Reinforcement Learning - Coursework 1

CID: 01727810

I. Dynamic Programming

To employ Dynamic Programming (DP) methods, we initially make the fundamental assumption that a perfect and complete model of the environment is at our disposal [1]. Given the relatively small number of states in the maze, we choose to proceed with synchronous DP methods. This is also because the dependency structure of the problem at hand is well-known. Our method of choice for this scenario is Policy Iteration, renowned for its exhaustive search approach and less computationally expensive algorithm. This ensures the convergence to the optimal policy, denoted as π_*, and the corresponding value function, V_*, as it intrinsically selects the policy that maximises the Bellman equation [1]. Additionally, the maze's structure is non-dynamic and inherently predictable, facilitating the derivation of a deterministic policy. Nonetheless, for small problems like this, the choice between the two methods may have a smaller impact on overall performance and computational time (with a small measured difference of ~ 0.5s).

In practice, this approach isn't commonly employed because agents face constraints related to memory and computational resources per time step, even when equipped with a precise environment model. It faces the curse of dimensionality, because in complex problems with many state variables (position, orientation, etc.), the state space can become incredibly large, and in fact increases exponentially with these variables. These limitations limit the agent's capacity to leverage the model fully. It's important to highlight that constructing accurate approximations of value functions, policies, and models often demands substantial memory resources, and, as a result, approximations are frequently employed. In our report, we will not be considering this scenario since the environment is small, and the limitations related to memory and computation per time step do not apply.

A. Parameters

To determine the optimal threshold value, θ, we conducted an analysis that considered the trade-off between convergence speed and accuracy. This trade-off was quantified using the Mean Squared Error (MSE) relative to V_* calculated with $\theta = 10^{-20}$. The relationship between θ, MSE, and the number of epochs is visually represented in Fig. 1. After examining the graph, we selected a value of $\theta = 0.001$ as the optimal threshold. This choice strikes a favourable balance between the MSE, which is $\mathcal{O}(10^{-12})$, and the number of epochs required per run is $\mathcal{O}(10^2)$. Using a larger threshold introduces noticeable rounding errors, and the reduction in the number of epochs achieved is negligible, making it an impractical choice.

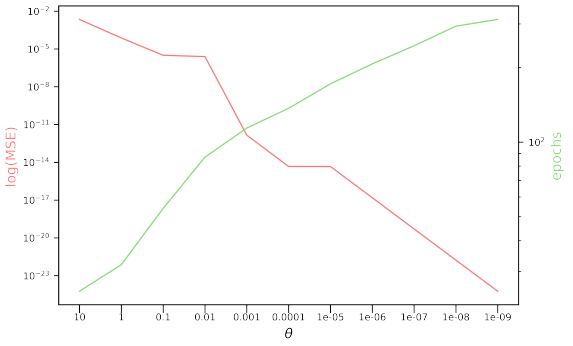


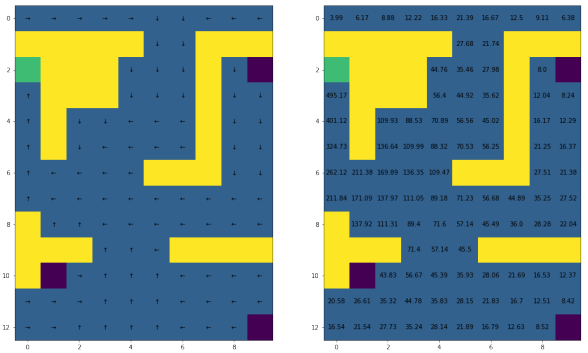
Fig. 1: **MSE and Epochs of ΔV_* as a function of θ:** Graph used to visually obtain the parameter θ, showing the exponential increase of epochs of Policy evaluation, and exponential decrease of MSE with θ. The MSE error of the threshold θ is obtained by calculated between $V_*(\theta)$ and $V_*(\theta = 10^{-20})$.

Nevertheless, it is crucial to consider the potential need for near-optimal solutions as the problem size increases. Deeper analysis and exploration may be necessary in such scenarios because achieving the precision specified by θ can be highly computationally demanding, as a significant portion of time is dedicated to verifying optimality in the code [2].

