L – U, which pushes the agent closer towards the global solution, while maximising the reward in doing so. Hence the right policy was learnt.

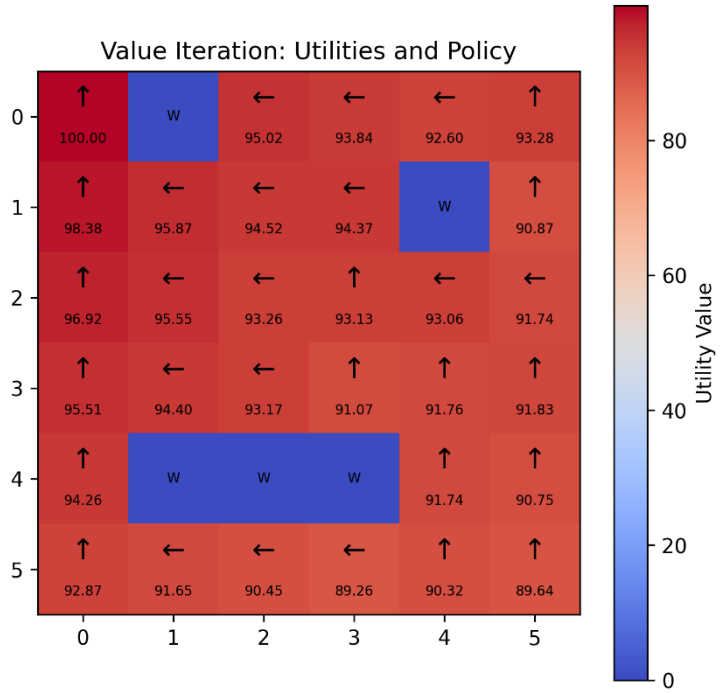


Figure 10: Policy of default environment

At grid size = 8, complexity = 0.5, run = 0, there are 2 global solutions and 1 local solution.

