Adaptive Wizard for Removing Cross-Tier Misconfigurations in Active Directory

Huy Q. Ngo[1] , **Mingyu Guo**[1] , **Hung X. Nguyen**[1]
[1]The University of Adelaide
{quanghuy.ngo, mingyu.guo, hung.nguyen}@adelaide.edu.au,

Abstract

Security vulnerabilities in Windows Active Directory (AD) systems are typically modeled using an attack graph and hardening AD systems involves an iterative workflow: security teams propose an edge to remove, and IT operations teams manually review these fixes before implementing the removal. As verification requires significant manual effort, we formulate an Adaptive Path Removal Problem to minimize the number of steps in this iterative removal process. In our model, a wizard proposes an attack path in each step and presents it as a set of multiple-choice options to the IT admin. The IT admin then selects one edge from the proposed set to remove. This process continues until the target t is disconnected from source s or the number of proposed paths reaches B. The model aims to optimize the human effort by minimizing the expected number of interactions between the IT admin and the security wizard. We first prove that the problem is $\#\mathcal{P}$-hard. We then propose a set of solutions including an exact algorithm, an approximate algorithm, and several scalable heuristics. Our best heuristic, called DPR, can operate effectively on larger-scale graphs compared to the exact algorithm and consistently outperforms the approximate algorithm across all graphs. We verify the effectiveness of our algorithms on several synthetic AD graphs and an AD attack graph collected from a real organization.

1 Introduction

We propose the Adaptive Path Removal Problem, a model motivated by the challenge of eliminating attack paths in cybersecurity. We begin by describing the cybersecurity use case that motivates our approach and by explaining the design rationale behind our model. The main contributions of this paper are the introduction of a novel theoretical model and the exploration of scalable algorithms for solving this problem. Our model's design rationale is heavily influenced by practical cybersecurity scenarios and by the urgent demand for workable solutions from security teams.

Windows Active Directory (AD) is Microsoft's directory service that enables IT administrators to manage security permissions and control accesses across Windows domain networks. An AD environment is naturally described as a graph where nodes are accounts/computers/groups, and the directed edges represent accesses/permissions/vulnerability. One of the main focus in this line of work is minimizing "attack paths"—routes an attacker might use to escalate privileges and move laterally within the network.

Existing security models [Guo *et al.*, 2023; Zhang *et al.*, 2024; Goel *et al.*, 2023] and commercial tools such as BloodHound [Robbins, 2023] reduce these paths by suggesting actionable fixes, typically presented as sets of edges to remove from the graph. Unfortunately, not every proposed fix (edge) is implementable. Some edges may appear redundant, but removing them could cause significant disruptions. Since removing edges equates to revoking permissions or accesses within the network, each fix must be approved and implemented by IT operations teams. This has been referred to in the literature as the "implementable fixes" problem [Dunagan *et al.*, 2009; Guo *et al.*, 2024]. In industry practice, network hardening workflow typically unfolds in two stages: the security team first proposes necessary fixes, and then IT operations team review those fixes before implementation. This practical constraint has motivated the development of adaptive security models, models that incorporate human feedback and are thus better suited to real-world usage.

In the same way a "proxy" in auction theory places bids on user's behalf, our proposed wizard model acts as a "proxy" security operator that guide the IT administrator through the attack path removal process. At every step, the wizard model proposes an attack path to remove. The IT admin view this as a multiple-choice list of edges and will require to choose one edge to remove. This process continues until all attack paths are eliminated, or until the number of proposals reaches a preset limit. The wizard's goal is to minimize the expected number of proposals. The wizard is adaptive, meaning it proposes subsequent edges to remove based on the IT admin's choices in previous steps. Unlike previous work [Guo *et al.*, 2024; Zheng *et al.*, 2011; Dunagan *et al.*, 2009], which modeled IT admin's decision as simply removing or retaining an edge (binary decision), without guaranteeing that all attack paths would be eliminated; our path-based proposal mechanism provides a cut-guarantee solution.

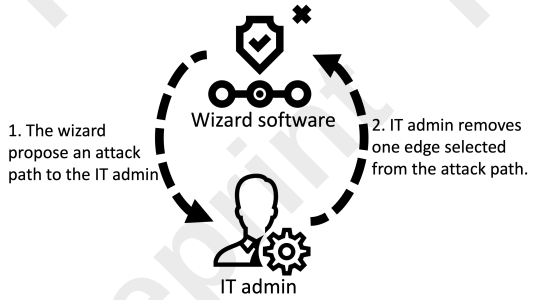

Figure 1: The wizard is a software step-by-step guide to assist the user in performing correction actions without requiring extensive technical knowledge.

Our key contributions can be summarized as follows:

- We introduce a new theoretical combinatorial optimization model called Adaptive Path Removal, motivated by the network security use case in AD systems. This is the first adaptive graph-focused model to incorporate path proposals and provide a cut-guarantee solution.
- We prove that the problem is $\#\mathcal{P}$-hard and introduce both an exact and an approximate algorithm.
- We develop a scalable heuristic called Dynamic Programming with Restriction (DPR), which builds on our exact and approximate algorithms. DPR achieves better scalability than the exact algorithm and outperforms the approximate algorithm.
- We also introduce several baseline methods, including two RL-based heuristics, and evaluate them on multiple synthetic graphs and a real AD network. Our experimental results show that DPR consistently achieves superior performance over all other methods.

