(a)

Algorithm 1 Value Iteration, for estimating $\pi \approx \pi^*$ [1]

```
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\text{terminal}) = 0$
repeat
    $\Delta \leftarrow 0$
    for each $s \in \mathcal{S}$ do
        $v \leftarrow V(s)$
        $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    end for
until $\Delta < \theta$
Output a deterministic policy, $\pi \approx \pi^*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)[r + \gamma V(s')]$
```

The Value Iteration is implemented based on the pseudo-code given in Reinforcement learning: an introduction[1].

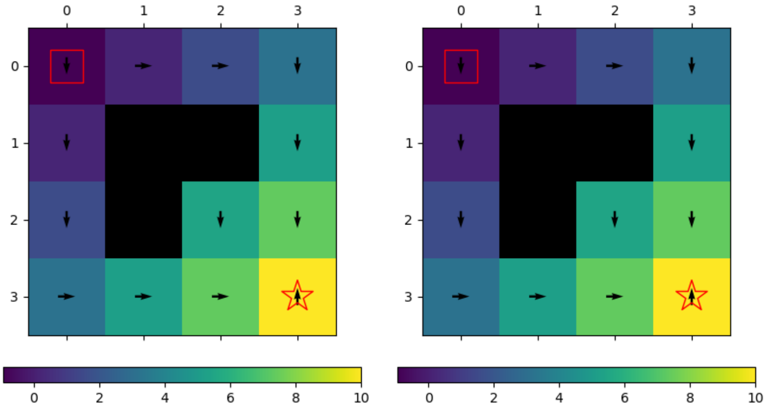


Figure 1: Comparison of Policy Iteration and Value Iteration in the small world set up.

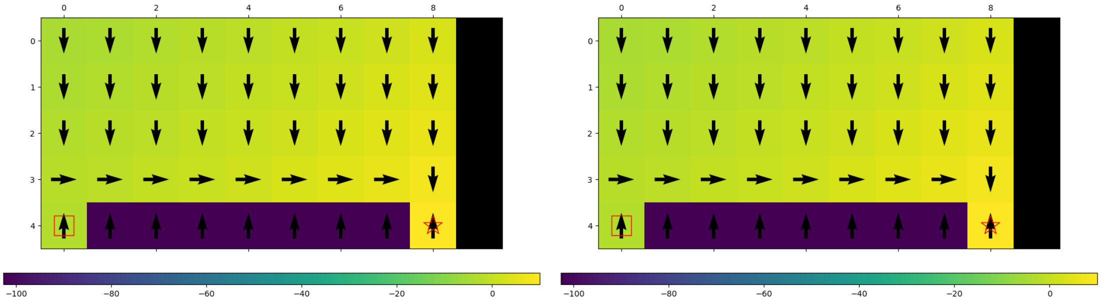


Figure 2: Comparison of Policy Iteration and Value Iteration in the cliff world set up.

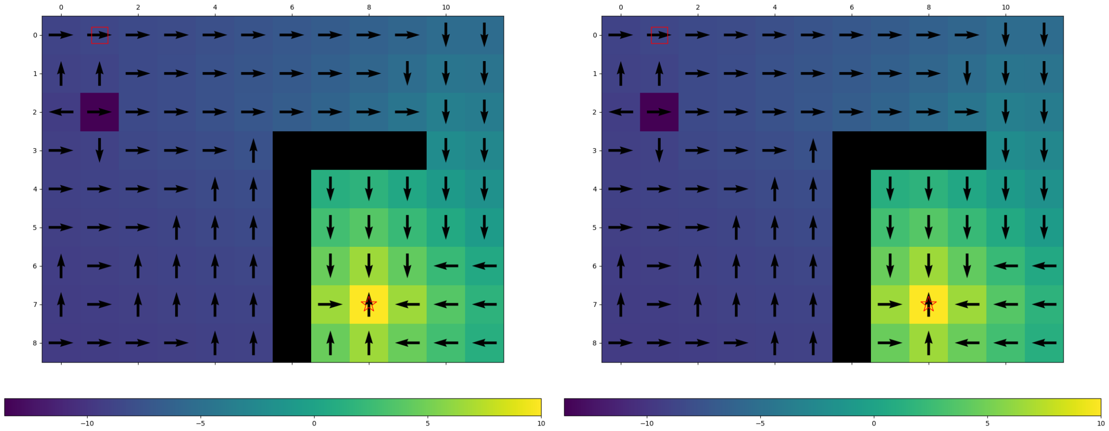


Figure 3: Comparison of Policy Iteration and Value Iteration in the grid world set up.

From Figure 1, 2 and 3, we could verify that the Policy Iteration and Value Iteration has converged to the same policy.

The stopping criteria for Value Iteration and Policy Iteration can be compared by examining the specific values they aim to optimize. Value iteration focuses on optimizing the value of each state, whereas Policy Iteration focuses on optimizing the policy itself. Therefore, the convergence in Value Iteration can be determined by assessing the change in the value of states across consecutive iterations. Specifically, in Value Iteration, the outcome from the value function is encapsulated in a vector V, with one entry for each state. The convergence criterion involves calculating the maximum change between the V vectors from two consecutive iterations. In contrast, the convergence of Policy Iteration can be verified by observing the number of changes in the policy across two consecutive iterations. This method checks for stability in the policy, signifying that an optimal policy has likely been reached when there are no or minimal changes between iterations.

