# Optimal Edge Weight Perturbations to Attack Shortest Paths

**Benjamin A. Miller**[1], **Zohair Shafi**[1], **Wheeler Ruml**[2], **Yevgeniy Vorobeychik**[3], **Tina Eliassi-Rad**[1] and **Scott Alfeld**[4]
[1]Northeastern University, Boston, MA, USA
[2]University of New Hampshire, Durham, NH, USA
[3]Washington University in St. Louis, St. Louis, MO, USA
[4]Amherst College, Amherst, MA, USA
{miller.be, shafi.z, t.eliassirad}@northeastern.edu, ruml@cs.unh.edu, yvorobeychik@wustl.edu, salfeld@amherst.edu

## Abstract

Finding shortest paths in a given network (e.g., a computer network or a road network) is a well-studied task with many applications. We consider this task under the presence of an adversary, who can manipulate the network by perturbing its edge weights to gain an advantage over others. Specifically, we introduce the *Force Path Problem* as follows. Given a network, the adversary's goal is to make a specific path the shortest by adding weights to edges in the network. The version of this problem in which the adversary can cut edges is NP-complete. However, we show that Force Path can be solved to within arbitrary numerical precision in polynomial time. We propose the `PATHPERTURB` algorithm, which uses constraint generation to build a set of constraints that require paths other than the adversary's target to be sufficiently long. Across a highly varied set of synthetic and real networks, we show that the optimal solution often reduces the required perturbation budget by about half when compared to a greedy baseline method.

## 1 Introduction

The shortest path problem is a seminal task in graph theory with numerous real-world applications in computer networks, transportation networks, etc. Given two nodes in a network, the shortest path between them is the set of edges that connects the two nodes with the minimum sum of edge weights. For a given network, two nodes can have more than one shortest path; and the network can be directed or undirected. Here we only consider undirected networks.

In this paper, we present the *Force Path Problem*, where there is a specific path that the adversary wants to be the shortest path between a pair of source and destination nodes. The adversary can increase weights of edges and has a fixed budget with which to achieve this goal. The *Force Path Problem* is similar to the *Force Path Cut Problem* [Miller *et al.*, 2021]. The difference is in the attack vector: in this case the adversary makes edges more expensive (by increasing edge weights) rather than removing edges. This difference may seem relatively minor, but it has a profound implication for the computational complexity of the problem. While Force Path Cut is NP-complete, we show in this paper that Force Path can be solved within arbitrary precision in polynomial time. We demonstrate that Force Path can be formulated as a linear program with a constraint set that is potentially factorial in the number of nodes. There is, however, a natural polynomial-time oracle to find violated constraints in any candidate solution (which we use to include a subset of constraints as necessary). We propose the `PATHPERTURB` algorithm that uses the oracle to iteratively refine the graph perturbations until the target path is the shortest.[1]

The main contributions of the paper are as follows:

- We formally define the Force Path problem: an adversarial attack on shortest paths.
- We formulate an oracle to identify the most violated constraint at any given point, the existence of which implies that Force Path can be optimized within arbitrary precision in polynomial time.
- We propose the `PATHPERTURB` algorithm, which uses the oracle to minimize the required perturbation budget.
- We present the results of experiments on synthetic and real networks, in which `PATHPERTURB` reliably reduces the required perturbation budget compared to a greedy baseline method.

## 2 Force Path Problem Definition

Consider a graph $G = (V, E)$, with a set of nodes $V$ and edges $E$, where $|V| = N$ and $|E| = M$. The edges in $E$ are undirected and have nonnegative weights $w : E \to \mathbb{R}_{\geq 0}$. The edge weights denote distances (a.k.a. lengths) between the adjacent nodes. In addition to the weighted graph, we are given a pair of source and destination nodes $s, t \in V$. The adversary's goal is to make a specific path, $p^*$, the shortest path from $s$ to $t$ in $G$. The adversary can achieve this by arbitrarily increasing the weight of any edge in $G$, all of which are visible to him/her. Within a budget constraint $b$, the adversary increases edge weights to obtain new weights $w'$ such that $\sum_{e \in E} (w'(e) - w(e)) \leq b$ and $p^*$ is the (possibly exclusive) shortest path from $s$ to $t$.

[1]We use the terms graph and network interchangeably.

In the next section, we will show that the Force Path Problem can be formulated as a linear program (LP), which implies a polynomial time algorithm to get a solution within any specified precision, despite a very large number of constraints.

