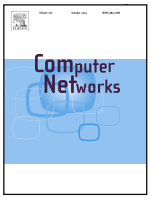
Computer Networks 281 (2026) 112183

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet



Multi-Objective and deep Q-Learning for countermeasure selection in 5G intrusion response systems

Arash Bozorgchenani [a,*], Dimitris Manolakis [b], Antonios Lalas [b]

[a] School of Computer Science, University of Leeds, Leeds, UK
[b] Centre for Research and Technology Hellas, Greece

ARTICLE INFO

Keywords:
Countermeasure selection
quality of service
optimization
reinforcement learning
intrusion response systems and 5G

ABSTRACT

Network connectivity exposes network infrastructure and assets to vulnerabilities exploitable by attackers. Safeguarding these assets necessitates implementing security countermeasures. However, deploying countermeasures incurs various costs, including preparation and deployment time. Therefore, an Intrusion Response System (IRS) must consider both security and Quality of Service (QoS) costs when dynamically selecting countermeasures to address detected attacks. To address this challenge, we introduce a joint Security-vs-QoS optimization problem akin to the Weighted Set Cover Problem (WSCP), which is NP-complete. We propose two learning-based solutions leveraging Multi-Objective Reinforcement Learning and Deep Q-learning to navigate the security and QoS cost trade-off. Through extensive simulations under diverse settings, we validate the performance of our proposed solution, compare it with benchmark methods, and evaluate it using a project-derived 5G cybersecurity dataset.

1. Introduction

The 5G environment facilitates the integration of various technologies and services, including vehicular communication, unmanned aerial vehicles, smart homes, fog/edge computing, smart grid, smart parking, blockchain services, and Industry 4.0, among others. Network operators empower mobile devices to seamlessly transition between networks and services based on their service requests, ensuring a consistently high level of Quality of Service (QoS) [1,2]. However, during rapid handovers between networks and while communicating with other nodes, these devices are exposed to various vulnerabilities, including access control issues, communication security risks, threats to data confidentiality, potential availability concerns, and privacy breaches [3]. Therefore, meeting the high-demand requirements of 5G services for all users while upholding stringent security standards poses a significant challenge.

Network security management can be broken down into three main phases: asset-based threat profiling, network attack identification and attack mitigation. In the first phase, valuable assets within the network are identified and ranked based on their value according to perceived necessity and importance to the network's operations. In the second phase, the network attacks targeting the assets are detected by the design of Intrusion Detection Systems (IDS) which can involve techniques such as continuous monitoring of network traffic, and anomaly detection. Finally, in the third phase, the security advisory system comes into play, providing actionable recommendations for mitigation. These recommendations may include implementing security patches, adjusting configurations, isolating compromised systems, or deploying additional security controls. The goal is to swiftly respond to detected threats and minimize the impact on network integrity and operations. This is also in line with three phases of OCTAVE-S risk-based strategic assessment and planning technique for security deployed in some organisations [4].

IDS techniques are executed on Security Agents (SAs) or stubs, strategically deployed to sniff encrypted traffic and extract features for attack detection. When suspicious activities are detected, agents notify the orchestrator, which may take necessary mitigation actions [5]. Intrusion Response Systems (IRSs) are engineered to react against these suspicious activities in real or near real-time by continuously monitoring the IDS alerts [6]. In a 5G network, responses to security incidents can involve various actions, such as notifying the network operator or vendor, filtering traffic, relaunching a node, reconfiguring virtual network functions, replacing one node with another, providing patches to prevent or remedy identified attacks, and more [7].

The effectiveness of different countermeasures can be assessed based on their ability to mitigate the risks faced by network assets. One approach involves employing a combination of different countermeasures to address affected assets comprehensively. While it is crucial from a security standpoint to mitigate as many detected attacks as possible to minimize threats to the network, implementing countermeasures can impact the system's QoS requirements, including time costs associated with preparing and deploying countermeasures. This paper aims to

* Corresponding author.
E-mail address: a.bozorgchenani@leeds.ac.uk (A. Bozorgchenani).

https://doi.org/10.1016/j.comnet.2026.112183
Received 21 May 2025; Received in revised form 6 November 2025; Accepted 3 March 2026
Available online 13 March 2026
1389-1286/© 2026 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

reconcile this trade-off within 5G systems. The proposed framework can be integrated as a module in network architecture enabling automated decision-making based on the network status.

Hence, we formulate a joint Security-vs-Cost countermeasure selection to optimize such decision-making in the IRSs. To tackle this dual challenge, we propose a Multi-Objective Q-Learning (MOQL) solution. It accounts for both security and costs as primary objectives, training the system to make optimal decisions regarding countermeasure selection. This decision-making process must strike a balance between the efficacy of risk mitigation measures and their associated costs. This delicate balance is overseen by the security administrator, tasked with maintaining sufficient protection while ensuring timely implementation of countermeasures within defined constraints. Our contributions can be summarized as follows:

1. Providing a security and time model for the countermeasure selection problem by modelling both sequential and parallel deployment of countermeasures, and considering both atomic and non-atomic countermeasure execution,
2. Formulating a joint security-vs-cost problem for the optimal selection of countermeasures and proving its resemblance to the Weighted Set Cover Problem (WSCP),
3. Presenting, for the first time to the best of our knowledge, a MOQL solution using a linear scalarized ϵ-greedy method to solve the formulated problem,
4. Proposing a Deep Q-Learning (DQL)-based solution to solve the problem,
5. Conducting extensive simulations using project-driven experimental data to evaluate the impact of key parameters and to compare the performance of the two proposed learning-based solutions,

Our main contribution lies in modelling the countermeasure selection problem in IRSs with both sequential and parallel execution of atomic and non-atomic actions, introducing a novel MOQL-based solution (to the best of our knowledge, the first of its kind), and benchmarking it against a DQL approach to evaluate their performance and trade-offs.

The rest of the paper is organized as follows. In Section 2, we review the state of the art. In Section 3 the system model is described. Section 4 formulates the problem. In Section 5, we present both the MOQL and Deep Q-Learning-based solutions. In Section 6 we present the simulation results. Section 7 concludes the paper.

2. Related works

2.1. Intrusion response systems

Countermeasure selection strategies have been widely explored across several domains, including cyber-physical systems risk analysis, cybersecurity investment planning, and infrastructure protection. These diverse application areas emphasise the importance of efficient and cost-effective intrusion response mechanisms.

A considerable number of studies have investigated the use of optimisation algorithms for countermeasure selection. Bio-inspired techniques, such as Genetic Algorithms (GAs) and Artificial Immune Systems (AISs), have been among the most frequently utilised methods. In [8], an AIS-based approach was introduced to select appropriate countermeasures against cyber threats by employing cloning and mutation operations. A context-driven termination condition was proposed to control the evolutionary process based on experimental observations and subjective thresholds. Building upon this work, the authors in [9] developed AISGA, a hybrid system that integrates a Genetic Algorithm to optimise the AIS parameters, effectively aiming to minimise overall risk and reduce computational overhead.