B. Optimal Policy and Value functions

The DP agents' V_* and π_* is shown in Fig. 2.

This figure clearly pictures the Principle of Optimality of DP, which states that wherever the agent starts and whatever the agent's initial decision is, we will reach an optimal policy for that initial state [3]. The recursive nature of



(a) **Optimal Policy** (b) **Optimal Value function**

Fig. 2: **Graphical representation of the Optimal Policy and Value function for Dynamic Programming (Policy Iteration):** Derived using a threshold $\theta = 0.001$, and CID specific discount factor $\gamma = 0.82$ and probability of success $p = 0.96$.

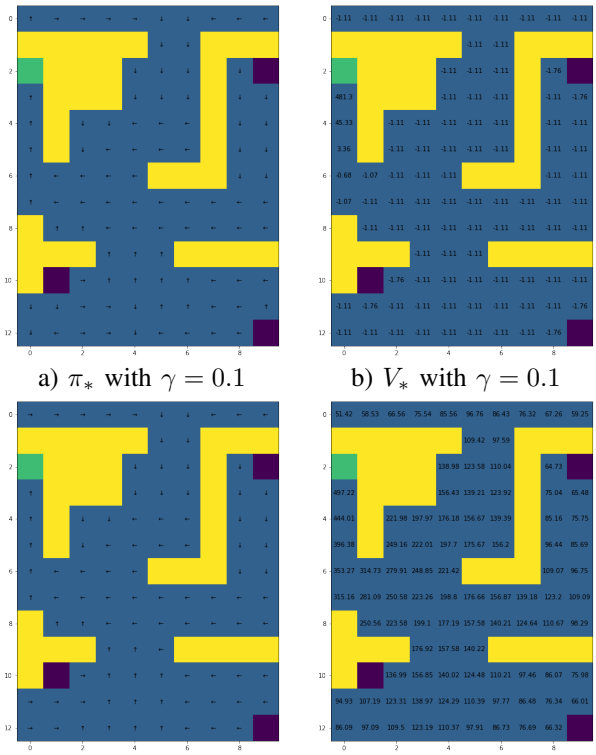
the Bellman equation guarantees that the agent examines the implications of all the actions in each state, ensuring that no possibilities are left unexplored.

C. Discount factor and Probability of success

The parameters γ and p affect the agents' behaviour, the stability of learned policies, and the convergence of value functions. As a result, π_* and V_* will be altered by changing these parameters.

When $\gamma < 0.5$, the agent significantly devalues (or *discounts*) future rewards, with the limit of $\gamma = 0$ meaning it only cares about immediate rewards. This effect is evident in the policy iteration algorithm, where reducing the value of γ leads to the agent giving less weight to the anticipated value of future states. The Value function and Policy obtained with $\gamma = 0.1$ are plotted in the top row of Fig. 3. We can clearly see the impact of γ on V_*, where now, most states have a similarly low value, except those that are immediately next to an absorbing state that has a negative reward of $r = -50$, and those that are *close enough* to the terminal state with a larger reward of $r = 500$. Given these values, the agents π_* take some inefficient actions, as it's 'short-sighted'.

On the other hand, when $\gamma > 0.5$, the agent places a higher value on long-term rewards and is willing to invest in actions that will lead to greater future rewards. This can be visually summarised in the bottom row of Fig. 3, plotted for $\gamma = 0.9$. In this scenario, the policy and



a) π_* with $\gamma = 0.1$ b) V_* with $\gamma = 0.1$

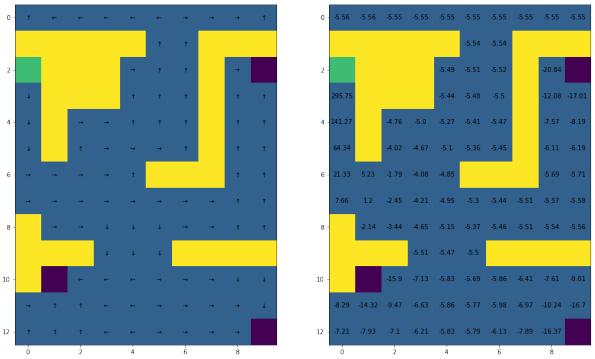
c) π_* with $\gamma = 0.9$ d) V_* with $\gamma = 0.9$

Fig. 3: **Graphical representation of the variation of the Optimal Policy and Value function with different γ values, for Dynamic Programming (Policy Iteration):** This figure illustrates π_* and V_*, with $\gamma < 0.5$ and $\gamma > 0.5$. Derived using a threshold $\theta = 0.01$ and CID $p = 0.96$.

