Adversarial Attacks Against Online Learning Agents

Alicia Li
aliciali@mit.edu

Matan Yablon
matiyablon@gmail.com

Mentor: Mayuri Sridhar
Massachusetts Institute of Technology
MIT PRIMES

Abstract
Consider a typical streaming problem, where an agent dynamically interacts with its environment to learn an optimal behavior. Such methods are used in a variety of applications, including playing Atari games and robotic hand manipulation. We analyze an agent that learns the rewards of each path in its environment, which can be modeled as determining the edge weights of a graph. We study an agent that follows an $\epsilon$-greedy sampling strategy because this model is widely used and has been successfully applied to many problems. However, in recent years, numerous attacks have been devised against graph learning algorithms, with some methods exploiting graph structure and node features. To ultimately create a robust graph streaming algorithm based on $\epsilon$-annealing, we first construct, implement, and analyze worst-case attacks against random-sampling and $\epsilon$-greedy victim models. Our adversarial strategy exploits path overlaps and stalls the victim to effectively increase the corruption budget.

1 Introduction

1.1 Background
The use of online reinforcement learning to solve many classes of problems has skyrocketed over the last decade and has achieved stellar results in several tasks and fields [14, 19, 23]. Online reinforcement learning (RL) is a branch of online learning distinct from both supervised and unsupervised learning, where an agent dynamically interacts with an environment to garner information about it and learn a policy, which dictates its behavior. It does not require the use of large, labeled data sets in the way supervised learning does; how could a computer learn to play chess by shoveling chess game data through a giant neural network? Instead, it simply needs access to the training environment (e.g. chess, Go, or Atari), and the computer plays on its own and gradually builds up skill. The goal of the agent is to find a policy that optimizes some objective, usually pertaining to the cumulative reward from the environment. For example, an agent learning to play Atari games might try to learn a policy that allows it to get the most points in the game before it dies [14].

Because the agent is simultaneously trying to gather information about the environment while also maximizing reward, the agent is said to balance exploration with exploitation.
In an often-used model termed $\epsilon$-greedy, the parameter $\epsilon$ represents the relative weighting between exploration and exploitation, as it is the probability that controls whether we choose the best known action (exploitation) or choose a random action (exploration).

However, many online learning algorithms suffer from poor performance upon deployment, owing to significant differences between training conditions and test conditions, which can be caused by human biases, modeling errors, or actual adversaries. These problems can be rectified by training the agent to be robust; that is, maintain some degree of performance despite a shift from initial conditions. To model such disturbances, many works introduce an adversary, which aims to disrupt or degrade the victim agent’s performance by perturbing various aspects of the learning environment such as the victim’s rewards [12, 29], the victim’s observations [10], the victim’s memory, and the environment transition dynamics [27, 16].

Yet, the question of how myopic sampling strategies such as $\epsilon$-greedy perform under such adversarial corruptions remains unanswered. $\epsilon$-greedy is one of the most popular sampling strategies due to its simplicity and empirical success for many practical problems [5]. Studying epsilon-greedy’s performance under adversarial conditions is vital to understanding the robustness of these numerous implementations. What happens when an $\epsilon$-greedy agent is exposed to an adversary? How much can an adversary degrade an $\epsilon$-greedy victim’s performance by poisoning the training time data it learns from? These questions, in the context of graph-structured environments, drive the motivation behind this paper.

In particular, we fill a gap in the field of graph streaming algorithms, where an agent has access to a stream of graph data that it uses to make decisions. In our adversarial model, the victim learns a behavior based on the stream of its previous graph trajectories, while an adversary needs to select when and which edges to perturb, with the goal of degrading the victim’s performance as much as possible. We provide analysis of maximum corruption for both a random-sampling victim and the epsilon-greedy victim, along with theoretical and experimental analysis of the corruption’s degradation over time. These investigations are fundamental to understanding the robustness of a highly adaptive model in streaming settings.

2 Related Work

2.1 Streaming Algorithms

In the typical streaming algorithm problem, the learning agent dynamically interacts with its environment and utilizes the stream of data generated to adjust its behavior incrementally. Some real-world examples include making trading decisions in financial markets and real-time speech processing [15, 9].

Q-Learning

Q-Learning is one of the most popular online RL algorithms and the algorithm we implemented empirically to test some of our attacks on. In this method, each pair of states and actions is given a $Q$-value which prescribes which action to take next [21]. These $Q$-values are updated according to the data stream generated as the agent navigates its environment.

Attacks on Streaming Algorithms. Previous works on attacking streaming algorithms include Gong et al. [9] and Natali et al. [15]. The former of which uses partially observable decision process concepts from reinforcement learning such as imitation learning, while the
latter draws parallels to the $k$-secretary problem from optimal stopping theory. These works consider adversaries which do not have access to the entire original data set and must learn alongside the victim. Our setting considers a more specific and stronger adversarial model which has access to the generator of the stream’s contents, the MDP’s structure and rewards, along with whether the victim is currently sampling randomly or greedily, but it does not have access to the victim’s current Q-values.

Moreover, we focus on a poisoning attack, in which the adversary corrupts the stream during the lesser-explored training process instead of the evaluation phase. This is more computationally difficult as the adversary needs to factor in the victim’s ever-changing policy into its corruptions. Previous work on poisoning streaming algorithms include NETTACK, which exploits graph structure and node features to attack deep learning models relying on graph convolutions [30]. Our work also exploits graph structure, but for the different goal of maximizing adversarial budget to manipulate the perceived reward of paths.

Zhang et al. [29] is one of the few adaptive poisoning attacks against an $\epsilon$-greedy Q-learning victim. This work investigates manipulating the victim to follow the target policy in some number of important states, a subset of the environment’s state space. The paper implements a Fast Adaptive Attack that ranks each of these important states based on distance from the start state. Instead of attacking all states at the same time like Xu et al. [27], Zhang et al. [29] attacks each important state one at a time, allowing the target policy to be preserved at each state. Although the paper proves that for its “Non-Adaptive Attack,” the objective value and $\epsilon$-greedy’s covering time is $O(e^{|S|})$, so that attack is slow. We aim to further investigate what exactly an optimal adversarial strategy looks like against an $\epsilon$-greedy victim.

