# How Optimal Can We Get: Stochastic and Adversarial Reinforcement Learning

MIT PRIMES, Mentor: Mayuri Sridhar

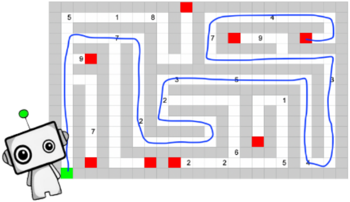
Alicia Li and Mati Yablon

MIT

October 16, 2022

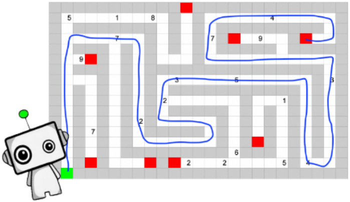
# A Cute Robot in A Cute Maze

We (a cute robot) need to find the optimal path in this maze!



# A Cute Robot in A Cute Maze

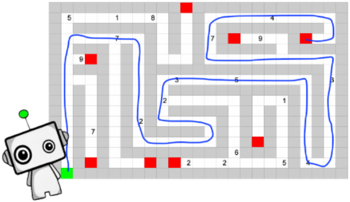
We (a cute robot) need to find the optimal path in this maze!



We could try every path in the maze, but this is inefficient :(

# A Cute Robot in A Cute Maze

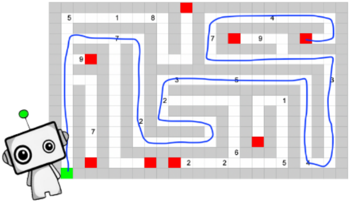
We (a cute robot) need to find the optimal path in this maze!



We could try every path in the maze, but this is inefficient :(
Let's use Reinforcement Learning! Every time we take an **action**, we receive a **reward**, which shapes our future actions.

# A Cute Robot in A Cute Maze

We (a cute robot) need to find the optimal path in this maze!



We could try every path in the maze, but this is inefficient :(
Let's use Reinforcement Learning! Every time we take an **action**, we receive a **reward**, which shapes our future actions.
**Let's formalize this notion...**

# Markov Decision Processes

## Definition of MDP (Markov Decision Process)

$$\mathcal{M} := (\mathcal{S}, \mathcal{A}, R, \mathcal{P})$$

# Markov Decision Processes

## Definition of MDP (Markov Decision Process)

$$\mathcal{M} := (\mathcal{S}, \mathcal{A}, R, \mathcal{P})$$

- $\mathcal{S}$ is **state space**: Set of all states in which the agent may be
- $\mathcal{A}$ is **action space**: Set of all actions which the agent may take in a state
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is **reward function**: Outputs the reward given to the agent when taking action $a$ in state $s$
- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is **transition dynamics function**: Outputs the probability of the agent transitioning to new state $s'$ if it takes action $a$ in state $s$

# $\epsilon$-greedy Policy



## Definition of policy $\pi$

A policy $\pi$ is a mapping of the state and action spaces to a probability that dictates the agent's behavior.

# $\epsilon$-greedy Policy



## Definition of policy $\pi$

A policy $\pi$ is a mapping of the state and action spaces to a probability that dictates the agent's behavior.

$\epsilon$-greedy:

- Probability $\epsilon$: sample random action
- Probability $1 - \epsilon$: take best perceived action $\arg\max_a Q(s, a)$.

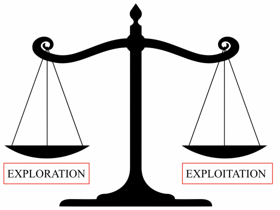
# $\epsilon$-greedy Policy

## Definition of policy $\pi$

A policy $\pi$ is a mapping of the state and action spaces to a probability that dictates the agent's behavior.

$\epsilon$-greedy:

- Probability $\epsilon$: sample random action
- Probability $1 - \epsilon$: take best perceived action $\arg\max_a Q(s, a)$.



# Q-values

Now how does RL work? Central goal is to learn an optimal policy (i.e. behavior)

# Q-values

Now how does RL work? Central goal is to learn an optimal policy (i.e. behavior)

Q-values store how "good" a state is

Approaches the expected value $Q(s_t, a_t) \approx \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t]$.

