StructuRL Bridge Maintenance: Leveraging Reinforcement Learning

Punna Chowdhury, Jane Slagle, and Diana Krmzian

Tufts CS 138 - Reinforcement Learning

Instructor: Yash Shukla

December 20, 2024

Abstract

Without well-maintained bridges, the integrity of infrastructure in the United States would be at significant risk. A major challenge in ensuring the safety of bridges lies in poorly maintained structures, mismanaged maintenance tasks, and budgetary constraints. In this project, we address these issues by training agents using various reinforcement learning algorithms to optimize maintenance tasks. Our methods demonstrate strong adaptability to realistic scenarios while effectively incorporating budget constraints, contributing to a plausible solution for managing the maintenance of numerous bridges - ensuring there is always a way to cross whatever bridge when we come to it!

1 Introduction

Bridge maintenance planning is a challenging endeavor, with obstacles that extend across various aspects. The complexity of maintaining bridges is vast, stemming from the endless possibilities of structural components and the various environmental factors that vary based on location. Reinforcement learning has become a commonly used approach for exploring the challenges in infrastructure management to address maintenance planning issues. However, existing reinforcement learning solutions often face difficulties in scaling effectively due to the larger state and action spaces inherent in bridge environments.

To combat these issues, in this project, we investigate a simplified bridge environment at a higher level of abstraction and evaluate the performance of various reinforcement learning algorithms when interacting with this environment, including Semi-Markov Decision Processes (SMDP), Deep SARSA, and Deep Q-Learning. Our approach aims to lessen the limitations of traditional reinforcement learning methods by decomposing extensive state and action spaces into smaller, more manageable subspaces. This enables a more efficient learning process while also enhancing the condition of bridges.

We evaluate our models on a single bridge over a 100-year period, selecting this time frame for testing to reflect real-world scenarios, motivated by how the majority of bridges in the United States were initially designed with a service life of around 50 years [1]. To ensure a realistic representation, we model a bridge that is initially in a less-than-good condition, allowing us to assess how effectively each algorithm in our approach either improves or contributes to the further deterioration of the bridge's condition over time. We rank the algorithms based on two key factors: their ability to improve the bridge's condition and how well they are able to manage the available budget while doing so. While some outcomes were unexpected, our overall results are reasonable and contribute meaningfully to the improvement of *structuRL* integrity of bridges.

2 Bridge Infrastructure Background and Related Work

2.1 Challenges Facing Bridge Infrastructure

America's aging infrastructure has significantly deteriorated its bridges, pushing many of them to the brink of failure. As of 2024, the American Road & Transportation Builders Association (ARTBA) reports that one in three U.S. bridges is in urgent need of repair or replacement [2]. More specifically, nearly 221,800 out of the nation's 623,147 bridges demand repair, with 76,175 bridges deemed in need of full replacement [2]. To emphasize the gravity of the situation, 42,067 of those bridges calling for repair are rated in poor condition and classified as "structurally deficient" [2]. ARTBA warns that this number continues to rise across many states year after year. A structurally deficient bridge is one that requires a substantial amount of maintenance to remain functional and safe for those who rely on crossing it [3]. Every day, drivers in America make approximately 168.5 million trips across these structurally deficient bridges, which span over 6,1000 miles [2], highlighting the widespread nature of

this issue.

Unavoidable elements such as natural aging, weather conditions, and frequent usage have contributed to the gradual deterioration of bridges across the U.S., with the average bridge now being 44 years old [3]. This is cause for concern as the majority of the country's bridges were originally designed with an expected service life of 50 years [3].

In addition, more controllable factors, including ineffective maintenance practices, inadequate intervention strategies, and inaccuracies in inspection and monitoring, have contributed to delayed repairs and magnified the decline of these structures. These issues, worsened by limited funding and resource constraints, have accelerated the rate of deterioration, leaving many bridges increasingly vulnerable to further damage. As Kevin Heaslip, a civil and environmental engineering professor at Virginia Tech, aptly stated, "We deferred maintenance for a long time, and now we have a significant backlog of repairs, with insufficient funding to address it" [3], highlighting the lack of resources available to address this problem.

Moreover, William Ibbs, a civil engineering professor at the University of California Berkeley, who also studies this issue, affirms that "the state of bridges in the U.S. is not good, and we're losing the battle" [3]. This statement reinforces that we can no longer approach this issue with the mindset of "we'll cross that bridge when we come to it", because by the time we need to cross, the bridge may no longer be safe— or even standing.

2.2 Conceptual Approach to Bridge Maintenance

2.2.1 Building On Existing Research

In our pursuit of a research topic, we were fortunate to discover *Hierarchical Reinforcement Learning for Transportation Infrastructure Maintenance Planning* [4], a research paper that addresses many of the challenges associated with bridge maintenance. The paper uses reinforcement learning to create adaptive decision-making processes tailored to the specific conditions of a bridge. It focuses on resource optimization, long-term policy planning, and cost savings— all elements that are crucial for the sustainable upkeep of critical infrastructure, such as bridges.

Upon stumbling across this paper, we were immediately intrigued, each of us having a personal interest in the upkeep of public infrastructure. We were drawn to this paper in particular due to its specific focus on bridge maintenance, as bridges play a vital, yet often overlooked, role in our daily lives. Heavily inspired by the potential of all we could accomplish with the approach introduced in this paper, we too aim to explore how reinforcement learning can be applied to optimize bridge maintenance planning, using our results to ensure that bridges are maintained safely and efficiently.

2.2.2 Overview of Our Proposed Approach

The authors of the paper described above focus on a specific bridge within a network of bridges in Quebec, Canada. Similarly, we will focus on a single bridge in our implementation to maintain simplicity and address time and resource constraints for this project. Again, being motivated by the referenced paper, a central component of our project is implementing a decision-making framework with an action space reflecting maintenance tasks and a state space reflecting the condition of the bridge through reinforcement learning. Leveraging the bridge environment described below in **Section 3**, we will use the decision-making framework alongside Semi-Markov Decision Processes (SMDP), Deep SARSA, and Deep Q-Learning to explore the problem.