2.2 Sampling Strategies

Ideally, an agent is able to operate well in stochastic settings, wherein the agent can sample rewards that are not subject to adversarial corruption. The manner in which the agent samples from the MDP is crucial to its performance, and this work considers $\epsilon$-greedy as the sampling strategy of focus.

**Purely Random Sampling or Purely Greedy Sampling.** One naïve sampling approach is to simply take a random action. While this strategy carries the advantage that no single path will become excessively favored and the agent will get a fairly accurate picture of the MDP given enough samples, it is not efficient because much time is wasted sampling paths that are already known to be suboptimal.

In contrast to random sampling, the agent could instead always sample the path that, according to its current knowledge, yields the most reward. This strategy is also primitive because such an agent would continually choose the first path it samples without variation—it would have no ability to test other paths to see if they are better.

Ultimately, neither strategy is robust. A victim operating under these strategies would be easily susceptible to adversarial corruption.

**Epsilon-Greedy.** $\epsilon$-greedy thus represents a balance between these two approaches. Although it is one of the most popular sampling strategies, its theoretical guarantees are underexplored. Simply stated, $\epsilon$-greedy policy involves sampling a random suboptimal ac-
tion\textsuperscript{1} $\epsilon$ fraction of the time, and sampling the highest-$Q$-value action $1 - \epsilon$ fraction of the time \cite{21}. That is,

$$\pi(s) = \begin{cases} \arg \max_{a \in A(s)} Q(s, a) & \text{with probability } 1 - \epsilon, \\ \text{rand}(A(s) - \arg \max_{a \in A(s)} Q(s, a)) & \text{with probability } \epsilon \end{cases}$$

Previous works about $\epsilon$-greedy’s theoretical guarantees include Dann et al. \cite{5}. In order to categorize the kinds of problems $\epsilon$-greedy performs well in, this paper proposes a complexity measure, the myopic exploration gap, which correlates to MDP structure, exploration policy, and value function. Moreover, the paper shows that sample-complexity of myopic exploration strategies, such as $\epsilon$-greedy, is proportional to the inverse square of the myopic exploration gap.

**Epsilon Annealing** \cite{21}. A slight variation on $\epsilon$-greedy, annealing entails the gradual decrease of $\epsilon$’s value over time. The justification for such an approach is that as time goes on, the agent is expected to attain a more precise picture of the MDP, and consequently, there is less uncertainty in its decisions. Therefore, the chance with which it makes a random and exploratory decision should decrease over time.

**Upper Confidence Bound (UCB) Action Selection.** An alternative to $\epsilon$-greedy, UCB weights actions according to their potential to actually be optimal \cite{21}, as per the equation

$$a(t) = \arg \max_a \left[ Q_t(a) + c \sqrt{\frac{\ln t}{N_t(a)}} \right]$$

The term being added to the $Q$-value represents an upper-bound on our uncertainty about the action’s actual value. As time increases, that term approaches 0 (since $\lim_{t \to \infty} \frac{\ln t}{t} = 0$), so our uncertainty naturally decreases with time \cite{21}.

This work does not consider UCB, which is significantly more complex mathematically speaking, opting instead to focus on $\epsilon$-greedy strategies. After all, $\epsilon$-greedy is well-known and commonly used so there is practical benefit in focusing on it \cite{5}.

### 2.3 Adversarial Attacks and Defenses

An adversarial attack in the context of reinforcement learning is any protocol employed against the agent, with the intent of negatively influencing its behavior \cite{8}. That intent may include coercing the victim into learning a specified target policy or degrading the victim’s performance in test-time.

Previous work in Adversarial RL includes Rakhsha et al. \cite{17}, which aims to poison the environment such that the target policy is optimal in this poisoned setting. Thus, the victim ends up learning the target policy by optimizing its rewards in the poisoned environment. This paper only considers a universal perturbation: a single, non-adaptive attack at the very start of the training period. While we also consider a poisoning attack, our strategy is online, meaning the adversary continues to attack throughout the training process.

\textsuperscript{1}This work considers a victim who makes a decision exactly once per episode. Thus, if making a random decision, it chooses uniformly from the set of all paths it can take through the MDP, other than the optimally perceived path.
Essentially, a basic approach to defense is to train an agent over a wide variety of environments with the goal of learning a policy that performs well when evaluated in an arbitrary environment. However, policies that perform well in the average case may perform poorly on a small fraction of particularly difficult environments and are therefore susceptible to worst-case adversarial attacks [6].

Therefore, the Robust RL objective is to find a policy that performs optimally under the worst-case environment scenario. The victim tries to maximize its reward by choosing an optimal policy, while the adversary tries to force the victim to get lower reward by choosing the worst environment drawn from some set of possible ones [8]. The Robust RL objective is to find a policy \( \pi \) that satisfies:

\[
\max_{\pi} \min_p \mathbb{E}_{\pi,p} \left[ \sum_t R_t \right],
\]

where \( p \) is the environment and \( R_t \) is the reward at time \( t \).

There are several methods that satisfy the Robust RL objective [24, 13, 1] that all incorporate the maximin framework above in unique ways. For instance, Tessler, Efroni, and Mannor [24] introduces an adversary with the ability to perturb the agent’s actions during training time.

2.4 Our Focus

In our research, we want to tackle the following questions:

1. How can we identify the optimal adversarial strategy in this strongly adaptive model?
2. How well do stochastic online learning models perform under our adversarial strategy?
3. How does the structure of an MDP affect the performance of an adversarial strategy?

3 Model

3.1 MDPs

The environment in which the RL agent operates is often modeled as a Markov Decision Process (MDP).

Definition 3.1 (Markov Decision Process). An MDP is characterized by the quadruple \((S, A, R, P)\), where:

- The state space \( S \) is the set of all possible states the agent may enter,
- The action space \( A \) is the set of all possible actions the agent may take,
- The reward function \( R : S \times A \rightarrow \mathbb{R} \) prescribes the reward attained by the victim upon taking a given action in a given state,
- The transition dynamics function \( P : (s', s, a) \rightarrow [0, 1] \) denotes the probability of the agent transitioning to a new state \( s' \) given its presence in \( s \) and taking the action \( a \).
3.2 Adversarial Model

We consider an adversary that has access to all the rewards and transition probabilities in the MDP. For every path traversal in the victim samples in the stream, the adversary has probability $p$ of corrupting the path by a maximum magnitude of $\delta$. This style of corruption suits the incremental nature of the online victim’s learning process.