# Q-values



Now how does RL work? Central goal is to learn an optimal policy (i.e. behavior)

Q-values store how "good" a state is

Approaches the expected value $Q(s_t, a_t) \approx \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t]$.

Learned via Bellman optimality equation:

$$Q(s_t, a_t) \leftarrow (1 - \alpha) Q(s_t, a_t) + \alpha (R_t + \gamma \max_a Q(s_{t+1}, a)).$$

# Q-values

Now how does RL work? Central goal is to learn an optimal policy (i.e. behavior)

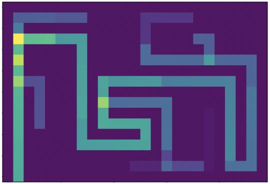
Q-values store how “good” a state is

Approaches the expected value $Q(s_t, a_t) \approx \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_t]$.

Learned via Bellman optimality equation:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(R_t + \gamma \max_a Q(s_{t+1}, a)).$$

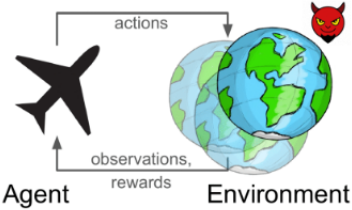
Heat map of learned Q-values:



# Adversarial RL

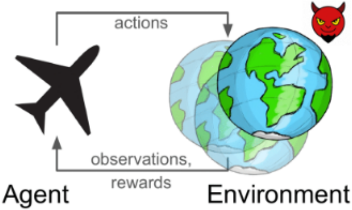


What if something perturbs the MDP?



# Adversarial RL

What if something perturbs the MDP?




Performance can be degraded by:

- Human biases
- Modeling errors
- Actual adversaries

# Robust RL

## Definition

Robust RL aims to find the best-performing policy in the worst-case scenario. It can be framed as a 2-player zero-sum game.

Objective: Find the policy $\pi$ that satisfies:

$$\max_{\pi} \min_{\mathcal{P}} \mathbb{E}_{\pi,\mathcal{P}}\left[\sum_t R_t\right],$$

where $\mathcal{P}$ is the environment and $R_t$ is the reward at time $t$.

# Robust RL



## Definition

Robust RL aims to find the best-performing policy in the worst-case scenario. It can be framed as a 2-player zero-sum game.

Objective: Find the policy $\pi$ that satisfies:

$$\max_{\pi} \min_{\mathcal{P}} \mathbb{E}_{\pi,\mathcal{P}}\left[\sum_{t} R_t\right],$$

where $\mathcal{P}$ is the environment and $R_t$ is the reward at time $t$.

Robust RL Methods Include:

- Injecting noise into the environment during training (Maximum Entropy)

# Robust RL

## Definition

Robust RL aims to find the best-performing policy in the worst-case scenario. It can be framed as a 2-player zero-sum game.

Objective: Find the policy $\pi$ that satisfies:

$$\max_{\pi} \min_{\mathcal{P}} \mathbb{E}_{\pi,\mathcal{P}}\left[\sum_t R_t\right],$$

where $\mathcal{P}$ is the environment and $R_t$ is the reward at time $t$.

Robust RL Methods Include:

- Injecting noise into the environment during training (Maximum Entropy)
- Train the agent in an environment with an adversary that corrupts the reward function

# Best of Both Worlds

We want to perform well in **all** environments, not just worst-case scenarios...

# Best of Both Worlds

We want to perform well in **all** environments, not just worst-case scenarios... Best of Both Worlds!