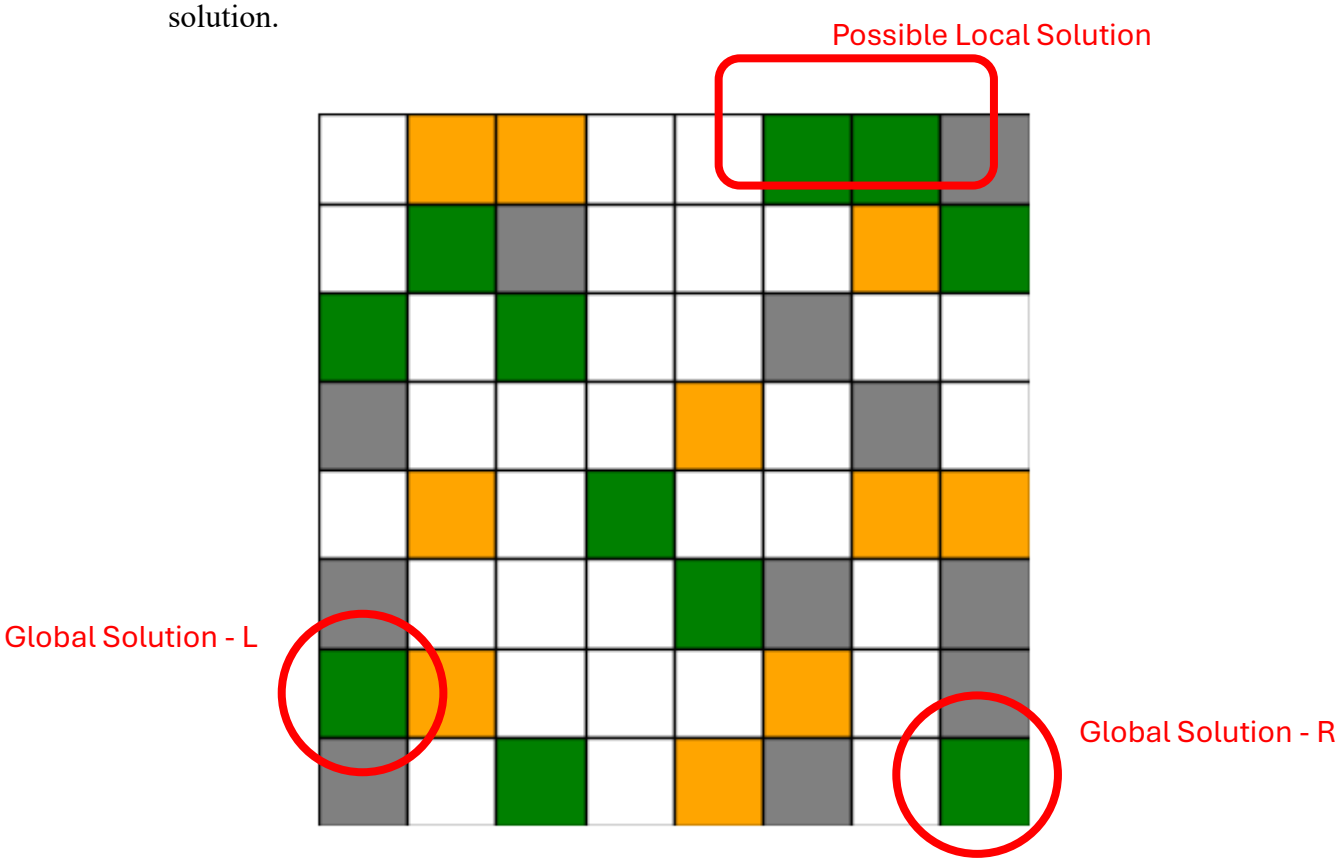


Figure 11: env_grid_grid8_complex0.5_run0.png

The global solution on the left should have policy of L, while the other global solution should have policy of R. These policies will ensure that the agent remains in the same state indefinitely while reaping the maximum positive reward.

A possible local solution may exist in the top right corner, as the agent may be able to remain in the states with positive reward for an extended period of time, but ultimately the expected reward is not maximum over an indefinite period of time due to the possibility of the agent leaving the local solution area.

Hence the right policy in this case will be to direct an agent to either of the global solution.

However, in Figure 12 we can observe that the learned policy is roughly split into 3 areas. First, the policy for the left side of the environment is to direct an agent towards the global solution on the bottom left. Second, the policy for the bottom right side of the environment directs an agent towards the global solution at the bottom right corner. Lastly, the policy for the top right side of the environment is to lead an agent towards the top right corner of the environment. While the utility value for the local solution is

rather high, it is not the solution that gives the maximum possible reward for an indefinite period. Hence, the right policy was not learnt for this environment.

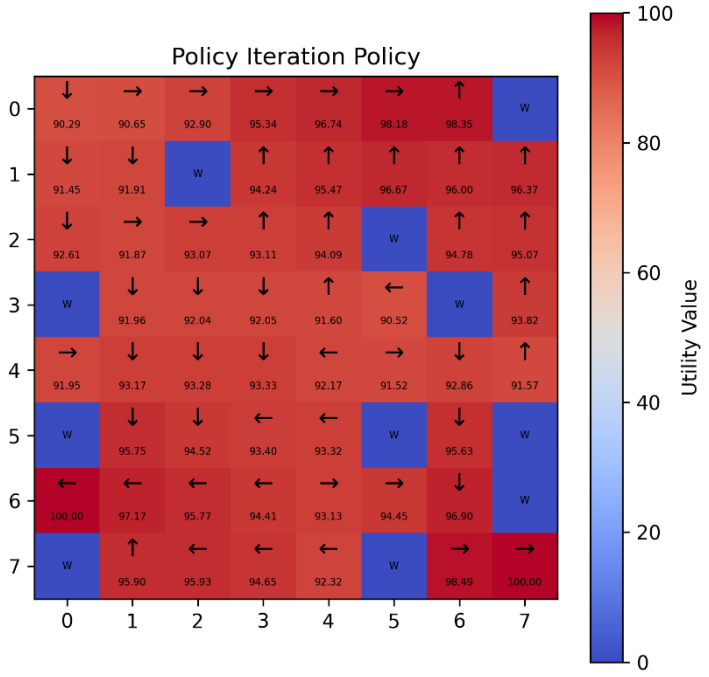


Figure 12: policy_grid_grid8_complex0.5_run0.png

At grid size = 10, complexity = 0.4, run = 9, there are there is 1 global solution and 2 local solutions.

