Evaluating the performance of Reinforcement Learning algorithms in a delivery robot setting

1. Introduction

This report presents a comparative study of three reinforcement learning algorithms: **Value Iteration** (from Dynamic Programming), **On-policy Monte Carlo** (from Monte Carlo methods), and **Q-learning** (from Temporal Difference learning). The goal of the algorithms is to learn the optimal path to a given destination on a grid with obstacles. The performance of these algorithms is evaluated across a number of carefully designed experimental setups, each chosen to test the effect of one specific model parameter. Justifications for each setup are provided to ensure alignment with the evaluation goals. The metrics used are supported by visualizations to illustrate trends and outliers. We conclude the report by summarizing the strengths and limitations of each agent given the experimental results, thus providing insights into their practical applicability and performance trade-offs.

2. Algorithms

In this section, we briefly outline our three algorithms. Three parameters that are shared by all agents are the **discount factor** γ, determining how much the agent values future rewards compared to immediate rewards, the **environmental noise** σ, which is the probability that the environment ignores the agent's submitted move and takes a random move instead, and the **convergence threshold** δ, which determines when the algorithm has converged. For training, value iteration also requires a (max) **number of iterations**, and MC and Q-learning require both a (max) **number of episodes** and a **number of iterations per episode**.

2.1 Value Iteration

Value Iteration is a model-based Dynamic Programming method used to find the optimal policy (Value-Iteration, 9 Feb 2024). "Model-based" means that the method needs to know the full underlying Markov model, i.e. the states, the reward function, and especially the transition probabilities.

The algorithm starts by initializing the value function of each state s to $V(s) = 0$. It then performs a series of *sweeps*; in each sweep across all state-action pairs, we compute the action-value function $Q(s, a)$, which is the value of taking action a in state s:

$$Q(s,a) = \sum_{s'} P(s' \mid s,a) \left[r + \gamma V(s') \right]$$

where r is the reward received for taking action a in state s. After doing this for every state s and action a, we set the new value function $V(s)$ to be:

$$V(s) = \max_a Q(s, a)$$

The sweep over all states for updating $Q(s, a)$ and then $V(s)$ together form a single iteration; we run the algorithm for a fixed number of max iterations or until it has converged, whichever comes first. To check for convergence, we track how much $V(s)$ changed for each state during the iteration, and record the highest absolute difference in value across all states:

$$\Delta = \max_s |V_{\text{old}}(s) - V_{\text{new}}(s)|$$

If Δ is smaller than the convergence threshold δ, the algorithm has converged, and we stop prematurely. At the end of training, we extract our optimal policy from $V(s)$ as follows:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s' \mid s, a) \left[r + \gamma V(s')\right]$$

2.2 On-Policy Monte Carlo

On-policy Monte Carlo relies on complete episodes - sequences of states, actions, and rewards that end in a terminal state - and estimate value functions by averaging returns from multiple episodes (Trabelsi, 2023). The algorithm works by alternatively *evaluating* the current policy and *updating* the policy based on the evaluation results, executing both of these steps once for each episode of experience. Unlike Value Iteration, it does not require knowledge of the underlying model; it is a *model-free* algorithm.

In the policy evaluation step, we update the action-values $Q^\pi(s, a)$ of all state-action pairs (s, a) using the newly observed episode $s_0, a_0, r_0, s_1, a_1, r_1, ...$ under the current policy π; if we are in state $s = s_t$ and take action $a = a_t$, then we update

$$Q(s, a) = Q(s, a) + \alpha(G_t - Q(s, a))$$

at the end of the episode, where α is a step size parameter and G_t is the *return*, i.e. discounted total reward, of the episode from this timestep onwards:

$$G_t = r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t} r_T.$$

In our case, we use the *first-visit* approach, meaning we only update the value of $Q(s, a)$ for the *first* time t that we see (s, a) in the episode.