2 Problem Formulation and Related Work

2.1 Problem Formulation

The Adaptive Path Removal (APR) problem can be formally defined as follows: Given a directed attack graph $G = (V, E)$ with a source s and a destination node t. Each edge $e \in E$ is associated with a confidence score, defined by a function $conf : E \mapsto [0, 1]$. Every round, the system will propose a simple path from the current attack graph. A simple path p is defined as a sequence of edges $p = \langle (v_0 = s, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k = t) \rangle$ such that no edge is repeated, i.e., $((v_i, v_{i+1}) \neq (v_j, v_{j+1})), \forall (i \neq j)$. When a path p is queried, the IT admin selects exactly one edge $e \in p$ to remove. We model the IT admin's choice using the Bradley–Terry model, which assigns a probability to each edge $e \in p$ proportional to its confidence score relative to the others in p:

$$Pr(e|p) = \frac{conf(e)}{\sum_{e' \in p} conf(e')} \tag{1}$$

In other words, an edge with a higher confidence score is more likely to be chosen for removal. This models the administrator's relative preference or belief about which edge's removal is most effective. Let C be the set of edges removed by the IT administrator after $|C|$ round. At round $|C| + 1$, the system will proposed a path $p \in P_{G'}$ in the temporary graph $G' = (V, E \setminus C)$ where G' is called the temporary graph which evolved from the original graph $G = (V, E)$ by removing set of C edges and $P_{G'}$ is the set of every possible path from s to t in G'. The query process terminates when either C forms an (s, t)-cut (i.e., s is disconnected from t) or the query budget B is reached (i.e. $|C| = B$). In this paper, all cuts refer to $s - t$ cuts. To minimize human effort during the cutting process, our optimization goal is to design a policy that minimizes the expected number of queries (or iterations) required to complete the cutting process.

Theorem 1. *The APR Problem is $\#\mathcal{P}$-hard*

We defer the proof to the Extended Version [Ngo *et al.*, 2025].

Reason for edge's confidence score and how to assign it Integrating confidence scores helps us effectively embed domain-specific security information into our model, making it easier to identify edges that are more likely to be removable. This matters because not all edges are equally prone to be removed by IT admin; some edges, such as outdated privileges or overly permissive group assignments, are clear candidates for removal. Using insights from the security context in our edge preference model could substantially reduce the number of required queries. To automate the assignment of confidence scores, we can train a binary classifier that predicts the likelihood of each edge being safely removable. For example, Zheng et al. [Zheng *et al.*, 2011] propose an active learning approach that learns an IT admin's decisions about which edges to remove. We can automatically assign confidence scores to edges by using a binary classifier, defined as a function $f : E \mapsto [0, 1]$ where the output represents the classifier's confidence that a given edge can be safely removed.

2.2 Related Works

Active Directory and non-adaptive defense models. The seminal work by Dunagan et al. [Dunagan *et al.*, 2009] proposed the Active Directory (AD) attack graph which modelled the identity snowball attack that developed further and commercialized by Bloodhound [Robbins, 2023]. Follow-up works by Guo et al. [Guo *et al.*, 2022; Guo *et al.*, 2023] and Zhang et al. [Zhang *et al.*, 2023] formulate the problem of hardening the AD system as the shortest path interdiction via edge-removing problem. [Goel *et al.*, 2022; Goel *et al.*, 2023] proposed the Evolutionary Diversity Optimization (EDO) algorithm to defend against attackers in a configurable environment. Another work by Zhang et al. [Zhang *et al.*, 2024] studied the problem of minimizing the number of users with paths to the domain admin via edge removal. Another approach for defending Active Directory found in the literature involves node-removal, which abstracts the concept of decoy allocation as introduced in Ngo et al. [Ngo *et al.*, 2024b; Ngo *et al.*, 2024a]. The main drawback of non-adaptive models in real-world deployments is that they are not amenable to include human feedback.

Adaptive models for Active Directory defense. Several studies have integrated manual feedback from IT admin into

network defenses process, emphasizing the importance of human involvement in configuration changes. Dunagan et al. [Dunagan *et al.*, 2009] proposed Heat-ray, a system aimed at minimizing snowball identity attacks in Active Directory (AD) by iteratively proposing edge removals to IT administrators based on the sparest cut. Zheng et al. [Zheng *et al.*, 2011] enhanced Heat-ray with active learning to improve edge cost learning process. Guo et al. [Guo *et al.*, 2024] introduced an adaptive defense model called the Limited Query Graph Connectivity Test (LQGCT), which is closely related to our approach. In their model, a proxy algorithm proposes one edge at a time, and the IT admin's decision is binary (i.e. whether to remove or retain it). By contrast, the proxy of our model proposes an entire attack path instead of a single edge which offers a multiple-choice selection rather than a binary decision. Proposing a path provides several practical advantages over proposing an edge. Firstly, an edge proposal can fail to form a graph cut if the IT admin is overly conservative and retains too many edges. This leaves the possibility of an attack even after the clean-up. In our experiments, path-based proposals guarantee that no attack path remains, provided the proxy algorithm has a sufficiently large budget. Secondly, by presenting a list of edges to compare, our model encourages more deliberate choices, whereas a binary question as in LQGCT may incentivize conservative behaviour. From a theoretical view, path-based proposals fundamentally differ and are harder to solve than the previous edge-based model. In LQGCT, the policy tree is binary, while our policy tree can branch into up to l outcomes at each step, where l is the length of the longest proposed path. As a result, existing algorithms cannot be directly applied to our setting, requiring us to develop an entirely new class of solutions.

Related models from other research communities. The sequential testing problem in operations research [Ünlüyurt, 2004], is often described through medical testing use cases. For instance, [Short and Domagalski, 2013; Yu *et al.*, 2023] employs adaptive strategies to reduce testing costs to diagnose diseases. Another related area is the problem of learning with attribute costs problem in machine learning [Sun *et al.*, 1996; Kaplan *et al.*, 2005; Golovin and Krause, 2011]. In this problem, each feature incurs a cost, and the task is to construct a classification tree that minimizes the total feature costs. Stochastic Boolean Function Evaluation (SBFE) problem [Allen *et al.*, 2017; Deshpande *et al.*, 2014] is also closely related. An SBFE instance involves a Boolean function f with multiple hidden binary inputs and one binary output. Each input bit can be queried at a cost, and the objective is to find a query strategy that minimizes the expected cost to determine f's output. While these models are relevant, they are not designed for our graph-based problems and lack scalability for large graphs. Consequently, similar to LQGCT, solutions for these models cannot be directly applied to our work.