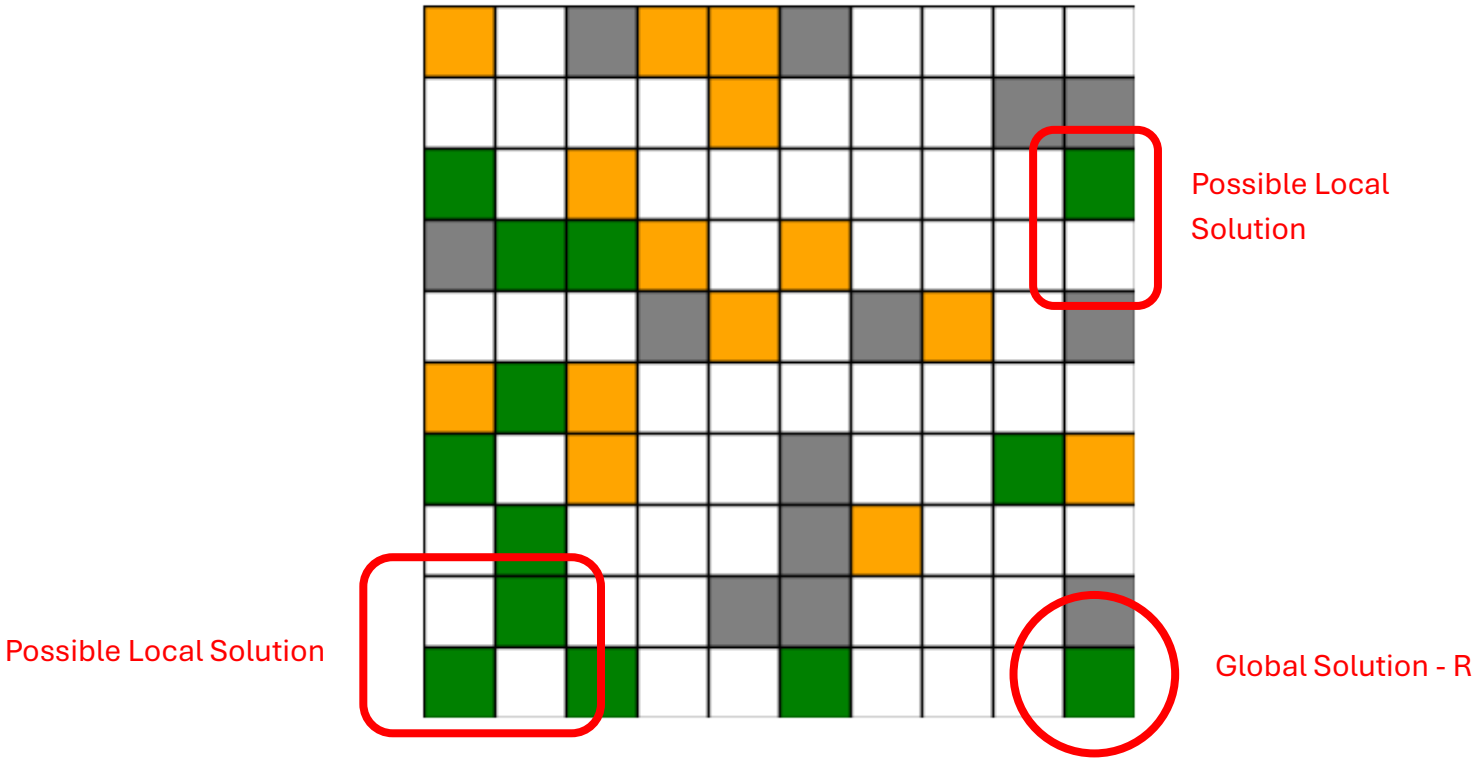


Figure 13: env_grid_grid10_complex0.4_run9.png

The policy for the global solution will be to always choose to go right. The policy to achieve a local solution for the bottom left will be to direct the agent towards the bottom left corner, and to achieve a local solution for the top right will be to direct the agent towards the state with a positive reward.

