Spring 2022

Reinforcement Learning: Low Discrepancy Action Selection for Continuous States and Actions

Jedidiah Lindborg

REINFORCEMENT LEARNING: LOW DISCREPANCY ACTION SELECTION FOR CONTINUOUS STATES AND ACTIONS

by

JEDIDIAH LINDBORG

(Under the Direction of Stephen Carden)

ABSTRACT

In reinforcement learning the process of selecting an action during the exploration or exploitation stage is difficult to optimize. The purpose of this thesis is to create an action selection process for an agent by employing a low discrepancy action selection (LDAS) method. This should allow the agent to quickly determine the utility of its actions by prioritizing actions that are dissimilar to ones that it has already picked. In this way the learning process should be faster for the agent and result in more optimal policies.

INDEX WORDS: Reinforcement learning, Machine learning, Discrepancy, Low discrepancy action selection

2009 Mathematics Subject Classification: 62H30, 68H10

REINFORCEMENT LEARNING: LOW DISCREPANCY ACTION SELECTION FOR CONTINUOUS STATES AND ACTIONS

by

JEDIDIAH LINDBORG

B.S., College of Coastal Georgia, 2016

A.S., College of Coastal Georgia, 2014

A Thesis Submitted to the Graduate Faculty of Georgia Southern University in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

REINFORCEMENT LEARNING: LOW DISCREPANCY ACTION SELECTION FOR CONTINUOUS STATES AND ACTIONS

by

JEDIDIAH LINDBORG

Major Professor:	Stephen Carden
Committee:	Zheni Utic
	Ionut Iacob

Electronic Version Approved:
May 2022

ACKNOWLEDGMENTS

I acknowledge God, first and foremost. Without His grace, I would have no accomplishments. I dedicate this to my mother and father for always believing in me, the professors who invested so much in me, my advisor, Dr. Stephen Carden, for helping to bring me to a master's level, and my friends for supporting me.

TABLE OF CONTENTS

Page

ACKNOWLEDGMENTS 2

LIST OF TABLES 5

LIST OF FIGURES 6

CHAPTER

1 INTRODUCTION 9

2 REINFORCEMENT LEARNING BACKGROUND 12

2.1 Definitions 12

2.2 Examples and Intuition 18

2.3 Continuous cases 21

2.4 Efficient Exploration 22

3 LOW DISCREPANCY SEQUENCES 24

3.1 Discrepancy 24

3.2 Low Discrepancy Sequence examples 36

3.3 Low Discrepancy Action Selection 38

4 LOW DISCREPANCY ACTION SELECTION 40

4.1 Set-up 40

4.2 Uniform Selection 41

4.3 LDAS Candidate 1 43

4.4 LDAS Candidate 2 45

4.5 LDAS Candidate 3 50

4.6 Ornstein-Uhlenbeck selection 53

5 EXPERIMENTAL RESULTS 55

5.1 Technical Set-up 55

5.2 Exploration Experiments Mountain Car 55

5.3 Exploration Experiments Racing Car 61

5.4 Exploration Experiments Lunar Lander 66

6 CONCLUSION AND FUTURE WORK 73

REFERENCES 74

LIST OF TABLES

Table Page

4.1 Discrepancy estimates for the four methods described 53

5.1 Numerical results for the amount of steps taken to reach the goal for the mountain car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck. 60

5.2 Numerical results for the amount of time taken in seconds per episode for the mountain car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck. 61

5.3 Numerical results for the amount of reward received per episode for the race car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck. 65

5.4 Numerical results for the amount of time taken per episode for the race car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck. 65

5.5 Numerical results for the amount of episodes taken to receive a reward of 5 for the lunar lander problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck. 71

5.6 Numerical results for the amount of time taken per episode for the lunar lander problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck. 72

LIST OF FIGURES

Figure Page

2.1 [9] Interaction between an agent and environment within a Markov Decision Process. 13

2.2 Grid Maze 15

3.1 1-dimensional Uniform spread 24

3.2 1-dimensional Non-Uniform spread 25

3.3 Diagram for $P((x \in A) \cap (X < Y)) + P((x \in A) \cap (X > Y))$ for $x = .5$ 27

3.4 Diagram for $P((x \in A) \cap (X < Y)) + P((x \in A) \cap (X > Y))$ for $x = .3$ 28

3.5 Graph of $P(x \in A) = 2x - 2x^2$ 29

3.6 Graph of $P((X < x) \cap (x < X + Y) \cap (X + Y < 1))$ 31

3.7 Graph of $P(x < (X + Y - 1) \cup (X < x) \cap (X + Y > 1))$ 32

3.8 1-dim discrepancy estimation method under event B. 33

3.9 1-dim discrepancy estimation before method application under event B^c. 34

3.10 1-dim discrepancy estimation method under event B^c. 34

3.11 Hyper-rectangles that are allowed to wrap around the graph for discrepancy estimation. 35

3.12 Halton Sequence generated by using the co-primes of 2 and 3 37

3.13 Pseudo-random Sequence 38

4.1 Uniform action selection method to demonstrate the principle of selecting current actions that are dissimilar to previously selected actions 42

4.2 Plot of points generated by a uniform selection process. 42

4.3 Increased uniformity but decreased efficiency due to using the "Brute Force" action selection method. 44

4.4 Over-represented extreme values due to using the "electrons in a field" action selection method. 46

4.5 Under-represented extreme values due to using the updated "electrons in a field" action selection method 48

4.6 Result of using "Gradient Descent" action selection method with step-size = .1, decay rate = .7, and $\epsilon = .001$ 51

5.1 Visual representation of the mountain car problem 56

5.2 Comparison of the amount of steps taken to reach the goal of the mountain car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. Lower values are better. . . . 57

5.3 Comparison of the amount of time taken in seconds per step for the mountain car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. 58

5.4 Visual representation of the race car problem. 62

5.5 Comparison of the amount reward received per episode for the race car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. 63

5.6 Comparison of the amount time taken in seconds per episode for the race car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. 64

5.7 Visual representation of the lunar lander problem. 67

5.8 Comparison of the amount episodes taken to receive a reward of 5 for the lunar lander problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. 69

5.9 Comparison of the amount of time taken per episode for the lunar lander problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. 70

CHAPTER 1

INTRODUCTION

As technology increases ever forward at an incredible pace it can feel like the science fiction of yesterday is becoming the science reality of today. What once was thought to be impossible is now commonplace due to the advent of new technology. In particular, computers have revolutionized the ways in which we interact with others and our world in general. Computers have advanced so far that we are even trying to teach them how to learn.

To teach anything how to learn is very difficult. If you have ever trained an animal to do a trick or even tried to teach a child a skill, then you know the struggle of teaching. Maybe you have even tried to learn a difficult skill yourself and have found it far more intricate than you anticipated. However, you are highly equipped to change your thinking and learn something new. You are equipped with sensors all throughout your body that can give you immediate feedback if what you are doing is good or bad. If you stub your toe by turning a corner too quickly, you receive a feedback of pain. This tells you that you still need to refine your skills when it comes to moving around obstacles. Maybe a delicious morsel you cooked titillates your tastebuds and gives you immediate positive feedback that your cooking skills are at a satisfactory level. You were not taught that pain is bad or that things taste good. This is because you are not only equipped with the sensors to receive feedback, but also the ability to interpret their meanings. At a much higher and more abstract level you know if you are happy or sad. It may not be as clear what inputs need to change or stay the same to change these feelings, but these feelings are giving you feedback. What is more, you know just how to interpret them. In contrast, a computer is not so well equipped.

Computers need to be given sensors, which is a difficult task, but trivial in comparison to teaching a computer to interpret their feedback. Computers do not have our natural

ability to receive and interpret feedback about our current state of life. What's worse, since we were never taught it, it is exceedingly difficult for us to teach it. If a robot vacuum is cleaning the rug and bumps into the table how much negative feedback should you give it? Is it as bad as stubbing your toe or just brushing your shoulder against a wall by mistake? If you make the feedback extremely negative the robot may decide to never clean around tables to avoid the whole situation. If you make the feedback too mild then the robot might bump into things with no care and damage them or itself. Teaching a computer how to learn is a daunting task, but the rewards are incredible.

As far back as twenty-four years ago a computer was able to beat one of the world's best chess players. In 1997 a machine called Deep Blue and developed by IBM defeated Garry Kasparov, the world champion of chess at the time [3]. However, the field of machine learning had been in existence for decades before that event. In 1959, Arthur Samuel, an artificial intelligence expert, coined the term machine learning [8]. He also developed the Samuel Checkers Playing Program which, as you may have guessed, was a program that learned how to play checkers. From checkers to backgammon to chess, machine learning would be employed to master all of these games [10]. That was just the beginning for the field of machine learning. We have progressed it to new heights. Forget about a robot that can vacuum your carpet for you, we are all the way to self-driving cars! This technology is integrated all around us. As this field moves forward it is important that we move forward with it and help it develop as much as we can. The topic of this thesis will be the specific branch of machine learning called reinforcement learning and specifically how to optimize aspects of its learning.

Reinforcement learning deals with programs, robots, agents, etc. who will interact with an environment around them. Through their interaction, they will gather data through their experience. With this data they will begin to "learn" and understand what they can do in this environment, and the relative "goodness" of their actions. These actions must

be chosen somehow. They could be picked randomly, or through some sort of design. No matter what the case, there must be a balance between how this agent explores its environment to "learn" and then exploits its learning to pick "good" actions.

There are many ways to map out actions for the agent, and many potential hindrances that can arise from action selection. For instance, depending on how the selection is set up the same action could be picked multiple times in a row. It could take a long time to see all of the actions if this were allowed, and thus take longer for the agent to learn all it can do. So devising a good selection method can be tricky. One method is to generate sequences. Sequences can range vastly in complexity, and there are nearly limitless ways to produce them. For example, some are generated so that every subsequent point picked in the sequence differs as much as possible from the previous points picked. We could say this would be a "Low Discrepancy" sequence. This would be an interesting method to generate actions with for the agent. This is because as it tries one action, its next action will be drastically different than the first. Perhaps this would allow an agent to learn more quickly which actions are "good" by exposing it to more of the options it has to interact with the environment.

Before we dive into the details of the thesis, it will be important to first understand the basics of reinforcement learning with more detail than is presented here.

CHAPTER 2

REINFORCEMENT LEARNING BACKGROUND

2.1 DEFINITIONS

What is reinforcement learning and how does it differ from other forms of machine learning? To start with, reinforcement learning is distinct from supervised learning. Supervised learning is just as it sounds, the learner has an experienced or knowledgeable supervisor to guide it as it learns. The supervisor may be meticulously picking examples of correct behavior that make up a training set. Reinforcement learning does not make use of direct guidance. Reinforcement learning is also distinct from unsupervised learning which is more focused on finding structure concealed in non-categorized information. How reinforcement learning differs here is that it is not concerned with finding structure, but instead on maximizing a reward feedback. The development of these concepts will follow that of [9] for the remainder of the chapter.

Now let us focus on the detail of what defines reinforcement learning and what it can do for us. First, there are several terms that will be defined right at the start.

- Markov Decision Process - A stochastic selection process that satisfies the Markov property. We will denote this as MDP.
- Agent – The one that learns and decides what actions to take from state to state.
- Environment – Everything that is outside of the agent which the agent can interact with.
- State – Any set of criteria that describes a state of being for the agent. We will denote a specific non-random state as s.
- State Space - The space containing all possible states. We will denote the state space as $\mathcal{S}$.

- Action – Any act that the agent is allowed to perform. We will denote a specific non-random action as a.
- Action Space - the space containing all possible actions. We will denote the action space as $\mathcal{A}$.
- Reward – The measure of the short-term goodness of behavior of the agent. This is a random variable whose distribution depends on state, action, and next state. We will denote a specific non-random reward as r.
- Value – The measure of the long-term goodness of behavior of the agent.
- Exploration – Taking an action that is not necessarily optimal to learn more about the environment.
- Exploitation – Using past experiences to take an action that is considered optimal.
- Policy – A map from states to action that govern the behavior of the agent.

A MDP is a 3-tuple made up of states, actions, and probabilities. It is a stochastic process that provides a framework for a selection process. To understand this more thoroughly it will be helpful to have a diagram, look at Figure 2.1.

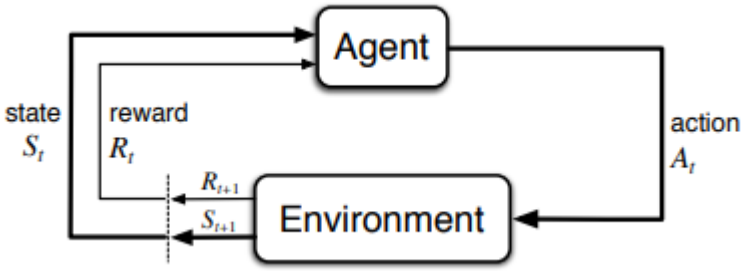

Figure 2.1: [9] Interaction between an agent and environment within a Markov Decision Process.

We stated above that the agent is the learner. The agent will be constantly interacting with the environment. As the agent interacts it will receive a reward based on the action it takes. We will think of these interactions as happening at discrete time steps $t = 0, 1, 2, 3, ...$ At each time step, t, the agent will receive some representation of the environments state, $S_t \in \mathcal{S}$. From this the agent will select an action $A_t \in \mathcal{A}$. One time step later the agent will receive a reward R_{t+1} for its action, and will move to state S_{t+1}. The process repeats from state S_{t+1}. To simplify the notation, let s be the current state and s' be the next state. Let the action we select be a. Then this process is governed by the MDP. The MDP is a collection of objects, specifically the following objects: $\{S_t, A_t, P(s', r|s, a)\}$.

Definition 1. Let s, a, and r be the current state, action and reward respectively and let s' be the next state. Then $P(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$ is the probability of moving to state s' and receiving reward r given that the agent started in state s and chose action a.

Let us look more in depth at the agent and environment. Firstly, an agent can be hardware or software. It can be a physical robot or a computer program or anything that has the above standards for being a reinforcement learning agent. The line that divides the agent from the environment should be considered carefully. For example, in an animal, its nerves, muscles, and bones would be considered part of the agent. However, in reinforcement learning these will be considered part of the environment. In a robot we will consider the motors, sensors, and mechanical links to be part of the environment that the agent can interact with.

States make up all of the information one would need to know to completely define what is happening to an agent regarding a task. States can be a physical description of where the agent is or an indication from sensors that the agent possesses. Think of a grid maze, where every grid represents a space where an agent could stand. To help, observe Figure 2.2.