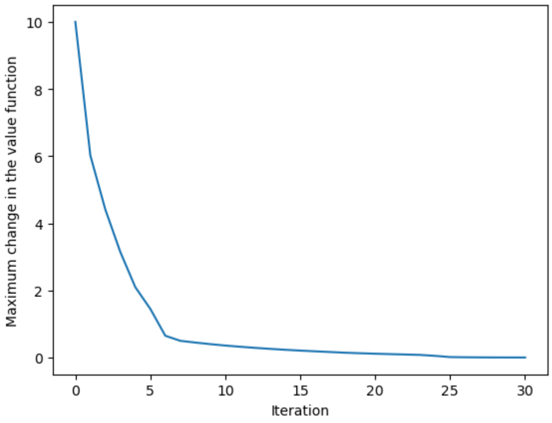


Figure 4: Maximum change in the value function with threshold 0.001 in grid_world

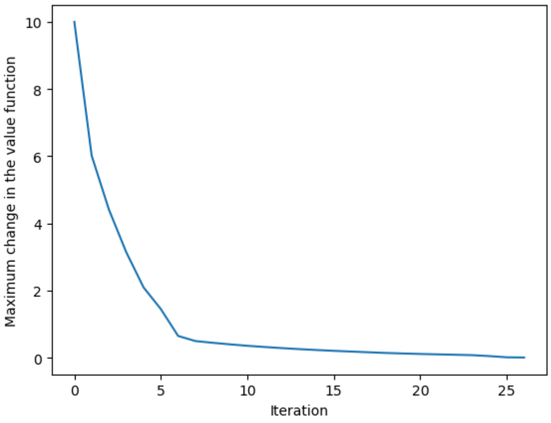


Figure 5: Maximum change in the value function with threshold 0.01 in grid_world

From Figure 4 and 5, which is the maximum change in the value function in asynchronous Value Iteration, as we change the threshold to a larger value, the algorithm would converge to the optimal solution in an earlier iteration.

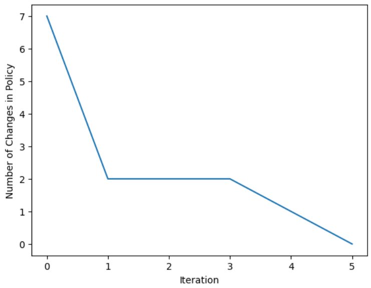


Figure 6: Number of changes in policy over iterations in small_world

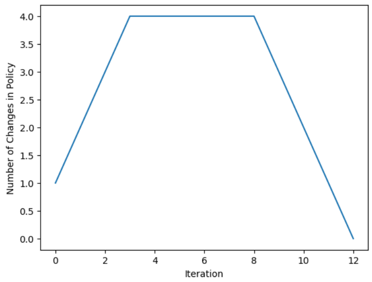


Figure 7: Number of changes in policy over iterations in cliff_world

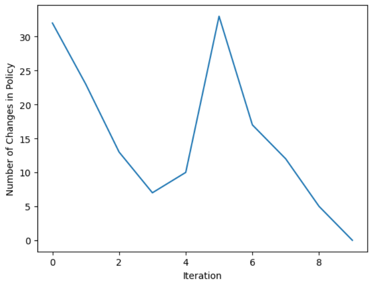


Figure 8: Number of changes in policy over iterations in grid_world

From Figure 6, 7 and 8, we could observe that the Policy Iteration tends to change policy more frequently at early iterations and the number of changes decreased as the algorithm converged.

In summary, an appropriate stopping criterion for Value Iteration could be when the maximum change in state values between two consecutive iterations falls below a specified threshold. For Policy Iteration, a suitable stopping criterion would be no change in the policy across two consecutive iterations.

The comparison between asynchronous and synchronous Value Iteration hinges on their differences in memory complexity. Asynchronous Value Iteration, which updates state values in place, is more memory-efficient, requiring only one copy of the state values. In contrast, synchronous Value Iteration requires maintaining an additional copy of the state values and sweeps of the entire state values at the end of each iteration, which is potentially expensive when the state space is large. In summary, asynchronous Value Iteration is more efficient in terms of memory complexity and does not copy the entire state values at the end of each iteration. However, avoiding sweeps does not mean that the asynchronous methodology needs less computation. It simply implies that the algorithm does not need to compute the long sweep before it can improve a policy.

The computational efficiency of Value Iteration and Policy Iteration will be compared. In each iteration, the time complexity of Value Iteration is $O(|S| \cdot |A| \cdot |S|) = O(|S|^2 |A|)$, where $|S|$ is the number of states, and $|A|$ is the number of actions. By contrast, for Policy Iteration, in the policy evaluation step, the time complexity is $|S||S| = O(|S|^2)$; in the policy improvement step, the time complexity is $|S| \cdot (|A| \cdot |S|) + |A| = O(|S|^2 |A|)$. The complexity of Value Iteration and Policy Iteration are both $O(|S|^2 |A|)$. In summary, the Policy iteration in each iteration would have higher complexity due to the policy evaluation step. However, Policy Iteration, in general, converges faster than Value iteration. So it would be preferred when the number of states is relatively small.

(b)

The SARSA is implemented based on the pseudo-code given in Reinforcement learning: an introduction[1].

Algorithm 2 Sarsa (on-policy TD control) for estimating $Q \approx q_*$ [1]

