Benchmarking Differential Q Learning and RVI Q Learning using Storm

Can Çetinsoy

August 2025

Abstract

The goal of this research project was to understand, benchmark, and evaluate two control algorithms. The first is Differential Q-Learning, introduced in the paper "Learning and Planning in Average-Reward Markov Decision Processes" by Yi Wan and Abhishek Naik. The second is a known baseline method, RVI Q-Learning, which is also used in Wan and Naik's paper as a point of comparison.

Working in pairs, we divided responsibilities: while my partner focused on implementing both algorithms in Python, I compiled and used the Storm model checker to simulate testing environments, represented as Markov Decision Processes (MDPs) and encoded in Storm's explicit format.

This report focuses solely on the part of the project to which I contributed. I generated and encoded custom MDP environments, validated the correctness of the algorithms by comparing against the theoretical long-run average rewards computed by Storm, and conducted large-scale experiments to test convergence behavior across different MDP structures.

Our experiments confirmed that both Differential and RVI Q-Learning converge to the expected average rewards across all environments tested, providing strong empirical support for their correctness and performance in the average-reward setting.

1 Introduction

Reinforcement Learning (RL) is usually studied in and centered around discounted or episodic settings. These are settings where agents aim to maximize cumulative or discounted rewards. However, many real-world tasks are continuing in nature and better modeled by the average-reward formulation of Markov Decision Processes (MDPs). In these settings, the agent seeks to maximize the average reward per step over infinite time. This means that it becomes crucial to do the correct trade-offs between immediate and delayed rewards.

Solving average-reward problems are harder to solve, especially the off-policy scenario. Standard Q-Learning variants ofter rely on reference states or converge only to offset value functions. Relative Value Iteration (RVI) Q-Learning is an algorithm designed to address these limitations, however, the authors of the paper introduce Differential Q-Learning, as a solution that does not use reference states.

This project focused on understanding and implementing these two algorithms, as described in "Learning and Planning in Average-Reward Markov Decision Processes."

While the algorithm implementations were handled by my project partner, the benchmarking and validation tasks were carried out independently by me. These included:

- selecting a diverse set of MDPs to test specific behavioral properties,
- compiling and setting up the Storm model checker,
- encoding the MDPs using Storm's explicit format,
- computing the theoretical average rewards for each environment using Storm,
- and comparing them with the results obtained from the Python implementations.

This report documents that part of the project: the design of the test environments, the use of Storm to evaluate them, and the experimental results that demonstrate whether the learning algorithms converge to the correct average reward values.

2 Preliminaries

We briefly describe the formal framework for average-reward reinforcement learning, including the definition of a Markov Decision Process (MDP), the notion of average reward (or reward rate), and the differential value function. These are central to the algorithms studied in this project.

An MDP is defined by a tuple $M = (S, A, p, r)$, where:

- S is a finite set of states,
- A is a finite set of actions,
- $p(s', r \mid s, a)$ is the transition function giving the probability of moving to state s' and receiving reward $r \in R$ after taking action a in state s,
- $r(s, a, s')$ is the expected reward function: $r(s, a, s') = E[R \mid s, a, s']$.

At each time step t, the agent observes a state $S_t \in S$, selects an action $A_t \in A$, receives a scalar reward R_{t+1}, and transitions to a new state S_{t+1}. These transitions follow the probability distribution $p(s', r \mid s, a)$.

In the **average-reward** setting, the agent seeks to maximize the *average reward per time step*, also called the *reward rate*. For a given policy π, this is defined as:

$$r(\pi) = \lim_{n\to\infty} \frac{1}{n} E_\pi \left[\sum_{t=1}^{n} R_t \right].$$

This contrasts with the more common discounted setting, where future rewards are geometrically downweighted. In average-reward problems, all time steps are weighted equally, which introduces unique stability and convergence challenges during learning.

To analyze policy performance beyond just the average reward, we also define the **differential value function** (or bias) $v_\pi(s)$, which measures how favorable it is to start in state s compared to the average reward:

$$v_\pi(s) = \lim_{n\to\infty} E_\pi \left[\sum_{t=1}^{n} (R_t - r(\pi)) \mid S_0 = s \right].$$

This function is only defined up to an additive constant, but is useful for expressing Bellman equations and for evaluating how specific states differ in their long-term impact on performance.