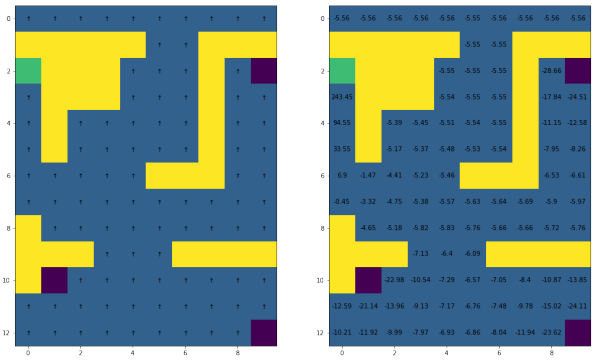
value function more closely resemble the optimal Policy and Value function depicted in Fig. 2a. The value of all states increases, and the agent anticipates higher rewards as it moves further away from its current state towards the terminal state. As a result, the optimal policy minimises the path length from each state to the terminal state. Therefore, setting γ to a relatively large value in our environment is preferable, as there are not many risks present.

The impact of the probability of success p on the agent's behaviour is even more visible, as it directly impacts the chance (or confidence) of taking the action the agent wants to take. A low probability of success $p < 0.25$ implies that if the agent wants to take one of the four actions, it will be more likely to take one that it didn't choose ($\frac{1-p}{3} > 0.25$), as a result, the π_* of this agent will diverge away from the goal state. If $p = 0.25$, all four actions are equally likely, leading to a completely random policy. However, because DP is a greedy algorithm, if all the actions are equally likely, it will stay like that for the rest of the epochs and the

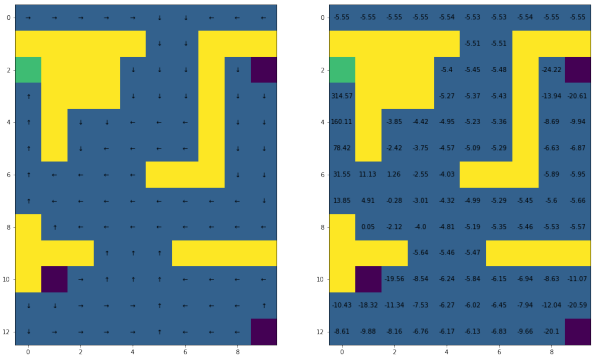
algorithm will never converge. To fix this, we use the *numpy* package with its method $argmax$ to determine the best policy. This picks the same action every time even if all the actions are equally likely, which is why the agent always moves in the same direction, as shown in the middle row of Fig. 4. In this case, the agent will struggle to move towards the goal as it has no preferences. The last situation to consider is when $p > 0.25$, in this situation we approach the optimal solutions π_* and V_*, as the agent is more likely to go where it wants - where it will find the highest rewards.



a) π_* with $p = 0.15$ b) V_* with $p = 0.15$



c) π_* with $p = 0.25$ d) V_* with $p = 0.25$



e) π_* with $p = 0.35$ f) V_* with $p = 0.35$

Fig. 4: **Graphical representation of the variation of the Optimal Policy and Value function with different $p \in [0.15, 0.25, 0.35]$, for Dynamic Programming (Policy Iteration):** This figure illustrates π_* and V_*, with $p < 0.25$, $p = 0.25$ and $p > 0.25$. Derived using a threshold $\theta = 0.001$ and CID discount factor $\gamma = 0.82$.

