Journal of Machine Learning Research 5 (2022) 8 Submitted 5/2022; Published

# Implementing Various Reinforcement Learning Methods in Different Racetrack Environments

**William S. Ventura** WVENTUR1@JHU.EDU
*Department of Computer Science*
*Johns Hopkins University*
*Baltimore, VA 21218-2682, USA*

**Editor:** William S. Ventura

## Abstract

Reinforcement Learning, unlike the other machine learning methodologies explored this semester takes into consideration how the 'Agent' interacts with its environment and in such a method that the 'Agent' can maximize the cumulative rewards gained. Reinforcement Learning further differs from other methodologies such as Supervised Learning, where the Agent in not told what is the best move to achieve the best reward, instead it must come to that conclusion through Trial-and-Error.

This paper will explore implementing various Reinforcement Learning and apply them to three different racetrack environments. In this paper the implementations of the Value Iterative Algorithm, Q-Learning Algorithm, and the SARSA Algorithm will be used to solve the Racetrack problem on three different shaped tracks, 'R' Shaped , 'L' Shaped, and 'O' Shaped.

**Keywords:** Value Iterative Algorithm, Q-Learning, SARSA, Reinforcement Learning

## 1. Introduction

Unlike the typical machine learning approaches explored we have explored this semester, Reinforcement Learning aims to mimic an agent interacting and reacting to its environment and abide by rules to perform an action within that environment that could lead to a reward. In Reinforcement Learning, the agent aims to find a balance between exploration of the uncharted space and exploitation of its current knowledge of the environment. These trade-offs in Reinforcement Learning, between exploration and exploitation has been thoroughly studied through Markov Decision Processes(MDPs) and the multi-arm bandit problem. MDPs are widely used as the mathematical framework for modeling a wide range of optimization problems that are attempted to be solved by Reinforcement Learning and Dynamic Programming.

This paper implements three different Reinforcement Learning Methods, the Value Iterative Algorithm (VIA), Q-Learning, and SARSA. Those three different learning methods will then be evaluated on the 'R', 'L', and 'O' shaped racetracks. The Value Iterative Algorithm is a fairly simple algorithm that attempts to find the optimal policy and converges to correct mappings between the states in the environment and their expected values. Q-Learning on the other hand learns an action-value function, that gives the agent the expected reward of performing said a action at said state. However while this may be advantageous over

VIA because it allows for the comparison between the expected usefulness of the action, it comes at a disadvantage as it would increase the size of the state space, the time to converge, and iteration completion time. In comparing VIA and Q-Learning, VIA has a specific state transition probability, whereas Q-Learning learns transition probabilities through Trial-and-Error. VIA can be viewed as a model-based Reinforcement Learning, where Q-Learning is model-free. SARSA on the other hand is a variant of Q-Learning. The name SARSA means State-Action-Reward-State-Action. It's name simply reflects that main process of updating Q-values is dependent on the current state $S_1$, the action of the agent $A_1$, the reward, $R$ for choosing the action, the new state after choosing such action $S_2$, and the next action $A_2$. While typical Q-Learning learns the Q-values associated with taking an optimal policy while following an exploration/exploitation policy, SARSA attempts to learn the Q-values associated with taking the policy it itself follows.

It is hypothesized that while the VIA may take less iterations to converge, it has limitations in terms of applicability as the complexity reaches a lot sooner than Q-Learning as VIA cycles through all possible states and can assess the right actions to take using a known transition model. While Q-Learning and SARSA may increase the complexity and state space, it is hypothesized that SARSA being an on-policy algorithm will be more useful in optimizing the value of an agent's exploration, than Q-learning which is an off-policy algorithm, however since it follows the policy that it is learning, instead of any policy that fulfils the convergence requirements that is distinct of Q-Learning, SARSA will take a longer time to converge.

## 2. Methods

### 2.1 Racetrack Environment

For this racetrack problem the goal is to control the movement of the race-car(agent) along one of the three tracks ('R', 'L', 'O') in order for the racer to reach the finish line in the least number of steps possible. At each time step $t$ the state of the racer is given by these four variables, $X_t$, $Y_t$ , $V_{xt}$, $V_{yt}$. The variables $\dot{X}_t$, and $\dot{Y}_t$ represent the $x$, and $y$ components of the displacement vector at time $t$. Additionally, $V_t$ and $A$ are the velocity and acceleration vector at time step $t$. Within this racetrack environment all the positions are integer values and the kinematics equations governing this environment are given as such:

1. $X_t \equiv X$ position
2. $Y_t \equiv Y$ position
3. $V_{xt} = \dot{X}_t = X_t - X_{t-1} \equiv$ X Speed
4. $V_{yt} = V_{yt} = Y_t - Y_{t-1} \equiv$ Y Speed
5. $A_{xt} = \ddot{X}_t = \dot{X}_t - \dot{X}_{t-1} \equiv$ X Acceleration
6. $A_{yt} =\ddot{Y}_t = \dot{Y}_t - \dot{Y}_{t-1} \equiv$ Y Acceleration

The car's agent only has control over the values of $A_x$ and $A_y$ at any given time step $t$, and uses these to alter the car's state, giving the car the ability to accelerate, decelerate and turn. As a further restriction in this environment the values of $A$ and $V$ restricted as such:

$$A_{x,y} \in \{-1, 0, 1\}$$

$$(\dot{X}_t, \dot{Y}_t) \in [\pm 5, \pm 5]$$

Furthermore, each time the agent attempts to accelerate there is a 20% probability that the attempt will fail incorporating non-determinism to this environment. There will also be two rules implemented for crash scenarios, a Soft Crash Rule, where the agent returns to the nearest position prior the crash and a Hard Crash Rule where the agent returns to the beginning. Lastly the discount factor, $\gamma$ is set to 0.8, the learning rate, $\alpha$ is 0.2. In accordance to the structure of this environment VIA will perform 50 iterations while Q-Learning and SARSA will be perform $2.0e + 6$ episodes and each episode has 50 iterations.

### 2.2 Value Iterative Algorithm

The Value Iterative Algorithm attempts to calculate an optimal policy so that the agent will be able to choose its actions based on that policy. The optimal action in each state is selected by looking at each of the state utilities. The Value Iterative Algorithm calculates a states given utility through the Bellman Equation. The Bellman Equation is defined as such:

$$U(s) = max_a(R(s, a) + \gamma \sum_{s'} P(s'|s, a)U(s')$$

Where $s$ is the current state, $s'$ is the next state, $a$ is the action taken, $\gamma$ is the discount factor, $R(s)$ is the reward function, and $U(s)$ is the utility/value function of being in state $s$, and $P(s'|s, a)$ is the probability of going into state $s'$ from state $s$ when choosing to take action $a$.There is one Bellman equation for each and the utility of each state is initialized to zero and once the algorithm converges, those utility values are used to choose an action in each state allowing the agent to reach the finish line.

### 2.3 Q-Learning

Q - Learning on the other hand is considered to be more of a model-free type of learning. It is described as a type of active reinforcement learning, specifically an on-policy algorithm. What makes the Q-Learning algorithm different than VIA, is that Q-Learning compared the expected utility of the available actions thus not requiring a model of the environment. Thus more specifically the Q-Learning algorithm learns an action-utility representation instead of just the utilities. The Q-function $Q(s, a)$, means the value of doing action $a$ in state $s$ and the relationship between Q-values and Utility values is demonstrated by:

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

The update equation for Q-learning is given by:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R(s_{t+1}) + \gamma \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t))$$

These Q-values are continually updated until the algorithm converges.

### 2.4 SARSA

The SARSA algorithm stands for State-action-reward-state-action. It stems off Q-learning, where Q-Learning is an Off-policy algorithm, SARSA is an on-policy algorithm. In SARSA the agent interacts with the environment and updates the policy based on actions taken. The Q-function differs in SARSA as such:

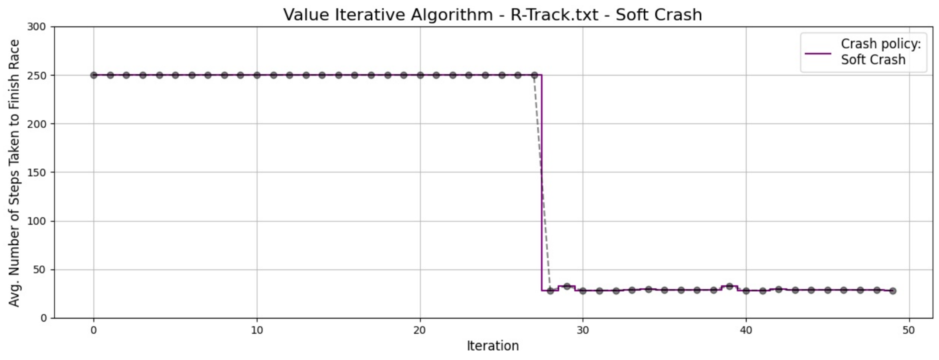
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(R(s_{t+1}) + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