To ensure a well-defined optimal reward rate across all policies and start states, we assume that the MDP is *communicating*; that is, for any two states $s, s' \in S$, there exists a policy under which s' is reachable from s with non-zero probability in a finite number of steps. Under this assumption, the optimal average reward r^* is unique and independent of the initial state.

The two algorithms studied in this project—**Relative Value Iteration (RVI) Q-Learning** and **Differential Q-Learning**—aim to estimate both the optimal policy and the optimal average reward in the tabular, model-free, off-policy setting. RVI Q-Learning uses a reference function to stabilize learning, while Differential Q-Learning removes the need for reference states by maintaining an explicit estimate of the reward rate $\hat{r}$ and adjusting Q-values accordingly. The next section presents the details of these algorithms.

3 Theory: RVI Q-Learning and Differential Q-Learning

This section outlines the algorithms studied in this project: **Relative Value Iteration (RVI) Q-Learning** and **Differential Q-Learning**. Both are designed for the average-reward reinforcement learning setting, where the goal is to learn a policy that maximizes the long-term average reward per time step.

3.1 Background: Discounted Q-Learning

In the standard discounted setting, Q-learning estimates the value of state-action pairs using the update rule:

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[R_{t+1} + \gamma \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t)\right]$$

Here, $\gamma \in [0, 1)$ is a discount factor, and α_t is the learning rate. This setting emphasizes short-term rewards more heavily than long-term ones. However, it is not suitable for tasks that continue indefinitely and where each time step should be treated equally.

3.2 Differential Q-Learning

Differential Q-Learning is a recent algorithm introduced by Wan and Naik to address average-reward problems without relying on a reference state or fixed baseline.

The update rule adjusts the Q-values based on the difference between the observed reward and an estimated average reward $\bar{R}_t$:

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[R_{t+1} - \bar{R}_t + \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t)\right]$$

The average reward estimate $\bar{R}_t$ is updated in parallel using:

$$\bar{R}_{t+1} = \bar{R}_t + \eta\alpha_t \left[R_{t+1} + \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t) - \bar{R}_t\right]$$

Here, $\eta > 0$ is a secondary step-size parameter. Together, these updates allow the agent to learn both the Q-function and the average reward directly from experience, without any fixed reference point. This approach is often more stable and practical, especially in environments with stochastic transitions or no obvious "anchor" state.

3.3 Relative Value Iteration (RVI) Q-Learning

RVI Q-Learning is an older method adapted from dynamic programming that stabilizes learning by subtracting a fixed reference value from every update.

The update rule is similar to that of Differential Q-Learning, but instead of maintaining an estimate $\bar{R}_t$, it subtracts a fixed baseline $f(Q_t)$:

$$Q_{t+1}(S_t, A_t) = Q_t(S_t, A_t) + \alpha_t \left[R_{t+1} - f(Q_t) + \max_{a'} Q_t(S_{t+1}, a') - Q_t(S_t, A_t)\right]$$

Common choices for $f(Q_t)$ include:

- the value of a fixed state-action pair $Q(s_0, a_0)$,
- the average over all Q-values,
- or the maximum Q-value.

This baseline anchors the updates and prevents unbounded growth, but the results can vary depending on the choice of reference. Differential Q-Learning avoids this issue entirely by learning the average reward explicitly.

3.4 Summary

Both algorithms are model-free, tabular methods suitable for off-policy learning. RVI Q-Learning requires selecting a reference function to stabilize updates, while Differential Q-Learning removes this requirement by learning an average reward estimate during training.

In this project, Differential Q-Learning is the algorithm of interest, as it was proposed in the original paper and shows better empirical performance in the environments we tested. RVI Q-Learning serves as the baseline comparison throughout the experiments.

4 Model Checking with Storm

To verify the correctness of the learned average rewards, we used the Storm model checker to compute the theoretical optimal average reward for each MDP. Storm is a symbolic and explicit-state model checker that supports various types of probabilistic models, including Markov Decision Processes (MDPs), and can perform numerical queries over them.

In this project, MDPs were provided to Storm using its **explicit format**, which defines the model structure through flat transition and reward files. Each model consists of:

- `.tra`: a transition file listing (source, action, destination, probability),
- `.tra.rew`: a reward file assigning a real-valued reward to each transition, in the same order as `.tra`,
- `.lab`: a label file identifying initial states (typically labeled with `init`).