II. Monte Carlo

To employ Monte Carlo (MC) methods, non-model algorithms, operate under the assumption that there is no prior information available about the environment, and all the information available is gathered by experience. Therefore we do not assume that we possess any information on transition probabilities or reward matrices between states. These methods solve this problem by averaging returns over all sample trajectories for each state. However, to ensure this works, we assume the experience is divided into episodes, and that all episodes eventually terminate no matter where the agent starts and what actions it takes. Once the episode finishes, the value function and policies are updated. Furthermore, we assume that the dynamics of the environment are stationary.

In this report, we chose to implement the First-Visit MC. Both First-Visit and Every-Visit MC are assured to converge to $V_*(s)$ as the number of visits to each state s goes to infinity [1]. Prioritising first visits allows the agent to focus on unique experiences, which can be more efficient in terms of information gain when there are no complex dynamics, such as loops, in the environment. In the Maze, we have fixed starting states, so we decide to employ an ϵ-greedy exploration strategy for choosing actions, so to avoid the assumption of exploring starts. This ensures the agent will continue to select paths that it still hasn't explored enough.

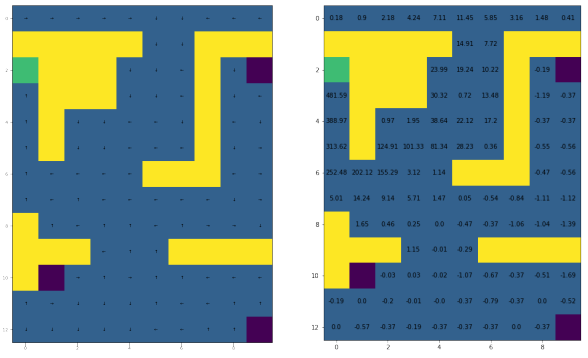
A. Parameters

Given the p and γ are constrained by the CID, the only free parameters in our algorithm are ϵ and the number of episodes. If we set ϵ to be large, we increase the chance of exploring in comparison to acting greedily ($\epsilon = 1$ sets a completely random policy). On the other hand, if we set it to a low value, the agent will act more greedily and stick to the actions that it has seen to give more rewards in the past. Usually, we want to keep some exploration happening, as the maze environment may still have some unexplored states, especially at the start. However, there needs to be a balance between exploration and convergence of the learning curves of our agent. We can reach a balance between both of these, by varying our epsilon according to a decaying distribution such as $\epsilon = x^k$ where k is

the episode number the agent is in, and x is a number close to, but smaller than 1, dictating the rate of decay, and therefore $\epsilon \in (0, 1]$. The closer x is to 1 the slower decay of ϵ. This ensures, that as the agent learns from the Maze, it advantageously reduces exploration and prioritises exploitation, as we will visually see in the learning curve of the agent in Fig. 6. By setting $x = 0.99$, we achieve an $\epsilon \sim \mathcal{O}(10^{-6})$ after just 1000 episodes, which is a small enough probability to ensure we act greedily at the end and converge to a policy, but on the other hand, still provides a large enough ϵ at the beginning to explore enough states. This decay function has been chosen based on the time constraints of running the agent experiments. With more computational speed, other more slowly decaying functions could've been chosen, with a larger number of episodes. Faster decaying functions (such as $1/k$), decay, decay too fast and as a result the agent doesn't have enough time to explore. The number of episodes is set to 3000, as we'll see in Fig. 6, this provides enough episodes for convergence at our rate of exploration decay.

B. Optimal Policy and Value function

The MC agents' V_* and π_* is presented in Fig. 5. In the limit we set an infinitely slowly decaying ϵ, and an infinite number of episodes, we would expect these states to be visited an infinite number and hence approach a policy that resembles DP more closely.



(a) **Optimal Policy** (b) **Optimal Value function**

Fig. 5: **Graphical representation of the Optimal Policy and Value function for Monte Carlo first-visit Control:** Derived using a decaying $\epsilon = x^k$, where $x = 0.99$ and k is the episode number, and CID specific discount factor $\gamma = 0.82$ and probability of success $p = 0.96$.

C. Total non-discounted sum of rewards

The learning curve of the MC agent averaged over 25 runs is plotted in Fig. 6.

