Adversarial Attacks Against Online Learning Agents

MIT PRIMES, Mentor: Mayuri Sridhar

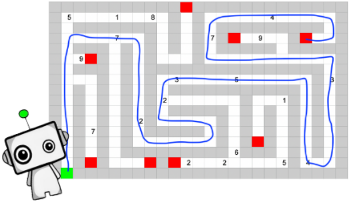
Alicia Li and Mati Yablon

MIT

May 14, 2023

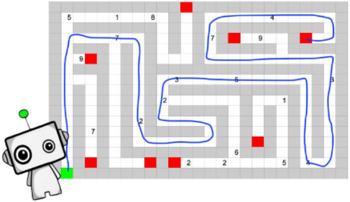
A Cute Robot in A Cute Maze

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



A Cute Robot in A Cute Maze

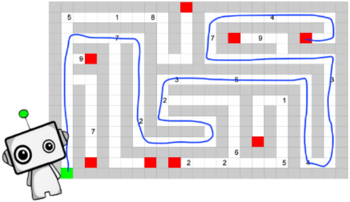
We (a cute robot) need to find the optimal path in this maze!



- Maze rewards are noisy

A Cute Robot in A Cute Maze

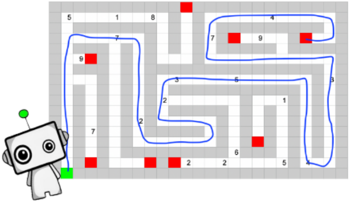
We (a cute robot) need to find the optimal path in this maze!



- Maze rewards are noisy
- We could run through each path a lot of times and average their rewards.

A Cute Robot in A Cute Maze

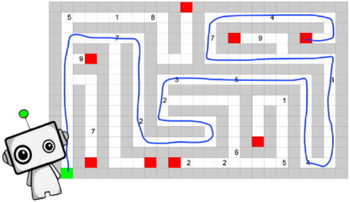
We (a cute robot) need to find the optimal path in this maze!



- Maze rewards are noisy
- We could run through each path a lot of times and average their rewards.
- Can we do better?

A Cute Robot in A Cute Maze

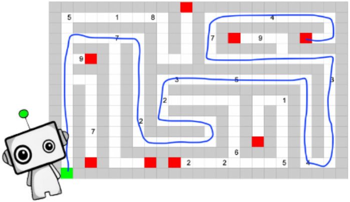
We (a cute robot) need to find the optimal path in this maze!



- Maze rewards are noisy
- We could run through each path a lot of times and average their rewards.
- Can we do better?
- Let's use Online Learning on Graphs!

A Cute Robot in A Cute Maze

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

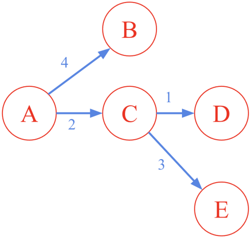


- Maze rewards are noisy
- We could run through each path a lot of times and average their rewards.
- Can we do better?
- Let's use Online Learning on Graphs!
- Other use cases: playing Atari games and robotic hand manipulation

Reward Estimation

Robot (alternatively **agent** or **victim**) navigates **graph**,

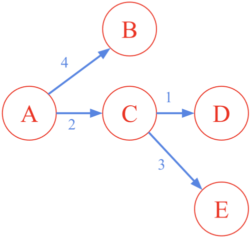
- Every node on the graph is state
- Every edge is action
- Every edge is weighted by some reward



Reward Estimation

Robot (alternatively **agent** or **victim**) navigates **graph**,

- Every node on the graph is state
- Every edge is action
- Every edge is weighted by some reward



Streaming setting: in each sample (path taken through graph), agent observes stream of data
Goal: find true edge weights, averaging observed values for each edge

Agent Sampling

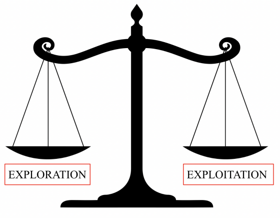
- Beginning phase is **Warm Start**: Agent samples a random path and traverses it.