SMDP offers temporal flexibility, enabling us to accurately model real-world scenarios with our project, such as maintenance scheduling, where actions like bridge repairs extend over time, making it well

suited to the task at hand. We will use Deep SARSA methods to refine our strategies based on actual performance data, by updating the action-value function based on the action taken by the current policy. This will help us make more informed decision-making with our model. Deep Q-learning, which combines deep and reinforcement Q-learning, will be used to address the complexities of the large state-action spaces in bridge maintenance. It offers a scalable alternative to traditional tabular Q-learning, which will allow our model to generalize from experiences and efficiently solve problems in our complex environment.

While our work draws considerable inspiration from the original paper, our project will introduce new research questions, implementation methods, and refinements tailored to our objectives, as outlined in more detail in **Sections 3.2, 3.3, and 3.4**, where we delve further into the motivation for selecting these techniques and their roles in our project.

2.2.3 Planned Outcomes and Long-Term Vision

The primary goal of our proposed approach is to optimize bridge maintenance by reducing costs, prioritizing repairs, and efficiently allocating resources.

Our main objectives include:

1. **Lowering maintenance costs through strategic repair planning**
2. **Addressing urgent issues promptly to ensure safety**

The future potential of our work spans providing a cost-effective hierarchical framework for managing repairs, enabling proactive responses to prevent larger issues, and simplifying the overall management of extensive networks by supporting long-term planning.

3 Technical Design

3.1 Environment

3.1.1 Inspiration for Environment

As discussed in **Section 2.2.1**, our approach to this problem is heavily inspired by the research paper *Hierarchical Reinforcement Learning for Transportation Infrastructure Maintenance Planning* [4]. The authors of this paper provided a public GitHub repository, *InfrastructuresPlanner*, which hosts all the related code for their paper, including a completely implemented environment called *InfraPlanner* [5].

Initially, we were hoping to use the already implemented and complete *InfraPlanner* environment [5] directly in our project to lighten the workload, given the limited time availability to complete this project during the semester. However, to much of our dismay, we encountered significant challenges in integrating it with our code. The *InfraPlanner* environment [5] is highly complex, spanning over 2,000 lines of code across multiple classes, and is not extensively documented. Despite very substantial efforts, we were ultimately unable to get the environment working with our implementation.

As a result, to continue making progress with our project, we opted to instead create a simplified custom version of the *InfraPlanner* environment. Our custom implementation of *InfraPlanner* is inspired by the original, much more simplified, but tailored to meet our specific needs. We prioritized the following key aspects from the original *InfraPlanner* environment when building our custom implementation:

- **action space:** the specific maintenance-related tasks such as **do nothing**, **maintenance**, and **replace**. In the original *InfraPlanner* environment, the authors broke the maintenance tasks down into smaller and specific maintenance tasks, having specific actions for various types of

maintenance tasks. To simplify this for the limited time we had to implement the project, we chose to combine all of those smaller maintenance tasks under one umbrella called "maintenance". We touch on how we could expand upon this to further enhance our project in **Section 6** below.

- **action costs:** the idea that each possible action in the action space has an associated cost that is deducted from the total budget available each time that action is taken per time step.
- **budget constraints:** the idea that the agent is limited by a running budget while taking actions over the total time period.

This simplified custom version of *InfraPlanner*, described in more detail below, allowed us to proceed with our project without losing the defining concepts of the original environment.

3.1.2 Custom Implementation of Environment

Our *InfraPlanner* is a simulation environment for maintenance of bridge infrastructure on a single bridge over a 100-year period. Again, key components that are essential to our environment are:

- **action costs:** assigning a cost to each action to reflect real-world scenarios
- **budget constraints:** ensuring that actions taken from the action space are balanced and chosen based on the amount of budget available
- **reward determination:** balancing two objectives - improving the bridge's condition over time while maintaining efficient and effective budget management

Our environment was designed to simulate various bridge maintenance tasks that closely mirror real-world scenarios, incorporating pragmatic considerations such as budget constraints and progress tracking. The reward function was developed with prioritization in mind, assigning rewards based on the current condition of the bridge, its improvement over time, and the agent's ability to manage the budget efficiently. This approach aims to prioritize maintenance strategies that will ultimately lead to long-term improvement. Additionally, our environment was built to emphasize cost-efficiency by penalizing actions that accelerate bridge deterioration or exceed the budget, while incentivizing those that enhance the condition of the bridge and adhere within budgetary limits.

All of which is more explicitly defined by the following:

State Space consists of a single state representing the condition of the bridge. This condition of the bridge is represented as an integer value between 0 and 100, where 0 indicates a completely deteriorated and unsafe bridge while 100 represents a bridge in perfectly pristine condition. The condition of the bridge is always initialized as 40 so that we may observe how our model improves the condition of the bridge over time.

Action Space consists of three possible actions related to bridge maintenance: **do nothing**, **maintenance**, and **replace**. Each action is associated with a fixed cost that is deducted from the available total budget for every time step the agent selects it.

The **do nothing** action represents neglecting bridge maintenance, and it collects no cost, meaning it does not affect the budget.

The **maintenance** action, with a cost of 2, is selected to gradually improve the condition of the bridge over time. Specifically, each time the action is chosen by the agent, the bridge condition improves by 1%, reflecting the gradual nature of real-world bridge maintenance.

The **replace** action, which involves completely replacing the bridge with a new one in pristine condition, has a higher cost of 5 per time step, as it represents a more significant investment compared to regular maintenance.