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in S^+$, $a \in A(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$
**for** each episode **do**
    Initialize $S$
    Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
    **while** $S$ is not terminal **do**
        Take action $A$, observe $R, S'$
        Choose $A'$ from $S'$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$
        $S \leftarrow S'$; $A \leftarrow A'$;
    **end while**
**end for**
```

The following experiments are all conducted on small_world.

In the first set of experiments, we would investigate the effect of learning rate α.

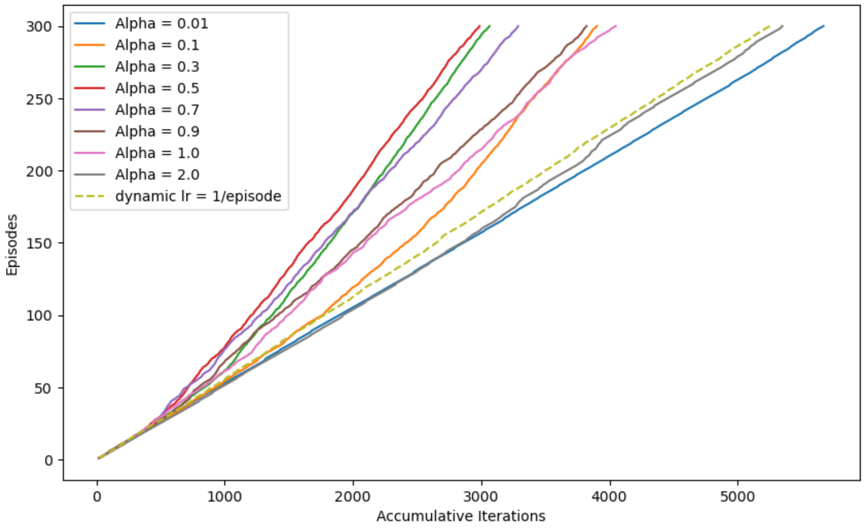
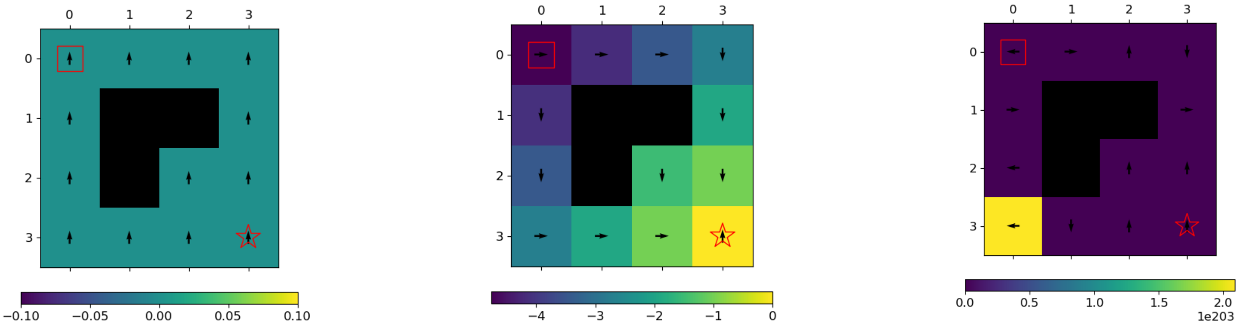


Figure 9: Episodes vs. Accumulative Iterations for difference Learning Rate Setup

An increasing slope in Figure 9 shows the goal was reached more quickly over time. No change in slope from the beginning to the end indicates the model failed to converge in general; too small and too large learning rate will not converge; within the appropriate range, a smaller learning rate will converge faster. 0.5 will be the most optimal learning rate.



(a) Learning rate $\alpha = 0$

(b) Learning rate $\alpha = 0.5$

(c) Learning rate $\alpha = 5.0$

Figure 10: Comparative study of different learning rates

Figure 10 shows the converged policy for three different learning rate setups. It could be observed that if the learning rate is too low, or for the extreme case, set to 0, the algorithm will not learn at all. By contrast, if the learning rate is set to a large value, for example, 5.0, the algorithm fails to find an optimal solution because all rewards are huge.

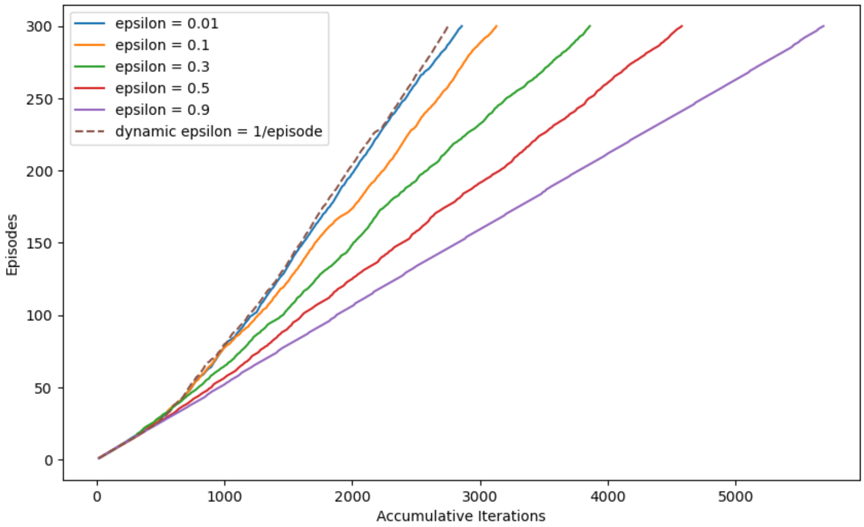
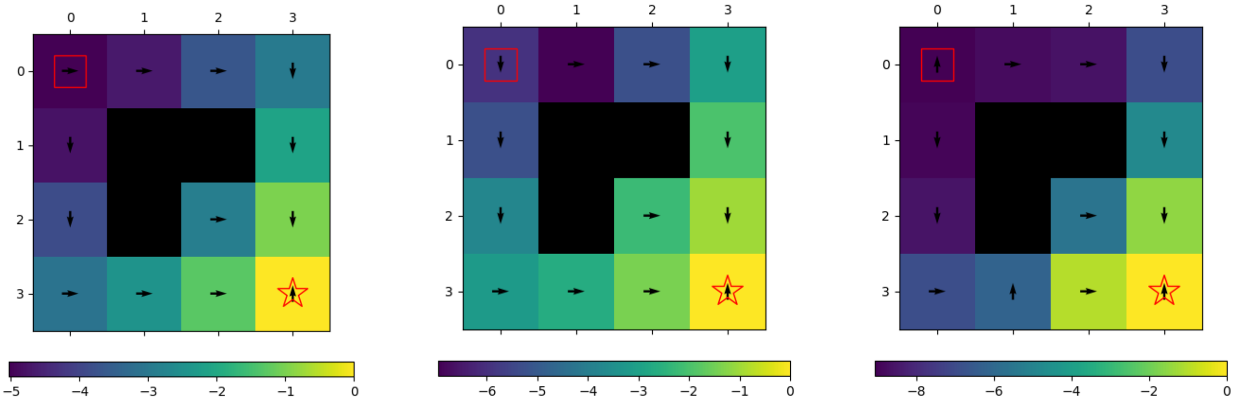


Figure 11: Episodes vs. Accumulative Iterations for difference Epsilon Setup

When ϵ is smaller, the algorithm favours exploitation over exploration. In the small_world setting, a smaller value of ϵ converges faster since the environment has a small state space. Additionally, the most optimal exploration rate is $\epsilon = 1/episode$, which implies that SARSA converges faster using a decaying exploration rate ϵ.



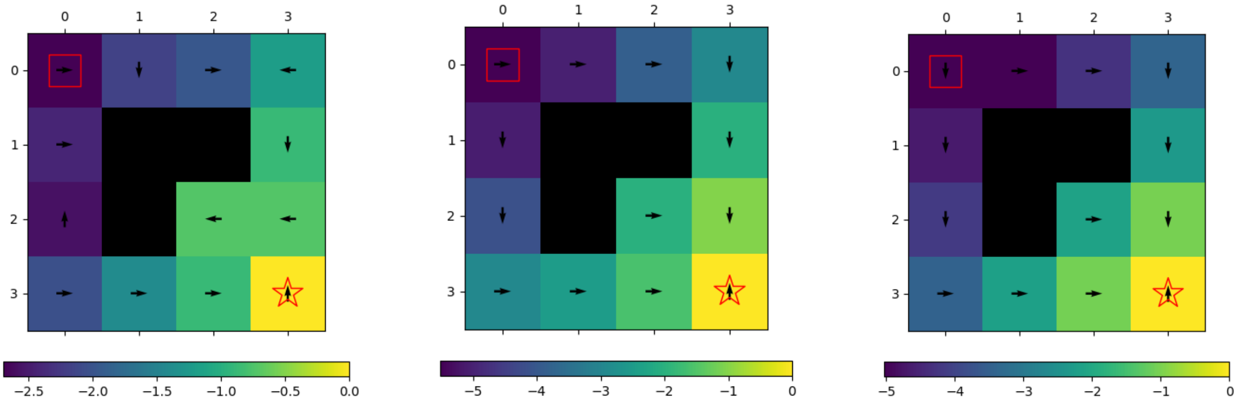
(a) Dynamic Epsilon

(b) Epsilon 0.1

(c) Epsilon 0.9

Figure 12: Policies converged with different epsilon values

According to Figure 12, if ϵ is set to a small value, for example, 0.1, the model will fail to find the optimal solution. By contrast, if ϵ is set to a large value, the model seldom exploits, and the algorithm is almost exploring randomly, which makes it hard to converge.