3 Algorithms

In this part, we will present our solution for the APR problem. To help with the solution formulation, we will convert our problem into an equivalent Markov Decision Process (MDP).

3.1 Markov Decision Process formulation and preliminary

Let us define the MDP as a tuple $\langle \mathcal{S}, \mathcal{A}, \Phi, R \rangle$, where $\mathcal{S}$ is the set of state, $\mathcal{A}$ is the set of action, $\Phi : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the state transition and the reward function $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$.

State: In an APR problem, the IT admin will remove an edge from a proposed path in every round. This process will evolve the graph into a series of temporary graphs by removing edges. We present these temporary graphs using a temporary state variable s with $|B|$-dimension: $s = \{(x_1, x_2, \cdots, x_B) : x_i \in E \cup \{*\}, \forall i \in \{1, 2, \cdots, B\}\}$ where x_i will be the edge that is removed by the IT admin at round i and $x_i =$ '*' means we have not query any path in this round. We define the state of the original attack graph G as the root state s_r, which will have the form: $(*, *, \cdots, *)$. A state s' evolves from a state s by removing an edge e expressed as $s' = s \setminus e$. We denote $\mathcal{S}_i$ the set of possible states in round i of the process. Hence, the state space can be represented as $\mathcal{S} = \mathcal{S}_0 \cup \mathcal{S}_1 \cup \mathcal{S}_2 \cup \cdots \cup \mathcal{S}_B$. We also define two sets of terminal states: $\perp_b$ is the terminal state reached when the budget is exhausted without identifying a cut, and $\perp_d$ is the terminal state reached when a cut is successfully found as a result of the query sequence.

Action: Each path proposal in the APR problem is associated with an action in MDP. Let's say we are at state s' associated with a temporary graph G' and $A_{s'}$ is the action space at state s'. The action $a \in A_{s'}$ associates with a simple path $p \in P_{G'}$. We have the following Lemma for the action space in our problem:

Lemma 2. *Given an MDP construction $\langle \mathcal{S}, \mathcal{A}, \Phi, R \rangle$ for the APR problem. We have $A_s \subseteq A_{s_r}$ for every $s \in \mathcal{S}$ where s_r is the root state.*

Proof. The action available at each temporary state s is the enumeration of every possible path in the corresponding graph G. A temporary graph G' is actually the subgraph of the root graph G (as G' is evolved from G by removing edge) which implies $P_{G'} \subseteq P_G$. Therefore, we have $A_s \subseteq A_{s'}$, $\forall s, s'$ where $s \in successor(s')$ which imply $A_s \subseteq A_{s_r}$. □

Lemma 2 states that action set A_s of every state $s \in \mathcal{S}$ is a subset of the action set A_{s_r} of the root state s_r. As a result, the overall action space for the APR problem can be expressed as $A = \bigcup_{s \in \mathcal{S}} A_s = A_{s_r}$. Lemma 2 is particularly useful in the design of our algorithms, as discussed in the following sections.

Transition Probabilities: In each state, an action can lead to different outcomes, which are defined by the transition probabilities in the MDP. In our problem, a transition probability shows the probability of an edge being removed by the IT admin when a path is proposed. The removal probability is defined by the Bradley-Terry preference model, as defined in Equation (1). Let's say we have a state $s' = s \setminus e$ where s evolved to s' by removing edge e. The transition probability from s to s' when taking action a can be expressed as $\Phi(s'|s, a) = \Phi(e|s, a) = Pr(e|p)$.

Reward: In our problem, each query will be penalized by a cost of exactly one unit of budget with no discount factor. The reward is difference at two terminal states: when $\perp_b$ is

reached, meaning that we ran-out of budget before identifying any cut, we will penalize it with a constant of $\alpha > 0$;

$$R(s,a) = \begin{cases} -\alpha, & \text{if } s \in \perp_b . \\ 0, & \text{if } s \in \perp_d . \\ -1, & \text{otherwise.} \end{cases} \quad (2)$$

Realization: While "realization" is not a standard notation in MDP literature, it is commonly used in the context of adaptive submodularity optimization [Golovin and Krause, 2011]. We introduce this concept here as it will help us in describing the algorithms. We define a function $\phi : P_{s_r} \mapsto E$ as the full realization. We can view function $\phi(p)$ as an oracle that returns the IT administrator's edge removal decision for a given path p. Additionally, we define the partial realization $\psi_s : P_{s_r} \mapsto E$ as the observations made so far at state s. Specifically, $\psi_s(p)$ returns the edge removed by the IT administrator when p is proposed, and $\psi_s(p) = *$ if p has not yet been proposed or contains any edge have been removed by the IT admin. The domain of the partial realization is defined as $dom(\psi) = \{p \in P_{s_r} : \psi(p) \neq *\}$, representing the set of actions for which outcomes have been observed. The range of the partial realization is defined as $range(\psi) = \{\psi(p) : p \in P_{s_r}, \psi(p) \neq *\}$, representing the edges that have been removed by IT admin. A partial realization ψ is consistent with realization ϕ if they are equal everywhere in the $dom(\psi)$, denoted $\phi \sim \psi$. We say a partial realization ψ is a subrealization of ψ', denoted by $\psi \subseteq \psi'$, if they are both consistent with some ϕ and $dom(\psi) \subseteq dom(\psi')$.

