Published: 29 August 2025
Online AM: 12 August 2024
Accepted: 14 July 2024
Revised: 09 July 2024
Received: 06 January 2024

Citation in BibTeX format

RESEARCH-ARTICLE

Hardening Active Directory Graphs via Evolutionary Diversity Optimization-based Policies

DIKSHA GOEL, Commonwealth Scientific and Industrial Research Organisation, Canberra, ACT, Australia

MAX WARD, The University of Western Australia, Perth, WA, Australia

ANETA NEUMANN, The University of Adelaide, Adelaide, SA, Australia

FRANK NEUMANN, The University of Adelaide, Adelaide, SA, Australia

HUNG X NGUYEN, The University of Adelaide, Adelaide, SA, Australia

MINGYU GUO, The University of Adelaide, Adelaide, SA, Australia

Hardening Active Directory Graphs via Evolutionary Diversity Optimization-based Policies

DIKSHA GOEL, CSIRO's Data61, Clayton, Australia
MAX WARD, Department of Computer Science and Software Engineering, University of Western Australia, Perth, Australia
ANETA NEUMANN, FRANK NEUMANN, HUNG NGUYEN, and MINGYU GUO, School of Computer and Mathematical Sciences, University of Adelaide, Adelaide, Australia

Active Directory (AD) is the default security management system for Windows domain networks. An AD environment can be described as a cyber-attack graph, with nodes representing computers, accounts, and so forth, and edges indicating existing accesses or known exploits that enable attackers to move from one node to another. This article explores a Stackelberg game model between one attacker and one defender on an AD attack graph. The attacker's goal is to maximize their chances of successfully reaching the destination before getting detected. The defender's aim is to block a constant number of edges to minimize the attacker's chance of success. The article shows that the problem is #P-hard and, therefore, intractable to solve exactly. To defend the AD graph from cyberattackers, this article proposes two defensive approaches. In the first approach, we convert the attacker's problem to an exponential-sized Dynamic Program that is approximated by a neural network (NN). Once trained, the NN serves as an efficient fitness function for defender's Evolutionary Diversity Optimization-based defensive policy. The diversity emphasis on the defender's solution provides a diverse set of training samples, improving the training accuracy of our NN for modeling the attacker. In the second approach, we propose a RL-based policy to solve the attacker's problem and Critic network-assisted Evolutionary Diversity Optimization-based defensive policy to solve defender's problem. Experimental results on synthetic AD graphs show that the proposed defensive policies are scalable, highly effective, approximate attacker's problem accurately and generate good defensive plans.

CCS Concepts: • **Security and privacy → Network security**; • **Computing methodologies → Artificial intelligence**;

Additional Key Words and Phrases: Attack graph, active directory, reinforcement learning, evolutionary diversity optimization

Authors' Contact Information: Diksha Goel (corresponding author), CSIRO's Data61, Clayton, Australia; e-mail: diksha.goel@data61.csiro.au; Max Ward, Department of Computer Science and Software Engineering, University of Western Australia, Perth, Australia; e-mail: max.ward@uwa.edu.au; Aneta Neumann, School of Computer and Mathematical Sciences, University of Adelaide, Adelaide, Australia; e-mail: aneta.neumann@adelaide.edu.au; Frank Neumann, School of Computer and Mathematical Sciences, University of Adelaide, Adelaide, Australia; e-mail: frank.neumann@adelaide.edu.au; Hung Nguyen, School of Computer and Mathematical Sciences, University of Adelaide, Adelaide, Australia; e-mail: hung.nguyen@adelaide.edu.au; Mingyu Guo, School of Computer and Mathematical Sciences, University of Adelaide, Adelaide, Australia; e-mail: mingyu.guo@adelaide.edu.au.

ACM 2688-3007/2025/8-ART19
https://doi.org/10.1145/3688401

ACM Reference format:
Diksha Goel, Max Ward, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. 2025. Hardening Active Directory Graphs via Evolutionary Diversity Optimization-based Policies. *ACM Trans. Evol. Learn. Optim.* 5, 3, Article 19 (August 2025), 36 pages.
https://doi.org/10.1145/3688401

1 Introduction

Cyber-attack techniques utilize attack graphs to identify the possible ways an attacker can exploit to gain unauthorized access to the systems. Although cyber-attack graphs are immensely popular in the industry and academic sector, they do not have a standard definition. Lallie et al. [2020] studied around 180 attack graphs from the literature and found more than 90 self-proposed definitions for attack graphs. Industrial practitioners are actively using the **Active Directory (AD)** attack graph, which is an attack graph model. Microsoft *AD* is a security control system for Windows domain networks [Dias, 2002]. AD is a critical service that holds user and server account information and is, therefore, essential for the correct working of the Microsoft domain. Microsoft domain network constitutes significant market shares among small and big organizations globally, due to which ADs are promising targets for cyberattacks. Microsoft reported that 90% of Fortune 1000 companies use AD. According to Enterprise Management Associates [2021], 50% of the organizations surveyed had experienced an AD attack since 2019. The AD structure depicts an attack graph where nodes are the computers, accounts, applications, and an edge from node X to node Y denotes that an attacker may access node Y from node X using some existing access or known exploits.

Various applications and tools are designed to investigate the AD attack graphs; however, BloodHound[1] is the most popular tool. BloodHound can be used to identify various attack paths in AD, including users, groups, trust relationships, and unique AD objects. BloodHound models an *identity snowball attack* where the attacker initiates an attack by gaining access to a low privileged account (attacker's entry node) via a phishing attack and tries to move to other nodes aiming to reach the most-privileged account, **Domain Admin (DA)** [Dunagan et al., 2009]. Figure 1 illustrates an example of BloodHound snowball identity attack. One fundamental functionality of BloodHound is that it uses Dijkstra's algorithm to generate the shortest attack paths from the entry node of the attacker to the DA and measures the length of the attack path in number of hops. BloodHound has made it much easier for the attacker to attack AD. Due to the popularity of AD attack graphs, defenders also study AD graphs for designing defensive strategies. Dunagan et al. [2009] proposed a heuristic solution for blocking a few edges to partition the attack graph into various disconnected components. The resulting disconnected components of the graph disable attackers from reaching DA only if the attacker's entry nodes and DA are in different components. Edge blocking in an AD environment can be attained by either monitoring the edges or revoking access.[2]

Evolutionary Diversity Optimization (EDO) is an **evolutionary computation (EC)** paradigm aimed at generating a diverse array of solutions for complex problems. Unlike traditional optimization methods that seek a single best solution, EDO focuses on exploring a variety of solutions. This approach fosters less competitive interactions among evolving solutions, facilitating a more thorough exploration of potential outcomes. As a result, EDO supports the development of robust and adaptable solutions across fields such as engineering, robotics, biology, and beyond. EDO approaches have been used to solve various attacker–defender problems [Goel, 2023; Hu et al., 2020;

[1]https://github.com/BloodHoundAD/BloodHound
[2]ImproHound is another tool that guides in blocking the edges by creating an information table of all edges that may cause "tier violations."

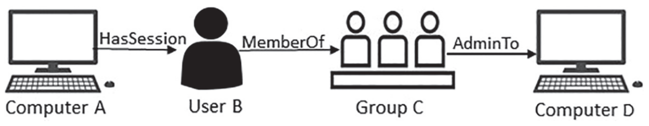

Fig. 1. Example of BloodHound snowball identity attack.

Hemberg et al., 2018; Zhang and Hemberg, 2019]. Ulrich and Thiele [2011] and Ulrich et al. [2010] studied EDO that aims to find a set of diverse solutions. EDO has gained considerable attention in the EC community. A new solution deviating from its predecessors leads to less competitiveness and high evolvability [Lehman and Stanley, 2013]. *In this article, the defender aims to block a set of edges to minimize the strategic attacker's probability of reaching DA.* To solve the defender's problem, we consider the blocking plans of the defender as a population and propose EDO-based defensive approach to generate a diverse set of blocking plans.

In the literature, Guo et al. [2022] proposed several algorithms to address this problem. However, the authors have considered a scenario where once the attacker is detected, the attack ends and when an attacker chooses a path, the attacker continues to move on to that path until the attacker gets caught or reaches DA. The attacker's problem in Guo et al. [2022] can be solved using Dijkstra's algorithm, but in our model, the attacker's problem is computationally hard. In our model, we assume that every edge e has a different *detection probability* $p_{d(e)}$, i.e., if the attacker is detected, the whole attack ends and a *failure probability* $p_{f(e)}$, i.e., if the attacker fails and is not able to pass through an edge, the attacker can try another edge from the same node or different node. For example, an edge requires cracking a password to pass through it; if the password is weak, then the attacker can crack it. In this case, the probability of having a strong password is the failure probability. With every edge's $p_{d(e)}$ and $p_{f(e)}$, the attacker can successfully go through an edge e with a *success probability* $p_{s(e)} = (1 - p_{d(e)} - p_{f(e)})$, where $p_{d(e)} + p_{f(e)} \leq 1$.

Therefore, in our model, the attacker plays strategically in the sense that the attacker first starts an attack, and if, at some point, fails (instead of being detected), the attacker can try again until the attacker is detected or has tried all possible options. The attacker aims to design an attacking policy that maximizes their probability of reaching DA. Initially, the attacker has access only to a set of entry nodes, from where the attacker can start. While trying to reach DA, the attacker expands the set of accessible nodes by exploring edges and saves this information, i.e., set of successful edges the attacker currently has control of, set of edges that the attacker has lost (failed but not detected, for not being able to guess the password, etc.). All the strategies that attacker has tried are an "*investment*" that the attacker can use later. In this way, the attacker has "*secured*" a set of nodes at any point and can attempt an unattempted edge originating from any of the secured nodes or entry nodes, which we combinedly call checkpoints, Checkpoints = {Entry nodes ∪ Secured nodes}. Generally, the attacker prefers the attack paths with low detection and failure probability and covers the valuable checkpoints along its way. The attacker often needs to consider the tradeoff between these desired traits. For example, an attacker may decide to go for a path having low detection probability and relatively high failure probability; however, the covered checkpoints on this path make it convenient for the attacker to launch alternative attack attempts if failures happen. The optimal attacking policy is the one that maximizes attacker's chances of reaching DA without getting detected. On the other hand, the defender wants to deterministically block k **block-worthy (BW)** edges, where k is the defender's budget, in turn increasing the corresponding edge's failure rate from original p_f to 100%. We follow a *Stackelberg game model* [Yin et al., 2010], where the defender devises a strategy, and the attacker observes the defender's strategy and plays his best to develop an attack strategy on the target. In practice, the attacker can scan the AD environment using SharpHound[3] tool and get information about which edges are blocked. Therefore, it is reasonable to consider that the attacker knows the defender's strategy and plays the best response to it.

We aim to propose an effective approach for computing defender strategy (to block a set of edges that minimizes strategic attackers' probability of reaching DA) that scales to large AD attack graphs. We prove that the attacker's problem of deriving an optimal attacking policy is $\#P$-hard. Also, we prove that the defender's problem is also $\#P$-hard, even if the blocking budget is one. Therefore, the problem is intractable to solve exactly with the current methods. Hence, we propose heuristic approaches to address the attacker–defender problem.

In this work,[4] we describe the attacker's problem as **Markov Decision Process (MDP)**, which can be solved using a **Dynamic Program (DP)** [Bellman, 1966]. The size of state space is $3^{|\text{Edges}|}$, where an edge is either unattempted, attempted, and failed, or attempted and successful. However, this state space is too large considering that practical AD graphs may have tens of thousands of edges. We use a *fixed-parameter analysis technique* that focuses on determining easy-to-solve instances of a problem. A problem is **fixed-parameter tractable (FPT)** if it has a solution that runs in $O(f(b)n^c)$ time, where b is the input parameter, c is a constant, n is the problem size, and f is any arbitrary function. Here, we say that the problem is FPT w.r.t. b, and the runtime of the problem is polynomial for input size n. We propose kernelization technique that exploits the structural features of the AD attack graphs to obtain a much smaller *condensed graph*. Using our kernelization technique, the state space becomes FPT with respect to a parameter called the number of **Non-Splitting Paths (NSPs)**. Guo et al. [2022] proposed NSP idea to describe tree likeness of AD graphs. NSP is a path on which every node has only one successor except the last node. Once the original AD graph is converted to a smaller graph, this article proposes two approaches to solve the attacker–defender problem.

Our first approach involves training a **Neural Network (NN)** to approximate attacker's problem and EDO to solve the defender's problem. The attacker's MDP can be solved using DP. The original AD graph is reduced to a condensed graph via kernelization. For small AD graphs, we can solve DP, and for larger AD graphs, our FPT special parameter $\#NSP$ is practically too large, so we propose to use NN to approximate the DP [Yang et al., 2018]. Our main idea is to train NN to learn both the base cases and the DP recursive relationships. Considering that the state space is exponential and we do not have resources to train NN to learn the value of every state; however, not all states are useful. It is important to learn the values of the states that are referenced by the optimal decision path, therefore, we only consider the important states in the state space, which reduces the state space size to a large extent. With the strong flexibility power of NN, we aim to train the NN to approximate the attacker's policy. Once trained, NN acts as an efficient fitness function for the defender's EDO (the exact fitness function is $\#P$-hard to compute). Notably, it is essential to design a sophisticated attacker's policy, as we cannot have an effective defense without having an accurate attacker's policy. Moreover, we solve the defender's problem of blocking k BW edges using EDO. The defender uses EDO to come up with a set of diverse blocking plans so as to feed diverse training data to NN and improving the accuracy of our NN for modeling the attacker. Our fitness function (NN) is an approximated function, so having the "best" defensive plan according to the fitness function is not too meaningful, which is why we instead try to search for a diverse set of "good enough" defensive plans. The defensive plans are given as input to the NN, and it outputs a value, i.e., attacker's probability of reaching DA corresponding to the blocking plan. We go back and forth

[3]https://github.com/BloodHoundAD/SharpHound

[4]This article is an extension of our previous works [Goel et al., 2022] and [Goel et al., 2023], accepted at GECCO 2022 and GECCO 2023, respectively.

between the processes, generating blocking plans using EDO and training NN on blocking plans, to get a well-trained NN that can act as an efficient fitness function for EDO. *In this way, EDO and NN help each other in order to improve.*

In our second approach, we propose a **reinforcement learning (RL)**-based policy to maximize the attacker's chances of successfully reaching the DA. We propose a **Critic Network-Assisted Evolutionary Diversity Optimization (C-EDO)**-based defensive policy to find defensive plan configurations (blocking plans) that minimize the attacker's success rate. We propose RL-based policy to approximate the attacker's problem that we modeled as MDP and use **Proximal Policy Optimization (PPO)** RL algorithm, an *actor–critic-based approach* to train the attacker policy. The RL agent interacts with "*multiple environments*" simultaneously by suggesting actions with the goal of maximizing the overall reward. We propose C-EDO-based policy to solve the defender problem. C-EDO generates numerous RL environment configurations based on different underlying defensive plans. In this context, the term "environment" refers to the setting in which the RL agent operates and learns. In our model, each environment encapsulates a defensive plan, which is why we use the terms "environment configurations" and "defensive plans" interchangeably. Our approach uses the trained RL critic network to estimate the fitness of environment configurations. The defender continuously monitors the RL training process, and after regular intervals, the defender uses the trained critic network to evaluate the current configurations and uses C-EDO to generate better ones. The attacker and defender play against each other in parallel. Overall, the defender's C-EDO generates numerous high-quality, diverse environment configurations worth learning for the RL agent to train a better attacking policy. We emphasize on high-quality, diverse environments to prevent the RL agent from spending time on learning environments that are irrelevant to an effective defense. We train our attacker's policy using an actor–critic-based algorithm, so we inherently have a critic network that gives us a reasonable estimation of the state values and can be used as a fitness function for the defender's C-EDO. In this way, these two problems are interconnected, and the solution strategies complement each other.

