Evaluating Reinforcement Learning Algorithms in Navigational Control: A Comparative Study on Racetrack Performance

Srikanta Mehta SMEHTA44@JH.EDU
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21205, USA

Abstract

This study investigates the performance of three prominent reinforcement learning algorithms—Value Iteration, Q-Learning, and SARSA—across different racetrack configurations to test specific hypotheses about their efficiency in a simulated time-trial racing environment. Our primary hypothesis suggestsd that Value Iteration, due to its model-based approach that computes an optimal policy directly, will outperform Q-Learning and SARSA in terms of fewer average steps required to complete the racetracks. The secondary hypothesis suggests that Q-Learning will generally perform better than SARSA, as its off-policy nature allows for more robust learning under varying conditions of track complexity and crash penalties. Conducted on multiple tracks with distinct layouts and crash penalties, the experiments demonstrate that Value Iteration significantly outperforms the other two algorithms, consistently requiring fewer steps to reach the finish line, with statistical analysis confirming the significance of these results. The comparison between Q-Learning and SARSA reveals a less consistent pattern, indicating that the choice between these two algorithms should be influenced by specific track characteristics and hyperparameter settings. This paper highlights the critical role of algorithm selection in autonomous navigation tasks and discusses the implications of these findings for optimizing reinforcement learning strategies in real-world applications.

Keywords: Reinforcement Learning, Value Iteration, Q-Learning, SARSA, Racetrack Navigation

1. Introduction

In the realm of reinforcement learning, the application of various algorithms plays a pivotal role in optimizing agent behavior across different controlled environments. Prominent among these algorithms are Value Iteration, Q-Learning, and SARSA, which offer distinct approaches to learning optimal policies. Value Iteration is a model-based method that directly computes the optimal policy by fully understanding the environment dynamics. Whereas Q-Learning and SARSA are model-free methods that learn policies based on sampled experiences and adjust their strategies incrementally, without requiring a model of the environment.

The theoretical foundations of these algorithms can be traced back to the early development of dynamic programming principles by (Bellman, 1957) and their subsequent adaptation into the reinforcement learning framework by (Sutton, 1988). These algorithms have since been applied in diverse scenarios, ranging from simple pathfinding tasks to complex decision-making environments in real-world applications.

This paper explores the hypothesis that Value Iteration, with its ability to compute policies directly from a complete model, will consistently outperform Q-Learning and SARSA in

terms of the efficiency measured by the average number of steps required to complete various racetrack configurations. Additionally, we hypothesize that Q-Learning, due to its off-policy learning capabilities, will generally outperform SARSA, especially in environments where the learning conditions vary significantly, such as different track complexities and crash handling scenarios.

To examine these hypotheses, the structure of this report is organized as follows: Section 2, *Algorithms and Experimental Methods*, details the implementation of Value Iteration, Q-Learning, and SARSA, alongside the simulated racetrack environment used to test these algorithms. Section 3, *Results*, presents the experimental data for comparison of the algorithms across different racetracks, utilizing the number of steps to finish and statistical analyses to validate differences in performance. Section 4, *Discussion*, delves into the experimental results of the algorithms' performance and discusses the impacts of these findings. The report concludes in Section 5 with a summary of conclusions and proposals for further research.

2. Algorithms and Experimental Methods

The objective of this project is to evaluate the performance of three well-known reinforcement learning algorithms—Value Iteration, Q-Learning, and SARSA—across different racetrack configurations. This study aims to optimize the agents navigation from the starting line to the finish line, minimizing the steps taken and adapting to various track complexities and crash penalties. This section provides a detailed description of the racetrack environments, the specifics of each algorithm, and the experimental approach used to investigate the research hypothesis.

2.1 Racetracks and Configuration

The four racetracks used in this study provide distinct layouts and complexity, each designed to test the algorithms' navigation abilities under controlled conditions. These tracks are represented as grids in text files, where different symbols denote track features:

- **Starting Points (S)**: Marked in the files as 'S', these are the positions where the race car can start.
- **Finish Lines (F)**: Marked as 'F', these represent the targets the race car needs to reach.
- **Open Racetrack (.)**: These are spaces where the car can move freely.
- **Walls (#)**: These indicate barriers that the car cannot cross and will result in a crash if contacted.