3.3 Maze MDP

Although our theoretical and general results apply to a more general class of MDPs, a simpler and more concrete way of conceiving of an MDP is a maze challenge situated in a grid. This setting can be easily visualized and also can represent a generalized MDP with each open slot in the maze representing a state. The action space is \{RIGHT, LEFT, UP, DOWN\}. However, our maze is nondeterministic, so choosing the action right does not guarantee the agent will take a step in that direction.

An example grid layout is depicted above. The agent must start at the green square and, without ever visiting a previously traversed square, make its way to a red terminal state. The reward it acquires is represented by numbers strewn across the maze; if the agent lands in a given square, receives this reward. When no attack is supplied and $p$ is deterministic, the optimal policy is the one that leads the agent along the path highlighted by the blue line.

A representation of the learned Q-values in this maze is depicted below. It was created by coloring each square $s_t$ with the average Q-value over all four actions that can be taken in that square. Lighter values correspond to higher values, and the deep purple corresponds with 0 value:
4 Constructing Adversarial Strategies

The goal of the adversary is to manipulate the victim into returning a path with lowest reward possible after \(N\) episodes. Thus, a naive strategy is to perturb the optimal path \(P^*\) downwards and the path \(\hat{P}\) with lowest reward possible upwards. The paths are corrupted by the maximum of \(p\delta\) each for a budget of \(2p\delta\), meaning that \(R(\hat{P}) < R(P^*) - 2p\delta\).

\[
\begin{align*}
R(AD) &= 3 \\
R(AE) &= 6 \\
R(AC) &= 12
\end{align*}
\]

(a) Without adversarial corruption, \(AC\) is optimal. \(p\delta = 4\)

\[
\begin{align*}
R(AD) &= 7 \\
R(AE) &= 6 \\
R(AC) &= 8
\end{align*}
\]

(b) With the naive corruption strategy, the adversary corrupts edge \(BD\) upwards and \(AC\) downwards. However, there is not enough corruption to trick the victim into choosing \(AD\).

\[
\begin{align*}
R(AD) &= 9 \\
R(AE) &= 8 \\
R(AC) &= 8
\end{align*}
\]

(c) By corrupting edge \(AB\) when \(P_2\) is traversed, the adversary adds free corruption of 2 to its budget. Now, the adversary can trick the victim into choosing \(AD\).

Figure 1: \(p\delta = 4\)

The total amount of corruption on \(\hat{P}\) is \(N_p \cdot p\delta\) averaged among all of the samples. However, not all the edges on \(\hat{P}\) are sampled the same number of times. For instance, in section 4, edge \(AB\) is traversed \(\frac{1}{2}N\) while \(BD\) is traversed \(\frac{1}{4}N\) times. If the adversary were to corrupt \(AB\) each time \(P_1\) is traversed, \(AD\)'s reward would only be perturbed by \(\frac{p\delta N}{\frac{1}{2}N} = \frac{p\delta}{2}\). On the contrary, if the adversary corrupted \(BD\) instead, \(AD\)'s reward would be perturbed by \(\frac{p\delta N}{\frac{1}{4}N} = p\delta\). Thus, it is more optimal for the adversary to corrupt the edge with smallest degree, which is the disjoint edge \(BD\). Additionally, the non-disjoint edge \(AB\) can be corrupted when \(AE\) is traversed. This allows the adversary to perturb \(AE\)'s reward by more than just \(p\delta\).

We call this additional perturbation free corruption.

Definition 4.1 (Free Corruption). For edge \(e_s \in E_s\) on path \(P_s \in S_s\), corrupting \(e_s\) when \(P_s\) is traversed will add a free corruption \(f_{P_s,P_f} = \frac{p\delta N_{P_s}}{e_{P_s,P_f}}\) to the adversarial budget, where \(P_f = \hat{P} \lor P^*\).

Free corruption depends on the number of paths non-disjoint to \(\hat{P}\) and \(P^*\). As \(|S_s|\) increases, so does the fraction of traversals in which shared edges are corrupted. Moreover, free corruption depends on the victim’s sampling strategy. During the warm start, \(f_{P_s,P_f} = \frac{p\delta}{N}\) because each path is sampled equally. Therefore, the optimal adversarial strategy is: for every path, corrupt the edge it has in common with either \(\hat{P}\) or \(P^*\) and is also on the least number of paths.

For a small \(N - n\), we provide an algorithm which outputs the lowest reward path that the adversary can force the victim to choose at the end. The adversary calculates its budget for each path. Then, it returns the path with least reward while still having sufficient budget to trick the victim into choosing this path.
Algorithm 1 Adversarial Attack For Small \( N \)

**input:** MDP \( M \)

1: \( P \leftarrow P^* \quad \triangleright \text{Adversary finds the optimal path to trick victim into picking} \)
2: \textbf{for all} \( P_i \in M \land P_i \neq P^* \) \textbf{do} \quad \triangleright \text{Iterate through suboptimal paths}
3: \( b_i \leftarrow 2p\delta \quad \triangleright \text{Default corruption budget is} \ 2p\delta \)
4: \textbf{for all} \( P_s \notin \{P_i, P^*\} \) \textbf{do} \quad \triangleright \text{Iterate through other paths to calculate free corruption}
5: \( f_{P_s} \leftarrow 0 \)
6: \textbf{for all} \( e \in P_s \) \quad \triangleright \text{Iterate to find the edge with greatest free corruption}
7: \quad \textbf{if} \ (e \in P^* \oplus e \in P_i) \land \frac{1}{a_e} > f_{P_s} \ \textbf{then} \quad \triangleright e \ is \ on \ exactly \ one \ of \ the \ paths \ \text{we wish to corrupt}
8: \quad f_{P_s} \leftarrow p\delta \frac{1}{a_e} \quad \triangleright \text{Calculate free corruption from corrupting} \ e; \quad a_e := |\{P : e \in P\}| \end{array}
9: \textbf{end if}
10: \textbf{end for}
11: \quad b_i \leftarrow b_i + f_{P_s} \quad \triangleright \text{Add free corruption from} \ P_s \ \text{to budget}
12: \textbf{end for}
13: \quad \textbf{if} \ R(P_i) < R(P) \land R(P^*) - b_i < R(P_i) \textbf{then} \quad \triangleright \text{Compare current path to best path found thus far}
14: \quad P \leftarrow P_i \end{array}
15: \textbf{end if}
16: \textbf{end for}
17: \quad \textbf{output} \ P, f

### 4.1 Proof of Optimality of Algorithm 1

We begin our proof by reducing the task of showing that this algorithm chooses the path with lowest reward to the task of showing that this algorithm calculates the budget optimally. For the sake of contradiction, suppose the algorithm did not pick the path with lowest reward. This means the adversary could have found a higher budget for some other path with a lower reward, meaning the adversary did not choose the edges to corrupt optimally. Therefore, we prove this algorithm picks the set of corrupted edges optimally.