## 3 Force Path LP Formulation

Let $\mathbf{w} \in \mathbb{R}^M_{\geq 0}$ be a vector of edge weights in the original graph $G$ and $\Delta \in \mathbb{R}^M_{\geq 0}$ be a vector of edge-weight perturbations. For any path $p$ from $s$ to $t$ in $G$, let $\mathbf{x}_p \in \{0,1\}^M$ be an edge indicator vector for $p$: the entries for $\mathbf{x}_p$ associated with the edges that comprise $p$ are 1, while all other entries are 0. Thus, $\mathbf{w}^\top \mathbf{x}_p$ is the length of $p$ in the original graph and $(\mathbf{w}+\Delta)^\top \mathbf{x}_p$ is its length in the perturbed graph.

The linear program formulation of Force Path is based on two key observations. First, any path $p$ that is not longer than $p^*$ must be perturbed to be longer than $p^*$. This is clear from the fact that, if it were not the case, $p^*$ would not be the shortest path. Second, we do not perturb $p^*$, formalized as follows:

**Proposition 1.** *The minimum-budget solution to Force Path includes no perturbation of any edge along $p^*$.*

*Proof.* Let $\hat{\Delta}$ be the minimum-budget solution. Suppose that the claim is not true—i.e., there is an edge $e$, which is part of $p^*$, where $\hat{\Delta}(e) > 0$. Since $\hat{\Delta}$ is the solution to Force Path, $p^*$ is the shortest path when $\hat{\Delta}$ is added to the edge weights. Let $\Delta'$ be the same vector with the entry at $e$ reduced to 0, i.e., $\Delta'(e') = \hat{\Delta}(e')$ for $e' \in E \setminus \{e\}$ and $\Delta'(e) = 0$. Since $e$ is part of $p^*$, $\mathbf{x}_{p^*}^\top \hat{\Delta} - \mathbf{x}_{p^*}^\top \Delta' = \hat{\Delta}(e)$. For any path $p$, since $\mathbf{x}_p$ consists of only zeros and ones, we have

$$\mathbf{x}_p^\top \hat{\Delta} - \mathbf{x}_p^\top \Delta' \leq \hat{\Delta}(e). \tag{1}$$

Again, $p^*$ must be the shortest path using $\hat{\Delta}$, and, therefore, for any path $p$ from $s$ to $t$, $\mathbf{x}_p^\top \hat{\Delta} \geq \mathbf{x}_{p^*}^\top \Delta'$. Combining this with (1), we have, for any path $p$

$$\mathbf{x}_{p^*}^\top \Delta' = \mathbf{x}_{p^*}^\top \hat{\Delta} - \hat{\Delta}(e) \leq \mathbf{x}_{p^*}^\top \hat{\Delta} - \hat{\Delta}(e) \leq \mathbf{x}_p^\top \Delta'. \tag{2}$$

Thus, if $\hat{\Delta}$ were replaced with $\Delta'$, $p^*$ would still be the shortest path. This implies that the total perturbation could be reduced by $\hat{\Delta}(e) > 0$ and still achieve the objective, so $\hat{\Delta}$ is not the minimum-budget solution. Thus, we have a contradiction, and the proof is complete. □

The implication of this observation is that there is a fixed lower bound for the lengths of all paths. Let $\ell = \mathbf{w}^\top \mathbf{x}_{p^*}$ be the length of $p^*$ and $P_\ell$ be the set of paths from $s$ to $t$ whose length is less than or equal to $\ell$. Finally, let $\delta$ be the "buffer" we use to ensure $p^*$ is the unique shortest path: the difference between the length of $p^*$ and the length of the second shortest path[2]. We formulate the linear program as follows:

$$\hat{\Delta} = \arg\min_{\Delta} \mathbf{1}^\top \Delta \tag{3}$$

$$\text{s.t. } \Delta_i \geq 0, \;\; 1 \leq i \leq M \tag{4}$$

$$(\mathbf{w}+\Delta)^\top \mathbf{x}_p \geq \ell + \delta, \;\; \forall p \in P_{\ell+\delta} \setminus \{p^*\} \tag{5}$$

$$\mathbf{x}_{p^*}^\top \Delta = 0. \tag{6}$$

As with the approximate version of Force Path Cut discussed in [Miller *et al.*, 2021], the number of paths in $P_{\ell+\delta}$ may be too large to enumerate all constraints. In an $N$-node clique, for example, the number of paths of length $N-1$ between any two nodes is $(N-2)!$. Thus, specifying all constraints in the linear program is computationally intractable. We use constraint generation to iteratively incorporate constraints as they are needed (see, e.g., [Ben-Ameur and Neto, 2006; Letchford and Vorobeychik, 2013]). In order to use constraint generation, however, there must be an oracle that returns a constraint being violated at a given point.