Our Contributions.

(1) *Hardness Proofs.* We first prove that the attacker's problem of deriving an optimal attacking policy is $\#P$-hard, and the defender's problem is also $\#P$-hard.
(2) *Kernelization Technique.* We design a kernelization technique that exploits the structural features of original AD graph and reduces it to a much smaller condensed graph.
(3) *NNDP+EDO Defensive Approach.* We propose an approach that train a NN for approximating the attacker's problem and EDO to solve the defender's edge-blocking problem. Our experimental results on the synthetic $R500$[5] AD attack graph shows that the proposed approach (attacker's policy and defender's policy) is highly effective and is less than 1% away from the optimal solution.
(4) *RL+C-EDO Defensive Approach.* We propose a defensive policy that combines RL and EDO to generate edge-blocking plans. Our approach trains the attacker's policy on multiple independent defensive plan environments simultaneously so as to obtain an effective defensive policy. Our experimental results show that the proposed approach is highly effective, approximates the attacker's problem more accurately, and generates better defense and scalable to $R4000$ AD graphs.

Organization. Section 2 presents the background and Section 3 discusses the related work. In Section 4, we outline the problem description, and Section 5 presents the hardness results. Section 6 presents our proposed FPT kernelization technique, and Section 7 describes our proposed NN and

[5]R500 AD graph is an AD graph containing 500 computers.

EDO-based attacker–defender approach in detail. Following that, Section 8 presents our proposed RL and EDO-based policy. Section 9 reports the experimental results. Section 10 presents the discussion, and Section 11 concludes the article.

2 Background

Evolutionary Diversity Optimization. EDO is an important paradigm in EC, focusing on creating a variety of solutions that are distinctly different from each other. Initially introduced by Ulrich et al. [2010], EDO has become a key area of research. This approach aims to explore the solution space thoroughly, aiming to find creative and reliable solutions across various fields. The formal objective of EDO is to maximize solution diversity while ensuring high quality, highlighting its crucial role in advancing computational methods toward greater innovation and resilience [Neumann et al., 2018]. Unlike conventional optimization techniques that focus on finding the most optimal solution to a problem, EDO seeks to generate a multitude of effective solutions. This method acknowledges that in many real-world scenarios, having several good options can be more advantageous than having a single optimal solution [Neumann et al., 2019]. For instance, in complex decision-making environments such as air traffic control, multiple viable solutions can provide the flexibility to adapt to sudden changes or unexpected situations. Similarly, in the context of personalized agent planning, diverse solutions can cater to varying preferences and circumstances, enhancing user satisfaction and system robustness [Do et al., 2022]. EDO utilizes algorithms that encourage genetic diversity within the population of solutions, thus avoiding the common pitfall of converging too quickly toward similar solutions. This is achieved through specific mechanisms that measure and maintain diversity throughout the evolutionary process, such as niche preservation techniques. By promoting a wide exploration of the solution space, EDO can uncover novel solutions that might be overlooked by more traditional methods focused solely on optimization. Moreover, the approach has been successfully applied in various domains that benefit from diverse solution sets. In addition to the previously mentioned fields of air traffic management and personalized planning, EDO is increasingly being used in areas such as bioinformatics, robotic path planning, and complex system design [Liu et al., 2023]. These applications highlight the versatility and practicality of EDO in tackling problems where the solution requirements are continually evolving or where multiple criteria need to be satisfied simultaneously [Ming et al., 2022; Wang et al., 2021a].

Dynamic Programming. DP is a widely used technique for designing efficient optimization algorithms [Bellman, 1966]. This technique is particularly effective for problems that can be decomposed into smaller, overlapping subproblems, each reflecting a stage in a decision-making process [Bellman and Dreyfus, 2015]. The fundamental approach of DP involves breaking down an overarching problem into simpler subproblems, solving each of these individually, and using the solutions of these subproblems to construct a solution to the larger problem. A critical feature of DP is its use of memoization, which involves storing the solutions of these subproblems in a structured manner (often a table or other data structure) [Korel and Laski, 1988]. This prevents the need for recalculating the solution of a subproblem each time it is needed, thereby significantly enhancing the efficiency of the algorithm. One of the key characteristics of DP is its ability to find both local solutions for the subproblems and a global solution for the entire problem. The process typically starts at the base cases—subproblems small enough to be solved straightforwardly—and progressively works toward solving larger and larger subproblems until the overall problem is solved [Hollingsworth et al., 1994]. By solving for the optimal solutions at each subproblem and cleverly combining these solutions, DP ensures that the final solution is optimal.

Neural Networks. NNs are the foundation for various **artificial intelligence (AI)** applications such as speech recognition, image recognition, self-driving cars, and detecting cancer. Currently, NNs are outperforming human accuracy on a range of tasks. The exceptional performance of NNs can largely be attributed to their ability to autonomously extract high-level features from raw data. This capability allows them to develop sophisticated representations of the input space, which is critical in making accurate predictions or decisions based on vast and complex datasets [Gawlikowski et al., 2023; Samek et al., 2021]. NNs achieve this through a layered architecture, where each layer builds an increasingly abstract representation of the data. Theoretically, NNs with a sufficient number of parameters, known as deep NNs, have the potential to approximate any complex function [Li et al., 2021]. This characteristic is known as universal approximation capability, and it is crucial for tasks such as recognizing subtle patterns in images or understanding nuanced speech in voice recognition systems. Moreover, the efficiency of NNs in processing large datasets has significantly accelerated the pace at which machine learning models can be trained. This rapid training capability, combined with their high accuracy, makes NNs invaluable for tasks such as detecting cancerous cells in medical imaging or enabling real-time decision-making in self-driving cars [Abdulsatar et al., 2024; Babiker et al., 2019; Kasban et al., 2015].

Reinforcement Learning. RL is a type of machine learning that focuses on teaching agents how to make a sequence of decisions. The agent learns to achieve a goal in a complex, uncertain environment by performing actions and experiencing their results. In essence, the agent learns to maximize cumulative rewards through a process of trial and error, continuously adjusting its strategy based on the outcomes of its actions [Ladosz et al., 2022; Wang et al., 2020]. At the core of RL is the use of MDPs, which provide a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. RL utilizes techniques such as value functions, which estimate how good it is for an agent to be in a given state, and policy optimization, which seeks to find a policy that maximizes the reward by mapping states to the most effective actions [Ding et al., 2020; Matsuo et al., 2022]. The versatility of RL allows it to be applied in numerous fields. In autonomous robotics, for instance, RL helps robots learn complex tasks that involve a series of interconnected decisions, such as navigating through a dynamic environment [Akalin and Loutfi, 2021; Brunke et al., 2022]. In gaming, RL algorithms can be employed to develop AI that adapts to the strategies of human players [Yang and Wang, 2020]. Moreover, RL has significant applications in recommendation systems where it helps in personalizing content dynamically as per user interaction and feedback. Additionally, the healthcare sector has seen applications of RL in areas such as personalized medicine, where algorithms learn to recommend treatment plans tailored to individual patient responses [Tang and Wiens, 2021; Yu et al., 2021]. Through these applications, RL continues to demonstrate its capacity to solve complex, real-world problems in a way that mimics some aspects of human learning. Its ongoing development promises to uncover even more sophisticated ways in which machines can learn from their environment, making it a central field of study in AI.

3 Related Work

Active Directory. Guo et al. [2022] investigated the edge interdiction problem for AD attack graphs where the defender's goal is to maximize the expected path length of the attacker. The authors proposed an NSP idea to describe the tree-likeness of AD attack graphs. The authors used Graph Neural Networks to determine the attacker's local decision at each split node, where when an attacker chooses a path, the attacker continues to move on to that path until the attacker reaches a split node (node with multiple outgoing edges), and the attacker needs to decide which outgoing edge to choose. However, in our model, the attacker may have many checkpoints across the whole graph, and the attacker can explore the graph from any checkpoint. Therefore, the attacker makes a global decision by considering a set of split nodes. Due to this reason, the solution [Guo et al., 2022] is not applicable to our model. Guo et al. [2023] proposed various scalable algorithms for defending AD attack graphs. However, the problem settings that they considered are different from ours. The authors proposed a tree decomposition-based DP and RL-based approaches that are scalable to large AD graphs. Ngo et al. [2023] designed a near-optimal policy for the placement of honeypots on computer nodes when the AD graph is dynamically changing; however, the authors studied identifying blocking nodes, whereas we focused on blocking edges. Zhang et al. [2023] proposed a scalable double oracle algorithm and compared their solution against various industry solutions; the authors considered different problem settings than ours. Goel et al. [2024] develops advanced edge-blocking defenses for dynamic AD systems and introduces a RL Training Facilitator to efficiently train strategies on large, dynamic AD graphs, enhancing scalability and defender proficiency.

The AD graph problem investigated in this study is crucial due to its direct impact on organizational security. AD is a critical component in enterprise environments, managing user access and permissions. Attackers often target AD to exploit vulnerabilities and gain privileged access, resulting in severe security breaches. Once inside the AD environment, attackers can manipulate user permissions, forge credentials, and spread malware across the network. This enables them to compromise sensitive data, disrupt operations, and potentially cause extensive damage within the organization. Compromised AD accounts can also serve as launching points for further attacks, allowing attackers to maintain persistence and evade detection while continuing their malicious activities. Protecting AD from these threats is essential for maintaining organizational security and integrity. By addressing this specific AD graph problem in our study, we aim to develop robust defensive mechanisms that effectively mitigate these risks. While previous studies have explored defense mechanisms for AD graph problems, they have typically focused on different scenarios than ours. As a result, there is currently no solution in the literature that directly aligns with our specific problem statement. This gap underscores the need to design a tailored solution that addresses our unique AD graph problem, ensuring comprehensive protection against evolving cyber threats.

Generative Adversarial Networks (GAN). Chivukula et al. [2020] proposed a game-theoretic approach using a Stackelberg game between a variational adversary and a CNN, optimizing adversarial manipulations to mislead CNNs and enhance classifier robustness. Wu et al. [2021] developed a cyberdefense model using game theory to obscure system configurations, applying linear programming and NNs to compute equilibria, supported by experimental validation. Vorobeychik et al. [2012] enhanced Defender–Attacker Stackelberg games by incorporating attacker observability and improving the model with advanced Nonlinear Programming and Mixed-Integer Linear Programming techniques, leading to better strategic outcomes. He et al. [2024] introduced a Generative Artificial Intelligence-enabled game theory approach for mobile network optimization, integrating generative AI into game theory and demonstrating effectiveness in secure Unmanned Aerial Vehicle network contexts. Wang et al. [2021b] propose min-max optimization for adversarial training to enhance adversarial attack design across multiple domains. Introducing domain weights and a unified framework, they improve attack generation strategies, showing enhancements in attacking model ensembles, universal perturbations, and attacks resilient to data transformations.

Moreover, GANs and coevolutionary algorithms both employ adversarial and evolutionary principles, albeit in different applications. GANs involve a generator and discriminator competing in a game to improve data generation quality, while coevolutionary algorithms optimize solutions through competitive or cooperative interactions among multiple entities. GANs focus on generating realistic data, whereas coevolutionary algorithms generalize this approach to diverse optimization tasks, making them applicable in cybersecurity and other dynamic environments requiring adaptive strategies. Additionally, our adversarial algorithm shares the similar principles of GANs, specifically the mechanism of adversarial learning where both attacker and defender iteratively refine their strategies through feedback. Similar to the interplay between generator and discriminator in GANs, our model features a continuous strategic evolution among network security actors. However, unlike GANs, which focus on data generation, our approach is tailored for network security within

AD environments, emphasizing decisions such as edge interdictions within complex AD graphs. This necessitates a sophisticated understanding of network topology and security protocols. While we adopt the adversarial competition framework from GANs, our method is specifically engineered to tackle the unique challenges of cybersecurity in large-scale AD systems, moving beyond mere data simulation to provide targeted, strategic security solutions. This specialization ensures our approach is precisely suited to address the intricacies of AD systems, underscoring the necessity for customized cybersecurity measures.

Evolutionary Diversity Optimization. Identifying high-quality, diverse solutions has attained huge attention in EC under the concept of EDO [Ulrich et al., 2010] and Quality Diversity [Cully and Demiris, 2017]. Both approaches have recently been applied to solve traditional combinatorial optimization problems. Hebrard et al. [2005] proposed a solution to find diverse solutions for constrained programming. Do et al. [2020] studied the computation of high-quality and diverse solutions for the traveling salesman problem. In another study, Do et al. [2022] analyzed EDO techniques for various permutation problems. Huang et al. [2023] designed a co-evolutionary approach to improve the searchability and convergence of competitive swarm optimizer. Bossek and Neumann [2021] investigated EDO for computing a diverse set of solutions for minimum spanning tree problem. Neumann et al. [2022] proposed a coevolutionary pareto diversity optimization approach for optimizing constrained single-objective problems. Nikfarjam et al. [2023] investigates evolutionary algorithms with a Boolean Satisfiability solver to maximize diversity among solutions for the heavily constrained Boolean satisfiability problem. Neumann et al. [2023] present EDO algorithms for constructing wireless communication networks, minimizing transmission coverage, and avoiding adversaries. Hu et al. [2020] proposed a stochastic evolutionary game model to optimize cyber defense strategies. Winterrose et al. [2020] designed a game-theoretical model demonstrating how attackers adapt their strategies to defender tactics, with diversity defenses proving more effective in short-term engagements. Cui et al. [2021] developed a EC-based model to enhance data processing in Cyber-Physical Social Systems, achieving superior accuracy over existing models. Table 1 provides a comprehensive comparison between the proposed EDO-based defensive approach against various defensive methodologies in adversarial environments.

Neural Networks. Bello et al. [2017] proposed a pointer network-based approach for solving traveling salesman and knapsack problems. The authors have trained the pointer network with RL to avoid generating high-quality labelled data. Yang et al. [2018] proposed a general approach to boost the performance of the DP with NNs. The authors used a solution reconstruction procedure that samples solutions, and a NN is trained to assess the quality of each solution. Xu et al. [2020] designed a model that integrates NNs with DP to solve various optimization problems. The authors proposed two solutions, value and policy-based, which considerably reduce time with reasonable performance loss.