**Lemma 1.** There exists some edge such that if the adversary focuses all perturbation on this edge, it will be more optimal than if the adversary distributes its perturbation.

**Proof.** Suppose that the adversary corrupts the edge \( e \). In order for the adversary to increase its effective corruption budget, it would have to sometimes corrupt an edge \( e' \) which has a higher corruption. In this case, corrupting just \( e' \) is more optimal than the weighted sum of \( e \) and \( e' \). Thus, choosing multiple edges to corrupt in a traversal is not optimal, and it is more favorable to always choose a single edge to corrupt. \( \square \)

**Theorem 4.1.** Algorithm 2 picks the set of corrupted edges optimally.

**Proof.** Suppose, for the sake of contradiction, that there is a set of corrupted edges that is more optimal.

Each edge in this optimal set that differs from the algorithm’s chosen set is one of two cases: either it is not on the two paths the adversary aims to swap, or it is. If the edge is not on the two paths, it does not affect the budget because when corrupted, it does not contribute
at all to switching those two paths. Otherwise, a differing edge indicates that there must be some corruption replaced in the optimal set as the adversary only corrupts one edge per path traversed by Lemma 1. However, we now prove that this substitution of edges will not yield a greater corruption. Fundamentally, this is because the algorithm chooses the edge on the least number of paths, which is optimal.

If the adversary corrupts an edge that is on \( h \) paths, including either the optimal or other chosen path but not both, it will corrupt this edge \( p \) fraction of the \( N \) times it is traversed for a total corruption of \( p \cdot N \cdot \delta \), and this perturbs the reward of the target path by \( \frac{p \cdot N \cdot \delta}{N} = \frac{N \cdot \delta}{N} \).

This means that edges that are on a smaller number of paths will contribute a greater corruption to the budget. As our algorithm picks the edge that is on the least number of paths, it chooses the edge with the maximum corruption.

Thus, our algorithm is optimal, in that it calculates budget optimally and chooses to swap the optimal path with the path with lowest possible reward. \( \square \)

4.2 Free Corruption for Larger \( N \)

While free corruption is effective for small \( N \) in the warm start, it is not stable once the victim changes its sampling strategy for larger \( N \). After the warm start, the victim follows an \( \epsilon \)-greedy sampling strategy. Let \( P_1 \) be the path returned by the victim at the end of warm start. The victim samples \( P_1 \) a fraction of \( 1 - \epsilon \) times, compared to the \( \frac{\epsilon}{|S| - 1} \) fraction for which all other paths are sampled. Since paths other than \( P_1 \) are traversed less often, the adversary is unable to corrupt the edge set \( E_s \) as frequently as before. As the fraction of times the adversary corrupts \( E_s \) decreases, the free corruption decreases as well. Therefore, if the adversary maintains a fixed strategy, \( P_1 \) will decrease in perceived reward. For a sufficiently large number of samples \( N \), \( P_1 \)'s reward may decrease enough so that the victim ultimately chooses a different path where \( R(\hat{P}) > R(P_1) \).

5 Stable Paths

Free corruption is only effective for small \( N - n \) as it degrades when the number of samples increases. Therefore, when \( N - n \) is large, the adversary uses a stable path to effectively extend the warm start period and reduce the number of samples where \( \hat{P} \) is greedily sampled. The adversary uses a stable path as an intermediate step before switching to \( \hat{P} \).

Definition 5.1 (Stable Path). A stable path \( P_b \)'s reward must satisfy \( R_f(\hat{P}) - p\delta < R(P_b) \) in order for the victim to perceive it as the optimal path for large \( N \).

Intuitively, \( R(P_b) \) is at most \( p\delta \) away from the path with highest reward, so the adversary doesn't need any free corruption to make the victim believe \( P_b \) has the highest reward instead. Thus, regardless of the victim’s sampling strategy, the adversary will be able to corrupt \( P_b \) so that the victim chooses it. The adversary would corrupt \( \epsilon \) \( P_b \) upwards during the warm start so the victim perceives \( P_b \) to be optimal. After the warm start, the adversary continues corrupting \( P_b \) up by \( p\delta \) until a path switch is needed.
Figure 2: $n = 100$; Graph of free corruption from section 4 with respect to number of samples after warm start. Free corruption is \[ \epsilon \left( N - n \right) + 100 \left( 1 - \frac{1}{2} \left( 1 - \epsilon \right) \right) \left( N - n \right) \]. For $\epsilon = 1, \epsilon = 0.5, \epsilon = 0.25, \epsilon = 0.1, \epsilon = 0.05, \epsilon = 0.001$.

(a) Without adversarial corruption, $AJ$ is optimal.
(b) Rewards right after warm start for a stalling strategy. Stable path $AE$ is perceived as optimal.
(c) Rewards some episodes after warm start. Corruption degrades on stable path $AE$ so that path $AG$ is now perceived as optimal.

Figure 3: $p\delta = 4$

5.1 Adversarial Strategy using Stable Paths

We provide an adversarial heuristic which we call a *stalling strategy* where the adversary doesn’t switch to $\hat{P}$ immediately. In this strategy, the adversary chooses to stall with a stable path $P_b$. After the warm start, the adversary corrupts $P_b$ so the victim always views it as optimal, either corrupting $e_{P_b, \hat{P}}$ or $e_{\hat{P}, P_b}$. Near the end of learning, it corrupts $e_{P_b}$ downwards to switch to $\hat{P}$.