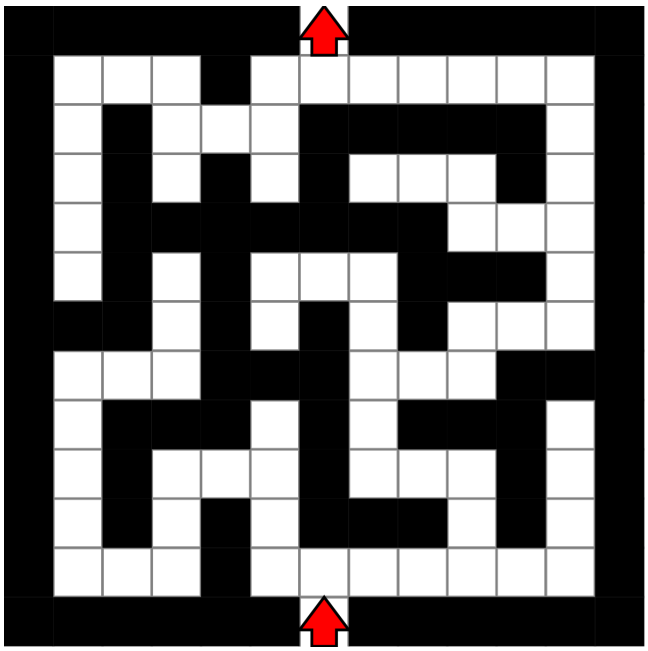

Figure 2.2: Grid Maze

The state would simply be what grid the agent is in the maze. In a finite example like this, we could label every grid with a number and that number could represent the position of the agent. Let us say there are N grids in this case and label the first as 1. Then the state could be completely described if we know what grid it is on.

$$\mathcal{S} = \{1, 2, 3, ..., N\}$$

For this example, an action would be moving up, down, left, or right. Then the action space could be described as follows.

$$\mathcal{A} = \{\uparrow, \downarrow, \leftarrow, \rightarrow\}$$

An agent will have all of the actions available to choose from per state, unless the state restricts the actions. In the maze if the agent is in a state where certain actions will push it into a wall, those actions may be restricted in those states.

The agent will receive a reward for each action taken. These rewards are immediate

feedback for the agent after it takes an action. We can compute the expected reward for a state-action pair with the following function:

Proposition 2.1. *The expected reward for a state s and action a can be found by summing over the possible rewards and future states. This can be expressed as follows*

$$r(s,a) = E[R_t|S_{t-1} = s, A_{t-1} = a] = \sum_{r\in\mathcal{R}} r \sum_{s'\in\mathcal{S}} p(s,r|s,a).$$

This does not tell us the long-term reward of taking an action, only the short-term goodness of selecting that action. The lower the reward compared to the other rewards received the worse that action is assumed to be by the agent. A relatively smaller reward implies that the action associated to this reward should be viewed as one to avoid. It is very important to incentivize actions correctly. It is possible to incentivize sub-tasks that you believe will help the agent complete the overarching task. However, if done improperly this can result in the agent only completing the sub-tasks and never actually completing the main task. Additionally, depending on how actions are rewarded the actual goal of learning may never be achieved. Before we talk about that let us discuss value. If you string together many rewards you arrive at the following:

Definition 2. Let R_{t+i} represent the reward at i steps in the future. Let $\gamma \in [0,1]$ be a parameter called the discount rate. Then the **total discounted reward** G_t **at time** t is

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}.$$

This can be taken a few steps further letting the total discounted future reward depend on the next reward plus the future total discounted reward after that. This breaks down as follows:

Proposition 2.2. *Let $\gamma \in [0,1]$ be the discount factor. The return at time t can be expressed as immediate reward plus discounted return at the following time,*

$$G_t = R_{t+1}+\gamma R_{t+2}+\gamma^2 R_{t+3}+\gamma^3 R_{t+4}+... = R_{t+1}+\gamma(R_{t+2}+\gamma R_{t+3}+\gamma^2 R_{t+4}+...) = R_{t+1}+\gamma G_{t+1}.$$

Now that we have an idea of how the states, actions, and rewards interact within the MDP, the next question is to ask how are the actions selected? This is where the policy comes into play.

Definition 3. The policy, denoted π is a mapping of the state-action space to the interval $[0, 1]$.

$$\pi : S \times A \to [0, 1]$$

this gives the probability of selecting actions.

If all of the probabilities are either zero or one the policy can be seen as a mapping of states to the probability of selecting an action. If an agent is following policy π at time t then the probability $A_t = a$ and $S_t = s$ is $\pi(a|s)$. Now that we have the idea of a policy we can discuss value.

Value is very closely related to rewards; the difference is that this will represent long-term goodness instead of short-term. This is not the immediate worth of taking an action, but instead what benefits that action will lead to far in the future. The value is what the agent will try to optimize. Since value is determined by reward, and rewards are gained after taking an action, and actions are taken according to a policy, then value depends on the policy being used. It can be expressed as follows.

Definition 4. Let π be a policy. The value of a state s under π is

$$v_\pi(s) = E_\pi[G_t \mid S_t = s] = E_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s].$$

If we take this one step further we arrive at the Bellman equation.

Proposition 2.3. *(Bellman's Equation) [4] Let π be a policy, then*

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \quad \textit{for all} \quad s \in S.$$

This equation is important because it expresses a relationship between the value of a state and the values of its successor states. However we view it, we can see that value is our expected total return of discounted rewards under policy π given we start in state S. In a discrete state and action space we can calculate and optimize this value function. However, in a continuous state and action space optimizing this can become very challenging. In the simplest sense, since the value depends on the policy, finding an optimal policy will result in a superior value function. Our overall goal is to optimize this value function. If an agent has optimized this function than it is always choosing the action that gives it the greatest long term reward.

2.2 Examples and Intuition

States can be more complicated than what we discussed above. For example, consider a self-parking car. One thing that we need to know for the state is its speed. Now the speed can be zero and that is completely acceptable, but we will need to know its speed for every state. We will also need to know its acceleration and what direction it is moving. We need to know the current position which can tell us where the car is relative to the parking space. We should also have sensors that tell us how far away any nearby obstacle that the car can hit is from any side of the car. So now any given state for this agent is comprised of multiple information.

$$\mathcal{S} = [\text{position, speed, acceleration, direction, distance to obstacles}]$$

All of these variables combined make up one state for this agent. If any one of them change then the agent is now in a new state and the action it takes should be adjusted accordingly. An action for this example could be moving the steering wheel or accelerating the car backwards or forwards.

Even though there are many different problems with unique states and actions; there

are basic policies that can be used for exploration, exploitation, or a mixture of both. A classic example of a policy is the greedy policy. This policy always has the agent select an action that it currently knows gives it the highest value. This is a simple policy that has advantages and disadvantages. At first the policy will assume every action has a value of zero. In other words it values all actions the same at the start. To demonstrate let us create an action space with three actions and let us say we know the true values of each. We will let $\mathcal{A}$ be our actions with values being $\mathcal{V}$ respectively.

$$\mathcal{A} = \{1, 2, 3\}$$

$$\mathcal{V} = \{-5, 2, 20\}$$

At first the policy will assume every action has a value of zero. In other words it values all actions the same at the start. So it has a guess of the values, let that be denoted $\mathcal{V}_{\text{guess}}$.

$$\mathcal{V}_{\text{guess}} = \{0, 0, 0\}$$

Let us say the agent picks the first action and receives a reward of -4. Now the values update.

$$\mathcal{V}_{\text{guess}} = \{-4, 0, 0\}$$

Under this policy now the agent will not pick the first action because it has a lower value than the other options. So let us say the agent picks again and between the second and third action randomly picks the second, and receives a reward of 2. The values update again.

$$\mathcal{V}_{\text{guess}} = \{-4, 2, 0\}$$

Now under the policy it must pick the second action, because it is the best action according to the values. As long as the value of the second action stays larger than zero the agent will never try the third action. This is an issue because the third action has the best true value, but the agent may never realize this due to the policy it is using. That is one drawback

to using the greedy policy. The agent may get stuck in a loop once it has found a good action to take, but it may not be the best action. There are many approaches to remedy this problem, we will discuss one later called the epsilon greedy policy.

A policy will have to choose what is optimal between two things at any given time, exploration or exploitation. These are as straightforward as they sound. If an agent is exploring it is not necessarily picking the action it believes to be optimal, but instead it is trying to explore its environment more. The hope is that through exploration the agent can find an optimal action for a given state. If an agent decides instead to exploit then it uses what experience it has already gained to take what it ranks as the most optimal action. There is an intricate and delicate balance between these two. An agent who never explores is very unlikely to find the true best action for its state. However, an agent who never exploits will likely find the action, but is unlikely to select it. The balance is in doing enough exploration that its exploitation of the experience is effective.

Now we should distinguish between a on-policy and off-policy approach. An on-policy approach means that the agent's behavior is determined by the policy. As it learns more it updates its policy to try to obtain an optimal policy. An off-policy approach still uses a policy, but the agents actions are not solely dictated by the policy. Let's say that it has a greedy policy, and is watching what happens when it follows this policy. It updates the value of states based on what it is seeing. If it believes a move is more valuable even if the policy states it should pick the greedy move, and off-policy agent can make actions that do not follow the policy. Off-policy methods tend to make the agent more flexible, as it can adapt more quickly than following a rigid policy.

2.3 CONTINUOUS CASES

Here we will distinguish between continuous states and actions, and discrete states and actions or any combination thereof. Let us look at the example of the car, and say the action for moving the car can be described with three discrete numbers in the following way.

$$\mathcal{A} = \{-1, 0, 1\}$$

Let's say that negative one means the car will reverse at maximum speed, zero means it will not move, and one means it will go forward at maximum speed. This would be discrete. Now let us describe it in this continuous way.

$$\mathcal{A} = (-1, 1)$$

Now the action can be any number between negative one and one. A positive number will move the car forwards, but a smaller positive number will do this with less speed than a greater one. Likewise with negative numbers, but for reverse. The closer to zero the number is the less force is applied to the car. In contrast to the discrete approach which has three actions, the continuous approach has infinite actions. States can be described the same way, either being continuous or discrete.

It is of great importance to note that an MDP with discrete spaces can be solved in a tabular method. A table of all possible combinations can be made, for the agent to map out and optimize. Although this can be very complicated for some problems, this is still a method that will lead to its solution. On the other hand a continuous problem cannot be solved in this manner. These problems must be approached with function approximators. These will approximate the policy and value function for the problem. Then as this function is optimized, the policy and value should be optimized. A common way to approximate policy and value is with a neural network. This is a complicated and sophisticated method that works similar to the human brain. This is far more complicated than a discrete

approach, and does not describe the problem as perfectly. However, if the function approximates well, then given enough time the agent should be able to find nearly optimal values and policies.

2.4 Efficient Exploration

Now that the basics are covered, let us touch on optimization. As you may have guessed there could be countless policies, and some may be superior to others. In fact, there is always an optimal policy and there is a method to obtain it only in the discrete case. Let it be sufficient for our purposes to know that there is an optimal policy and that we can find it. However, it can be a major drain to efficiency to find the best policy. For this thesis we will use staple policies, like the epsilon-greedy policy.

This policy is similar to the greedy policy, with a small adjustment. We pick a number ϵ and determine if we will explore or exploit based off of this number. If ϵ is $.1$ we will explore ten percent of the time. If we set it to be higher than we will incur more exploration. We can set it to start at a high rate like $.9$ and decay over time. So that in the beginning the agent will explore its environment far more than it exploits. However, as time goes on ϵ decays and the agent explores less and less. Because of all the exploring it did at the start, when it starts to exploit it should be using optimal or near optimal actions.

Traditionally, when the agent explores with this policy it randomly picks an action, typically with independent uniform selection of the available actions. You could think of this as a "with replacement" selection process. Each new selection is independent of the previous selections. However, perhaps even at this stage the learning could be improved if the exploration method was altered. What if instead of randomly picking, it remembered the last selections it made when exploring and selected an action that differed from previous selections. Something more akin to "without replacement" selection, where the new selection was dependent on the previous selections in some way. Would this allow its ex-

ploration, and therefore overall learning, to be more productive? Can we devise a way to select actions that differ from previously selected ones? Perhaps most importantly can we find a way to measure if the actions we are selecting are different from previous ones? Let us explore this idea in more depth later on in the thesis.

CHAPTER 3

LOW DISCREPANCY SEQUENCES

3.1 DISCREPANCY

At the end of the previous chapter we discussed the idea of actions that differ from ones that have been previously selected. To discuss this we will need to understand the idea of discrepancy. Discrepancy will be formally defined as the measure of deviation from an ideal uniform distribution. This will be a way to measure how well data has been generated from a sequence when compared to a uniform distribution. In an equation discrepancy can be viewed as the following.

Definition 5. [5] Let $\{x_n\}_{n=1}^{N}$ be a sequence taking values in the interval $[0, 1]$. Let $A()$ count the number of terms x_n where $\alpha < x_n < \beta$, then:

$$D_N = D_N(x_1, ..., x_N) \ = \sup_{0 \leq \alpha < \beta \leq 1} \left| \frac{A([\alpha, \beta); N)}{N} - (\beta - \alpha) \right|$$

This is a formula for a one-dimensional representation. We will generalize it to multiple dimensions soon, but before we do let us explain the concept for one and two dimensions with graphs. First, we will look at two number lines with points representing the data.

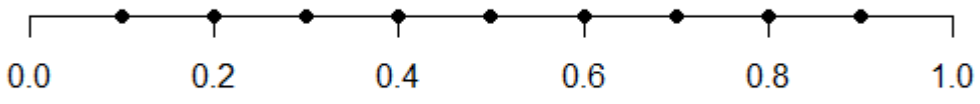

Figure 3.1: 1-dimensional Uniform spread

Figure 3.2: 1-dimensional Non-Uniform spread

Clearly, the Figure 3.1 is more uniform than Figure 3.2. Now let's say that we have $\alpha = 0$ and $\beta = .2$. Then a component that will factor into the discrepancy calculation inside the supremum for the first sequence is $|\frac{2}{9}-(.2-0)| = .0222$. A component that will factor into the discrepancy calculation inside the supremum for the second sequence is $|\frac{4}{9}-(.2-0)| = .2444$.

It is important to know that the discrepancy formula requires a supremum. It is therefore useful theoretically, but not practical for determining the discrepancy of a given sequence. Therefore to estimate the discrepancy, we will repeat the above process thousands of times with random $\alpha's$ and $\beta's$ and take a maximum. Any segment of the sequence that has a high cluster of points that is too dense or is missing too many points will show a large discrepancy. So long as enough segments of the sequence are taken the discrepancy test should catch any parts that do not align with a uniform distribution. This will be our practical approach to estimating discrepancy. Let us look at the multi-dimensional formula for discrepancy.

Definition 6. [5] Let J run through all sub-intervals of I^k of the form

$J = (x_1, ..., x_k) \in I^k : \alpha_i \leq x_i < \beta_i \quad \text{for} \quad 1 \leq x_i \leq k$

and let λ represent the k-dimensional Lebesgue measure, then:

$$D_N = D_N(x_1, ..., x_N) \ = \sup_J \left| \frac{A([J); N)}{N} - \lambda(J) \right|$$

One difficulty in estimating discrepancy of a sequence is generating the α and β for the segments to be checked. A natural idea is to let X and Y be uniformly distributed in the unit interval, and then define α to be the smaller of the two and let β be the larger. This procedure is biased. The segments will favor the middle of the graph, but will very rarely if ever sample the extreme values at either end. To demonstrate this formally, let us state a proposition.

Proposition 3.1. *Let X and Y be independently and identically distributed uniform random variables between zero and one. Let $A = [\min(X, Y), \max(X, Y)]$ Then for $x \in [0, 1]$ we have*

$$P(x \in A) = 2x(1 - x).$$

That is, the inclusion probability is not constant with respect to x.

Proof. Begin by partitioning according to whether X or Y is larger.

$$P(x \in A) = P((x \in A) \cap (X < Y)) + P((x \in A) \cap (X > Y)).$$

Figure 3.4 shows a graph of the above expression with $x = .5$.