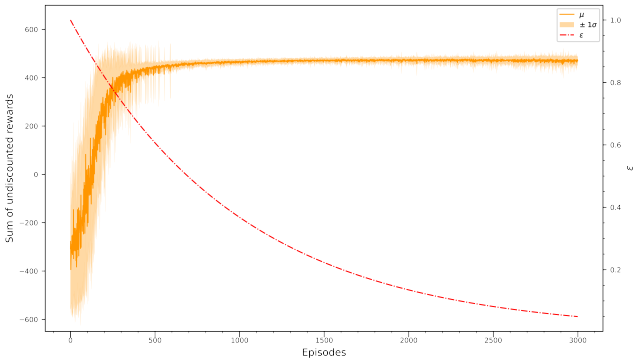


Fig. 6: **Learning curve of MC agent:** The plot shows how the mean μ and standard deviation σ of the sum of undiscounted rewards ($\gamma = 1$), varies with episode number, averaged over 25 different runs. Alongside this, the decaying ϵ is also visualised with $x = 0.999$ (with larger γ, slower decay). Derived using the CID-specific probability of success $p = 0.96$.

III. Temporal Difference

Temporal Difference (TD) learning holds a central position within Reinforcement Learning, offering a unique blend of advantages from both MC and TD methods. TD leverages the model-free approach of MC, eliminating the need for a model of the environment, and integrates aspects of DP, which involve bootstrapping from intermediate estimates without the necessity of waiting for an episode to conclude. Generally, TD converges quicker than MC [1][4]. Nevertheless, it's important to note that TD, like its counterparts, operates under the assumption of a finite, discretised environment. However, like the other reinforcement learning methods, TD relies on the fundamental assumption that the agent can actively engage with the environment and receive feedback in the form of rewards.

In this report, we have opted to implement the Q-learning (QL) algorithm with an ϵ-greedy policy, a significant breakthrough in the field of Reinforcement Learning introduced in 1989 [4]. The key premise underlying QL is the requirement that all state-action pairs continue to be updated for correct convergence. Our choice of QL stems from the understanding of our environment and the distinctions between this algorithm and SARSA. Since our environment does not pose significant risks, on the way to the

terminal state, we do not need to adopt an overly cautious approach. Q-learning represents a more assertive agent, as it learns from the optimal policy. Nonetheless, in the limit as $\epsilon \to 0$, Q-learning and SARSA are identical.

It's important to highlight that for stochastic algorithms such as MC and TD, the methods implemented in this report use a seeding mechanism for the sake of reproducibility and hyperparameter tuning.

A. *Parameters*

The parameters p and γ are predefined by the CID. This leaves us with a set of tuning hyperparameters - the exploration parameter $\epsilon \in (0, 1]$, the learning rate $\alpha \in (0, 1]$, and the number of episodes.

Of particular significance is the exploration parameter ϵ and the learning rate α, as they exert a substantial influence on the agent's behaviour. To determine these, we pursued a methodology that involved a balance between two distinct learning strategies. In our initial approach, we considered DP as a benchmark, assuming that DP provides us with the most optimal value function V_* and policy π_*. We then systematically compared the outcomes of QL against those of TD, for specific pairs of hyperparameters (α, ϵ). To quantitatively evaluate the performance, we computed the Mean Squared Error (MSE) between V_* obtained through QL and the reference value function from DP. The variation in the MSE for different hyperparameters was visualised in a three-dimensional plot, as illustrated in Fig. 7. Through this process, we identified the hyperparameter values that minimised the discrepancy between the QL value function and the DP reference, with MSE reaching a minimum value of approximately $\mathcal{O}(3)$ or ~ 150 around $\alpha = 0.95$ and $\epsilon = 0.74$. It's worth noting that QL is a bootstrapping method, enabling us to use this performance comparison against the value function of TD.

