Learning to Drive: Reinforcement Learning Approaches to the Racetrack Problem

Erik Moore MOOREERI091104@GMAIL.COM
School of Computing
Montana State University
Barnard Hall, 357, Bozeman, MT 59717, USA

Taron Moe-Stull TARONMOE@ICLOUD.COM
School of Computing
Montana State University
Barnard Hall, 357, Bozeman, MT 59717, USA

Editor: Text Editor (Overleaf)

Abstract

The racetrack problem is a common barometer for reinforcement learning wherein an agent must traverse a car from some start line to some finish line as quickly as possible with various restrictions. In project 4 we implemented and compared three reinforcement algorithms to complete the racetrack problem and assess the various algorithms efficiency. To complete the racetrack problem we used three reinforcement learning approaches: model based Value Iteration, Q learning methods, and SARSA. The track environment is defined using an ASCII grid for distinct positions, velocity is two dimensional and bounded with acceleration being represented by *-1, 0, 1* for each deceleration, stop, and acceleration, respectively. Randomness was introduced with a 20 percent likelihood that any attempted acceleration will fail with the velocity staying the same. Outside of those rules, we experimented with two different crash handling techniques. Resetting the car to the nearest valid track cell, *(NRST)*. Or, returning to the original start line with zero velocity *(STRT)*. Our groups hypothesis is that Value Iteration will create a strong policy when given complete model knowledge. Q learning will likely learn shorter but more aggressive policies than SARSA, particularly with the *NRST* crash handling rule. SARSA will likely converge at safer, more simple policies then those produced by Q learning. We evaluated all three algorithms on the provided tracks then compared performance in terms of steps to finish, quality of policy, and overall path shapes. The results ideally provided an outline on how differing crash penalties, crash handling, step counts, track complexity, and algorithm selection will shape strategies and policy efficiency of the various reinforcement learning methods we put to the test.

Keywords: Reinforcement Learning, Value Iteration, Q learning, SARSA, Racetrack Problem

1 Problem Statement

In this project our goal is to find optimal solutions to a racetrack problem using three Artificial Intelligence algorithms. These algorithms are value iteration, Q-learning, and state-action-state-reward-action (SARSA). The puzzle itself involves a car (our agent) on a racetrack with a start, barriers, and a finish. Additionally, the car itself is considered a moving object with (partial) direct control over its acceleration, and thus indirect control over its velocity and position. These controls are in the x and y directions. The goal of the agent is to finish the racetrack with the minimum number of acceleration-action-steps. Finally, the agent is impacted by a penalty if it collides with a barrier: its velocity is always set to 0, and it either ends up in the previous valid position or must restart the track altogether. This rule is set by the user prior to running the program.

All of our algorithms will be tested using the same tracks and crash handling rules while attempting to create a policy where the agent can actually reach the finish efficiently. We will then compare the performances of our Value Iteration, Q learning, and SARSA algorithms using the step count metric counting steps required to complete each track, consistency of successful paths found, and crash/navigation behavior in avoidance situations. Because we have randomness and crash penalties in our track environments, we expect the algorithms learning behavior to be determined by punishment and uncertainty fairly frequently. (CSCI 446 Course Materials, 2025)

Hypothesis

We hypothesize that our agent will generally seek out a more risk-averse path when it must re-start due to errors as opposed to simply losing its velocity. Although we can only guarantee that the actual optimal path is found via value iteration, we should be able to hypothetically assume that the penalty will manifest itself in Q-learning and SARSA as well. In particular we expect value iteration to output the strongest and consistently most efficient policies simply because it has total knowledge of the model and doesn't need real exploration to learn. For the methods with no model our expectation for Q learning is it will converge faster than SARSA with more aggressive pathing as a result of its off policy update strategy. With SARSA however we believe it will behave with more caution because its learning updates are correlated with actions actually taken throughout its exploration. The last thing we need to address in our hypothesis is the random acceleration failure handling. We expect all of our models to exhibit a range of variability in the learned policies, especially with Q learning and SARSA given no model knowledge. (Russell and Norvig, 2020)