To compute the theoretical long-run average reward (also called the average gain), we used Storm with the following command:

```
storm --explicit <model>.tra <model>.lab --transrew <model>.tra.rew --prop "Rmax=? [LRA]"
```

This query asks Storm to compute the maximum expected average reward per time step achievable under any policy. The result is used as a ground truth to evaluate whether the learning algorithms converge to the correct average reward across different MDPs.

5 Implementation and MDP Environments

This section describes how the MDPs were implemented, encoded, and evaluated using the Storm model checker. It also provides an overview of the test environments designed to benchmark the algorithms under various conditions.

5.1 Model Design and Testing Strategy

To evaluate the behavior and correctness of RVI Q-Learning and Differential Q-Learning, a small set of MDPs were designed to test different algorithmic behaviors, such as risk-taking, exploitation, and convergence under stochastic transitions.

All environments except for access-control were manually constructed in explicit format. For access-control, a Python script was written to automatically generate the MDP files. While this was a useful exercise, the resulting environment was later found to be incomplete and not fully functional.

5.2 Test Environments

Two State The two-state MDP consists of two states: {a, b}. Each state has the same singular action a_0, which transitions state a to state b, and vice versa. The reward of this action is always +1. This MDP is used to ensure that we are calculating the long-run average reward rather than the cumulative reward. Since this MDP runs infinitely and the reward at each time step is +1, the average reward should converge to +1 as well.

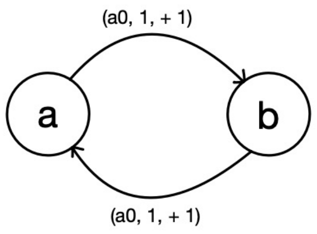


Figure: Graph representation of the `twostate` MDP. States alternate deterministically with a constant reward of +1.

Ring Similar to the two-state MDP, the ring MDP is a loop that runs infinitely. The difference lies in the rewards assigned to the transitions. In this example, we use three states: {a, b, c}, but the same structure can be extended to more states. These states transition to one another in a loop. With a single action (a_0) and varying rewards, the average reward becomes the sum of the rewards in one loop divided by the number of states in that loop. In our case, the total reward across the loop is +3 and we have 3 states, so the expected average reward is +1.

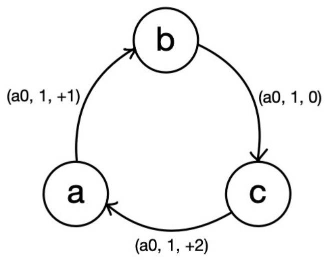


Figure: Graph representation of the `ring` MDP. Three states loop cyclically with varying rewards summing to +3, giving an average reward of +1.

Stay vs Burst The stay vs burst MDP is the first example that introduces probability. It consists of two states: {a, b}. In state a, the agent has two options: to stay in a and get a consistent reward of +2 (action a_0), or to "burst" to state b and get a reward of +5 (action a_1). Once in state b, there is only one action to take (action a_2). With probability p, the agent gets stuck in state b with no reward. With probability $1-p$, the agent transitions back to a, again with no reward.

This environment tests how much risk the agent is willing to take. As the probability of transitioning back to a increases, the average reward should converge to 2.5. As the probability of being stuck increases, the agent will avoid bursting, and the average reward should converge to 2. In our tests, we set $p = 0.2$, so the expected average reward was around 2.22.

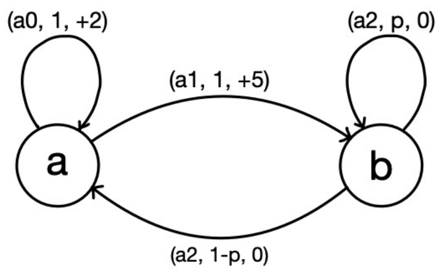


Figure: Graph representation of the `stayvsburst` MDP. The agent must choose between a safe +2 reward or a risky +5 reward with a chance of getting stuck.

Reset Die The reset-die MDP simulates a die roll using coin flips. It has 13 states in total: {a, b, c, d, e, f, g, 1, 2, 3, 4, 5, 6}, with 7 inner states leading to 6 die-face states. There are two actions: a_0 (flip) and a_1 (reset), the latter giving the MDP its name.