3.2 Dynamic Programming Exact Algorithm (OPT)

As we establish the equivalence between the APR problem and the MDP, we also find that our problem satisfies the "Principle of Optimality" in MDP. In our problem, given a state s and $s' = s \setminus e, \forall e \in E_s$ and E_s is the set of edges in the graph corresponding to state s, the optimal path query in s is independent of previous queries and solving for the optimal strategy at s' can be viewed as a subproblem of s. This allows us to introduce the optimal utility function following the Bellman equation:

$$u_\pi(s) = \min_{p \in A_s} \{r(s,a) + \sum_{e \in p} \Phi(e|s,p) u_\pi(s \backslash e)\}, \ \forall s \in \mathcal{S} \quad (3)$$

Based on the the optimal utility function in (3), we can design a Bellman's style dynamic programming, called OPT. Now, let G_s be the graph that is associated with state s. The Dynamic Programming follows a top-down approach. In each subproblem, we require the utility $u_\pi(s)$. However, as shown in Equation (3), obtaining the optimal utility requires invoking the action set A_s for every subproblem, which in turn requires enumerating every simple path P_{G_s} in G_s. This approach is impractical to run on any graph of reasonable size because the number of subproblems can grow as large as $|E|^B$, and path enumeration, known to be $\#\mathcal{P}$-hard, takes $\mathcal{O}(|V|^k)$ time complexity under a DFS-based approach [Peng *et al.*, 2019], where k is the longest path length. However, thanks to Lemma 2, we can run the enumeration one time only for the original problem. The action space for the subproblem is simply $A_s = A_{s_r} \setminus \{p : e \in p, p \in A_{s_r}, e \in range(\psi_s)\}$, i.e., we can obtain A_s by removing paths in A_{s_r} that contains edges that have been chosen to be removed by the IT admin. Since running Depth First Search procedure to check if s, t connectedness in every subproblem will take $\mathcal{O}(|E| + |V|)$. The overall Exact algorithm will take us $\mathcal{O}(|V|^k + (|E| + |V|) * |E|^B)$. We will defer the detailed pseudoscope of this algorithm to the appendix.

3.3 Adaptive Submodular Approximation Algorithms (APP)

In this section, we present an approximation algorithm, called APP, by utilizing the adaptive submodularity framework [Golovin and Krause, 2011]. The proposed algorithm provides square-logarithmic approximation to the number of enumerations of possible simple paths in the original graph G. The Adaptive Submodular Algorithm is proposed for the Stochastic Submodular Set Coverage (SSSC) problem in [Golovin and Krause, 2011] which has a close connection with APR problem.

Problem 1. *The SSSC problem involves a ground set of elements $U = \{u_1, u_2, \cdots u_n\}$ and a collection of items $E = \{e_1, e_2, \cdots, e_m\}$, where each item e is associated with a distribution over subsets of U. When an item is selected, a set is sampled from its distribution, i.e., it will reveal which subset of U will be covered. The objective of this problem is to find an adaptive policy π that selects items to cover all elements in U while minimizing the expected number of items. To define the coverage, we define a utility function $f : 2^E \mapsto \mathbb{R}$ that quantifies the coverage achieved by the current state. The complete coverage is represented as states with utility meeting a predefined quota Q, i.e., $f(s) = Q$.*

We can see that our problem can be viewed as a special case of the SSSC problem. In the APR problem, the ground set consists of all simple paths P_{s_r}. In every round, when a path p is proposed, the IT admin will choose an edge $e \in p$ to remove, every path in the set $P' = \{p' | e \in p', \forall p' \in P_{s_r}\}$ (the set of paths containing e) will also be removed. Each path in the action space can be viewed as an item in the SSSC problem, with each path associated with a distribution over the potential removal of other paths. This distribution is presented as in Equation 1. The goal of APR problem is to cover all paths in P_{s_r} (a cut eliminates all paths) while minimizing the number of queries. Next, we have the following definitions.

Definition 1. ***(Conditional expected marginal benefit)*** *Given a state s, an action a and a utility function g, the expected marginal benefit of a is defined as:*

$$\Delta(a \mid s) = \sum_{e \in a} \{\Phi(e \mid s, a) * [g(s \setminus e) - g(s)]\} \quad (4)$$

Definition 2. ***(Adaptive Monotonicity)*** *A utility function $g : S \mapsto \mathbb{R}_{\geq 0}$ is adaptive monotone if the benefit of selecting an action is always nonnegative. Formally, function g is adaptive monotonic if $\forall s \in S$ and $\forall e \in \{e \mid e \in a, a \in A_s\}$, we have:*

$$g(s \setminus e) - g(s) \geq 0 \quad (5)$$

Definition 3. ***(Adaptive Submodular)*** *A utility function $g : S \mapsto \mathbb{R}_{\geq 0}$ is adaptive submodular if the marginal benefit of selecting an action does not increase as more actions are selected. Formally, function g is adaptive submodular if for all temporary state s, s' such that $\psi_s \subseteq \psi_{s'}$, $\forall a \in A_{s_r}$, we have:*

$$\Delta(a \mid s) \geq \Delta(a \mid s') \tag{6}$$

The reason for introducing these concepts is to port algorithms from the Adaptive Submodular framework to the APR problem while ensuring the theoretical approximation bound. To do so, we need to design a utility function g associated with the APR problem that satisfies two key conditions: (1) adaptive monotonicity and (2) adaptive submodularity.