Agent Sampling

- Beginning phase is **Warm Start**: Agent samples a random path and traverses it.
- Then **Adaptive Sampling** phase: Agent controls choices, can use strategies e.g. ϵ-greedy
  - Probability ϵ: sample random path
  - Probability $1 - \epsilon$: traverse path with highest perceived reward [2].

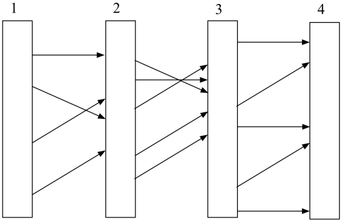
Agent Sampling

- Beginning phase is **Warm Start**: Agent samples a random path and traverses it.
- Then **Adaptive Sampling** phase: Agent controls choices, can use strategies e.g. ϵ-greedy
  - Probability ϵ: sample random path
  - Probability $1 - \epsilon$: traverse path with highest perceived reward [2].



Graph Properties

We consider DAGs (directed acyclic graphs)
Of these, we only consider layered graphs, for instance:

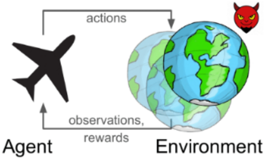


Attacks on Graphs

What if something perturbs our environment?

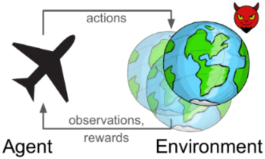
Attacks on Graphs

What if something perturbs our environment?



Attacks on Graphs

What if something perturbs our environment?

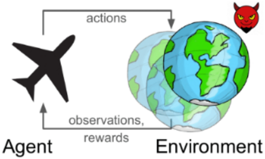


Motivation: performance can be degraded by:

- Human biases

Attacks on Graphs

What if something perturbs our environment?

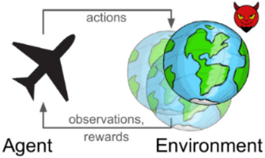


Motivation: performance can be degraded by:

- Human biases
- Modeling errors

Attacks on Graphs

What if something perturbs our environment?

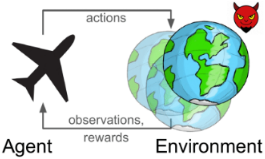


Motivation: performance can be degraded by:

- Human biases
- Modeling errors
- Actual adversaries

Attacks on Graphs

What if something perturbs our environment?



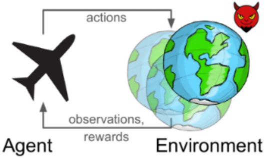
Motivation: performance can be degraded by:

- Human biases
- Modeling errors
- Actual adversaries

So *robustness* against perturbation is important!

Attacks on Graphs

What if something perturbs our environment?



Motivation: performance can be degraded by:

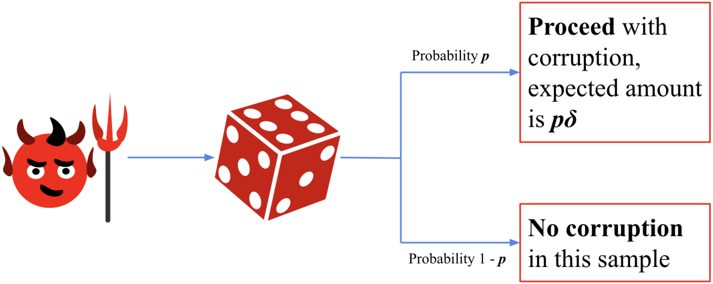
- Human biases
- Modeling errors
- Actual adversaries

So *robustness* against perturbation is important!
We study *training time attacks*.