The four tracks featured in the experiments are designed to increase in complexity from basic layouts with minimal turns to more complex designs requiring advanced navigational strategies:

1. **L-Track**: Features a simple elongated 'L' shape, suitable for testing basic pathfinding and speed control.

2. **O-Track**: A circular or oval track testing continuous steering adjustments and consistent speed management.

3. **W-Track**: A complex path with multiple sharp turns, simulating challenging driving conditions that require rapid decision-making.

4. **R-Track**: The most diverse track, including various route options and sharp turns combined with long straights, offering scenarios with multiple strategic decisions.

2.2 Preprocessing and Experimental Setup

The racetrack files undergo preprocessing to convert them from ASCII representations into structured formats suitable for simulation. This involves parsing the track files to identify and categorize each cell according to its function—start, finish, open, or wall. This standardized preprocessing ensures that each algorithm encounters the environment in a consistent manner.

2.2.1 Experimental Parameters

The algorithms are tested in a controlled simulation environment where each timestep allows the car to change its velocity based on the chosen action. The actions available are constrained to accelerations in both the x and y directions with values {-1, 0, 1}, while keeping the velocity within the limits {-5, 5} per axis. Crashes and their consequences are handled according to the specific rules detailed for each track, influencing the strategic decisions made by the algorithms during the race.

This structured experimental approach enables a detailed assessment of each algorithm's ability to learn and apply effective strategies under varying conditions, directly addressing the project's goals of minimizing steps to the finish and adapting to environmental challenges.

2.3 Algorithms

This section outlines the simulation model used to evaluate the reinforcement learning algorithms—Value Iteration, Q-Learning, and SARSA—in their ability to navigate through different racetrack configurations efficiently.

2.3.1 Simulation Model and Dynamics

The simulation models the racetrack and the race car's movements through discrete states, focusing on the dynamic interactions between the car's actions and the racetrack's features.

Model Representation The model is characterized by:

- **State** (**S**): Defined by the tuple (x, y, vx, vy), representing the car's position (x, y) and velocity (vx, vy).

- **Actions** (**A**): Comprised of acceleration changes (ax, ay), which adjust the car's velocity within defined limits.

State Transitions and Dynamics Velocity and position updates are governed by the following equations, assuming no stochastic disruptions:

$$vx' = \max(\min(vx + ax, 5), -5), \quad vy' = \max(\min(vy + ay, 5), -5)$$

$$x' = x + vx', \quad y' = y + vy'$$

A 20% chance exists where the intended acceleration does not affect the car's velocity.

Collision Detection Using Bresenham's Algorithm The Bresenham's line algorithm is used for determining if the car's path intersects with any obstacles (walls). This algorithm is used whenever the car moves from one point to another, ensuring the simulation accurately detects and handles collisions.

The Bresenham's line algorithm calculates the points of a line between two coordinates using integer calculations. The equation used to determine the line points is given by:

$$\text{error} = 2\Delta y - \Delta x$$

where Δx and Δy are the differences in the x and y coordinates of the line's endpoints. The algorithm adjusts the error term as it increments the coordinates, ensuring it selects the nearest pixel at each step:

$$\text{if error} \geq 0 : \begin{cases} y = y + 1, \\ \text{error} = \text{error} - 2\Delta x \end{cases}$$

$$x = x + 1, \quad \text{error} = \text{error} + 2\Delta y$$

This method is highly efficient for checking collision paths on the grid-based racetrack, avoiding floating-point calculations.

Rewards and Objectives The simulation's reward structure is simple and designed to encourage reaching the finish line with the fewest steps:

$$R(s, a) = \begin{cases} 0 & \text{if } (x', y') \text{ is a finish line} \\ -1 & \text{otherwise} \end{cases}$$

The objective is to minimize the total accumulated penalty, guiding the learning algorithms toward the most efficient paths through the racetrack.

2.3.2 Implementing Value Iteration

Value Iteration is a foundational reinforcement learning algorithm used to compute the optimal policy by systematically updating the expected utilities of actions at each state. This algorithm is model-based and assumes complete knowledge of the environment's dynamics.