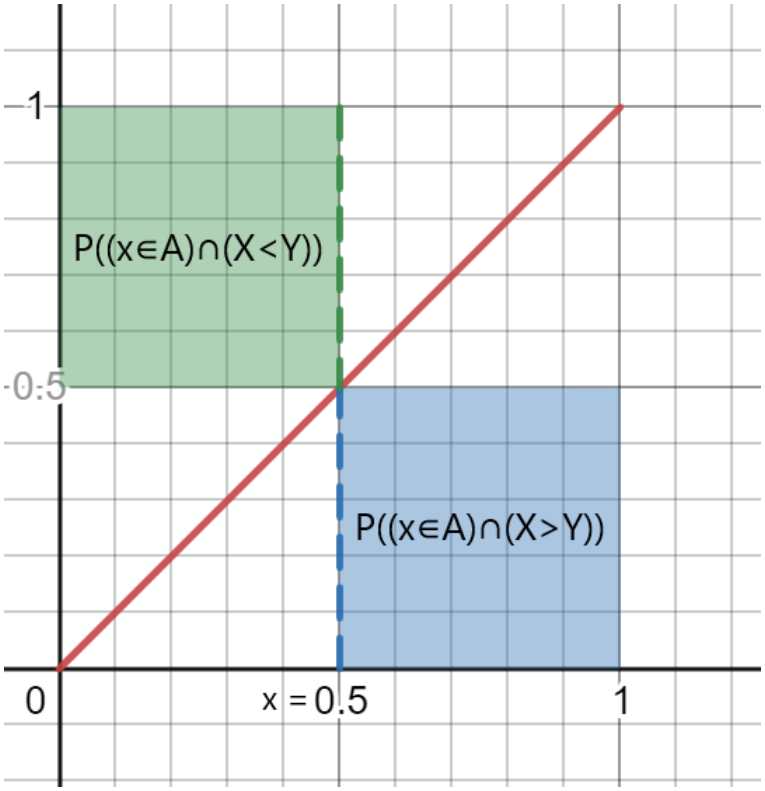

Figure 3.3: Diagram for $P((x \in A) \cap (X < Y)) + P((x \in A) \cap (X > Y))$ for $x = .5$

The green shaded region in the top left corresponds to $P((x \in A) \cap (X < Y))$ and the blue shaded region in the bottom right corresponds to $P((x \in A) \cap (X > Y))$. We can see that each shaded region has an area of $x(1 - x)$. We can also see the regions are symmetric, and the whole area does equate to $2x(1 - x)$, which finishes the proof. □

Even from this we can start to see a bias. Notice what happens if we have the same graph but $x = .3$.

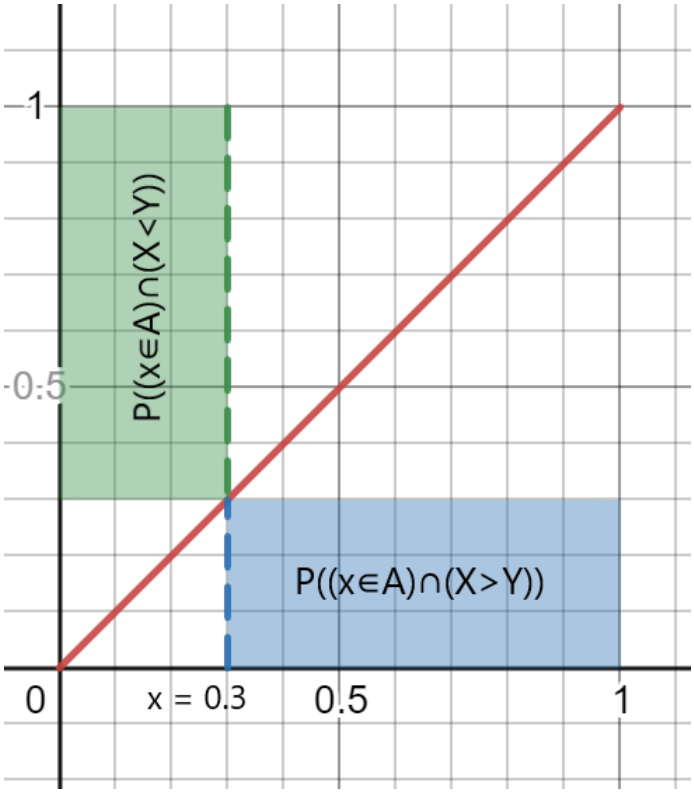

Figure 3.4: Diagram for $P((x \in A) \cap (X < Y)) + P((x \in A) \cap (X > Y))$ for $x = .3$

When $x = .5$ we have a total area of $.5$, but when $x = .3$ we have a total area of $.42$. As x moves closer to the upper or lower boundary the area will decrease.

Returning to our main problem this all leads to

$$P(x \in A) = 2x - 2x^2$$

which is maximized at $x = \frac{1}{2}$.

Where is this equation maximized? We could take first derivative and set it equal to zero to arrive at the answer $\frac{1}{2}$. It will be simpler to look at the graph of the function, observe Figure 3.5.

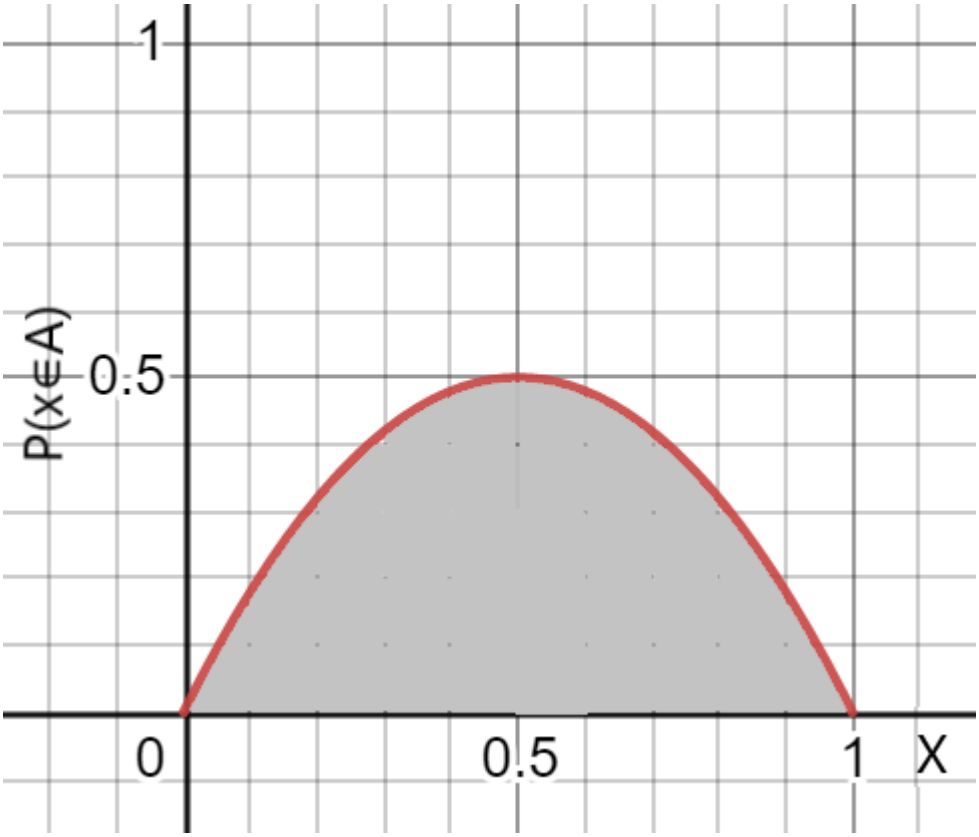

Figure 3.5: Graph of $P(x \in A) = 2x - 2x^2$

It is clear that the area, and thus probability is maximized in the middle of this graph. This demonstrates a bias in this approach, and confirms that this set up favors values in the middle over extreme values.

To achieve a method that has a constant probability with respect to x throughout the whole region we propose this theorem.

Theorem 3.2. *Let X and Y be independent and identically distributed uniform random variable from zero to one. Set $L = X$, $U = X + Y$ and define*

$$A = \begin{cases} [L, U] & U \leq 1 \\ [0, U-1] \cup [L, 1] & U > 1. \end{cases}$$

Then we have the following.

a) For $x \in [0,1]$, $P(x \in A) = \frac{1}{2}$, a constant that does not depend on x.

b) If in d dimensions, let X_i, Y_i, L_i U_i, and A_i, for $i = 1, 2, ..., d$, be defined as before but independently for each value of i. Define $A = \cap_{i=1}^{d} A_i$. Then for $x \in [0,1]^d$, $P(x \in A) = (\frac{1}{2})^d$.

Proof. First we will prove part a and when we move up a dimension for part b, we can view the next dimension as independent from the last dimension. In this way we only need to prove this method is correct in one dimension, then we can repeat the process as many times as the number of dimensions we have. Now note, $L \in (0,1)$ and $U \in (0,2)$. In A let the event that $U \leq 1$ be called B and the event $U > 1$ be B^c. Now we have the following.

$$P(x \in A) = P((x \in A) \cap B) + P((x \in A) \cap B^c)$$

Begin by evaluating $P((x \in A) \cap B)$.

$$P((x \in A) \cap B) = P((x \in [L,U]) \cap B) = P((L < x) \cap (x < U) \cap B)$$

$$= P((X < x) \cap (x < X + Y) \cap (X + Y < 1))$$

Figure 3.6 shows a graph of this expression.

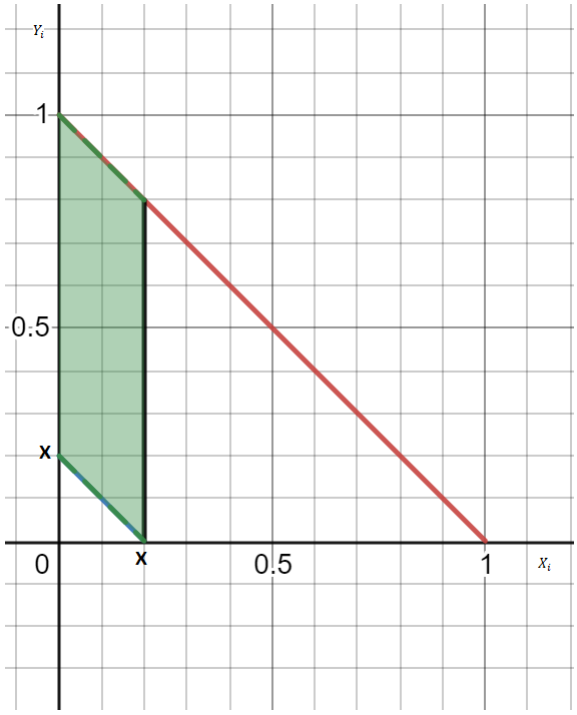

Figure 3.6: Graph of $P((X < x) \cap (x < X + Y) \cap (X + Y < 1))$

This Figure 3.6 depicts the X and Y axis with the green shaded region depicting where x lies within our given intersections. Now note, the height of the shaded region is clearly $1 - x$ and the base is x. The height times the base is equal to the area of a parallelogram. Therefore the shaded region is $(1 - x)x$. Thus,

$$P((x \in A) \cap B) = (1 - x)x.$$

Now for the more complicated probability we will evaluate the expression $P((x \in A) \cap B^c)$.

$$\begin{aligned} P((x \in A) \cap B^c) &= P(x \in ([0, U - 1] \cup [L, 1]) \cap B^c) \\ &= P(x \in ([0, X + Y - 1] \cup [X, 1]) \cap (X + Y > 1)) \end{aligned}$$

$$= P(x < (X + Y - 1) \cup (X < x) \cap (X + Y > 1)).$$

The above expressions graph is depicted in Figure 3.7.

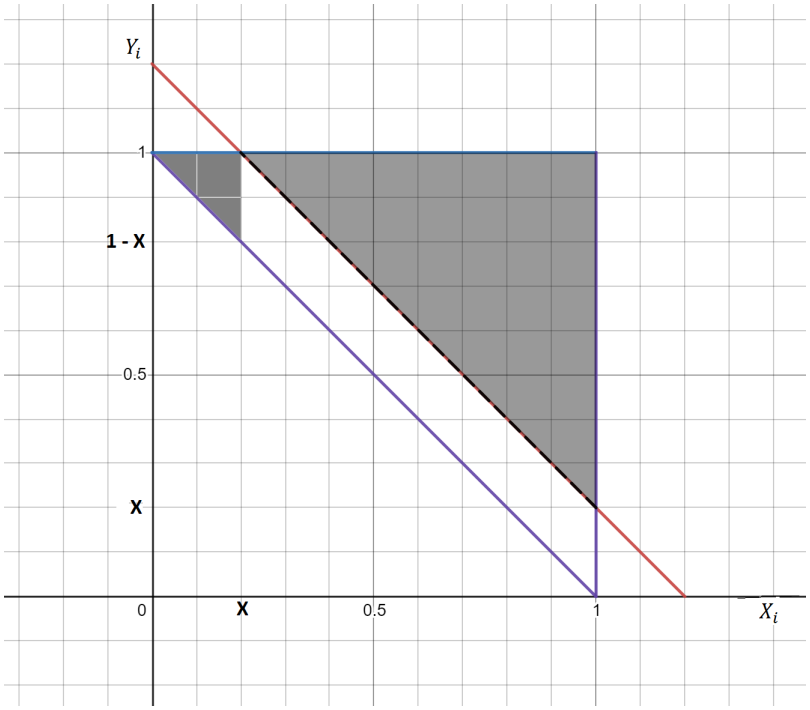

Figure 3.7: Graph of $P(x < (X + Y - 1) \cup (X < x) \cap (X + Y > 1))$

The grey shaded region describes the probability that we are interested in for our problem. Now note this is made of two triangles. The large triangle has a height of $(1 - x)$ and a base of $(1 - x)$. Therefore, it has a area of $\frac{1}{2}(1 - x)^2$. Now the smaller triangle has a height of x and a base of x. Therefore, it has a total area of $\frac{1}{2}x^2$. So the total area for both shaded regions is $\frac{1}{2}(1 - x)^2 + \frac{1}{2}x^2$.

Now let us total the full probability that we are interested in

$$P(x \in A) = x(1 - x) + \frac{1}{2}x^2 + \frac{1}{2}(1 - x)^2$$

$$= x - x^2 + \frac{1}{2}x^2 + \frac{1}{2} - x - \frac{1}{2}x^2 = \frac{1}{2}.$$

□

This is the proof that we needed. For if our action selection has no bias we need this probability to be equal to a constant. That means that no matter where x is it will have the same chance of being selected, and therefore we favor no part of the region over another. In our case the probability that x was in A is equal to $\frac{1}{2}$ in the first dimension. In general for any dimension d the probability would be $\frac{1}{2}^d$.

Now that we have an unbiased manner of selecting points for estimating discrepancy let us see the details of this method. In one dimension this method can be visualized with a line segment. For the event B where $U < 1$ we have Figure 3.8.

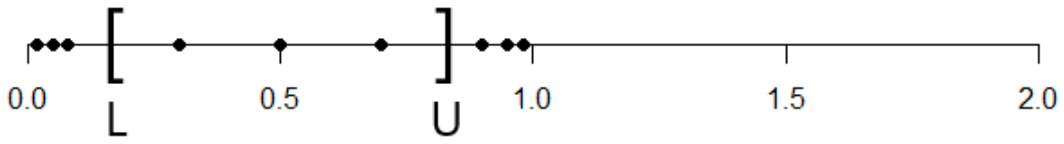

Figure 3.8: 1-dim discrepancy estimation method under event B.