Step Function incorporates variable time step lengths when running the SMDP algorithm. For non-SMDP algorithms, each action takes a single time step, as per usual. The budget is directly linked to the time steps. If the agent exhausts the available budget before the end of the total 100-year time period, a significant penalty of -10 is applied to the reward. When there is sufficient budget to take an action, the cost of that action is deducted from the total budget for each time step. Depending on the chosen action, the bridge's condition will either improve, deteriorate, or be completely replaced. When running the SMDP algorithm, the condition of the bridge and reward are scaled for each time step according to the variable time duration of each action.

Reward Function is based on three key factors: the current condition of the bridge, whether the condition improves over time, and the effectiveness of budget management. We established thresholds to classify bridge conditions, defining a state of 80 or above as a bridge in "good" condition and 20 or below as a bridge in "bad" condition. To refine the reward function, we experimented with various weight configurations, analyzing how different values affected the outcomes of our model. This process helped us identify weights that were either overly strict or too lenient, ultimately leading us to define the reward function as follows:

	Reward Penalties	**Reward Incentives**
Current Bridge Condition	≤ 20: *-10*	≥ 80: *+10*
Improvement from Previous Condition	*0*	*+3*
Budget Management	exceeds budget: *-5*	else: *+2*

Table 1: Reward function weights for our *InfraPlanner* environment.

3.2 Semi-Markov Decision Processes (SMDP)

Semi-Markov Decision Processes model sequential decision-making by integrating states and actions with varying durations. Unlike traditional Markov Decision Processes, where each action occurs in a fixed time interval, with SMDP, each action can take a different amount of time to complete. Actions with flexible durations make SMDP an approach that accommodates tasks requiring variable amounts of time, ensuring a more realistic representation of decision-making processes where timing influences outcomes.

We implement a **Q-learning based SMDP** algorithm in our model. Using Q-learning, the agent observes the outcomes, interacts with the *InfraPlanner* environment, and receives feedback in the form of rewards. Again, a key characteristic of SMDP is its ability to handle actions with variable durations. The duration of each action is randomly selected during training from a range of one to five years. We chose to determine this value randomly to better reflect real-world uncertainty, as maintenance tasks rarely have predictable durations. By introducing this randomness in the way we set the duration for each action, we are able to capture a more realistic representation of the variability encountered in real-world maintenance tasks. At each time step, the agent selects actions by following an epsilon-greedy policy. The Q-function values are updated by adjusting the standard Q-learning formula to account for the variable action durations, as shown below [6]:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha[R_t + \gamma^{\triangle A_t}(Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t))]$$

where $\triangle A_t$ represents the duration length of action A at time step t.

SMDP is well-suited to the task at hand as it models sequential decision-making, handles actions that span varying time intervals, and provides temporal flexibility. These characteristics make SMDP ideal for simulating bridge maintenance, where tasks vary in duration and need to be performed over time, leading to a more realistic and accurate representation of the maintenance environment.

3.3 Deep SARSA

Deep SARSA is a combination of SARSA (State-Action-Reward-State-Action), an on-policy reinforcement learning algorithm, and deep learning techniques. In the **SARSA** algorithm, the agent learns an action-value function Q(s,a) by observing and interacting with the environment. The algorithm uses an epsilon-greedy exploration policy, in which the agent balances exploration (trying new actions) and exploitation (choosing the best action). The Q(s,a) represents the estimated value after taking an action in a given state. The SARSA process is as follows:

1. **State 1**: the agent starts in a state s and selects an action a based on its ϵ-greedy policy. After selecting an action, the agent receives a reward r and reaches a new state s. Q(s,a) is updated in the next state.[7]
2. **State 2**: In the new state s', the agent then selects another action a' based on the (ϵ-greedy policy like in state 1. The Q(s,a) function from state 1 is updated with the reward r and the estimated Q(s',a'. This process continues and the Q(s,a) continuously improves as the agent interacts with the environment.[7]

The SARSA formula is defined as this:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big]$$ [6]

Additionally, SARSA applies Temporal Difference (TD) learning to update action value function Q(s,a). This involves calculating the difference between expected and observed rewards, known as the TD error. This process enables the agent to better improve its predictions and learning.

Deep learning, on the other hand, uses artificial neural networks to learn data. The neural networks are made of layers of interconnected nodes, where each node is responsible for specific parts of the data. The networks learn from different layers as weights are assigned and adjusted during training [8]. This technique is quite useful for tasks such as image recognition where the agent can help find objects and features in images, such as people, animals, places, etc. It can also be used for natural language processes, finance, text to images, and more applications. [8]

Together SARSA and deep learning form Deep SARSA, where the agent uses a neural network to approximate action-value Q(s,a). The neural network is trained to predict q-values for station-action pairs, with updates based on the SARSA rule and TD learning with targets calculated using SARSA. The combination of on-policy learning and neural networks makes Deep SARSA both stable and adaptable. This enables it to tackle complex tasks effectively, particularly when the state space is as large or continuous as our environment.

3.3.1 Deep SARSA Related Works

Deep SARSA has been integrated into various research and studies. An article called *Learning with Deep SARSA & OpenAI Gym* demonstrates the implementation of Deep SARSA in an OpenAI Gym environment, like Cartpole. Throughout the experiments, Deep SARSA consistently performed high

rewards during testing episodes, ranging from 140 - 160 steps, demonstrating its ability to keep the cartpole balanced for a long period of time. This performance highlights Deep SARSA's ability to learn in dynamic environments. [7]

Furthermore, the GitHub repository *Deep Q-Learning and Deep SARSA in LunarLander-v2* uses both Deep SARSA and Deep Q-Learning in solving the LunarLander-v2 environment, which is a more complex task compared to the cartpole environment. In this study, Deep SARSA demonstrated strong results in learning optimal policies by balancing exploration and exploitation. The agent was able to consistently perform well that needed coordination of actions to land the spacecraft successfully. Based on the results, Deep SARSA demonstrates smoother reward convergence compared to Deep Q-Learning, with fewer fluctuations in its performance. This suggests that Deep SARSA may provide more stable learning dynamics, particularly in environments where consistent policy improvement is needed. [9]

3.4 Deep Q-Learning

Deep Q-Learning is a type of Reinforcement learning algorithm that combines Q-Learning with a deep neural network to make smarter decisions in complex environments.[10] It is like an extension of the basic Q-Learning algorithm which uses deep neural networks to approximate the Q-Values. Traditional Q-Learning works well for environments with a small and finite number of states but it usually struggles with the large and continuous state spaces due to the size of the Q-table. [11] Deep Q-learning overcomes the limitation by replacing the Q-table, with a neural network that has the ability to approximate the Q-Values for every state-action pair.