There is, fortunately, a natural oracle for the constraints specified in (5), which not only returns a violated constraint, but the constraint *most* violated at the proposed solution. We find this constraint as follows. The candidate solution is a perturbation to the edge weights, $\hat{\Delta}$. We apply the perturbation to get the new edge weights $w'$, where

$$w'(e) = w(e) + \hat{\Delta}(e). \tag{7}$$

Using $w'$ as distances, we find the shortest path $p$ from $s$ to $t$ in $G$. If $p$ is $p^*$, we find the second shortest path if it exists. If there is no such path, there is no violated constraint. If there is, we let $p$ be this path. If $p$ is at least $\delta$ longer than $p^*$, there is no violated constraint. If not, the length of $p$ needs to be incorporated as a constraint. Algorithm 1 provides the pseudocode for this procedure.

**Algorithm 1** ConstraintOracle

**Input**: graph $G$, weights $w$, target path $p^*$, buffer $\delta$
**Output**: A path $p$ from $s$ to $t$ in $G$

```
s ← first node in p*
t ← last node in p*
p ← shortest path from s to t in G
if p is p* then
    p ← second shortest path from s to t in G
    ⟨⟨p will be ∅ with length ∞ if p* is the only path⟩⟩
end if
if length(p) ≥ length(p*) + δ then
    p ← ∅
end if
return p
```

While the number of constraints may be extremely large, each one is a standard linear inequality constraint, which implies that the feasible region is convex. Since we have a

[2] In a scenario where being tied for shortest is acceptable, $\delta$ can be set to 0. If it is acceptable for $p^*$ to be shortest by any $\epsilon > 0$, we can set $\delta$ to 0 and distribute an arbitrarily small value across edges not on $p^*$. In this case, the budget must be strictly larger than the sum of the computed perturbations.

**Algorithm 2** PATHPERTURB

**Input**: graph $G = (V, E)$, weights $w$, target path $p^*$, buffer $\delta$
**Output**: perturbation vector $\hat{\Delta}$

$\ell \leftarrow$ length of $p^*$
$\mathbf{w} \leftarrow$ weight vector for $w$
$P_{\ell+\delta} \leftarrow \emptyset$
$M \leftarrow |E|$
$p \leftarrow p^*$
**repeat**
  $P_{\ell+\delta} \leftarrow P_{\ell+\delta} \cup \{p\}$
  $\hat{\Delta} \leftarrow$ solution to (3)–(6)
  $w' \leftarrow w + \hat{\Delta}$  $\langle\langle$as in (7)$\rangle\rangle$
  $p \leftarrow$ ConstraintOracle($G$, $w'$, $p^*$, $\delta$)
**until** $p$ is empty
**return** $\hat{\Delta}$

constraint oracle that runs in polynomial time,[3] this system can be optimized in polynomial time regardless of the number of constraints. Using the ellipsoid algorithm introduced by Khachiyan (see [Gács and Lovász, 1981]), we can solve a linear program within finite precision in a polynomial number of iterations [Grötschel *et al.*, 1981]. This results in the following proposition.

**Proposition 2.** *Force Path can be optimized within precision of any constant $\epsilon > 0$ in polynomial time.*

# 4 Proposed Method: PATHPERTURB

While the ellipsoid algorithm provably converges in polynomial time, it is considerably slower in practice than simplex methods. Thus, our proposed algorithm iteratively solves a linear optimization procedure, adding constraints via the oracle as necessary. We call this algorithm `PATHPERTURB`.

Our perturbation algorithm uses a linear program where each constraint is associated with a path from $s$ to $t$ that is not longer than $p^*$. Each path must have weights added to the edges so that the path's length will become sufficiently long. `PATHPERTURB` operates in an iterative fashion as it builds the set of necessary constraints. At each iteration, it finds a solution based on a subset of constraints, perturbs the weights based on this solution, and, if there is still a path from $s$ to $t$ that is shorter, it adds the corresponding constraint to the linear program. Algorithm 2 provides `PATHPERTURB`'s pseudocode.

# 5 Experiments

This section presents the baseline methods, the networks used in experiments, the experimental setup, and the results.