The brackets section off the interval from L to U and all the points inside this interval would be used to calculate a component of the discrepancy estimation. Under event B this looks like a normal uniform selection. Now let us see what happens under event B^c where $U > 1$ in Figure 3.9.

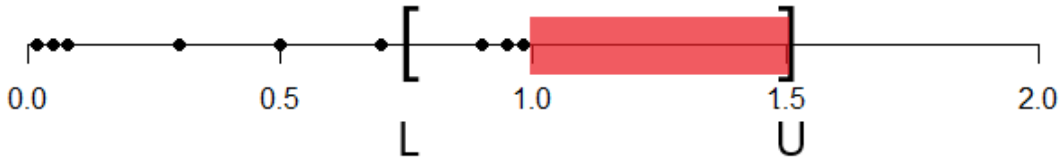

Figure 3.9: 1-dim discrepancy estimation before method application under event B^c.

Now U, our upper limit, extends past the upper boundary of our segment. All of the red shaded region is not a valid section to select points from. This is where our method comes into play. It will now create two valid intervals from which to select points, specifically $[0, U-1]$ and $[L, 1]$. We can see this represented in Figure 3.10.

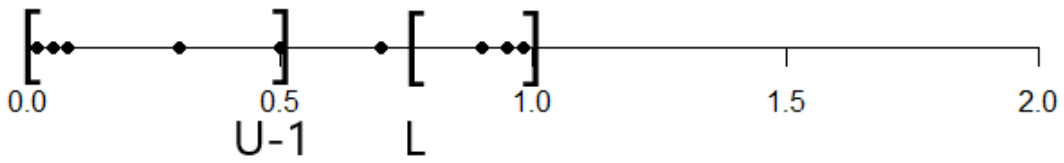

Figure 3.10: 1-dim discrepancy estimation method under event B^c.

What the method has done is created an interval from $[L, 1]$ and taken the red shaded region from Figure 3.9 and started it at 0 creating the interval $[0, U-1]$. This method takes any part of our generated interval that extends beyond a boundary and wraps it around to the start of the segment. With this method we will not under represent any of the extreme data points.

For two dimensions or higher, instead of taking segments of a line and finding the

points it contains, we take the number of data points within a randomly generated hyper-rectangle. The hyper-rectangle's size and position are generated randomly just like α and β were, and again this will help catch any discrepancy in the graph.

The method we propose allows any hyper-rectangle that would hit a boundary to act as if it could teleport to the opposite side of the graph and continued from there. Now we can sample any extreme values that we were missing in the previous bias method regardless of dimensions. To demonstrate, look at Figure 3.11.

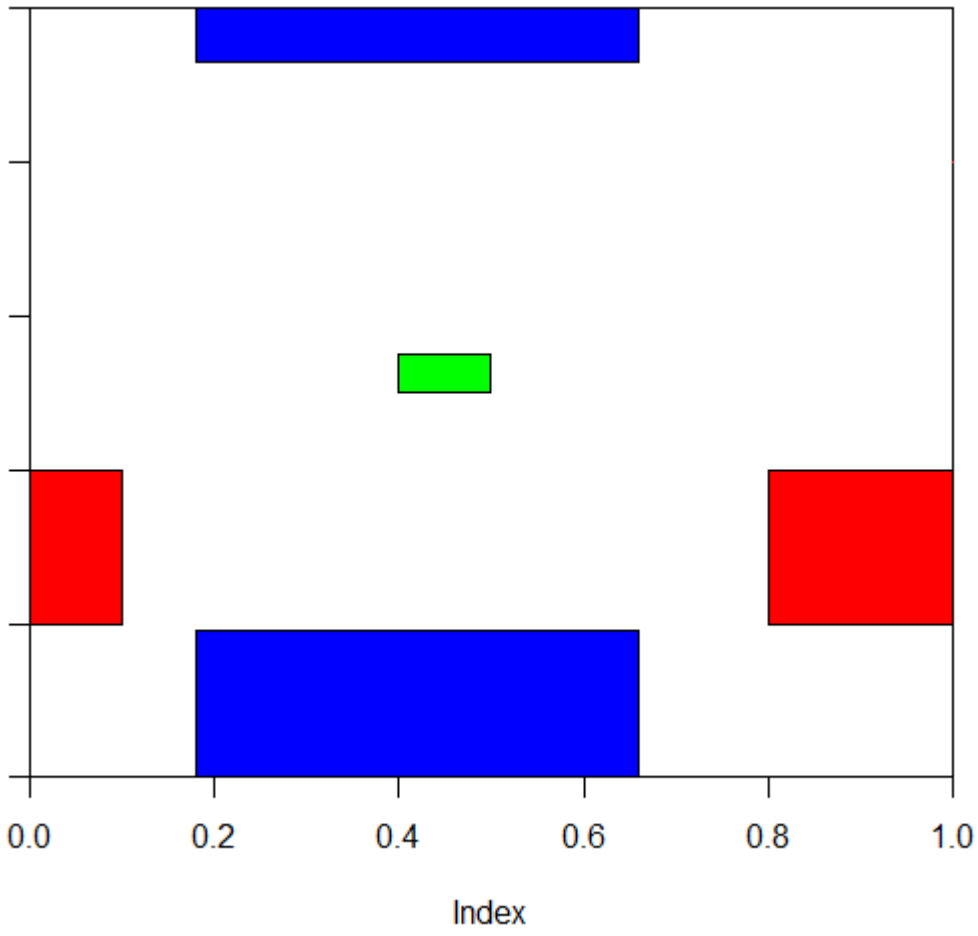

Figure 3.11: Hyper-rectangles that are allowed to wrap around the graph for discrepancy estimation.

Notice all of the hyper-rectangles are randomly sized and placed. Also, that the red and blue larger hyper-rectangles would have hit the edge so they appeared at the opposite side of the graph and continued. This method of selecting data by allowing the selection region to extend when it hits a boundary by continuing to select from the opposite boundary (or wrapping around) creates no bias in selection as detailed in the proof.

3.2 Low Discrepancy Sequence examples

There are many sequences that fit the criteria of low discrepancy. One such example would be the Van der Corput sequence. It is constructed by reversing the sequence of natural numbers of a certain base across the decimal place. For example in base ten, 1 would be flipped along the decimal place to become $.1$. Likewise, 10 would become $.01$. For the classic example we will use base two, and flip binary numbers along the decimal. Let's look at the first seven binary numbers, with leading zeros to emphasize the place values after reversing about the decimal.

$$[001, 010, 011, 100, 101, 110, 111]$$

Now reverse about the decimal to get the following:

$$[.100, .010, .110, .001, .101, .011, .111]$$

Converting these numbers to base ten to make the meaning of the sequence we are generating more obvious.

$$\left[\frac{1}{2}, \frac{1}{4}, \frac{3}{4}, \frac{1}{8}, \frac{5}{8}, \frac{3}{8}, \frac{7}{8}\right]$$

It is clear that these are filling in the unit interval in a very uniform, or in our terminology, low discrepancy manner. The data generated is not favoring any area of the graph.

Another famous example of low discrepancy sequence, is the Halton sequence. This sequence is connected to the Van der Corput sequence, but it pairs co-prime numbers together to generate the sequence. To see the details of this let us look at an example. First

generate the Van der Corput sequence for base two and three. We have already seen the beginning of this for base two, for base three after we convert back to base ten we would have the following:

$$\left[\frac{1}{3}, \frac{2}{3}, \frac{1}{9}, \frac{4}{9}, \frac{7}{9}, \frac{2}{9}, \frac{5}{9}\right]$$

Now let us make ordered pairs from the two sequences using only the co-prime among them. Here would be the first seven pairs.

$$\left[(\frac{1}{2}, \frac{1}{3}), (\frac{1}{4}, \frac{2}{3}), (\frac{3}{4}, \frac{1}{9}), (\frac{1}{8}, \frac{4}{9}), (\frac{5}{8}, \frac{7}{9}), (\frac{3}{8}, \frac{2}{9}), (\frac{7}{8}, \frac{5}{9})\right]$$

To show visually how the shape of the sequence generated by paring numbers this way, please look at Figure 3.12.

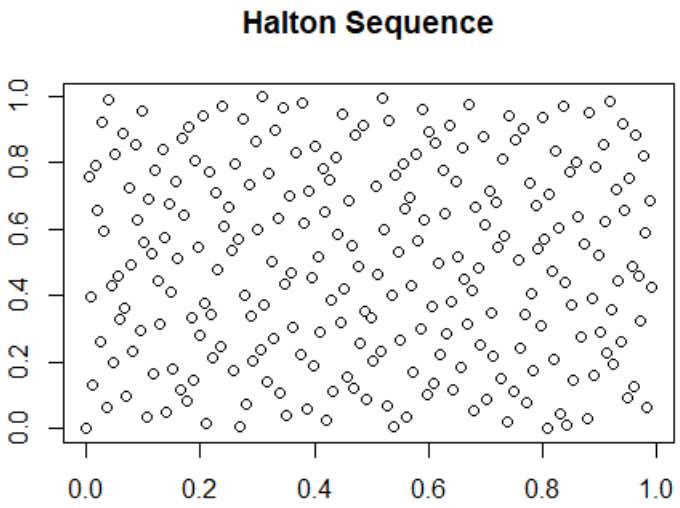

Figure 3.12: Halton Sequence generated by using the co-primes of 2 and 3

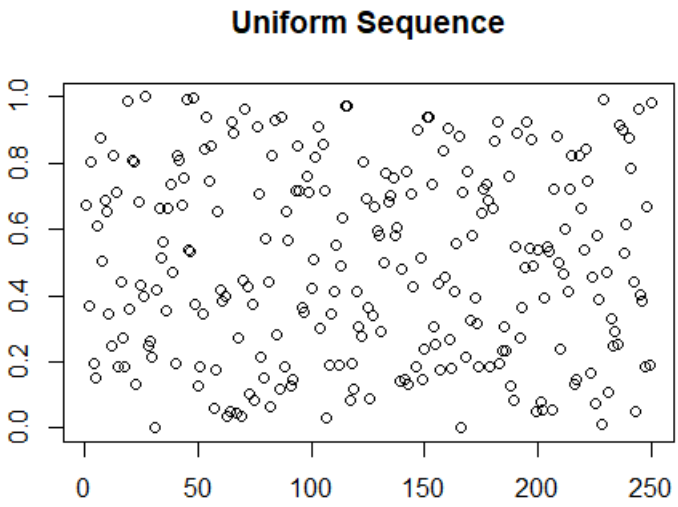

Figure 3.13: Pseudo-random Sequence

Figure 3.12 is generated with a Halton sequence whereas Figure 3.13 is made using a uniform sequence. It is clear that the top graph has lower discrepancy than the bottom. As you can see there are many approaches for generating low discrepancy sequences, but they assume that all dimensions are free. This does not readily apply to reinforcement learning.

3.3 Low Discrepancy Action Selection

To apply these sequences to selecting actions for reinforcement learning, we have to understand a state's relationship to actions. The state an agent is in cannot be selected by the agent. It can only select its actions, and it enters new states or states it has previously been in through those action. Therefore, if we look at the graph for the Halton sequence, and base two data represent the state space and the base three data represent the action space, the base two data cannot be picked. The base two data (the states) will not be chosen, but

based on that the base three data (the actions) will be selected.

Unfortunately, this makes these sequences not readily applicable to this problem. In addition, there is no guarantee that the state space and action space will be equal. You could have a 100-dimensional state space and a 1-dimensional action space. That means one-hundred of those are not free to pick from. They are not able to be selected and are determined from the actions taken. Therefore, for any arbitrary dimension of states and actions we will need to devise a way to create a low discrepancy method for selecting actions.

CHAPTER 4

LOW DISCREPANCY ACTION SELECTION

4.1 SET-UP

The intention of this thesis is to discover if a new action selection method can improve exploration in reinforcement learning and in what contexts it will succeed. If it is possible to find a method that creates a more efficient exploration stage for the agent in certain contexts, then in these contexts this should enhance any agent's success. Our main innovation will be to select actions by using some low discrepancy action selection (LDAS) method. We will have to develop a suitable method and when we do we will apply this selection process to classic benchmark reinforcement problems and policies. We will then compare it to existing standard exploration strategies and see if it is superior. Initially we will work in two-dimensional spaces so that we can investigate discrepancy visually.

As we develop a method we will want it to satisfy desirable properties. In our case, We will prioritize the following three properties of action selection.

- Choose actions so that the current state-action pair is as dissimilar from previous ones as possible.
- Boundaries of the space have non-zero selection probabilities.
- Action selection is reasonably computationally efficient.

When we have a method that meets these qualifications to our satisfaction we will need to test it compared to other methods. This will allow us to determine the relative effectiveness of our method for the specific problems we give it. One method of selection we will compare against is the uniform selection method.

4.2 Uniform Selection

A baseline to compare against is to uniformly and independently pick actions for each state for the agent. As a visual aid consider a two-dimensional state-action space. The horizontal axis will represent the state space and the vertical axis will represent the action space. Since this is the case, we will use s for the values on the horizontal axis and a for the values on the vertical axis. The s's are picked randomly, mimicking an environment that transitions to states uniformly, and from that we pick an a uniformly from the action space. This means once we have a s value we count that as fixed, and are only allowed to change the a value. For simplicity, we will restrict attention to the unit square.

An independent uniform selection method simply picks a random action from all available. It does not depend on previous actions selected. This makes the method very efficient, but can have some drawbacks. For one, you could select similar actions to those previously selected, and thereby take a longer time to learn how all of the actions interact with the environment. Another drawback is that a uniform selection method has a zero probability to select an extreme value and it compounds in higher dimensions. Think of a unit interval and you have a standard uniform distribution. Although 0 and 1 are in this interval the uniform pick has a zero probability of selecting them. Now go to the second dimension. You can think of this as two uniform picks (one for each dimension). The odds that both picks will be extreme values, i.e. $(0,0),(1,0),(0,1),(1,1)$ has a zero probability. This means that as the dimensions increase the actions that are represented by the corners of those dimensions will not be selected with a uniform selection process. In contrast, the method we create will have a non-zero selection probability for extreme values.

In discrete settings, LDAS can be based on the principle of not picking the same value twice until all values had been picked once. In continuous setting, the analogous principle is to prioritize values that are dissimilar from previous ones. To demonstrate how this is more desirable than a uniform action selection method, please look at Figure 4.1.