Adversarial Setting

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 δ each

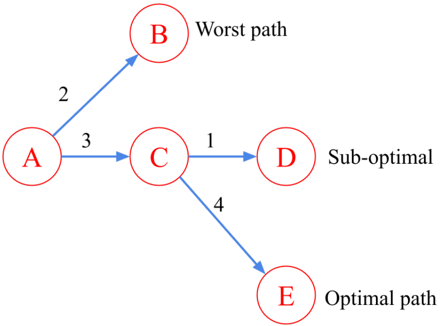


Naïve Adversarial Strategy

Adversary wants to make optimal path seem worse than some suboptimal path.

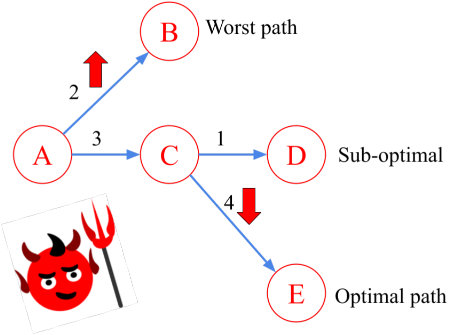
Naïve Adversarial Strategy

Adversary wants to make optimal path seem worse than some suboptimal path.
Consider the following Graph:



Naïve Adversarial Strategy

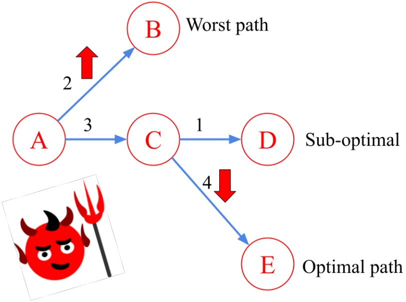
Adversary wants to make optimal path seem worse than some suboptimal path.
Consider the following Graph:



Naïve Approach: $2p\delta$

Naïve Adversarial Strategy Corruption

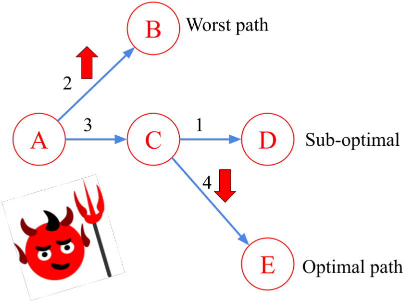
Naïve Approach: $2p\delta$



Effective corruption is $\frac{p\delta}{a_e}$ where a_e is the number of paths edge e is on.

Naïve Adversarial Strategy Corruption

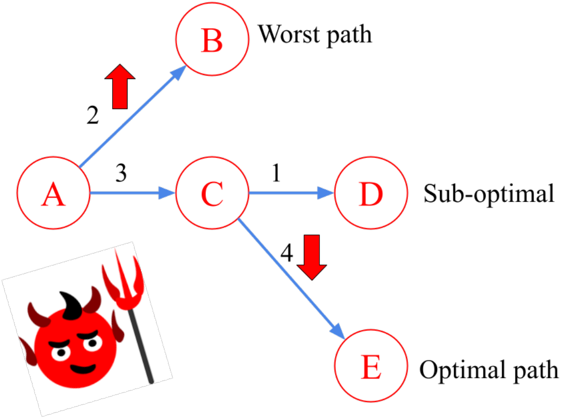
Naïve Approach: $2p\delta$



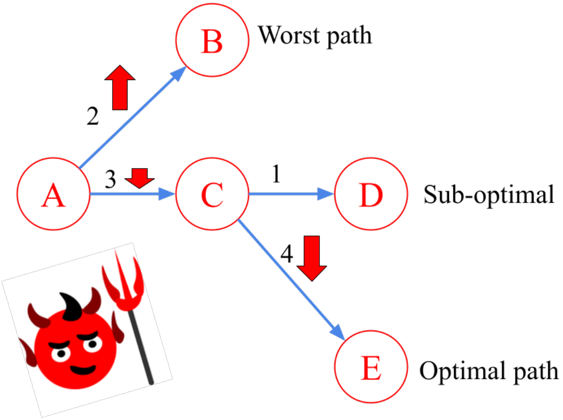
Effective corruption is $\frac{p\delta}{a_e}$ where a_e is the number of paths edge e is on.