Reinforcement Learning. In the domain of cyber defense and RL, several approaches have been designed. Piplai et al. [2022] developed a two-player game-based RL environment. They utilized expert-guided autonomous agents within the CyberBattleSim framework to enhance cyber defense training, aiming to improve both attack and defense capabilities. Alavizadeh et al. [2022] introduced an application of Deep Q-Learning by integrating it with a deep feedforward NN, aimed at dynamic and continuous network intrusion detection. Apruzzese et al. [2020] proposed a framework to increase the resilience of botnet detectors against adversarial attacks. Their method involves generating realistic attack samples for training, thereby maintaining the performance of these detectors even against novel threats. Hammar and Stadler [2020] explored the automatic evolution of attack and defense strategies through a Markov game-based method that utilizes RL and self-play. Their findings indicate that effective security strategies comparable to human-devised ones can

Table 1. Comparative Analysis of the Proposed Evolutionary Diversity Optimization Base Defensive Approach against Various Defensive Methodologies in Adversarial Environments

Feature	**Proposed EDO-based defense approach**	**Standard evolutionary computation approaches** [Mirjalili, 2019; Sampson, 1976; Vikhar, 2016; Xue et al., 2015; Zames, 1981]	**Reinforcement learning-based defense** [Feng and Xu, 2017; Lin et al., 2017]	**Game theory-based defense** [Li et al., 2019; Sharma, 2021]
Objective	Focuses on minimizing the strategic attacker's success probability through diversified defensive planning.	Typically optimize for static defense measures or pre-defined reactive strategies.	Aims to learn optimal defense policies through trial and error interactions with the environment.	Develops strategies based on mathematical models of attacker–defender interactions.
Methodology	Employs NN and RL as fitness function, adapting defensive strategies based on attacker behavior analysis.	Often use fixed fitness functions based on pre-determined scenarios, lacking adaptability to changing attack patterns.	Leverages RL algorithms to dynamically adjust defenses in response to observed attacks.	Applies game-theoretic models to predict and counteract attack strategies.
Diversity	Ensures a high diversity in defensive plans, aiming to cover a broad spectrum of potential attack strategies.	May lack emphasis on solution diversity, potentially leading to effectiveness against only specific types of attacks.	Can explore diverse policies, though effectiveness depends on the exploration–exploitation balance.	Produces varied strategies based on different game scenarios, but might be limited by the accuracy of game models.
Adaptability	Adapts using feedback from attacks, enhancing defensive responsiveness and effectiveness.	Typically static once deployed, requiring manual updates to tackle new or evolving threats effectively.	Adaptable, continuously improving policies based on feedback from the environment.	Adaptable to game model strategies, but may need recalibration for drastically different scenarios.
Training Demands	High, due to ongoing NN and RL training to simulate the attacker's problem accurately.	Generally lower training needs, but at the expense of reduced adaptability and predictive accuracy.	High, as RL requires substantial interaction data to effectively learn optimal policies.	Moderate, requiring complex math and scenario analyses.

Table 2. List of Symbols Used in the Article

Symbol	Description
G	Directed graph
V	Set of nodes
E	Set of edges
n	Number of nodes in graph
m	Number of edges in graph
k	Defender's budget
s	Entry nodes
t	Splitting nodes
A	Admissible set of actions
P	Population
S	State space
A	Action space
R	Reward function
T	Transition function
$p_{d(e)}$	Detection probability of edge e
$p_{f(e)}$	Failure probability of edge e
$p_{s(e)}$	Success probability of edge e
MSE	Mean squared error
OPT	Optimal solution

emerge, despite challenges in policy convergence. Simoes et al. [2020] ventured into complex, multi-agent settings by developing a Pokemon Battle Environment to test advanced RL algorithms.

4 Problem Description

AD graph can be represented as a directed graph $G = (V, E)$, where V is nodes set, and E is edges set. The highest privilege accounts in AD are called DA. This article considers a two-player Stackelberg game between one defender and one attacker to defend AD graphs. There are s entry nodes. The attacker can start from one of the entry nodes and aims to design a strategy to maximize their chances of successfully reaching DA. In our model, every edge in the AD graph has a detection rate, failure rate, and success rate. Table 2 shows the list of symbols used in the article. The *detection rate* $p_{d(e)}$ signifies that if an attacker is detected, the attack ends immediately. In real-world scenarios, various detection mechanisms, such as intrusion detection systems and real-time monitoring solutions, may be deployed across different edges. These mechanisms include specific monitoring protocols that trigger alerts for suspicious activities or unauthorized access attempts. Implementing these strategies enhances overall security by swiftly identifying and mitigating potential attacks, thereby minimizing the risk of unauthorized access to critical resources. The *failure rate* $p_{f(e)}$ indicates the obstacles that hinder an attacker's progress through edge e in the AD graph. These barriers encompass strong authentication mechanisms, robust password policies, and multi-factor authentication requirements.

If an attacker fails to bypass these security measures, such as correctly guessing a password, they cannot proceed along edge e. Importantly, a failed attempt means the attacker is unable to traverse edge e successfully. In this case, the attack does not terminate, and the attacker can continue from other checkpoints by trying unexplored edges. The *success rate* $p_{s(e)}$ denotes the attacker's likelihood of successfully passing through edge e and is calculated as $p_{s(e)} = 1 - p_{f(e)} - p_{d(e)}$. This metric considers factors such as exploit effectiveness, evasion of detection mechanisms, and

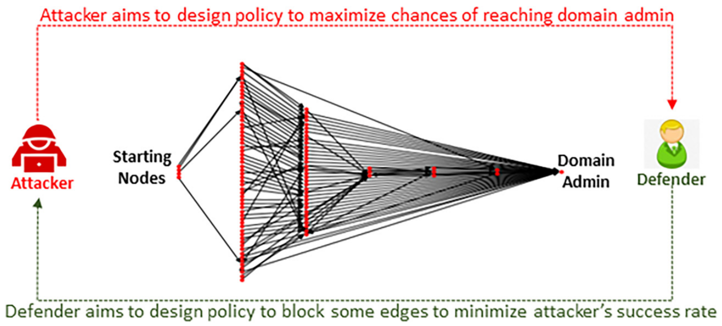
Attacker aims to design policy to maximize chances of reaching domain admin
Attacker
Starting Nodes
Domain Admin
Defender
Defender aims to design policy to block some edges to minimize attacker's success rate

Fig. 2. Attacker–defender problem on AD graph.

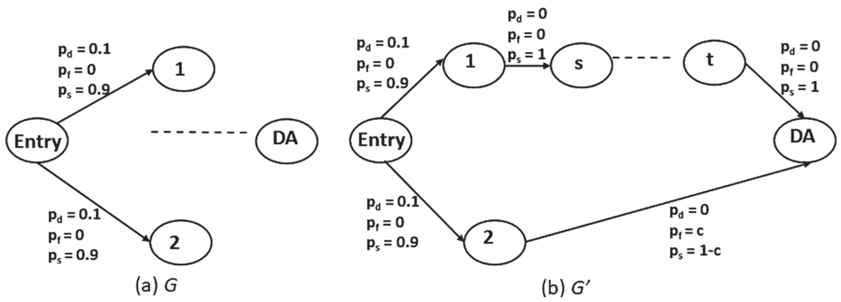

Fig. 3. G' is constructed from G.

the successful exploitation of vulnerabilities. It quantifies the attacker's strategic ability to exploit weaknesses in the AD infrastructure and navigate toward their objective. The strategic attacker starts from one of the entry nodes and tries unexplored edges in order to reach DA. At any time during the attack, the checkpoint indicates the set of nodes that the attacker has control of and can continue the attack from (in case the attacker fails to pass through the edge). The defender seeks to block a set of edges so as to minimize the attacker's success rate. The defender has a limited budget and can only block k edges. Not all edges are blockable; therefore, the defender can only block "*blockable*" edges. Edge blocking is costly and requires efforts (auditing access logs) to safely remove an edge; due to this, we cannot remove too many edges and have assumed a limited budget. We assume that the attacker can observe the defensive action and accordingly come up with the best-response attacking policy. Figure 2 illustrates the overall attacker–defender problem in an AD graph, with an instance of AD attack graph created using DBCreator.[6]

5 Hardness Results

Let G be a graph with one entry node ENTRY and one destination node DA as shown in Figure 3(a). For both the edges ENTRY $\longrightarrow$ 1 and ENTRY $\longrightarrow$ 2 in G, $p_d = 0.1$ (p_d can be any arbitrary value greater than 0) and $p_f = 0$.

Theorem 1. *The attacker's value is #P-hard to calculate.*

Proof. The attacker's value is the attacker's probability for reaching DA before getting detected under the attacker's optimal attacking policy, facing a given defense (Figure 3(a)). A known #P-complete result for the $s - t$ connectedness problem [Valiant, 1979] states that, given nodes s and t

[6]DBCreator is a synthetic AD graph generator tool developed by the BloodHound team.

in a directed graph, if every edge's probability of existence is 0.5, then it is #P-complete to calculate the probability that s and t are connected. We can apply this result to our problem by assuming an edge's $p_f = 0.5$ and $p_d = 0$ for all edges in the attack graph.[7] Therefore, the attacker's value is #P-hard to compute. □

Theorem 2. *The attacker's optimal policy is #P-hard to calculate.*

Proof. We construct a graph G' from G as shown in Figure 3(b). Let G' be a directed graph with a source node s and a destination node t. All the edges in graph G' have $p_d = 0$ and $p_f = 0.5$. We add two edges $1 \longrightarrow s$ and $t \longrightarrow DA$ to G', each with $p_d = p_f = 0$. In addition, we add another edge $2 \longrightarrow DA$ with $p_d = 0$, $p_f = c$ where c is any constant. In Figure 3(b), the attacker needs to determine whether the attacker should go up (Entry⟶1) or down first (Entry⟶2). Let us assume that the total failure rate for going from s to t is p_{up}, and the failure rate for going from 2 to DA is p_{dn}. If the attacker goes up (Entry⟶1) first, then,

$$\text{Attacker's value} = 0.9(p_{up} + (1 - p_{up})0.9p_{dn}).$$
$$\text{Attacker's value} = 0.9p_{up} + 0.81p_{dn} - 0.81p_{up} \times p_{dn}.$$

If the attacker goes down first (Entry⟶2), then,

$$\text{Attacker's value} = 0.9(p_{dn} + (1 - p_{dn})0.9p_{up}).$$
$$\text{Attacker's value} = 0.9p_{dn} + 0.81p_{up} - 0.81p_{up} \times p_{dn}.$$

Therefore, the attacker needs to determine a higher failure rate from two, p_{up} or p_{dn}. We call this problem an up-down deciding problem, and the attacker's optimal policy is at least as difficult as the up-down deciding problem. Let's assume we have an oracle that solves the up-down deciding problem. We can compute the exact value of p_{up} by calling the oracle polynomial number of times (value of p_{up} is from a finite set, $\{\frac{0}{2^m}, \frac{1}{2^m}, \dots \frac{2^m}{2^m}\}$, where m is the number of edges in the directed graph containing s and t). We can perform binary search on p_{up} using our up-down oracle, i.e., every time we construct one attack graph, we can reduce the range of possible values for p_{up} by a factor of 2, and we need to call the up-down oracle only m times. This is Turing reduction as we use the up-down oracle polynomial number of times to get the exact value of p_{up}, and the construction of m attack graphs is also polynomial. By Turing reduction, the up-down deciding problem behind the oracle is at least as hard as calculating p_{up}; however, p_{up} is #P-hard to computes (Theorem 1); therefore, the up-down deciding problem is #P-hard. Since the attacker's optimal policy is at least as hard as the up-down problem; therefore, the attacker's optimal policy is also #P-hard to calculate. □

Corollary 1. *The defender's value and policy are both #P-hard to calculate.*

Proof. The defender's value is the attacker's chance of reaching DA (assuming the attacker plays the optimal policy) under the best defense. Therefore, we can reuse the same reduction for the defender's problem. The defender's policy with a budget of 1 is the same as the attacker's policy, since the attacker wants to use the same last edge the defender wants to block. Let us assume that the defender's budget is one for the defender's policy. Now, the defender has 2 paths to block, 1⟶DA or 2⟶DA, and assuming only two edges t⟶DA and 2⟶DA are blockable. Therefore, the defender's policy is also at least as hard as the up-down deciding problem, which is #P-hard and hence, the defender's is also #P-hard to calculate. □

[7]This is a simple reduction from the known #P-complete $s - t$ connectedness problem to the problem of calculating the attacker's value.

6 Proposed FPT Kernelization

This section discusses the proposed kernelization technique that reduces the original graph to a condensed graph. *Kernelization* is a common technique for fixed-parameter analysis that processes a problem and reduces it to a smaller equivalent problem, called *kernel.* We can consider an attack graph as a tree with h extra edges, known as *feedback edges (h)* and $h = m - (n - 1)$. *Splitting nodes (t)* are the nodes with more than one outgoing edge. DA does not have any successor in the graph, and even if DA have any successors, that can be ignored as once the attacker reaches DA, the attack ends. When $t = 0$, the attack graph is exactly a tree. Also, $t \leq h$. We denote the *Number of entry nodes* using s. For kernelization, we consider one parameter of AD attack graphs, which is the number of NSPs (paths from entry nodes, or other splitting nodes).

Definition 6.1. NSP. NSP(i, j) is a path that goes from node i to j, where j is $i's$ successor and then continually moves to $j's$ sole successor, until we reach DA or another splitting node [Guo et al., 2022].

$$\text{NSP} = \{\text{NSP}(i, j) |\, i \in \textsc{Split} \cup \textsc{Entry}\}$$

where Split represents the set of splitting nodes and Entry represents the set of entry nodes. An NSP is *blockable* only if at least one of its edge is blockable. In addition, once the attacker chooses to move onto a NSP, it is without the loss of generality to assume that under the attacker's optimal policy, the attacker has to complete the NSP until the attacker (1) gets caught, (2) fails, (3) reaches DA, or (4) reaches any splitting node. Otherwise, the attacker risks getting detected without securing any new checkpoints (splitting nodes).

Definition 6.2. BW. A BW $bw(i, j)$ is any furthermost blockable edge on path NSP(i, j). Two NSPs may share the same BW edge.

$$BW = \{bw(i, j) |\, i \in \textsc{Split} \cup \textsc{Entry}, j \in \text{ Successor } (i)\}$$

Notably, for the edge-blocking strategy, we only need to spend one unit of budget on NSP(i, j); otherwise, we could simply block $bw(i, j)$ to eliminate this NSP from the attacker's consideration. Size of BW edge set $|BW|$ can be bounded as

$$\begin{aligned} |BW| &\leq s + t + h \\ |BW| &\leq s + 2h, \qquad \text{since } t \leq h \end{aligned}$$

Figure 4 illustrates an attack graph, where the entry node is s, destination node is DA, split nodes are $\{a, d, f\}$, and NSPs are $\{(s, a), (a, b, c, d), (a, e, f)\}$. The thick edges are blockable, so blockable edges are $\{(b, c), (c, d), (a, e)\}$ and BW set is $\{(c, d), (a, e)\}$. *In the original AD attack graph, there are n nodes and m edges. The kernelization technique converts the original graph into a condensed AD attack graph with only* $(|\textsc{Entry}| + |\textsc{Split}| + 1)$ *nodes and* $|NSP|$ *edges.*

7 Proposed Approach 1: Combining NN and EDO

This section first discusses the mechanism to convert the attacker's problem to a DP. We then discuss the proposed attacker's policy that trains a NN to approximate the attacker's problem. We then present the proposed defensive policy. Finally, we discuss the overall proposed NNDP+EDO attacker–defender approach.