In fig. 3, it’s possible to switch to $AG$, or first to $AF$ and then to $AG$, during the warm start. However, corruption would degrade on both paths, so it’s necessary to first trick the victim into greedily sampling a stable path that can easily be switched with $AG$. 

---

For the mathematical expression, the free corruption is given by $\epsilon \left( N - n \right) + 100 \left( 1 - \frac{1}{2} \left( 1 - \epsilon \right) \right) \left( N - n \right)$. This expression quantifies the amount of free corruption based on the number of samples $n$ and the corruption factor $\epsilon$. The graph in Figure 2 illustrates how this free corruption evolves over the number of samples after the warm start. The graph shows different lines for various values of $\epsilon$, ranging from $0.001$ to $1.0$, demonstrating the impact of different corruption levels on the graph of free corruption.
Algorithm 2: Adversarial Attack Against $\epsilon$-Greedy Victim

**Algorithm 2 Adversarial Attack Against $\epsilon$-Greedy Victim**

**Input:** $\epsilon$, MDP $M$

1. $P \leftarrow P^*$  \hspace{1cm} $\triangleright$ The final path we switch to
2. $D, U \leftarrow \emptyset$  \hspace{1cm} $\triangleright$ Initialize final edge sets corresponding to corruptions down/up
3. **for all** $\hat{P} \in M \land \hat{P} \neq P^*$ **do**  \hspace{1cm} $\triangleright$ Choosing $P$
4. $\text{hasDown} \leftarrow \text{true}$  \hspace{1cm} $\triangleright$ Some paths need to be corrupted down for $P$ to be reachable
5. $D_{\hat{P}}, U_{\hat{P}} \leftarrow \emptyset$  \hspace{1cm} $\triangleright$ Initialize edge sets corresponding to $\hat{P}$
6. **if** $\text{warm}(\hat{P}) + \frac{(1-\epsilon)p_\delta}{1-(1-\alpha)\epsilon} < R(P^*) - R(\hat{P})$ **then**  \hspace{1cm} $\triangleright$ Budget needed below upper bound
7. **continue**
8. **end if**
9. $U_{\hat{P}} \leftarrow \text{GetUpwardsCorruptions}(M, \hat{P}, U_{\hat{P}}, D_{\hat{P}}, n, N, \epsilon)$  \hspace{1cm} $\triangleright$ Corruptions on low-reward paths
10. $D_{\hat{P}} \leftarrow \text{GetDownwardsCorruptions}(M, \hat{P}, D_{\hat{P}})$  \hspace{1cm} $\triangleright$ Corruptions on high-reward paths
11. $\text{hasDown} \leftarrow \text{Reachability}(M, \hat{P}, D_{\hat{P}})$  \hspace{1cm} $\triangleright$ Check viability of assigned corruptions
12. **if** $\text{hasDown} \land R(\hat{P}) < R(P)$ **then**
13. $P \leftarrow \hat{P}, D \leftarrow D_{\hat{P}}, U \leftarrow U_{\hat{P}}$
14. **end if**
15. **end for**
16. **return:** $U, D$

Algorithm 2 performs better than Algorithm 1 in several key cases, illustrated in the next section. Currently, we are investigating a proof of optimality.

5.2 Stalling With Multiple Stable Paths

Let’s define a **phase** as a period of learning where the victim views the same path as optimal. Adversarial corruption for an $\epsilon$-greedy victim can be modeled as:

$$\sum_{e \in P} (\text{warm}(e) - \sum_{j=1}^{i} N_j G_{e,j}(1 - \epsilon)(\text{warm}(e) - p\delta C_{e,j})) \sum_{j=0}^{i} (G_{e,j}(1 - \frac{\epsilon}{\sum_{e \in P} A_{e,P}})) + \sum_{j=0}^{i} (\frac{(1-G_{e,j})}{\sum_{e \in P} A_{e,P}})$$

where: $i$ is the total number of phases, $N_j$ is the number of samples in the $j$th phase, $\text{warm}(e)$ is the optimal amount of corruption on edge $e$ in the warm start period,

$$A_{e,P} = \begin{cases} 1 & e \in P \\ 0 & e \notin P \end{cases}$$

$$C_{e,j} = \begin{cases} 1 & e \text{ corrupted upwards in phase } j \\ 0 & \text{else} \end{cases}$$

$$G_{e,j} = \begin{cases} 1 & e \text{ in } j\text{th greedily chosen path} \\ 0 & \text{else} \end{cases}$$
This quantity increases when the number of phases increases, indicating that a stalling strategy utilizing multiple stable paths may be optimal. This is a direction we hope to explore in the future.

6 Interfering Paths

While free corruption allows the adversary to switch to paths with lower reward, this technique comes with several negative implications. In small MDPs, the adversary is able to successfully switch the optimal path $P^*$ with a low reward path $\hat{P}$. In section 4 and fig. 3, the rewards of the paths other than the optimal path are sufficiently low so they do not affect the switching of $P^*$ with $\hat{P}$. However, in most cases like fig. 4, there will be paths $P_{in}$ with reward $R(P_{in}) \geq R_f(\hat{P})$ which we call interfering paths. When such a path $P_{in}$ exists, the victim will ultimately choose this interfering path rather than the desired path $\hat{P}$.

**Definition 6.1** (Interfering Path). A path $P_{in}$ is an interfering path if $R_f(P_{in}) > R_f(\hat{P})$.

![Figure 4:](image)

(a) Without adversarial corruption. (b) Interfering path $AH$ is chosen by victim. (c) Adversary tricks victim into choosing $AI$.

For instance, in fig. 4, if the adversary used Algorithm 2, it would attempt to switch $P^*$ with $AI$. However, after corrupting these paths the reward of $AH$ is greater than that of $AI$, so the victim would choose $AH$.

Nevertheless, the existence of such interfering paths does not mean that $AI$ is unreachable. Instead, the adversary should corrupt $AH$ downwards on its disjoint edge and corrupt all other edges the same as before. With this strategy, $AI$ has a higher perceived reward.

