# Adversarial Attacks Against Online Reinforcement Learning Agents in MDPs

Alicia Li[2,3]
ilfangliu@gmail.com

Matan Yablon[2,4]
matiyablon@gmail.com

Mentor: Mayuri Sridhar[1]

[1]Massachusetts Institute of Technology
[2]MIT PRIMES
[3]Lexington High School
[4]Sharon High School

## Abstract

Online Reinforcement Learning (RL) is a fast-growing branch of machine learning with increasingly important applications. Moreover, making RL algorithms *robust* against perturbations is essential to their utility in the real world. Adversarial RL, in which an attacker attempts to degrade an RL agent's performance by perturbing the environment, can be used to understand how to robustify RL systems. In this work, we connect an adversarial attack model to streaming algorithms: the victim samples paths based on its interactions with the environment, while the adversary corrupts this stream of data. We construct an attack algorithm in Markov Decision Processes (MDPs) for a random-sampling victim and prove its optimality, in addition to investigating an adversarial strategy against an epsilon-greedy victim with a warm start period. In the epsilon-greedy setting, we bound adversarial corruption and analyze how to exploit this highly adaptive model to improve upon warm start budget. Experimentally, we show that our algorithm outperforms baseline attacks, and we generate random MDPs to characterize how their general-case structure affects the adversary's ability to maintain its warm start corruption.

## 1 Introduction

### 1.1 Background

The use of Reinforcement Learning (RL) to solve many classes of problems has skyrocketed over the last decade and has achieved stellar results in several tasks and fields [9, 14, 16]. RL is a branch of Machine 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 RL 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 [9].

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 RL 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 [7, 19], the victim's observations [6], the victim's memory, and the environment transition dynamics [18, 11].

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 RL sampling strategies due to its simplicity and empirical success for many practical problems [2]. 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 MDP environments, drive the motivation behind this paper.

Simultaneously, we fill a gap in the field of online learning, where an agent has access to a stream of data that it uses to make decisions. We frame an adversarial attack as a streaming algorithm in which the victim learns a behavior based on the stream of its previous samples, while the adversary needs to select when and where 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

The area of Reinforcement Learning we focus on is Online RL, in which 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 [10, 5].

**Online RL Algorithms**

*Q-Learning* is one of the most popular Online RL algorithms and the algorithm we implemented empirically to test our attacks on. In this method, each pair of states and actions is given a *Q-value* which prescribes which action to take next [15]. These Q-values are updated according to the data stream generated as the agent navigates its environment.

Another well-known Online RL algorithm is *PPO*, which computes an update at each step to minimize a cost function while maintaining proximity to the previous policy. It does so by optimizing a clipped surrogate objective function using stochastic gradient descent [13].

**Attacks on Streaming Algorithms.** Previous works on attacking streaming algorithms include Gong et al. [5] and Natali et al. [10]. The former of which uses partially observable decision process concepts from RL 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 [20]. 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. [19] 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. [18], Zhang et al. [19] 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 $O(e^{|\mathcal{S}|})$, 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 action[1] $\epsilon$ fraction of the time, and sampling the highest-Q-value action $1-\epsilon$ fraction of the time [15]. That is,

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

Previous works about $\epsilon$-greedy's theoretical guarantees include Dann et al. [2]. 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 [15].** 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 (explorative) 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 [15], as per the equation

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

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 [15].

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 [2]

[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.

## 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 [4]. 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. [12], 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.

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 [3].

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 [4]. The Robust RL objective is to find a policy $\pi$ that satisfies:

$$\max_{\pi} \min_{p} \mathbb{E}_{\pi,p}\Big[\sum_{t} R_t\Big],$$

where $p$ is the environment and $R_t$ is the reward at time $t$.

There are several methods that satisfy the Robust RL objective [17, 8, 1] that all incorporate the maximin framework above in unique ways. For instance, Tessler, Efroni, and Mannor [17] 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. Which kinds of MDPs are robust to adversarial perturbations? That is, how does the structure of an MDP affect the performance of an adversarial strategies?

# 3 Model

## 3.1 MDPs and Q-Learning

The environment in which the RL agent operates is classically conceived of as a Markov Decision Process (MDP).

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

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

In many implementations of Reinforcement Learning, each state-action pair is assigned a *Q-value*, and these values dictate which action the agent takes. In addition to these components, we also consider a learning rate $\alpha$, and a discount factor $\gamma \in [0, 1)$. The learning rate $\alpha$, present in many other branches of machine learning as well, describes the degree to which we change the Q-values every iteration; it represents the algorithm's overall resistance to change. Also, the discount rate weights future reward relative to immediate reward [15]. Its implementation is necessary to avoid the explosion of $Q$-values during $Q$-learning, a fundamental RL learning algorithm used in a variety of applications:

**Algorithm 1** Q-learning [15]

**input:** $\alpha, \epsilon, \gamma,$ MDP $\mathcal{M},$ initialized $Q,$ initial state $s_0$

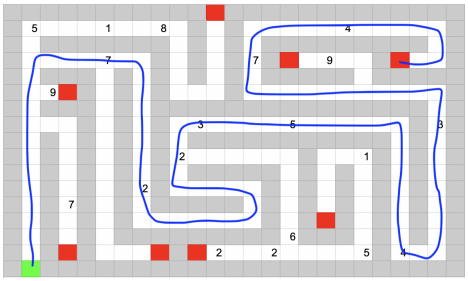
**for** episode = 1 to $N$ **do**
  $s_t \leftarrow s_0$
  **while** $s_t$ is not terminal **do**
    $a_t \leftarrow \text{sample}(\pi(\cdot|s_t))$
    $s_{t+1} \leftarrow \text{sample}(p(s_t, a_t))$
    $r_t \leftarrow R(s_{t+1})$
    $Q(s_t, a_t) \leftarrow (1 - \alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}))$
    $s_t \leftarrow s_{t+1}$
  **end while**
**end for**

Moreover, we consider an adversary that may perturb a sample with probability $p$, and if it happens that it may do so, the adversary's total reward perturbation may not exceed $\delta$.