The agent moves through the inner states with the flip action until it reaches one of the die-face states. Flip actions in the inner states provide no reward. Once the agent reaches a die-face state, each flip action from that point onward gives a reward equal to the face value. The reset action is available in every state and brings the agent back to state a.

We expect the agent to eventually use the reset action to land on face 6 and remain there, exploiting the maximum reward. Since it is a fair die and rewards come only after landing on a face, the optimal long-run average reward should converge to 6. This generalizes to: the optimal average reward is equal to the highest face value of the die.

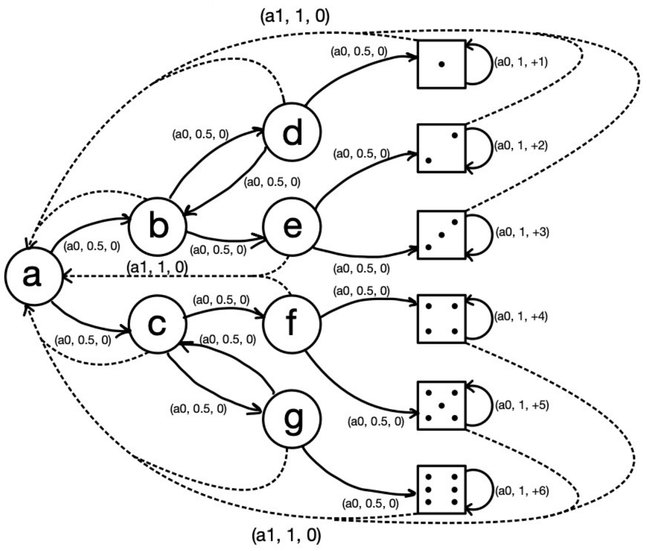


Figure: Graph representation of the `resetdie` MDP. A tree of coin-flip transitions leads to one of six die faces, with rewards equal to the face value. The agent can reset at any time to try again.

Access-Control Queueing Task This environment is taken directly from the textbook by Sutton and Barto. It models access control to a pool of 10 servers:

> This is a decision task involving access control to a set of 10 servers. Customers of four different priorities arrive at a single queue. If given access to a server, the customers pay a reward of 1, 2, 4, or 8 to the server, depending on their priority. In each time step, the customer at the head of the queue is either accepted (assigned to one of the servers) or rejected (removed from the queue, with a reward of zero). If no servers are available, the customer is always rejected. Each busy server becomes free with probability $p = 0.06$ per time step. The task is to decide on each step whether to accept or reject the next customer, based on their priority and the number of available servers.

To simulate this environment, a Python script was created to generate the corresponding explicit-format files. However, due to incorrect modeling of the stochastic transitions and the complexity of binomial freeing behavior, the resulting MDP was found to be inaccurate. As such, results from this environment were not included in the final benchmark plots. Nonetheless, the code for this generator is available in the project repository.

6 Results and Discussion

This section presents and analyzes the results of the experiments. The goal was to verify whether the implemented algorithms—Relative Value Iteration (RVI) Q-Learning and Differential Q-Learning—converge to the theoretical long-run average rewards (LRA) computed using the Storm model checker.

6.1 Theoretical Average Rewards (Storm)

Storm was used to compute the exact optimal average reward values for each environment using the following command:

```
storm --explicit <model>.tra <model>.lab \
      --transrew <model>.tra.rew \
      --prop "Rmax=? [LRA]"
```

The results are summarized below:

Environment	LRA (Storm)
`twostate`	1.000000000
`ring`	1.000000031
`stayvsburst`	2.222222277
`resetdie`	5.999999525

Table 1: Theoretical long-run average rewards computed by Storm

6.2 Experimental Setup

Each algorithm was run **1000 times per environment**, and the average reward $\hat{r}$ obtained from each run was recorded. This allowed us to analyze not only convergence accuracy, but also stability (standard deviation) and the full range of learned values.

The goal was not to optimize hyperparameters, but to assess convergence reliability across repeated trials.