2 Experimental Approach

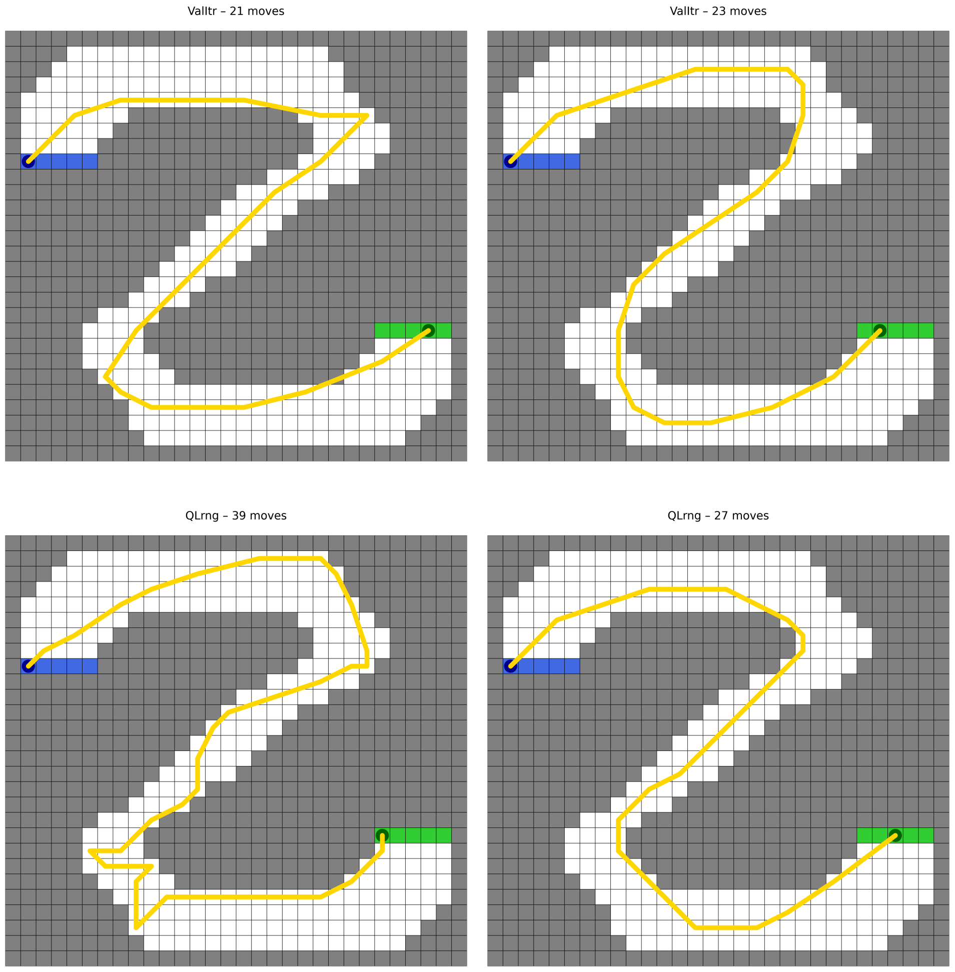
In order to evaluate and compare the performance of our three reinforcement learning algorithms we ran each method on the same tracks with identical starting parameters. We decided to use our value iteration model as the baseline because its running with complete model knowledge. For both Q learning and SARSA we trained them through repeated interaction with the track environments where the agent can learn through iterative trial and error rather than having that full model knowledge like value iteration. Each of the learning based methods required several training runs before we could evaluate the final policy. Then, the final policy was tested separately from the training policies to avoid exploration bias.

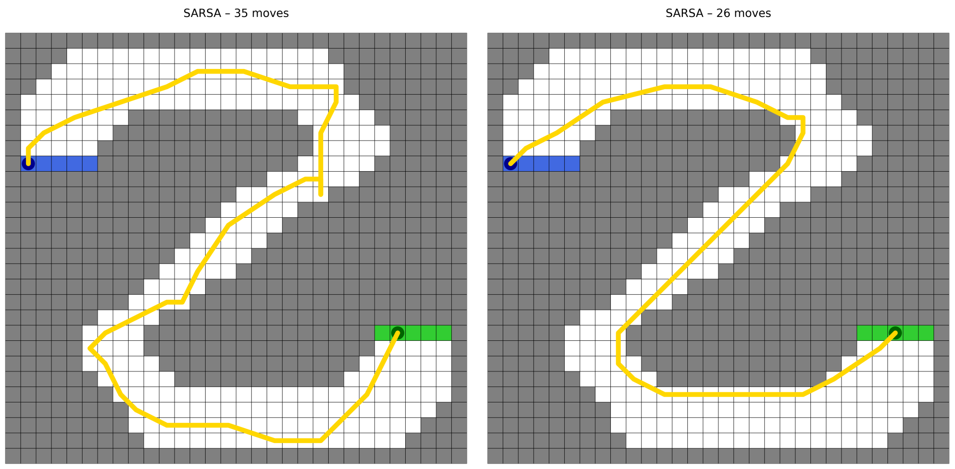
All of our algorithms treated their environments as a state space made up of the agents position and velocity, then the available actions were simply changes in acceleration represented by -1, 0, and 1. Then, with the random 20 percent acceleration failure rate a significant amount of randomness was introduced. In our Q learning and SARSA algorithms, a greedy exploration approach was taken, with both learning rate and discount factor being chosen through reasonable convergence rather than exhaustive tuning. Our group also decided that resetting the agent after a crash was a better way to structure the cost in the racetrack environment. This means that the consequences were introduced more naturally and influenced the strategies and policies rather than adding more reward structures. Using this set up, the differences in each reinforcement algorithm were examined and tracked as shown in our results. (Sutton and Barto, 2015)