It is possible that an action similar to that of the previous time step is chosen. This results

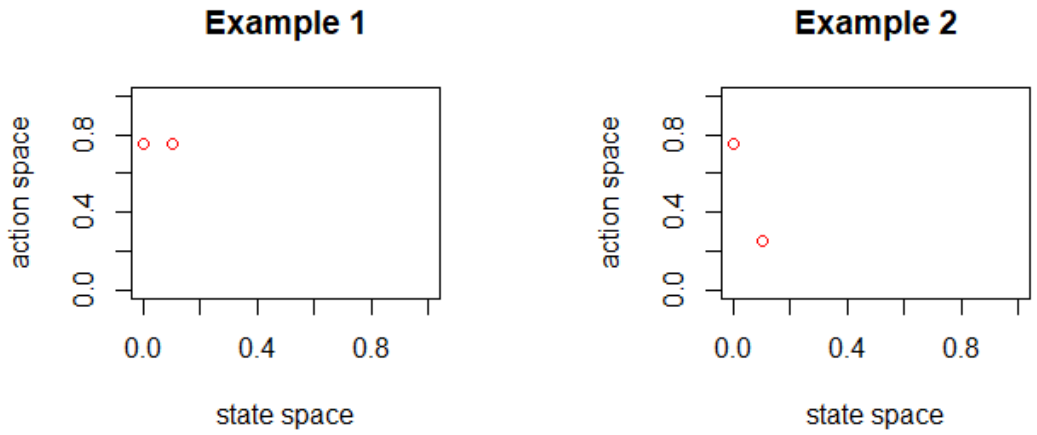

Figure 4.1: Uniform action selection method to demonstrate the principle of selecting current actions that are dissimilar to previously selected actions

in two similar state-action pairs, which is undesirable for the purpose of exploring the entire state-action space efficiently. Incentivizing a different action, such as in Example 2, could give new information about the transition dynamics and result in better overall learning. A plot of uniform selection process can be seen in Figure 4.2

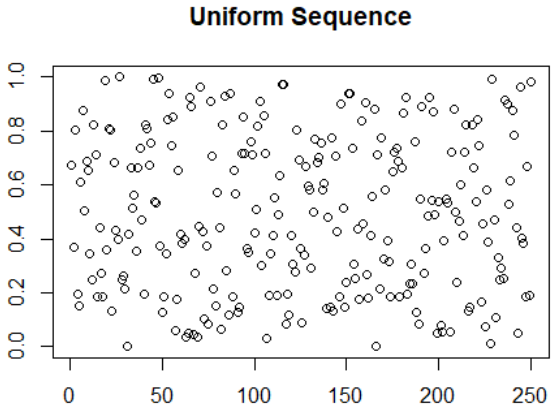

Figure 4.2: Plot of points generated by a uniform selection process.

4.3 LDAS CANDIDATE 1

This method will be considered a brute force method. It takes any new point that is placed and moves it (only along the 'a' axis) to the point furthest from all previous points. To do this it needs to calculate a huge matrix that tracks the position and from that the distance between all points. This search is along a fine grid. This is why we call this the brute force method. Although the results of this method are very good in achieving low discrepancy, it is computationally inefficient. Please observe Figure 4.3 below and note the following; the spread of the points is very uniformly even. Our discrepancy is near zero for this method and both the extreme and middle values are picked evenly. However, there is a major downside. The processing power required to complete this method is significantly more than required for the previous method. It takes several seconds more to run this program than the previous and this is only the most basic case in two dimensions. To employ this method in higher dimensions would be very time consuming. Therefore, although the uniformity is better, we were still not satisfied with the efficiency of this method and decided to try a new method. Please observe Figure 4.3 to see the result of this selection process.

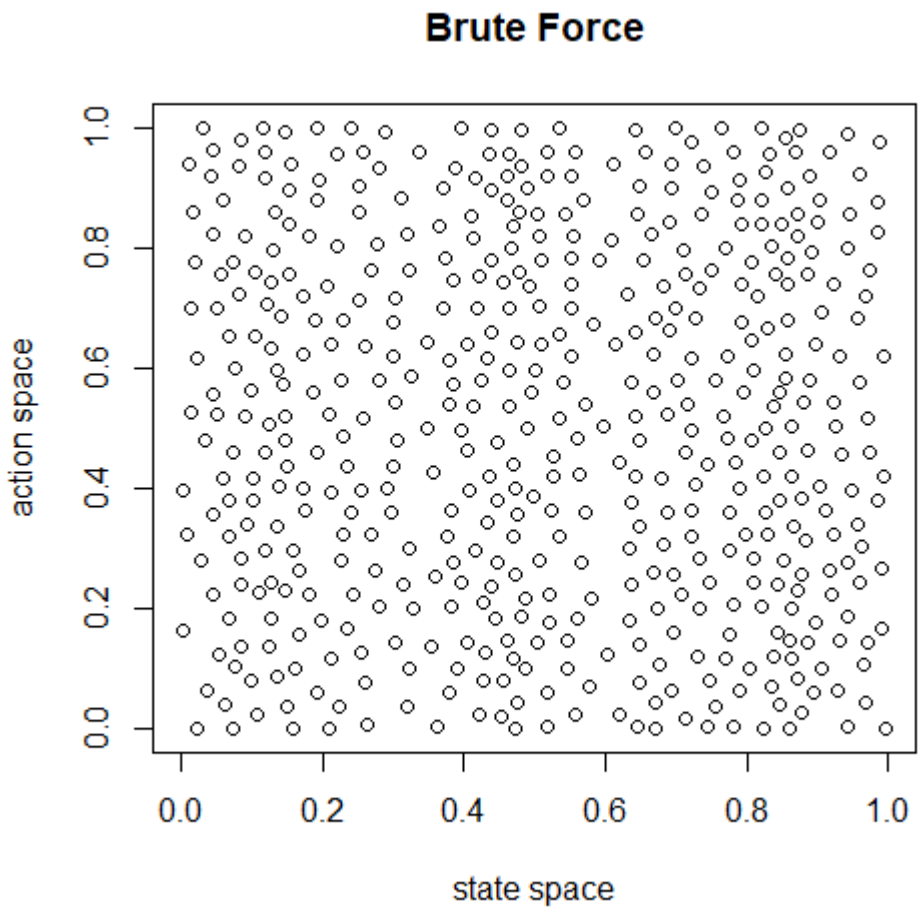

Figure 4.3: Increased uniformity but decreased efficiency due to using the “Brute Force” action selection method.

Algorithm 1 is an example of the algorithm for the above method.

Algorithm 1 Brute Force

Input: $num\_iterations$, $divisions$ (parameter for number of segments in grid)
Output: sa_hist (a state-action history matrix)
procedure MAIN
    for i in 1 to $num\_iterations$ **do**
        Generate random s_i
        $best\_objective \leftarrow 0$
        $a_{\text{best}} \leftarrow$ NULL
        for j in 0 to $divisions$ **do**
            $a_{test} \leftarrow \frac{j}{divisions}$
            Sweep sa_hist and find min_dist between (s_i, a_{best}) and (s_j, a_j) for $j < i$
            if $min\_dist < best\_objective$ **then**
                $a_{\text{best}} \leftarrow a_{test}$
                $best\_objective \leftarrow min\_dist$
            end if
        end for
        Store (s_i, a_{best})
    end for
end procedure

4.4 LDAS CANDIDATE 2

First, we tried to achieve dissimilarity to previous state-actions by treating the points like electrons in a field. Again the state dimension is considered fixed, but we allow the actions to move. Each previous point would repel the current point. Only the newly placed point would be allowed to move, the others were fixed, and if the new point moved past a boundary, it was stopped there and given the boundary value as its value. The new point

would calculate the forces from the previous action point using an inverse square method. This is the same as the real eclectic force formula. That means points closer had a much bigger force than those far away. Also, the new point would only move once after all of these forces were calculated. Wherever it landed would be its new permanent position. This method started out promising, but we soon learned that it was very difficult to find ideal values for the forces to be imparted on the new actions. This led to oversampling of the boundary/extreme values. The middle values would be under-represented. It is sometimes beneficial to over-sample extreme values, due to the fact that they may be ideal in a particular environment. However, this method over represented them to an undesirable degree. Even after many iterations the new actions were still forced to the boundaries and did not settle on the non-extreme values. Please observe Figure 4.4.

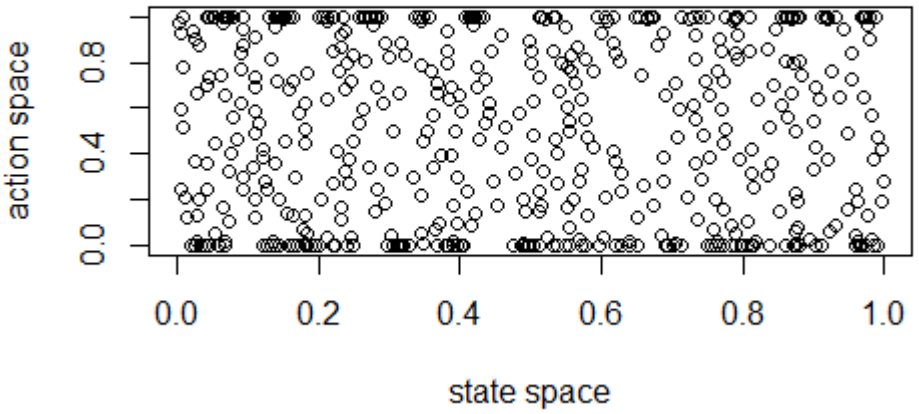

Figure 4.4: Over-represented extreme values due to using the “electrons in a field” action selection method.

Since this was happening, we decided to change the program and account for the fact that there were no actions pushing back from beyond the boundaries. Thus, we

programmed in “shadow points” just beyond the maximum and minimum action values. Therefore, when the new actions were moving toward the boundary, they would have a force countering them from reaching the boundary. In addition, we allowed our new point to move multiple times until it found a point that was balanced by all of the other forces. Since it might not find a balance, we added in a friction term to the field. This became stronger as the point moved. Every move would add more friction until the point would become stuck. This prevented the program from endlessly bouncing the point up and down and never finding a balance. Now we had many things to calculate. This was less computationally efficient, but perhaps it would give better results.

This seemed promising, but just like before it was very difficult finding an ideal value for the force of shadow points to balance the new actions. Now we ended up with the reverse problem. Instead of over representing the boundary values we were under representing them. Please observe Figure 4.5.

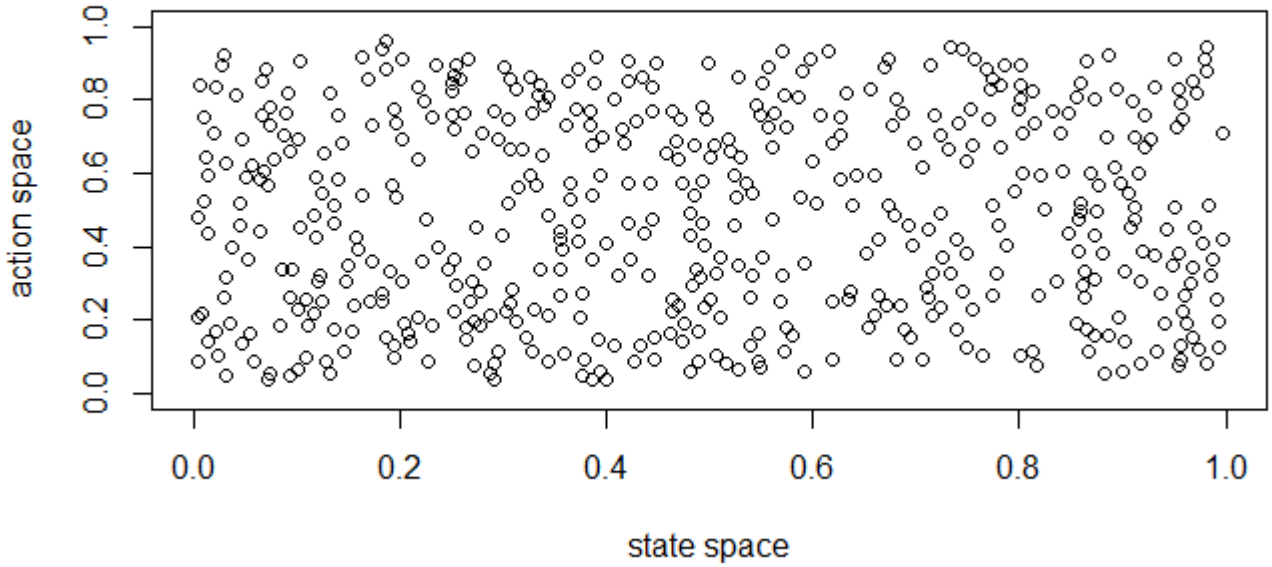

Figure 4.5: Under-represented extreme values due to using the updated "electrons in a field" action selection method

Algorithm 2 gives pseudo code for this method.

Algorithm 2 Electrons in a Field

Input: $num\_iterations$

Output: sa_hist (a state-action history matrix)

procedure MAIN

    Initialize first state and action s_0, a_0 in sa_hist

    for i from 1 to $num\_iterations$ **do**

        Create s_i and a_i

        Set $friction = 0$

        Set $movement = \infty$

        while $movement > \epsilon$ **do**

            Sweep sa_hist, find distances between (s_i, a_i) and each (s_j, a_j) for $j < i$

            Calculate force as proportional to inverse of square distance

            Apply $friction$

            Normalize the force

            $movement \leftarrow normalized\_force$

            Project onto the action dimension

            $a_i \leftarrow a_i + movement$

            Increase $friction$

            if a_i is on boundary **then**

                break

            end if

        end while

        Store s_i and a_i in sa_hist

    end for

end procedure

4.5 LDAS CANDIDATE 3

Our next method employs gradient descent. A gradient is the direction of steepest ascent or steepest increase. We can find or approximate the gradient and then move in the opposite direction. Then we will find the steepest descent, thus a gradient descent. We can set up equations of distance between the previous points. Then we can move towards the optimal placement of our new action so that it is as far away from all other points as possible. If we let x be the distance from the current action to the closest previous action, then we want to maximize x. This is equivalent to minimizing $\frac{1}{x}$. We can apply the gradient descent method to $\frac{1}{x}$. Since the gradient descent method employs a derivative of the objective function we should have $\frac{-1}{x^2}$ in our calculations. Upon reviewing the code on a later date we noticed that we have $\frac{-1}{x}$ where we should have $\frac{-1}{x^2}$. To the best of our knowledge this is an error. However, the direction and final position of the actions will be comparable, but the magnitude of the movement will be slightly different. The overall effect of the method will be similar. As the new action moves due to the method, the previous action that was closest before the move may not be after. If we move too far and go past the optimal point that is all taken into account in the gradient descent method and it will move our action back towards the optimal position. This method requires a step-size and a decay rate. If the step-size becomes smaller than some chosen number, say ϵ, then we stop the process and say we are close enough to the optimal position. To prevent an infinite loop from occurring where the new action bounces above and below the ideal position with movement greater than ϵ, we have a decay rate. The decay rate ensures that the magnitude of the movements are decreasing.

Although finding the right parameters for decay and step-size can be difficult, the rewards of using the method are worth it. Gradient descent allows us to find an approximately minimal distance between the point without requiring intense processing power. This produces Figure 4.6.