Expanding the bio-inspired approaches, [10] proposed a framework to generate comprehensive response strategies that simultaneously address countermeasure selection, deployment order, and duration optimisation. The solution employed a three-dimensional encoding within a GA to capture the multifaceted nature of the response problem, considering criteria such as attack impact, deployment overhead, negative side effects, and security gains. However, the convergence speed of GA-based methods is often limited when dealing with complex decision spaces. Similarly, other evolutionary techniques for IRSs, such as those explored in [11], face challenges related to slow convergence and sub-optimality.

Graph-based models have also been leveraged extensively. In [12], countermeasure selection was formulated over Probabilistic Attack-Response Trees to handle multi-path attack scenarios. A greedy algorithm was proposed to select countermeasures by evaluating the expected security gains, cost, and potential negative impacts. Meanwhile, the work in [13] proposed a modelling approach based on Directed Acyclic Graphs to formalise attack-defense interactions. Defence strategies were then derived and optimised using Integer Linear Programming (ILP), with a supporting software tool developed for practical deployment. Attack Graph-based methodologies were also enhanced in [14], where a refined risk assessment model was proposed to avoid frequent re-generation of attack paths. A heuristic method was subsequently used to determine optimal countermeasure deployment under budget constraints, demonstrating efficacy within organisational IT environments.

In our recent work [7], we proposed a many-to-one stable matching game to assign attacks to countermeasures. However, in that study, all countermeasures were assumed to be applied sequentially. In contrast, the current work accounts for the more realistic scenario where some countermeasures can be executed sequentially while others can be applied in parallel.

Machine learning-based approaches have also emerged to tackle the countermeasure selection problem. In [15], Deep Reinforcement Learning (DRL) was applied to automate intrusion response actions in systems with stationary conditions. This was later extended in [16] to non-stationary environments, where system components are dynamic and possess multiple states (e.g., active, updated, corrupted, vulnerable). These approaches trained agents in simulated environments to minimise response time and costs. Comparative evaluations demonstrated the potential advantages of DRL over traditional Q-learning, although stability and generalisation remain open challenges.

There have recently been some new papers on IRSs that explore security-performance trade-offs in modern networked systems. Li et al. [12] formulated the response to multi-path attacks as an optimization problem but relied on a greedy heuristic, limiting scalability and adaptability. Luo [17] proposed a fog-based intrusion detection and IRS model using support vector machine and an improved NSGA-II, though the approach remains static and computationally heavy for real-time use. Hamad et al. [18] developed REACT, a vehicle-level IRS enabling fast local reactions, yet its scope is restricted to automotive contexts and lacks broader network coordination. A comprehensive survey of IRS for cyber-physical systems including taxonomy, countermeasures, general architecture, and decision-making models is given in [19].

Finally, broader studies have investigated the economics of cybersecurity defences, including the evaluation of private versus societal costs [20], uncertainty quantification in risk models [21], investment optimisation in supply chains [22], and balancing different security safeguards [23]. Governmental and enterprise decision-making on cybersecurity planning has also been explored, such as in [24] and [25], using stochastic programming and dynamic optimisation models. For a detailed review of the progression in IRS designs from 2012 to 2017, the survey in [26] offers a broad synthesis of key proposals, highlighting their strengths and limitations.

2.2. Intrusion detection and response systems

The proposed optimisation model in this work focuses on countermeasure selection following attack detection. While the model itself does not perform intrusion detection, it can be integrated into broader network defence frameworks that couple detection with mitigation. For

Table 1
Comparison of the most related works on countermeasure selection.

Ref.	Objectives		Execution mode		Countermeasure granularity		Solution
	Time	Security	Sequential	Parallel	Atomic	Non-atomic	
[7]	✓	✓	✓	✗	✓	✓	Game theory
[9]	✓	✓	✓	✗	✓	✗	GA and AIS
[10]	✓	✓	✓	✓	✓	✗	GA
[12]	✓	✓	✓	✗	✓	✗	Heuristic
[13]	✗	✓	-	-	✓	✓	ILP
[14]	✗	✓	✗	✓	✓	✓	Heuristic
[15]	✓	✓	✓	✗	✓	✗	DRL
[16]	✓	✓	✓	✗	✓	✗	DRL
Our work	✓	✓	✓	✓	✓	✓	MORL and DQL

instance, systems such as PEDDA [27] employ progressive multi-stage detection and orchestration mechanisms that could, in principle, benefit from an intelligent countermeasure selection module. Building on this integration perspective, several adaptive Detection-and-Response (IDRS) frameworks [28–34], provide examples where countermeasure optimisation could be practically embedded.

Rose et al. [28] introduced an integrated IDRS for IoT networks that combines profiling, machine learning, and attack-graph reasoning to detect and mitigate threats. Although their game-theoretic IRS dynamically selects mitigation actions, it does not account for temporal dependencies, sequential or parallel countermeasures, nor the trade-off between mitigation time and achieved security. In a similar direction, Hussain et al. [29] proposed a Calibrated Random Forest-based IDRS that quantifies uncertainty in attack predictions and uses probabilistic confidence scores to guide cost-sensitive mitigation. While this approach strengthens decision rationality by linking response actions to confidence levels, it omits modelling of temporal dynamics, countermeasure interdependencies, and joint optimisation of response time, security impact, and action concurrency.

Further, Chen et al. [30] developed a hierarchical UAV IDRS using artificial immune principles, achieving high detection accuracy through air-ground collaboration. However, their model focuses primarily on detection and does not optimise countermeasure selection or balance time-security trade-offs, as addressed in this work. Similarly, Lee et al. [31] proposed a real-time intrusion detection and prevention system for in-vehicle networks using a lightweight Random Forest model implemented on an ASIC chip. Although their design achieved microsecond-level latency for on-chip attack blocking, it remains restricted to binary response decisions and lacks multi-level countermeasure selection, and QoS-security trade-off modelling–key aspects explicitly optimised in our framework.

Vieira et al. [32] advanced the field by introducing an autonomic IRS that integrates autonomic computing and big-data analytics for rapid detection and self-healing in distributed environments. Despite its probabilistic decision-making using the expected utility principle, it does not jointly optimise security, response time, and QoS impacts, as our model does. Likewise, Huang and Su [33] designed a deep-learning-based IDRS for power grid networks to capture temporal patterns and improve detection accuracy. While effective in detection, its response is limited to alerting functions, without modelling the selection or coordination of defensive actions based on cost or timing considerations. Finally, although El-Hajj [34] presents a hybrid IDRS combining Random Forest for static analysis and LSTM for temporal learning, the "response" component is limited to alert generation rather than selecting or optimising defensive countermeasures. Hence, the work remains primarily detection-focused.

While previous research has made significant strides in the field of countermeasure selection, several limitations persist. Many evolutionary methods offer only approximate solutions, often requiring extensive iterations without guarantees of convergence to optimal responses. Additionally, some prior works focus on optimising security metrics without fully incorporating dynamic QoS factors, limiting adaptability in real-world, variable network conditions. Furthermore, in many existing studies, the datasets used for modeling attacks, countermeasures, and system behavior are often based on synthetic or simulated scenarios rather than real-world environments, which can limit the practical applicability and robustness of the proposed solutions.