## 3.2 Maze MDP

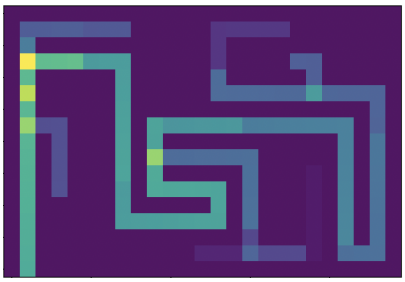
Although our theoretical and general results apply to a more general class of MDPs, a simpler and more concrete way of conceiving of an MDP is a maze challenge situated in a grid. This setting can be easily visualized and also can represent a generalized MDP with each open

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

In order to ultimately create a victim defense, we consider the adversary's strategy against an online agent which follows the $\epsilon$-greedy sampling strategy. We do so in the context of **path switches**, as the adversary's goal is to make the reward of a "bad" path appear higher to the victim than the reward of a "good" path. We are interested in the adversary's *budget*, how much corruption it has to switch two paths, as we can use this to determine if switching the paths is possible and after how many samples this will occur. First, we analyze the probability that paths switch in the context of two paths that are sampled equally and provide a bound for regret, the difference between the reward of the victim's chosen actions and the reward of the optimal action in hindsight.

## 4.1 Preliminaries

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.

For an MDP with two paths $P_i$ and $P_j$, we denote the path lengths as $n_i$ and $n_j$ respectively. We denote the actual rewards as $R(P_i)$ and $R(P_j)$ where $R(P_i) < R(P_j)$ and the observed (and possibly perturbed) rewards as $r_i'$ and $r_j'$.

## 4.2 Disjoint Path MDP

**Theorem 4.1.** *The regret of a completely greedy victim in the two-path adversarial setting is upper-bounded by*

$$\frac{(n_i + n_j)p\delta^2(1-p)}{gap}$$

*where* $gap = R(P_j) - R(P_i)$.

**Proof Sketch.** We first calculate the expected rewards and variances of each path, and then use Chebyshev's Inequality to bound the probability that the paths switch. Using this probability, we calculate regret.

*Proof.* By linearity of expectation, $\mathbb{E}[r_i'] = R(P_i) - n_a \cdot p \cdot \delta$ and $\mathbb{E}[r_j'] = R(P_j) + n_j \cdot p \cdot \delta$.

We can see that $P(r_i' \geq r_j') < P(|r_i' - E[r_i']| + |r_j' - E[r_j']| \geq \text{gap})$.

By Chebyshev's Inequality, $P(|r_i' - E[r_i']| + |r_j' - E[r_j']| \geq \text{gap}) \leq \frac{Var(r_i' + r_j')}{\text{gap}^2}$.

We can determine that $Var(r_i' + r_j') = (n_i + n_j)p\delta^2(1-p)$ to simplify to

$$P(|r_i' - E[r_i']| + |r_j' - E[r_j']| \geq \text{gap}) \leq \frac{(n_i + n_j)p\delta^2(1-p)}{\text{gap}^2}.$$

Thus, the probability that paths switch is bounded as $P(r_i' \geq r_j') < \frac{(n_i+n_j)p\delta^2(1-p)}{\text{gap}^2}$.

The expected regret of a completely greedy strategy is

$$P(r_i' \geq r_j')(r_j - r_i) \leq \frac{Var(r_i' + r_j')}{\text{gap}^2} \cdot gap.$$

Thus, regret of greedy strategy has an upper bound of

$$\frac{(n_i + n_j)p\delta^2(1-p)}{gap}.$$

□

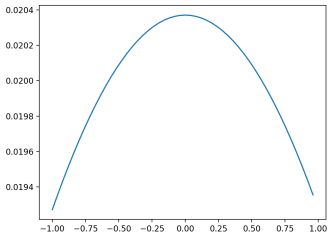
# 5 Random-Sampling Victim

In this section, we consider a victim following a random-sampling strategy, which samples each path with equal probability. Intuitively, one might think to corrupt in the same manner as the previous section, perturbing just the disjoint edges, and thus achieving a similar budget calculation. However, our method improves on this budget by exploiting the shared edges of target paths. In addition to the corruption from the disjoint edges, when the victim traverses other paths, the adversary is able to gain *free corruption* by perturbing the edges those paths have in common with the two target paths. This corrupts the edge's perceived value, thereby helping to switch the two paths and increasing the budget. Thus, the adversary may be able to trick the victim into choosing a worse path when paths are non-disjoint.

## 5.1 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.

## 5.2 Adversarial Algorithm for Random-Sampling Victim

Our algorithm outputs the path with lowest possible reward that the adversary may switch with the optimal path, assuming the victim samples all paths with equal probability. It calculates the adversary's budget for each path; then, it returns the path that minimizes reward while still having sufficient budget to switch with the optimal path. Note that we assume that the gap between the optimal reward and the second-best reward is large enough so that when the switch takes place, there is no path in between that remains higher than the path we are trying to switch.