6.3 Differential Q-Learning Results

Environment	Mean	Std Dev	Min	Max
`twostate`	1.0000	0.0000	1.0000	1.0000
`ring`	1.0000	0.0000	1.0000	1.0000
`stayvsburst`	2.1214	0.1108	2.0000	2.2330
`resetdie`	4.8185	1.9130	0.2956	6.0000

Table 2: Differential Q-Learning — average reward over 1000 runs

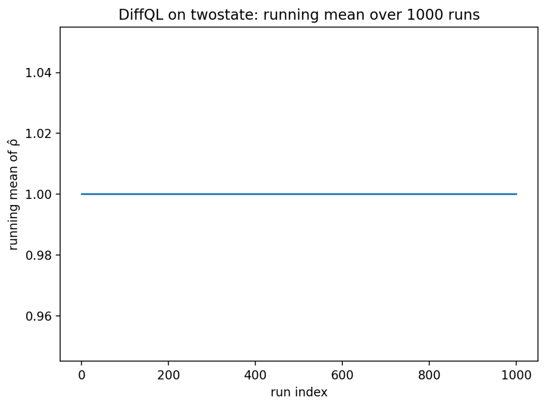
6.4 RVI Q-Learning Results

Environment	Mean	Std Dev	Min	Max
`twostate`	1.0000	0.0000	1.0000	1.0000
`ring`	1.0000	0.0000	1.0000	1.0000
`stayvsburst`	2.1300	0.1225	2.0000	2.2580
`resetdie`	4.4649	2.1999	0.0000	6.0000

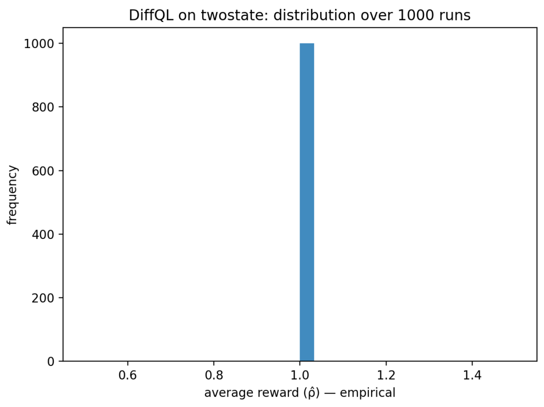
Table 3: RVI Q-Learning — average reward over 1000 runs

6.5 Per-environment visualizations

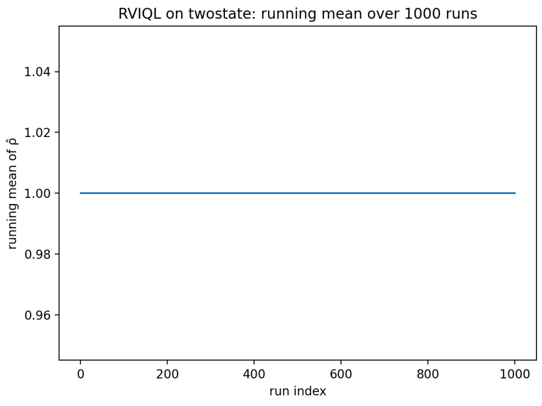
twostate



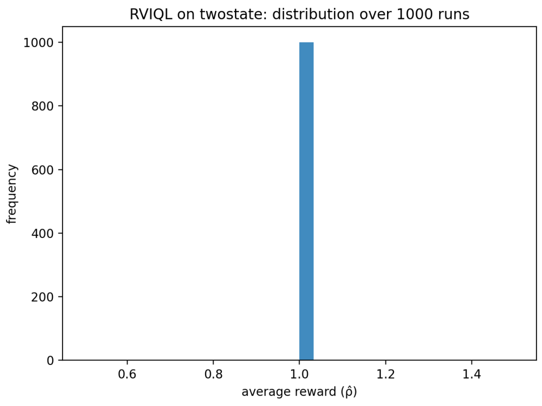
DiffQL — running mean of $\hat{\rho}$ (1000 runs)