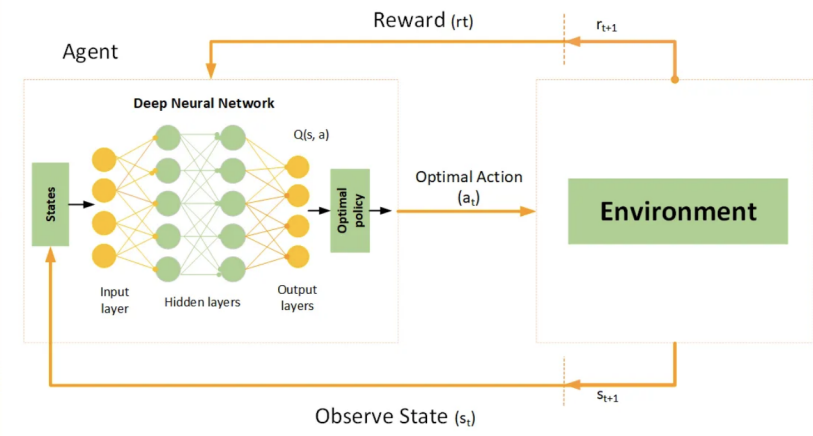

Figure 1: Structure of Deep-Q Learning

3.4.1 Structure of Deep-Q Learning

One of the major challenges with Deep-Q Learning is that some of the predictions made by the neural network can be unstable, making it hard for the algorithm to learn and execute correctly. To fix this issue there are two techniques that can be used:

1. **Experience Replay:** This is a replay memory technique where the algorithm uses a memory buffer to store experiences such as state, action, reward, next state. The network is trained on random mini-batches of experiences from this buffer which breaks the correlation between consecutive experiences and improving the sample efficiency. [12]

2. **Target Networks:** This is a second neural network, which is primarily used to calculate the target Q-values. The target network is updated less frequently than the main network to prevent constantly going back and forth in the learning. [13]

4 Hypothesis

Based on the algorithms we researched, we believe **SMDP** will perform the best over Deep SARSA and Deep Q-Learning. Due to its nature of incorporating states and actions with varying durations, SMDP is well suited to bridge maintenance tasks as repair actions and deteriorating processes do not follow uniform timelines. For instance, certain repairs might take weeks to complete, while inspections could be completed in only a matter of hours.

In addition, unexpected bridge repairs can arise at any moment due to sudden damage or unforeseen deterioration, requiring immediate action that cannot always be anticipated with prior planning. SMDP accommodates this unpredictability by allowing the model to adapt dynamically to varying time demands, making it possible to reallocate resources and prioritize certain repairs over others without negatively affecting long-term maintenance plans. By modeling these varying time durations, SMDP can prioritize repairs and anticipate the timing of maintenance needs, while also accounting for delays that could impact the overall condition of a bridge.

Overall, SMDP can be utilized to make proactive, resource-efficient decisions that improve safety, minimize costs, and extend the lifespan of a bridge by targeting the repairs needed at the exact optimal time.

5 Experiment Results

To evaluate the performance of each algorithm, we conducted experiments across 10,000 episodes. The results were analyzed by plotting the rewards, remaining budget, and deterioration over all episodes. We tracked the frequency with which each of the three possible actions in the action space was selected to gain insights into the decision-making behavior of each algorithm. Additionally, we calculated the cost efficiency, defined as the ratio of the average cumulative reward to the total cost sustained for a better understanding of how effectively each algorithm balances maximizing the reward with resource management. To assess the improvement in bridge condition over the 100-year period, we compare the initial and final states of the bridge for each algorithm. We also examine the percentage of the total budget spent by each agent during the process to analyze how well each algorithm manages the available budget.

5.1 SMDP Results

The following plot shows the cumulative rewards and also the rewards per episode from running SMDP with *InfraPlanner*:

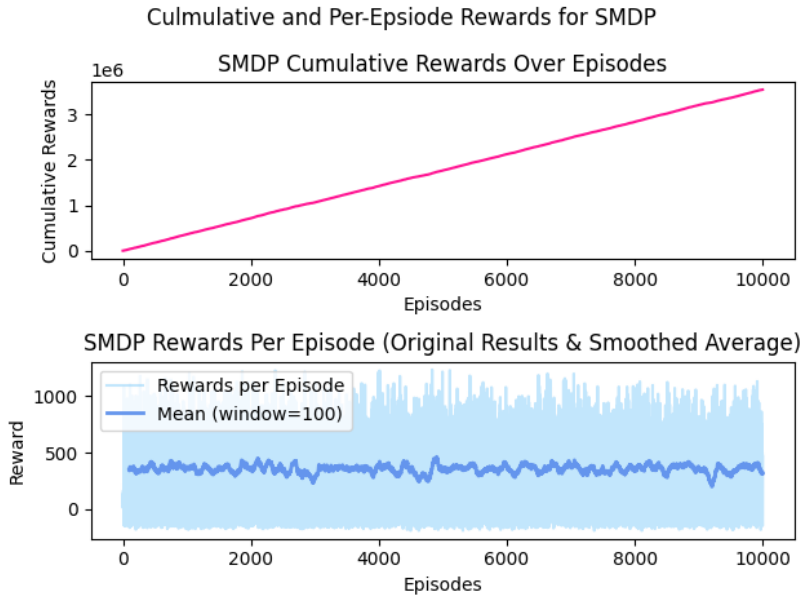

Figure 2: Cumulative and Per-Episode Rewards from Running SMDP with *InfraPlanner*.