**Algorithm 2** Adversarial Attack: Switching Non-disjoint Paths in Warm Start

**input:** MDP $\mathcal{M}$

$P \leftarrow P^*$ ▷ Current path to perturb; we will iterate and find a better one
**for all** $P_i \in \mathcal{M} \wedge P_i \neq P^*$ **do** ▷ Iterate only through suboptimal paths
  $b_i \leftarrow 2$ ▷ Corruption from $P_i$ and $P^*$ disjoint edges
  **for all** $P_j \notin \{P_i, P^*\}$ **do** ▷ Iterate through paths that are not optimal or $P_i$
    $a^* \leftarrow \infty$
    **for all** $e \in P_j$ **do** ▷ Iterate through edges
      **if** $(e \in P^* \oplus e \in P_i) \wedge a(e) < a^*$ **then**
        $a^* \leftarrow a(e)$ ▷ Calculate budget; $a(e) := |\{P : e \in P\}|$
      **end if**
    **end for**
    $b_i \leftarrow b_i + \frac{1}{a^*}$
  **end for**
  **if** $R(P_i) < R(P) \wedge R(P^*) - b_i p\delta < R(P_i)$ **then**
    $P \leftarrow P_i$ ▷ Compare current path to best path found thus far
  **end if**
**end for**
**output** $P$

## 5.3 Proof of Optimality of Algorithm 2

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. □

**Theorem 5.1.** *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 $\frac{p \cdot N \cdot \delta}{h \cdot N} = \frac{p \cdot \delta}{h}$. 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. □

# 6 $\epsilon$-Greedy Victim

We consider the sequence of paths that the victim perceives to be optimal. The adversary's goal is to perturb the victim such that the last path in the sequence has as low of a reward as possible. We consider the phases in the sequence, where each switch to a new optimally perceived path marks the start of a new phase. Let $N_i$ denote the number of samples in phase $i$.

Moreover, our victim begins with a warm start, where it samples each path $n$ times. During the warm start, the adversary corrupts as according to Algorithm 2.

Corrupting based off of Algorithm 2 *after* the warm start may be suboptimal due to the victim's unequal sampling. In the phase after warm start, much of the previous corruption on the greedily sampled path will be lost due to the adversary only being able to corrupt $p\delta$ when that path is traversed. We analyze the degradation of the warm start strategy empirically in section 7 and give an example in section 6, along with an alternative strategy.

## 6.1 Improving on Random-Sampling Victim Corruption

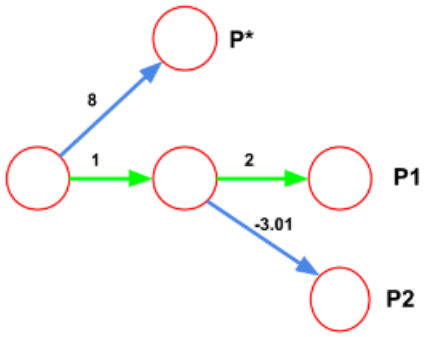
Intuitively, an $\epsilon$-greedy victim allows the adversary to gain even greater corruption budget than from the warm start. This is because the adversary can corrupt intermediate value paths as a middleman to then switch to lower reward paths that would have been unobtainable with just the warm start. The adversary can corrupt the edge that this intermediate path has in common with some target path or with the most optimal path, which results in a higher corruption budget as the intermediate path is sampled more often. For certain

MDP structures with bounded reward gaps, multiple path switches are possible, a regime we aim to parameterize further.

However, it is also important to note that $\epsilon$-greedy sampling causes the effectiveness of corrupting paths upwards to decrease over time. This is because the adversary is unable to maintain all the corruption on the path when it is greedily sampled.

### 6.1.1 Improving on Random-Sampling in a Simple MDP

Let's consider another MDP with three paths: $P_1$, $P_2$, and $P_3$ where $R(P_2) < R(P_1) < R(P_3)$. Below is a picture of the MDP. First, we will solve for bounds on gaps of rewards, $N$, and $x$ for all MDPs of this structure. Then we will numerically calculate possible $N$ and $x$ with this specific MDP's rewards, and demonstrate the path switch.




This MDP has disjoint edges on each path, along with $P_1 \cap P_2$. Also, $R(P_3) - R(P_1) < 2p\delta$ in this example.

**Warm Start.** The adversary perturbs the disjoint edges of $P_1$, $P_2$, and $P_3$. After the end of warm start, the victim will perceive the rewards of the paths as:

$$(P_1, P_2, P_3) = (R(P_1) + p\delta, R(P_2) + p\delta, R(P_3) - p\delta).$$

**After Warm Start.** After the warm start, the adversary wants to switch $P_2$ with $P_3$. Its strategy is to first corrupt edge $P_1 \cap P_2$ upward by $p\delta$ for $x$ fraction of samples, then corrupt the disjoint edge of $P_1$ downward by $p\delta$ for $1 - x$ fraction samples. Whenever the victim samples $P_2$ or $P_3$, we corrupt the disjoint edges up and down respectively. After all sampling ends, the rewards are:

$P_1$ is $R(P_1) + \frac{np\delta - (1-x)(1-\epsilon)Np\delta}{n+(1-\epsilon)N} + \frac{x(1-\epsilon)Np\delta}{2n+(1-\frac{\epsilon}{2})N}$,
where the second term comes from warm start corruption on $P_1$'s disjoint edge subtracted by the downwards corruption after warm start, and the third term is the corruption on $P_1 \cap P_2$ edge after warm start.

$P_2$ is $R(P_2) + p\delta + \frac{x(1-\epsilon)Np\delta}{2n+(1-\frac{\epsilon}{2})N}$,
where the second term is corruption on disjoint edge and the third term is corruption on $P_1 \cap P_2$.

$P_3$ is $R(P_3) - p\delta$.

Thus we have the following two relations corresponding to the gaps for which this strategy will work:

$$R(P_1) + \frac{np\delta - (1-x)(1-\epsilon)Np\delta}{n + (1-\epsilon)N} + \frac{x(1-\epsilon)Np\delta}{2n + (1-\frac{\epsilon}{2})N} \leq R(P_2) + p\delta + \frac{x(1-\epsilon)Np\delta}{2n + (1-\frac{\epsilon}{2})N},$$

and

$$R(P_3) - p\delta \leq R(P_2) + p\delta + \frac{x(1-\epsilon)Np\delta}{2n + (1-\frac{\epsilon}{2})N}.$$

Simplified, this is:

$$R(P_1) - R(P_2) \leq \frac{(2-x)(1-\epsilon)Np\delta}{n + (1-\epsilon)N},$$

and

$$R(P_3) - R(P_2) \leq 2p\delta + \frac{x(1-\epsilon)Np\delta}{2n + (1-\frac{\epsilon}{2})N}.$$

When these conditions are satisfied, the adversary is able to switch $P_2$ with $P_3$, a feat it would not have been able to do otherwise.