3 Results

In line with our initial hypothesis, the STRT requirement did consistently result in finding solutions that mitigate risk against collision. However, it also resulted in substantially better solutions (requiring fewer moves) in both Q-learning and SARSA. Images of our 2-track solutions convey this trend relatively clearly:

—— Nearest Position Settings —— —— Restart Requirement Settings ——





Figures 0 a-f: 2-Track Found Pathways From Our 3 Algorithms and 2 Rule-sets

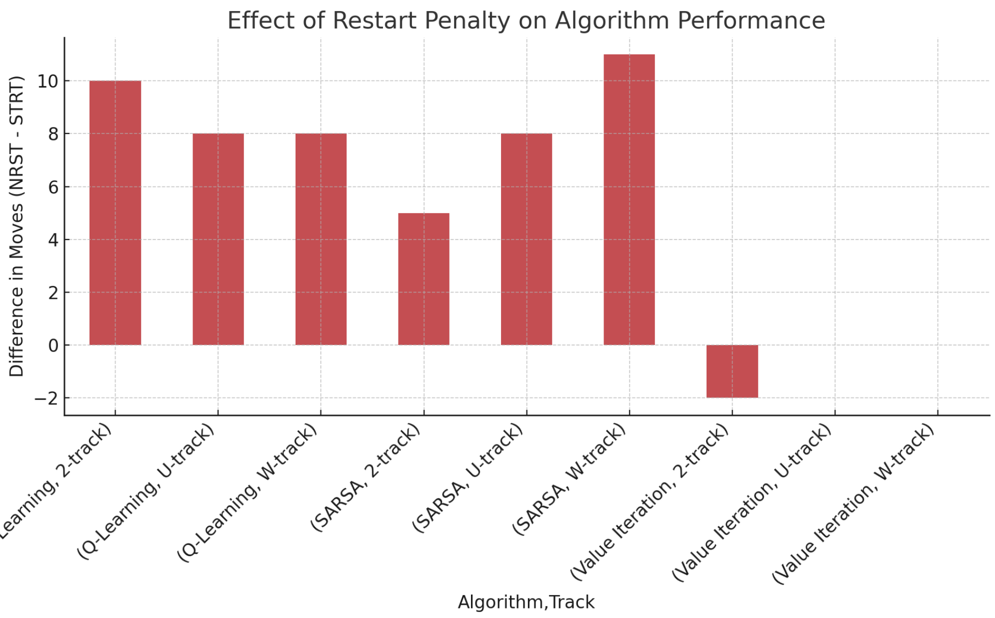
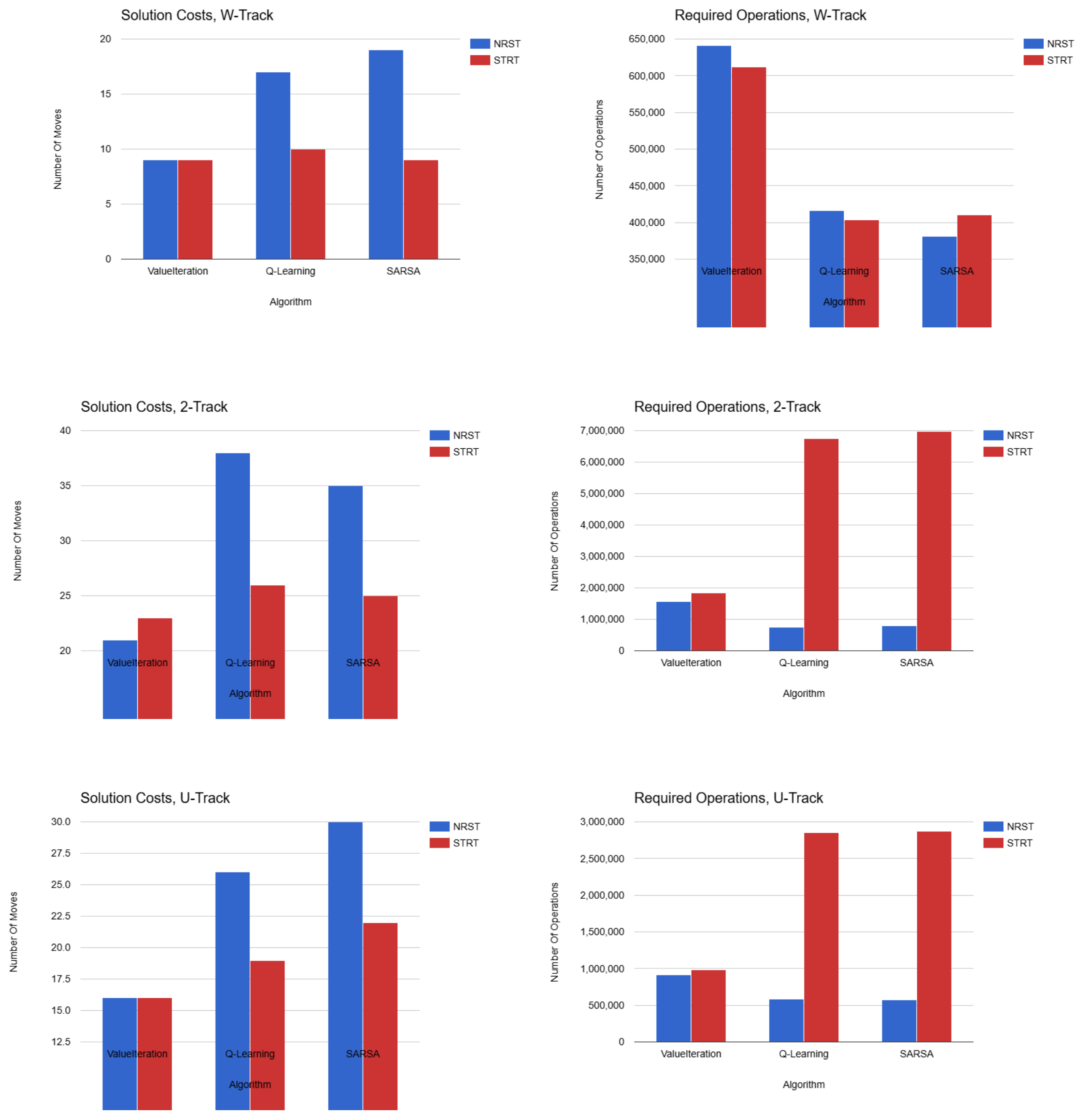