For policy improvement, the policy is updated using this new Q function. We use an ϵ-greedy policy to ensure exploration, so the new policy will be:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \frac{\epsilon}{|A|}, & \text{if } a = \arg\max_{a'} Q(s, a') \\ \frac{\epsilon}{|A|}, & \text{otherwise} \end{cases}$$

i.e. the best action is taken with probability $1 - \epsilon$ and a random action with probability ϵ. The algorithm converges to the optimal ϵ-soft policy under the condition that all state-action pairs are visited infinitely often, i.e. there is sufficient exploration. To check for convergence, we use a similar criterion as above; we stop when

$$\max_{s,a} |Q_{\text{old}}(s, a) - Q_{\text{new}}(s, a)| < \delta$$

at the end of an episode.

Besides the standard parameters γ, σ, δ, this algorithm also requires a **step size** α and an **exploration probability** ϵ. After training, we set ϵ to 0 so the agent only takes the best moves it knows during evaluation.

2.3 Q-Learning

Q-Learning is a model-free Temporal Difference (TD) learning algorithm used to find the optimal policy for an agent through interaction with the environment, by *sampling* episodes (Q-Learning, 25 Feb 2025). It is an *off-policy* algorithm since it interacts with the environment by using a certain policy (behavior policy) and then it updates the value estimates toward a *different* policy (target policy). In our case the target policy is the optimal policy that we are trying to learn. Like MC, the agent alternates between evaluation and improvement steps, but now these are executed after every *iteration* rather than every *episode*.

After executing an action using the behavior policy, the agent updates the Q-values for the current state-action similarly to MC, except instead of using the *actual* return G_t, it uses an *estimated* return $r + \gamma \max_{a'} Q(s', a')$:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right]$$

where we estimate the value based on the best-known action a' for the new state s'. Policy improvement works the same as MC, using the same ϵ-greedy policy. The convergence check also happens in the same way, and we again set ϵ to 0 for evaluation.

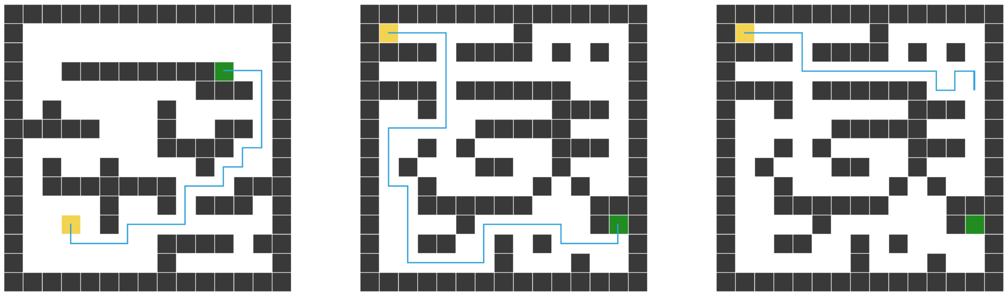
Like MC, this algorithm requires a step size α and an exploration probability ϵ to function.

3. Experiments

We now evaluate the effect of various parameters on the performance of our three agents. In each experiment, we vary 1 parameter, and keep the rest at some default values which are shared between all experiments. We then test all three agents for all different values of this parameter, and repeat this experiment on both the `A1_grid` provided by the course and our own `Maze` grid. The starting position and random seed are the same in all cases; random seed 42 and starting position $(3, 11)$ for the `A1_grid` and $(1, 1)$ for `Maze` (these positions are far away from their respective targets). Figure 1, contains a visual on what the two grids look like.

The performance of the agents is compared through two different metrics. One is the number of **iterations until the (action-)value function tracked by the agent has converged** (i.e. when the max-norm difference with the last iteration is $< 10^{-6}$); the lower this number, the faster the agent learns the optimal policy. Another is the **number of steps the agent goes over the optimal policy**; if the agent is taking more steps than the optimal policy, it has learned a suboptimal route, and the more steps it goes over, the worse it is. The optimal number of steps on the `A1_grid` is 22, while the optimal number of steps on `Maze` is 34.

The parameter values we test and their defaults are as follows:



(a) A1_grid (b) Our Maze Grid (c) An optimal path is not found for the maze grid

Figure 1: The two test environments with agents reaching the a goal as well as an example of the goal not being reached by the agent.