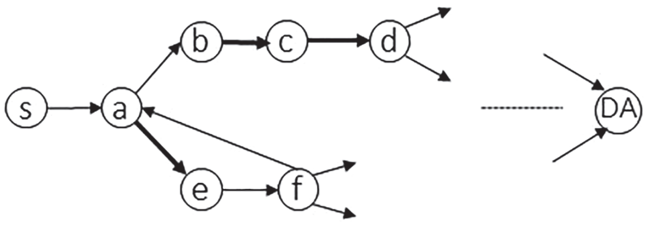

Fig. 4. Example of attack graph.

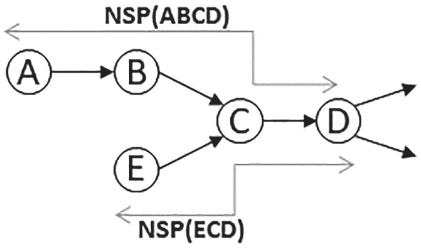

Fig. 5. Example of state transition process.

7.1 Converting Attacker's Problem to DP

We describe the attacker's problem of devising an attacking policy that maximizes the probability of reaching DA as a MDP, where the state s is a vector of size $|NSP|$ and can be represented as

$$\text{Attacker state s} = \underbrace{< S, F, ?, ?, ?, S, F, ?, ?, ?, ?, ?, S, F, F >}_{\text{Length = Number of NSP}} \tag{1}$$

where

"S" represents that the attacker has tried NSP and it is successful (the attacker has reached at the end of NSP)
"?" represents that the attacker has not yet attempted the NSP
"F" represents that attacker has tried NSP and failed (not detected).

Given a state vector as shown in Equation (1), the attacker tries one of the NSP (action for a state) with status "?" (not attempted) in order to reach DA. The realization of the NSP that the attacker tries can turn out to be either successful, fail, or detected. Accordingly, the status of that NSP changes to "S" or "F," and the attacker gets a new state. However, if the attacker is detected, the attack ends. All the NSPs that the attacker has tried and are successful act as checkpoints for the attacker. The attacker can explore an unexplored edge from these checkpoints in the future. In this way, the attacker tries unexplored NSPs until the attacker reaches DA or is detected.

7.1.1 State Transition Function. For a state and action (actions for a state s are the unexplored NSPs outgoing from the end node of successful NSPs in state s), we may have a distribution of future states. We have described the state transition process using Figure 5. For simplicity, assume that every edge in Figure 5 has $p_d = 0.1$ and $p_f = 0.2$. We have two NSPs, <NSP(ABCD), NSP(ECD)> visible in our figure, so we focus on these two NSPs for presentation purposes. The initial state vector is $< ?, ? >$. We can try one of the NSPs. Let us try NSP(ABCD), and we go through each edge on this NSP.

If AB fails, the status of its NSP changes to "F," and the new state is <F, ?>. This state is added to the future state set with 0.2 transition probability. There is 0.1 probability the attack ends and 0.7 probability of successfully passing through edge. If edge AB succeeds but BC fails, <F,?> state is already in set; therefore, its transition probability is updated to (0.7 × 0.2 + 0.2) = 0.34. There is (0.7 × 0.1) = 0.07 probability that the attack ends and the probability of successfully passing through edge is (0.7 × 0.7) = 0.49. If edge AB succeeds, BC succeeds, but CD fails: CD is a common BW edge between NSP (ABCD) and NSP (ECD). If CD fails, both the NSPs sharing CD fail, and the new state is <F, F>. The state is added to the future state set with (0.7 × 0.7 × 0.2) = 0.098 transition probability. There is (0.7 × 0.7 × 0.1) = 0.049 probability that the attack ends, and the probability of successfully passing through edge is (0.7 × 0.7 × 0.7) = 0.343. If AB succeeds, BC succeeds, CD succeeds: All the edges on NSP (ABCD) have been explored, and we can successfully go through this NSP. Therefore, we have a new state <S, ?>. This state is added to the future state set with 0.343 transition probability. On trying NSP (ABCD), we get three future states and their corresponding transition probabilities, $\{(< F, ? >,\ 0.34), (< F, F >,\ 0.098), (< S, ? >,\ 0.343)\}$.

State Transition Rules. Two or more NSPs may share the same BW edge, making the state transition process complicated. In Figure 5, there is one splitting node D and two NSPs, i.e., $ABCD$ and ECD. However, the two NSPs are not independent as they share the same BW edge CD. We design the following rules for the state transition:

(1) If $NSP(ABCD)$ is successful, we change the status of all unexplored NSPs ending at node D to successful. Since we already have access to node D, we do not have to try these NSP(ECD). We can directly change the status of $NSP(ECD)$ to "S."
(2) If all the NSPs that end on a split node D fail, then the status of all the NSPs from the split node D is changed to "F."
(3) If a $NSP(ABCD)$ fails, then it may or may not affect the status of other NSPs ending or originating from node D. It depends on which edge on the $NSP(ABCD)$ fails. The failure on $NSP(ABCD)$ may be due to any edge AB, BC, or CD. If edge AB or BC fails, then it will not affect the status of $NSP(ECD)$, and we can try $NSP(ECD)$ later; however, if the common BW edge CD fails, then the status of all the NSPs sharing this BW edge is changed to "F." Therefore, we need to determine where exactly the failure occurs and if this failure will affect the other NSPs or not.

During state transition, for each new state s, we first check if it is already a determined state or not. Determined states are the states which are already present in the future state set, and for each determined state, we have identified set of actions that can be performed on that state. If the state is not determined yet, we compute *admissible set of actions* $A(s)$, for this state. Admissible set of actions are the actions available for the state, i.e., set of unexplored NSPs that can be explored from the checkpoints of this state. Notably, admissible actions only include unexplored NSPs and ignore those NSPs for which we do not have access to their source node. The maximum size of the attacker's state space can be $3^{|NSP|}$, which is very large; however, not all state vectors are possible or relevant for the attacker. We only consider the state vectors that are relevant for the attacker by following the state transition process; given an initial state and admissible set of action, we determine the future states that an attacker can encounter on the way to DA, and we call these states as important states. Moreover, Guo et al. [2022] showed that the attack paths in AD graphs are usually short, which indicates that there are only a limited number of future states to DA. The state transition process reduces the size of the state space to a great extent. We can solve the attacker's problem using the DP technique. For a given state s and action a from *admissible set of actions* $a \in A(s)$, the attacker problem of maximizing the probability of reaching DA can be written as

$$V(s) = \max_{a \in A(s)} \left(\sum_{s'} Pr(s' \mid s, a)\, V(s') \right), \tag{2}$$

where $V(s)$ denotes the value function for the current state s, i.e., probability of reaching DA when the attacker is in state s, s' denotes the distribution of future states that follows after choosing action a and $Pr(s' \mid s, a)$ denotes the state transition probability of s', when an action a is performed on s. Equation (2) can be solved by computing the value functions for smaller problems and the overall value function step by step. Nevertheless, backward induction can be computationally challenging for large state spaces. Therefore, we use NNs to approximate the DP function.

7.2 Attacker's Policy: Neural Network-Based Dynamic Program (NNDP)

We use NN to approximate the attacker's problem, and it acts as a fitness function for the EDO. Given a blocking plan, NN outputs a value that indicates the attacker's chances of reaching DA. For approximating the attacker's value function, we first supervise NN to learn the DP base states. Base states are the states where the status of NSPs that end at DA is either "S" or "F." NSPs that end at DA with status "S" are always 100% successful, and therefore, the value of these states is 1; the attacker will reach DA. If all the NSPs leading to DA have status "F," the attack fails 100%; therefore, the value for these states is 0. Once the attacker reaches DA, there is no state transition, as the attack ends immediately. For other states, we train the NN to learn the recursive relationship of Equation (2). We select an initial state vector (actual state vector corresponding to a blocking plan; the state where some coordinates are "F," which are the NSPs blocked by the blocking plan and the rest of the coordinates are "?") as shown in Equation (1) and generate a batch of future states to train NN. Given an initial state, NN makes an optimal decision (according to the NN model) with 0.5 probability or makes a random decision with 0.5 probability to explore other possibilities. It is possible that the action performed by NN is not optimal; therefore, we employ randomness to explore other actions as well. After performing an action, we may have a distribution of future states and their transition probabilities. We select one of the future states weighted by its probability. Similarly, we keep moving to other states until we reach the base state. We train the NN to learn the recursive relationship between states and minimize the **mean squared error (MSE)** of the estimated results of the value function. Let $V(s;\theta)$ be the value predicted by the NN, where s represents the input state vector, and θ be the model parameter. Ideally, $V(s;\theta)$ is exactly a DP function. The loss is computed as follows:

$$Loss = \sum_{s \in S} \left(V(s;\theta) - \max_{a \in A(s)} \left(\sum_{s'} Pr(s' \mid s, a)\, V(s') \right) \right)^2, \tag{3}$$

where S is the set of all states. It is impractical to compute the gradient for all states in one iteration; therefore, we adopt a common approach of performing gradient descent on a mini-batch of data to train the NN. As the NN value function gets better, there are higher chances that the generated states are optimal or near-optimal, which indicates that the attacker may go via these states to DA. In addition, given a deep enough NN model and unlimited time, this will give us optimal results.

7.3 Defender's Policy: EDO

The defender uses EDO to block k BW edges in order to minimize the strategic attacker probability of reaching DA. NN acts as a fitness function for EDO, and the fitness function computes the attacker's probability of reaching DA, for a given blocking plan. The defender uses EDO to obtain a diverse set of defensive blocking plans to train NN, with an aim to improve the accuracy of trained NN for modeling the attacker. We have only considered the BW edges for the defensive policy. The

Algorithm 1: Evolutionary Diversity Optimization based Defender's Policy

```
1: Initialise population P with $\mu$ defensive plans
2: Select individual $p'$ uniformly at random from $P$ and create an offspring $p'_{new}$ by mutation or
   crossover
3: If $(OPT - 0.1) \leq fitness(p'_{new}) \leq (OPT + 0.1)$, add $p'_{new}$ to $P$
4: If $|P| = \mu + 1$, remove individual $r$ from $P$ that contributes least to diversity, i.e., minimum
   $SortedDiver(C(bw)\backslash p_r)$
5: Repeat steps 2 to 4 till the termination condition is met
```

Algorithm 2: Mutation Operator

```
Input: Population $P$, bits to mutate $x$
1: $p' \leftarrow RandomSelect(P)$
2: $p'' \leftarrow$Flip $x$ $Bs$ to $Ns$ and $x$ $Ns$ to $Bs$ in $p'$
3: return $p''$
```

defensive state vector can be represented as:

$$\text{Defensive plan} = \underbrace{< N, B, \ldots, B, N, B >}_{\text{Length = Number of block-worthy edges}} \tag{4}$$

where

"B" represents blocked edges
"N" represents non-blocked edges

EDO. Algorithm 1 outlines the overall defender's policy. We initially generate a random population P of μ defensive blocking plans, and each blocking plan is a vector of size $|BW|$. The values in a state vector are either B or N, and the number of Bs is always equal to k (defensive budget). We randomly select individuals from P for mutation and crossover. We also draw a random number x from the Poisson distribution with a mean value equal to 1 and perform either mutation or crossover with a probability of 0.5 to create new offspring. Our mutation and crossover operators ensure that the number of blocked edges in an offspring does not exceed k. The mutation and crossover operations are discussed in detail below:

Mutation. We pick a random individual p' from the population P and flip x Ns to Bs and x Bs to Ns. Algorithm 2 presents the mutation operator. For example, we pick a random individual $p' = < B, N, N, N, B, N, N, B >$ from the population P. To mutate p', we flip two Ns to Bs and two Bs to Ns. New individual obtain is: $< N, B, N, N, B, N, B, N >$.

Crossover. We pick two random individual state vector p' and p'' from the population P to crossover. We find x coordinates where p' has Ns on those coordinates, and p'' has Bs on those coordinates. For these coordinates, we change Ns in p' to Bs and change Bs in p'' to Ns. We then find x coordinates where p' has Bs on those coordinates and p'' has Ns on those coordinates. We again change Bs to Ns and Ns to Bs. Algorithm 3 presents the crossover operator.

Maximizing Blocked Edges Diversity. We add the newly created offspring to the population only if its fitness score lies within the range of $(OPT \pm 0.1)$; otherwise, the offspring is rejected even though it is beneficial for diversity. If the new offspring is added to the population, we aim to maximize the blocked edge diversity and remove the individual that contributes least to the diversity. We define the diversity measure as "all block-worthy edges are equally blocked." Our proposed diversity measure aims to maximize the diversity of blocked edges in the defensive plan population. It

Algorithm 3: Crossover Operator

```
Input: Population $P$
1:  $p', p'' \leftarrow RandomSelect(P)$
2:  for $i = 1, ..., len(p')$ do
3:      if $(p'[i] == 0 \wedge p''[i] == 1)$ then
4:          $C.append(i)$
5:      end if
6:  end for
7:  $C'$ = Randomly select $x$ bits from $C$
8:  for all $i \in C'$ do
9:      set $p'[i] = 1$ and $p''[i] = 0$
10: end for
11: for $i = 1, ..., len(p')$ do
12:     if $(p'[i] == 1 \wedge p''[i] == 0)$ then
13:         $D.append(i)$
14:     end if
15: end for
16: $D'$ = Randomly select $x$ bits from $D$
17: for all $i \in D'$ do
18:     set $p'[i] = 0$ and $p''[i] = 1$
19: end for
20: return $p'$
```

calculates how often each BW edge is blocked in the population, with the aim of making this frequency equal. When a new offspring is created using mutation or crossover, the offspring is added to the population only if its fitness score is close to the best fitness score and rejects the individual that contributes least to the diversity. Let us assume there are μ individuals in P. Each individual p_i can be described as:

$$p_i = \big((B/N, bw_1), (B/N, bw_2), ..., (B/N, bw_{|BW|})\big),$$

where B/N denotes the status of BW edge; "B" indicates blocked, "N" indicates non-blocked, and $i \in \{1, ..., \mu\}$. We then compute the BW edge count vector $C(bw)$, which is defined as "for each block-worthy edge bw_j, $j \in \{1, ..., |BW|\}$, the number of individuals who have blocked bw_j edge."

$$C(bw) = (c(bw_1), c(bw_2), ..., c(bw_{|BW|})),$$

where $c(bw_1)$ denotes the total individuals out of μ who have blocked bw_1 edge. For each individual p_i, we then determine the vector $Diver(C(bw)\backslash p_i)$, which computes the diversity of the population without individual p_i as

$$Diver(C(bw)\backslash p_i) = C(bw) - p_i.$$

Our goal is to maximize the blocked edge diversity; therefore, we calculate $SortedDiver$ $(C(bw)\backslash p_i)$ as

$$SortedDiver(C(bw)\backslash p_i) = sort\Big(Diver(C(bw)\backslash p_i)\Big).$$

To maximize the diversity of blocked edges, our goal is to minimize the $SortedDiver(C(bw)\backslash p_i)$ in lexicographic order where sorting is carried out in descending order. The individual l with minimum $SortedDiver(C(bw)\backslash p_l)$ is the one, removal of which maximizes the diversity. Therefore,

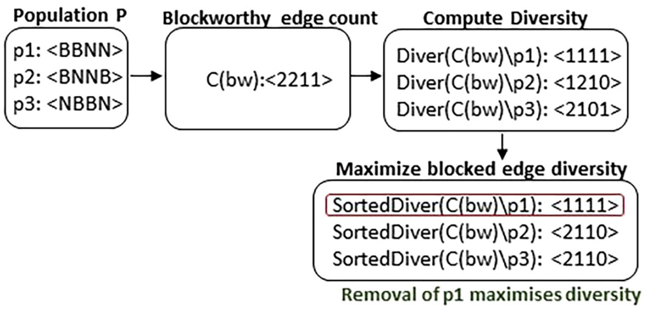

Fig. 6. Example of maximizing blocked edge diversity.

the individual l is removed from the population, if its removal maximizes the diversity and its fitness score is not close to the best. In special cases, when the newly created offspring has the best fitness score, we add it to P (even though it is worst in terms of diversity), and the individual with the lowest fitness score is discarded. Using this process, the defender generates a diverse population of defensive plans. Figure 6 presents an example of maximizing the blocked edge diversity in the population.