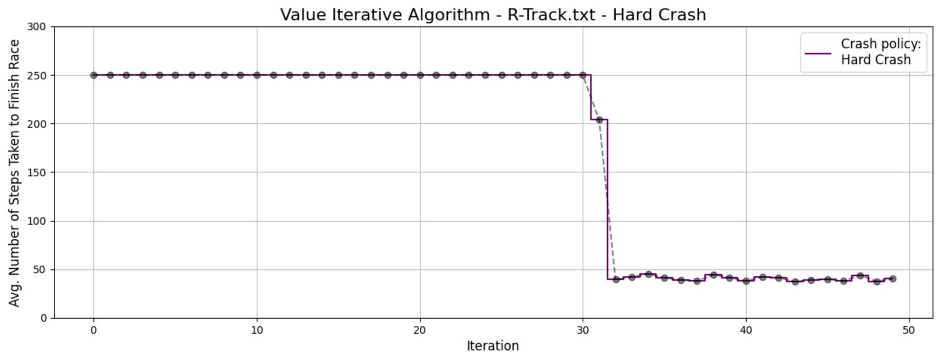
In SARSA, the update equation is dependent on the current state, current action, the reward obtained, the next state and the next action.

## 3. Results

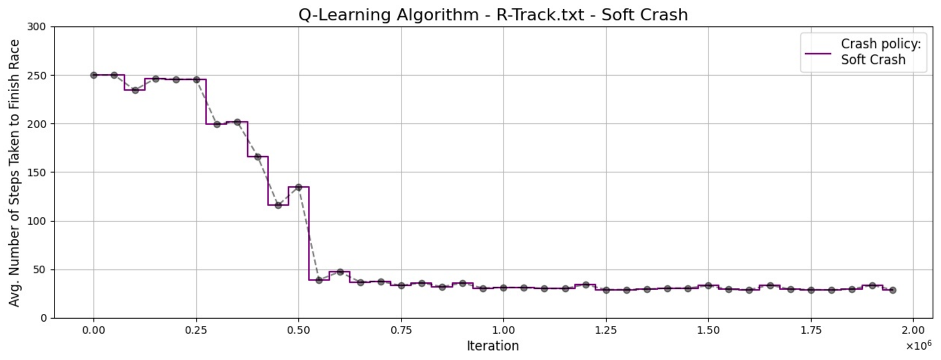
### 3.1 R-Track

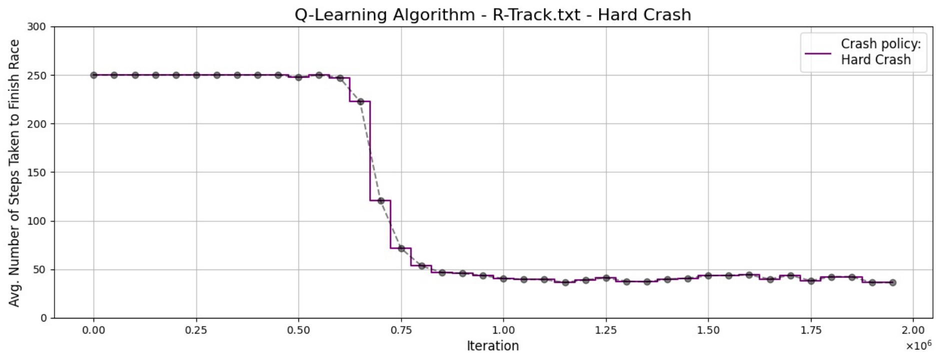
#### 3.1.1 Value Iterative Algorithm



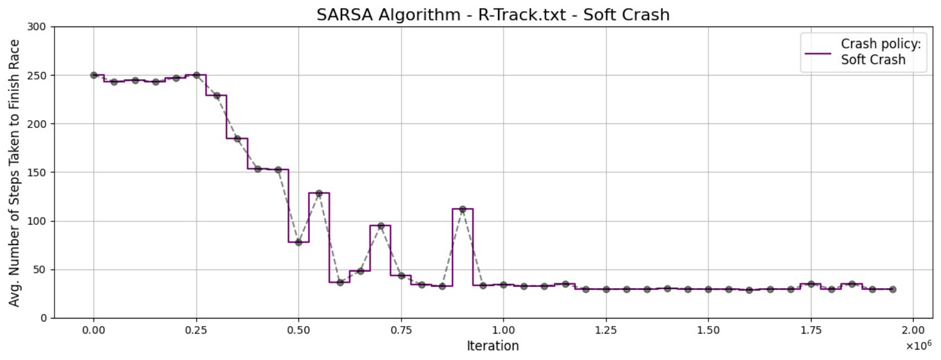


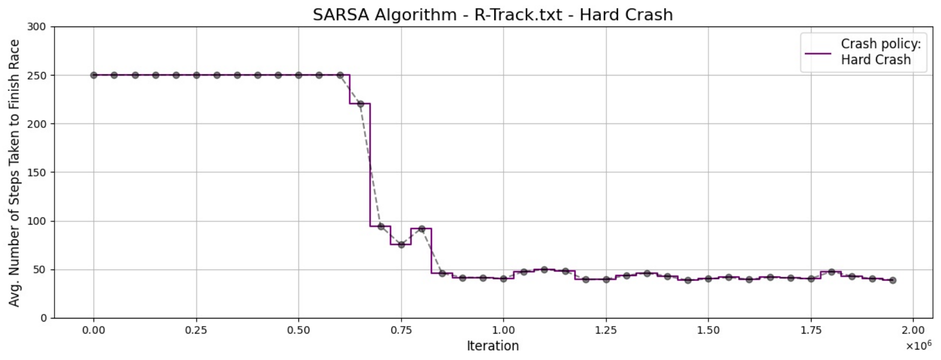