Corrupt CE because it is traversed half as much as AC, doubling effective corruption

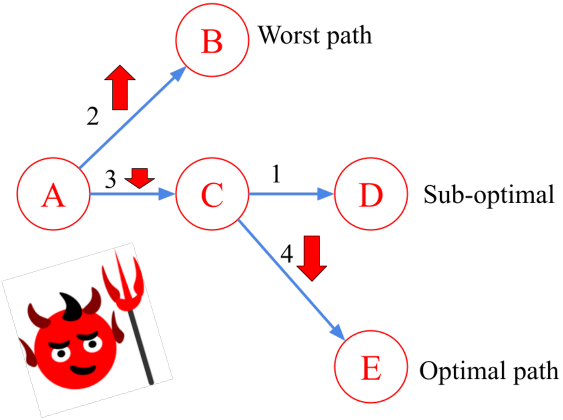
A More Optimal Adversarial Strategy



A More Optimal Adversarial Strategy



A More Optimal Adversarial Strategy



Our Approach: $2p\delta+$ extra $\frac{1}{2}p\delta$ of "free corruption"

Adversarial Algorithm 1

- Corrupt optimal path downwards as much as possible, maximizing free corruption

Adversarial Algorithm 1

- Corrupt optimal path downwards as much as possible, maximizing free corruption
- For every path, calculate the maximum amount the adversary can corrupt this path upwards

Adversarial Algorithm 1

- Corrupt optimal path downwards as much as possible, maximizing free corruption
- For every path, calculate the maximum amount the adversary can corrupt this path upwards
- Check if there is enough corruption to switch with optimal path

Adversarial Algorithm 1

- Corrupt optimal path downwards as much as possible, maximizing free corruption
- For every path, calculate the maximum amount the adversary can corrupt this path upwards
- Check if there is enough corruption to switch with optimal path
- Return the path with smallest reward that can be switched

Adversarial Algorithm 1

- Corrupt optimal path downwards as much as possible, maximizing free corruption
- For every path, calculate the maximum amount the adversary can corrupt this path upwards
- Check if there is enough corruption to switch with optimal path
- Return the path with smallest reward that can be switched

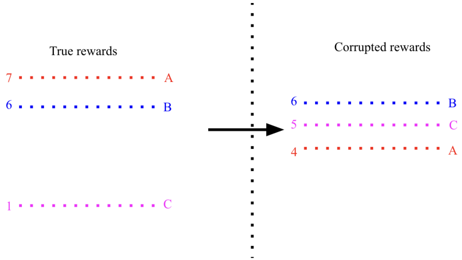
Proved optimality for a naive setting

Issues with Algorithm 1

Is not always optimal when victim samples each path equally. Why?

Issues with Algorithm 1

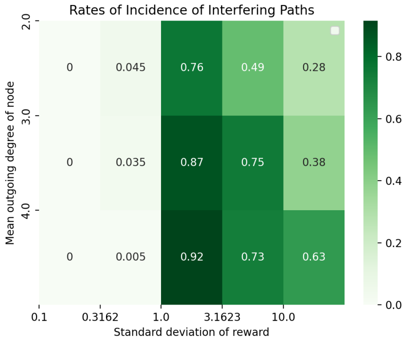
Is not always optimal when victim samples each path equally.
Why?
Because of *interfering paths*



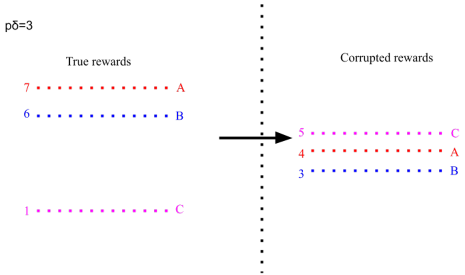
Even if we switch a low-reward path (C) with the optimal one (A), there still may be other paths (B, an interfering path) which initially were in between, but are now viewed as optimal!