In contrast, our work addresses these gaps by formulating the countermeasure selection problem and proving its resemblance to the NP-hard WSCP. We consider both sequential and parallel execution of countermeasures, as well as atomic and non-atomic actions, to capture realistic system dynamics. To solve this problem, we propose two learning-based approaches, MOQL and DQN, that explicitly model the Security-QoS trade-off for dynamic, context-aware decision-making. The models are trained and validated using datasets derived from attack and countermeasure scenarios defined in an EU 5G security project [35]. Hence, our novelty lies in the problem formulation, the proposed modeling approach, and the learning-based solutions validated on project-driven experimental data. Table 1 summarises the key differences and research gaps between this work and the most closely related studies.

3. System model

A wireless network is typically composed of various devices such as Internet of Things (IoT) units, user equipment, Base Stations (BSs), servers, network functions and cloud facilities, among others, which are the network assets. Let us show the set of all of these nodes as $\mathcal{U} = \{u_1, \dots, u_n, \dots u_N\}$. The inherently untrusted nature of the internet, coupled with its insecure connections, renders all nodes susceptible to attacks.

In order to protect a network against cyberattacks, it is essential to design IRSs. These systems must respond effectively to eliminate potential consequences and mitigate security risks while also considering their impact on QoS costs [36]. A critical aspect of applying countermeasures lies in understanding the three-time components involved: decision-making time, preparation time, and deployment time. While certain countermeasures may offer a complete resolution of the problem, they could lead to extended overall latency, thereby risking attack leakage and system-wide propagation. Thus, a delicate trade-off arises between the level of system security and the QoS cost (or latency in this context). Finding the right balance between these two aspects becomes essential for optimizing IRSs.

3.1. Security model

A key facet of countermeasure selection is comparing the level of risk measured in the network against the potential security gain of a particular countermeasure.

Let us assume there exist A types of attacks in the system, where a shows a generic attack type. Let us define $\theta_c^{n,a}$ as the cth countermeasure applied on the ath attack of the nth node. For the sake of simplicity in the notation, $\theta_c^{n,a}$ might be written as θ_c throughout the article. For each attack type, we consider a mitigation action list that shows the possible countermeasures that can be taken. Let us show $C(a)$ as the list

of countermeasures that can be taken for attack type a and define it as

$$C(a) = \{\theta_c | \mathbb{U}_c^{n,a} = 1, \forall u_n \in \mathcal{U}\} \tag{1}$$

where $\mathbb{U}_c^{n,a}$ is an indicator function which is 1 if countermeasure c can address the ath attack of node n. Let us show C as the total number of system countermeasures for all attack types, i.e., $|\bigcup_{a=1}^{A} C(a)| = C$.

On the other hand, each countermeasure might address several attack types. Let us show the attack types the cth countermeasure can address as

$$\mathcal{W}(\theta_c) = \{\omega_a | \mathbb{U}_c^{n,a} = 1, \forall u_n \in \mathcal{U}\} \tag{2}$$

where ω_a shows the ath attack type (e.g. DoS or eavesdropping). It should be noted that one attack type might be found in different nodes across the network. That is, different nodes may be affected by a particular attack type.

Let $\mathcal{V} = \{v_n^a\}_{\substack{a=1,\cdots A \\ n=1,\cdots,N_a}}$ represent the set of all detected attacks across the nodes. Each element v_n^a correspondsds to attack type a detected on node n and N_a is the number of nodes affected by that attack. Thus, the total number of detected attacks across all nodes is given by $|\mathcal{V}|$. We also define $\mathcal{S}_{\text{att}}$ as a collection set of θ_C subsets covering all of the elements (attacks) in $\mathcal{V}$ as below:

$$\begin{aligned}\mathcal{S}_{\text{att}} &= \bigcup_{c=1,\ldots,C}\left(\bigcup_{\omega_a \in \mathcal{W}(\theta_c)} \{v_n^a\}_{n=1,\ldots,N_a}\right) \\ &= \Bigg\{ \underbrace{\Big\{ \ldots, \overbrace{\{v_1^a, \ldots v_{N_a}^a\}}^{\omega_a \in \mathcal{W}(\theta_1)}, \ldots \Big\}}_{\theta_1}, \ldots \\ &\quad \underbrace{\Big\{ \ldots, \overbrace{\{v_1^a, \ldots v_{N_a}^a\}}^{\omega_a \in \mathcal{W}(\theta_C)}, \ldots \Big\}}_{\theta_C} \Bigg\}\end{aligned} \tag{3}$$

The set $\mathcal{S}_{\text{att}}$ is the complete list of all attack types, where each type includes all the nodes affected by that attack and is categorized under the countermeasure capable of addressing it. In order to cover/address all of the attacks in the network one solution is to take all of the countermeasures. However, this imposes a high cost on the system. Let us show the matrix Θ as an $[N \times C \times A]$ binary allocation matrix, where $A = 3$ (for the sake of simplicity of presentation) as below:

$$\Theta = \begin{array}{c} \\ \text{Nodes} \\ \\ \\ \\ \\ \end{array} \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array} \begin{array}{|c|c|c|c|c|} \hline \theta_1^{1,3} & \theta_2^{1,3} & \theta_3^{1,3} & \cdots & \theta_C^{1,3} \\ \hline \theta_1^{1,2} & \theta_2^{1,2} & \theta_3^{1,2} & \cdots & \theta_C^{1,2} \\ \hline \theta_1^{1,1} & \theta_2^{1,1} & \theta_3^{1,1} & \cdots & \theta_C^{1,1} \\ \hline \theta_1^{2,1} & \theta_2^{2,1} & \theta_3^{2,1} & \cdots & \theta_C^{2,1} \\ \hline \theta_1^{3,1} & \theta_2^{3,1} & \theta_3^{3,1} & \cdots & \theta_C^{3,1} \\ \hline \vdots & \vdots & \vdots & \cdots & \vdots \\ \hline \theta_1^{N,1} & \theta_2^{N,1} & \theta_3^{N,1} & \cdots & \theta_C^{N,1} \\ \hline \end{array} \begin{array}{c} \\ \\ \theta^{2,3}_C \\ \theta^{3,2}_C \\ \theta^{N,3}_C \\ \\ \end{array} \begin{array}{l} \\ \\ \\ \\ \\ \\ \text{Attack types} \end{array} \\ \text{Countermeasures} \tag{4}$$

Eq. (4) shows the countermeasures that can be taken for each attack of all the nodes. $\theta_c^{n,a}$ shows the cth countermeasure that can be taken for addressing attack a of node n, and the value is binary (0 if the node does not have the ath attack, the cth countermeasure does not address that attack or the cth countermeasure is not selected to address it, and 1 if the cth countermeasure is used to address it). The decision is to assign some of the elements (i.e., countermeasures) in the above matrix to take the value of 1 in order to address all the attacks across the network assuming no attack is addressed by more than one countermeasure. Please note that our model accounts for scenarios where multiple countermeasures are combined to address a single attack. In this context, we treat these combinations as separate countermeasures. Therefore, our model encompasses both atomic and non-atomic countermeasures. In essence, some of the C countermeasures in our model are atomic, while others are non-atomic.