Algorithm Initialization The Value Iteration algorithm is initialized with:

- **Simulation Environment** (**Simulator**): This encapsulates the racetrack, including its states and the transitions possible from each state due to actions.
- **Discount Factor** (γ): This parameter quantifies the importance of future rewards. A higher discount factor places more significance on future outcomes.
- **Convergence Threshold** (θ): This value determines when the algorithm should stop iterating, based on the changes in the value function being sufficiently small.

Value Function and Policy Initialization Upon initialization, the value function $V(s)$ for each state s is set to zero. This function estimates the maximum expected utility starting from state s under any policy. The policy $\pi(s)$, which suggests the best action to take from each state, is also initialized but not assigned any actions initially.

Iterative Update Procedure The Value Iteration algorithm involves updating the value function iteratively using the Bellman Optimality Equation:

$$V_{k+1}(s) = \max_a \left\{ \sum_{s'} P(s' \mid s, a)[R(s, a, s') + \gamma V_k(s')] \right\}$$

where:

- $P(s' \mid s, a)$ is the probability of transitioning to state s' from state s under action a.
- $R(s, a, s')$ is the reward received after transitioning from s to s', due to action a.
- $V_k(s')$ is the value of state s' at iteration k.

Convergence and Policy Extraction The iteration continues until the maximum change in the value function across all states between two successive iterations is less than the threshold θ. At this point, the algorithm converges, and the optimal policy is extracted by choosing the action that maximizes the expected utility at each state:

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

Performance Metrics The algorithm also tracks the number of iterations it takes to converge, providing insights into the computational efficiency and complexity of finding an optimal policy for different racetrack configurations.

This approach to calculating the optimal path through iterative updates and systematic evaluation of possible actions ensures that Value Iteration can effectively optimize the race car's navigation strategy, ideally minimizing the time and moves required to reach the finish line.

2.3.3 Implementing Q-Learning and SARSA

Q-Learning and SARSA are reinforcement learning algorithms known for their model-free approach, where they learn optimal policies based on experiences without a model of the environment. These algorithms are particularly suitable for the racetrack navigation problem where precise modeling of all environment dynamics might be infeasible.

Q-Learning Q-Learning is an off-policy learner which means it learns the value of the optimal policy independently of the agent's actions. It updates its value estimates based on the equation:

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

where:

- s, a are the current state and action,
- s' is the new state after taking action a,
- r is the reward received after taking action a at state s,
- α is the learning rate,
- γ is the discount factor, and
- $\max_{a'} Q(s',a')$ is the maximum predicted reward achievable in the new state s'.

The algorithm chooses actions based on an ϵ-greedy policy, balancing exploration and exploitation by selecting random actions with a probability of ϵ, and the best-known action with a probability of $1 - \epsilon$.

SARSA SARSA, is an on-policy learner which updates its policy and Q-values based on the action actually taken, as opposed to the optimal action:

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

where a' is the action taken in the new state s'. Like Q-Learning, SARSA uses an ϵ-greedy policy for action selection, adjusting ϵ dynamically based on the episode number to reduce exploration as the agent becomes more confident in its policy.

Training Procedure Both algorithms follow a similar training procedure:

1. Initialize Q-values arbitrarily for all state-action pairs.
2. For each episode:
   (a) Initialize the state.
   (b) Choose an action based on the current policy (either ϵ-greedy for both or another policy depending on the algorithm specifics).
   (c) Take the action, observe the reward and the next state.

(d) Update the Q-values using the Q-Learning or SARSA update rule.

(e) Move to the next state and repeat the process until a terminal state is reached or a maximum number of steps is exceeded.

3. Repeat for the desired number of episodes.

Policy Extraction After sufficient training, the policy is derived from the Q-values:

$$\pi(s) = \arg\max_a Q(s, a),$$

where $\pi(s)$ is the optimal action to take from state s. This policy is expected to guide the agent to navigate the racetrack efficiently by minimizing the number of steps to reach the goal.

2.4 Experiment Design

The experimental design aimed at evaluating the performance of the reinforcement learning algorithms: Value Iteration, Q-Learning, and SARSA, across different racetrack configurations. The main objective was to identify the optimal hyperparameter settings for each algorithm to ensure peak performance.