We note from the steadily increasing cumulative rewards plot in **Figure 2** that the SMDP agent is learning as it interacts with *InfraPlanner*. **Figure 2** displays positive progress and a steady increase in performance, indicating that the agent is improving its actions over time to ultimately lead to a better bridge condition. The bottom subplot of **Figure 2** shows how the reward fluctuates as the agent learns over all the episodes. This might indicate that the agent is alternating between exploring and exploiting when selecting which action to take. We note that the reward seems to be stabilized around the $300 - 400$ range.

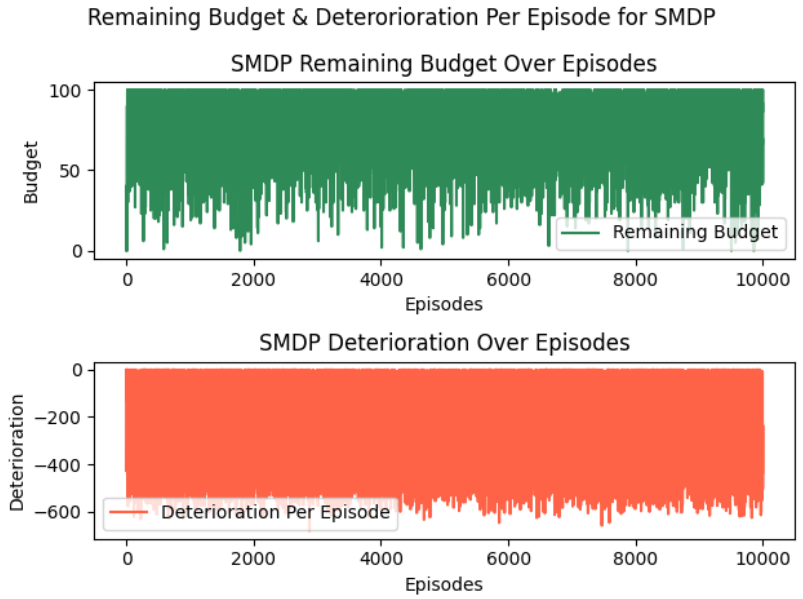

Figure 3: Remaining Budget & Deterioration Per Episode from Running SMDP with *InfraPlanner*.

From the above **Figure 3**, we learn that the budget fluctuates significantly, likely due to the varying costs associated with different actions. This variability reflects the strategy of the SMDP agent, which changes based on the chosen action and the remaining budget available. From the graph, we observe that the remaining budget typically falls between 0-30% across the majority of episodes. It rarely exceeds 30%, with some reaching around 50%, although such instances are infrequent. The fact that the remaining budget rarely surpasses 30% suggests that the SMDP agent may not be optimizing its budget allocation as well as it could be. The agent may need to improve its ability to balance the condition of the bridge with more efficient budget management, it could be spending more to further improve the bridge.

The deterioration plot shows that, across all episodes, the deterioration remains consistently between $500 - 600$. This suggests that the bridge is deteriorating at a relatively stable rate over time. In real-world context, this might reflect the natural wear and tear that a bridge sustains as it ages.

Initial to Final Bridge Condition	*40* to *85*
% **Bridge Condition Improvement**	*75* %
% **Budget Spent**	*31* %
Cost Efficiency	*11.44*
% **Each Action Taken**	**do nothing:** *93.23* % **maintenance:** *3.35* % **replace:** *3.42* %
Average Cumulative Reward	*354.49*

Table 2: Results from running SMDP with *InfraPlanner* over 10,000 episodes.

From **Table 2**, we observe that the bridge condition score improved from 40 to 85, representing a 75% improvement. This percentage reflects the progress made by the agent toward achieving the maximum possible score of 100. By the thresholds defined with our reward function in **Section 3.1.2**, the bridge is now considered to be in a "good" condition as it now has a condition of 80 or above at 85. These results demonstrate a significant improvement, showing the SMDP agent's ability to effectively optimize its maintenance strategies to enhance the structural condition of the bridge.

In spite of this notable improvement, the SMDP agent only spent 31% of its total budget. While this suggests cost-effective performance, it also indicates that the agent has room for improvement, as taking advantage of the available budget could potentially improve the condition of the bridge even further. The cost efficiency was 11.44, which means that for every dollar spent, a reward of 11.44 was generated. This showcases a high level of cost-effectiveness.

In terms of the actions taken, SMDP highly favors the **do nothing** action the majority of the time. These results algin with real-world decision-making, where maintenance is held off until absolutely necessary, especially when bridge conditions are deemed good enough for the time being. However, this strategy might not be optimal for a model whose primary goal is to maximize the overall condition of the bridge. We aim to explore this further in our future work.

5.2 Deep SARSA Results

The following plot shows the cumulative rewards and also the rewards per episode from running Deep SARSA with *InfraPlanner*:

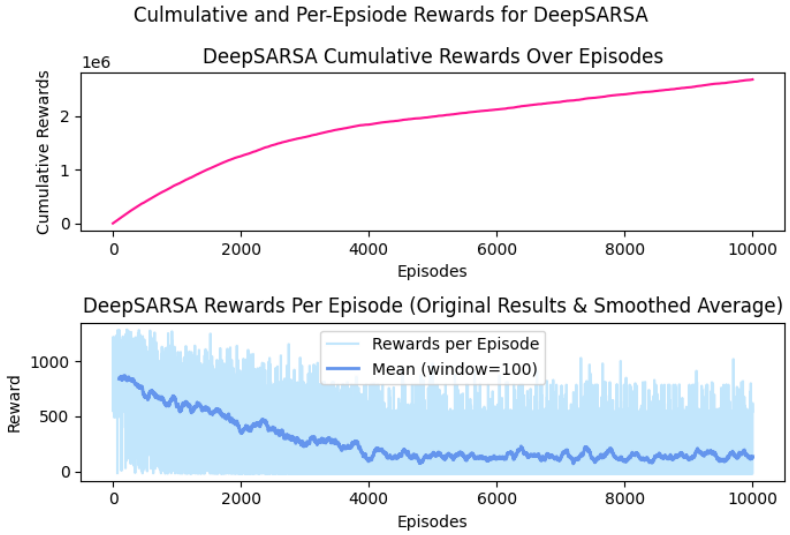

Figure 4: Cumulative and Per-Episode Rewards from Running Deep SARSA with *InfraPlanner*.