Let us define $\boldsymbol{\Theta}$ as the vector of the selected countermeasures to address the detected attacks in the network (the ones in (4) with the value of 1 are selected). As we mentioned earlier one attack cannot be addressed with two different countermeasures (unless the selected countermeasure is non-atomic), that is:

$\omega_a \in \bar{\mathcal{W}}(\theta_c) \Rightarrow \omega_a \notin \bar{\mathcal{W}}(\theta_j) : \forall \omega_a, \forall \theta_c, \theta_j \in \boldsymbol{\Theta}, \theta_j \neq \theta_c$ where $\bar{\mathcal{W}}(\theta_c)$ is the set of attacks that will be addressed with the countermeasure θ_c. This ensures that, in the final solution, each attack is addressed by only one countermeasure, which may be either atomic or non-atomic. Now the set of attacks covered with a solution vector $\boldsymbol{\Theta}$ (selected countermeasures) can be rewritten from eq.(3) as:

$$\bar{\mathcal{S}}_{\text{att}}(\boldsymbol{\Theta}) = \bigcup_{\theta_c \in \boldsymbol{\Theta}} \left(\bigcup_{\omega_a \in \bar{\mathcal{W}}(\theta_c)} \{v_n^a\}_{n=1,\ldots,N_a} \right) \tag{5}$$

This provides the list of attacks that are covered by one of the selected countermeasures in $\boldsymbol{\Theta}$. In order to address all the attacks with the selected countermeasures in $\boldsymbol{\Theta}$, we shall yield $|\bar{\mathcal{S}}_{\text{att}}(\boldsymbol{\Theta})| = |\mathcal{V}|$, i.e., the number of elements (covered attacks) in $\bar{\mathcal{S}}_{\text{att}}(\boldsymbol{\Theta})$ equals the number of identified attacks.

IDSs offer risk assessment metrics, including the severity and probability of attacks. Leveraging this information, the Risk Factor (RF) for the a-th attack can be determined as $R_a = S(a) \cdot P(a)$, where $0 \leq P(a) \leq 1$ is the probability/likelihood of occurrence of an attack and $S(a) \in [0 \quad 10]$ is its severity. IDSs can also evaluate the degree to which security is enhanced by applying specific security countermeasures. This capability aids IRSs in comparatively quantifying the effectiveness of various countermeasures [37]. After implementing a countermeasure, the RF will be updated to assess the effectiveness of the selected countermeasure. Throughout the remainder of the paper, our focus will be solely on the RF, as it indicates the severity and likelihood of an attack. Let $\bar{R}_a(\theta_c)$ denote the updated RF of the a-th attack after applying the countermeasure θ_c, and let R_a represent its RF before the countermeasure is applied. As part of the threat mitigation process, we would like to reduce the updated RF as much as possible by implementing the most suitable countermeasure, thus, similar to our previous work in [7] we define $\Delta R_a(\theta_c) = R_a - \bar{R}_a(\theta_c)$ as the discrepancy between the initial value of RF and the updated RF that should be maximized. Please note that $\Delta R_a(\theta_c) > 0$, i.e., the updated RF for those addressed attacks after taking a countermeasure is always reduced. We define the security utility function as

$$\frac{\sum_{\theta_c \in \boldsymbol{\Theta}} \sum_{v_n^a \in \mathcal{V}} \Delta R_a(\theta_c)}{\sum_{v_n^a \in \mathcal{V}} R_a} \tag{6}$$

Eq. (6) calculates the reduced RF of those addressed attacks across the nodes over the weighted initial RF values.

3.2. Time model

Applying a countermeasure requires some preparation and deployment time, which varies across different countermeasure types. The overall spent Time for the cth countermeasure can be written as

$$T(\theta_c) = T^{\text{dep}}(\theta_c) + T^{\text{pre}}(\theta_c) \tag{7}$$

where $T^{\text{pre}}(\theta_c)$ is the time spent for the preparation of the countermeasure (also known as service preparation time), and $T^{\text{dep}}(\theta_c)$ is the time spent for the deployment of the countermeasure (also known as service deployment time).

In an IRS, the selection and deployment of countermeasures are influenced by various factors, including the nature of the attack, available

resources, and the desired outcome of the response. This often leads to a combination of sequential and parallel execution of countermeasures. Sequential execution allows the IRS to address the attack in a structured manner, prioritizing critical tasks and ensuring that resources are allocated effectively. However, in certain situations where time is of the essence or when multiple countermeasures can be deployed concurrently without interfering with each other, parallel execution offers the advantage of quicker response times and improved resilience against sophisticated attacks. For instance, in response to a Distributed Denial of Service (DDoS) attack, the IRS may initiate sequential countermeasures, starting with blocking malicious traffic. Following this initial step, additional countermeasures may be implemented to fortify the network against the attack. Considering the combination of sequential and parallel execution of countermeasures, we define the overall time spent for applying the selected countermeasures in $\boldsymbol{\Theta}$ as

$$T^{\text{Tot}}(\boldsymbol{\Theta}) = \sum_{\theta_c \in \boldsymbol{\Theta_1}} T(\theta_c) + \max\{T(\theta_c) : \forall \theta_c \in \boldsymbol{\Theta_2}\} \tag{8}$$

where $\boldsymbol{\Theta_1}$ and $\boldsymbol{\Theta_2}$ refer to those sets of countermeasures which can be implemented sequentially and in parallel, respectively. Please note that our approach accommodates scenarios where countermeasures are selected only sequentially, only in parallel, or through a combination of both. Therefore, the formula above comprehensively considers all possibilities.

4. Problem formulation and analysis

The optimal countermeasure selection can be formulated as an optimization problem. The objective is maximizing network security while minimizing the associated QoS costs by selecting the best countermeasures, $\boldsymbol{\Theta}$, such that all attacks are addressed. Considering the joint Security and QoS cost functions, the optimization problem can be written as

$$\mathbf{P1}: \underset{\boldsymbol{\Theta}}{\text{argmin}} \left\{ \frac{\sum_{v_n^a \in \mathcal{V}} R_a}{\sum_{\theta_c \in \boldsymbol{\Theta}} \sum_{v_n^a \in \mathcal{V}} \Delta R_a(\theta_c)}, \bar{T}^{\text{Tot}}(\boldsymbol{\Theta}) \right\} \tag{9}$$

subject to

$$\mathbf{C1.1}: \left|\bar{S}_{\text{att}}(\boldsymbol{\Theta})\right| = \left|\mathcal{V}\right| \tag{10}$$

$$\mathbf{C1.2}: \omega_a \in \bar{\mathcal{W}}(\theta_c) \Rightarrow \omega_a \notin \bar{\mathcal{W}}(\theta_j): \quad \forall \omega_a, \forall \theta_c, \theta_j \in \boldsymbol{\Theta}, \theta_j \neq \theta_c \tag{11}$$

where $\boldsymbol{\Theta}$ is the countermeasure selection decision vector to be obtained. Please note that among the selected countermeasures, there may be a combination of atomic and non-atomic countermeasures, as well as a mix of sequential and parallel implementations. $\bar{T}^{\text{Tot}}$ is the normalized spent time from (8). This normalization is needed to ensure both security and Time terms in the objective function are in the same range. The objective function (9) aims to find the best countermeasures (subsets) to be selected to address the attacks in order to maximize the security (please note that the security term is reversed) while minimizing the implementation time. Constraint (10) implies that the selected countermeasures in the solution vector $\boldsymbol{\Theta}$ shall cover all the attacks in the network. Constraint (11) states that one attack type can be addressed by only one countermeasure (either atomic or non-atomic).