Despite this, the quest for optimal hyperparameters extends beyond minimising error; we are equally concerned with maximising the smoothness and stability of the agent's learning, particularly as we approach the final episodes. To comprehensively assess the impact of our hyperparameters, considering $\gamma = 1$, we turn our focus to the learning curve. Here, we evaluate a Coefficient of Variation (CV) of the curve,

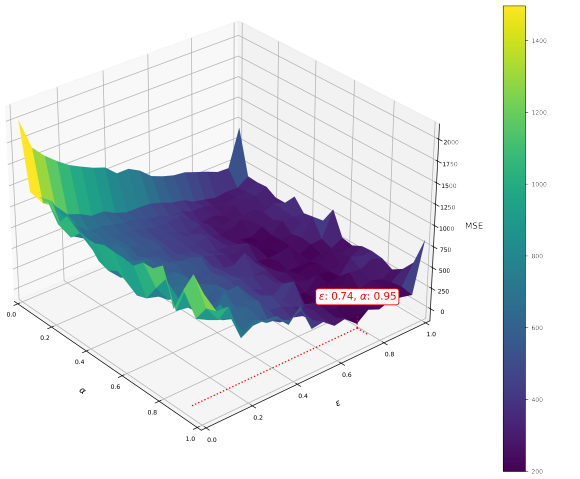


Fig. 7: **Mean Squared Error (MSE) of TD versus DP:** The plot shows the variation of the MSE with varying α and ϵ parameters, with a minima of MSE $= \mathcal{O}(3)$, at $\alpha = 0.95$ and $\epsilon = 0.74$. Ran over 3000 episodes and averaged over 5 runs, using the CID specific $\gamma = 0.82$ and $p = 0.96$.

calculated as the mean divided by the standard deviation over the episodes. Employing the same range of hyperparameters, we present the results in Fig. 8. Extracting the hyperparameters that smoothen and maximise the learning curve of the agent the most, we want to maximise the CV value, which extracted from the figure is at $\alpha = 0.17, \epsilon = 0.01$, where we obtain a $CV_{max} \sim 200 \sim \mathcal{O}(3)$.

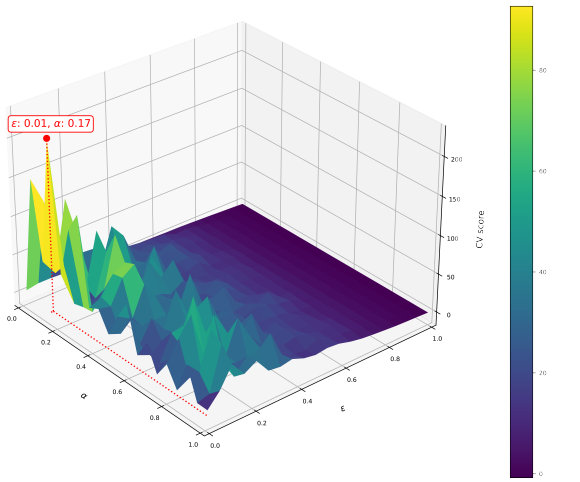


Fig. 8: **Coefficient of Variation (CV) of QL in TD:** The plot shows the variation of CV with different α and ϵ parameters, with a maximum CV $= \mathcal{O}(3)$, at $\alpha = 0.17$ and $\epsilon = 0.01$. Ran over 3000 episodes and averaged over 5 runs, using the CID specific $\gamma = 0.82$ and $p = 0.96$.

After conducting these two experiments, it becomes evident that the selection of α and

ϵ entails a trade-off between minimising MSE and maximising the CV of the learning curve. This trade-off is not unexpected. Opting for larger hyperparameters allows for more extensive exploration of the state space, leading to a greater understanding of previously unknown aspects of the value function. However, it can hinder convergence, as the larger step size and exploration, may encourage the agent to explore instead of exploiting the knowledge acquired throughout the episodes.

But what if we could have the best of both worlds? In response to this challenge, we propose employing decaying hyperparameters, as we previously did in MC. It involves initiating the learning process with large ϵ to facilitate thorough exploration of the maze but scaling this down slowly to promote convergence, yielding a stable and reliable learning curve. This similar reasoning applies to α, as a large value allows us to reach V_* quickly, nonetheless, if it's not reduced, the step size will be too large to converge, and will just oscillate around the optimum value function.

By adopting the same form as in MC, ϵ and $\alpha = x^k$, where $x = 0.999$ dictates the rate of decay and k signifies the episode number, we can compare the respective MSE and CV scores. This analysis is illustrated in Fig. 9, making it clear how the decaying hyperparameters provide the best return for both scores. The choice of x is based on the number of episodes we chose. As mentioned previously, initialising the agent with more episodes and a slower decaying function could lead to better results theoretically.