#### 3.1.2 Q-Learning Algorithm


Q-Learning Algorithm - R-Track.txt - Soft Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10^6
Crash policy: Soft Crash

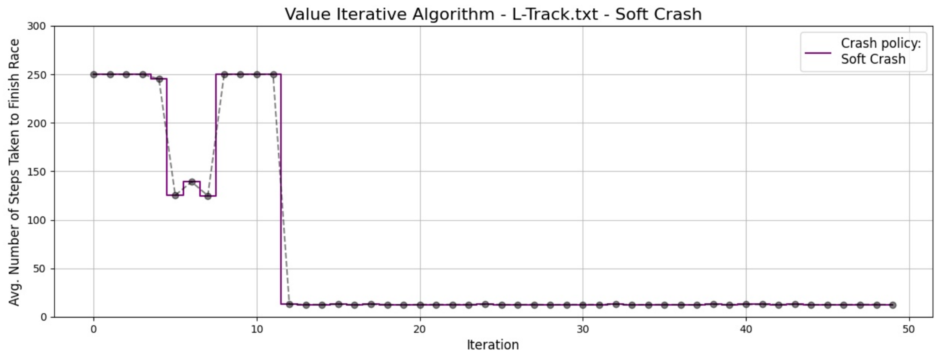

Q-Learning Algorithm - R-Track.txt - Hard Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10^6
Crash policy: Hard Crash

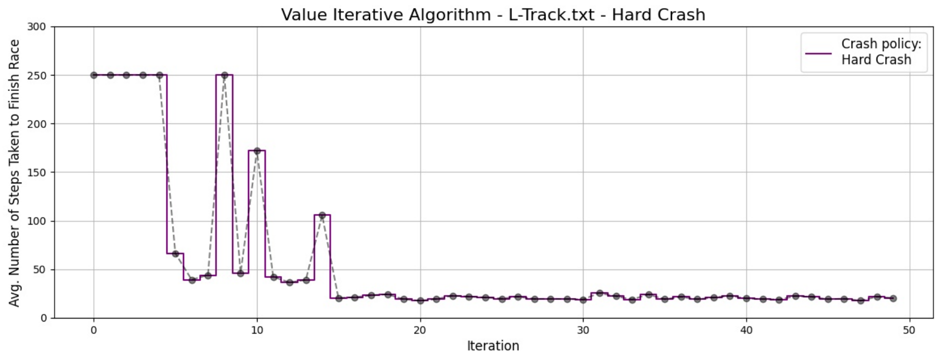
### 3.1.3 SARSA


SARSA Algorithm - R-Track.txt - Soft Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10^6
Crash policy: Soft Crash