Utility Function: The utility function $g : S \mapsto \mathbb{N}$ is defined as:

$$g(s) = |\bigcup_{e \in range(\psi_s)} h(e, P_{s_r})| \tag{7}$$

where the function $h(e, P) = \{p | e \in p, \forall p \in P\}$ returns the set of paths $p \in P$ that contains e. To remind, $range(\psi_s)$ is the set of edges that have been removed upon state s. We have the following lemma for the utility function g:

Lemma 3. *Function g is both adaptive monotonic and adaptive submodular*

Proof. First, consider the function g, which counts the number of paths removed up to the current state. Suppose we are at any state s, and path p is proposed to the IT admin, who then chooses to remove an edge e. Removing e eliminates at least the proposed path p since $e \in p$. This means $g(s \setminus e) \geq g(s) + 1$ and satisfying Equation (5). Therefore, g is adaptive monotone.

Moreover, since the utility function g, as defined in Eq. (7), returns the number of paths removed by IT admin decision from the root state. Furthermore, removing an edge e in a state s with $\psi_s \subseteq \psi_{s'}$ will result in more paths being eliminated than removing e in the successor state s' which is formally expressed as $g(s \setminus e) - g(s) \geq g(s' \setminus e) - g(s')$. This implies:

$$\begin{aligned}\Delta(a \mid s) &= \sum_{e \in a} \{\Phi(e \mid s, a) * [g(s \setminus e) - g(s)]\} \\ &\geq \sum_{e \in a} \{\Phi(e \mid s', a) * [g(s' \setminus e) - g(s')]\} = \Delta(a \mid s')\end{aligned}$$

The transition from line 1 to line 2 is valid because $\Phi(e \mid s, a) = \Phi(e \mid s', a), \forall s, s', a$ as the transition probabilities of an action do not depend on the state, as defined in the equation 1. This proves the adaptive submodularity of g. □

Note, to ensure Theorem 4 holds, g must be both *strongly adaptive monotonic and submodular*. While Definition 2 aligns with the concept of strongly adaptive monotonicity [Golovin and Krause, 2011], Definition 3 is only adaptive submodularity. A function is strongly adaptive submodular if it is (1) adaptive submodular and (2) pointwise submodular. Although we have proven the former, we admit the second property for g and hence g is also strongly adaptive submodular. We defer the definition and proof of (2) to Extended Version [Ngo *et al.*, 2025].

Greedy with marginal gain strategy (Algorithm 1): By applying a greedy strategy with our problem-tailored utility function g, we have our Adaptive Submodular Algorithm as shown in Algorithm 1. In this algorithm, during each query round, we greedily propose the action $a \in A_s$ that yields the highest expected marginal benefit with respect to the utility function g. Here again, due to Lemma 2, we only need to enumerate the action space (for the G_{s_r}) once beforehand. Note, while the path enumeration problem is known to be $\#\mathcal{P}$-hard, our experiments with attack graphs demonstrate that this enumeration can be performed within a reasonable runtime. The efficiency of this process is largely because attack graphs typically involve only subgraphs of the larger AD structure.

Algorithm 1 Adaptive Submodular Strategy (APP)

Input: Directed graph $G(V, E)$, source s and destination t
Output: approximate propose strategy
1: Initialise set of simple path A_{s_r} of original state s_r, set of proposed path A_π
2: **while** s is not terminate state
3: **foreach** $a \in A_{s_r} \setminus A_\pi$
4: $\Delta(a|s) = \sum_{e \in a} \{\Phi(e|s, a) * [g(s \setminus e) - g(s)]\}$
5: $a^* = \arg\max_a \Delta(a|s)$
6: $A_\pi = A_\pi \cup \{a^*\}$, proposed a^*, observe outcome $e*$
7: $s = s \setminus e^*$, progressing to new state

Once we have proven that g is both adaptive monotonic and adaptive submodular in Lemma 3, the following theoretical approximation ratio for Algorithm 1 follows.

Theorem 4. *Algorithm 1 achieves a $(\ln |P_{s_r}|)^2$-approximation for the APR problem with $B = |P_{s_r}|$.*

3.4 Scalable Heuristics Algorithm

In this section, we present a scalable heuristic designed based on the exact algorithm and the approximate algorithm as shown in Algorithm 2

Heuristics based on Exact Algorithm (DPR) The exact dynamic programming algorithm struggles to scale in realistic scenarios due to two main issues. First, APR problem's MDP often has large action space, particularly in the initial states, where the action space size corresponds to the enumeration of simple paths in the original graph. Second, for each action, the number of child subproblems to solve can grow up to $\mathcal{O}(|E|^k)$. We proposed a scalable heuristic in Algorithm 2, designed to restrict the subproblem space in dynamic programming to a manageable size and enable efficient execution on graphs of practical scale. We call this algorithm dynamic programming with restriction (DPR). Function $DPR(s', r, B')$ in Algorithm 2 shows a modification of the Exact Dynamic Programming algorithm with restriction. The first restriction is that instead of considering subproblems from B steps ahead, ***DPR*** reduces the lookahead to B' steps, where $B' < B$. The second restriction is to avoid enumerating every possible state (which can be problematic in the early stages). Instead, we only consider a set of τ candidate paths, implemented as the $path\_sampling(A_{s'}, \tau)$ function in Algorithm 2. In general, we modify each heuristic to return

the top $k < \tau$ candidate paths, rather than a single best path, based on each heuristic's ranking criterion. For example, the approximate heuristic ranks paths by their marginal gain according to g, then selects the top k among them. This function will draw from multiple heuristic methods—such as those derived from an approximation strategy (Algorithm 1), shortest paths, approximate strategies on sets of shortest paths, or paths likely to remove an edge in a minimum cut. As our experiments show that these approaches provide strong performance. This modification decreases the number of subproblems to $\mathcal{O}((\tau)^{B'})$ for each DP, making DP more feasible in larger settings. While this adjustment may affect the optimality of the solution (compared to the exact DP algorithm), it significantly improves the scalability. Empirically, we will experimentally demonstrate that the ***DPR*** algorithm scales better than the exact algorithm and outperforms the approximate algorithm on every graph.