Figure 1: Effect of crash handling strategy on performance.

As previously stated and confirmed via observation, the STRT requirement generally results in cleaner found solutions in Q-learning and SARSA. The reason for this likely has to do with both the nature of the algorithms and our implementation. By this, we mean to say that the stricter rule-set for the agent results in longer required training time to find solutions, and thus more moves taken and an effectively longer training time. In our code, we require our agent to either find the solution or fail (exceeding a maximum of 500 moves results in failure and the run resets) a set number of times before training concludes. Thus, the stricter rule-set averages many times more moves per run early on, resulting in greater experimentation and exploration of action trees. To confirm this, we are able to compare the number of operations required to complete our training with and without the return-to-start collision requirement as shown below:



Figures 2 a-f: Comparison Between Number of Moves in Found Solution and Runtime Operations Required in Computation

In our short W-track, there is no immediately discernible difference between our runtime operations and our rule selection. However, the more complex 2-track shows a very obvious difference with a full order-of-magnitude increase in computational demand (7×10^5

compared to 7×10^6). This confirms our suspicion that a stricter rule-set demands more exploration. In the next section, we will briefly discuss a few potential methods to balance these traits for this racetrack problem. As a note: value iteration performed relatively consistently across rule-sets both in runtime operations and moves required.

4 Discussion

Our hypothesis is somewhat solidified by our observations across all three algorithms, but is also proven to be incomplete. With the simplistic W-track, the best solution as found by value iteration is identical for both cases. However, the harsher rule-set results in our Q-learning and SARSA algorithms finding better solutions at similar computational cost (it likely acts as a stabilizer for agent direction, more harshly punishing wasted movement). For 2-track, the solutions for 2-track are similar between solutions for value iteration, following a slightly more strict pathway (on top of requiring more moves) in the case of the harsher rule-set. The Q-learning and SARSA algorithms do find better pathways that are also more strict (safer) at the cost of massively increased computational demand. Finally, considering the U-track, we see results similar to the 2-track which are slightly less exaggerated.