The problem resembles the WSCP. In WSCPs, we try to cover elements using a collection of sets, each of which has a weight, with an optimization goal [38,39]. Similarly, in our problem, $\mathcal{V}$ is the set of all the elements (attacks across nodes) to be covered and there exists a maximum of C subsets (countermeasures) covering the elements in $\mathcal{V}$, where each subset has a security-vs-QoS weight. The goal is maximizing the security while keeping the implementation time low by finding the vector $\boldsymbol{\Theta} \subseteq \{\theta_1, \cdots, \theta_C\}$ such that $|\bar{S}_{\text{att}}(\boldsymbol{\Theta})| = |\mathcal{V}|$, i.e., covering all the detected attacks and no attack belongs to two sets (countermeasures).

Fig. 1 illustrates how the countermeasure selection problem can be represented as a WSCP. Each sub-figure depicts a different combination of countermeasures that collectively mitigate all identified attacks. In sub-figure (a), three countermeasures are sufficient to cover all attacks, while in sub-figure (b), four countermeasures achieve the same coverage. Both represent valid solutions to the problem, but each carries a different total weight, reflecting the trade-off between security effectiveness and QoS cost. The figure therefore demonstrates that the objective of the WSCP formulation is to identify the optimal combination of countermeasures that minimizes this overall weight.

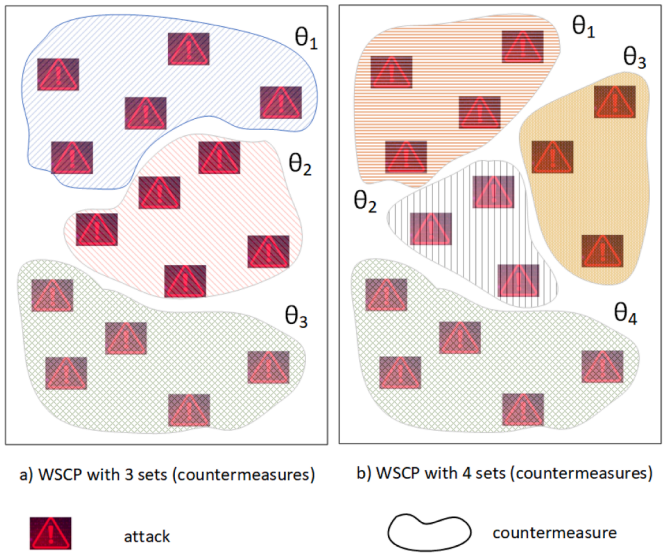


Fig. 1. Illustration of the mapping between the countermeasure selection problem and the WSCP, showing two possible ways to cover all attacks.

Given the pair $(\mathcal{V}, S_{\text{att}})$, the WSCP problem needs to find $\boldsymbol{\Theta}$, which involves comparing all the possibilities in combining atomic, non-atomic, sequential and parallel countermeasures such that the two constraints are met. The WSCP is NP-complete and cannot be solved in a polynomial time [39]. Hence, we resort to learning algorithms to find a semi-optimal solution.

5. MOMDP And MOQL solution for countermeasure selection problem

The question is how to choose an appropriate policy for the set covering problem. That is, how to cover all attacks with a selected number of countermeasures ensuring the security and implementation time are optimized. In the following, we model our problem with the Markov Decision Process (MDP) and later resort to MOQL to solve the problem.

5.1. MOMDP Model for countermeasure selection problem

In this section, we define our problem in the form of MDP which is a sequential decision-making process, where the agent observes its current state and makes the best decision to transition to the next state.

The countermeasure selection problem in our work is a Multi-Objective (MO) problem that can be modelled by the Multi-Objective Markov Decision Process (MOMDP) [40]. An MOMDP can be defined by tuple $\langle S, \mathcal{A}, \mathcal{P}, \mathbf{r}, \gamma, D \rangle$, where $S, \mathcal{A}, \mathcal{P}$ and D represent the state, action, probability and initial state space, respectively. Moreover, $\mathbf{r} \in \mathbb{R}^M$ represents the vector-valued reward function for M objectives and $(0 \leq \gamma \leq 1)$ is the discount factor.

5.1.1. State space

The state space in our MDP model is finite-dimensional space which is the observation of the environment. In our model, we define the set of states as

$$S = \{s_t | s_t = (\mathcal{V}, \mathbf{R})\}, \tag{12}$$

where $\mathcal{V}$ contains the list of all detected attacks across nodes, and $\mathbf{R} = \{R_1, \cdots, R_{\bar{A}}\}$ represents the RF vector for all the detected attack types, $\bar{A}$ shows the total number of detected attack types in the system.

5.1.2. Action set

The action space in MDP refers to the actions that the agent can perform in a particular state which results in the agent ending up in another state. Let us define the action space as

$$\mathcal{A} = \{a_t | a_t = \mathbf{\Theta}_t\}, \tag{13}$$

where $\mathbf{\Theta}_t$ is the selected countermeasures at time instant t. This means at time instant t the agent can select a set of countermeasures to optimize both security and the QoS cost. The selected action transitions the agent into a new state where the values in $\mathcal{S}$ will be updated.

5.1.3. Reward function

Let us denote $\mathbf{r}_t = (r_t^1, \cdots, r_t^M)$ as the immediate vector-valued reward at time instant t, where r_t^m represents the immediate reward for the mth objective. Since our problem involves two objectives, namely security and QoS costs, we have $M = 2$. We define the reward of each objective as

$$r_t^1 = \frac{\sum_{\theta_c \in \mathbf{\Theta}} \sum_{v_n^a \in \mathcal{V}} \Delta R_a(\theta_c)}{\sum_{v_n^a \in \mathcal{V}} R_a} \tag{14}$$

$$r_t^2 = -\sum_{c=1}^{|\mathbf{\Theta}_t|} \bar{T}^{\text{Tot}}(\theta_c) \tag{15}$$

The value of the reward is related to the action (i.e., countermeasure selection decision vector $\mathbf{\Theta}$) the agent takes at a particular state.