Algorithm 2 Heuristics based on Exact Algorithm (DPR)

Input: Directed Attack Graph $G(V, E)$, budget B, lookahead budget B'
Output: Heuristic query strategy

while s is not a terminate state
  $\pi = DPR(\mathbf{s}, |A_\pi|, B')$
  $a^* = \pi(\mathbf{s})$
  $A_\pi = A_\pi \cup \{a^*\}$, proposed a^*, observe outcome $e*$
  $\mathbf{s} = \mathbf{s} \setminus e^*$, progressing to new state

function $DPR(\mathbf{s}', r, B', \tau)$
  for $i \in [0, B']$
    for $\mathbf{s}' \in \mathcal{S}_{|A_\pi|+i}$
      if $\mathbf{s}'$ is in $\perp_b$ or $\perp_d$
        $u_\pi(\mathbf{s}' \setminus e) = \begin{cases} \alpha, & \text{if } \mathbf{s}' \in \perp_b \\ 0, & \text{if } \mathbf{s}' \in \perp_d \end{cases}$
        $\pi(\mathbf{s}' \setminus e) = \emptyset$
      else
        $A = path\_sampling(A_{\mathbf{s}'}, \tau)$
        $a^* = \text{argmin}_{p \in A}\{\sum_{e \in p}[\Phi(e|\mathbf{s}', a) * u_\pi(\mathbf{s}' \setminus e)]\}$
        $u_\pi(\mathbf{s}') = 1 + \sum_{e \in p}[\Phi(e|\mathbf{s}', a) * u_\pi(\mathbf{s}' \setminus e)]$
        $\pi(\mathbf{s}') = p^*$
  return π

4 Experiment

In this section, we present the evaluation of our algorithm on 13 synthetic graphs of different sizes and an Active Directory (AD) attack graph from a real organization.

4.1 Experiment Set Up

In our experiment, we evaluate our algorithm using synthetic AD attack graphs generated by ADSynth [Nguyen *et al.*, 2024], a state-of-the-art AD graph generator. ADSynth models AD graphs based on Microsoft's best practices tiering model [Microsoft, 2024a; Microsoft, 2024b], where Tier 0 contains the highest privilege nodes with administrative control, Tier 1 includes high-privileged servers, and Tier 2 and beyond contain non-administrative nodes. ADSynth simulates the AD attack graph in two steps: (1) generating a best-practice AD infrastructure and (2) creating cross-tier edges.

If every node had a predefined tier, the defense problem would become trivial, as attack paths could be easily identified and removed by removing all edges connecting lower-privilege nodes to higher-privilege nodes [Knudsen, 2021]. Open-source tools like ImproHound [Knudsen, 2021] are designed to automate this process. However, assigning roles to nodes is inherently challenging due to the dynamic nature of roles, overlapping responsibilities, and exceptions such as temporary access [Knudsen and Schmitt, 2023]. We called these *undefined tier nodes*. In our simulated attack graph, we assume the presence of a set of nodes with undefined tier connections which create attack paths from lower-privilege nodes to higher-privilege nodes. We assume that IT admin use our adaptive model with the goal of removing all attack paths from the lowest tier to Tier 0. Our model treats the attack graph as a single-source, single-target graph so we merge all Tier 0 nodes into a single supernode t and all lowest-tier nodes into a single supernode s.

For our synthetic attack graphs, we labelled them from $G1$ to $G9$. In these graphs, the number of tiers is fixed at 3, and 95% of nodes in graph have well-defined tier assignments. Additionally, we also have 4 smaller versions of the graph denoted from $GS1$ to $GS4$, used in the experiment in Table 1. In this small graph, defined-role ratio is about 99%. We also included one real AD graph that we collected from a University, we denoted this graph as ORG.

All of the experiments are carried out on a high-performance computing cluster with 1 CPU and 24GB of RAM allocated to each trial. In Tables 1 and 2, we report the average number of queries over 16,000 trials. The budget constraint B is set at 10 for all experiments on synthetic graphs. For the real AD graph ORG, due to computing resource limitations, we report the average number of queries over 200 trials. Also for ORG, we reserve a higher budget of 20 and 30 queries due to the size of this graph, denoted ORG(20) and ORG(30) respectively. For the DPR algorithm, we set $\tau = 16$ actions and a lookahead budget of $B' = 4$ step. We reserve a higher budget of 20 and 30 queries due to the size of this graph. For the DPR algorithm, we set $\tau = 16$ actions and a lookahead budget of $B' = 4$ step.

	$GS1$	$GS2$	$GS3$	$GS4$
OPT	**2.513**	**2.592**	**2.545**	**2.383**
APP	**2.513**	**2.592**	2.546	2.385
OTH1	**2.513**	**2.592**	2.546	**2.383**
OTH2	**2.513**	2.596	**2.545**	2.388
PPO	**2.513**	**2.592**	2.546	**2.383**
SAC	2.514	**2.592**	2.546	2.384
DPR	**2.513**	**2.592**	2.546	**2.383**

Table 1: Expected number of query under different algorithm (↓ **is better**). Here, we only consider graphs where OPT can run on.