## 5.1 Baseline Methods

We compare `PATHPERTURB` to two simple greedy baseline algorithms. Each algorithm iteratively perturbs a single edge on the shortest path $p$ from $s$ to $t$ until $p^*$ is the shortest path (and, if the buffer $\delta$ is greater than 0, until the second shortest path is at least $\delta$ longer than $p^*$). The first baseline we consider, `GreedyFirst`, perturbs the first edge (in path traversal order) in $p$ that deviates from $p^*$. We also use a method in which we perturb the edge with the smallest weight of all edges that are in $p$ but not $p^*$. We refer to this baseline as `GreedyMin`. In both cases, the selected edge is perturbed enough to make the path at least $\delta$ longer than $p^*$—i.e., if $\mathbf{x}_p$ is the edge indicator vector for the current shortest path and $\ell' = (\mathbf{w} + \Delta)^\top \mathbf{x}_p$, the entry in $\Delta$ associated with the selected edge is increased by $\ell + \delta - \ell'$.

## 5.2 Synthetic and Real Networks

We ran `PATHPERTURB` and the baseline algorithms on both synthetic and real networks. All networks are undirected.

Our synthetic networks span a wide variety of topologies:

- Erdős–Rényi (ER) random networks with 16,000 nodes and an edge probability of 0.00125
- Barabási–Albert (BA) graphs with 16,000 nodes, where each new node connects to 10 existing nodes
- Watts–Strogatz (WS) graphs with 16,000 nodes, average degree 20, and edge rewiring probability $0.02$
- Stochastic Kronecker (KR) graphs with $2^{14}$ nodes and a density parameter of $0.0125$
- $285 \times 285$ two-dimensional lattice (LAT) networks
- 565-vertex complete (COMP) graphs

In all cases, we generate 100 networks from random (or fixed) network generators and add edge weights. Note that the number of edges is approximately 160,000 in all synthetic networks. We also consider various edge-weight distribution for each synthetic network:

- Option 1: Give all edges weight 1.
- Option 2: For each edge, draw a value from a Poisson distribution with rate parameter 20, and add 1 to get the weight.
- Option 3: Draw each weight from a uniform distribution over integers from 1 to 41.

In addition to synthetic graphs, we use the following real networks:

- Oregon autonomous systems (AS) [Leskovec *et al.*, 2005]
- Wikispeedia (WIKI) [West *et al.*, 2009]
- Pennsylvania roads (PA-ROAD) [Leskovec *et al.*, 2009]
- Northeast US roads (NEUS)[4]
- Central Chilean power grid (GRID) [Kim *et al.*, 2018]
- Lawrence Berkeley National Laboratory computer network traffic (LBL)[5]
- DBLP coauthorship graph (DBLP) [Benson *et al.*, 2018]

[3] Finding the two shortest simple paths between two nodes takes $O(NM)$ time using Yen's algorithm [Yen, 1971]. If $\delta = 0$ and edge weights are strictly positive, we can use an algorithm not restricted to simple paths that runs in $O(M + N \log N)$ time [Eppstein, 1998].

[4] Available at https://bit.ly/2QWcug9.

[5] Available at https://bit.ly/2PQbOsr.

Networks AS, WIKI, and PA-ROAD do not have weights on their edges, so we add weights similar to the synthetic networks. In LBL and DBLP, the weights represent similarities rather than distances—number of connections and number of coauthored papers, respectively—so we invert the weight for use in the shortest path computation. In these cases, we set $\delta$ to 0.1, while we use $\delta = 1$ in all other cases.

### 5.3 Experimental Setup

We run 100 trials for each network (or network generator) and each weighting scheme if applicable. In each experiment, we select $s$ and $t$ uniformly at random from the largest connected component of $G$, with the exception of LAT, PA-ROAD, and NEUS. In these grid-like graphs, computing the sequence of shortest simple paths is extremely time consuming, and we instead select $s$ at random and choose $t$ from among the nodes 50 hops away from $s$. We then compute the 100th, 200th, 400th, and 800th shortest paths from $s$ to $t$ and use these as $p^*$. For LAT, PA-ROAD, and NEUS, we compute these paths only using the induced subgraph of nodes that are at most 60 hops away from $s$.