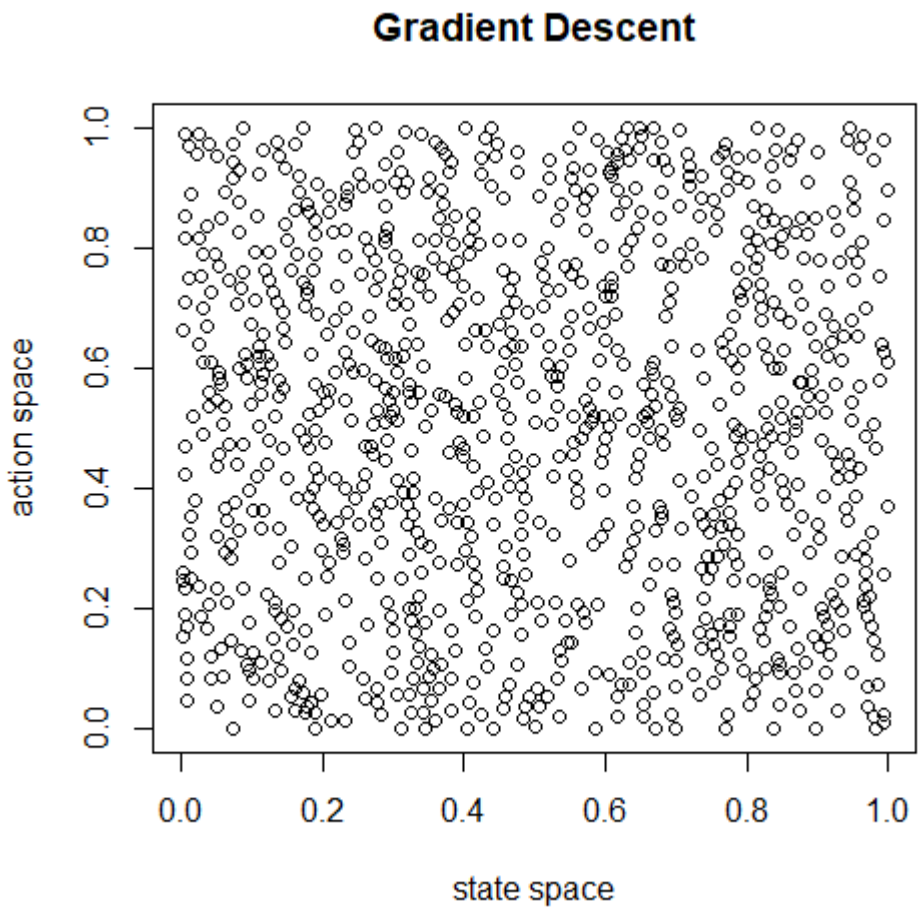

Figure 4.6: Result of using "Gradient Descent" action selection method with step-size = .1, decay rate = .7, and $\epsilon = .001$

Although, this is not quite as accurate as the brute force method in terms of discrepancy it is significantly more efficient in terms of time and processing power. This method was what we decided would be the one we would pursue as the most efficient and accurate.

Algorithm 3 is an example of the pseudo code for the gradient decent method.

Algorithm 3 Gradient Descent

Input: $\epsilon, num\_iterations, gradient\_stepsize, decay\_rate$
Output: sa_hist (a state-action history matrix)
procedure MAIN
    Initialize random (s_0, a_0)
    for i in 1 to $num\_iterations$ **do**
        $stop\_flag \leftarrow$ True
        Generate random (s_i, a_i)
        $current\_stepsize \leftarrow gradient\_stepsize$
        while $stop\_flag$ **do**
            Sweep sa_hist, find distances between (s_i, a_i) and each (s_j, a_j) for $j < i$
            Find $min\_distance$
            Calculate $gradient$ of $\frac{1}{min\_distance}$
            $a_i \leftarrow a_i - gradient * current\_stepsize$
            if a_i is on boundary **then** $stop\_flag$ = False
            end if
            if $gradient * current\_stepsize < epsilon$ **then** $stop\_flag$ = False
            end if
            $current\_stepsize \leftarrow current\_stepsize * decay\_rate$
        end while
        Store (s_i, a_i) in sa_hist
    end for
end procedure

The discrepancy for the methods are described in Table 4.1. Each method generated 500 points within one state dimension and one action dimension, and then the discrepancy was measured using 500 rectangles of random size. This process was repeated 250 times

and the mean was taken of all the discrepancy estimates for each method.

	Method	Discrepancy
1	Electrons in a Field	0.08323688
2	Brute Force	0.0490949
3	Gradient Decent	0.07164366
4	Uniform	0.05692339

Table 4.1: Discrepancy estimates for the four methods described

Although the Brute Force method has a lower discrepancy, the processing time lost to achieve this is not worth the decrease in discrepancy when compared to the gradient descent method. In addition, gradient descent favoring the boundaries, but not over-representing them to an extreme amount is an advantage in certain contexts. We will discuss this more with specific examples in chapter 5. For the remainder of the thesis when we refer to LDAS it will be assumed that it is operating off of the gradient descent method of selecting the actions.

4.6 Ornstein-Uhlenbeck selection

In addition to uniform selection, the other selection method that we will compare LDAS to is the Ornstein-Uhlenbeck selection method [11]. This method was originally conceived in a physics context as a stochastic differential equation and is equivalent, in discrete time, to a time series model. This method intentionally correlates actions in time. In some ways, this is an opposite selection method to LDAS. Correlated actions will be similar to ones previously selected. This can be a desirable method in the context of robotics. Since it correlates actions, it can reduce the wear on mechanical parts. Ornstein-Uhlenbeck can be thought of as an auto-regressive (AR(1)) process. Mathematically it could be represented as follows.

$$a_{t+1} = a_t(1 - \theta) + Z$$

Where θ represents the process parameter and is strictly greater than zero, and Z represents standard normal noise. Note that as θ is closer to zero then the auto correlation is stronger, but as θ gets closer to one the effect is weaker. If $\theta = 1$ then action selection is independent. This method has some drawbacks in that if it finds a poor action it will take it awhile to move away from that action in future picks. This is because it correlates the actions so that they are only slightly dissimilar to previous selections. As stated before, this is close to the opposite of the LDAS method and will make a great comparison method for testing. Now that the premise is understood we will go through the different approaches we tried to achieve an effective LDAS algorithm.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 TECHNICAL SET-UP

There are different aspects of the LDAS method that we want to compare to more standard methods. We start by devising experiments that will solely test the relative effectiveness of the methods during the exploration stage. We pick a few different, basic reinforcement learning problems to test on. Before going into the details of the problems let us discuss the set-up.

Experiments were run in Python version 3.6.12, making use of Spyder 5.05 as a development environment. For reinforcement learning environments, we used the Gym library [2], version .17.2. For exploration strategies, we used the Garage library [1], version 2021.3.0. Both Gym and Garage are intended to be run on a Linux operating system. Because both student and advisor use Windows machines, we employed Docker [6], a virtualization software, to run Ubuntu 18.04 in a container. This library contains a class for implementing Ornstein-Uhlenbeck action selection. To implement LDAS, we started with the built-in EpsilonGreedyPolicy and modified it to carry out Algorithm 3.

5.2 EXPLORATION EXPERIMENTS MOUNTAIN CAR

Now that we have the set-up, the first problem to test is the MountainCarContinuous-v0 problem. This is based off of Andrew Moore's dissertation. where he creates the details of this problem, but has since become a classic reinforcement learning benchmark environment [7]. This problem consists of a car starting at the bottom of a valley between two hills, with a goal being at the top of the hill on the right. The car can move to the right and to the left, alternating between forward and reverse. If the car only accelerates to the right it will not make it to the top of the hill. It needs to rock back and forth until it builds just

enough speed to overcome the hill. It can be a difficult problem since it starts in a state of failure and needs to complete a very specific sequence of events to reach the goal. To visualize this problem observe Figure 5.1

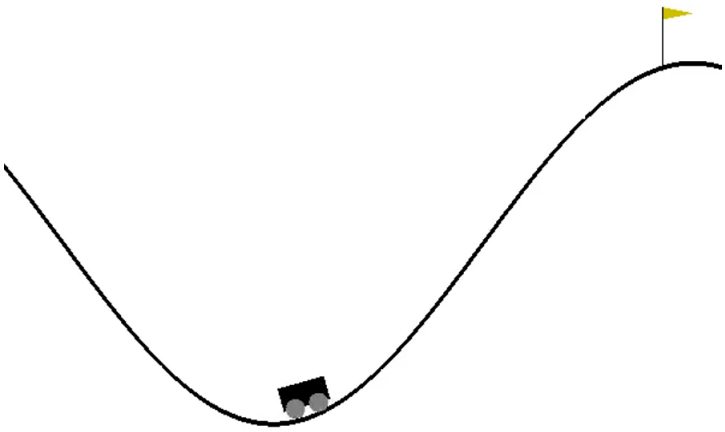

Figure 5.1: Visual representation of the mountain car problem

The details of this problem are as follows. The action space is one-dimensional between $[-1, 1]$. A negative number will accelerate the car to the left and a positive will accelerate it to the right. The state space is two-dimensional, and takes into account the car's position and its velocity. The reward for reaching the top of the right hill is 100. Reaching the top will end the episode, but it will then subtract the squared sum of actions from start to goal. This incentivizes the car to not make many actions carelessly, but that may mistakenly lead the agent to think that it is better to not move at all.

We did not train an agent to learn how to solve this problem. We instead focused on exploration to determine if action selection based on LDAS would be superior to a uniform action selection. We let the problem run for a maximum of 10,000 steps for 300 episodes. We would then measure the number of steps per episode for the LDAS method and compare this to the uniform and Ornstein-Uhlenbeck method. The lower the amount of steps

needed to reach the goal the better the exploration would be considered. The results can be visualized in Figure 5.2.

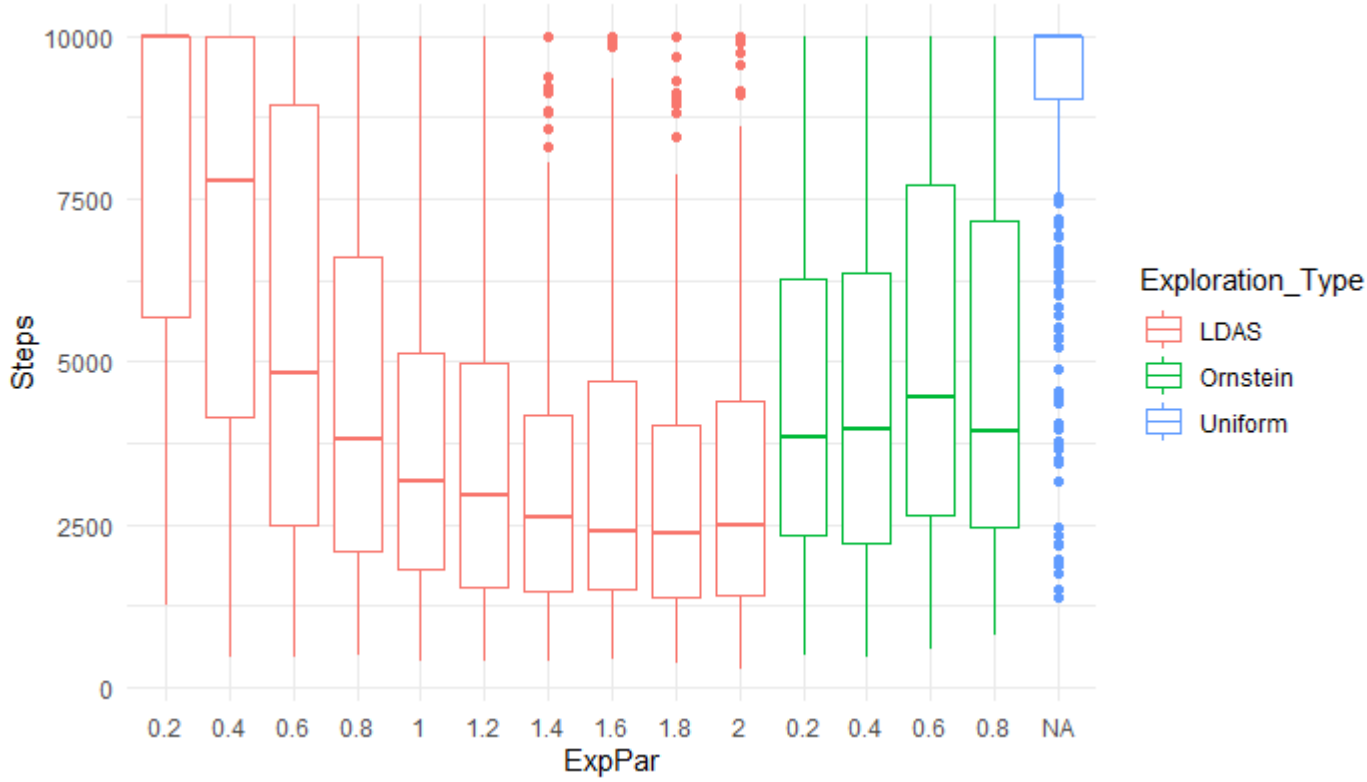

Figure 5.2: Comparison of the amount of steps taken to reach the goal of the mountain car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck. Lower values are better.

Starting from the far left of the graph we have the LDAS method ranging in step-size from 0.2 to 2. As we increase our step size the strength of the LDAS effect increases. It is clear that the number of steps to reach the goal goes down the more LDAS is employed for the selection process. Near the middle of the graph starts the Ornstein-Uhlenbeck method with a step size ranging from 0.2 to 0.8. The lower the step size is for this method the stronger the Ornstein-Uhlenbeck effect is, and the best performance this gives is at an auto correlation parameter of 0.2. Even at this parameter this method does not out-perform the

LDAS method. Lastly, the far right box plot in the image is uniform selection. It has no parameter which is why there is an "NA" below that particular box plot. It also has high values compared to the other methods, meaning it has a worse performance.

In addition to performance, we should also take into account the time needed to complete each episode. The detail of those results can be seen in Figure 5.3.

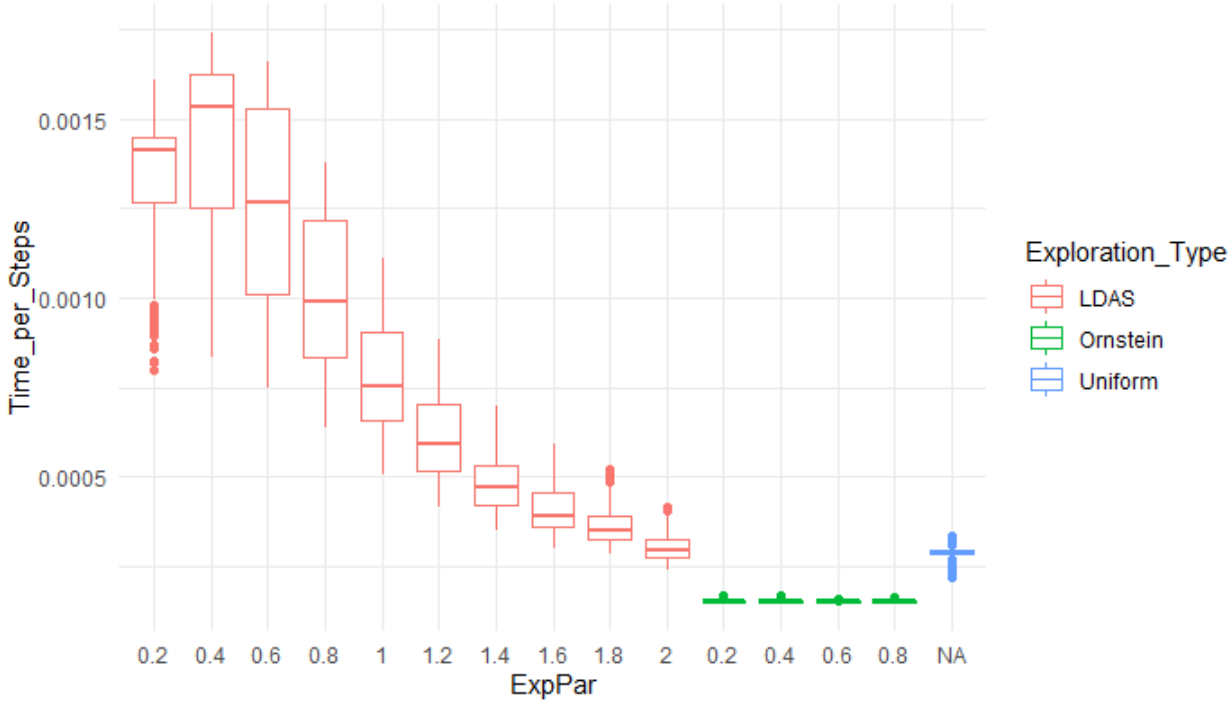

Figure 5.3: Comparison of the amount of time taken in seconds per step for the mountain car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck.