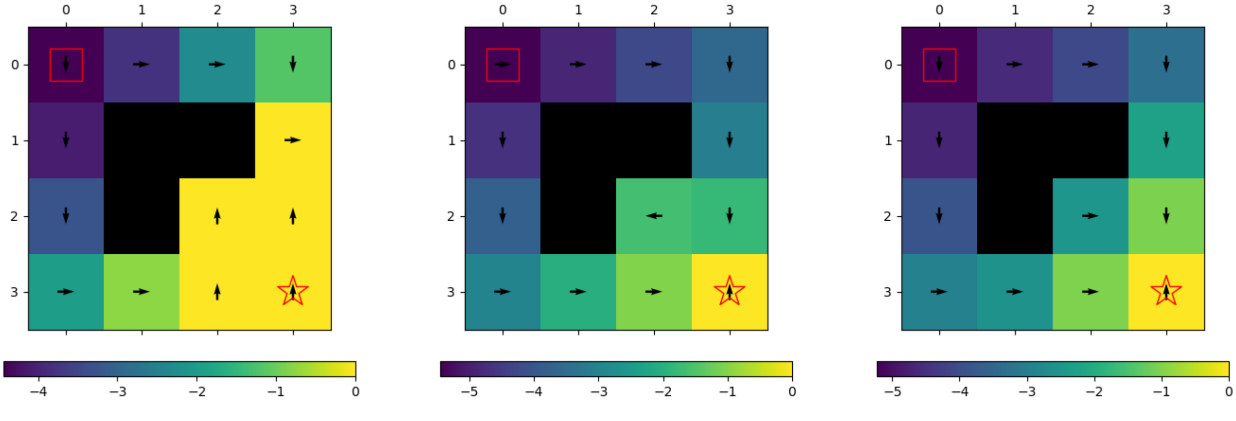
(a) Episode 10, MaxIter 30

(b) Episode 100, MaxIter 30

(c) Episode 200, MaxIter 30

Figure 13: Policies converged for episodes 10, 100, and 200 with MaxIter 30

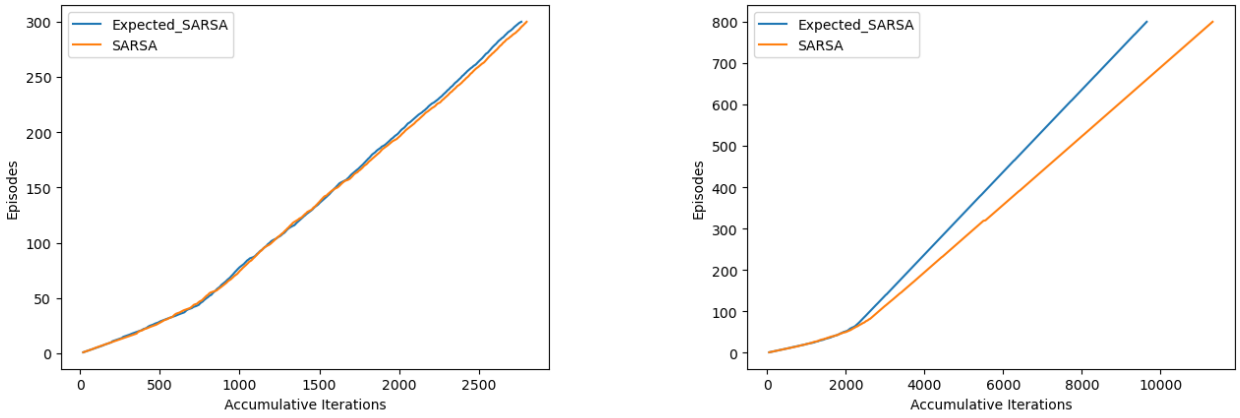
Based on Figure 13, it's evident that employing a minimal number of episodes results in inadequate exploration of the environment by the agent, leading to its failure in converging to the optimal solution. Enhancing the episode count allows the agent to thoroughly explore the environment, thereby facilitating convergence to the optimal solution.



(a) Episode 100, MaxIter 5 (b) Episode 100, MaxIter 10 (c) Episode 100, MaxIter 20

Figure 14: Policies converged for 100 episode with MaxIter 5, 10, and 20

Figure 14 illustrates that the maximum iteration parameter, which sets the cap on the number of steps an agent can take in a single episode, plays a crucial role in the convergence process. A very low maximum iteration limit restricts the agent's ability to thoroughly explore the environment, preventing it from converging to the most optimal solution.



(a) small_world (b) cliff_world

Figure 15: Episodes vs. Accumulative Iterations for Expected SARSA and SARSA in small_world and cliff_world using same hyperparameter setup, with $\alpha = 0.5$ and $\epsilon = 1/episode$

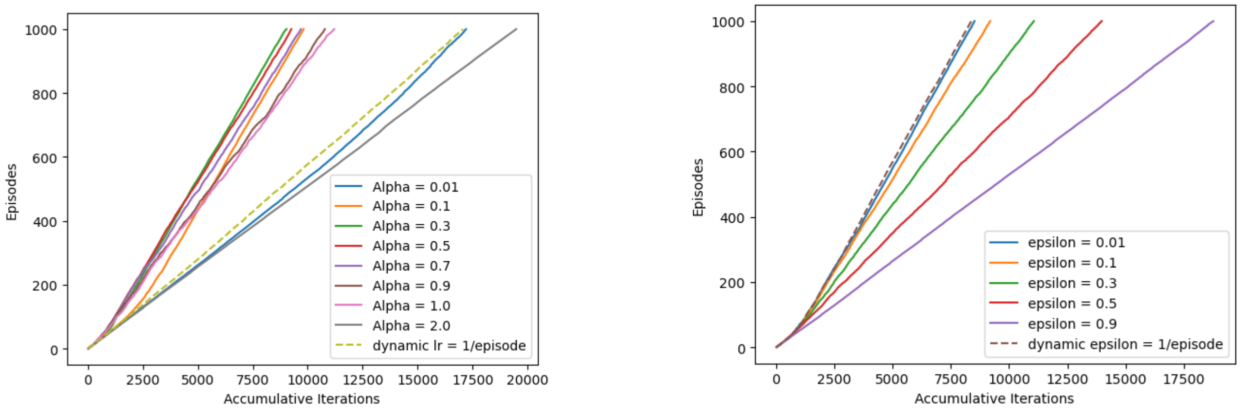
Expected SARSA is generally more stable and efficient in learning since it incorporates the expected outcome of the current policy instead of the outcome of a specific action, which results in faster convergence when compared with SARSA in some scenarios. From Figure 15a, we could observe that two algorithms converged almost simultaneously since small_world is very simple. However, it could be observed that in cliff_world, the Expected SARSA converged faster than SARSA.

(c)

The Q-learnig is implemented based on the pseudo-code given in Reinforcement learning: an introduction[1].

Algorithm 3 Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+$, $a \in \mathcal{A}(s)$, arbitrarily except that $Q(\text{terminal}, \cdot) = 0$
repeat
    Initialize $S$
    repeat
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\epsilon$-greedy)
        Take action $A$, observe $R, S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
        $S \leftarrow S'$
    until $S$ is terminal
until convergence
```