We ran the experiments using a Linux cluster with 32 cores and 192 GB of memory per node. We implemented the linear program in `PATHPERTURB` using the Python interface to Gurobi 9.1.1, and sequential shortest paths were computed using `shortest_simple_paths` in NetworkX.[6]

### 5.4 Results

We treat the result of `GreedyFirst` as our baseline budget and report the value optimized by `PATHPERTURB` as a reduction from the baseline. With few exceptions, `GreedyFirst` outperforms `GreedyMin` in both running time and perturbation cost, so we omit the `GreedyMin` results for clarity of presentation. For each graph, we use the algorithms in an attempt to minimize the budget, after which the adversary would determine whether or not the attack is possible within the constraints. Figure 1 shows the results on the synthetic networks, while Figure 2 shows the results on real networks with synthetic edge weights; and Figure 3 shows the results on real weighted networks. The figures show the results where $p^*$ is the 800th shortest path. Due to space limitations, we omit the other results; they are substantially similar.

Across all experiments, we see a substantial improvement over `GreedyFirst` by using `PATHPERTURB`, for the most part ranging from a 25% reduction in the required perturbation budget to a decrease of more than a factor of two. This comes at the expense of increased running time: an increase of an order of magnitude appears typical. In the synthetic networks, `PATHPERTURB` provides a greater improvement for heterogeneous Kronecker (KRON) and BA graphs rather than Erdős–Rényi graphs. This could be an effect of hubs: nodes with high degree that tend to facilitate short paths may make it more difficult to obtain a low budget via a greedy procedure.

The main exception to the substantial budget improvement is cliques (COMP, blue in Figure 1) with Poisson weights. To understand this result, consider an unweighted clique, and note that the 800th shortest path is a 3-hop path. The optimal perturbation to make a particular 3-hop path the shortest is to perturb all edges adjacent to $s$ and $t$ except those on $p^*$. With weights drawn from a Poisson distribution, we get weights typically near the mean, which gives us a similar effect to the unweighted graph: 2-hop paths are all similar lengths, as are 3-hop paths. In this context, `GreedyFirst` identifies a near-optimal perturbation. Looking deeper into the data, we see that for path ranks of 100, 200, and 400, the required budgets using `GreedyFirst` and `PATHPERTURB` are exactly the same when all edge weights are equal.

We observe one major difference from the results with `PATHATTACK` [Miller *et al.*, 2021], where the adversary's goal is the same but his/her attack vector is to cut edges. We see more substantial gains over the greedy baseline in grid-like networks. In the lattice network and the road networks, `PATHATTACK` took substantially more time for very modest improvements in edge removal cost. Here, we see a relatively high computational burden in these grid-like networks—reliably an order of magnitude in the real datasets—but the budget reductions are among the best. In addition, we note that lattices with Poisson weights are one of the few cases where `GreedyMin` outperforms `GreedyFirst` (the other being cliques with Poisson weights), though the difference is small and does not explain the extent of the difference. This may be due to the difference in cost. In [Miller *et al.*, 2021], costs were proportional to the weights of removed edges, while in the present work the cost of perturbing a path will be smaller if the edge weights are larger. Thoroughly investigating this phenomenon is a subject for future work.

## 6 Related Work

This paper expands the work on adversarial graph analysis that was introduced recently. Examples include attacks against vertex classification [Zügner *et al.*, 2018; Zügner and Günnemann, 2019] and node embedding [Bojchevski and Günnemann, 2019], as well as community detection when an adversary does not want to be grouped with other individuals [Kegelmeyer *et al.*, 2018]. In [Miller *et al.*, 2021], an adversary cut edges to attack shortest path algorithms; here, an adversary adds edge weights.

Prior network science work on attacks against graphs was focused on attempts to disrupt infrastructure, e.g., disconnecting a power grid graph. In this area, it was shown that graphs with heterogeneous degree distributions (like BA and KR graphs) are much more robust to random node removals, but highly susceptible to targeted attacks against the nodes with the most connections [Albert *et al.*, 2000].

There has been previous work on altering shortest paths, though it has been primarily focused on removal of a single edge or node. The objective of the "most vital edge" (or most vital node) problem is, given two nodes in a graph, to find the edge (node) whose removal most increases the shortest path between the source and destination [Nardelli *et al.*, 2001; Nardelli *et al.*, 2003]. In an adversarial context, this would be an instance of an adversary intending to divert the user from the best solution, rather than being motivated to push traffic along a particular path of interest.

[6] Gurobi is available at https://www.gurobi.com. NetworkX is available at https://networkx.org.