**Bound on $N$ and Formula for $x$.** From these relations, we can get a lower bound on $N$, and a formula for $x$. These quantities will be useful in crafting a robust victim defense. From the second inequality, we get a bound on $N$:

$$N \geq \frac{2n(R(P*) - R(P_2) - 2p\delta)}{p\delta x(1-\epsilon) - (R(P_3) - R(P_2) - 2p\delta)(1-\frac{\epsilon}{2})}.$$

We can solve for the equality case of the first relation, which gives us that

$$x = \frac{2(1-\epsilon)Np\delta - (R(P_1) - R(P_2))(n + (1-\epsilon)N)}{(1-\epsilon)Np\delta}.$$

For this specific MDP, we can simplify the bound on $N$ derived from the inequality by accounting for budget/free corruption, since $R(P*) - R(P_2) > \frac{5}{2}p\delta$. We can bound $N$ by

$$N \geq \frac{2n}{2x(1-\epsilon) - (1-\frac{\epsilon}{2})}.$$

**Example Calculations.** Using the above relations, we can calculate that for this specific MDP, $N = 1500$ and $x = 0.654$ will allow for $P_2$ and $P_3$ to switch. Here, $R(P_1) = 3$, $R(P_2) = -2.01$, and $R(P_3) = 8$. Let's set $n = 100$, $p\delta = 4$, and $\epsilon = 0.1$.

After warm start, the rewards are: $(P_1, P_2, P_3) = (3 + 4, -2.01 + 4, 8 - 4) = (7, 1.99, 4)$.

After another 1500 samples, the rewards are:

$$(P_1, P_2, P_3) = (3 - 1.012 + 2.173, 1.99 + 2.173, 4) = (4.161, 4.163, 4).$$

$P_2$ is now perceived as most optimal by the victim!

## 6.2 Upper Bound on Total Corruption

While the adversary is able to improve upon its warm start corruption in some cases, the lowest reward path it can switch to depends highly on the amount it can possibly corrupt during warm start. We get a maximum of $\frac{(1-\epsilon)p\delta}{1-(1-a)\epsilon}$ budget total whenever the victim's greedy path is traversed and the edge (on $a$ fraction of the paths) which the greedy path has in common with $P*$ or $\hat{P}$. For every other traversal, we will get a maximum of warm start budget of switching $\hat{P}$ with $P*$.
Thus, we have an upper bound of

$$\boxed{\max_{\hat{P}} \text{warm}(\hat{P}) + \text{warm}(P*) + \frac{(1-\epsilon)p\delta}{1-(1-a)\epsilon}}$$

on total possible corruption.

## 6.3 Lower Bound on $N_i$

In order to construct an $\epsilon$-annealing algorithm, we must parameterize the lengths of phases $N_i$. We can use the upper bound of corruption to derive a lower bound on $N_i$. By the upper bound, we must have the inequality

$$R(P*) - R(P_i) < \text{warm}(P_i) + \text{warm}(P*) + c_{i-1} + \frac{(1-x)(1-\epsilon)N_i p\delta}{(1-(1-a)\epsilon)(\sum_{j=1}^{i} N_j)}.$$

in order for $P*$ and $P_i$ to be switchable, where $x$ is the fraction of the time that the current greedy path's shared edge with $P_i$ is corrupted and $c_{i-1}$ is the amount of previous corruption from greedy paths. This simplifies to

$$\boxed{N_i > \frac{(R(P*) - R(P_i) - \text{warm}(P_i) - c_{i-1})(1-(1-a)\epsilon)(\sum_{j=1}^{i} N_j)}{(1-x)(1-\epsilon)p\delta - (1-(1-a)\epsilon)(R(P*) - R(P_i) - \text{warm}(P_i)) - c_{i-1}}}.$$

## 6.4 Adversarial Algorithm Against an $\epsilon$-Greedy Victim

We can use the bound in section 6.3 to guide us in selecting the path we aim to ultimately switch to at the end of training. We seek the path $\hat{P}$ with lowest possible reward such that

$$\text{warm}(\hat{P}) + \text{warm}(P*) + \frac{(1-\epsilon)p\delta}{1-(1-a)\epsilon} < R(P*) - R(\hat{P}),$$

where the left hand side is the upper bound on total corruption. Therefore, $\hat{P}$ the worst that the adversary can possibly manipulate the victim into choosing.

Simultaneously, the adversary must ensure that the victim does not accidentally switch to paths with greater reward than $\hat{P}$'s final reward. This is because if these paths are sampled greedily, it is difficult to recover the downward corruption lost as the perturbations on greedy paths degrades. Thus, the adversary corrupts these paths sufficiently so that they are not perceived as optimal.

### 6.4.1 Heuristic Using Stalling

Intuitively, we "save" our corruption for the final switch by not directing corruption in an attempt to make the greedy paths strictly decreasing in reward. Rather, we focus our corruption onto $\hat{P}$, along with the paths for which it is absolutely necessary to be corrupted.

In particular, we can ignore the order of the greedily perceived paths before the penultimate switch because after a certain amount of samples, all the top perceived few paths will have already degraded. Therefore, we corrupt some paths with a lower reward than $\hat{P}$'s final reward upwards enough to just stall in the meantime. This stalling allows us to effectively manipulate the total fixed number of samples $N$ so that we can just consider switching the "important" paths, obtaining the shortest sequence of path switches possible. This is typically just one penultimate switch.