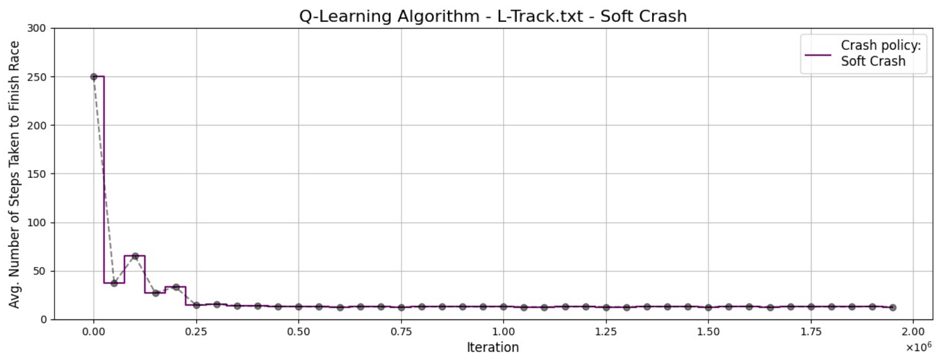

SARSA Algorithm - R-Track.txt - Hard Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10⁶
Crash policy:
Hard Crash
0
50
100
150
200
250
300
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

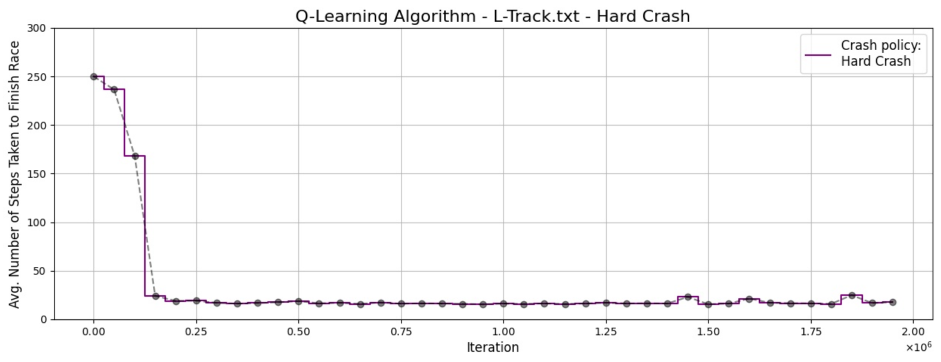
## 3.2 L-Track


Value Iterative Algorithm - L-Track.txt - Soft Crash
Avg. Number of Steps Taken to Finish Race
Iteration
Crash policy:
Soft Crash
0
50
100
150
200
250
300
0
10
20
30
40
50

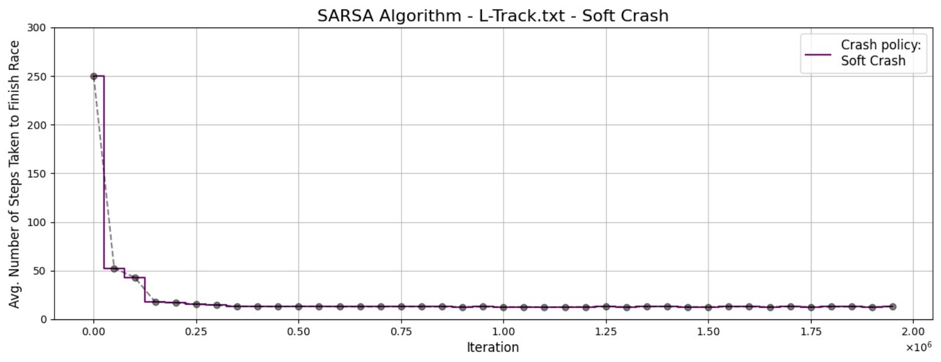

Value Iterative Algorithm - L-Track.txt - Hard Crash
Avg. Number of Steps Taken to Finish Race
Iteration
Crash policy:
Hard Crash
0
50
100
150
200
250
300
0
10
20
30
40
50