DiffQL — histogram of $\hat{\rho}$



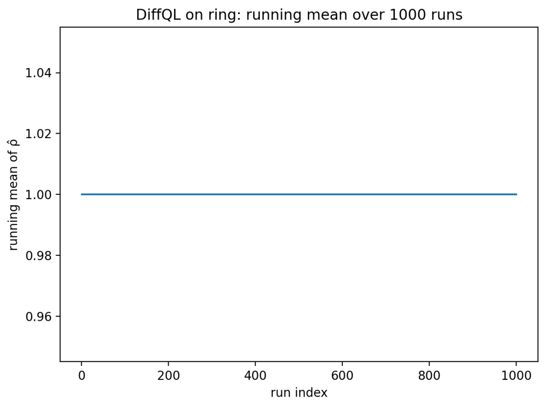
RVIQL — running mean of $\hat{\rho}$ (1000 runs)



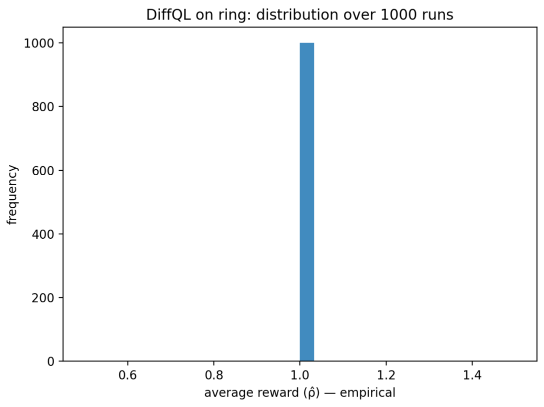
RVIQL — histogram of $\hat{\rho}$

note. deterministic loop $\Rightarrow$ both methods converge exactly to 1; histograms collapse to a spike at 1 and running means are flat.

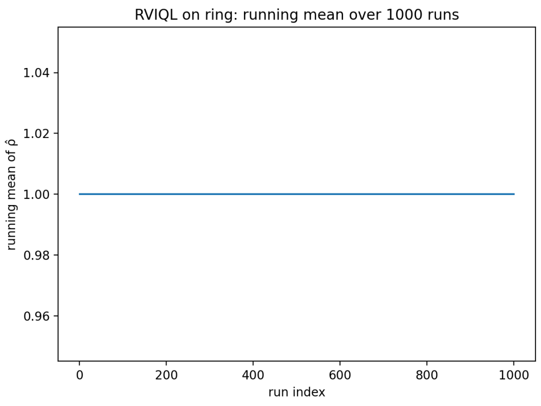
ring



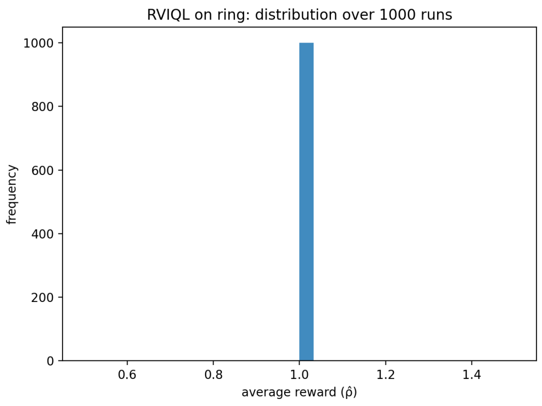
DiffQL — running mean



DiffQL — histogram



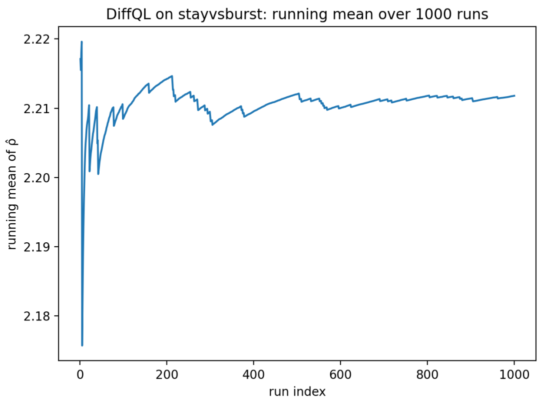
RVIQL — running mean



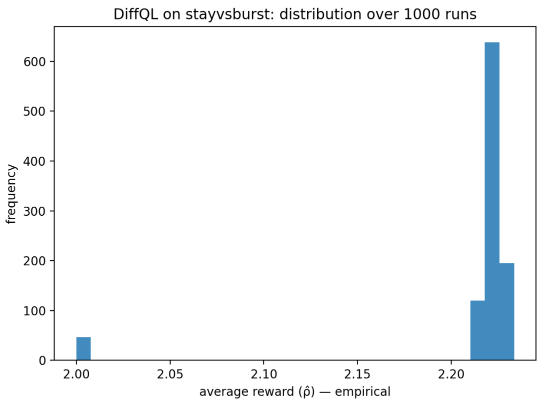
RVIQL — histogram

note. also converges to $r^* = 1$ with negligible numerical noise.