## Definition

Best of Both Worlds: We want performance that degrades gracefully with an increasing corruption level, can be used in RL

Best of Both Worlds Methods:

- Layering algorithms designed for varying corruption levels

# Problem Setting



Previous work [2] in Best-Of-Both-Worlds has focused on bandit MDPs We consider layered

# Problem Setting

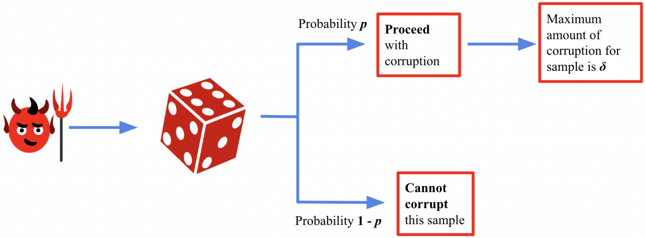
Previous work [2] in Best-Of-Both-Worlds has focused on bandit MDPs We consider layered For every sample, our adversary is able to:

- Corrupt the edges that victim traverses with probability $p$
- Corrupt that edge's reward by a maximum of $\delta$ each

# Problem Setting

Previous work [2] in Best-Of-Both-Worlds has focused on bandit MDPs We consider layered For every sample, our adversary is able to:

- Corrupt the edges that victim traverses with probability $p$
- Corrupt that edge's reward by a maximum of $\delta$ each



# Calculating Adversarial Budget to Switch Paths



Adversary wants to make optimal path seem worse than some suboptimal path, how much budget does it have? (victim traverses each path equally)

# Calculating Adversarial Budget to Switch Paths

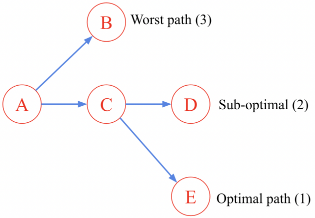
Adversary wants to make optimal path seem worse than some suboptimal path, how much budget does it have? (victim traverses each path equally)
Consider the following MDP:

# Calculating Adversarial Budget to Switch Paths



Adversary wants to make optimal path seem worse than some suboptimal path, how much budget does it have? (victim traverses each path equally)

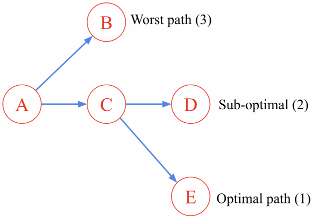
Consider the following MDP:



# Calculating Adversarial Budget to Switch Paths

Adversary wants to make optimal path seem worse than some suboptimal path, how much budget does it have? (victim traverses each path equally)

Consider the following MDP:

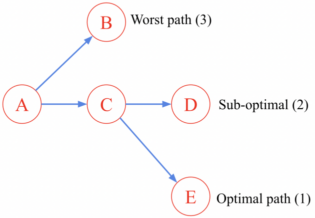



Naive Approach: $p\delta$ each from corrupting AB up and CE down whenever paths 3 and 1 are traversed, yielding $2p\delta$

# Calculating Adversarial Budget to Switch Paths



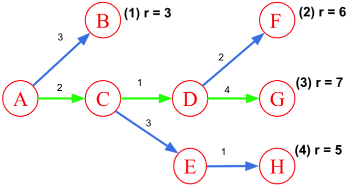
Adversary wants to make optimal path seem worse than some suboptimal path, how much budget does it have? (victim traverses each path equally)
Consider the following MDP:




Naive Approach: $p\delta$ each from corrupting AB up and CE down whenever paths 3 and 1 are traversed, yielding $2p\delta$
Our Approach: $2p\delta +$ extra $\frac{1}{2}p\delta$ of "free corruption" from corrupting AC whenever path 2 is traversed

# Adversarial Attack

Let's attack! Given that $p = 0.25$ and $\delta = 4; p\delta = 1$

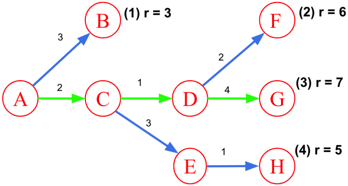



**Algorithm 1** Adversarial Attack: Switching Non-disjoint Paths

$P \leftarrow P^*$ ▷ 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
  $b_i \leftarrow 2$ ▷ Add budget for the two disjoint edges on the paths we want to switch
  **for all** $P_j \notin \{P_i, P^*\}$ **do** ▷ Iterate through paths that are not optimal or $P_i$
    $a_e^* = \infty$
    **for all** $e \in P_j$ **do**
      **if** $(e \in P^* \oplus e \in P_i) \wedge a_e < a_e^*$ **then**
        $a_e^* \leftarrow a_e$ ▷ Calculate budget
      **end if**
    **end for**
    $b_i \leftarrow b_i + \frac{1}{a_e}$
  **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** $P$