Converting Defensive Blocking Plan to Actual State Vector. For each defensive blocking plan vector (Equation (4)), we first need to convert it to the *actual state vector* (Equation (1)) in order to determine the attacker's chances of reaching DA (corresponding to this blocking plan). "Actual state vector" is the state vector used in the attacker's MDP. For each BW blocked edge *bw* in the defensive state vector, we first determine the number of NSPs blocked by this *bw* edge and then change the status of those NSPs to "F," keeping the status of the rest to "?."

7.4 Overall Approach: NNDP+EDO

Initially, we have a NN which is highly inaccurate (attacker's policy). We generate a diverse set of blocking plans as training data for NN using EDO (Defender's policy), and NN acts as an efficient fitness function for EDO. Notably, the exact fitness function is #P-hard to compute; therefore, we use NN as a fitness function. We convert the blocking plans to the actual state vectors and in each training epoch, select a random blocking plan out of the population to train NN, which we call an initial state vector. We use the state transition process to determine the future states set, transition probabilities and admissible action set. We then train NN using Equation (3) on the future state set to approximate the value function, i.e., NN outputs the attacker's probability of reaching DA with the given blocking plan. We repeatedly train NN on diverse blocking plans aiming to improve the accuracy of NN for modeling the attacker. We repeatedly go back and forth between training NN and EDO processes for many rounds to get a well trained NN that can act as an efficient fitness function for EDO. In the last round, we train NN once again on the population obtained from the last round of EDO. EDO prevents the NN model from getting stuck into local optima too early, especially in the early phases of the training when the value function is highly inaccurate. In addition, not all states are important for the attacker; therefore, we let the value of states that are referenced by the attacker's optimal decision path, be more accurate by training NN on these states.

8 Proposed Approach 2: Combining RL and EDO

This section outlines our second proposed approach for defending AD graphs. We first present our proposed RL-based attacker's policy. Following that, we discuss the C-EDO-based defender's policy. Later, we describe our overall attacker–defender approach.

8.1 Attacker Policy: RL

The attacker's goal is to design a strategy that maximizes their chances of successfully reaching the DA. We describe the attacker's problem as an MDP and propose a RL-based policy to address the attacker's problem. Our proposed RL-based attacker policy uses *PPO* algorithm, an actor–critic-based approach, to train the RL agent. *We train our RL agent in parallel against multiple instances of environment configurations at a time, and each environment contains a defensive plan from the defender.* Training the RL agent in parallel on numerous environments speeds up the training process as the agent is able to learn from various defensive plans at a time. Moreover, the shared experience that the RL agent gains from the different environments can be used to make better decisions and achieve a higher reward in the final stages of the game. This section discusses our proposed RL-based attacker's policy in detail.

8.1.1 Environment. We model the attacker's problem of designing a policy to maximize their chances of successfully reaching the DA as MDP. We call the attacker's MDP $M = (S, A, R, T)$ as an environment, where S denotes the state space, A denotes the action space, R is the reward function, and T represents the transition function. The description of the environment is discussed below.

State Space (S). State space S is a finite set of attacker's states, and state "'s" is a vector as described in Equation (1). At any time t during the attack, the attacker's current state s_t acts as a knowledge base and conveys the set of NSPs that the attacker has control of, NSPs that the attacker has failed on and NSPs that the attacker can try in future. The attacker has two base states: (1) When the attacker reaches DA, the attack is successful and terminates; (2) When the attacker is not able to reach the DA; the reason can be that they got detected or has tried all possible NSPs, and there is no option left to explore; in this case, the attack fails and ends.

Action Space (A). Action space A is the action set available for state s, which are the outgoing NSPs from the successful NSPs in state s. The attacker's action space is discrete. Action a is one of the NSPs from the action space of state s and indicates that the attacker tries this NSP to reach the DA.

Reward Function (R). The reward $r(s, a)$ for state s on performing an action a is 1 if the attacker is able to reach the DA without getting detected. Otherwise, the reward is 0.

Transition Function (T). For a given state s and action a, the transition function performs action a on state s, and may have a set of future states. Each future state is associated with its transition probability, and one state is selected as the next state (weighted by its transition probability). We discussed the details of the state transition function in Section 7.1.1.

8.1.2 Policy Training. We propose to utilize *PPO* RL algorithm to train the attacker's policy so as to maximize the attacker's success rate. PPO follows an *actor–critic approach* that exploits the advantages of policy-based and value-based approaches while eliminating their disadvantages. In this approach, the policy and value networks help each other improve. In our approach, we train two networks: the actor network and the critic network. *Actor network*, also referred to as policy network, takes the current state s as input and outputs the action a to be performed on s. The actor network updates the policy parameters in the direction implied by the critic network. *Critic network*, also known as value network, takes the state as input and outputs the value of state. *For an attacker problem, value of the state is the attacker's success rate.*

Our RL agent uses the actor–critic-based PPO algorithm to train the attacker's policy by interacting with the environment (each environment is associated with a configuration, i.e., defensive plan). The RL agent does not possess any prior knowledge of the environment. Instead of training the RL agent against a single environment, we train the agent against *multiple parallel environments.* Each environment is initialized with the attacker's initial state (defensive plan is converted to obtain attacker's initial state). The following process is executed in all environments simultaneously. At each timestep t of an episode, the RL agent receives state s_t. The trained actor network issues an action a_t to be performed on the current state s_t, and action a_t is sent to the environment. The environment performs action a_t on state s_t and reaches a new state s_{t+1} (following the transition function) and receives a reward r_{t+1} (following the reward function). The process repeats until we reach the base state, i.e., the attacker reaches DA or gets detected. In this manner, we obtain a sequence of states, actions, and rewards that terminates at the base state. The designed policy intends to maximize the total reward received during an episode. The final reward is the attacker's success rate. *This way, the attacker designs RL-based policy to maximize their achievable reward (success rate).* Training the RL policy in multiple environments at a time helps the attacker's policy to learn quickly from multiple defensive plans at a time. As a result, the training process is more efficient, due to which our proposed approach is scalable to large AD graphs as well.

8.2 Defender Policy: C-EDO

The defender's goal is to block k BW edges to minimize the attacker's chances of successfully reaching the DA. We propose a C-EDO-based defensive policy that computes high-quality and diverse environmental configurations (defensive plan). We aim to identify the valuable environments, i.e., the environment configurations that correspond to potentially good defense. Our main idea is to let the RL agent play against the environment configuration, and if, after training for some time, the configuration proves to be unfavorable for the defender (i.e., the attacker has a high success rate against the configuration), we discard this environment configuration. We do not waste our computational effort on this environment and allocate our limited computational resources to other challenging environments. The high-quality and diverse characteristics of environments enhance the accuracy of modeling the attacker.

In contrast to our previously proposed defender's policy outlined in Section 7.3, which employed a NN as the fitness function for evaluating the defensive plans, our current defensive policy takes a different approach. In this, the trained RL critic network serves as a fitness function for the defender's C-EDO. The defender only blocks BW edges to generate environment configuration. The defender's environment configuration is represented as shown in Equation 4, and we follow the same procedure to generate the environment configuration (defensive plan) population as discussed in Section 7.3; however, instead of NN, we use the trained RL critic network as the fitness function in this approach.

8.3 Attacker–Defender Overall Approach

The defender employs C-EDO to generate high-quality and diverse environment configurations, each containing a defensive plan worth learning for the attacker policy. The attacker uses an actor–critic-based RL algorithm to train their policy against the defender's environment configurations. The attacker's trained RL critic network serves as a fitness function for the defender C-EDO. The attacker first converts the defensive plans in environment configurations (Equation (4)) to the attacker's initial state (Equation 1)). The RL agent then interacts and learns from the environments in parallel by issuing actions according to the trained policy. The environments perform the action and return a new state and reward. The quality of the trained policy is determined based on the total reward collected by the agent during an episode. Initially, the policy is not trained, and the action suggested might result in low rewards, but with training, it results in high rewards (attacker's success rate). In this way, the RL agent trains the policy to maximize the reward. The RL learning process is continuous and is not disrupted by any other process.

Besides, our defensive policy is based on the idea of "replicating the environment configurations that perform well for the defender and diminishing the ones that do not." The defender process

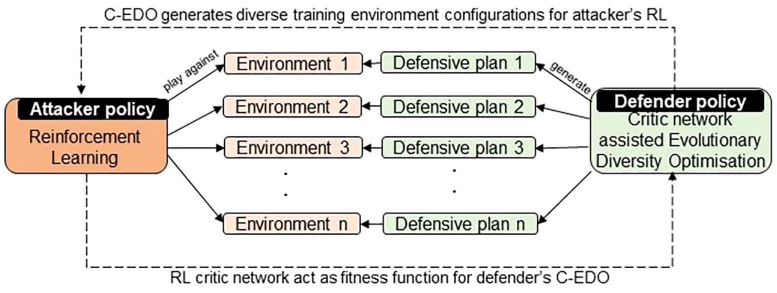
C-EDO generates diverse training environment configurations for attacker's RL
play against
Environment 1
Environment 2
Environment 3
Environment n
Defensive plan 1
Defensive plan 2
Defensive plan 3
Defensive plan n
generate
Attacker policy
Reinforcement Learning
Defender policy
Critic network assisted Evolutionary Diversity Optimisation
RL critic network act as fitness function for defender's C-EDO

Fig. 7. Overall attacker–defender proposed approach.

continuously monitors the RL training process and, after every regular interval, uses the trained critic network to evaluate the current set of environments. The defender discards those environments that are good for the attacker and replaces them with better ones. This way, at the beginning of every episode, the RL agent checks if the defensive plan configuration corresponding to each environment is changed or not. If changed, the attacker initializes the environment with the new defense configuration and starts training the agent against it. In this manner, the attacker and defender play against each other in parallel, where the attacker's RL process continuously learns, and the defender process evaluates the current environment configurations and generates better ones. C-EDO's diversity characteristic helps RL not get stuck in the local optimum. For the RL agent, the diversity in environmental configurations primarily differs only in the initial stages, referred to as opening games. However, the later stages, including end-games or mid-games, tend to be similar across environments. Therefore, we utilized the similarity in later stages by training the agent against multiple environments in parallel, due to which our RL agent gains shared experience, resulting in faster convergence and enhanced performance. The RL agent experiences differences in environmental configurations only in the early stages, but the later stages tend to be similar across environments. We took advantage of this similarity and trained the agent on multiple environments in parallel, due to which our agent gained shared experience, resulting in faster convergence and enhanced performance. Overall, the trained attacker's RL policy improves as the defender generates better environmental configurations using C-EDO. The defender's policy generates better environments as the attacker's critic network is trained. This way, these two processes assist each other to perform better. Figure 7 illustrates our overall proposed approach.

9 Experimental Results

We executed experiments on a high-performance cluster server with Intel Gold 6148/6248 CPUs, utilizing 1 CPU and 1 Core for NNDP+EDO approach; and 1 CPU and 20 cores for RL+C-EDO approach, for each trial. We implemented the code in PyTorch. We used OpenAI Gym [Brockman et al., 2016] to implement the RL environments and trained the RL model using the PPO algorithm from the Tianshou library [Weng et al., 2022]. In our results, "Success Rate" represents the attacker's chances of successfully reaching DA before getting detected (given the attacker's policy) under the defensive blocking plan from the defender's policy. "Time (hour)" is the number of hours per trial.

9.1 Dataset

Real-world AD graphs are highly sensitive; therefore, we used BloodHound team's synthetic graph generator DBCreator to generate synthetic AD graphs. We generate AD graphs of four sizes, i.e., $R500$, R1000, $R2000$, and $R4000$, where 500, 1,000, 2,000, and 4,000 indicate the number of computers in the AD graph. R500 AD graph contains 1,493 nodes and 3,456 edges; R1000 AD graph

Table 3. Summary of Synthetic AD Attack Graphs

AD attack graph	Nodes	Edges	Max out degree	Number of splitting nodes	Attacker's state cardinality
R500	1,493	3,456	3	4	24
R1000	2,996	8,814	4	9	35
R2000	5,997	18,795	4	15	42
R4000	12,001	45,780	6	37	111

contains 2,996 nodes and 8,814 edges; $R2000$ AD graph contains 5,997 nodes and 18,795 edges; $R4000$ AD graph contains 12,001 nodes and 45,780 edges. Table 3 presents the summary of synthetic AD attack graphs. Our experiments consider only three specific kinds of edges, by default present in BloodHound: HasSession, MemberOf and AdminTo. To set the attacker's starting nodes, we first find 40 faraway nodes from DA and then arbitrarily set 20 nodes from them as the starting nodes. Each edge e is blockable with a probability:

$$\text{Probability} = \frac{\text{Min \#hops between e and DA}}{\text{Max \#hops between any e and DA}}$$

This way, the edges farthest from the DA, i.e., less significant edges, are more likely to be blocked. It is challenging to perform computations on a large AD graph; therefore, we pre-process it to obtain a smaller graph. Pre-processing includes merging all DA into 1 DA, removing nodes and edges irrelevant for the attacker (outgoing edges from DA, all incoming edges to the entry nodes and the nodes without any incoming edges). We also consider an NSP (Definition 6.1) as one edge.

Furthermore, to investigate the relationship between the failure rate $p_{f(e)}$ and the detection rate $p_{d(e)}$ of an edge e on the attacker's chances of success, we use three different distributions: Independent distribution, Positive correlation, and Negative correlation.