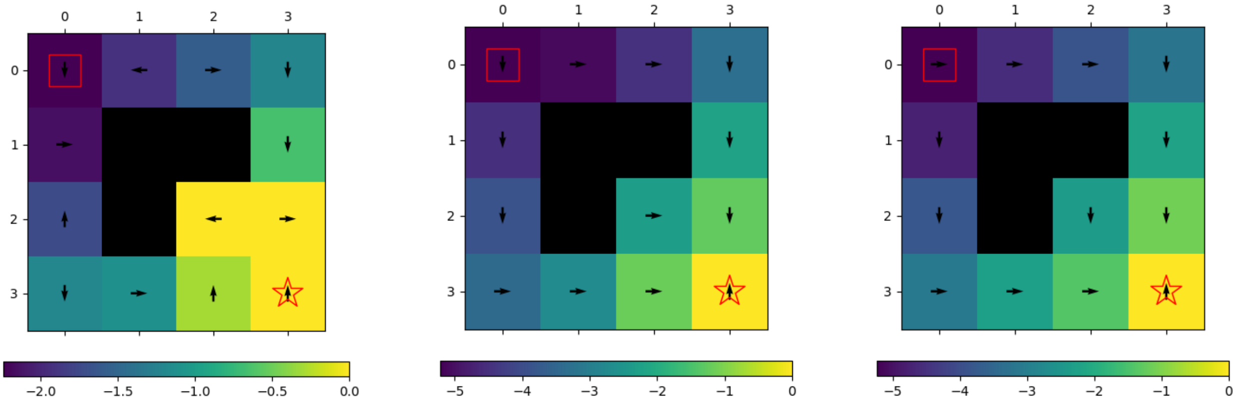


(a) Learning rate α

(b) Epsilon ϵ

Figure 16: Comparative analysis of Q-learning parameters' impact on episodes over accumulative iterations.