To amend certain discrepancies in runtime between rule-sets for Q-learning and SARSA, we may consider a few potential adjustments to our code. The first option is fairly trivial, but might only provide limited or marginal benefit: tuning existing parameters based upon the rule-set. This includes the training rate, the number of moves allowed before run reset, and the number of runs taken in the training process. If we consider our baseline 'nearest-position' rule performance to be satisfactory, then we would prefer to solely adjust our 'return to start start' rule parameters and settings. Our goal is thus to make earlier runs more likely to complete as found solutions. Thus, we would increase the maximum number of moves allowed in a run and decrease the number of runs in the training. But the issue arising is already due to the lack of distinction between a run restarting and a collision occurring, so this seems like a very limited fix. A more prudent option would be to adjust the scoring to consider the distance traveled from the start in scoring. This would work by building a purely positional value iteration table to start, and then scoring less harshly for moves that move the agent further from the start (actually, closer to the finish is likely a better heuristic). The distance lost due to a collision could also be considered in the scoring. Thus, the agent would not be required to wander somewhat aimlessly for initial runs until it (entirely by chance) finds a path to the finish to build upon, and would instead have a measurable way to track progress.

Of course, this would introduce its own challenges on other hypothetical tracks. For our tracks, the agent is forced down a relatively narrow pathway to find the finish, so moving further from the start is almost always valuable. But, more maze-like tracks (maybe multiple valid pathway with varied twists and turns) could be encouraged to find and reinforce sub-optimal solutions due to this greedy approach. Still, it seems clear that the limitations on the agent's positive reinforcement introduced by the restarting failure penalty is best compensated in some similar method to this. Other tweaks could include scaling prioritization of the greedy partial value iteration approach when runs fail to find solutions and diminishment of the reliance on the distance heuristic as the agent becomes better.

For the purposes of this class, these ideas are best noted as hypotheticals and not directly implemented.
The trends that we found solidified and confirmed our understanding of the nature of these algorithms. The results are reasonably explained by logic, and help us to build on our underlying hypothesis regarding system behavior. It is also useful in helping us to find methods of improving these algorithms.
As a quick comparison between our deterministic and non-deterministic algorithms, it is clear that depending on the rule-set we may find solutions more quickly using the non-deterministic algorithms, but this is not a guarantee. The rule-set which did not require the agent to restart on collision required less computational power than every case of value iteration, and still managed to find a workable solution in most cases. However, as stated: improper or short-sighted management of rules quickly diminishes the potential benefits of these non-deterministic systems. We can therefore conclude that with proper consideration non-deterministic systems may often be a better choice, but only if implemented with sufficient rigor to minimize potential downside cases. Ultimately, it is a matter of desired reliability.
Overall, our discoveries confirm our suspicions and inferences regarding the behavior of these algorithms and reinforce our observed trend of use cases for deterministic and non-deterministic algorithms in general. Clearly, deterministic algorithms always provide the optimal solution (when usable due to finite system state-space) with rigid limitations on optimization; while, non-deterministic algorithms often fail to provide the very best solution, but exhibit more room for potential improvement and increased flexibility in tradeoffs.

5 Summary

In project 4, we implemented and compared reinforcement learning and dynamic programming approaches to a racetrack problem with a varied rule/consequence set: value iteration, Q-learning, and state-action-reward-state-action. Using the former as a baseline, we analyzed their accuracy, efficiency, and the impact of puzzle complexity and scale on the representation of these variables. Additionally, this project massively expanded on our implementation of discrete-space decision-making pattern tracking compared to previous assignments with the resulting position abstracting from velocity, which was abstracted from the acceleration variable the agent is able to (mostly) control. This created a discrete space solvable by deterministic reasoning, but with the potential to be made substantially cheaper using informed reinforcement learning.
We were adequately prepared to both implement and draw conclusions from observed behavior. Our initial hypothesis was verified and expanded upon, and we were able to observe the impacts both of reinforcement learning and rule definition on the scalability of our path-finding.
In conclusion, we were able utilize the value iteration, Q-learning, and SARSA algorithms to find optimal and efficient solutions to the racetrack problem, and to substantially vary our prioritization of both outcome properties.

References

Montana State University CSCI 446 Course Materials. Project 4 reinforcement learning and the racetrack problem. `https://montana.instructure.com/courses/19506/files/`, 2025. Accessed: December 2025.

Overleaf. Overleaf, online LaTeX editor. `https://www.overleaf.com`. Accessed December 2025.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, Hoboken, NJ, 4th edition, 2020. Accessed: December 2025.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2015. Accessed December 2025.