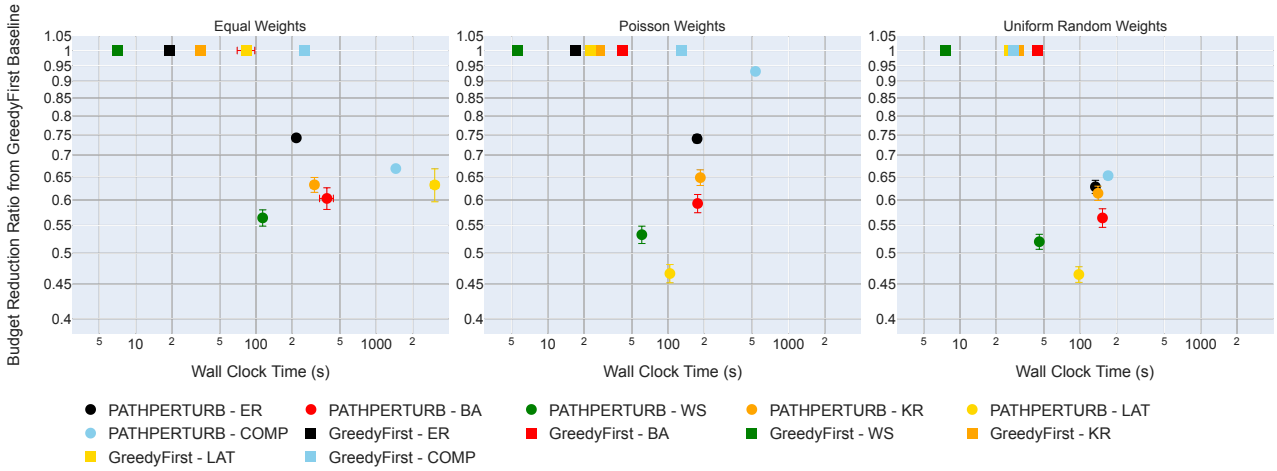


Figure 1: Results on synthetic networks. Results are shown using `PATHPERTURB` (○) and `GreedyFirst` (□), and each color represents a different network. The plots present results when the weights are equal (left), when they are drawn from a Poisson distribution (center), and drawn from a uniform distribution (right). Each plot shows the required budget as a proportion of the budget required using `GreedyFirst` (vertical axis) with respect to wall clock running time (horizontal axis). Lower cost reduction ratio and lower wall clock time (toward the lower left) is better. Error bars represent standard errors. In nearly all cases, `PATHPERTURB` yields a substantial cost reduction for its additional running time, though cliques with Poisson weights are nearly optimized with the baseline. Note: the black square (`GreedyFirst` on ER) in the right-hand plot is obscured by the yellow square.