— *Independent Distribution (I).* For the *independent distribution*, the values of $p_{d(e)}$ and $p_{f(e)}$ are set as follows:

$$p_{d(e)}, p_{f(e)} = \text{Independent uniform } (0, 0.2)$$

— *Positive Correlation (P).* For the *positive correlation*, the values of $p_{d(e)}$ and $p_{f(e)}$ are set as follows:

$$p_{d(e)}, p_{f(e)} = \mathcal{N}(\mu, \Sigma),$$

where

$$\mu = [0.1, 0.1]$$
$$\Sigma = [[0.05^2, 0.5 \times 0.05^2], [0.5 \times 0.05^2, 0.05^2]]$$

Here, $\mathcal{N}$ represents the multivariate normal distribution, μ represents the mean vector, and Σ represents the covariance matrix.

— *Negative Correlation (N).* For the *negative correlation*, the values of $p_{d(e)}$ and $p_{f(e)}$ are set as follows:

$$p_{d(e)}, p_{f(e)} = \mathcal{N}(\mu, \Sigma),$$

where

$$\mu = [0.1, 0.1]$$
$$\Sigma = [[0.05^2, -0.5 \times 0.05^2], [-0.5 \times 0.05^2, 0.05^2]]$$

Again, $\mathcal{N}$ represents the multivariate normal distribution, μ represents the mean vector, and Σ represents the covariance matrix.

9.2 Training Parameters

This section discusses the training parameters used for training the NN and RL.

Neural Network-Based Dynamic Program. In our approach, we use a simple fully connected NN, and a ReLU activation function follows each NN layer. For the $R500$ graph, a small NN with four fully connected layers is used, whereas, for R1000 and $R2000$, we use a NN with 10 layers. The size of each layer is 256, and the last layer of NN is followed by a sigmoid activation function that maps the model output to a value between 0 and 1 (the attacker's chances of reaching DA). We train the NN in a batch of 16 states. We use the MSE to compute the loss, and train the parameters using Adam Optimizer with a learning rate of 0.001. The model is trained for 500 epochs in each round. We set the defender budget to 5. We generate a population of 100 defensive blocking plans in 10000 iterations and perform mutation or crossover with a probability of 0.5. We repeat the combined process (training NN and generating blocking plans using EDO) for 100 rounds. We perform experiments with a seed from 0 to 9 for 10 separate trials (we conduct experiments on 10 different AD graphs with different entry nodes and different blockable edges).

Reinforcement Learning. We used a simple fully connected multi-layer NN to implement the actor and critic networks. The hidden layer size is 128 for R500 and R1000 AD graphs, and 256 for $R2000$ and $R4000$ AD graphs. The parameters are trained using Adam optimizer, learning rate of $1e^{-4}$ and batch size of 800 states. For PPO-specific hyper-parameters, we used the standard hyper-parameters as specified in the original article [Schulman et al., 2017]. We created 20 environments. For experimental setup 1, we parallelly train the RL policy for a total of 700 epochs (1,200 epochs for $R2000$ and $R4000$ AD graph) on 20 environments. After every 20 training epochs, the defender evaluates and resets the environments. When the terminating condition is met (number of epochs), the defender chooses the defensive plan with the lowest attacker success rate as their *best environment configuration*. For experimental setup 2 and 3, we train the RL policy for 150 epochs on 20 environments parallelly. The trained attacker's policy is then simulated on the best environment configuration for $5,000$ episodes, and the average reward over $5,000$ episodes is the attacker's success rate. Notably, we intensively train the RL-based attacker's policy to obtain a good attacking strategy, as the effectiveness of defensive plans depends on the attacker's model accuracy. For the defender's policy, the defender can only block 5 edges. In $20,000$ iterations, the defender creates a population of 20 environment configurations (defense).

9.3 Performance Evaluation: Experimental Setup 1

This section evaluates the performance of proposed NNDP+EDO and RL+C-EDO approaches and reports the results in comparison to the baselines.

9.3.1 Baselines. We compare our proposed approach with a combination of various attacking and defensive policies, and the details are described below:

(1) *NNDP+EDO (Proposed).* NNDP approach is used to approximate the attacking policy and EDO for the defensive policy. The defensive plan that contributes least to the diversity is rejected, and the fitness value is evaluated using NNDP.

(2) *RL+C-EDO (Proposed).* In this approach, RL is utilized as the attacker's strategy, whereas C-EDO is used as the defender's policy. In C-EDO, the defender rejects those defensive plans that contribute least to diversity.
(3) *NNDP+EDO+DP.* NNDP is used to approximate the attacking policy and EDO for defensive policy. However, instead of using NNDP to evaluate the success rate of the best blocking plan (as in our proposed approach), we use accurate DP to determine the value of the defensive plan.
(4) *Exact Solution.* A DP is used as the attacker's policy, and a defensive blocking plan is obtained by exhaustively trying each in order to get the best plan.
(5) *NNDP+EC.* In this approach, NNDP is used to approximate the attacking policy, and EC is used to design the defensive policy. In EC, the blocking plans with the worst fitness values are rejected, and the fitness value is evaluated using NNDP.
(6) *RL+EC.* In RL+EC approach, RL is utilized as attacker's policy, and EC is used as the defender's policy. In EC, the defensive plan with the worst fitness score are discarded.
(7) *NNDP+Greedy.* The attacker uses NNDP to approximate the attacking policy, and the defender adopts a greedy approach to design the defensive policy. The defender greedily blocks the single best BW edge to minimize the attacker's chances of reaching DA using NNDP. The defender repeats this for k times.
(8) *RL+Greedy.* In RL+Greedy approach, RL is utilized as attacker's strategy. The defender uses a greedy technique to generate defensive policy and greedily blocks one edge that minimizes the attacker's success rate; and iteratively discovers k edges to be blocked.

Notably, NNDP+EDO+DP and Exact solution use DP to evaluate the blocking plan; however, it is infeasible for DP to process large graphs. Therefore, for the $R500$ graph, we compare our proposed approach NNDP+EDO, with NNDP+EDO+ DP and the Exact solution. For larger graphs $R1000$ and $R2000$, we compare our proposed approaches NNDP+EDO, RL+C-EDO with NNDP+EC, NNDP+Greedy, RL+EC and RL+Greedy to determine our defensive strategies' effectiveness. NNDP attacker's policy is not scalable to $R4000$ AD graphs; therefore, we only report the results obtained from RL-based attacker's policy.

For NNDP attacker's policy, the trained NNDP may not give us accurate values of the success rate for the defensive plan; therefore, we use Monte Carlo simulations to determine the effectiveness of the defensive plan. We first perform 100 rounds of NNDP+EDO and as a result, we have a trained NN (attacker's policy) and a set of blocking plans (defender's policy). We simulate the attacker's policy (NNDP) on the best defensive blocking plan (predicted by NNDP) using Monte Carlo simulations over 100,000 runs to determine attacker's chances of reaching DA. We performed all the experiments ten times with a seed from 0 to 9, and the average results over all seeds are presented as final results.

For RL attacker's policy, we first train the RL-based attacker's policy on the environment configurations (defensive plan) generated by the defender's policies, i.e., C-EDO, EC, and Greedy. We then test the effectiveness of the attacker's policy against the defender's best environmental configurations. We report the average success rate by simulating the attacker's strategy on the best environment for 5, 000 episodes. For each AD graph, we perform experiments on 5 seeds from 0 to 4, and report the average success rate over 5 seeds.[8]

9.3.2 Results on R500. Table 4 presents the success rate of the proposed approach NNDP+EDO and other baselines on synthetic $R500$ graph under various distributions. The attacker's average success rate is 87.96% when simulated using Monte Carlo over 100,000 runs with our proposed

[8]We opted to perform experiments on 5 seeds due to the high training time required for training the RL-based attacker's policy.

Table 4. Comparison of Attacker's Chances of Success under Various Attacker–Defender Policies on *R*500 AD Attack Graph

Graph	Approach	I	P	N	Average
	NNDP+EDO (Proposed)	87.96%	84.77%	84.5%	85.74%
R500	NNDP+EDO + DP	90.49%	84.8%	84.46%	86.58%
	Exact solution	89.73%	84.78%	84.45%	86.32%

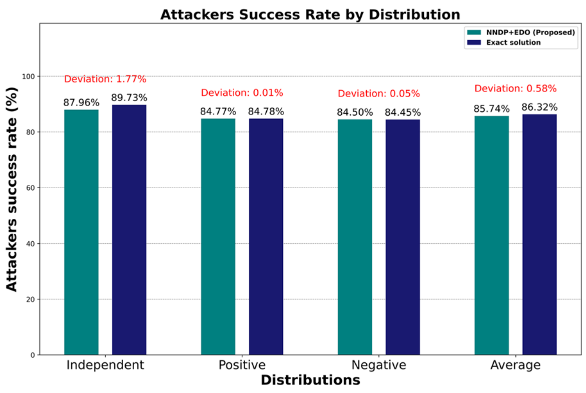

Fig. 8. Deviation of proposed NNDP+EDO approach from the exact solution for R500 AD graph across various distributions.

defense EDO and NNDP (under independent distribution). Moreover, the exact actual success rate for our proposed defense when evaluated using NNDP+EDO+DP is 90.49%. This indicates that for a blocking plan from EDO, our trained NNDP generates an error of 2.53% in the success rate. For exact optimal defense, the attacker's success rate is 89.73%, which indicates that our proposed defense is 0.76% (less than 1%) away from the optimal. Similarly, the proposed defense NNDP+EDO performs near-optimal under positive correlation; NNDP+EDO believes the best defense has a success rate of 84.77%, but in reality, the accurate success rate of the proposed defense is 84.8%, which is slightly worse than the exact solution 84.78%. In negative correlation as well, NNDP+EDO defense is nearly as effective as optimal. Figure 8 shows the deviation of proposed NNDP+EDO approach from the exact solution across various distributions.

9.3.3 Results on R1000. Table 5 presents the results for R1000 AD attack graph. It is impossible to determine the exact attacker's success rate for R1000 graph using DP; therefore, we use NNDP and RL to approximate the attacker's policy. Bold results indicate the best results among the proposed and all baselines.

For NNDP attacker's policy, results show that, on an average, the proposed EDO-based defense NNDP+EDO leads to the best defensive policy as compared to the other defense (NNDP+EC and NNDP+Greedy). For instance, under independent distribution, the attacker's success rate is 42.69% with EDO defensive policy; however, the success rate increases to 43.19% and 53.89% when facing EC defense and Greedy defense, respectively. Under positive correlation, NNDP+EDO is the best defense among the three and leads to minimum attacker's success rate. However, under a negative correlation, the EC defense is slightly better than EDO. Corresponding to NNDP attacker's policy, the results show that overall EDO's best defense has an average attacker's success rate of 42.14%; EC best defense has a success rate of 43.04%, which is slightly worse than EDO. The greedy defense has an average success rate of 52.35%, which is far worse than the EC defense.

Table 5. Comparison of Attacker's Chances of Success under Various Attacker–Defender Policies on R1000 and R2000 AD Attack Graph (Smaller Number Represents Better Performance)

		Success rate				Time (hour)		
Graph	Approach	I	P	N	Average	I	P	N
R1000	NNDP+EDO (Proposed)	**42.69%**	**42.44%**	41.29%	**42.14%**	10.94	16.73	17.8
	NNDP+EC	43.19%	44.7%	**41.24%**	43.04%	11.05	17.1	16.03
	NNDP+Greedy	53.89%	52.86%	50.31%	52.35%	10.67	17.93	18.73
	RL+C-EDO (Proposed)	**40.16%**	**41.36%**	**41.51%**	**41.01%**	43.82	44.73	47.27
	RL+EC	41.69%	48.45%	42.89%	44.34%	44.43	43.88	48.43
	RL+Greedy	56.45%	47.86%	47.88%	50.73%	43.32	44.95	47.38
R2000	NNDP+EDO (Proposed)	**31.07%**	**33.9%**	35.56%	**33.51%**	7.58	18.19	16.62
	NNDP+EC	33.23%	36.45%	**34.19%**	34.62%	7.45	18.63	15.87
	NNDP+Greedy	38.32%	42.69%	39.02%	40.01%	6.79	18.22	15.94
	RL+C-EDO (Proposed)	**25.09%**	**32.45%**	**30.44%**	**29.33%**	112.74	118.58	119.53
	RL+EC	28.13%	35.51%	37.28%	33.64%	114.22	112.57	118.51
	RL+Greedy	33.43%	34.30%	40.06%	35.93%	113.84	116.43	113.76

Results show that the proposed C-EDO defensive policy leads to the best defense. Bold results indicate the best results among the proposed and all baselines.

For RL attacker's policy, results from the table show that under independent distribution, the attacker's chances of success under C-EDO-based defensive policy is 40.16%. In contrast, the attacker's chances of success increase to 41.69% and 56.45% under EC-based and greedy defense, respectively. Similarly, under a negative correlation, the attacker success rate is 41.51% under C-EDO defense, which is far lower than the EC-based and greedy defense. For R1000 AD graph, the average attacker's success rate is 41.01% under C-EDO-based policy, 44.34% under EC-based defense and 50.73% under greedy defense. The results prove C-EDO-based defensive policy as the best policy among the baselines.

Overall, C-EDO proves to be a more effective defense than EDO-based defense, as the attacker's success rate is minimized under C-EDO-based defense.

9.3.4 *Results on R2000.* Table 5 presents the results for R2000 AD attack graph. For NNDP attacker's policy, results show that EDO-based defense outperforms EC and Greedy-based defense in terms of average success rate. With the EDO defense (under independent distribution), there are only 31.07% chances of attacker's reaching DA; however, with the Greedy defense, the attacker's success rate increases to 38.32%. EDO performs better than EC and Greedy in both independent and positive correlation; however, EC performs slightly better than EDO in negative correlation. On an average best defense from EDO has an average success rate of 33.51%; EC has success rate of 34.62%, slightly worse than EDO, and Greedy has an average success rate of 40.01%, which is very high as compared to EDO defense.[9]

For RL attacker's policy on $R2000$ AD graphs, our results show that under independent distribution, the attacker chances of success are minimum, i.e., 25.09% when the defender uses C-EDO-based policy; the success rate increases to 28.13% and 33.43% under EC-based defense and Greedy defense, respectively. C-EDO-based defense turns out to be the best defense for R2000 AD graph as well. On

[9]For R1000 and R2000 graphs, 6/180 trails (around 3%) ended up with unconverged NN training, i.e., after 500×100 epochs, the cost function remains to be large.

Table 6. Comparison of Attacker's Chances of Success under Various Attacker–Defender Policies on R4000 AD Attack Graph (Smaller Number Represents Better Performance)

		Chances of success				Time (hour)		
Graph	**Policy**	**I**	**P**	**N**	**Average**	**I**	**P**	**N**
R4000	RL+C-EDO (Proposed)	**22.02%**	**17.29%**	**21.81%**	**20.37%**	127.54	121.11	137.73
	RL+EC	22.78%	21.87%	24.71%	23.12%	132.55	120.67	135.31
	RL+Greedy	25.16%	24.18%	22.34%	23.89%	125.61	120.73	127.29

Results show that the proposed C-EDO defensive policy leads to the best defense. Bold results indicate the best results among the proposed and all baselines.

an average, the attacker success rate is minimum, 29.33% under C-EDO-based defensive policy, 33.64% with EC-based defense, and the success rate is worst, i.e., 35.93% with greedy defense.