	G1	**G2**	**G3**	**G4**	**G5**	**G6**	**G7**	**G8**	**G9**	**ORG**(20)	**ORG**(30)	AVG.RANK
#n/#e	1047/5078	1047/5091	1047/5116	5147/25376	5139/25153	5139/25161	10070/48161	10070/48170	10070/48192	125444/1195432	-	-
MC	3	3	3	3	3	4	3	3	3	8	-	-
APP	3.821	3.762	4.534	4.334	3.879	4.594	3.807	3.869	3.590	17.605	18.840	3.889
OTH1	3.816	**3.755**	**4.409**	**3.885**	3.880	4.593	3.810	3.893	3.584	17.485	18.600	3.185
OTH2	**3.813**	3.756	4.437	3.904	3.883	4.592	3.799	3.871	3.570	17.535	**18.535**	3.333
PPO	3.816	**3.755**	4.425	3.905	3.876	4.587	3.797	3.876	3.573	17.665	18.835	2.667
SAC	3.854	3.799	4.490	3.901	**3.874**	4.606	**3.792**	3.876	3.575	18.005	18.560	2.667
DPR	3.816	**3.755**	**4.409**	3.901	3.876	**4.589**	3.797	**3.869**	**3.568**	**17.480**	18.555	**1.444**

Table 2: Expected number of query under different algorithm (↓ **is better**). AVG.RANK represents the average head-to-head performance ranking of each algorithm across all evaluated graphs. $\#n/\#e$ show the number of nodes and edge in the graph. MC is the min-cut.

4.2 Baseline Algorithms

Reinforcement Learning. This approach shares a similar concept with DPR but replaces the use of Dynamic Programming with restricted lookahead by a model-free reinforcement learning to learn the query strategy. We utilize two model-free reinforcement learning models: Proximal Policy Optimization (PPO) [Schulman *et al.*, 2017] and Soft Actor-Critic for Discrete Action (SAC) [Christodoulou, 2019]. We encode the observation space as a vector of $(E + \tau B)$ binary bits. The first E bits represent a one-hot encoding of the edges that have been removed through queries, while the remaining τB bits encode the taken actions. We allocated a100 GPUs for the training of RL agents.

Others Heuristics We also introduce two other heuristics called OTH1 and OTH2 which are designed based on the approximate algorithm. For OTH1, we modify the utility function to ensure it will propose paths with the highest likelihood of removing an edge in the minimum cut set. Formally, it selects the path $a = \arg\max_{a \in P_{G'}: a \cap mc(G')} \sum_{e \in a} \{\Phi(e|s, a) * [g(s \setminus e) - g(s)]\}$ where $mc(G')$ return the $s - t$ minimum cut of the temporary graph G'. For the OTH2, we restrict the approximate algorithm to run on the set of shortest paths only.

4.3 Performance Interpretation

In Table 1, we report the performance of our proposed algorithm in the graph where OPT can optimally come up with the query policy without out-of-memory error. As we mentioned, OPT is very costly computationally, we are only able to scale it to a graph with 17 nodes, 32 edges and 16 attack paths ($GS4$ graph). Overall, all of our heuristics (OTHs, PPO, SAC and DPR) perform very well with the small optimality gap.

In Table 2, we present the performance of our algorithm on 13 large synthetic graphs and one real-world AD graph (**ORG**) from a University. We observed that DPR consistently achieved the best performance. Noticeably, DPR outperformed APP in every graph. All proposed algorithms outperformed APP. Nevertheless, APP's performance is theoretically guaranteed which may be useful for some worst-case scenarios. This algorithm is also useful as a path sampling scheme for the DPR algorithm as the action space of DPR algorithm contain the approximate strategy which somewhat helps DPR to have a guaranteed performance. While the RL algorithms (PPO and SAC) performed well on synthetic graphs, their performance was worse on the real AD graph. We suspect this is due to the real graph requiring a significantly larger number of queries. The RL policies are myopic, meaning they excel in scenarios with fewer queries by prioritizing short-term gains but struggle when a higher number of queries is needed, as they fail to account for long-term gains.

The adaptive hardening of AD security has been studied as the LQGCT problem by Guo et al. [Guo *et al.*, 2024]. Their model simulates IT admin' behaviour as a binary decision-making process: at each step, an edge is queried, and the administrator labels it as either "cut" or "retain." However, this approach often fails when IT admins are overly conservative, retaining too many edges and leading to unsuccessful cuts. In contrast, our model queries a path in each step, presenting a multiple-choice decision for the IT admin to select one edge to remove. In table 3, we compare the graph-cutting performance of the RL algorithm from LQGCT with our DPR algorithm. The results from 512 trials demonstrate that our model achieved more successful cuts compared to Guo's model. We observe that a larger budget leads to a higher cutting success rate in our model, this means we can guarantee a successful cut in every trial by allocating a sufficiently larger budget.

	LQGCT's RL		DPR	
	B = 5	B = 10	B = 5	B = 10
G7	128	144	278	459
G8	96	96	419	504
G9	96	96	339	487

Table 3: The number of trials with successful cuts between RL from LQGCT model and DPR algorithm over 512 trials. (↑ **is better)**

5 Conclusion

In this paper, we proposed a practical human-in-the-loop combinatorial problem for network security called Adaptive Path Removal problem. This problem was motivated by the technical requirements and limitations of current industrial models. The goal of our model is to reduce the workload for security teams in an adaptive manner. We proposed a comprehensive set of solutions, including an exact algorithm, an approximate algorithm, and several scalable heuristics. Among these, our DPR heuristic, designed based on both the exact and approximate algorithms, exhibited superior performance. Specifically, DPR demonstrated the ability to run effectively on larger-scale graphs compared to the exact algorithm and consistently outperformed the approximate algorithm across all tested graph scenarios. We verify the effectiveness of our algorithm on several synthetic AD graphs and an AD attack graph collected from a real organization.

Acknowledgments

We acknowledge the supercomputing resources provided by the Phoenix HPC service at the University of Adelaide.

References

[Allen *et al.*, 2017] Sarah R Allen, Lisa Hellerstein, Devorah Kletenik, and Tonguç Ünlüyurt. Evaluation of monotone dnf formulas. *Algorithmica*, 77:661–685, 2017.