We can clearly see in Fig. 9, that by adopting decaying hyperparameters, we achieve a perfect balance of learning convergence and smoothness, as well as minimising the $MSE \sim \mathcal{O}(2)$ or ~ 20. The optimal value function and policies will then be calculated using hyperparameters following 0.999^k.

Given the insights gained from Fig. 9, we will employ a default setting of 3000 episodes for our QL agent, as the learning curve of the agent stabilises well before reaching this episode limit.

B. Optimal Policy and Value Function

The QL agents' V_* and π_* is shown in Fig. 10. As expected, the figure exhibits a remarkable similarity to the outcomes of DP, as showcased in Fig. 2.

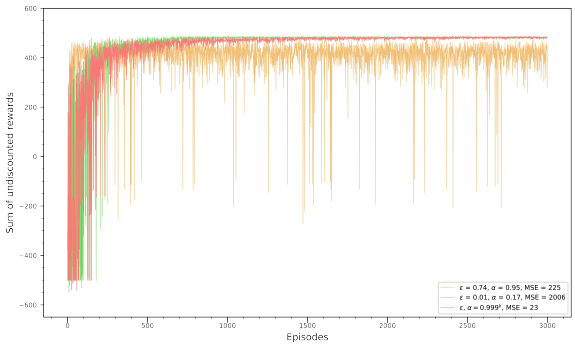
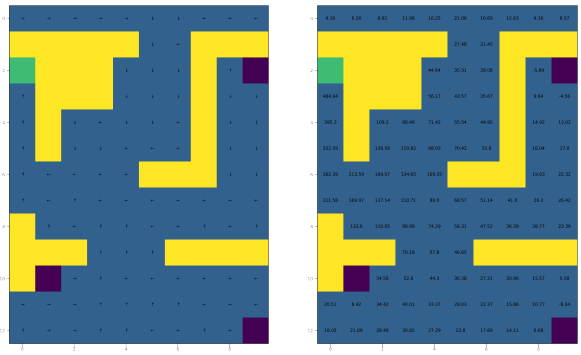


Fig. 9: **Comparison of optimal CV and MSE α and ϵ against decaying hyperparameters for QL:** The depicted plot serves to highlight the pronounced advantages of adopting decaying hyperparameters, specifically α and $\epsilon = 0.999^k$, where k represents the episode number. In comparison to fixed hyperparameters, the use of decaying hyperparameters results in a significantly higher CV score, denoting the smoother and more stable nature of the learning curve. Simultaneously, it yields lower MSE relative to DP, a measure of accuracy, underscoring the effectiveness of this approach. This plot is derived using the CID specified $p = 0.96$, $\gamma = 0.82$ and ran over 3000 episodes.



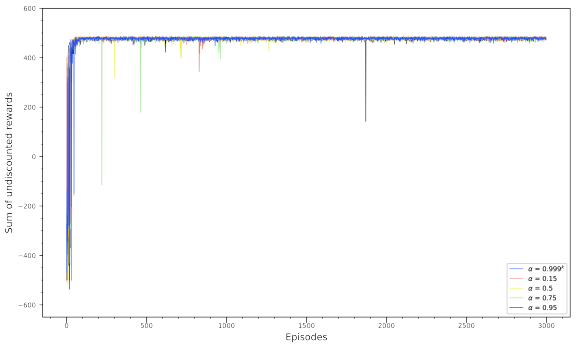
(a) **Optimal Policy** (b) **Optimal Value function**

Fig. 10: **Graphical representation of the Optimal Policy and Value function for TD (Q-Learning):** Derived using $\epsilon, \alpha = x^k$, with $x = 0.999$, and CID specific discount factor $\gamma = 0.82$ and probability of success $p = 0.96$.