From the above **Figure 4**, we can observe a steady increase in cumulative rewards initailly. At the end of 10,000 episodes, the cumulative rewards have reached a plateau, which suggests stabilization in learning. The average cumulative reward is around 268. The second graph presents the rewards per episode and a smoothed average. Initially, the rewards are quite high but decrease as the agent begins to explore and update its strategy. Over time, the per-episode rewards stabilize, indicating that the agent is successfully converging.

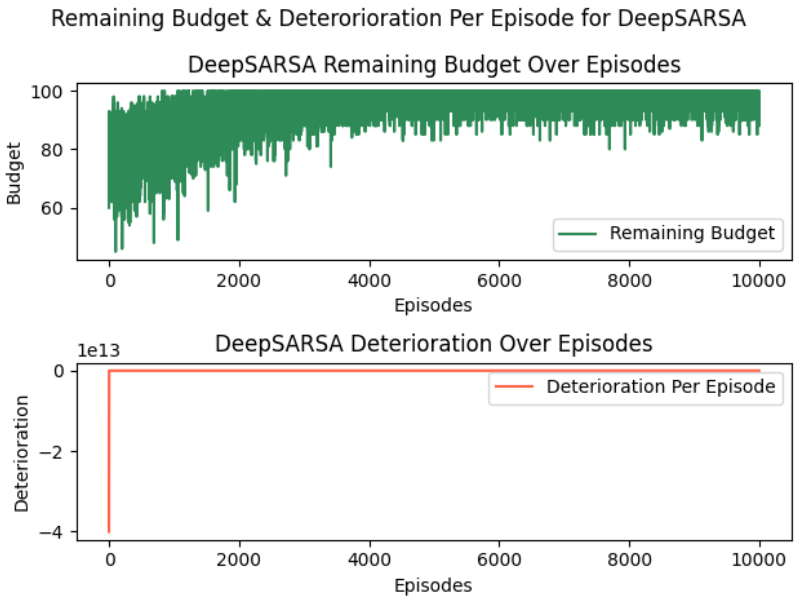

Figure 5: Remaining Budget & Deterioration Per Episode from Running SMDP with *InfraPlanner*.

From the above **Figure 5**, we learn that the budget fluctuates as the agent explores different strategies for resource allocation. Over time, the remaining budget stabilizes, indicating the agent has learned

to optimize spending, reaching an optimal policy. Moreover, in the second graph, the deterioration is substantial, reflecting the poor structural conditions. However, the agent quickly learns to mitigate deterioration, and the values remains zero. We believe this is a possible bug in our code because the deterioration drops sharply and remains at zero for the rest of the episodes which is a very odd. This is something we will have to investigate further in our study.

Initial to Final Bridge Condition	*40* to *68*
% **Bridge Condition Improvement**	*46.67* %
% **Budget Spent**	*5* %
Cost Efficiency	*53.66*
% **Each Action Taken**	**do nothing:** *98.44* % **maintenance:** *0.76* % **replace:** *0.80* %
Average Cumulative Reward	*268.28*

Table 3: Results from running Deep SARSA with *InfraPlanner* over 10,000 episodes.

The bridge condition score improved from 40 to 68, representing a 46.67% improvement. This shows that the agent optimized maintenance strategies to enhance structural conditions effectively. Despite achieving a significant improvement, only 5% of the total budget was spent, which is something we need to further evaluate as we were hoping the agent would use more of the budget to ensure better conditions of the bridge. The primary reason why budget spent was low was because the agent chosen **do nothing 98.44 %** of the time, indicating that the agent prioritized waiting out situations where conditions did not require intervention. This may not be most effective method and is something we will have to look into further.

5.3 Deep Q-Learning Results

The following plot shows the cumulative and per-episode rewards for the Deep Q-Learning algorithm:

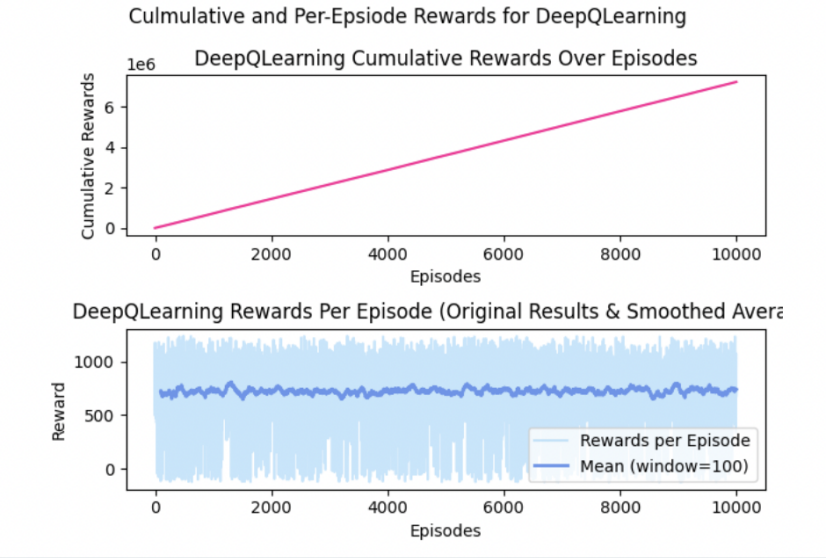

Figure 6: Cumulative and Per-Episode Rewards from Running Deep Q-Learning with *InfraPlanner*.