Characterizing Occurrence of Interfering Paths

Graphs randomly and automatically generated, 4-layer graph used, mean 6 nodes per layer, $p\delta = 1$

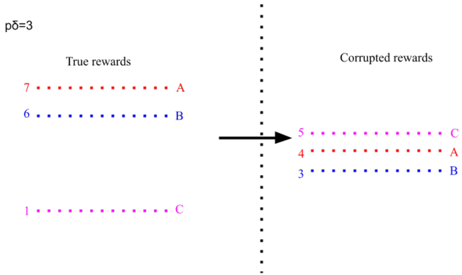


Heuristic For Interfering Paths



Heuristic For Interfering Paths:

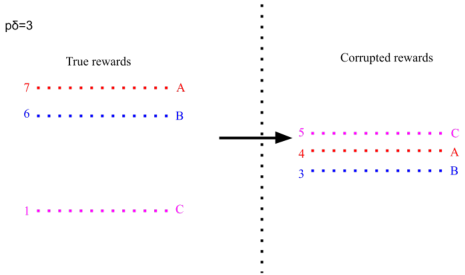
Heuristic For Interfering Paths



Heuristic For Interfering Paths:

- Corrupt path optimal path (A) downwards as much as possible

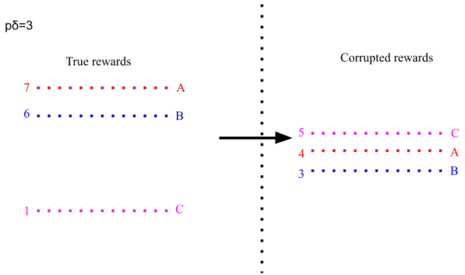
Heuristic For Interfering Paths



Heuristic For Interfering Paths:

- Corrupt path optimal path (A) downwards as much as possible
- Corrupt interfering path (B) downwards as much as possible

Heuristic For Interfering Paths

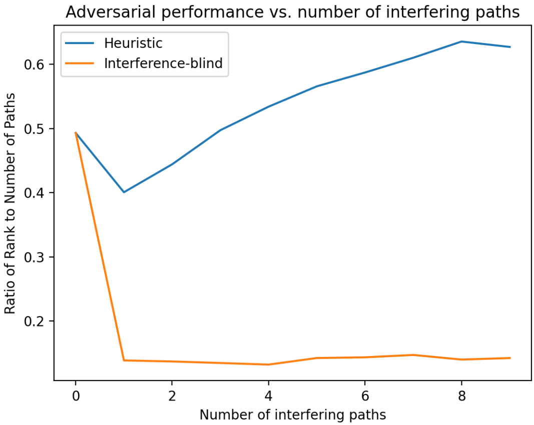


Heuristic For Interfering Paths:

- Corrupt path optimal path (A) downwards as much as possible
- Corrupt interfering path (B) downwards as much as possible
- Upwards corruption on the lowest possible reward path C the victim will choose

Comparison of Both Algorithms' Performance

Higher rank $\rightarrow$ lower reward



Degradation of Corruption

- Let's consider an ϵ-greedy sampling victim

Degradation of Corruption

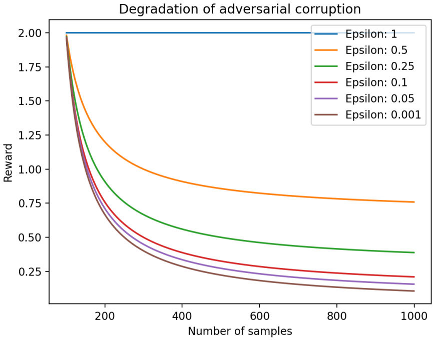
- Let's consider an ϵ-greedy sampling victim
- Path viewed as optimal is now sampled more often