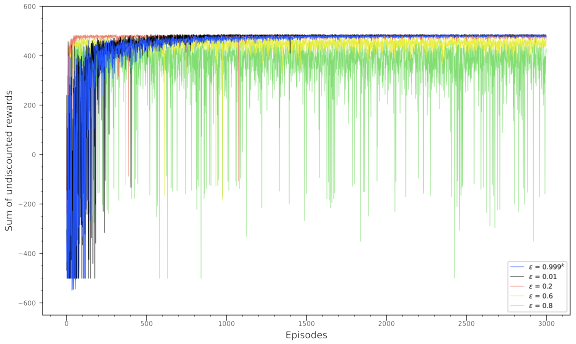
C. ϵ, α and the Learning curves

The learning curves of QL are very dependent on the values of ϵ and α as we saw in Fig. 8. Exploration helps to discover new information about the environment, while the learning rate governs how quickly that information is integrated into the value estimates. Looking at the plot in Fig. 8 again, we expect the learning curves to vary the most with ϵ, as this defines the policy choice directly, meanwhile α, will likely have a bigger influence on the convergence speed and resulting precision of V_*. To understand this behaviour more clearly, we can vary the

hyperparameters one at a time and look at the specific learning curves, to produce Fig. 11.



(a) **Learning curve variation with α:** Plot showing the learning curve and how it varies with different values of α, at a fixed $\epsilon = 0.17$. The plot doesn't show substantial variations with changing α, nonetheless, the decaying α pictures the nicest convergence of the agents' learning.



(b) **Learning curve variation with ϵ:** Plot showing the learning curve variation with different values of ϵ, at a fixed value of $\alpha = 0.1$. The plot shows that the learning curve of the agent is highly influenced by the exploration parameter, where a large value leads to unstable learnings, meanwhile, a smaller or decaying parameter follows a convergent and smoother curve.

Fig. 11: **Learning curves variation with α and ϵ for QL:** This plot is produced from 3000 episodes and using the CID specific parameters, $\gamma = 0.82$ and $p = 0.96$.

Analysing Fig. 11, we see that the agent's learning is primarily driven by its exploration strategies (ϵ), rather than the limited feedback from the rewards it gains (α). Additionally, the maze environment contains predominantly the same rewards, except for the 4 absorbing states - which could explain the small dips in the graph with larger α's, especially towards the start of the curves. The features the larger α's show, are from the agent overshooting the optimal values, which is the trade-off of having the agent learn faster. As the step size decreases leads to more stable learning, nonetheless, with a slower, barely noticeable, convergence speed.

For ϵ in Fig. 11b, we immediately see the overall difference varying the hyperparameter does on the convergence and instability of the agents learning. A higher ϵ leads to much more exploration (and therefore randomness), reflected by the less predictable learning curves and high standard deviation of the agent's learning. However, backed up by what we observed when we optimised the parameters in Section III-A, the choice of ϵ involves a trade-off between noisier and non-convergent learning with potentially higher cumulative rewards due to exploration, and smoother curves that lead to quicker convergence, at the risk of premature convergence to suboptimal policies.

Nonetheless, the decaying α and $\epsilon = 0.999^k$ showcase the best of both worlds. Once we have explored enough at large step sizes, we can adjust α and ϵ to start decreasing, as the paths no longer change significantly, so the learning rate can be finely adjusted by relying on its accumulated knowledge, eventually converging to the optimal policy.

Exploring further decay functions and longer episodes could be an extension of the current report. Additionally, performing further exploration with $\epsilon = 1$ for longer, and then setting a decaying function for more episodes, could produce better policies, given there were still fewer explored areas present in the maze. Finally, performing further analysis on the different decaying functions, and having different ones for α and ϵ could lead to more insightful results, but for the purpose of this experiment, we investigated the effects of each hyperparameter in the algorithm and considered the different methods we could implement.

REFERENCES

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

[2] C.-S. Chow and J. N. Tsitsiklis, "The complexity of dynamic programming," *Journal of Complexity*, vol. 5, no. 4, pp. 466–488, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0885064X89900216

[3] J.-M. Réveillac, "4 - dynamic programming," in *Optimization Tools for Logistics*, J.-M. Réveillac, Ed. Elsevier, 2015, pp. 55–75. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9781785480492500049

[4] C. Watkins and P. Dayan, "Technical note: Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 05 1992.