2.4.1 Hyperparameter Tuning

Hyperparameter tuning was performed using a grid search approach, where a predefined set of values for learning rates, discount factors, and exploration rates were systematically tested. The set of parameters tested where initially determined by manual testing runs with the aim of finding a range of suitable parameters. For Q-Learning and SARSA, the number of episodes was set to 2000 through initial testing, which ensured that the learning curves converged within that amount. The specific parameters tuned were:

- **Learning Rate Variations:** Values tested were $[0.1, 0.2, 0.3]$. This parameter influences the rate at which each algorithm updates the value function estimates, impacting the convergence speed and stability of learning.
- **Discount Factor Variations:** Values tested were $[0.9, 0.95, 0.99]$. The discount factor weighs the importance of future rewards, with higher values emphasizing long-term gains.
- **Exploration Rate (Epsilon) Variations:** Values tested were $[0.4, 0.5, 0.6]$, dictating the trade-off between exploring new actions and exploiting known ones.

For each parameter combination, the algorithms were trained and run across multiple simulations to gather data on the average steps required to complete the track. Each combination was evaluated over ten independent trials to mitigate the randomness in the simulation outcomes. This extensive testing allowed for the identification of the most effective parameter set for minimizing travel steps to the goal.

2.4.2 Testing and Validation

Here is how the testing was conducted for Value Iteration, Q-Learning, and SARSA:

- **Value Iteration Testing:** Once the Value Iteration algorithm converged to an optimal policy, this policy was tested by simulating its performance on the racetrack. The simulation was run 10 times to account for any stochastic variability in the racetrack environment, such as starting positions and crash recovery. The average number of steps required to complete the track across these simulations was recorded to assess the efficiency of the policy.
- **Q-Learning and SARSA Testing:** For Q-Learning and SARSA, the testing procedure was more extensive due to their reliance on episodic learning:
  1. **Policy Generation:** Each algorithm was trained independently to generate 10 different policies. This approach helped in capturing a range of potential behaviors and strategies that the algorithms could learn.
  2. **Multi-Run Simulations:** Each of the 10 policies obtained from the Q-Learning and SARSA algorithms was then tested by simulating 10 races on the racetrack for each policy.
  3. **Average Steps Calculation:** For each policy, the average number of steps taken to reach the goal was calculated across its 10 simulations. The final performance metric for each algorithm was then obtained by averaging these results across all 10 policies, providing a comprehensive view of each algorithm's effectiveness under optimal hyperparameter settings.

These detailed testing procedures ensured that each algorithm was not only optimized for performance through hyperparameter tuning but also validated through multiple simulations.

3. Results

This section details the performance outcomes of the Value Iteration, Q-Learning, and SARSA algorithms across different racetrack configurations. Performance is measured in terms of the average number of steps required to reach the finish line. Each algorithm was tested under varying conditions including different learning rates, discount factors, and exploration rates. The results are presented side by side in tables, summarizing the steps to finish for each track and the statistical significance of the differences between the algorithms.

Table 1: Performance and Statistical Comparison Across Tracks

Table 2: Average Steps to Finish

Track	Algorithm	Average Steps
L-Track	Value Iteration	16.8
	Q-Learning	97.46
	SARSA	115.23
O-Track	Value Iteration	30.1
	Q-Learning	252.77
	SARSA	334.03
W-Track	Value Iteration	13.6
	Q-Learning	215.08
	SARSA	143.59
R-Track (Nearest)	Value Iteration	34.2
	Q-Learning	316.5
	SARSA	374.14
R-Track (Original)	Value Iteration	317.9
	Q-Learning	394.97
	SARSA	417.21