5.1.4. Policy

In RL, a policy $\pi : \mathcal{S} \to \mathcal{A}$ is a mapping from state to action. The state-value function $\mathbf{V}_\pi(s) : \mathcal{S} \to \mathbb{R}^M$, which is the expected cumulative reward of a state in MDP, maps the state s to the vector of expected rewards and is used to evaluate a policy π. Let us define state-value function as

$$\mathbf{V}_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} \mathbf{r}_k \middle| s_t = s \right]. \tag{16}$$

In the above equation, $\mathbf{V}_\pi(s)$ is a multi-objective value function which evaluates a state by considering the immediate reward of each of the objectives through the T time horizon by following the policy π. The standard MDP aims to maximize the cumulative immediate rewards; hence the optimal state-value function can be written as

$$\mathbf{V}_{\pi^*}(s) = \max_\pi \mathbb{E}_\pi \left[\mathbf{r}_t + \gamma \mathbf{V}_\pi(s') | s_t = s \right] \tag{17}$$

where s' is the next state by following policy π. This means we would like to find an optimal policy π^* with which the cumulative immediate rewards starting from state s is maximized. Similarly, we can define the state-action value function as

$$\mathbf{Q}_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} \mathbf{r}_k \middle| s_t = s, a_t = a \right], \tag{18}$$

The optimal value for a state-action pair is represented by $Q^*(s, a)$ and can be obtained through

$$\mathbf{Q}_{\pi^*}(s, a) = \mathbf{r}_t + \mathbb{E}\left[\max_{a'} \gamma \mathbf{Q}_{\pi^*}(s', a') \right] \tag{19}$$

5.2. MOQL Algorithm for countermeasure selection problem

In order to solve the MOMDP model and find the optimal policy for eq. (17), we propose a Multi-Objective Q-Learning (MOQL) algorithm, which combines Monte Carlo and dynamic programming to learn an offline policy without prior knowledge [41]. In this section, we present a MOQL algorithm to address the optimal countermeasure selection decision problem. It should be noted that the probability space introduced in the MOMDP model can be neglected due to the episodic nature of the RL algorithm and the fact that Q-learning does not need the explicit transition probability model. RL assumes that the model of the system (e.g. state transition matrix) is not known by the decision-maker, i.e., model-free, which is the same in our scenario.

Let us rewrite our MO optimization problem in **P1** as

$$\mathbf{P2}: \max_\pi (f^1(\pi), \cdots, f^M(\pi)), \tag{20}$$

subject to

$$\mathbf{C2.1}: \pi \in \Pi, \tag{21}$$

$$\mathbf{C1.1},$$

$$\mathbf{C1.2},$$

where π is a policy belonging to the policy space Π and $f^m(\pi)$ represents the mth objective in the joint multi-objective function, which has been converted to a maximization problem.

Considering that the environment has several objectives simultaneously, different optimal policies can be found, where they differ by the priority/weight given to each objective. Moreover, in the presence of multiple conflicting objectives the concept of optimality transforms to Pareto optimality [42]. For this reason, we seek to find a Pareto front of solutions for the countermeasure selection problem.

Conventionally, the quality of a solution is determined by its Pareto dominance with respect to other solutions. In particular let $\Pi = \{\mathbf{\Theta}_1, \cdots, \mathbf{\Theta}_Z\}$ be the set of Z solutions where $\mathbf{\Theta}_z$ is the zth solution as described before. Considering two solutions $\mathbf{\Theta}_1$ and $\mathbf{\Theta}_2$ for a given problem with M conflicting objectives with $V_{\mathbf{\Theta}}^m(s)$ being as the value of the mth objective of solution $\mathbf{\Theta}$ (for all $m \in [1 \quad M]$), the Pareto-dominance is defined as

Definition 1. It is said that $\mathbf{\Theta}_1$ Pareto-dominates $\mathbf{\Theta}_2$ (i.e., $\mathbf{\Theta}_1 \succ \mathbf{\Theta}_2$) if $V_{\mathbf{\Theta}_1}^m(s) \geq V_{\mathbf{\Theta}_2}^m(s)$ for all $m \in [1 \quad M]$, and there exist some $p \in [1 \quad M]$ such that $V_{\mathbf{\Theta}_1}^p(s) > V_{\mathbf{\Theta}_2}^p(s)$.

This means that $\mathbf{\Theta}_1$ is considered better than $\mathbf{\Theta}_2$ if it achieves at least the same value in all objectives and strictly higher value in at least one of them.

Definition 2. The set of non-dominated solutions is defined as [43]

$$S = \{\mathbf{\Theta}_z | \nexists \mathbf{\Theta}_x \succ \mathbf{\Theta}_z, \forall \mathbf{\Theta}_z, \mathbf{\Theta}_x \in \Pi\} \tag{22}$$

We adopt a single-policy MOQL approach in order to learn Pareto-optimal solutions. Single-policy MOQL algorithms employ scalarization functions to define a utility over a vector-valued policy. This reduces the dimensionality of the multi-objective to a single, scalar value [44,45]. In the scalarization functions, the weight allows us some control over the nature of policy which is going to be found by tuning the weight values for each objective according to their priority for us. We employ a linear scalarization function and define it below.

Definition 3. Considering $\mathbf{w}$ as a weight vector, a scalarization function f is a function that projects a value-vector objective function $\mathbf{J}$ to a scalar, i.e., $f(\mathbf{J}, \mathbf{w})$.

In our MOQL approach, the value-vector objective function can be mapped to $Q(s, a, m)$, and the scalarized Q-value can be defined as

$$\hat{Q}(s, a) = \sum_{m=1}^{M} w_m \cdot Q(s, a, m) \tag{23}$$

This yields a single state-action value by combining the Q-values of the two objectives, each weighted by a respective objective weight.

In the MOQL algorithm the Q-table is extended to include the Q-values for each of the objectives. Since the Q-value is a recursive equation, it is updated for each objective as

$$Q(s, a, m) \leftarrow$$

Algorithm 1 Linear scalarized ϵ-greedy method.

Input: $\mathcal{A}$, ϵ, d, $Q(s,a,m)$ $\forall m$
Output: a
1: $\hat{\mathbf{Q}}(s) \leftarrow \{\}$
2: **for** each action $a \in \mathcal{A}(s)$ **do**
3: obtain $Q(s,a,m)$, $\forall m$
4: obtain ω_m, $\forall m$
5: $\hat{Q}(s,a) \leftarrow \sum_{m=1}^{M} w_m \cdot Q(s,a,m)$
6: $\hat{\mathbf{Q}}(s) \leftarrow \hat{Q}(s,a)$
7: **end for**
8: $\epsilon \leftarrow \epsilon - d$
9: $X \leftarrow binornd(\epsilon)$
10: **if** $X = 1$ **then**
11: $a \leftarrow$ a random action from $\hat{\mathbf{Q}}(s)$
12: **else**
13: $a \leftarrow \text{argmax}_{a'} \hat{\mathbf{Q}}(s)$
14: **end if**

Algorithm 2 MOQL Algorithm.