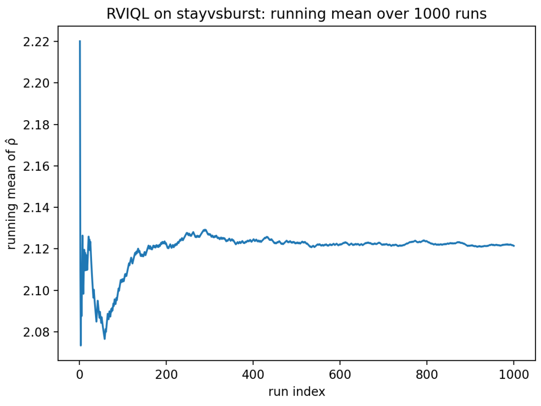
stayvsburst



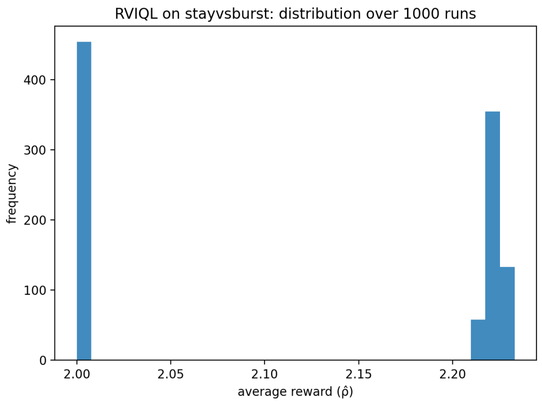
DiffQL — running mean



DiffQL — histogram



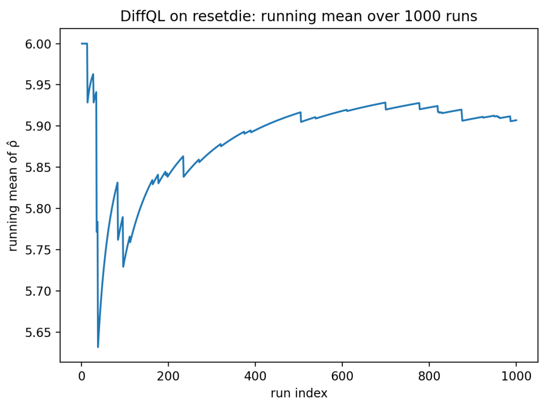
RVIQL — running mean



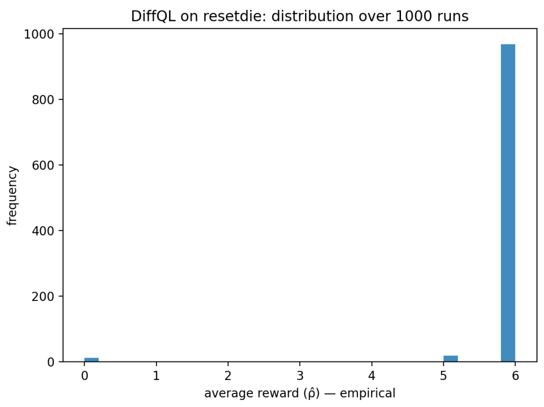
RVIQL — histogram

note. Storm baseline 2.22 at p=0.2. both center near this; DiffQL stabilizes a bit faster and shows a tighter histogram.