We can modify Algorithm 1 to better account for interfering paths. This heuristic is not optimal, but preliminary empirical testing show that it improves upon the baseline Algorithm 1. We plan on further refining and testing this heuristic.
Algorithm 3 Heuristic for Interfering Paths

input: MDP $\mathcal{M}$

1: $\hat{P}, f \leftarrow \text{Algorithm 2}$  \hspace{1em} \triangleright Adversary corrupts $\hat{P}$ and $P^*$ optimally
2: $R \leftarrow (\hat{P}, f)$
3: for all $P_i \in \mathcal{M} \land P_i \neq P^* \land R(P_i) > R$ do  \hspace{1em} \triangleright Iterate through interfering paths
4: \hspace{1em} $f_{P_i} \leftarrow p\delta$  \hspace{1em} \triangleright Default corruption is $p\delta$
5: \hspace{1em} for all $P_s \notin \{P_i, P^*, \hat{P}\}$ do  \hspace{1em} \triangleright Iterate other paths to calculate free corruption
6: \hspace{1em} \hspace{1em} $f_{P_s} \leftarrow 0$
7: \hspace{1em} \hspace{1em} if $f_{P_i} + R(P_i) > R \lor f(P_s) \neq 0$ then  \hspace{1em} \triangleright if sufficient corruption or path used
8: \hspace{1em} \hspace{1em} break
9: \hspace{1em} end if
10: \hspace{1em} for all $e \in P_s$ do  \hspace{1em} \triangleright Iterate to find the edge with greatest free corruption
11: \hspace{1em} \hspace{1em} if $(e \in P_i \land e \notin \hat{P}) \land \frac{1}{a_e} > f_{P_s}$ then  \hspace{1em} \triangleright $e$ should be corrupted downwards
12: \hspace{1em} \hspace{1em} $f_{P_s} \leftarrow \frac{p\delta}{a_e}$  \hspace{1em} \triangleright Calculate free corruption from corrupting $e$;
13: \hspace{1em} \hspace{1em} $a_e := |\{P : e \in P\}|$
14: \hspace{1em} end if
15: \hspace{1em} end for
16: end for
17: output $P, f$

7 Experiment

7.1 Baseline Attacks

Aside from the adaptive attack, as a baseline, this work considers two primitive attacks, as a means of comparing them to more sophisticated attacks.

**Primitive Attack A.** Under the first primitive attack, which we call $A$, when the victim traverses path $P_i$ of length $n_i$, the rewards are perturbed by a normal distribution centered at 0 and truncated between $-\frac{\delta}{n_i}$ and $\frac{\delta}{n_i}$ (the division is necessary to ensure that the total perturbation in the sample does not exceed $\delta$ as mandated by our problem setting). The probability density function of this distribution for a mean 0, standard deviation 3, and bounded between $-1$ and 1 looks like
Essentially, the idea behind this attack is to introduce noise into the observed reward of the MDP. It will be empirically demonstrated that in general, this methodology is far from the adversary's optimal strategy.

**Primitive Attack B.** Under the second primitive attack, which we call $B$, if the reward is drawn from the optimal path $P^*$ of length $n^*$, then the adversary perturbs it *down* by $\frac{\delta}{n^*}$; conversely, if the reward is drawn from a suboptimal path $P_i$, then the adversary perturbs it *up* by $\frac{\delta}{n_i}$. This strategy is not optimal for the adversary because it blindly assigns equal perturbation to all edges in a path, rather than singling out the best edge to perturb and targeting it. It is shown below (and confirmed empirically) that for a random victim with a warm start phase, neither strategies $A$ nor $B$ are optimal.

### 7.2 Preliminary Results

The following experiment$^2$ was conducted in this MDP:

It included three paradigms: whether the adversary was absent, baseline ($A$ & $B$), or dynamic (Algorithm 1); whether the victim was greedy or random during the main learning phase; and whether there was a warm start. Performance was measured starting in the post-warm-start phase, testing the learned Q-values with $\epsilon = 0$ after every episode and $p\delta = 1^3$.


$^3$For the purposes of these experiments, we remove the assumption that the gap is sufficiently large to prevent a path from remaining in between the optimal path and the path being switched. The code implementation of Algorithm 1 is modified to account for such cases.
Clearly, the baseline attacks were hardly effective at tampering with the learning of the victim, because when there was no warm start, the victim’s learning curve was still able to learn to achieve the optimal reward of 7. In contrast, the more sophisticated attack was successful in meaningfully perturbing the victim, despite being subject to the same $p\delta = 1$ constraint as the baseline adversaries. Moreover, it is apparent that when there is no warm start under the dynamic adversary\textsuperscript{4}, the $\epsilon$-greedy victim maintains higher performance than its random counterpart.

\textsuperscript{4}One noteworthy feature of this graph is the sharp spike in performance in the beginning, if there is no warm start. This spike can be attributed to the worst path in the MDP, of reward 3. When the victim quickly learns to avoid this path, it shifts its strategy from sampling it $\frac{1}{4}$ of the time to sampling it much less frequently, owing to a large short-term increase in the victim’s reward.
7.3 Random MDP Generation and General Analysis of their Structure

This work considers the following algorithm, which generates a random MDP given a set of parameters. Such a procedure is quite useful in evaluating the efficacy of MDP algorithms in general rather than focusing on one MDP, as before. Note that, because the class of MDPs which this work considers is structurally equivalent to an acyclic directed graph, we can guarantee acyclicity by assigning an integer to each state and only allowing transition between a lower-numbered state to a higher-numbered one:

Algorithm 4 Random MDP Generation

input: $n_s, p_e$  \(\triangleright\) input number of states in MDP and likelihood of edge existing between two given states

1. $S \leftarrow \{0, 1, \ldots, n_s\}$
2. while True do  \(\triangleright\) Keep on iterating until fully connected MDP is attained
3. $A \leftarrow \{\}$
4. for all $s_1$ in $S$ do
5. $a \leftarrow 0$ \(\triangleright\) Keep track of amount of actions in $s_1$
6. for all $s_2 \in S$ such that $s_2 > s_1$ do  \(\triangleright\) Iterate through pairs of states
7. if random(0, 1) $< p_e$ then  \(\triangleright\) Do with probability $p_e$
8. $R(s_1, s_2) \leftarrow \text{random}(-1, 1)$ \(\triangleright\) Expected edge reward is 0, so that longer paths do not have higher expected rewards
9. $P(s_1, a, s_2) \leftarrow 1$  \(\triangleright\) Draw edge between $s_1$ and $s_2$
10. $a \leftarrow a + 1$ \(\triangleright\) There is now one more possible action in $s_1$
11. end if
12. end for
13. if $a > |\{A\}|$ then  \(\triangleright\) We use $a$ to ensure that our action space contains the right number of actions
14. $A \leftarrow \{0, 1, \ldots, a\}$
15. end if
16. end for
17. if $\forall s \in S: (\exists P : s \in P) \land |\{P : P \in M\}| > 1$ then  \(\triangleright\) Verify that MDP is fully connected and not trivial; otherwise, reject and start over
18. return $M(S, A, R, P)$
19. end if
20. end while