As you can see from **Figure 6**, the rewards grow consistently and steadily showing us that the agent is consistently learning and improving its strategy throughout the training process. By the end of the 10,000 episodes, the cumulative reward reaches over 6 million, showing us the effectiveness of the algorithm in improving long-term goals. The second plot shows us the Per-Episode Rewards earned in each episode including a line showing us the smoothed average using a window of a 100 episodes. In the beginning, the rewards fluctuate significantly due to the agent exploring different actions and learning about the environment. However, we see that over time the fluctuations decrease and the rewards stabilize around a mean-value of 500. This shows us that the agent has figured out the best way to make decisions by trying out new actions when needed (exploration) and sticking with what works well (exploitation).

As with **Figure 7** below, we now see the results for the remaining budget and deterioration over episodes. In the top subplot, we see how the agent manages the remaining budget across episodes. Initially, the budget fluctuates as the agent is experimenting with different actions to improve the bridge condition. As the training continues and progresses, the budget stabilizes around a consistent range which helps suggest that the agent has learned how to allocate resources efficiently and avoid our initial concern of unnecessary spending. The bottom subplot in **Figure 7** tracks the deterioration rate over episodes. It is interesting because the deterioration rate started at an extremely high value, but it then stabilizes to a near zero for the rest of the episodes. This is likely to be a problem with out code but this behavior also can suggest that the agent prioritizes actions that mitigate long-term wear and tear, maintaining the condition of the infrastructure.

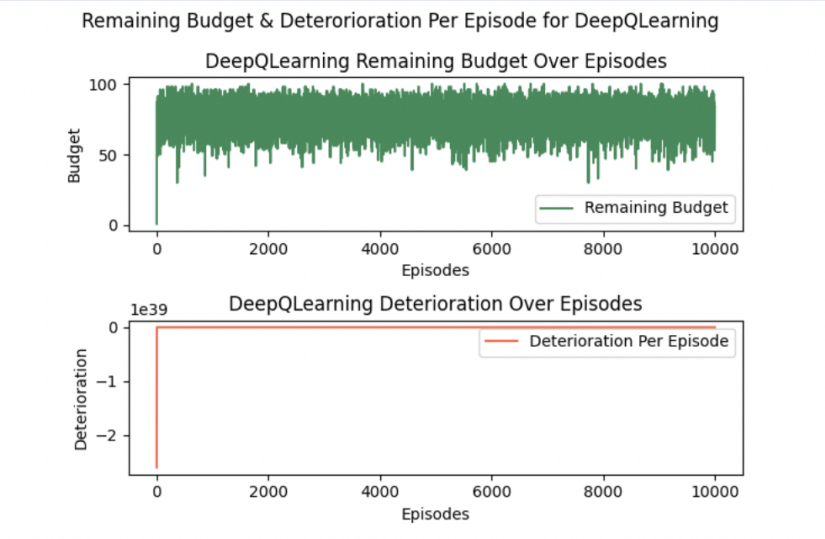

Figure 7: Remaining Budget & Deterioration Per Episode from Running Deep Q-Learning with *Infra-Planner*.

Table 4 below displays a run-down on how the data is being portrayed after running the Deep Q-Learning algorithm on bridge maintenance. We see that the initial bridge condition starts at 40 and improves to 57 by the end of the training showing a 28.33 percent improvement. The improvement was moderate compared to some other approaches, highlighting the agent's ability to get better infrastructure conditions with limited actions and resources. In terms of budget, the agent spends only 16 percent of the total budget to reach that goal of improvement. This shows that the agent is very efficient with the spending– saving as much money as possible while still improving the condition of the bridge. For every unit of the budget spent, the bridge improved drastically, with a cost efficiency of 45.20. This means that the agent focuses on actions that make the biggest difference. The agent chooses to do nothing 93.32 % of the episodes showing us that it is avoiding unnecessary interventions and is conservative on the resources. With only 3.35 % of the time, the maintenance is necessary, but probably more towards more server and high-priority situations where it can make a meaningful impact. Lastly, with 3.34 % of the time, the action is the most expensive but could have been used strategically to address the deterioration.

Initial to Final Bridge Condition	*40* to *57*
% **Bridge Condition Improvement**	*28.33* %
% **Budget Spent**	*16.0*%
Cost Efficiency	*45.20*
% **Each Action Taken**	**do nothing:** *93.32* % **maintenance:** *3.35* % **replace:** *3.34* %

Table 4: Results from running Deep Q-Learning with *InfraPlanner* over 10,000 episodes.

5.4 Further Explanation of the Results

We note both confirmations and discrepancies with our initial hypothesis from **Section 4**. In terms of bridge improvement, SMDP performed the best, achieving a 75% improvement of bridge condition. This is significantly higher compared to Deep SARSA's 47% improvement and Deep Q-Learning's 28%. These results demonstrate that SMDP highly excelled at optimizing maintenance strategies to enhance the condition of the bridge, making it the most effective algorithm in terms of structural improvement.

However, while SMDP was highly effective at improving the bridge condition, it was the least cost-efficient among the three different algorithms. The cost efficiency of SMDP was 11.44, meaning that it generated a reward of only 11.44 for each dollar spent. In contrast, Deep SARSA achieved the highest cost efficiency with a ratio of 53.66, which indicates that it provided a much better return on investment in terms of reward gained relative to the amount of budget spent. Deep Q-Learning also outperformed SMDP in this regard, with a cost efficiency ratio of 45.20, highlighting its ability to also balance cost and performance well.