After running Algorithm 3, the adversary corrupts the victim's data stream based off of the edge sets $D$ and $U$ returned.

Algorithm 3 performs better than Algorithm 2 in several key cases, illustrated in the next section. Currently, we are investigating a proof of optimality.

**Algorithm 3** Adversarial Attack Against $\epsilon$-Greedy Victim

**input:** $\epsilon$, MDP $\mathcal{M}$

$P \leftarrow P^*$ ▷ The final path we switch to
$D, U \leftarrow \emptyset$ ▷ Initialize final edge sets corresponding to corruptions down/up
**for all** $\hat{P} \in \mathcal{M} \wedge \hat{P} \neq P^*$ **do** ▷ Choosing $P$
    hasDown $\leftarrow$ true ▷ Some paths need to be corrupted down for $P$ to be reachable
    $D_{\hat{P}}, U_{\hat{P}} \leftarrow \emptyset$ ▷ Initialize edge sets corresponding to $\hat{P}$
    **if** warm$(\hat{P}) + \frac{(1-\epsilon)p\delta}{1-(1-a)\epsilon} < R(P*) - R(\hat{P})$ **then** ▷ Budget needed below upper bound
        continue
    **end if**
    $U_{\hat{P}} \leftarrow$ GetUpwardsCorruptions($\mathcal{M}$, $\hat{P}$, $U_{\hat{P}}$, $D_{\hat{P}}$, $n$, $N$, $\epsilon$) ▷ Corruptions on low-reward paths
    $D_{\hat{P}} \leftarrow$ GetDownwardsCorruptions($\mathcal{M}$, $\hat{P}$, $D_{\hat{P}}$) ▷ Corruptions on high-reward paths
    hasDown $\leftarrow$ Reachability$(\mathcal{M}, \hat{P}, D_{\hat{P}})$ ▷ Check viability of assigned corruptions
    **if** hasDown $\wedge R(\hat{P}) < R(P)$ **then**
        $P \leftarrow \hat{P}, D \leftarrow D_{\hat{P}}, U \leftarrow U_{\hat{P}}$
    **end if**
**end for**
**return:** $U$, $D$

**Helper Functions:**

**Algorithm 4 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; $\epsilon$

$R_f(\hat{P}) \leftarrow R(P*) - warm(P*)$ ▷ Calculate final reward of $\hat{P}$
**if** $\text{warm}(\hat{P}) < R(P*) - R(\hat{P})$ **then** ▷ Find path which shares edge with $\hat{P}$
  Let $e \in \hat{P}$ satisfy $e = \arg\max_e a_e$ ▷ on more paths, bigger difference during greedy
  **for all** $P_e$ **do** ▷ paths containing $e$
    **if** $R(P_e) + p\delta + \text{warm}(P_e) > R_f(\hat{P})$ **then**
      Solve for $N_{P_e}$ with $\frac{(1-\epsilon)(\text{warm}(P_e)-p\delta)(N_{P_e})}{((1-\epsilon)N_{P_e}+\epsilon(N-N_{P_e}))} = R(P_e) - R_f(P)$
      $U \leftarrow U + (e, P_e, N - N_{P_e})$ ▷ Start corrupting at sample $N - N_{P_e}$
      Find $P_{stall}$ where $P_{stall} = \arg\max_P R(P)$ and $R(P_{stall}) < R_f(\hat{P})$
      $U \leftarrow U + (e_d, P_{stall}, 0)$ ▷ Corrupt $P_{stall}$ upwards to stall victim
      **if** $R(P_e) + p\delta > R_f(\hat{P})$ **then**
        $U \leftarrow U + (e, P_e)$ ▷ Corrupt $P_e$'s common edge up
        $N_{P_e} \leftarrow (R(P_e) - R(\hat{P}))((1-\epsilon)N + n) - n$ ▷ Samples to switch $P_e$ and $\hat{P}$
        $D \leftarrow D + (e, P_e, N - N_{P_e})$ ▷ Corrupt disjoint edge down to switch to $\hat{P}$
        break ▷ Prefer initializing $U$ and $D$ here, there's no degradation estimate
      **end if**
    **end if**
  **end for**
**end if**
**if** $\text{warm}(\hat{P}) > R(P*) - R(\hat{P})$ **then** ▷ Determine a path to stall with
  Find $P_{stall}$ where $P_{stall} = \arg\max_P R(P)$ and $R(P_{stall}) < R_f(\hat{P})$
  $U \leftarrow U + (e_d, P_{stall}, 0)$ ▷ Corrupt $P_{stall}$ upwards to stall victim
  $N_{P_e} \leftarrow \frac{(R(\hat{P})-R(P_{stall}))((1-\epsilon)N+n)}{2p\delta} - \frac{n}{2} + (1-\epsilon)\frac{N}{2}$ ▷ samples to switch $P_{stall}$ and $\hat{P}$
  $D \leftarrow D + (e, P_{stall}, N_{P_{stall}})$ ▷ Corrupt disjoint edge down to switch to $\hat{P}$
**end if**
**return**: $U$, $D$

**Algorithm 5 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

$R_f(\hat{P}) \leftarrow R(P*) - \text{warm}(P*)$ ▷ Calculate final reward of $\hat{P}$
**for all** $\dot{P} \in \mathcal{M} \wedge R(\dot{P}) > R_f(\hat{P})$ **do** ▷ Track edges that need to be corrupted
    $D \leftarrow D + (e_d, \dot{P})$ ▷ add the disjoint edge to the set
**end for**
$e^* \leftarrow, b_e \leftarrow 0$ ▷ Edge to perturb and corresponding value
**for all** $\tilde{P} \in \mathcal{M} \wedge R(\tilde{P}) < R_f(\hat{P}) \wedge \notin P_p$ **do** ▷ We corrupt remaining paths as needed
    **for all** $\dot{P} \in \mathcal{M} \wedge R(\dot{P}) > R_f(\hat{P})$ **do**
        $e \leftarrow \tilde{P} \cap \dot{P}$
        **if** $\frac{1}{a_e} > b_e$ **then** ▷ Updating best edge to perturb
            $e^* \leftarrow e$, $b_e \leftarrow \frac{1}{a_e}$
        **end if**
    **end for**
    $D \leftarrow D + (e^*, \tilde{P})$
**end for**
**return**: $D$