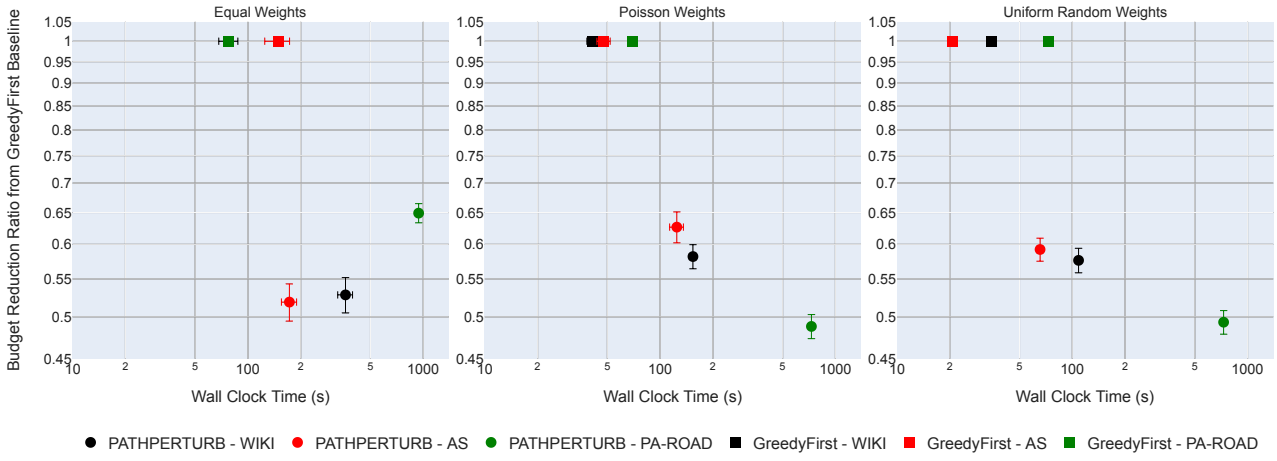


Figure 2: Results on unweighted real networks. Results are shown using `PATHPERTURB` (○) and `GreedyFirst` (□), and each color represents a different network. The plots present results when the weights are equal (left), when they are drawn from a Poisson distribution (center), and drawn from a uniform distribution (right). Each plot shows the required budget as a proportion of the budget required using `GreedyFirst` (vertical axis) with respect to wall clock running time (horizontal axis). Lower cost reduction ratio and lower wall clock time (toward the lower left) is better. Error bars represent standard errors. As with the synthetic networks, `PATHPERTURB` provides a significant cost reduction in all networks, though in this case we see a greater increase in running time for PA-ROAD. Note: the black square (`GreedyFirst` on WIKI) in the left-hand plot is obscured by the green square.

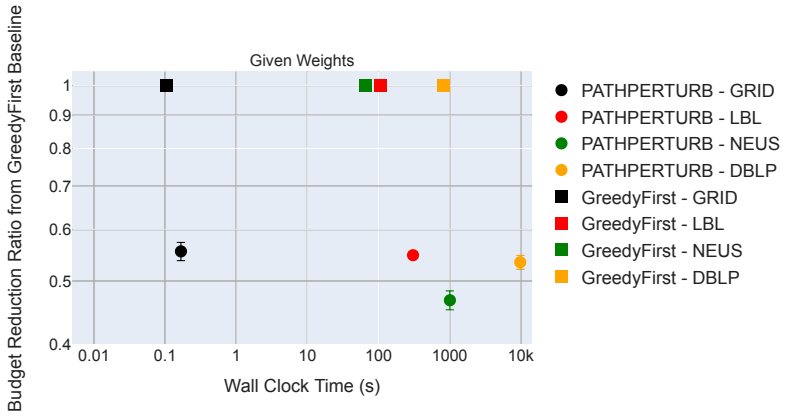


Figure 3: Results on weighted real networks. Results are shown using `PATHPERTURB` (○) and `GreedyFirst` (□), and each color represents a different network. The plot shows the required budget as a proportion of the budget required using `GreedyFirst` (vertical axis) with respect to wall clock running time (horizontal axis). Lower cost reduction ratio and lower wall clock time (toward the lower left) is better. Error bars represent standard errors. In all cases, `PATHPERTURB` reduces the cost of attacking the graph by about a factor of two.

In that sense, the most vital edge and node problem is similar to recent work on Stackelberg planning [Speicher *et al.*, 2018]. In this work, like in the most valuable node and edge problems, the goal of the attacker is to make it as costly as possible for the user to perform the task. While the most valuable edge only allows one move, the Stackelberg planning work uses a turn-based leader-follower framework, where the leader makes the follower's actions more costly at each step. A similar problem is the adversarial shortest path problem, where the state space has uncertainty that an adversary could exploit to decrease the user's reward as states are traversed [Neu *et al.*, 2012].

There are two complementary areas where path finding in an adversarial context is crucial. One is network interdiction, in which an adversary is attempting to traverse a network undetected [Washburn and Wood, 1995]. The other involves planning paths through hostile territory; for example, an unmanned aerial vehicle in enemy air space [Jun and D'Andrea, 2003]. Recent path interdiction work has focused on attack disruption [Letchford and Vorobeychik, 2013]. Work in this area also uses oracles to judiciously select from an extremely large set of potential strategies [Jain *et al.*, 2011].

## 7 Conclusions

We defined the Force Path Problem, in which an adversary adds weights to edges in order to make a particular path the shortest between a pair of source and destination nodes. The adversary has a budget, which he/she cannot exceed. We showed that Force Path can be optimized to within an arbitrarily small error in polynomial time. We demonstrated that Force Path can be formulated as a linear program with an intractable number of constraints. However, standard shortest-path algorithms can be used to obtain a polynomial-time constraint oracle, which allowed us to use constraint generation. We formalized this procedure in our `PATHPERTURB` algorithm, which we applied to a diverse collection of real and simulated data. We observed that the perturbation budget optimized using `PATHPERTURB` is often as little as half of what can be obtained using a greedy baseline perturbation procedure.

## Acknowledgments

This material is based upon work supported by the United States Air Force under Air Force Contract No. FA8702-15-D-0001 and the Combat Capabilities Development Command Army Research Laboratory (under Cooperative Agreement Number W911NF-13-2-0045). Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Air Force or Army Research Laboratory.