Degradation of Corruption

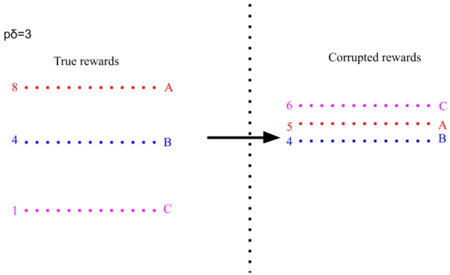
- Let's consider an ϵ-greedy sampling victim
- Path viewed as optimal is now sampled more often
- But adversary can only corrupt $p\delta$ per traversal

Degradation of Corruption

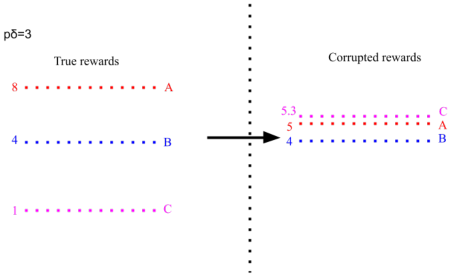
- Let's consider an ϵ-greedy sampling victim
- Path viewed as optimal is now sampled more often
- But adversary can only corrupt $p\delta$ per traversal
- Free corruption on optimally perceived path degrades



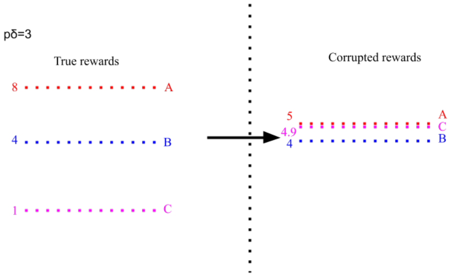
Degradation of Free Corruption Paths



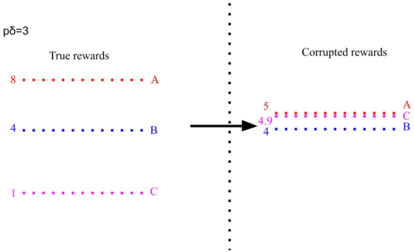
Degradation of Free Corruption Paths



Degradation of Free Corruption Paths

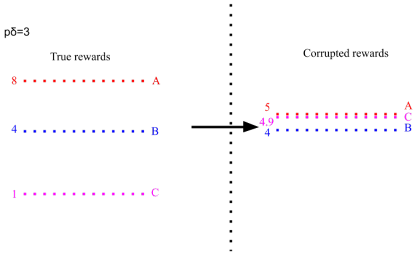


Degradation of Free Corruption Paths



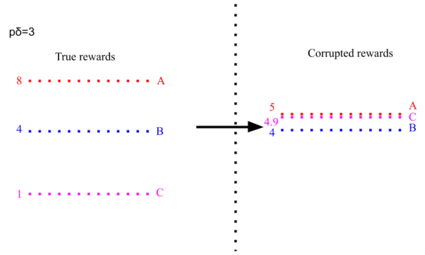
- Adversary doesn't want corruption on C to degrade

Degradation of Free Corruption Paths



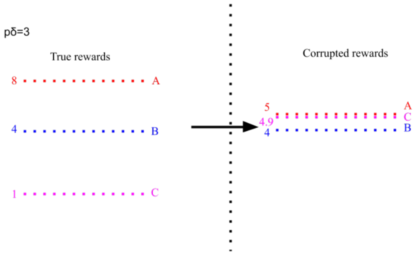
- Adversary doesn't want corruption on C to degrade
- Ensure that C is not sampled greedily

Degradation of Free Corruption Paths



- Adversary doesn't want corruption on C to degrade
- Ensure that C is not sampled greedily
- Instead, perturb a *stable* path to have highest perceived reward

Degradation of Free Corruption Paths

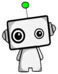


- Adversary doesn't want corruption on C to degrade
- Ensure that C is not sampled greedily
- Instead, perturb a *stable* path to have highest perceived reward

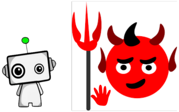
Definition

Stable Path: a path that corrupted no more than $p\delta$.
Corruption on this path can always be maintained.