Input: reward vector, maxSteps , $\alpha, \gamma, \hat{\mathbf{Q}}(s)$
Output: $Q(s,a,m)$, $\forall m$
1: steps $\leftarrow 0$
2: **for** each episode **do**
3: Generate an initial state s randomly.
4: **while** $\mathcal{V} \neq \emptyset$ OR $steps \leq maxSteps$ **do**
5: Select a from current state s using **Alg. 1**
6: Take action a, observe state s' and reward vector $\mathbf{r}$
7: $a' \leftarrow \text{argmax}_a \hat{\mathbf{Q}}(s')$
8: **for** each objective m **do**
9: $Q(s,a,m) \leftarrow Q(s,a,m) + \alpha[r(s,a,m) + \gamma Q(s',a',m) - Q(s,a,m)]$
10: **end for**
11: $s \leftarrow s'$
12: steps $\leftarrow$ steps + 1
13: **end while**
14: **end for**

$$Q(s,a,m) + \alpha[r(s,a,m) + \gamma Q(s',a',m) - Q(s,a,m)] \tag{24}$$

An ϵ-greedy approach is selected for evaluating the actions in each state of the MOQL environment and the algorithm is presented in Algorithm 1. In Algorithm 1, first a list $\hat{\mathbf{Q}}(s)$ is created containing the Q-value of all of the actions that can be taken from a state (see line 1). Then for each of the actions that can be taken from the state s, a) the Q-value for each of its objectives is obtained, b) the associated objective weight is obtained, c) the linear scalarized function is calculated and d) the result is stored in the list $\hat{\mathbf{Q}}(s)$ (see lines 2–7). Later an ϵ-greedy method is employed considering Bernoulli distribution where a random action with probability ϵ is selected and a greedy action (the best one) from the $\hat{\mathbf{Q}}(s)$ list is selected with probability 1-ϵ (see lines 8–12). To encourage exploration of the state-action space early in training, the exploration rate ϵ is initially set to 0.9, meaning that 90% of the time, a random action is selected. As training progresses, the focus gradually shifts toward exploitation through the use of the exploration decay rate d parameter, which steadily reduces the value of ϵ over time.

Now we incorporate the above linear scalarized ϵ-greedy approach into a MOQL algorithm to solve the countermeasure selection problem. As depicted in Algorithm 2, the MOQL algorithm runs for several episodes where each episode runs until all of the detected attacks in $\mathcal{V}$ are addressed or a predefined maximum number of iterations, denoted by $maxSteps$, is reached. For each episode, a random state is initiated as the starting point of the agent (see line 2). Then Algorithm 1 is executed to choose an action from the starting state based on the ϵ-greedy method (see line 4). Upon taking the action, the agent observes the following state and the reward vector (see line 5), and the action with the best Q-value function from the Q-value list is selected (see line 6). The Q-table is updated for each of the objectives as shown in line 8. Finally, the agent enters the next state.

5.3. Deep Q-Learning

The Q-learning algorithm employs the Q-table to store reward values for each state-action pair. This can become inefficient as the problem's complexity grows, leading to an exponential increase in the number of state-action pairs and, consequently, the size of the Q-table, making training both computationally demanding and memory-intensive. Google DeepMind introduced Deep Q-Network (DQN), merging deep neural networks with the Q-learning methodology [46]. Deep learning techniques excel in two key areas: function approximation and representation learning. These capabilities enable them to efficiently learn concise, low-dimensional representations from raw, high-dimensional data [47]. Most importantly, DQN replaces the Q-table with a neural approximator whose parameter count does not scale with the size of the state space. This allows the network to generalize by interpolating between unseen but similar states, thereby reducing the need for exhaustive exploration. Empirically, we observe these scalability benefits in our setup; as the network size increases, DQN maintains stable training and inference times, whereas tabular Q-learning's complexity grows sharply with the number of state-action pairs (see Section 6.5). Consequently, for large-scale networks, DQN achieves more efficient training and more robust deployment while optimizing the same action-value objective.

Having said that, using neural networks to approximate the Q-function presents challenges, as the correlations between the Q-values $Q(s,a)$ and the target values $Q(s',a')$ can lead to an unstable training process[46]. proposed the technique of experience replay which can enhance the convergence of the algorithm. All the experience obtained from the agent's interaction with the environment is stored in the experience memory. During training, the agent samples random mini-batches, disrupting the sequence of training samples. This randomness helps prevent the network from settling into one of a local minimum.

The stability of the training process is further enhanced by the use of a second Q-network, commonly referred to as the target network [46]. This network is an identical copy of the main Q-network, known as the estimation network. Its weights are not trained, but they are periodically synchronized with the parameters of the main Q-network. The loss function is described by Eq. 25 where β' and β are parameters of the estimation and target networks, respectively.

$$L(\beta) = E\left[(\mathbf{r}_t + \gamma \max_{a'} Q(s',a'|\beta') - Q(s,a|\beta))^2\right] \tag{25}$$

The DQN algorithm is capable of optimizing multiple objectives; in this work, the focus is placed on the security and time objectives. This formulation is reflected in Eq. 25, where the reward function $\mathbf{r}_t$ integrates both objectives to guide the learning process toward a balanced trade-off between security effectiveness and operational efficiency.

Fig. 2 presents a detailed overview of the Deep Q-Network model, highlighting the architecture of the estimation and target networks along with their optimization process. The experience replay buffer stores the agent's past interactions with the environment in the form of tuples $e = (s,a,r,s')$, where s and s' represent the current and next states, a the chosen action, and r the received reward. Mini-batches of these experiences are randomly sampled and used to train the estimation network, which consists of two fully connected hidden layers with 24 and 12 neurons, respectively, each followed by a ReLU activation, and a final dense output layer producing the Q-values $Q(s,a;\beta)$. The target network has an identical architecture and provides stable target Q-values $Q(s',a';\beta')$ used to compute the loss $L(\beta)$. The target network's parameters β' are periodically updated from the estimation network after every N steps. The loss function is minimized through gradient descent, and the estimated Q-values guide the agent's action selection to maximize cumulative rewards over time.

The state and action space, as well as the training procedure of the DQN algorithm are the same as in the MOQL method, with the key difference being that instead of storing information in a table, DQN represents