First, we use this generation algorithm to verify that in the general case, for a victim with a random warm start phase, Algorithm 1 achieves the greatest difference between the optimal reward and the victim’s returned reward:

\[5\] We ignore trivial MDPs in which the gap between the optimal path and all other paths is so large that the adversary cannot corrupt the victim into switching to any path at all. Thus, we also ignore MDPs that have only one path.
Attack A shows effectively no difference at all between optimal reward and returned reward. This is because Attack A randomly assigns noise to the observations of MDPs, so in the general case, averaged over many iterations, it imposes no meaningful corruption.

However, the fact that Algorithm 1 experiences a general trend of decreasing the difference between optimal and returned reward is of great interest. It demonstrates that for denser and denser MDPs (that is, MDPs with more chances at free corruption), the ability to maintain warm-start corruption wanes, as evidenced by the returned reward becoming closer to the optimal reward. This is due to the fact that when the victim enters the $\epsilon$-greedy phase, it focuses more on a single path to exploit, so as it traverses other paths less and less, less free corruption may be derived from them. Thus, Algorithm 1’s overall ability to maintain corruption through an $\epsilon$-greedy phase wanes for denser MDPs. Additionally, it is clear that Algorithm 2 is much more optimal than either primitive baseline attacks A or B, because it still manages to sustain a higher discrepancy between optimal reward and victim-returned reward than either baseline.

Next, we establish a relationship between the depth of an MDP (which we compute as the average number of states in each path) and its density (which we measure by $p_e$):
Indeed, denser MDPs are likely to have longer (deeper) paths. This is because a denser MDP has more edges, and is therefore more likely to possess longer, consecutive chains of edges; i.e. longer paths.

Moreover, below is a graph detailing the number of greedy episodes for which warm start corruption can be maintained. Note that for some randomly generated MDPs, the warm start corruption can be maintained perpetually since it is sufficiently unreliant on free corruption. In these cases, we manually set the number of episodes in which corruption is perpetually maintained to be a very high number (1000), since the mathematically accurate data point would be infinite.

Evidently, for denser MDPs, the number of episodes for which warm start corruption can be maintained drops. This reinforces the notion from previous data and reasoning that denser MDPs have more free corruption and therefore struggle to hold onto all of it against an $\epsilon$-greedy victim.
Below, the “amount” of free corruption allowed to the adversary is parameterized along the x-axis. Specifically, when the adversary is determining the path to perturb in Algorithm 1, with a probability \( p_f \), it is allowed to take into consideration budget present on paths that are neither the optimal one nor the one being switched. In this way, \( p_f = 1 \) corresponds to the usual Algorithm 1, whereas \( p_f = 0 \) corresponds to a situation in which the adversary may only perturb edges on the optimal path and the path being switched. The number of episodes for which warm start corruption is preserved in a subsequent \( \epsilon \)-greedy stage is depicted below:

![Graph](image)

In short, in an intuitive sense, the more the adversary is permitted to make use of free corruption, the less easily it can hold on to corruption. This, as before, is because free corruption relies on the adversary consistently sampling a wide variety of paths. But under \( \epsilon \)-greedy behavior, it focuses on just one path, losing all the free corruption, so the adversary’s strategy falls apart.

8 Conclusion

8.1 Future Work

The main task ahead is to generalize the victim sampling strategy from \( \epsilon \)-greedy to an arbitrary strategy. We aim to devise a method of constructing an optimal adversarial attack against any sampling strategy. Our current work in this direction includes modeling the sampling and attack processes as a matrix multiplication. We are solving the optimization problem to find a fixed adversarial strategy.

Another line of work is to further investigate the optimality of our stalling heuristic and to test it empirically. Moreover, we aim to improve upon this algorithm to balance correctness with computational complexity.

One approach to achieve complete optimality may be establishing a recursive strategy to determine the sequence of path switches. With this sequence, the adversary could construct edge sets to corrupt up and down accordingly. However, the computational complexity
required for the adversary to deploy this strategy is most likely exponential. Thus, a future direction may be approximating this recursive strategy to make it more practical.

Moreover, on the experimental side, it would be interesting to gain even more insight about how the general structure of an MDP reacts with the strategy of the adversary (e.g. how much it relies on free corruption) to affect the performance of the victim and adversary. For instance, when fixing the number of samples during which the adversary can maintain warm start corruption, what is the relationship between the density of the MDP and the distance between optimal reward and post-greedy-phase returned reward?

8.2 Acknowledgements

We would like to especially thank Mayuri Sridhar for her continued assistance in our project, guiding us at every turn and never failing to provide us with helpful suggestions. We would also like to thank MIT PRIMES for this amazing opportunity to conduct research, as well as Dr. Slava Gerovitch, Dr. Tanya Khovanova, Dr. Srini Devadas, Dr. Pavel Etingof, and Mr. Andre Dixon, for organizing this unique program.