Stalling Heuristic

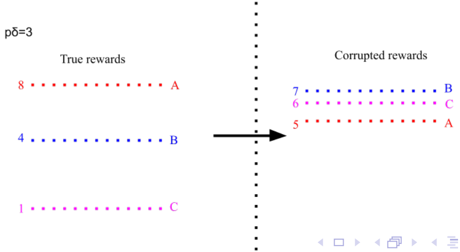
Stalling Heuristic



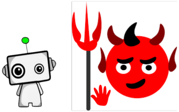
Corrupt A and C as before

Corrupt *stable path* B upwards as an intermediate step

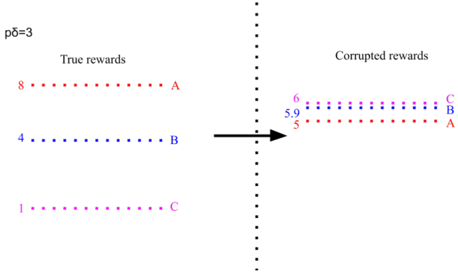
Adversary can maintain B indefinitely



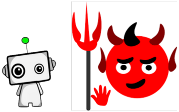
Stalling Heuristic



Near the end of learning, corrupt B downwards so victim chooses C. Reward of C does not degrade.

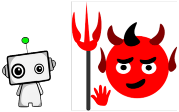


Stalling Analysis



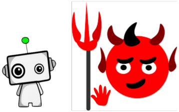
- Using stable paths can increase adversarial budget when $B \cap C$ is corrupted each time B is traversed

Stalling Analysis



- Using stable paths can increase adversarial budget when $B \cap C$ is corrupted each time B is traversed
- The fraction of times $B \cap C$ is corrupted increases from warm start, increasing effective corruption

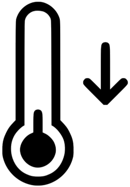
Stalling Analysis



- Using stable paths can increase adversarial budget when $B \cap C$ is corrupted each time B is traversed
- The fraction of times $B \cap C$ is corrupted increases from warm start, increasing effective corruption
- Stalling with multiple stable paths is likely optimal

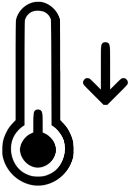
Advanced Victim Strategies

- ϵ-*annealing* decreases ϵ over time, natural decline in exploration



Advanced Victim Strategies

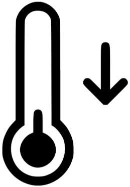
- ϵ-*annealing* decreases ϵ over time, natural decline in exploration



- Randomized ϵ-annealing may be robust to Stalling Heuristic

Advanced Victim Strategies

- ϵ-*annealing* decreases ϵ over time, natural decline in exploration



- Randomized ϵ-annealing may be robust to Stalling Heuristic
- If adversary can predict ϵ, it knows when to start switching to final path C

Future Work

- Further flesh out behavior of victim beyond simplistic sampling strategies; e.g. epsilon annealing
- Make approximations more reliable and efficient; too much looping even in heuristic strategy
- Provide more rigorous characterization of interference paths

References

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

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

[3] 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.

Acknowledgements

We would like to thank...

- MIT PRIMES; Dr. Slava Gerovitch, Dr. Srini Devadas, Dr. Tanya Khovanova, Dr. Pavel Etingof, and Mr. Dixon for this wonderful opportunity
- Mayuri Sridhar for being an amazing mentor!
- You!