- Discount factor γ: $0.6, 0.9, 0.99$, default 0.99.
- Environmental noise σ: $0, 0.1, 0.5$, default 0.1.
- Number of iterations per episode (for TD & MC): $100, 5000, 20000$, default 20000.
- SGD step size α (for TD & MC): 0.2 constant, 0.5 constant, 0.2 with decay, 0.5 with decay, default 0.5 with decay.
- Exploration probability ϵ (for TD & MC): 0.2 constant, 0.5 constant, 0.2 with decay, 0.5 with decay, default 0.5 with decay.

Some remarks on this:

- For TD and MC, we always train for 3000 episodes; the idea is that if the agents haven't converged before then, they never will.
- The default number of iterations per episode needs to be high enough that even a random walk can reach the target, otherwise MC doesn't actually learn anything. The other two values are mostly to test how well Q-learning handles short episodes.
- For α and ϵ, we test both constant values and values that are decayed exponentially by a factor of 0.99 every episode, until a minimum value. For both MC and Q-Learning, the minimum value for ϵ is 0.01; for Q-learning, the min value for α is 10^{-2}, while for MC it is 10^{-8}. The decay begins after 1250 episodes; until then, the parameter value is kept identical to the initial value.

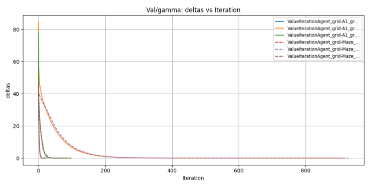
As mentioned, the five experiments above are repeated for two different grids, leading to a total of 10 different experimental setups.

3.1 Experimental Results

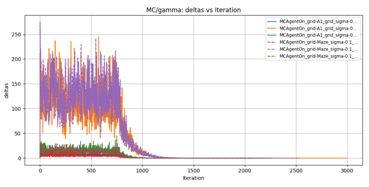
The full results of our experiments can be found in Table 1. For the γ and α experiments, we also provide plots in figures 2 and 3.

Grid		A1_Grid			Maze		
Agent		DP	MC	TD	DP	MC	TD
Defaults		913 / 0	2500 / *	173 / 0	929 / 0	2538 / *	182 / 0
gamma	0.6	25 / *	2080 / *	185 / 0	24 / *	2076 / *	124 / 0
	0.9	93 / 0	2195 / 0	203 / 0	100 / 0	2263 / *	186 / 0
sigma	0	888 / 0	2492 / *	204 / 0	894 / 0	2430 / *	225 / 0
	0.5	971 / 0	2509 / *	163 / 0	1035 / 0	2513 / *	121 / 0
iter/episode	100	-	2450 / *	270 / 0	-	2338 / *	344 / 0
	5000	-	2472 / *	205 / 0	-	3000 / *	194 / 0
α	0.2 const	-	3000 / *	547 / 0	-	3000 / *	433 / 0
	0.5 const	-	3000 / *	181 / 0	-	3000 / *	173 / 0
	0.2 decay	-	2194 / 6	450 / 0	-	2375 / *	476 / 0
ϵ	0.2 const	-	2482 / *	280 / 0	-	3000 / *	273 / 0
	0.5 const	-	2413 / *	183 / 0	-	2501 / 0	172 / 0
	0.2 decay	-	2400 / *	232 / 0	-	3000 / *	315 / 0

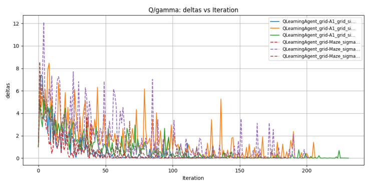
Table 1: Iterations (DP) or episodes (MC & TD) to reach convergence up to 6 digits / number of steps above optimal that agent's final learned policy takes, for each of the three agents on both `A1_grid` and `Maze`. In experiments marked with a *, the agent failed to reach the target.