**Algorithm 6 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

$R_f(\hat{P}) \leftarrow R(P*) - \text{warm}(P*)$ ▷ Calculate final reward of $\hat{P}$
hasDown $\leftarrow$ **True**
**for all** $\dot{P} \in \mathcal{M} \wedge R(\dot{P}) > R_f(\hat{P})$ **do**
    $b \leftarrow 0$ ▷ Initialize budget
    **for all** $e \in D \wedge e \in \dot{P}$ **do**
        $b \leftarrow b + \frac{1}{a_e}$ ▷ Update budget
    **end for**
    **if** $b < R(\dot{P}) - R_f(P)$ **then** ▷ Ensure sufficient budget to corrupt $\dot{P}$ downwards
        hasDown $\leftarrow$ **False**
    **end if**
**end for**
**return**: hasDown

### 6.4.2 Examples

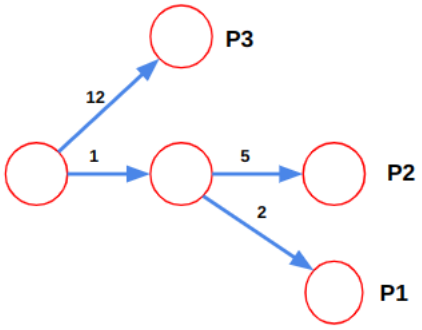
In this section, we provide two examples:

In the first, it is possible to switch to the worst path during the warm start period, but the corruption degrades over time. Thus, it is necessary to corrupt the middle path to be optimal right after warm start, and then switch to the worst path after.

In the second example, it is possible to switch both to the worst path, as well as a path that

shares a common edge with it, during the warm start. However, corruption will degrade on both paths, so it is necessary to first stall the victim.

Consider an MDP with three paths: $P_1$, $P_2$, and $P_3$ where $R(P_1) < R(P_2) < R(P_3)$. Below is a picture of the MDP.

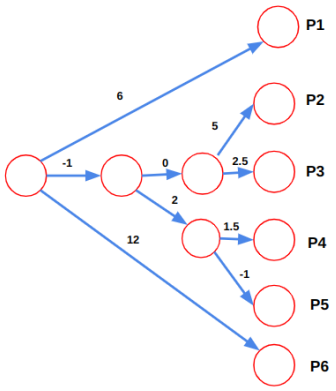



Let the length of the warm start $n = 150$ with 50 samples on each path, the $\epsilon$-greedy period be $N = 150$, $\epsilon = 0.2$and $p\delta = 4$.

Let's say that the adversary were to maintain the warm start corruption, perturbing disjoint edges on $P_1$ and $P_3$ as well as $P_1 \cap P_2$. Using the formulas derived in section 6.1, corruption on $P_1 \cap P_2$ would degrade after around 100 samples in the $\epsilon$-greedy phase and $P_2$ would be viewed as optimal.

However, the adversary could instead perturb all the disjoint edges during warm start. In the phase after, it could perturb $P_1 \cap P_2$ whenever the greedy path $P_2$ is sampled. After around 75 samples, the rewards of $P_1$ and $P_3$ will have switched and we can corrupt $P_2$'s disjoint edge for another 50 samples for it to be perceived as suboptimal. Thus, we are able to switch to the worst path with this strategy.

Below is a second example. Let $p\delta = 4$, $\epsilon = 0.2$, $n = 120$, $N = 5000$. The rewards are $P_1 = 6$, $P_2 = 4$, $P_3 = 1.5$, $P_4 = 2.5$, $P_5 = 0$, and $P_6 = 12$.

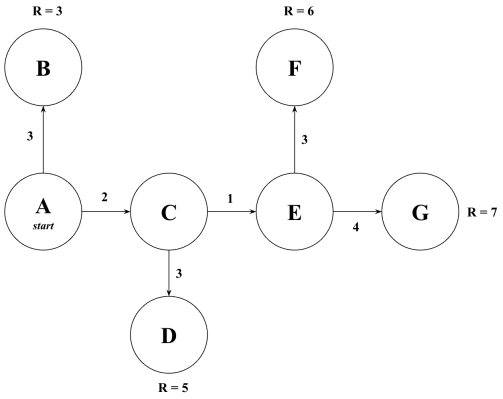


If we were to put full warm start corruption on path $P_3$, we would obtain free corruption of $p\delta = 4$ and budget of 8. $P_3$ would reach a reward of 9.9 and we could switch it with optimal path $P_6$, however it would degrade as this corruption relies so much on free corruption. If we tried to switch first from $P_4$ to $P_3$, we would either lose corruption on $P_4$ before being able to switch to $P_3$, or lose corruption after switching to $P_3$ as $N$ is large. Thus, we can stall by first switching to path $P_1$, then after a while corrupting A downwards so we can switch to $P_4$ and then to the desired $P_3$.