References

9 Appendix

Helper Functions for Algorithm 2

**Algorithm 5 GetUpwardsCorruptions**: Find which edges to corrupt upwards to stall the victim and gain greater corruption

**input**: MDP $\mathcal{M}$, optimal path $P^*$; the path we want to switch to $\hat{P}$; $U$ and $D$, edge sets to corrupt up/down; $n$ number of warm start samples; $N$ number of samples after warm start; $\epsilon$

1: $R_f(\hat{P}) ← R(P^*) - \text{warm}(P^*)$ \hspace{1cm} $\triangleright$ Calculate final reward of $\hat{P}$
2: if $\text{warm}(\hat{P}) < R(P^*) - R(\hat{P})$ then \hspace{1cm} $\triangleright$ Find path which edge with $\hat{P}$
3: Let $e \in \hat{P}$ satisfy $e = \arg \max_e \alpha_e$ \hspace{1cm} $\triangleright$ on more paths, bigger difference during greedy
4: for all $P_e$ do \hspace{1cm} $\triangleright$ paths containing $e$
5: if $R(P_e) + p\delta > R_f(\hat{P})$ then
6: Solve for $N_{\hat{P}}$ with $(1-\epsilon)(\text{warm}(P_e)-p\delta)(N_{\hat{P}}) = R(P_e) - R_f(P)$
7: $U ← U + (e, P_e, N - N_{\hat{P}})$ \hspace{1cm} $\triangleright$ Start corrupting at sample $N - N_{\hat{P}}$
8: Find $P_{\text{stall}}$ where $P_{\text{stall}} = \arg \max P \ R(P)$ and $R(P_{\text{stall}}) < R_f(\hat{P})$
9: $U ← U + (e_d, P_{\text{stall}}, 0)$ \hspace{1cm} $\triangleright$ Corrupt $P_{\text{stall}}$ upwards to stall victim
10: if $R(P_e) + \epsilon N_{\hat{P}} > R_f(\hat{P})$ then
11: $U ← U + (e, P_e)$ \hspace{1cm} $\triangleright$ Corrupt $P_e$’s common edge up
12: $N_{P_e} ← (R(P_e) - R(\hat{P}))(1-\epsilon)N + n$ \hspace{1cm} $\triangleright$ Samples to switch $P_e$ and $\hat{P}$
13: $D ← D + (e, P_e, N - N_{P_e})$ \hspace{1cm} $\triangleright$ Corrupt disjoint edge down to switch to $\hat{P}$
14: break \hspace{1cm} $\triangleright$ Prefer initializing $U$ and $D$ here, there’s no degradation estimate
15: end if
16: end if
17: end for
18: end if
19: if $\text{warm}(\hat{P}) > R(P^*) - R(\hat{P})$ then \hspace{1cm} $\triangleright$ Determine a path to stall with
20: Find $P_{\text{stall}}$ where $P_{\text{stall}} = \arg \max P \ R(P)$ and $R(P_{\text{stall}}) < R_f(\hat{P})$
21: $U ← U + (e_d, P_{\text{stall}}, 0)$ \hspace{1cm} $\triangleright$ Corrupt $P_{\text{stall}}$ upwards to stall victim
22: $N_{P_e} ← \frac{(R(P_e) - R(P_{\text{stall}}))(1-\epsilon)N + n}{2p\delta} - n$ \hspace{1cm} $\triangleright$ Samples to switch $P_{\text{stall}}$ and $\hat{P}$
23: $D ← D + (e, P_{\text{stall}}, N_{P_{\text{stall}}})$ \hspace{1cm} $\triangleright$ Corrupt disjoint edge down to switch to $\hat{P}$
24: end if
25: return: $U$, $D$
Algorithm 6 GetDownwardsCorruptions: Find which edges to corrupt downwards to ensure the victim doesn’t switch to an unintended high-reward path

**input:** MDP $\mathcal{M}$, optimal path $P^*$; the path we want to switch to $\hat{P}$; $D$, the edge set to corrupt down

1: $R_f(\hat{P}) \leftarrow R(P^*) - \text{warm}(P^*)$  \hspace{1cm} \triangleright \text{Calculate final reward of } \hat{P}
2: for all $\hat{P} \in \mathcal{M} \land R(\hat{P}) > R_f(\hat{P})$ do  \hspace{1cm} \triangleright \text{Track edges that need to be corrupted}
3: $D \leftarrow D + (e_d, \hat{P})$  \hspace{1cm} \triangleright \text{add the disjoint edge to the set}
4: end for
5: $e^* \leftarrow \emptyset, b_e \leftarrow 0$  \hspace{1cm} \triangleright \text{Edge to perturb and corresponding value}
6: for all $\hat{P} \in \mathcal{M} \land R(\hat{P}) < R_f(\hat{P}) \land \notin P_p$ do  \hspace{1cm} \triangleright \text{We corrupt remaining paths as needed}
7: \hspace{1cm} for all $\hat{P} \in \mathcal{M} \land R(\hat{P}) > R_f(\hat{P})$ do
8: \hspace{2cm} $e \leftarrow \hat{P} \cap \hat{P}$
9: \hspace{2cm} if $\frac{1}{a_e} > b_e$ then  \hspace{1cm} \triangleright \text{Updating best edge to perturb}
10: \hspace{3cm} $e^* \leftarrow e, b_e \leftarrow \frac{1}{a_e}$
11: \hspace{2cm} end if
12: \hspace{1cm} end for
13: $D \leftarrow D + (e^*, \hat{P})$
14: end for
15: return: $D$

Algorithm 7 Reachability: Determine if the assigned corruptions will ensure that the victim doesn’t switch to an unintended high-reward path

**input:** MDP $\mathcal{M}$, optimal path $P^*$; the path we want to switch to $\hat{P}$; $D$, the edge set to corrupt down

1: $R_f(\hat{P}) \leftarrow R(P^*) - \text{warm}(P^*)$  \hspace{1cm} \triangleright \text{Calculate final reward of } \hat{P}
2: hasDown $\leftarrow \text{True}$
3: for all $\hat{P} \in \mathcal{M} \land R(\hat{P}) > R_f(\hat{P})$ do
4: \hspace{1cm} $b \leftarrow 0$ \hspace{1cm} \triangleright \text{Initialize budget}
5: \hspace{1cm} for all $e \in D \land e \in \hat{P}$ do
6: \hspace{2cm} $b \leftarrow b + \frac{1}{a_e}$ \hspace{1cm} \triangleright \text{Update budget}
7: \hspace{1cm} end for
8: \hspace{1cm} if $b < R(\hat{P}) - R_f(P)$ then  \hspace{1cm} \triangleright \text{Ensure sufficient budget to corrupt } \hat{P} \text{ downwards}
9: \hspace{2cm} hasDown $\leftarrow \text{False}$
10: \hspace{1cm} end if
11: \hspace{1cm} end for
12: return: hasDown