Here we can see that the uniform and Ornstein-Uhlenbeck methods are quicker, however the knowledge gained during that time is not as valuable as the LDAS method. The additional time required to run LDAS does detract from its overall performance, but depending on the context, selecting better actions even though it takes longer may be preferred. If data is expensive to collect, for example collecting data from a physical robot, a

few extra seconds of computation time to plan the next action might be more desirable.

The following two tables will be the numeric summary of these experiments. Table 5.1 shows the summary for the steps each method took. Table 5.2 shows the summary of the time each method took. Of special interest compare the first and last row in the mean column for this data. The uniform has a mean time of 2.59 seconds, but also has a mean of 8923.05 steps. The Ornstein-Uhlenbeck method has the best mean of 4861.38 at a step size of 0.8, and a corresponding time of 0.74. The LDAS method has the best mean at a step size of 1.8. The mean for this step size is 3086.35, and the corresponding time of 1.25 seconds. Although LDAS is a more complicated method and takes longer to run since it is running for fewer steps each episode the time for LDAS is superior to the uniform method, but inferior to the Ornstein-Uhlenbeck. In comparison to Ornstein-Uhlenbeck the extra time is spent finding more valuable actions to pick, since the LDAS method takes less steps to complete the task.

	Exploration_Type	Parameter	mean	median	sd	IQR
1	LDAS	0.2	7980.81	10000.00	2791.12	4316.75
2	LDAS	0.4	7003.04	7765.00	3152.93	5836.25
3	LDAS	0.6	5495.69	4834.00	3226.01	6429.50
4	LDAS	0.8	4556.06	3823.00	2914.55	4498.00
5	LDAS	1	3877.61	3155.00	2645.28	3318.50
6	LDAS	1.2	3664.10	2939.00	2648.15	3426.25
7	LDAS	1.4	3120.65	2603.50	2174.24	2724.75
8	LDAS	1.6	3308.55	2399.00	2522.59	3194.75
9	LDAS	1.8	3086.65	2372.50	2336.04	2662.75
10	LDAS	2	3214.87	2478.50	2412.42	2986.50
11	Ornstein	0.2	4592.41	3842.00	2834.90	3923.00
12	Ornstein	0.4	4529.50	3959.50	2740.51	4123.50
13	Ornstein	0.6	5094.83	4463.50	2983.82	5082.50
14	Ornstein	0.8	4861.38	3951.50	2891.70	4718.25
15	Uniform	NA	8923.05	10000.00	2101.04	968.25

Table 5.1: Numerical results for the amount of steps taken to reach the goal for the mountain car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck.

	Exploration_Type	Parameter	mean	sd	IQR
1	LDAS	0.2	11.09	4.57	7.23
2	LDAS	0.4	10.76	5.74	11.08
3	LDAS	0.6	7.71	5.52	11.24
4	LDAS	0.8	5.23	4.17	6.25
5	LDAS	1	3.45	2.99	3.50
6	LDAS	1.2	2.57	2.35	2.66
7	LDAS	1.4	1.68	1.49	1.61
8	LDAS	1.6	1.54	1.47	1.59
9	LDAS	1.8	1.25	1.18	1.12
10	LDAS	2	1.06	0.96	1.01
11	Ornstein	0.2	0.70	0.43	0.60
12	Ornstein	0.4	0.69	0.42	0.63
13	Ornstein	0.6	0.78	0.45	0.79
14	Ornstein	0.8	0.74	0.44	0.71
15	Uniform	NA	2.59	0.67	0.30

Table 5.2: Numerical results for the amount of time taken in seconds per episode for the mountain car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck.

5.3 Exploration Experiments Racing Car

The next experiment we conducted was with an environment called CarRacing-v0. This is a top-down racing environment where a track is randomly generated and the agent gains rewards as it moves forward and discovers more track. As a visual representation look at Figure 5.4.

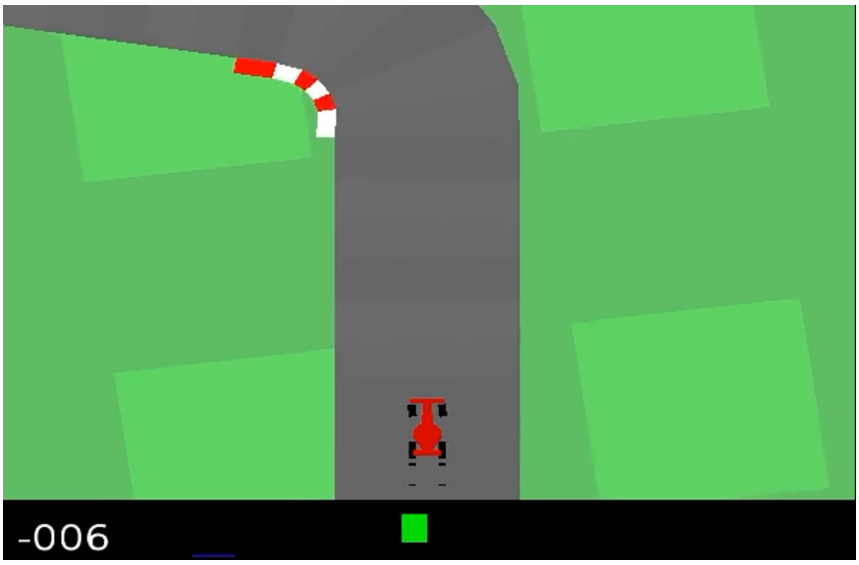

Figure 5.4: Visual representation of the race car problem.

The action space is three-dimensional and consists of the steering between $[-1, 1]$, gas between $[0, 1]$, and the brake between $[0, 1]$. A negative value for steering turns the agent left and a positive turns it to the right, while a value of zero keeps it going straight. For gas and brake the closer the value is to one the stronger the gas or brake is applied respectively. The state space is very large, because it is the pixel representation of the screen. That is the state space is 96 x 96 x 3. The 96 x 96 represents the amount of pixels on the screen, and we multiply that product by three to get the RGB representation of each pixel. The reward is 1000 divided by the total number of tiles in the track minus 0.1 for every frame. For example, if you finish in 500 frames, your reward is 1000 - 0.1*500 = 950 points.

Instead of measuring steps for this problem we measured the return of reward per episode for each exploration method. Again we will have a visual and numerical breakdown of the results. Now in this case the higher the box plot is the better because it indicates a higher reward.

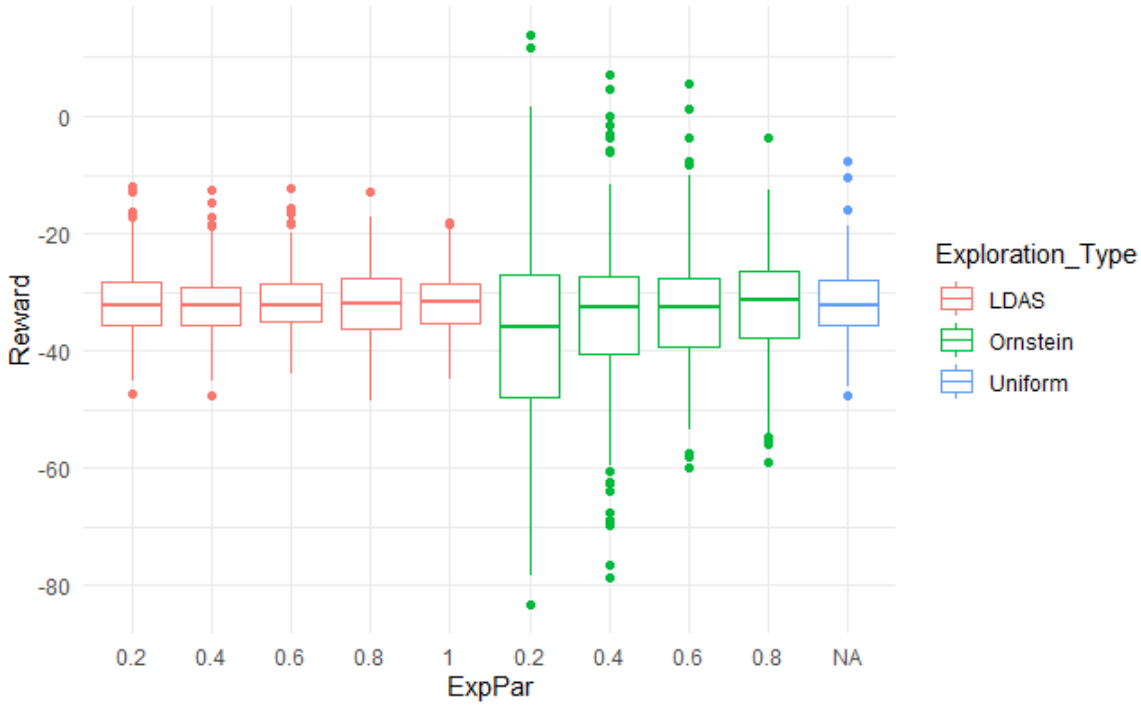

Figure 5.5: Comparison of the amount reward received per episode for the race car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck.

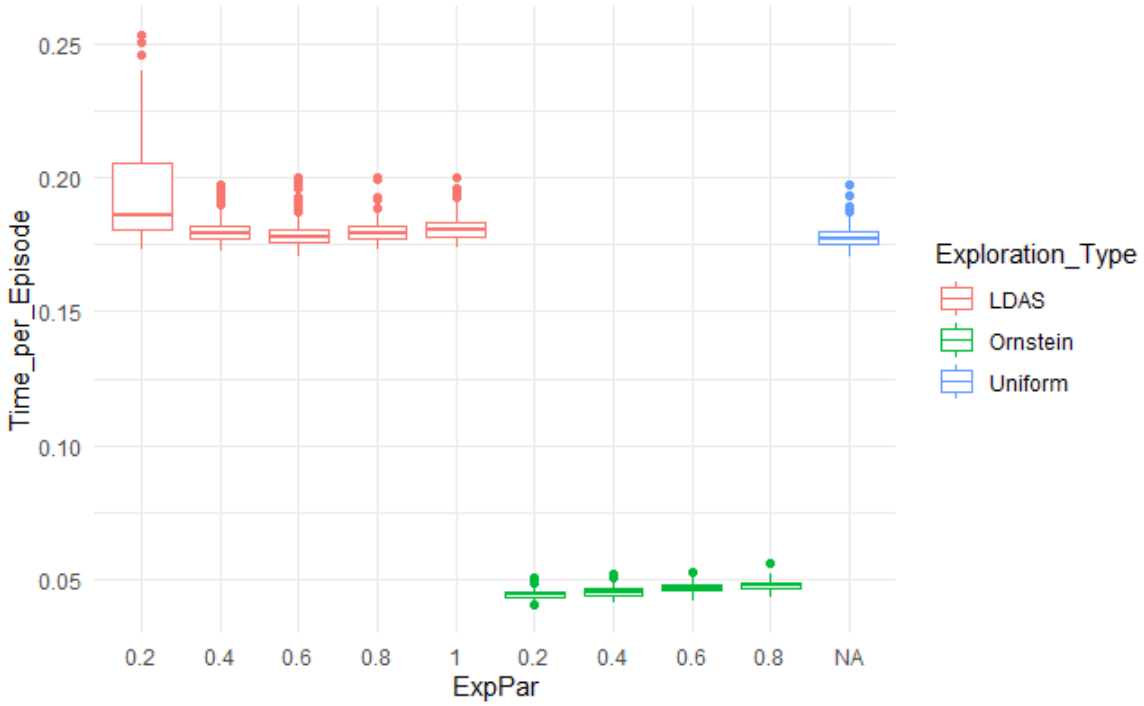

Figure 5.6: Comparison of the amount time taken in seconds per episode for the race car problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck.

Looking at Figure 5.5 and Figure 5.6 we see these results are far closer than the previous test. There is not a significant advantage to using one method over the other in this case. If we look at the numerical results they confirm this.

Table 5.3 covers the numeric results of the reward seen per episode, and Table 5.4 details the time taken per episode.

	Exploration_Type	Parameter	mean	median	sd	IQR
1	LDAS	0.2	-31.76	-32.14	5.78	7.24
2	LDAS	0.4	-31.96	-32.20	5.39	6.54
3	LDAS	0.6	-31.90	-32.32	5.33	6.58
4	LDAS	0.8	-31.83	-31.86	5.80	8.44
5	LDAS	1	-31.78	-31.74	5.25	6.83
6	Ornstein	0.2	-38.20	-36.06	17.08	21.03
7	Ornstein	0.4	-34.24	-32.57	13.17	13.16
8	Ornstein	0.6	-32.99	-32.69	10.32	11.71
9	Ornstein	0.8	-32.50	-31.48	9.17	11.08
10	Uniform	NA	-31.62	-32.20	5.79	7.45

Table 5.3: Numerical results for the amount of reward received per episode for the race car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck.

	Exploration_Type	Parameter	mean	sd	IQR
1	LDAS	0.2	0.19	0.02	0.02
2	LDAS	0.4	0.18	0.00	0.00
3	LDAS	0.6	0.18	0.00	0.00
4	LDAS	0.8	0.18	0.00	0.00
5	LDAS	1	0.18	0.00	0.01
6	Ornstein	0.2	0.04	0.00	0.00
7	Ornstein	0.4	0.05	0.00	0.00
8	Ornstein	0.6	0.05	0.00	0.00
9	Ornstein	0.8	0.05	0.00	0.00
10	Uniform	NA	0.18	0.00	0.00

Table 5.4: Numerical results for the amount of time taken per episode for the race car problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck.

These results are very close, and evaluating the problem reveals an explanation. The uniform method picks randomly with no favor to any action. This moves the car around the track slowly, but uniformly. The Ornstein-Uhlenbeckk method uses correlated data to pick actions that are similar to previous actions. When it has the car move forward it receives a large reward because it continues to move forward and see more of the track. However, if it selects an action to hit the brake or turn the car, it will continue to do actions similar to this for awhile and receive a very low reward. This is why the variance is much higher for this method. Depending on the first action selected it either does very well with many similar actions, or very poorly. The LDAS method takes actions that are as dissimilar as possible from previous actions to learn its environment. So if LDAS accelerates the car forward and gains a large reward from that, it is likely its next action will be to slam on the brakes and gain no reward next. This is because it is only exploring, it is not trying to learn to maximize its reward. Overall for exploration, both methods are very comparable in efficiency.