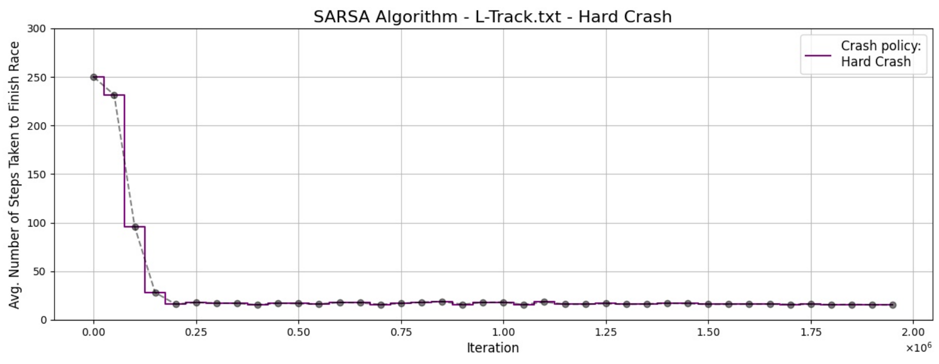
### 3.2.1 Q-Learning Algorithm



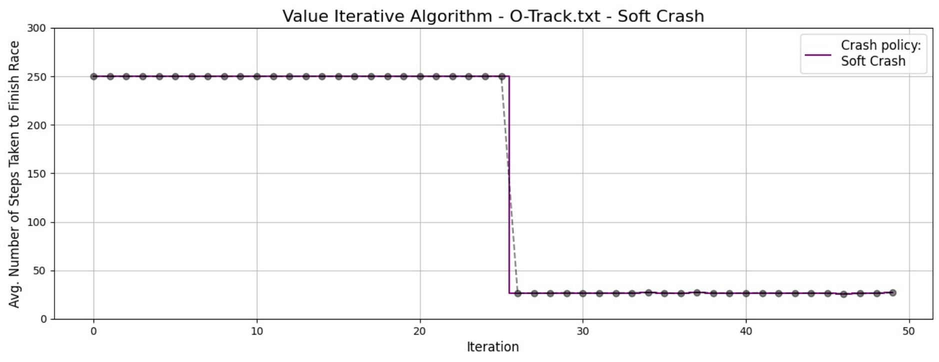


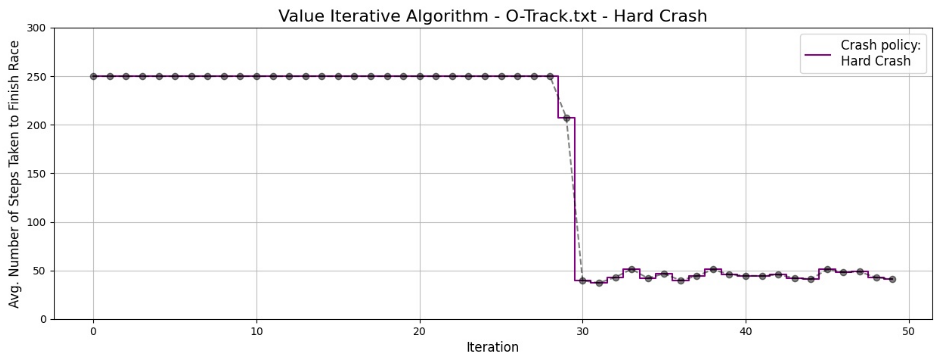
### 3.2.2 SARSA



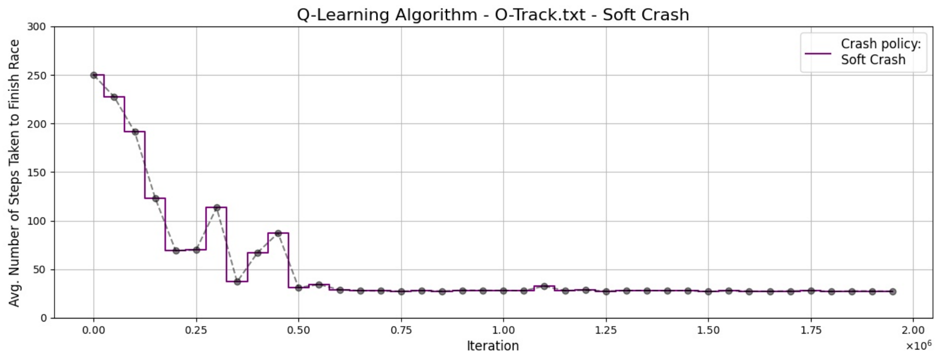


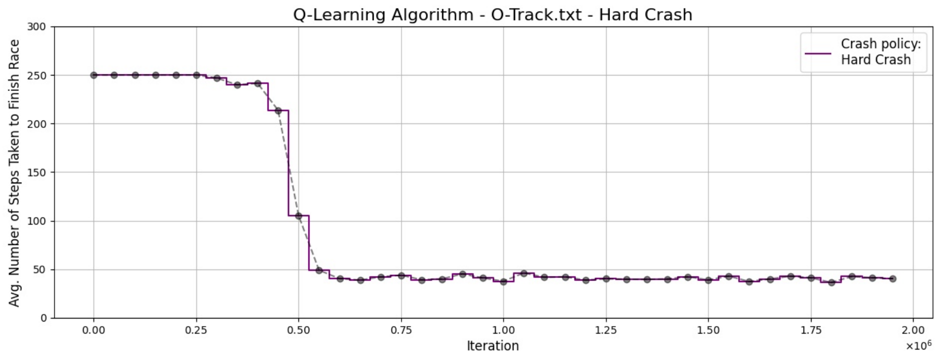
## 3.3 O-Track



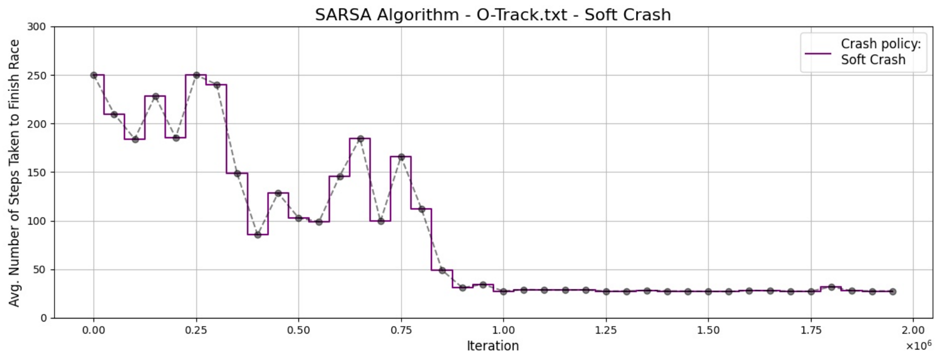


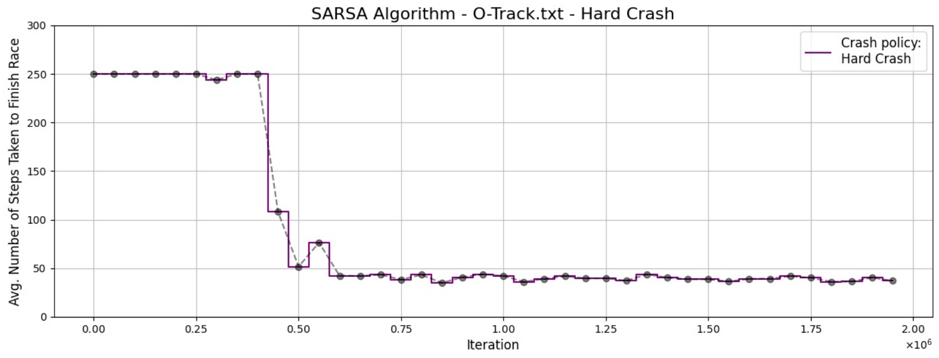
### 3.3.1 Q-Learning Algorithm


Q-Learning Algorithm - O-Track.txt - Soft Crash
Crash policy: Soft Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10^6


Q-Learning Algorithm - O-Track.txt - Hard Crash
Crash policy: Hard Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10^6

### 3.3.2 SARSA


SARSA Algorithm - O-Track.txt - Soft Crash
Crash policy: Soft Crash
Avg. Number of Steps Taken to Finish Race
Iteration
×10^6



## 4. Discussion

### 4.1 Value Iterative Algorithm

#### 4.1.1 R-Track

In implementing the Value Iterative Algorithm on the three different shaped racetracks, under two different types of crash rules, for the R-shaped track it is shown that average number of steps need to complete the rate remained at 250 steps until it reached about the 27th iteration, at that point the average number of steps dropped down to around 30 steps. it then starts to converge after 30 iterations as the VIA algorithm has found the best path to complete the race under the situation that if it crashed it would just return to the nearest spot prior the crash. However for the policy where it would return to the beginning (Hard Crash), the algorithm took longer to converge and decrease its average amount of steps needed to finish the race. In this scenario it took about 45-50 steps to finish the race and that was around the 33rd iteration. This makes sense, since the agent would need to start at the beginning if it crashed thus resulting in it taking longer to converge. Even then it appears as the algorithm in this scenario hasn't fully converged as there is still fluctuation between each iteration unlike the subsequent iterations in the Soft Crash run.

#### 4.1.2 L-Track

Under the Soft Crash rule, the agent starts off by taking the maximum number of steps allowed until it reaches the 5th iteration where it reduces its average number of steps, but then towards for the next 4 iterations it goes back to taking the maximum number of steps allowed, it is go to show that while attempting to perform the local optima policy, it would have started exploring other paths as then after the 11th iteration the average number of steps needed to reach the finish line drastically reduced to an average of around 10 steps, and the graph appears to converge as well. For the Hard Crash scenario, the agent's average number of steps needed to reach the finish line has a volatile fluctuation throughout the first 15 iterations, Which this can be owed to the policy that the agent returns to the beginning after crashing. Afterwards it appears the agent has found the optimal policy and the algorithm converges.