# 7 Experiment

## 7.1 Preliminary Results

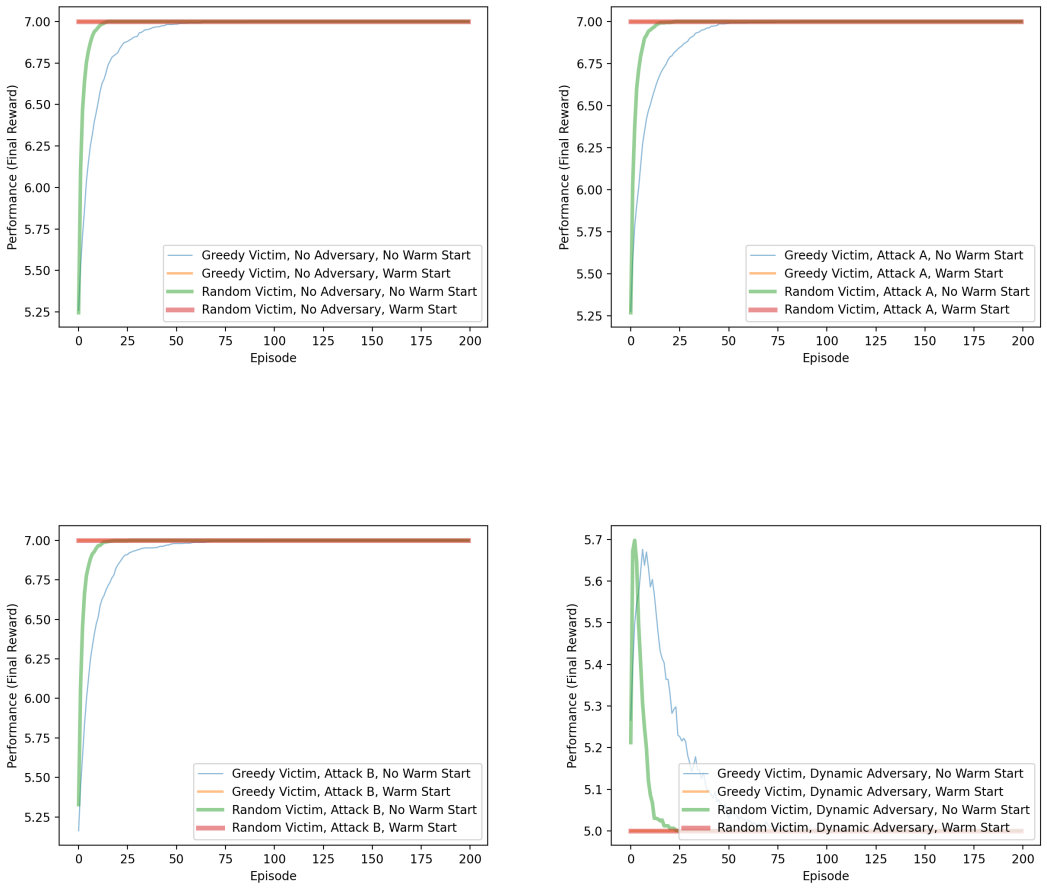
The following experiment[2] was conducted in this MDP:




It included three paradigms: whether the adversary was absent, baseline (A & B), or dynamic (Algorithm 2); 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].

[2]Code is made available at `https://github.com/Gerbil613/primes-rl-test`.

[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 2 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[4], the $\epsilon$-greedy victim maintains higher performance than its random counterpart.

[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.

## 7.2 Random MDP Generation and General Analysis of their Structure

This work considers the following algorithm, which generates a random[5] 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:

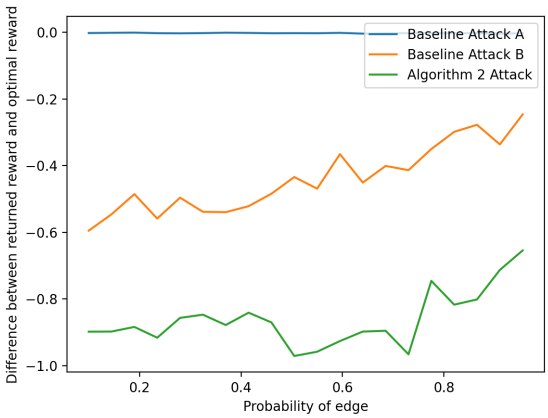
**Algorithm 7** Random MDP Generation

**input:** $n_s, p_e$ ▷ input number of states in MDP and likelihood of edge existing between two given states

$\mathcal{S} \leftarrow \{0, 1, \ldots, n_s\}$
**while True do** ▷ Keep on iterating until fully connected MDP is attained
  $\mathcal{A} \leftarrow \{\}$
  **for all** $s_1$ in $\mathcal{S}$ **do**
    $a \leftarrow 0$ ▷ Keep track of amount of actions in $s_1$
    **for all** $s_2 \in \mathcal{S}$ such that $s_2 > s_1$ **do** ▷ Iterate through pairs of states
      **if** random$(0, 1) < p_e$ **then** ▷ Do with probability $p_e$
        $\mathcal{R}(s_1, s_2) \leftarrow$ random$(-1, 1)$ ▷ Expected edge reward is 0, so that longer paths do not have higher expected rewards
        $\mathcal{P}(s_1, a, s_2) \leftarrow 1$ ▷ Draw edge between $s_1$ and $s_2$
        $a \leftarrow a + 1$ ▷ There is now one more possible action in $s_1$
      **end if**
    **end for**
    **if** $a > |\{\mathcal{A}\}|$ **then** ▷ We use $a$ to ensure that our action space contains the right number of actions
      $\mathcal{A} \leftarrow \{0, 1, \ldots, a\}$
    **end if**
  **end for**
  **if** $\forall s \in \mathcal{S} : (\exists P : s \in P) \wedge |\{P : P \in \mathcal{M}\}| > 1$ **then** ▷ Verify that MDP is fully connected and not trivial; otherwise, reject and start over
    **return** $\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$
  **end if**
**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 2 achieves the greatest difference between the optimal reward and the victim's returned reward:

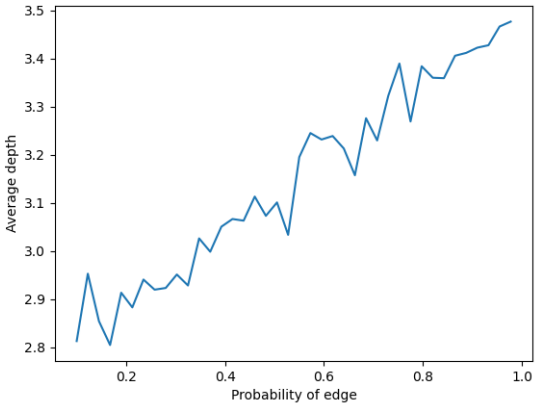
[5] We ignore trivial MDPs in which the gap between the optimal path and all other paths is so large that the adversary cannot corrupt the victim into switching to any path at all. Thus, we also ignore MDPs that have only one path.