# Adversarial Attack

Let's attack! Given that $p = 0.25$ and $\delta = 4; p\delta = 1$



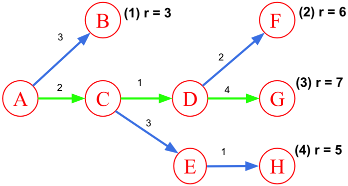

**Algorithm 1** Adversarial Attack: Switching Non-disjoint Paths

$P \leftarrow P^*$ ▷ 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
  $b_i \leftarrow 2$ ▷ Add budget for the two disjoint edges on the paths we want to switch
  **for all** $P_j \notin \{P_i, P^*\}$ **do** ▷ Iterate through paths that are not optimal or $P_i$
    $a_e^* = \infty$
    **for all** $e \in P_j$ **do**
      **if** $(e \in P^* \oplus e \in P_i) \wedge a_e < a_e^*$ **then**
        $a_e^* \leftarrow a_e$ ▷ Calculate budget
      **end if**
    **end for**
    $b_i \leftarrow b_i + \frac{1}{a_e}$
  **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** $P$

Budget of Switching:

# Adversarial Attack

Let's attack! Given that $p = 0.25$ and $\delta = 4; p\delta = 1$



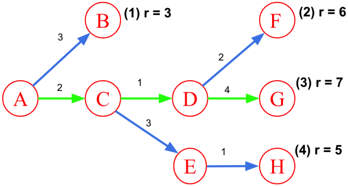

**Algorithm 1** Adversarial Attack: Switching Non-disjoint Paths

$P \leftarrow P^*$ ▷ 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
  $b_i \leftarrow 2$ ▷ Add budget for the two disjoint edges on the paths we want to switch
  **for all** $P_j \notin \{P_i, P^*\}$ **do** ▷ Iterate through paths that are not optimal or $P_i$
    $a_e^* = \infty$
    **for all** $e \in P_j$ **do**
      **if** $(e \in P^* \oplus e \in P_i) \wedge a_e < a_e^*$ **then**
        $a_e^* \leftarrow a_e$ ▷ Calculate budget
      **end if**
    **end for**
    $b_i \leftarrow b_i + \frac{1}{a_e}$
  **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** $P$

Budget of Switching:
1 with 3: $2\frac{5}{6}$, not enough to switch paths :(

# Adversarial Attack

Let's attack! Given that $p = 0.25$ and $\delta = 4; p\delta = 1$




**Algorithm 1** Adversarial Attack: Switching Non-disjoint Paths

$P \leftarrow P^*$ ▷ 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
  $b_i \leftarrow 2$ ▷ Add budget for the two disjoint edges on the paths we want to switch
  **for all** $P_j \notin \{P_i, P^*\}$ **do** ▷ Iterate through paths that are not optimal or $P_i$
    $a_e^* = \infty$
    **for all** $e \in P_j$ **do**
      **if** $(e \in P^* \oplus e \in P_i) \wedge a_e < a_e^*$ **then**
        $a_e^* \leftarrow a_e$ ▷ Calculate budget
      **end if**
    **end for**
    $b_i \leftarrow b_i + \frac{1}{a_e}$
  **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** $P$

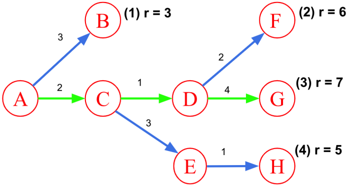
Budget of Switching:

1 with 3: $2\frac{5}{6}$, not enough to switch paths :(

2 with 3: 2, enough to switch paths :)