[Christodoulou, 2019] Petros Christodoulou. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

[Deshpande *et al.*, 2014] Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic boolean function evaluation and stochastic submodular set cover. In *Proceedings of the twenty-fifth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1453–1466. SIAM, 2014.

[Dunagan *et al.*, 2009] John Dunagan, Alice X Zheng, and Daniel R Simon. Heat-ray: combating identity snowball attacks using machinelearning, combinatorial optimization and attack graphs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 305–320, 2009.

[Goel *et al.*, 2022] Diksha Goel, Max Hector Ward-Graham, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. Defending active directory by combining neural network based dynamic program and evolutionary diversity optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1191–1199, 2022.

[Goel *et al.*, 2023] Diksha Goel, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. Evolving reinforcement learning environment to minimize learner's achievable reward: An application on hardening active directory systems. *GECCO '23: Genetic and Evolutionary Computation Conference, 2023, 2023*, 2023.

[Golovin and Krause, 2011] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

[Guo *et al.*, 2022] Mingyu Guo, Jialiang Li, Aneta Neumann, Frank Neumann, and Hung Nguyen. Practical fixed-parameter algorithms for defending active directory style attack graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9360–9367, 2022.

[Guo *et al.*, 2023] Mingyu Guo, Max Ward, Aneta Neumann, Frank Neumann, and Hung Nguyen. Scalable edge blocking algorithms for defending active directory style attack graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 5649–5656, 2023.

[Guo *et al.*, 2024] Mingyu Guo, Jialiang Li, Aneta Neumann, Frank Neumann, and Hung Nguyen. Limited query graph connectivity test. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 20718–20725, 2024.

[Kaplan *et al.*, 2005] Haim Kaplan, Eyal Kushilevitz, and Yishay Mansour. Learning with attribute costs. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 356–365, 2005.

[Knudsen and Schmitt, 2023] Jonas Bülow Knudsen and Alexander Schmitt. Hidden pathways: Exploring the anatomy of acl-based active directory attacks and building strong defenses. https://troopers.de/troopers23/talks/33fcyz/, 2023.

[Knudsen, 2021] Jonas Bülow Knudsen. Improhound: Identify the attack paths in bloodhound breaking your ad tiering. https://github.com/improsec/ImproHound, 2021.

[Microsoft, 2024a] Microsoft. Best practice guide for securing active directory installations. https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/best-practices-for-securing-active-directory#reduce-active-directory-attack-surface, 2024.

[Microsoft, 2024b] Microsoft. Mitigating pass-the-hash (pth) attacks and other credential theft, version 1 and 2. https://www.microsoft.com/en-au/download/details.aspx?id=36036, 2024.

[Ngo *et al.*, 2024a] Huy Ngo, Mingyu Guo, and Hung Nguyen. Optimizing cyber response time on temporal active directory networks using decoys. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1309–1317, 2024.

[Ngo *et al.*, 2024b] Huy Q Ngo, Mingyu Guo, and Hung Nguyen. Catch me if you can: Effective honeypot placement in dynamic ad attack graphs. In *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*, pages 451–460. IEEE, 2024.

[Ngo *et al.*, 2025] Huy Quang Ngo, Mingyu Guo, and Hung Nguyen. Adaptive wizard for removing cross-tier misconfigurations in active directory (extended version). https://arxiv.org/abs/2505.01028, 2025.

[Nguyen *et al.*, 2024] Nhu Long Nguyen, Nickolas Falkner, and Hung Nguyen. Adsynth: Synthesizing realistic active directory attack graphs. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 66–74. IEEE, 2024.

[Peng *et al.*, 2019] You Peng, Ying Zhang, Xuemin Lin, Wenjie Zhang, Lu Qin, and Jingren Zhou. Hop-constrained st simple path enumeration: Towards bridging theory and practice. *Proc. VLDB Endow.*, 13(4):463–476, 2019.

[Robbins, 2023] Andy Robbins. "bloodhound: Six degrees of domain admin. https://github.com/BloodHoundAD/BloodHound, 2023. Accessed: 2022-08-02.

[Schulman *et al.*, 2017] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[Short and Domagalski, 2013] Matthew W Short and Jason E Domagalski. Iron deficiency anemia: evaluation and

management. *American family physician*, 87(2):98–104, 2013.

[Sun *et al.*, 1996] Xiaorong Sun, Steve Y Chiu, and Louis Anthony Cox. A hill-climbing approach for optimizing classification trees. In *Learning from Data: Artificial Intelligence and Statistics V*, pages 109–117. Springer, 1996.

[Ünlüyurt, 2004] Tonguç Ünlüyurt. Sequential testing of complex systems: a review. *Discrete Applied Mathematics*, 142(1-3):189–205, 2004.

[Yu *et al.*, 2023] Zheng Yu, Yikuan Li, Joseph Kim, Kaixuan Huang, Yuan Luo, and Mengdi Wang. Deep reinforcement learning for cost-effective medical diagnosis. *arXiv preprint arXiv:2302.10261*, 2023.

[Zhang *et al.*, 2023] Yumeng Zhang, Max Ward, Mingyu Guo, and Hung Nguyen. A scalable double oracle algorithm for hardening large active directory systems. *The 18th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS)*, 2023.

[Zhang *et al.*, 2024] Yumeng Zhang, Max Ward, and Hung Nguyen. Practical anytime algorithms for judicious partitioning of active directory attack graphs. In *33rd International Joint Conference on Artificial Intelligence*, pages 7074–7081. International Joint Conferences on Artificial Intelligence, 2024.

[Zheng *et al.*, 2011] Alice X Zheng, John Dunagan, and Ashish Kapoor. Active graph reachability reduction for network security and software engineering. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1750, 2011.