## References

[Albert *et al.*, 2000] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

[Ben-Ameur and Neto, 2006] Walid Ben-Ameur and José Neto. A constraint generation algorithm for large scale linear programs using multiple-points separation. *Mathematical Programming*, 107(3):517–537, 2006.

[Benson *et al.*, 2018] Austin R Benson, Rediet Abebe, Michael T Schaub, Ali Jadbabaie, and Jon Kleinberg. Simplicial closure and higher-order link prediction. *Proc. Nat. Acad. Sci.*, 115(48):E11221–E11230, 2018.

[Bojchevski and Günnemann, 2019] Aleksandar Bojchevski and Stephan Günnemann. Adversarial attacks on node embeddings via graph poisoning. In *ICML*, pages 695–704, 2019.

[Eppstein, 1998] David Eppstein. Finding the $k$ shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[Gács and Lovász, 1981] Peter Gács and Laszlo Lovász. Khachiyan's algorithm for linear programming. In *Mathematical Programming at Oberwolfach*, pages 61–68. Springer, 1981.

[Grötschel *et al.*, 1981] Martin Grötschel, László Lovász, and Alexander Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[Jain *et al.*, 2011] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, pages 327–334, 2011.

[Jun and D'Andrea, 2003] Myungsoo Jun and Raffaello D'Andrea. Path planning for unmanned aerial vehicles in uncertain and adversarial environments. In *Cooperative Control: Models, Applications and Algorithms*, pages 95–110. Springer, 2003.

[Kegelmeyer *et al.*, 2018] W. Philip Kegelmeyer, Jeremy D Wendt, and Ali Pinar. An example of counter-adversarial community detection analysis. Technical Report SAND2018-12068, Sandia National Laboratories, 2018.

[Kim *et al.*, 2018] Heetae Kim, David Olave-Rojas, Eduardo Álvarez-Miranda, and Seung-Woo Son. In-depth data on the network structure and hourly activity of the central Chilean power grid. *Sci. Data*, 5(1):1–10, 2018.

[Leskovec *et al.*, 2005] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *KDD*, pages 177–187, 2005.

[Leskovec *et al.*, 2009] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[Letchford and Vorobeychik, 2013] Joshua Letchford and Yevgeniy Vorobeychik. Optimal interdiction of attack plans. In *AAMAS*, page 199–206, 2013.

[Miller *et al.*, 2021] Benjamin A. Miller, Zohair Shafi, Wheeler Ruml, Yevgeniy Vorobeychik, Tina Eliassi-Rad, and Scott Alfeld. PATHATTACK: Attacking shortest paths in complex networks. *arXiv preprint arXiv:2104.03761*, 2021.

[Nardelli *et al.*, 2001] Enrico Nardelli, Guido Proietti, and Peter Widmayer. A faster computation of the most vital edge of a shortest path. *Information Processing Letters*, 79(2):81–85, 2001.

[Nardelli *et al.*, 2003] Enrico Nardelli, Guido Proietti, and Peter Widmayer. Finding the most vital node of a shortest path. *Theoretical Computer Science*, 296(1):167–177, 2003.

[Neu *et al.*, 2012] Gergely Neu, Andras Gyorgy, and Csaba Szepesvari. The adversarial stochastic shortest path problem with unknown transition probabilities. In *AISTATS*, pages 805–813, 2012.

[Speicher *et al.*, 2018] Patrick Speicher, Marcel Steinmetz, Michael Backes, Jörg Hoffmann, and Robert Künnemann. Stackelberg planning: Towards effective leader-follower state space search. In *AAAI*, pages 6286–6293, 2018.

[Washburn and Wood, 1995] Alan Washburn and Kevin Wood. Two-person zero-sum games for network interdiction. *Operations Research*, 43(2):243–251, 1995.

[West *et al.*, 2009] Robert West, Joelle Pineau, and Doina Precup. Wikispeedia: An online game for inferring semantic distances between concepts. In *IJCAI*, 2009.

[Yen, 1971] Jin Y Yen. Finding the $K$ shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[Zügner and Günnemann, 2019] Daniel Zügner and Stephan Günnemann. Certifiable robustness and robust training for graph convolutional networks. In *KDD*, pages 246–256, 2019.

[Zügner *et al.*, 2018] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *KDD*, pages 2847–2856, 2018.