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.

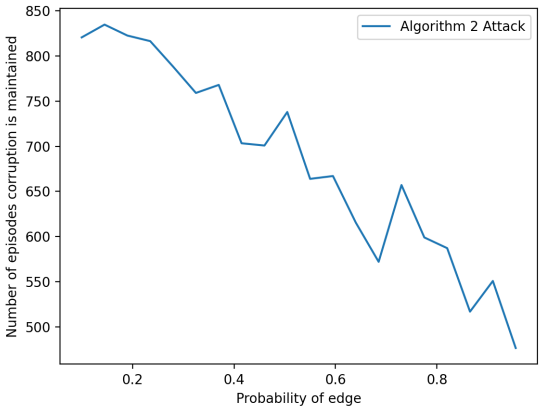
However, the fact that Algorithm 2 experiences a general trend of decreasing the difference between optimal and returned reward is of great interest. It demonstrates that for denser and denser MDPs (that is, MDPs with more chances at free corruption), the ability to *maintain* warm-start corruption wanes, 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 2'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 $p_e$):



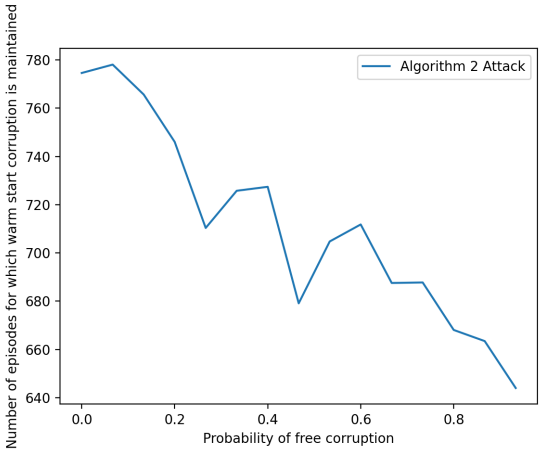

Indeed, denser MDPs are likely to have longer (deeper) paths. This is because a denser MDP has more edges, and is therefore more likely to possess longer, consecutive chains of edges; i.e. longer paths.

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 2, 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 2, 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:




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 investigate the optimality of Algorithm 3 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.

Once an optimal adversarial strategy is established, we can analyze the $\epsilon$-greedy victim's regret for varying amounts of corruption levels. From here, we can construct an $\epsilon$-annealing algorithm that is robust to the adversarial strategy.

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. Pavel Etingof, and Dr. Srini Devadas for organizing this unique program.

## References

[1] Kiarash Banihashem, Adish Singla, and Goran Radanovic. "Defense against reward poisoning attacks in reinforcement learning". In: *arXiv preprint arXiv:2102.05776* (2021).

[2] Chris Dann et al. "Guarantees for epsilon-greedy reinforcement learning with function approximation". In: *International Conference on Machine Learning*. PMLR. 2022, pp. 4666–4689.

[3] Ben Eysenbach. *Maximum Entropy RL (Provably) Solves Some Robust RL Problems*. `https://bair.berkeley.edu/blog/2021/03/10/maxent-robust-rl/`. Accessed 29 June 2022.

[4] Adam Gleave et al. "Adversarial policies: Attacking deep reinforcement learning". In: *arXiv preprint arXiv:1905.10615* (2019).

[5] Yuan Gong et al. "Real-time adversarial attacks". In: *arXiv preprint arXiv:1905.13399* (2019).

[6] Sandy Huang et al. "Adversarial attacks on neural network policies". In: *arXiv preprint arXiv:1702.02284* (2017).

[7] Thodoris Lykouris, Vahab Mirrokni, and Renato Paes Leme. "Stochastic bandits robust to adversarial corruptions". In: *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*. 2018, pp. 114–122.

[8] Daniel J Mankowitz et al. "Robust reinforcement learning for continuous control with model misspecification". In: *arXiv preprint arXiv:1906.07516* (2019).

[9] Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).

[10] Alberto Natali et al. "Online Graph Learning From Time-Varying Structural Equation Models". In: *2021 55th Asilomar Conference on Signals, Systems, and Computers*. IEEE. 2021, pp. 1579–1585.

[11] Lerrel Pinto et al. "Robust adversarial reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 2817–2826.

[12] Amin Rakhsha et al. "Policy teaching via environment poisoning: Training-time adversarial attacks against reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7974–7984.

[13] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[14] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. 2018. ISBN: 9780262352703.

[16] Gerald Tesauro. "TD-Gammon, a self-teaching backgammon program, achieves master-level play". In: *Neural computation* 6.2 (1994), pp. 215–219.

[17] Chen Tessler, Yonathan Efroni, and Shie Mannor. "Action robust reinforcement learning and applications in continuous control". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6215–6224.

[18] Hang Xu et al. "Transferable environment poisoning: Training-time attack on reinforcement learning". In: *Proceedings of the 20th international conference on autonomous agents and multiagent systems*. 2021, pp. 1398–1406.

[19] Xuezhou Zhang et al. "Adaptive reward-poisoning attacks against reinforcement learning". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 11225–11234.

[20] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. "Adversarial attacks on neural networks for graph data". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 2847–2856.