5.4 Exploration Experiments Lunar Lander

The next problem we decided to test against was the LunarLanderContinuous-v2 problem. This is a more complicated problem. It starts with a lander machine that needs to land on the moon between two goal posts. Observe Figure 5.7.

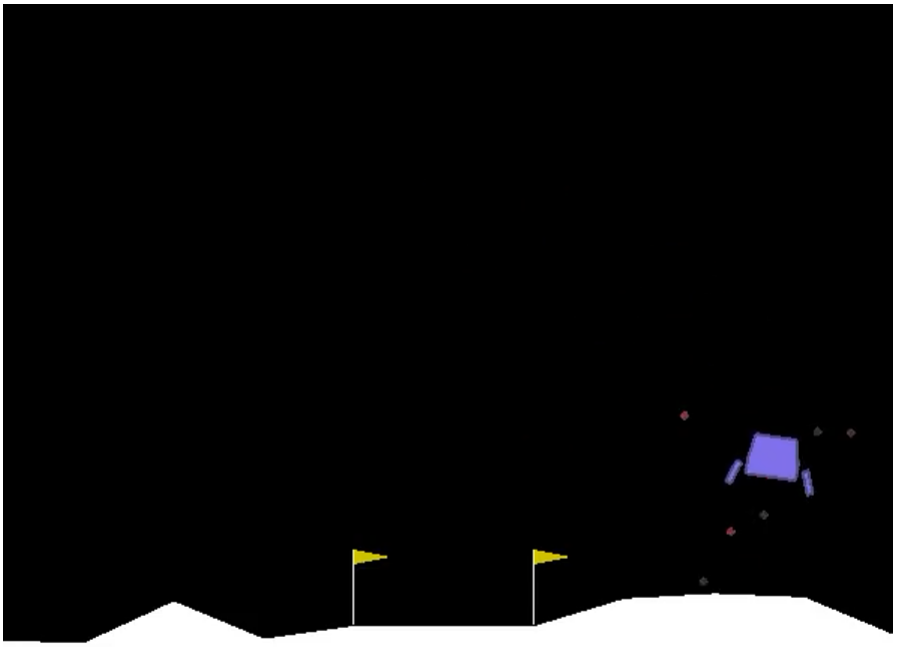

Figure 5.7: Visual representation of the lunar lander problem.

It has an engine that pushes the machine straight up, and an engine that can move the machine left and right. There is not enough energy in the engine to actually lift the machine, but only slow its decent to a pace that is safe for landing. The reward structure of this problem is that it gets a large positive reward of 100 if it lands gracefully anywhere without crashing. It will receive an additional 100-140 reward if it lands between the goal posts. However, if it crashes it can lose up to 200 reward depending on how rough the crash is.

The state space for this problem is eight-dimensional. It consists of horizontal and vertical position as well as horizontal and vertical velocity. It takes into account the angle that the lander has from an upright position and the angular velocity. Lastly, it sees if the right and left leg of the lander are touching the ground.

The action space is two-dimensional. It consists of the side engine moving the lander

right and left, and the main engine lifting the lander up.

The reward structure is as follows. It will lose .03 per frame for using the side engine and loses .3 per frame for using the main engine. This incentivizes the agent to conserve fuel. If the lander crashes it loses 100 reward, but if it lands without a crash it gains 100. If it lands between the goal posts it gains 100 to 140 reward, and loses reward for moving away from the goal posts. Finally, for each leg touching the ground when the lander lands it gains 10 reward.

Again we focused on the exploration portion of this problem, pitting LDAS against the Uniform and Ornstein-Uhlenbeck exploration methods to see which would perform best. Here we are seeing how many episodes it took to obtain a good reward. In our case, a reward of positive five. Therefore, the lower the box plot is the better it is, and we can see the result in Figure 5.8.

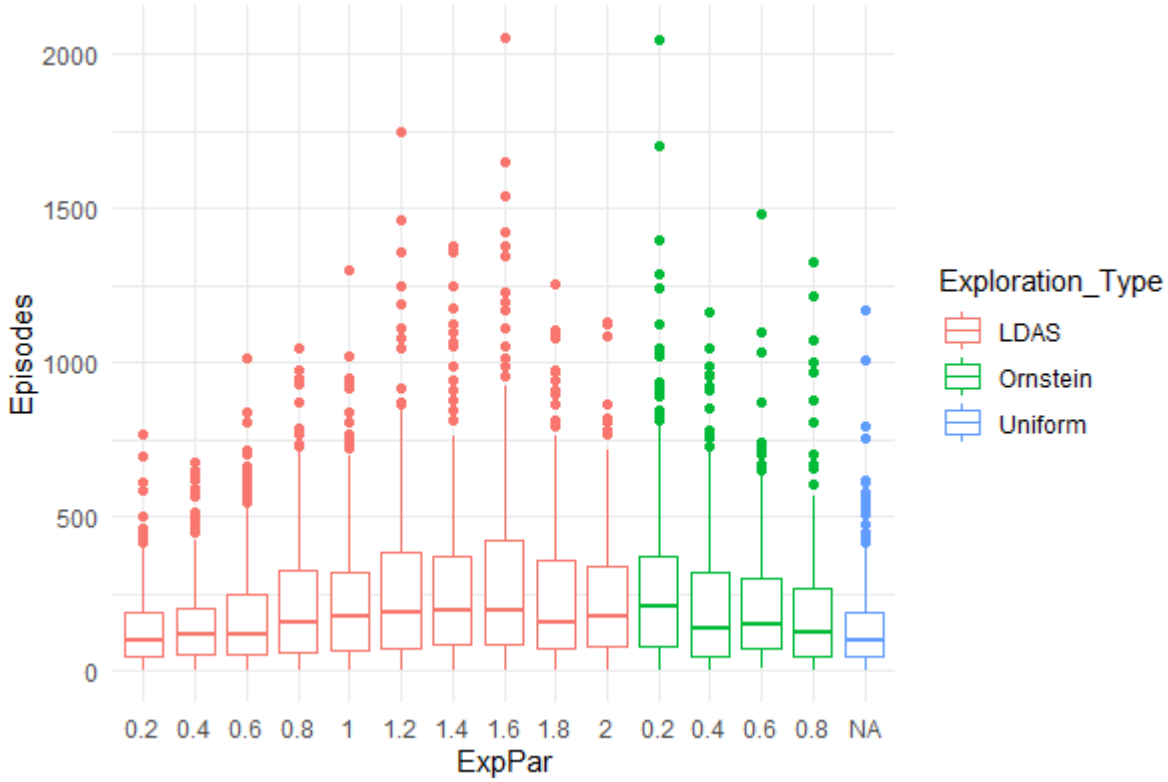

Figure 5.8: Comparison of the amount episodes taken to receive a reward of 5 for the lunar lander problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck.

There is not a large difference between the results, but it does not look like LDAS is superior. We can see that with a step-size of 0.2, LDAS and uniform's performance is almost the same, and both are comparable to Ornstein-Uhlenbeck with a step size of 0.8. Why would LDAS be worse here when it seems more clearly superior in the mountain car problem? This is likely due to the fact that LDAS samples from the extreme values at a higher rate then the uniform method. In a problem like mountain car this is an advantage as selecting the extreme values tend to get the car to the top of the hill. However, in the lunar lander problem where precise actions are needed to control the throttle this is a disadvantage. Let us also check the time required for these methods with Figure 5.9.

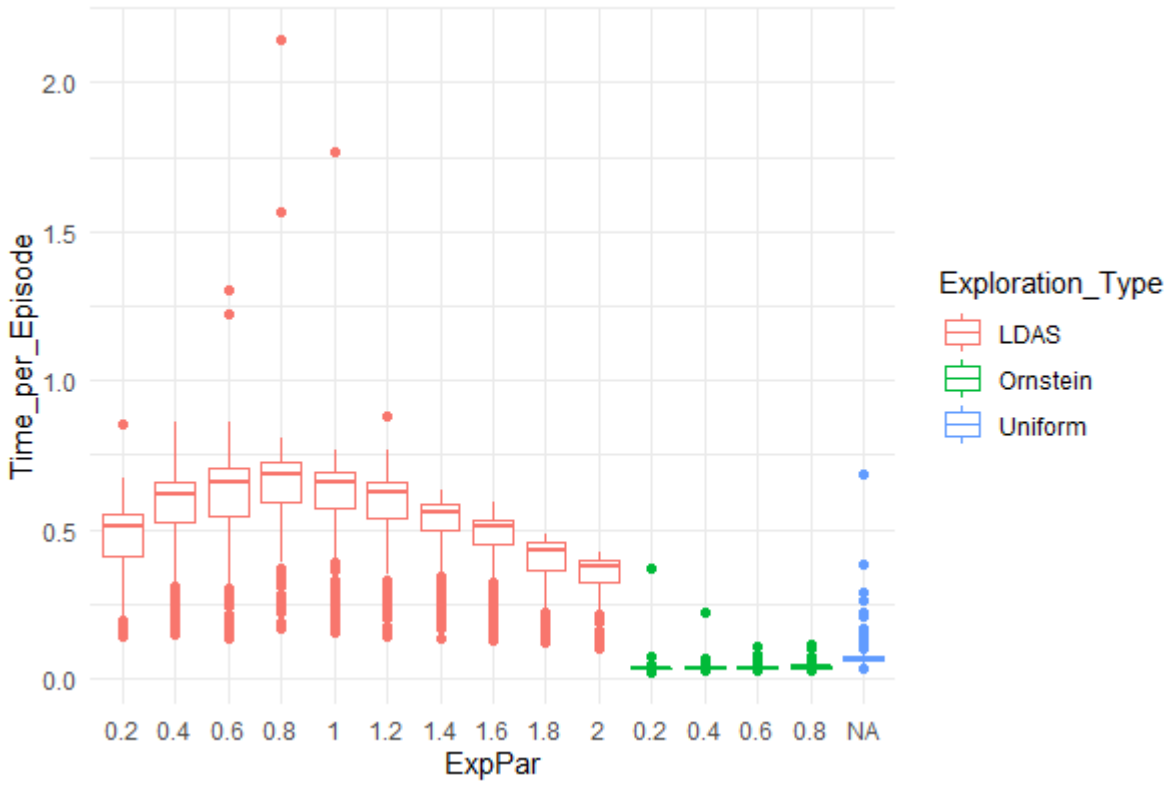

Figure 5.9: Comparison of the amount of time taken per episode for the lunar lander problem between the three action selection methods Uniform, LDAS, Ornstein-Uhlenbeck.

Now we see an issue with LDAS. Although the results for reward are similar between the methods the time taken is significantly different. Perhaps, in the learning phase the LDAS method would provide more meaningful learning, but strictly looking at exploration it seems to be inferior to the other methods. The numerical results for the amount of episodes taken to see a reward of 5 are summarized in Table 5.5, and the numeric results for the time taken per episode are summarized in Table 5.6.

	Exploration_Type	Parameter	mean	median	sd	IQR
1	LDAS	0.2	142.96	102.50	137.66	146.25
2	LDAS	0.4	157.97	116.50	148.66	148.75
3	LDAS	0.6	183.68	118.00	182.31	196.25
4	LDAS	0.8	226.90	160.00	218.72	261.50
5	LDAS	1	235.99	174.50	224.81	252.50
6	LDAS	1.2	283.30	190.50	286.68	314.50
7	LDAS	1.4	270.52	199.50	259.16	284.50
8	LDAS	1.6	310.38	195.50	332.91	339.00
9	LDAS	1.8	248.58	160.50	246.71	288.25
10	LDAS	2	245.13	178.50	212.64	259.00
11	Ornstein	0.2	297.94	211.50	309.76	293.00
12	Ornstein	0.4	226.82	135.50	233.75	270.00
13	Ornstein	0.6	216.99	153.00	208.25	229.00
14	Ornstein	0.8	198.58	123.50	219.42	219.50
15	Uniform	NA	156.09	101.50	170.04	146.00

Table 5.5: Numerical results for the amount of episodes taken to receive a reward of 5 for the lunar lander problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck.

	Exploration_Type	Parameter	mean	sd	IQR
1	LDAS	0.2	0.47	0.13	0.14
2	LDAS	0.4	0.56	0.15	0.14
3	LDAS	0.6	0.60	0.17	0.16
4	LDAS	0.8	0.64	0.19	0.14
5	LDAS	1	0.61	0.15	0.12
6	LDAS	1.2	0.57	0.14	0.12
7	LDAS	1.4	0.52	0.10	0.09
8	LDAS	1.6	0.46	0.11	0.08
9	LDAS	1.8	0.40	0.09	0.09
10	LDAS	2	0.35	0.07	0.07
11	Ornstein	0.2	0.04	0.02	0.00
12	Ornstein	0.4	0.04	0.01	0.00
13	Ornstein	0.6	0.04	0.01	0.00
14	Ornstein	0.8	0.04	0.01	0.00
15	Uniform	NA	0.08	0.05	0.02

Table 5.6: Numerical results for the amount of time taken per episode for the lunar lander problem between the three action selection methods, Uniform, LDAS, Ornstein-Uhlenbeck.

The mean time for the uniform method was $.08$, and the best mean time for the Ornstein-Uhlenbeck method was $.04$. However, the fastest time for the LDAS was $.35$. Although we wanted LDAS to be superior in all aspects, it was not the case for the exploration tests we did. However, that does not mean that the results will necessarily hold for the learning experiments.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In conclusion to the results of the experiments run for this thesis, we have determined that the best action selection method depends greatly on the particular environment the agent is trying to learn in. On one hand, using a low discrepancy action selection process produces superior results in the exploration stage when extreme values are more desirable and optimal actions will alternate, like the mountain car problem. On the other hand, if precise control is needed and extreme actions are undesirable, such as in the lunar lander problem, LDAS may be inferior. Finally, in an environment in which maintaining momentum in the action dimension is helpful, such as the racing car problem, Ornstein-Uhlenbeck is likely to perform best.

Although this thesis did not conduct any learning experiments, it is possible that LDAS would produce improved learning for the agent. The knowledge of the actions gained during the exploration stage under LDAS is extensive compared to the other methods. It will see a wider range of its actions more quickly and this could lead to a more optimized learning for the agent. Future work will be to determine how LDAS performs during the learning phase for an agent compared to other methods.

REFERENCES

[1] *Garage*, https://garage.readthedocs.io/en/latest/user/get_started.html.

[2] *Gym*, https://gym.openai.com/.

[3] *Ibm deep blue*, https://www.ibm.com/ibm/history/ibm100/us/en/icons/deepblue/.

[4] Richard Ernest Bellman, *The theory of dynamic programming*, RAND Corporation, Santa Monica, CA, 1954.

[5] Lauwerens Kuipers and Harald Niederreiter, *Uniform distribution of sequences*, Courier Corporation, 2012.

[6] Dirk Merkel, *Docker: lightweight linux containers for consistent development and deployment*, Linux journal **2014** (2014), no. 239, 2.

[7] Andrew William Moore, *Efficient memory-based learning for robot control*, Tech. report, 1990.

[8] Arthur L. Samuel, *Some studies in machine learning using the game of checkers*, IBM J. Res. Dev. **3** (1959), 210–229.

[9] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[10] Gerald Tesauro, *Temporal difference learning and td-gammon*, Commun. ACM **38** (1995), no. 3, 58–68.

[11] G. E. Uhlenbeck and L. S. Ornstein, *On the theory of the brownian motion*, Phys. Rev. **36** (1930), 823–841.

1480