Table 3: Statistical Comparison of Algorithms

Comparison	t-statistic	p-value
Value Iteration vs. Q-Learning	-5.53	2.97e-05
Value Iteration vs. SARSA	-4.66	0.0002
Q-Learning vs. SARSA	-0.69	0.4968
Value Iteration vs. Q-Learning	-6.95	1.70e-06
Value Iteration vs. SARSA	-7.48	6.25e-07
Q-Learning vs. SARSA	-1.57	0.1334
Value Iteration vs. Q-Learning	-6.12	8.74e-06
Value Iteration vs. SARSA	-3.26	0.0043
Q-Learning vs. SARSA	1.38	0.1831
Value Iteration vs. Q-Learning	-11.26	0.0000
Value Iteration vs. SARSA	-11.66	0.0000
Q-Learning vs. SARSA	0.28	0.7818
Value Iteration vs. Q-Learning	-1.22	0.2383
Value Iteration vs. SARSA	-1.67	0.1116
Q-Learning vs. SARSA	-0.45	0.6578

4. Discussion

The results presented in Table 1 support the initial hypothesis that Value Iteration, a model-based approach, would generally outperform the other two model-free methods, Q-Learning and SARSA, in terms of fewer average steps to reach the finish line across all track configurations.

The experiment across multiple tracks revealed significant disparities in performance. For instance, in the L-Track, Value Iteration significantly outperformed both Q-Learning and SARSA, as evidenced by the negative t-statistics of -5.53 and -4.66 respectively, both yielding low p-values. This suggests a substantial efficiency of Value Iteration in navigating simpler track layouts where the optimal policy can be more directly computed.

Q-Learning and SARSA showed a much closer performance to each other across most tracks, with the differences often not being statistically significant. For example, on the R-Track with the nearest crash handling, the performance difference between Q-Learning and SARSA was minimal (t-statistic of 0.28), indicating a comparable strategy development by these algorithms under a model-free setup. This aligns with the secondary hypothesis that Q-Learning, despite its off-policy advantages, does not always significantly outperform SARSA in environments where the action-outcome relationship is highly stochastic and the track complexity increases.

The introduction of severe penalties for crashing, as in the R-Track under 'original' crash conditions, presented a challenging scenario where all algorithms struggled to perform well. Value Iteration still maintained better performance compared to Q-Learning and SARSA, but with a reduced margin and higher variability, as indicated by the narrower t-statistics (-1.22 and -1.67 respectively) with higher p-values. This highlights the critical impact of environmental penalties on learning efficacy and algorithm performance.

These findings underline the importance of choosing the right algorithm based on specific task conditions and environment settings. While Value Iteration is generally more efficient

due to its comprehensive state-evaluation mechanism, a model might not always be available and the computational overhead associated with model-based approaches might not always justify the performance gains.

In conclusion, the study demonstrates that while Value Iteration often leads to faster convergence to an optimal policy, the choice between Q-Learning and SARSA should consider factors such as the environment's predictability and the severity of penalties for errors.

5. Conclusion

This study provided a comprehensive evaluation of three different reinforcement learning algorithms—Value Iteration, Q-Learning, and SARSA—applied to the simulated racetrack problem across multiple track configurations. The objective was to understand how each algorithm performs in minimizing the number of steps required to complete a race, showcasing their relative efficiencies and effectiveness under different conditions.

The findings confirmed our primary hypothesis that Value Iteration, due to its model-based nature, typically outperforms the model-free methods of Q-Learning and SARSA. This was particularly evident in simpler track layouts where the direct computation of an optimal policy is feasible.

The secondary hypothesis regarding the comparative performance of Q-Learning and SARSA revealed that neither algorithm consistently outperformed the other significantly. This suggests that the choice between these two model-free methods should be guided by specific environmental dynamics and the desired balance between exploration and exploitation.

For future research, it would be interesting to explore adaptations and enhancements to these algorithms to improve their robustness and efficiency, particularly in more complex and dynamic environments. Potential avenues could include integrating aspects of deep learning to handle higher-dimensional state spaces or experimenting with different reward structures and penalty conditions to better capture the nuances of real-world driving scenarios. One could also include eligibility traces to enhance learning speed and convergence.

In conclusion, this study shows the strengths and limitations of Value Iteration, Q-Learning, and SARSA in the context of the racetrack problem but also emphasizes the importance of algorithm selection based on specific task characteristics and environmental settings. These insights are crucial for understanding the field of reinforcement learning and enhancing its application to complex control tasks in various domains.

References

Richard Bellman. Dynamic programming. *Princeton University Press*, 1957.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.