9.3.5 Results on R4000. Our proposed attacker's policy, NNDP, is not scalable to large AD graphs such as $R4000$; however, the RL-based attacker's policy is scalable to $R4000$ graphs. Therefore, we only report the results obtained for the RL-based attacker's policy, and Table 6 presents the results for R4000 AD attack graph (Bold results indicate the best results among the proposed and all baselines). The results on $R4000$ AD graphs show that under a positive correlation, the attacker success rate is minimum, i.e., 17.29% under C-EDO-based defense; however, the success rate increases to 21.87% and 24.18% under EC-based and Greedy defense, respectively. Overall, our results demonstrate that for all three different distributions, on average C-EDO-based defense is the best defense where the attacker success rate is minimum. Also, EC-based defense outperforms Greedy defense.

9.4 Performance Evaluation: Experimental Setup 2

In this experimental setup, we determine the effectiveness of our proposed NNDP and RL-based attacker's policy.

9.4.1 Results. NNDP attacker's policy is only scalable to $R2000$ graph; therefore, we perform experiments on $R500$, $R1000$, and $R2000$ AD graphs. We randomly generate 10 environmental configurations (defensive plans) for each AD graph. We first train NNDP-based attacker's strategy on 10 defensive plans for 2,000 epochs[10] and perform Monte Carlo simulations for 5, 000 runs to compute the attacker's success rate on each defensive plans. We then train RL-based attacker's policy on the same set of 10 environments for 150 epochs and then evaluate the trained policy for 5, 000 episodes to compute the attacker's chances of success. We reported the attacker's average chances of success over 10 environments in Table 7. Figure 9 presents attacker's average success rate under our designed attacker's policies across various graphs. For a given environmental configuration, the attacker's policy that results in a higher success rate indicates that the corresponding policy is able to approximate the attacker's problem more accurately than others. Table 7 shows that for R500 graph, the attacker average success rate is 87.98% under the RL policy, which is slightly higher than NNDP-based policy. For the R1000 graph, attacker's average chance of success is 51.96%, which is again higher than the NNDP policy. Our results show that the under the proposed RL-based strategy, attacker success rate is higher compared to the proposed NNDP-based strategy, implying that RL policy is more effective at countering defense than the NNDP policy.

[10]We opted for a higher number of epochs in NNDP training because RL training is time-consuming per epoch, whereas NNDP training is much faster. This allowed us to balance the overall training time between the two models.

Table 7. Comparison of Attacker's Chances of Success under Both the Proposed Attacker's Policies (Larger Number Represents Better Performance)

Graph	Policy	I	P	N	Average
R500	RL (Proposed)	**88.01%**	**86.58%**	**89.37%**	**87.98%**
	NNDP (Proposed)	87.57%	86.08%	89.28%	87.64%
R1000	RL (Proposed)	**54.99%**	48.21%[a]	**52.69%**	**51.96%**
	NNDP (Proposed)	53.52%	**48.32%**	52.15%	51.33%
R2000	RL (Proposed)	**45.28%**[a]	**56.41%**	**42.43%**[a]	**48.04%**
	NNDP (Proposed)	45.11%	56.29%	42.39%	47.93%

Results show that the RL-based attacker policy approximates the attacker's problem more accurately than NNDP policy.
[a]Indicates that with our general parameter settings, RL policy results were slightly bad than NNDP. Therefore, we train the RL policy for 300 epochs instead of 150 epochs. Given enough time, the RL policy outperforms the NNDP policy.

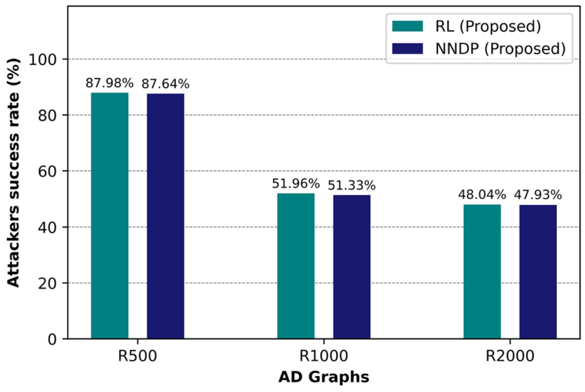

Fig. 9. Attacker's average success rate under our designed attacker's policies across various graphs (larger number represents better performance).

9.5 Performance Evaluation: Experimental Setup 3

In this experimental setup, we determine the effectiveness of our proposed EDO- and C-EDO-based defensive policies. We evaluate their performance by comparing the RL+C-EDO final defensive plan with the final defensive plan from NNDP+EDO approach.

9.5.1 Results. We run RL+C-EDO and NNDP+EDO approaches on 5 seeds from 0 to 4 to obtain the defender's best defensive plan. We train the RL attacker policy for 150 epochs on the best defensive plan from both approaches (on 5 seeds). We then evaluate the trained policy for 5,000 episodes to compute the attacker success rate. We reported the results in Table 8. A defensive plan against which the RL-based attacker policy is able to achieve a lower success rate is considered as the best defensive plan. Results in Table 8 show that the average attacker success rate for R1000 AD graph is 42.77% on the best defensive plan from NNDP+EDO approach. However, the attacker success rate is 41.01% on the best defensive plan from RL+C-EDO, which is 1.76% less than the NNDP+EDO approach. Similarly, for $R2000$ AD graph, the attacker success rate is minimal under RL+C-EDO-based defensive plan. The results demonstrate that RL+C-EDO approach is able to generate better defensive plans and minimize the attacker's success rate.

Table 8. Comparison of Attacker's Chances of Success on Best Defense from Both the Proposed Attacker–Defender Policies (Smaller Number Represents Better Performance)

Graph	Policy	I	P	N	Average
R1000	Best defense from RL+C-EDO (Proposed)	**40.16%**	**41.36%**	**41.51%**	**41.01%**
	Best defense from NNDP+EDO (Proposed)	42.02%	44.76%	41.53%[a]	42.77%
R2000	Best defense from RL+C-EDO (Proposed)	**25.09%**	32.45%	**30.44%**	**29.32%**
	Best defense from NNDP+EDO (Proposed)	30.31%	**30.17%**[a]	30.85%	30.44%

Results show that the RL-based attacker policy approximates the attacker's problem more accurately than NNDP policy. AIndicates that with our general parameter settings, RL policy results were slightly bad than NNDP. Therefore, we train the RL policy for 300 epochs instead of 150 epochs. Given enough time, the RL policy outperforms the NNDP policy.

9.6 Analysis of Experimental Findings and Discussions

The results show that both the proposed defensive approaches are highly effective. For NNDP+EDO approach, we have exact optimal results for *R*500 attack graph, and on average, the proposed NNDP+EDO approach is less than 1% away from the optimal defense. In addition, the proposed attacker's policy NNDP approximates the success rate for the defense with high accuracy and incurs a small error of 2.53%. This shows that EDO trains NNDP very effectively, and trained NNDP acts as a very efficient fitness function for EDO. It is impossible for large R1000 and R2000 graphs to run the exact DP evaluation function; therefore, we simulate the defense using Monte Carlo simulation to get the attacker's success rate. The results show that the proposed approach NNDP+EDO is better than the baselines, corresponding to the NNDP attacker's policy. However, the proposed NNDP+EDO approach is not scalable to R4000 AD graphs; therefore, we proposed our second approach, i.e., RL+C-EDO.

The experimental results of RL+C-EDO approach proved that the proposed diversity-based C-EDO defensive policy is better than EC-based defense and greedy defense. The results also showed that EC-based defense performs better than greedy defense. Our second experimental setup proved that the proposed RL-based attacker policy approximates the attacker's problem more accurately than the proposed NNDP policy. For the same defensive plan, the RL-based attacker's policy achieves a higher success rate than the NNDP policy. Our third experimental setup proves that the proposed RL+C-EDO approach generates better defensive plans than NNDP+EDO, in turn minimizing the attacker's success rate.

Overall we propose two novel approaches aimed at strengthening cyber defense in AD graphs, utilizing EDO-based techniques. The first approach integrates NN with EDO to defend AD graphs. To determine the attacker's optimal success rate, we convert the attacker's problem into a MDP and then utilize DP to obtain attacker's exact success rates. However, DP applicability is limited to smaller AD graphs (up to R500 AD graph), prompting us to use NN to approximate attacker's DP for larger graphs (R1000 and R2000 AD graph). Our experimental results show that our NNDP+EDO approach works is highly effective on the R500 AD graph, with deviations less than 1% from the optimal solution. Moreover, our approach, NNDP+EDO, achieves good results for the R1000 and R2000 AD graphs. Nonetheless, scalability challenges emerge with the NNDP+EDO approach for R4000 AD graphs. To address this limitation, we propose our second approach named RL+C-EDO that utilizes RL to solve the attacker's problem. Our experiment results shows that the RL-based approach effectively approximates the attacker's policy for R4000 AD graphs. Our proposed RL+C-EDO approach achieves better results than NNDP+EDO approach as we have used RL to approximate the attacker's problem. NNDP+EDO attacker's policy trains the model against one defensive plan at a time, due to which it forgets the previous plan. This way, it keeps learning and forgetting the plans. However, in RL+C-EDO, we train our RL-based attacker's policy against multiple defensive plans at a time, due to which it learns shared experience and performs better. For RL agent, diverse environment configurations are only different in the "opening games," whereas the "end games" or "mid games" are likely to be similar across different environments. The similarity in later stages can be utilized in parallel training, where the agent is trained against multiple environments simultaneously and gains shared experience, leading to faster convergence and improved performance. Besides, the NNDP approach is a value iteration-based RL algorithm, whereas RL approach is a policy iteration-based RL algorithm. In general, the policy iteration-based algorithms converge faster than value iteration-based algorithms [Kaelbling et al., 1996], which is another reason for the superior performance of RL+C-EDO approach. This way, our proposed RL+C-EDO approach is able to achieve the attacker's and defender's goals better than the NNDP+EDO approach. Notably, our proposed RL+C-EDO approach is scalable to large-scale $R4000$ AD graphs. *In summary, RL+C-EDO approach is highly effective, more scalable, approximates the attacker's problem more accurately and generates better defensive plans.*

10 Discussion

Balancing Security and Usability in Edge Blocking. In this article, we used edge blocking as the primary defensive mechanism within AD graphs, aiming to enhance security by minimizing potential attack vectors. By strategically blocking edges, especially those with lower access privileges or peripherally connected to the DA, this approach effectively reduces the number of exploitable pathways. This selective interference is designed to limit the impact on essential operations, thus addressing usability concerns to a considerable extent. While our strategy prioritizes security enhancement, it also affects network usability by restricting access flexibility. However, it decreases the administrative burden needed to manage and audit these modifications, making it easier for administrators to block edges based on our recommendations rather than evaluating every edge individually. Our implementation focuses on minimizing disruptions by targeting less critical access points, thus preserving network functionality and operational efficiency.

Generalizability and Adaptation of Security Techniques. We designed our approach to seamlessly integrate with AD systems across a diverse range of industries. Through extensive testing on various scales of AD systems, we ensure that our models are both robust and effective in defending large organizational networks against sophisticated cyber threats. Our approach can be particularly effective in sectors like financial services and government operations, which are frequent targets of advanced attacks. Additionally, by adapting our models for critical infrastructure protection [Ahmad et al., 2023], we can enhance security measures crucial for safeguarding essential services. The incorporation of attacker's behavioral analytics further strengthens our models, enabling proactive identification of unusual network activities. This not only helps in preventing potential breaches before they escalate but also ensures the continuity and reliability of operations that are critical to organizational security and economic stability.

Scalability. Scalability is a critical factor for the practical implementation of defense mechanisms in large-scale networks, which are common in enterprise environments that utilize AD. Our proposed NN-based policy is able to manage large graphs with thousands of nodes and edges. To further address the scalability concerns, the article introduces RL-based policy that are designed to operate in parallel environments. This approach is effective for large-scale graph, as evidenced by the performance on the extensive R4000 graphs.

11 Conclusion and Future Work

This article has investigated a Stackelberg game model on an AD attack graph between an attacker and a defender. The defender aims to block a number of edges to minimize the attacker's probability of reaching DA; however, the attacker aims to maximize their chances of reaching DA. We first proved that both the attacker's and defender's problems are #P-hard. In order to solve the attacker–defender problem, we proposed two approaches. We first proposed an NNDP+EDO approach where we trained a NN to solve the attacker's problem and an EDO-based policy to solve the defender's problem. The trained NN acted as an efficient fitness function for the defender's policy. The experimental results showed that the proposed EDO-based defensive policy is highly effective. For the $R500$ AD attack graph, our proposed approach NNDP+EDO is less than 1% away from the optimal defense. In our second approach, i.e., RL+C-EDO, we proposed a RL-based policy to address the attacker problem and a C-EDO-based approach to address the defender problem. We trained the attacker policy against numerous defensive plans simultaneously and leveraged the trained RL critic network to evaluate the fitness of the defensive plans. Our experimental results showed that the proposed RL+C-EDO approach is highly effective and scalable to large-scale $R4000$ AD attack graphs.

In the future, we aim to explore dynamic AD environments that undergo continuous changes due to updates, modifications, and user activities. We plan to develop dynamic models capable of making decisions in real time in response to changes in network configurations and user behaviors. These models will be based on robust feedback mechanisms that facilitate continuous learning, and by analyzing the outcomes of previous defenses under similar conditions, the models will progressively refine their predictions and strategies, thereby enhancing both their accuracy and operational efficacy over time. Additionally, for a balanced approach that maximizes both security and usability, our future work will also focus on developing a more integrated model that quantifies and optimizes this tradeoff. By employing advanced analytics and RL algorithms, we aim to predict and evaluate the effects of specific defensive actions on network usability in real time. This will allow for adaptive strategies that dynamically adjust defensive measures based on ongoing assessments of security needs and usability impacts.

Declaration

The authors declare that they have no competing interests to disclose.

References

Majid Abdulsatar, Hussain Ahmad, Diksha Goel, and Faheem Ullah. 2024. Towards deep learning enabled cybersecurity risk assessment for microservice architectures. arXiv:2403.15169. Retrieved from https://doi.org/10.48550/arXiv.2403.15169

Hussain Ahmad, Isuru Dharmadasa, Faheem Ullah, and Muhammad Ali Babar. 2023. A review on c3i systems' security: Vulnerabilities, attacks, and countermeasures. *Computing Surveys* 55, 9 (2023), 1–38.

Neziha Akalin and Amy Loutfi. 2021. Reinforcement learning approaches in social robotics. *Sensors* 21, 4 (2021), 1292.

Hooman Alavizadeh, Hootan Alavizadeh, and Julian Jang-Jaccard. 2022. Deep Q-learning based reinforcement learning approach for network intrusion detection. *Computers* 11, 3 (2022), 41.

Giovanni Apruzzese, Mauro Andreolini, Mirco Marchetti, Andrea Venturi, and Michele Colajanni. 2020. Deep reinforcement adversarial learning against botnet evasion attacks. *IEEE Transactions on Network and Service Management* 17, 4 (2020), 1975–1987.