The results of Q-learning parameter tuning in small_world environments, as depicted in Figure 16, reveal insights into the optimal settings for the learning rate (α) and exploration rate (ϵ). Specifically, a learning rate ranging from 0.1 to 0.7 yields the highest performance, indicating that rates below this range slow convergence, while higher rates slow done or even prevent the model from converging. Concerning the exploration rate (ϵ), a lower rate is associated with quicker convergence, as it increases the likelihood of selecting the optimal next action rather than engaging in random exploration. Moreover, a dynamic (decaying) exploration rate demonstrates superior performance by promoting extensive exploration in the initial phases and focusing on exploitation after identifying promising paths in later stages.

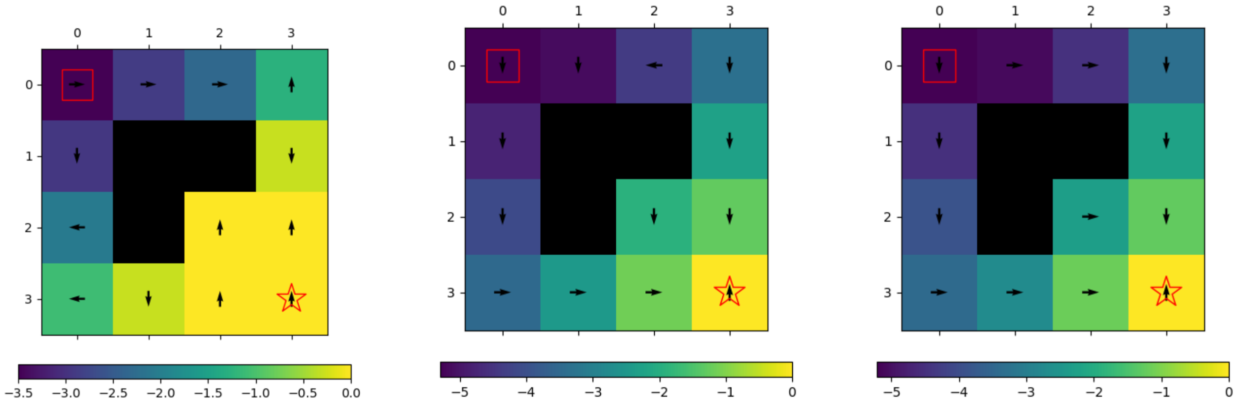


(a) Episode 10, MaxIter 30

(b) Episode 100, MaxIter 30

(c) Episode 200, MaxIter 30

Figure 17: Policies converged for 30 MaxIter with 10, 100, 200 episode $\alpha = 0.3$ and $\epsilon = 1/episode$



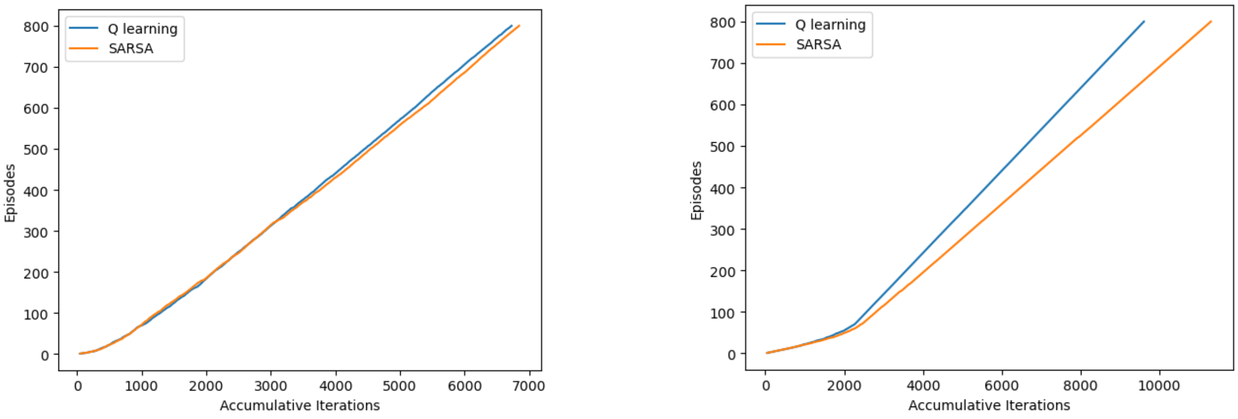
(a) Episode 100, MaxIter 5

(b) Episode 100, MaxIter 20

(c) Episode 100, MaxIter 30

Figure 18: Policies converged for 100 episodes with MaxIter 5, 20, and 30 with $\alpha = 0.3$ and $\epsilon = 1/episode$

Figures 17 and 18 demonstrate that an adequate number of episodes and maximum iterations play a crucial role in enhancing the quality of converged results. Specifically, the final converged policy of Q-learning, as shown in Figure 17c, is optimal. Similarly, SARSA has successfully converged to the optimal solution, indicating that both algorithms are capable of achieving high-quality outcomes when provided with sufficient training iterations and episodes.



(a) small_world

(b) cliff_world

Figure 19: Comparison of the efficiency of Q-learning and SARSA

Figure 19 shows the comparison of the speed of convergence for Q-learning and SARSA in small_world and cliff_world environments. It can be observed that in the small_world environment, Q-learning converges slightly faster due to its relative simplicity. However, for the cliff_world environment, it is shown that SARSA converges more safely, though Q-learning may reach convergence faster with a smaller number of cumulative iterations. In terms of safety, SARSA is considered safer during exploration, as it takes into account the current policy and the associated risks of actions. In contrast, Q-learning might choose a riskier strategy if it promises higher rewards. When taking the wrong step leads to major setbacks and sub-optimal actions that result in a significant drop in Q-values, SARSA may prove more efficient in such environments. Conversely, Q-learning could be more efficient in stable and deterministic environments where exploring the optimal actions without consideration of the current policy is preferred.

(d)

Given that learning rates of 0.3 and 0.5 yield optimal outcomes for both SARSA and Q-learning, with their performances in SARSA and Q-learning being nearly indistinguishable, we have decided to standardize the learning rate at 0.3 for subsequent experiments and focus on studying the difference observed when using difference epsilon setup.