#### 4.1.3 O-Track

For the O-Track, the agent drastically drops the average number of steps need to reach the finish line after the 25th iteration, from there one out it appears that the algorithm has instantly converge. For the hard crash policy, it takes longer for the agent to start converging. Where in the Soft Crash, it started converting at the 25th iteration, in the Hard Crash it started to converge around the 30th iteration with still fluctuating average number of steps.

### 4.2 Q-Learning Algorithm

#### 4.2.1 R-Track

Under the soft crash policy, the agent starts to lower its average number of steps from 250 to 130 for the first $0.5e + 6$ iterations. There is still slight volatility in the average number of steps until it reaches 0.75e + 6 iterations, where it then starts to converge. Comparing it to the hard crash rule, the soft crash rule appears to let the algorithm converge quicker. In the Hard Crash, the agent does not start decreasing its number of steps until 0.6e + 6 iterations and then starts to converge with still fluctuation around 1e + 6 iterations.

#### 4.2.2 L-Track

In contrast to the previous results, for the L-Shape track, the algorithm under the hard crash policy starts to converge before the soft crash policy. While the soft crash rule, starts decreasing before the hard crash rule, it takes a longer amount of iterations to reach the minimum number of steps needed to reach the finish line.

#### 4.2.3 O-Track

The algorithm appears to start converging around the same time for both the soft and hard crash policy, however in the soft crash policy the agent starts decreasing its average number of steps before the agent under the hard crash policy starts. The Soft crash policy is faster to converging by a negligible number of iterations. The soft crash policy also shows a more define convergence where the hard crash policy still has some fluctuation towards the end of its iterations.

### 4.3 SARSA

#### 4.3.1 R-Track

Under the hard crash policy it appears that the hard crash policy allowed for the algorithm to converge much faster than in the soft crash policy as the agent was about to drastically reduce its number of steps in the first 0.8e + 6 iterations in which afterwards there is signs of stabilization with slight variation. The soft crash policy however goes through a longer process of reducing its number of steps and doesn't start to stabilize until after 1.0e + 6 iterations.

#### 4.3.2 L-Track

For the L-Track, the soft crash policy resulted in the algorithm reaching a local optima policy around 0.15e + 6 iterations before fully stabilizing its average number of steps needed to reach finish line after 0.15e + 6 iterations. Where as the Hard crash policy resulted in the agent's average number of steps needed to reach the finish line at around 0.18e + 6 iterations.

#### 4.3.3 O-Track

For the SARSA algorithm, the algorithm is shown to experience convergence faster under the hard crash scenario as it starts to stabilize its number of average steps around the 0.75e + 6 iteration. The soft crash policy however takes longer to stabilize as it is showing high volatility in the number of steps in the first 0.8e + 6 iterations, with it finally stabilizing around 1.0e + 6 iterations.

### 4.4 Comparison of Algorithms

Overall it appears that the Value Iterative Algorithm is the fastest in reaching the time to converge compared to the Q-Learning Algorithm and the SARSA Algorithm, beating them both in all of the different race track environments. In regards to the Q-Learning and SARSA algorithms is appears that the Q-Learning algorithm starts to reduce and converge before the SARSA algorithm under the two different types of crash scenarios, however the SARSA algorithm beats the Q-Learning algorithm but a negligible amount in the L-shaped track. The difference in the time it takes for them to start converging can be attributed to the types of algorithms they are, Q-learning is an Off-policy algorithm whereas SARSA is an On-policy Algorithm, which could mean that because SARSA attempts to learn the Q-values associated with taking the policy itself it results in a longer time to converge, whereas Q-learning instead learns the any policy the fulfills the convergence requirements.

## 5. Conclusion

In the process of implementing the three different algorithms is has been shown the Value Iterative Algorithm performs faster than the Q-Learning and SARSA algorithm, while this is really good in practice in a real applications it may not be the most optimal approach. While in this paper, the implementation of Q-Learning took drastically longer than VIA, it is attributed to the increased complexity in the state space. The results of this paper thus concluded that our hypothesis is true, VIA is faster than the other two algorithms though impractical in real world applications, and Q-Learning converged faster than SARSA due to its principle of following an policy that fulfills the convergence requirements instead of following its on policy, which is notable of SARSA. In these various reinforcement learning methods it is important to note that the Q-function was approximated through means of linear classifiers, if such there were such a case where the Q-function be a non-linear function than other means such as neural networks would be necessary to approximate the Q-function.