Mohamed A. A. Babiker, Mohamed A. O. Elawad, and Azza H. M. Ahmed. 2019. Convolutional neural network for a self-driving car in a virtual environment. In *Proceedings of the 2019 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE '19)*. IEEE, 1–6.

Richard Bellman. 1966. Dynamic programming. *Science* 153, 3731 (1966), 34–37.

Richard E. Bellman and Stuart E. Dreyfus. 2015. *Applied Dynamic Programming*, Vol. 2050. Princeton University Press.

Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. 2017. Neural combinatorial optimization with reinforcement learning. In *Proceedings of the International Conference on Learning Representations*, 1–5.

Jakob Bossek and Frank Neumann. 2021. Evolutionary diversity optimization and the minimum spanning tree problem. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 198–206.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. arXiv:1606.01540. Retrieved from https://doi.org/10.48550/arXiv.1606.01540

Lukas Brunke, Melissa Greeff, Adam W. Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P. Schoellig. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), 411–444.

Aneesh Sreevallabh Chivukula, Xinghao Yang, Wei Liu, Tianqing Zhu, and Wanlei Zhou. 2020. Game theoretical adversarial deep learning with variational adversaries. *IEEE Transactions on Knowledge and Data Engineering* 33, 11 (2020), 3568–3581.

Zhihua Cui, Zhixia Zhang, Zhaoming Hu, Shaojin Geng, and Jinjun Chen. 2021. A many-objective optimization based intelligent high performance data processing model for cyber-physical-social systems. *IEEE Transactions on Network Science and Engineering* 9, 6 (2021), 3825–3834.

Antoine Cully and Yiannis Demiris. 2017. Quality and diversity optimization: A unifying modular framework. *IEEE Transactions on Evolutionary Computation* 22, 2 (2017), 245–259.

John Dias. 2002. *A guide to microsoft active directory (ad) design*. Technical Report. Lawrence Livermore National Lab. (LLNL), Livermore, CA.

Zihan Ding, Yanhua Huang, Hang Yuan, and Hao Dong. 2020. *Introduction to Reinforcement Learning. Deep Reinforcement Learning: Fundamentals, Research and Applications*, 47–123. DOI: https://doi.org/10.1007/978-981-15-4095-0_2

Anh Do, Mingyu Guo, Aneta Neumann, and Frank Neumann. 2022. Analysis of evolutionary diversity optimization for permutation problems. *ACM Transactions on Evolutionary Learning* 2, 3 (2022), 1–27.

Anh Viet Do, Jakob Bossek, Aneta Neumann, and Frank Neumann. 2020. Evolving diverse sets of tours for the travelling salesperson problem. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, 681–689.

John Dunagan, Alice X. Zheng, and Daniel R. Simon. 2009. Heat-ray: combating identity snowball attacks using machine-learning, combinatorial optimization and attack graphs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, 305–320.

Ming Feng and Hao Xu. 2017. Deep reinforecement learning based optimal defense for cyber-physical system in presence of unknown cyber-attack. In *Proceedings of the 2017 IEEE Symposium Series on Computational Intelligence (SSCI '17).* IEEE, 1–8.

Jakob Gawlikowski, Cedrique Rovile Njieutcheu Tassi, Mohsin Ali, Jongseok Lee, Matthias Humt, Jianxiang Feng, Anna Kruspe, Rudolph Triebel, Peter Jung, Ribana Roscher, Muhammad Shahzad, Wen Yang, Richard Bamler, and Xiao Xiang Zhu. 2023. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review* 56, Suppl 1 (2023), 1513–1589.

Diksha Goel. 2023. Enhancing network resilience through machine learning-powered graph combinatorial optimization: Applications in cyber defense and information diffusion. arXiv:2310.10667. Retrieved from https://doi.org/10.48550/arXiv.2310.10667

Diksha Goel, Kristen Moore, Mingyu Guo, Derui Wang, Minjune Kim, and Seyit Camtepe. 2024. Optimizing cyber defense in dynamic active directories through reinforcement learning. In *Proceedings of the 29th European Symposium on Research in Computer Security (ESORICS)*. Springer Nature Switzerland, 332–352.

Diksha Goel, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. 2023. Evolving reinforcement learning environment to minimize learner's achievable reward: An application on hardening active directory systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1348–1356.

Diksha Goel, Max Hector Ward-Graham, Aneta Neumann, Frank Neumann, Hung Nguyen, and Mingyu Guo. 2022. Defending active directory by combining neural network based dynamic program and evolutionary diversity optimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22),* 1191–1199.

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

Mingyu Guo, Max Ward, Aneta Neumann, Frank Neumann, and Hung Nguyen. 2023. Scalable edge blocking algorithms for defending active directory style attack graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Kim Hammar and Rolf Stadler. 2020. Finding effective security strategies through reinforcement learning and self-play. In *Proceedings of the 2020 16th International Conference on Network and Service Management (CNSM '20).* IEEE, 1–9.

Long He, Geng Sun, Dusit Niyato, Hongyang Du, Fang Mei, Jiawen Kang, Mérouane Debbah, and and Zhu Han. 2024. Generative AI for game theory-based mobile networking. arXiv:2404.09699. Retrieved from https://doi.org/10.48550/arXiv.2404.09699

Emmanuel Hebrard, Brahim Hnich, Barry O'Sullivan, and Toby Walsh. 2005. Finding diverse and similar solutions in constraint programming. In *Proceedings of the AAAI*, Vol. 5, 372–377.

Erik Hemberg, Joseph R. Zipkin, Richard W. Skowyra, Neal Wagner, and Una-May O'Reilly. 2018. Adversarial co-evolution of attack and defense in a segmented computer network environment. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1648–1655.

Jeffrey K. Hollingsworth, Barton Paul Miller, and Jon Cargille. 1994. Dynamic program instrumentation for scalable performance tools. In *Proceedings of the IEEE Scalable High Performance Computing Conference*. IEEE, 841–850.

Hao Hu, Yuling Liu, Chen Chen, Hongqi Zhang, and Yi Liu. 2020. Optimal decision making approach for cyber security defense using evolutionary game. *IEEE Transactions on Network and Service Management* 17, 3 (2020), 1683–1700.

Chen Huang, Xiangbing Zhou, Xiaojuan Ran, Yi Liu, Wuquan Deng, and Wu Deng. 2023. Co-evolutionary competitive swarm optimizer with three-phase for large-scale complex optimization problem. *Information Sciences* 619 (2023), 2–18.

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.

Hany Kasban, M. A. M. El-Bendary, and D. H. Salama. 2015. A comparative study of medical imaging techniques. *International Journal of Information Science and Intelligent System* 4, 2 (2015), 37–58.

Bogdan Korel and Janusz Laski. 1988. Dynamic program slicing. *Information Processing Letters* 29, 3 (1988), 155–163.

Pawel Ladosz, Lilian Weng, Minwoo Kim, and Hyondong Oh. 2022. Exploration in deep reinforcement learning: A survey. *Information Fusion* 85 (2022), 1–22.

Harjinder Singh Lallie, Kurt Debattista, and Jay Bal. 2020. A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review* 35 (2020), 100219.

Joel Lehman and Kenneth O. Stanley. 2013. Evolvability is inevitable: Increasing evolvability without the pressure to adapt. *PloS One* 8, 4 (2013), e62186.

Yapeng Li, Ye Deng, Yu Xiao, and Jun Wu. 2019. Attack and defense strategies in complex networks based on game theory. *Journal of Systems Science and Complexity* 32, 6 (2019), 1630–1640.

Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. 2021. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems* 33, 12 (2021), 6999–7019.

Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. 2017. Tactics of adversarial attack on deep reinforcement learning agents. arXiv:1703.06748. Retrieved from https://doi.org/10.48550/arXiv.1703.06748

Genggeng Liu, Zhenyu Pei, Nengxian Liu, and Ye Tian. 2023. Subspace segmentation based co-evolutionary algorithm for balancing convergence and diversity in many-objective optimization. *Swarm and Evolutionary Computation* 83 (2023), 101410.

Yutaka Matsuo, Yann LeCun, Maneesh Sahani, Doina Precup, David Silver, Masashi Sugiyama, Eiji Uchibe, and Jun Morimoto. 2022. Deep learning, reinforcement learning, and world models. *Neural Networks* 152 (2022), 267–275.

Fei Ming, Wenyin Gong, and Ling Wang. 2022. A two-stage evolutionary algorithm with balanced convergence and diversity for many-objective optimization. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52, 10 (2022), 6222–6234.

Seyedali Mirjalili. 2019. Evolutionary algorithms and neural networks. *Studies in Computational Intelligence* 780 (2019), 43–53.

Aneta Neumann, Denis Antipov, and Frank Neumann. 2022. Coevolutionary Pareto diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 832–839.

Aneta Neumann, Wanru Gao, Carola Doerr, Frank Neumann, and Markus Wagner. 2018. Discrepancy-based evolutionary diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 991–998.

Aneta Neumann, Wanru Gao, Markus Wagner, and Frank Neumann. 2019. Evolutionary diversity optimization using multi-objective indicators. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 837–845.

Aneta Neumann, Sharlotte Gounder, Xiankun Yan, Gregory Sherman, Benjamin Campbell, Mingyu Guo, and Frank Neumann. 2023. Diversity optimization for the detection and concealment of spatially defined communication networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1436–1444.

Quang Huy Ngo, Mingyu Guo, and Hung Nguyen. 2023. Near optimal strategies for honeypots placement in dynamic and large active directory networks. In *Proceedings of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS '23).* Extended Abstract, 2517–2519.

Adel Nikfarjam, Ralf Rothenberger, Frank Neumann, and Tobias Friedrich. 2023. Evolutionary diversity optimisation in constructing satisfying assignments. In *Proceedings of the Genetic and Evolutionary Computation Conference,* 938–945.

Aritran Piplai, Mike Anoruo, Kayode Fasaye, Anupam Joshi, Tim Finin, and Ahmad Ridley. 2022. Knowledge guided two-player reinforcement learning for cyber attacks and defenses. In *Proceedings of the 2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA '22).* IEEE, 1342–1349.

Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J. Anders, and Klaus-Robert Müller. 2021. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings of the IEEE* 109, 3 (2021), 247–278.

Jeffrey R. Sampson. 1976. Adaptation in natural and artificial systems (John H. Holland).

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. arXiv:1707.06347. Retrieved from https://doi.org/10.48550/arXiv.1707.06347

Shorya Sharma. 2021. Game theory for adversarial attacks and defenses. arXiv:2110.06166. Retrieved from https://doi.org/10.48550/arXiv.2110.06166

David Simoes, Simao Reis, Nuno Lau, and Luis Paulo Reis. 2020. Competitive deep reinforcement learning over a pokémon battling simulator. In *Proceedings of the 2020 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC '20).* IEEE, 40–45.

Shengpu Tang and Jenna Wiens. 2021. Model selection for offline reinforcement learning: Practical considerations for healthcare settings. In *Proceedings of the Machine Learning for Healthcare Conference*. PMLR, 2–35.

Tamara Ulrich, Johannes Bader, and Eckart Zitzler. 2010. Integrating decision space diversity into hypervolume-based multiobjective search. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 455–462.

Tamara Ulrich and Lothar Thiele. 2011. Maximizing population diversity in single-objective optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, 641–648.

Leslie G. Valiant. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8, 3 (1979), 410–421.

Pradnya A. Vikhar. 2016. Evolutionary algorithms: A critical review and its future prospects. In *Proceedings of the 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC '16)*. IEEE, 261–265.

Yevgeniy Vorobeychik, Bo An, and Milind Tambe. 2012. Adversarial patrolling games. In *Proceedings of the 2012 AAAI Spring Symposium Series*.

Hao-nan Wang, Ning Liu, Yi-yun Zhang, Da-wei Feng, Feng Huang, Dong-sheng Li, and Yi-ming Zhang. 2020. Deep reinforcement learning: A survey. *Frontiers of Information Technology & Electronic Engineering* 21, 12 (2020), 1726–1744.

Jingkang Wang, Tianyun Zhang, Sijia Liu, Pin-Yu Chen, Jiacen Xu, Makan Fardad, and Bo Li. 2021. Adversarial attack generation empowered by min-max optimization. *Advances in Neural Information Processing Systems* 34 (2021), 16020–16033.

Yutong Wang, Ke Xue, and Chao Qian. 2021. Evolutionary diversity optimization with clustering-based selection for reinforcement learning. In *Proceedings of the International Conference on Learning Representations*.

Jiayi Weng, Huayu Chen, Dong Yan, Kaichao You, Alexis Duburcq, Minghao Zhang, Yi Su, Hang Su, and Jun Zhu. 2022. Tianshou: A highly modularized deep reinforcement learning library. *The Journal of Machine Learning Research* 23, 1 (2022), 12275–12280.

Michael L. Winterrose, Kevin M. Carter, Neal Wagner, and William W. Streilein. 2020. Adaptive attacker strategy development against moving target cyber defenses. *Advances in Cyber Security Analytics and Decision Systems* (2020), 1–14.

Junlin Wu, Charles Kamhoua, Murat Kantarcioglu, and Yevgeniy Vorobeychik. 2021. Learning generative deception strategies in combinatorial masking games. In *Proceedings of the Decision and Game Theory for Security: 12th International Conference (GameSec '21)*. Springer, 98–117.

Shenghe Xu, Shivendra S. Panwar, Murali Kodialam, and T. V. Lakshman. 2020. Deep neural network approximated dynamic programming for combinatorial optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34, 1684–1691.

Bing Xue, Mengjie Zhang, Will N. Browne, and Xin Yao. 2015. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation* 20, 4 (2015), 606–626.

Feidiao Yang, Tiancheng Jin, Tie-Yan Liu, Xiaoming Sun, and Jialin Zhang. 2018. Boosting dynamic programming with neural networks for solving np-hard problems. In *Proceedings of the Asian Conference on Machine Learning*. PMLR, 726–739.

Yaodong Yang and Jun Wang. 2020. An overview of multi-agent reinforcement learning from game theoretical perspective. arXiv:2011.00583. Retrieved from https://doi.org/10.48550/arXiv.2011.00583

Zhengyu Yin, Dmytro Korzhyk, Christopher Kiekintveld, Vincent Conitzer, and Milind Tambe. 2010. Stackelberg vs. Nash in security games: interchangeability, equivalence, and uniqueness. In *Proceedings of the AAMAS*, Vol. 10, 6.

Chao Yu, Jiming Liu, Shamim Nemati, and Guosheng Yin. 2021. Reinforcement learning in healthcare: A survey. *ACM Computing Surveys (CSUR)* 55, 1 (2021), 1–36.

G. Zames. 1981. Genetic algorithms in search, optimization and machine learning. *Information Technology Journal* 3, 1 (1981), 301.

Linda Zhang and Erik Hemberg. 2019. Investigating algorithms for finding Nash equilibria in cyber security problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 1659–1667.

Yumeng Zhang, Max Ward, Mingyu Guo, and Hung Nguyen. 2023. A scalable double oracle algorithm for hardening large active directory systems. In *Proceedings of the 18th ACM ASIA Conference on Computer and Communications Security (ACM ASIACCS '23)*, 993–1003.

Received 6 January 2024; revised 9 July 2024; accepted 14 July 2024