In terms of reward, Deep Q-Learning is the most reward-efficient algorithm, achieving an average cumulative reward of 723.23. This is significantly higher than SMDP's average cumulative reward of 354.49 and Deep SARSA's of 258.28. These results suggest that Deep Q-Learning is more consistent in generating higher rewards, likely due to how it follows a policy to optimize focusing on maximizing both immediate and cumulative rewards.

SMDP balances actions, performing the best in maintaining bridge condition but inefficient in rewards and cost. Deep SARSA balances condition and reward well, but relies too heavily on doing nothing which limits improvements. Deep Q-Learning on the other hand maximizes rewards and efficiency with greedy updates, but struggles with long-term condition improvement. Ultimately, the best algorithm here depends on the specific criteria used for evaluation.

If the focus is primarily on improving the condition of the bridge, then SMDP is the best. However, if the priority is instead to achieve the best results while also maintaining cost efficiency, Deep SARSA provides a more balanced model, although it is quite static. However, when running Deep SARSA multiple times, we observed varying outcomes — ranging from achieving a 100% success rate in certain conditions to attaining a less favorable condition with a score of 14 due to significantly high deterioration. This variability may be due to factors such as the sensitivity of Deep SARSA to the hyperparameters we have in the code and the stochastic nature of the environment we have. While Deep SARSA can adapt to dynamic environmental changes, these factors can still influence its performance, leading to inconsistencies in certain scenarios when running the algorithm. On the other hand, if reward optimization is the biggest goal, Deep Q-Learning would be the top algorithm. It can prioritize immediate and long-term rewards through greedy updates, selecting actions that achieve the highest expected value. Additionally, when we ran Deep Q-Learning it consistently performed the best compared to the other two algorithms demonstrating its ability to adapt quickly to environment changes.

Overall, these results highlight the trade-offs that occur with each algorithm and how the prioritization of different tasks ultimately determines which algorithm is the very best one.

6 Extension

We hope to improve our existing code to further improve our research. We aim to:

- **Training:** extend the training duration to over 1,000 to ensure the agent is fully exploring the complex environment we have. This will ensure that the agent is interacting with the environment

allowing it to learn better policies to improve its predictions. This may also prevent the agent from doing nothing a majority of the time and do other useful actions.

- **Deterioration:** improve the deterioration tracking code to ensure it is accurately captured over 10,000 episodes, particularly for Deep SARSA and Deep Q-learning where deterioration remained zero after the first episode. We can add a deterioration debugging code to see whether the deterioration metric is being calculated at each episode.
- **Environment Actions:** break down maintenance and replacement actions into smaller tasks to make them more realistic for real bridge maintenance simulations. This will include defining specific maintenance tasks, such as cleaning, sealing cracks, or painting, and replacement tasks. [14]. By better categorizing these actions, we can make better decision-making process.
- **Budget Management:** help the agent spend the budget more wisely while ensuring the bridge is in good condition. We want to maximize the efficient use of the budget without severely overspending or underspending like Deep SARSA. We can do this by creating a sensitivity analysis to test various metrics, ensuring that the budget is effectively used while producing realistic results to bridge conditions. We can compare these results with real-world applications of budget management.

After improving upon the existing code, we hope to apply some sort of hierarchical RL framework. This advanced model will prioritize specific bridge components—like beams, pavement, and slabs—based on their urgency and condition. By addressing critical tasks first, the model can ensure smarter and more targeted resource allocation. Lastly, we want to focus on optimization goals. We want to improve cost by reducing costs while also ensuring budget is efficiently used, to enhance resource utilization. We also want to improve reinforcement learning policies to adapt to dynamic environments, ensuring robust decision-making across varying conditions of a bridge. Lastly, we want to extend and implement the framework to real-world bridge systems to get real-time results. Overall, these future extensions and advancements will enable us to fully integrate reinforcement learning to bridge infrastructure systems, paving the way to better structuRL integrity of bridges.

7 Conclusion

The benefits of the approach explored in this project enhance both operational efficiency and safety. By tactically planning repairs and prioritizing critical needs, the system reduces costly "emergency" repairs and minimizes disruptions to the public. Although initially designed for a single bridge, with simple modifications to the environment and the reinforcement learning algorithms employed, the model can be adapted to manage large networks of bridges, making it valuable for significant infrastructure systems. The clear-decision making structure provided by reinforcement learning is very clear and easy for maintenance teams to interpret, which helps enhance collaboration and ensures that the actions that are being taken go hand-in-hand with the infrastructure needs. All things considered, our *structuRL* bridge maintenance model is *structuRLly* sound.

References

[1] Luther Costa. How old are most bridges in the us?, June 2024.

[2] American Road & Transportation Builders Association. 2024 ARTBA Bridge Report. Technical report, American Road & Transportation Builders Association, 2024.

[3] Baldwin, Shawn. Why u.s. bridges are in such bad shape, March 2022.

[4] Zachary Hamida and James-A. Goulet. Hierarchical reinforcement learning for transportation infrastructure maintenance planning. *Reliability Engineering and System Safety*, 235:109214, 2023. Department of Civil, Geological and Mining Engineering, Polytechnique Montreal, 2500 Chem. de Polytechnique, Montreal, H3T 1J4, Quebec, Canada.

[5] Zachary Hamida and James-A. Goulet. Infraplanner github repo, 2023.

[6] Andrew G. Barto and Richard S. Sutton. *Reinforcement Learning: An Introduction*. Westchester Publishing Services, 2018.

[7] Gelana Tostaeva. Learning with deep sarsa & openai gym, n.d.

[8] Google Cloud. What is deep learning?, n.d. Accessed: December 18, 2024.

[9] JohDonald. Deep q-learning and deep sarsa in lunarlander-v2, 2024.

[10] The deep q-learning algorithm - hugging face deep rl course.

[11] GeeksforGeeks. Deep qlearning.

[12] Papers with code - experience replay explained.

[13] Samina Amin. Deep q-learning (dqn).

[14] Bridge Masters, Inc. Approaching bridge maintenance efficiently, 2024.