# Adversarial Attack

Let's attack! Given that $p = 0.25$ and $\delta = 4; p\delta = 1$




**Algorithm 1** Adversarial Attack: Switching Non-disjoint Paths

```
P ← P*    ▷ Current path to perturb; we will iterate and find a better one
for all P_i ∈ M ∧ P_i ≠ P* do    ▷ Iterate only through suboptimal paths
    b_i ← 2 ▷ Add budget for the two disjoint edges on the paths we want to switch
    for all P_j ∉ {P_i, P*} do ▷ Iterate through paths that are not optimal or P_i
        a*_e = ∞
        for all e ∈ P_j do
            if (e ∈ P* ⊕ e ∈ P_i) ∧ a_e < a*_e then
                a*_e ← a_e    ▷ Calculate budget
            end if
        end for
        b_i ← b_i + 1/a_e
    end for
    if R(P_i) < R(P) ∧ R(P*) − b_i pδ < R(P_i) then
        P ← P_i    ▷ Compare current path to best path found thus far
    end if
end for
output P
```

Budget of Switching:

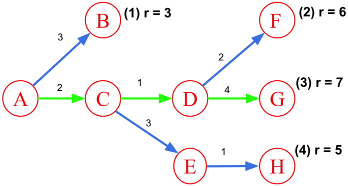
1 with 3: $2\frac{5}{6}$, not enough to switch paths :(

2 with 3: 2, enough to switch paths :)

4 with 3: $2\frac{1}{2}$, enough to switch paths :)

# Adversarial Attack

Let's attack! Given that $p = 0.25$ and $\delta = 4; p\delta = 1$




**Algorithm 1** Adversarial Attack: Switching Non-disjoint Paths

$P \leftarrow P^*$ ▷ 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
  $b_i \leftarrow 2$ ▷ Add budget for the two disjoint edges on the paths we want to switch
  **for all** $P_j \notin \{P_i, P^*\}$ **do** ▷ Iterate through paths that are not optimal or $P_i$
    $a_e^* = \infty$
    **for all** $e \in P_j$ **do**
      **if** $(e \in P^* \oplus e \in P_i) \wedge a_e < a_e^*$ **then**
        $a_e^* \leftarrow a_e$ ▷ Calculate budget
      **end if**
    **end for**
    $b_i \leftarrow b_i + \frac{1}{a_e}$
  **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** $P$

Budget of Switching:

1 with 3: $2\frac{5}{6}$, not enough to switch paths :(

2 with 3: 2, enough to switch paths :)

4 with 3: $2\frac{1}{2}$, enough to switch paths :)

**We choose to switch path 3 with path 4**

# Proof of Optimality (Sketch)



Our algorithm is **optimal**

# Proof of Optimality (Sketch)



Our algorithm is **optimal**

1. Reduce showing that our algorithm picks the optimal path to showing our algorithm calculates budget optimally

# Proof of Optimality (Sketch)



Our algorithm is **optimal**

1. Reduce showing that our algorithm picks the optimal path to showing our algorithm calculates budget optimally
   - Suppose otherwise that our algorithm didn't pick path with lowest reward. This means we didn't calculate budget optimally for a path with lower reward. Thus, we will prove our algorithm picks set of corrupted edges optimally.

# Proof of Optimality (Sketch)

Our algorithm is **optimal**

1. Reduce showing that our algorithm picks the optimal path to showing our algorithm calculates budget optimally
   - Suppose otherwise that our algorithm didn't pick path with lowest reward. This means we didn't calculate budget optimally for a path with lower reward. Thus, we will prove our algorithm picks set of corrupted edges optimally.
2. Picking just one edge in each traversal is optimal.

# Proof of Optimality (Sketch)



Our algorithm is **optimal**

1. Reduce showing that our algorithm picks the optimal path to showing our algorithm calculates budget optimally
   - Suppose otherwise that our algorithm didn't pick path with lowest reward. This means we didn't calculate budget optimally for a path with lower reward. Thus, we will prove our algorithm picks set of corrupted edges optimally.
2. Picking just one edge in each traversal is optimal.
3. Our algorithm picks the edge that is optimal in every traversal.