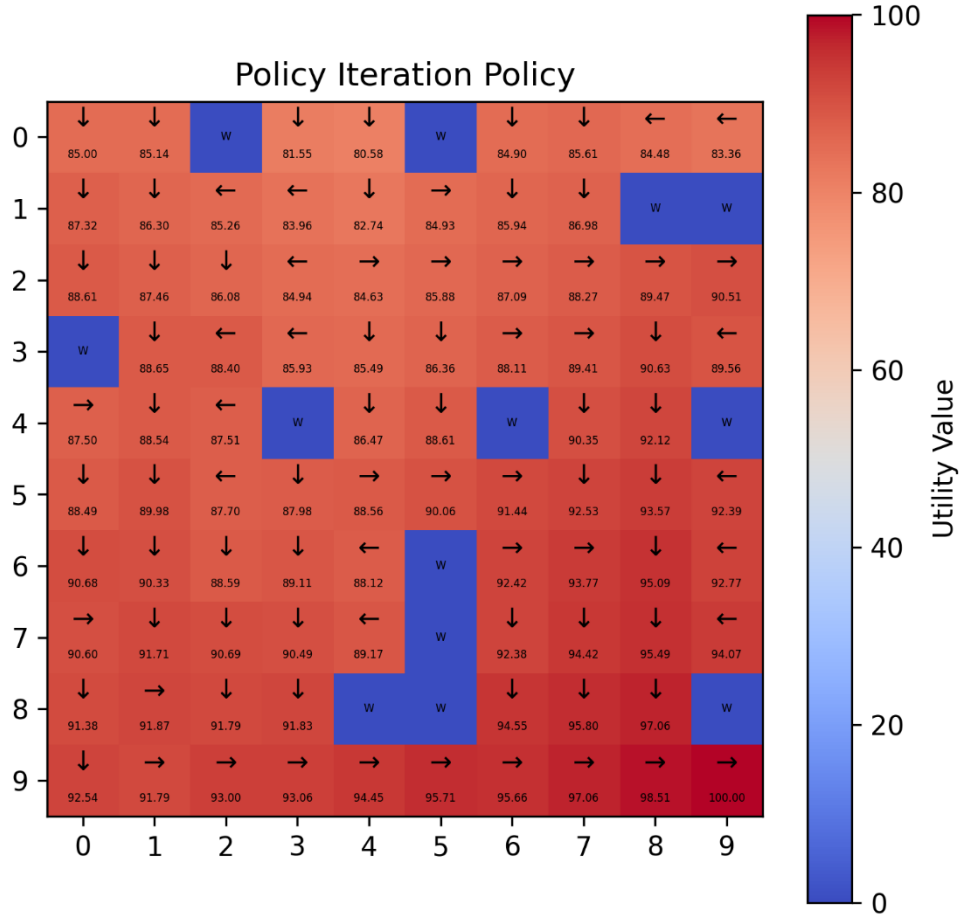


Figure 14: policy_grid_grid10_complex0.4_run9.png

In this policy grid, we can observe that in the local solution areas, while the policy in the state with positive reward (i.e., (0,9) and (9,2) using the plot axes) is to attempt to stay in the state, the surrounding state's policy (e.g., (1,9) - R and (9,3) - L) is to direct the agent towards the global solution at (9,9). This indicates that the right policy is indeed learnt as a local solution area was not being formed and overall, the policy directs an agent towards the sole global solution.

These 2 additional examples using randomly generated environments demonstrate that complexity is not the sole determinant of whether the right policy can be found. Indeed, based on my definition of complexity, the environment in Figure 13 is more complex than that in Figure 11 in both number of states and number of features, yet using the current algorithms, the right policy could still be found for a more "complex" environment and not for the "simpler" environment.

However, based on general observations, as the number of features and states increase, more possible local solutions may exist within the environment, and their longer distance away from the global solution may hinder the ability of the algorithm to find the right policy (i.e., to direct an agent towards the global solution and avoid forming local solutions). Hence the probability of finding the right policy for more complex environments should decrease but remain non-zero.

Possible extensions of this report will be to find alternative complexity metrics with more refined measures of complexity, such as connectivity of walls, clustering of rewards, or the ratio of negative to positive rewards.

Better heuristic valuations can also be proposed to find the “right” policy, such as measuring average return over a simulated run or the agent’s time to reach a high-reward state.