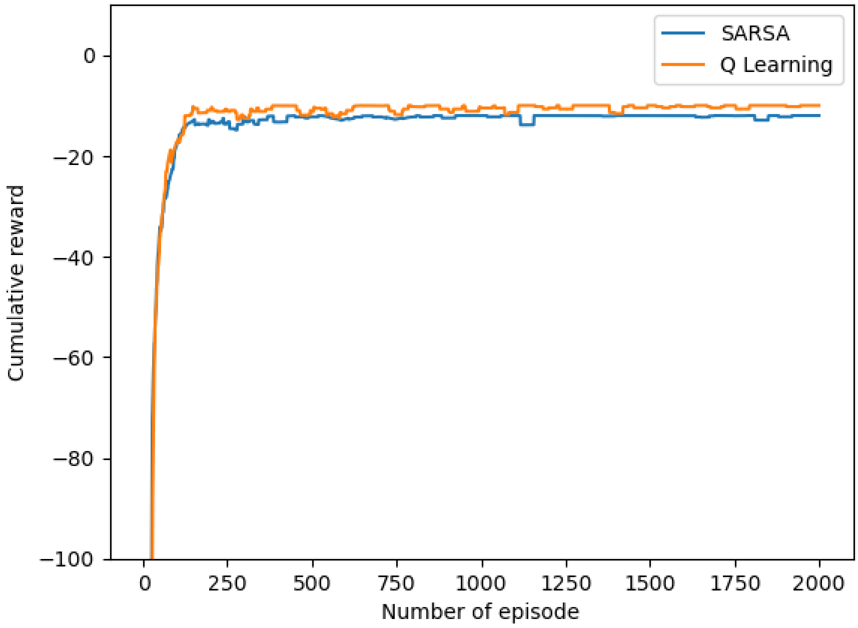
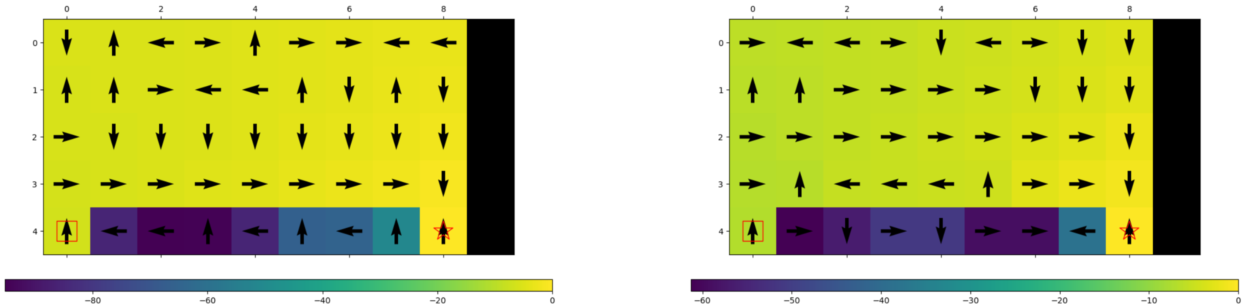


Figure 20: Accumulated reward per episode using learning rate $\alpha = 0.3$ and decay exploration rate ϵ

Figure 20 illustrates that, when employing a decaying ϵ strategy, the learning trajectories of the two algorithms exhibit a high degree of similarity, characterized by a gradual reduction in exploration over the course of training. Furthermore, both algorithms ultimately converge to comparable levels of reward.



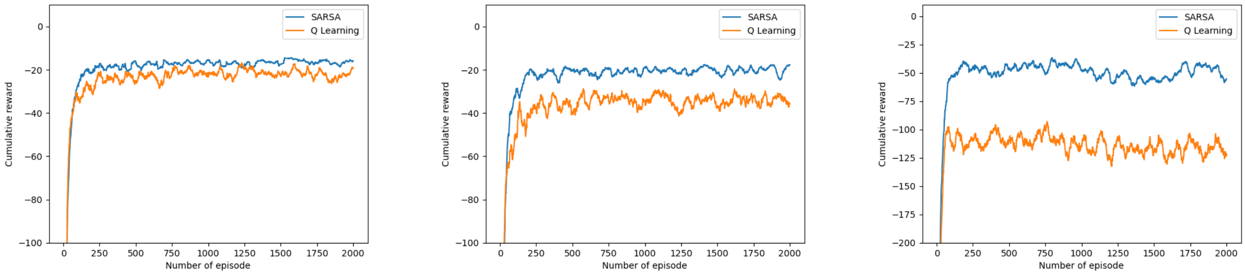
(a) Q learning

(b) SARSA

Figure 21: Comparison of Policy converged using decay epsilon ϵ in Q learning and SARSA

In Figure 21, it is evident that Q-learning has adopted a path that skirts closely alongside the cliff, whereas SARSA has opted for a path that maintains a safer distance from the cliff.

When both algorithms operate under a constant value for ϵ, the disparity in their performance becomes significantly evident.