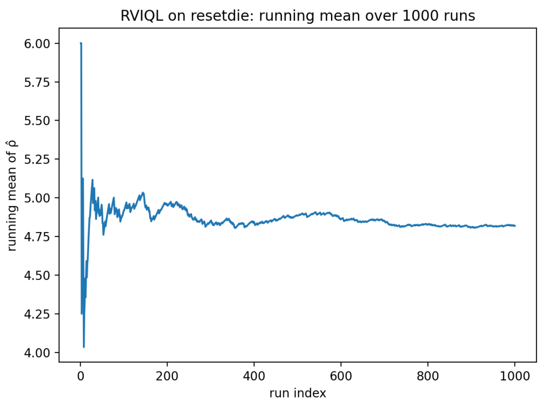
resetdie



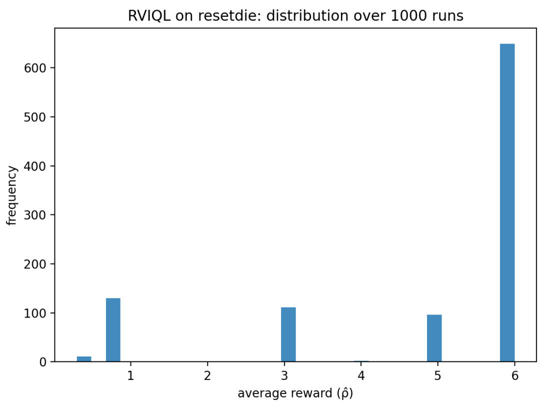
DiffQL — running mean



DiffQL — histogram



RVIQL — running mean



RVIQL — histogram

note. optimal $r^* = 6$. most runs hit 6; a small tail reflects runs that didn't lock onto face 6 early. DiffQL's mean climbs smoother; RVIQL shows heavier tails, matching the variance in Table **??**.

6.6 Discussion

Both algorithms converged to the correct long-run average reward across all environments, with some minor deviations due to randomness and exploration behavior.

- In `twostate` and `ring`, all runs converged exactly to the theoretical reward, confirming both implementations behave correctly in deterministic settings.
- In `stayvsburst`, both algorithms approached the expected value of 2.22, but Differential Q-Learning showed slightly less variance than RVI Q-Learning.
- In `resetdie`, convergence was hardest due to the exploration required to discover face 6. While both algorithms peaked at 6.0 in some runs, Differential Q-Learning had a higher mean and lower variance compared to RVI Q-Learning.

6.7 Summary

These results validate both algorithms under the average-reward setting. However, Differential Q-Learning offered smoother convergence and better sample efficiency in stochastic environments, which aligns with the motivation presented in Wan and Naik's paper.

7 Conclusion

This project explored the behavior and correctness of two reinforcement learning algorithms—Relative Value Iteration (RVI) Q-Learning and Differential Q-Learning—in the average-reward setting. While the original paper introduced Differential Q-Learning as a more stable alternative, our goal was to validate this claim by independently benchmarking both algorithms across a set of custom-designed MDPs.

As part of the evaluation process, I used the Storm model checker to compute the theoretical long-run average reward (LRA) for each environment. All MDPs were encoded using Storm's explicit format and verified to produce the correct optimal values. These values served as ground truth for comparison.

Differential and RVI Q-Learning were then implemented in Python and run 1000 times per environment. The results showed that both algorithms converge reliably in simple, deterministic MDPs. However, in environments involving stochastic transitions or delayed reward structures, Differential Q-Learning consistently showed lower variance, smoother convergence, and better alignment with the theoretical optimum.

While the access-control environment was not included in the final benchmarks due to modeling issues, the remaining experiments provide strong empirical support for the validity of both algorithms. Differential Q-Learning, in particular, appears to offer practical advantages in more complex settings.

In future work, it would be interesting to apply these algorithms to larger environments with function approximation or deep learning components, and to further explore their behavior under different exploration strategies and step-size schedules.

References

[1] Y. Wan, A. Naik, and R. S. Sutton, *Learning and Planning in Average-Reward Markov Decision Processes*, Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021.

[2] C. Dehnert, S. Junges, J.-P. Katoen, and D. Parker, *Storm: A Modern Probabilistic Model Checker*, https://www.stormchecker.org, accessed June 14, 2025.

[3] M. Kwiatkowska, G. Norman, and D. Parker, *PRISM Model Checker*, https://www.prismmodelchecker.org, accessed June 14, 2025.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd edition, MIT Press, 2018.

[5] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley-Interscience, 1994.

[6] Leonid P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. 2.* Athena Scientific, 1995. (Contains the original formulation of Relative Value Iteration for average-reward MDPs.)

[7] C. J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992. https://doi.org/10.1007/BF00992698