(a) Value Iteration



(b) On-policy Monte Carlo

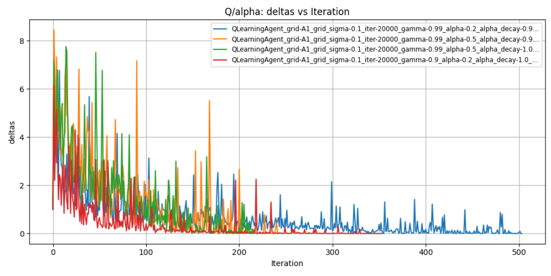


(c) Q-Learning

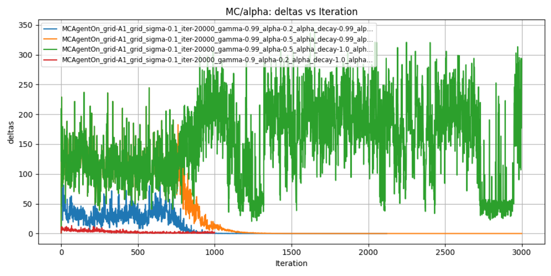
Figure 2: Delta (max-norm difference with previous iteration) for the three agents for different values of γ (color) on both the `A1` and `Maze` grids (solid/dashed). Once Δ gets small enough (i.e. at the end of the line), the agent has converged. We see that convergence is smooth for VI, noisy with a downward trend for Q-learning, and very noisy for MC (the convergence here is largely due to the decaying step size).

What immediately stands out is the poor performance of the MC agent. Even with the default settings (which were tuned on a few random seeds to produce a near-optimal policy), the agent doesn't consistently find the optimal path, and its performance rapidly drops further as we move away from the default parameter values, suggesting a high sensitivity to these parameters. It only converges if α and ϵ are both low and decayed; this suggest the default parameter values may have been poorly chosen, and required additional tuning.

The DP (Value Iteration) agent appears very sensitive to the discount factor, which makes sense; in each iteration, the agent effectively looks at the values "one step further ahead", and the higher γ, the more future steps actually matter. Interestingly enough, it



(a) Q-learning



(b) On-policy Monte Carlo

Figure 3: Deltas for the MC and Q-learning agents for different values of α, with and without decay. We see that the value of α and whether or not it decays has a significant effect on convergence, especially for MC.

does not find the optimal policy for a heavy discount of $\gamma = 0.6$; here the algorithm becomes too "short-sighted" to actually see and move towards the goal.

Finally, the TD (Q-learning) agent performs very well across the board. The parameter it appears most sensitive to is the step size α, but even that does not have too significant of an effect. What also stands out is that a higher environmental noise σ is actually *helping* the agent converge faster; this suggests the agent wasn't doing enough exploration on its own, and that a higher ϵ value may have been better. Regardless, the agent is robust against many different changes in environment and parameters; even the `Maze`, which was originally designed to be difficult for model-free methods, does not cause a performance drop.

4. Conclusions

In this report, we have conducted a comparative analysis of the Value Iteration, On-policy Monte Carlo, and Q-Learning algorithms for reinforcement learning in a grid-based delivery scenario, systematically varying a variety of parameters across two environments. Throughout our experiments, we found that Value Iteration delivers its best performance on both grids when the discount factor is high, but not too close to 1; from our experiments, $\gamma = 0.9$ appears to achieve a good balance between short- and long-term rewards. For the On-policy Monte Carlo agent, the performance is particularly dependent on the step size and exploration probability; the algorithm works best with small step sizes ($\alpha = 0.2$ with decay) and for high exploration ($\epsilon = 0.5$ constant). Finally, the Q-Learning agent performs best with a low discount factor and high exploration (either of its own or due to environmental noise).

Overall, the Q-Learning agent performed best in most scenarios due to its robust and complex solution. Given this and the fact that Monte Carlo performed much better with higher ϵ, future work should consider more complex algorithms that emphasize exploration to surpass the current benchmarks. This shortcoming is especially evident on the Maze grid.

References

Q-Learning. Q-learning in python, 25 Feb 2025. URL `https://www.geeksforgeeks.org/q-learning-in-python/`.

B. Trabelsi. On-policy vs off-policy monte carlo control methods for supply chain optimization: A use case of..., Oct 2023. URL `https://bechirtr97.medium.com/on-policy-vs-off-policy-monte-carlo-control-methods-for-supply-chain-optimization-a-use-`

Value-Iteration. What is the difference between value iteration and policy iteration?, 9 Feb 2024. URL `https://www.geeksforgeeks.org/what-is-the-difference-between-value-iteration-and-policy-iteration/`.