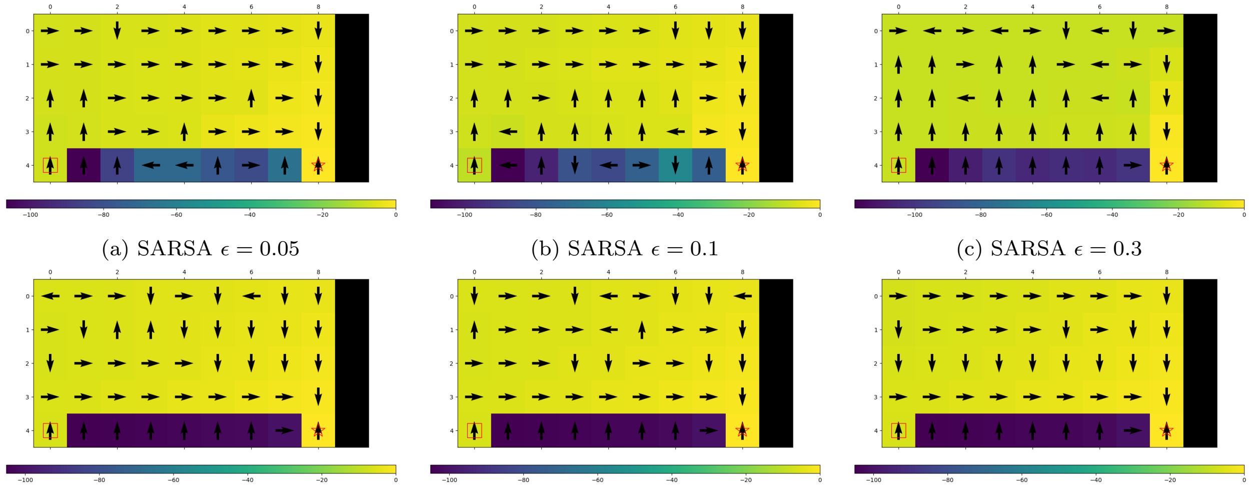
(a) $\epsilon = 0.05$

(b) $\epsilon = 0.1$

(c) $\epsilon = 0.3$

Figure 22: Accumulated reward per episode using different ϵ

In Figure 20, it is observed that SARSA generally achieves a higher cumulative reward compared to Q-learning. This distinction arises because SARSA, an on-policy learning algorithm, tends to adopt a more cautious strategy that aims to minimize the risk of receiving significant penalties, such as falling off the cliff. In contrast, Q-learning, being an off-policy algorithm, is more focused on discovering the optimal policy, even at the risk of incurring higher penalties during exploration. Furthermore, as the exploration parameter ϵ increases, leading to more random actions, the gap in cumulative rewards between the two algorithms widens. This increase in randomness affects Q-learning more severely, resulting in significantly lower cumulative rewards compared to those of SARSA, as Q-learning's pursuit of the optimal policy makes it more susceptible to the adverse effects of increased exploration.



(a) SARSA $\epsilon = 0.05$

(b) SARSA $\epsilon = 0.1$

(c) SARSA $\epsilon = 0.3$

(d) Q-Learning $\epsilon = 0.05$

(e) Q-Learning $\epsilon = 0.1$

(f) Q-Learning $\epsilon = 0.3$

Figure 23: Comparison of converged policy for SARSA and Q-Learning algorithms with different epsilon values

From Figure 23, under the same exploration rate ϵ, SARSA, which is an on-policy learning algorithm, demonstrates caution by accounting for the potential penalties associated with exploratory actions. This leads it to avoid risky paths near the cliff. Conversely, Q-learning, a model that ignores the outcomes of exploratory actions, focuses solely on maximizing rewards. This approach encourages it to undertake risky actions near areas with high penalties.

Furthermore, as the exploration rate ϵ increases, SARSA exhibits even greater caution, deliberately avoiding areas with high risk to ensure safety. This behaviour stems from its learning process, which incorporates the consequences of actions taken during exploration. On the other hand, the policy derived from Q-learning remains unaffected by changes in the exploration rate. This is because it prioritizes reward maximization, disregarding the risks introduced by exploration.

(e)

Why function approximation (of the value function) could be useful?

In environments with simple dynamics and small state spaces, it is feasible to use a tabular representation for the policy and value functions, assigning a unique value to each state or state-action pair. However, in scenarios with large or continuous state spaces, maintaining such a table becomes impractical due to constraints on memory, computation, and the need for generalization. In these cases, function approximation is advantageous as it employs a parameterized function to represent the policy or value functions, thus reducing the dimensionality and complexity of the problem.

In the small_world environment where the placement of barriers in three specific cells is randomized, a tabular approach without function approximation would need to account for a state space of $\overline{\mathcal{S}} = \mathcal{S} \times \mathbb{Z}_2^3 = 16 \times 8 = 128$ distinct states, considering all permutations of the agent's position and barrier configurations. Conversely, using function approximation, the value function $V(\overline{s})$ can be approximated as $V(\overline{s}, \mathbf{W})$, where $\overline{s}$ is a feature vector representing both the agent's position s and the presence or absence of barriers (encoded as $barrier_1, barrier_2, barrier_3$, with $barrier_i = 1$ if the i-th barrier cell contains a barrier, and $barrier_i = 0$ otherwise). Here, $\mathbf{W}$ denotes the weight vector of the function approximator.

This model effectively condenses the state representation from 128 to a 19-feature vector, where the first 16 features could be a one-hot encoding of the agent's location, and the last three features represent the binary states of the barriers. The function approximation thus captures the essential information required to estimate the value function while enabling the model to generalize across a range of states.

Would linear function approximation using $\phi(s) \doteq s$ suffice?

No, it is not sufficient. When employing $\phi(s) \doteq s$, the value function is expressed as a weighted sum of state values:

$$V(s, W) = w_1 b_1 + w_2 b_2 + w_3 b_3 + w_4 s + w_0, \tag{1}$$

where each weight w_i captures the importance of a single feature, such as the presence of a barrier in a specific cell. However, this method does not account for the interaction between features, which is crucial in grid worlds where understanding how different features combine affects the state's value. Linear functions struggle to capture these interactions.

To address this, another linear approach using basis functions is proposed:

$$V(s, W) = \mathbf{w}^\top \mathbf{x}(s) = \sum_{i=1}^{d} w_i x_i(s), \tag{2}$$

where $\mathbf{x}(s)$ is a feature vector representing state s and $x_i : S \to \mathbb{R}$, introducing interaction between features. This allows for a linear approximation of V that accounts for feature interplay.

In summary, function approximation with basis functions provides a useful linear approximation of the value function, capturing interactions between features.

References

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* Cambridge, MA, USA: A Bradford Book, 2018.