# Proof of Optimality (Sketch)



Our algorithm is **optimal**

1. Reduce showing that our algorithm picks the optimal path to showing our algorithm calculates budget optimally
   - Suppose otherwise that our algorithm didn't pick path with lowest reward. This means we didn't calculate budget optimally for a path with lower reward. Thus, we will prove our algorithm picks set of corrupted edges optimally.
2. Picking just one edge in each traversal is optimal.
3. Our algorithm picks the edge that is optimal in every traversal.
   - Suppose otherwise that there exists an edge set to corrupt that is more optimal. Consider edges that differ from algorithm's set to optimal set.

# Proof of Optimality (Sketch)



Our algorithm is **optimal**

1. Reduce showing that our algorithm picks the optimal path to showing our algorithm calculates budget optimally
   - Suppose otherwise that our algorithm didn't pick path with lowest reward. This means we didn't calculate budget optimally for a path with lower reward. Thus, we will prove our algorithm picks set of corrupted edges optimally.
2. Picking just one edge in each traversal is optimal.
3. Our algorithm picks the edge that is optimal in every traversal.
   - Suppose otherwise that there exists an edge set to corrupt that is more optimal. Consider edges that differ from algorithm's set to optimal set.
   - These substitutions will not yield greater corruption since algorithm chooses edge on least number of paths, which guarantees the maximum amount.

# Adversarial Algorithm Against $\epsilon$-Greedy Victim



We have an adversarial strategy against a simple victim... now we consider a smart one!

# Adversarial Algorithm Against $\epsilon$-Greedy Victim



We have an adversarial strategy against a simple victim... now we consider a smart one!
What is the optimal strategy for an adversary against a victim with an $\epsilon$-greedy policy?

# Adversarial Algorithm Against $\epsilon$-Greedy Victim



We have an adversarial strategy against a simple victim... now we consider a smart one!

What is the optimal strategy for an adversary against a victim with an $\epsilon$-greedy policy?

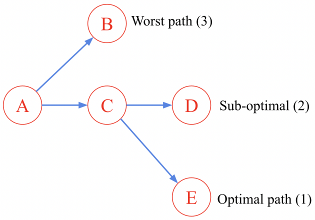
- Can't assume equal path traversal, sample complexity is tricky

# Adversarial Algorithm Against $\epsilon$-Greedy Victim

We have an adversarial strategy against a simple victim... now we consider a smart one!

What is the optimal strategy for an adversary against a victim with an $\epsilon$-greedy policy?

- Can't assume equal path traversal, sample complexity is tricky
- Perturbing edges not in the optimal path or path to be switched has an effect, especially for small budget

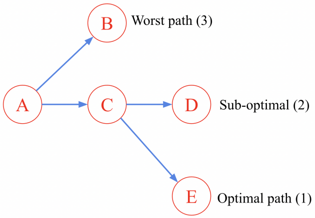


# Adversarial Algorithm Against $\epsilon$-Greedy Victim



We have an adversarial strategy against a simple victim... now we consider a smart one!
What is the optimal strategy for an adversary against a victim with an $\epsilon$-greedy policy?

- Can't assume equal path traversal, sample complexity is tricky
- Perturbing edges not in the optimal path or path to be switched has an effect, especially for small budget




- Chebyshev's Inequality bound on expected reward of this strategy: it is less than $(r_1 + r_3) \cdot (N_1 + N_3) p\delta^2 \frac{1-p}{(r_1 - r_3)^2}$

# Future Work



- How does victim defend against adversary strategy outlined above using Best-of-Both-Worlds?
  - Devise layering algorithm for victim defense
- More generally: set up minimax between victim and adversary to fully describe their behaviors in the MDP
  - What is the value of corrupting a path that is neither the optimal path nor the path we are trying to switch with it? Is there value in confusing the victim in this way? When is this helpful?

# Acknowledgements

We would like to thank...

- MIT PRIMES; Dr. Slava Gerovitch and Dr. Srini Devadas for this wonderful opportunity
- Mayuri Sridhar for being an amazing mentor
- You!

# References



[1] 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.

[2] 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.

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

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