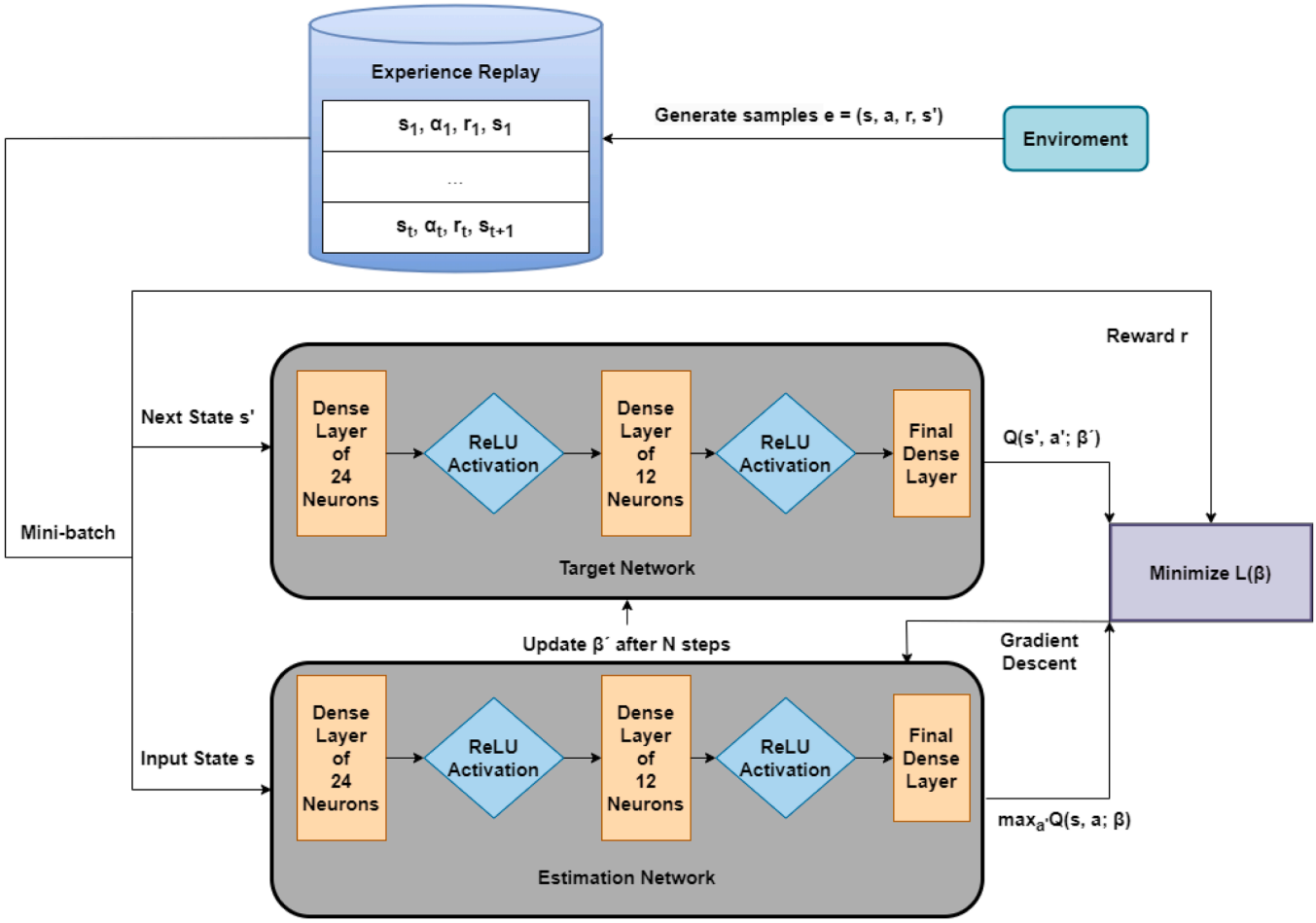


Fig. 2. Deep Q-Network Architecture with Experience Replay and Target Network. The Experience Replay stores past transitions sampled to train the Estimation Network, while the Target Network provides stable Q-value targets.

it through the weights of a neural network. Due to the use of a neural network, the state space is structured as a one-hot encoded vector $\mathcal{S}$, where the length of $\mathcal{S}$ corresponds to $\bar{A}$, the total number of detected attack types in the system. Additionally, based on the set of detected attacks $\mathcal{V}$, the total Risk Factor is computed and appended to $\mathcal{S}$, forming the input state s that is fed into the neural network.

The output layer of the neural network is designed to contain as many neurons as the number of available actions (countermeasures) in the action space $\mathcal{A}$. The optimal action for each input state s is then determined by selecting the neuron with the highest output value, ensuring that the chosen countermeasure corresponds to the maximum Q-value estimation.

6. Simulation results

In this section, we present the results of our experimental analysis, using both the MOQL algorithm, as well as the DQN algorithm, making use of a realistic dataset for 5G core-related attack, collected in the context of the H2020 SANCUS project [35]. In this study, we focus on software-related attacks that target the core network. These attacks were identified as the most common 5G-related threats within the EU-funded SANCUS project. The SANCUS dataset encompasses a diverse set of attack categories, including the core network layer, as listed in Table 2. Each attack instance is annotated with the initial RF, RF reduction achieved by each countermeasure, and the time required for deployment and execution–all of which were incorporated into our evaluation. The RF metric combines two components: severity, ranging from 1 to 10 and derived from the Common Vulnerability Scoring System (CVSS), and probability, ranging from 0 to 1, based on a uniform distribution to ensure unbiased attack generation. Considering both likelihood and severity, we provide the initial RF values for each attack type in Table 2. The corresponding time and RF reduction values vary across different mitigation actions. The detailed parameter values are omitted here for conciseness and can be found in [48]. The dataset's composition reflects realistic 5G network conditions and operational settings, providing strong real-world applicability. For simulation purposes, we assumed a uniform distribution for attack generation to ensure unbiased coverage of all attack types in the evaluation. Furthermore, we adopted the list of countermeasures defined in the SANCUS project, as summarized in Table 3. These include both atomic and non-atomic countermeasures designed to mitigate the identified attacks. These countermeasures can be deployed within SAs.

6.1. Simulation setting

As regards the DQN algorithm, the estimation network is trained after every 5 episodes, while its weights are copied to the target network after every 10 episodes. The architecture for both networks is detailed in Fig. 2. The neural networks of the DQN approach where trained and tested using Tensorflow [49]. The weights of both networks are initialised using He initialisation [50]. The training process was conducted on an Intel Xeon CPU operating at 2.2 GHz, using the Adam optimiser [51] and a learning rate of 0.01.

Table 2
List of attacks.

Abbr.	Attack types	Initial RF
ω_1	AMF-targeted attacks	7
ω_2	PFCP related attacks	8
ω_3	SUCI attack	6
ω_4	API injections	9
ω_5	Application layer DoS attacks over environments	9
ω_6	API patching and improper assets management	8
ω_7	ML-enhanced DDoS attacks on the application layer	9
ω_8	PDU session creation	7
ω_9	AI-powered attacks	9
ω_{10}	Multi-faced attacks	9

Table 3
List of countermeasures.

Abbr.	Countermeasure(s)
θ_1	Notify the network operator/provider
θ_2	Block the attacker
θ_3	Isolate the device
θ_4	Relaunch the infected node
θ_5	Reconfigure the VNF
θ_6	Replace the infected node
θ_7	Change the network topology
θ_8	Block the attacker & isolate the device
θ_9	Notify the network operator/provider & block the attacker
θ_{10}	Notify the network operator/provider & isolate the device
θ_{11}	Notify the network operator/provider & block the attacker & isolate the device

Table 4
Parameters of both algorithms during simulation.

Parameter	MOQL	DQN
Learning rate α	0.1	0.001
Discount factor γ	0.99	0.7
Maximum iteration steps per episode	20	20

Table 4 contains the values of the Bellman equation parameters (as shown in 24), as well as the number of iterations per episode (ending condition), which were fixed for both algorithms during the simulation. For the MOQL algorithm, a relatively high learning rate α and discount factor γ were chosen for faster convergence and to emphasize long-term rewards during the learning process. In contrast, for the DQN algorithm, lower values of α and γ were adopted to ensure stable gradient-based updates within the neural network, thereby reducing the risk of divergence caused by large parameter variations. Finally, the maximum number of iteration steps per episode was set to 20, as this was sufficient for the agents to handle all detected attacks in the network during each episode.

Our simulation process begins by training the MOQL and the DQN algorithms on the Sancus problem. Depending on the type of experiment, certain parameters are fixed, while others vary. As a final step, each possible state of the state space is given to each algorithm as an initial point. Subsequently, both algorithms produce corresponding solutions for each state. We then compute and average the security and time objective values across all solutions generated by each agent.

6.2. Impact of objective weights on the overall joint utility

In this experiment, we explore the impact of the weights assigned to the security objective w_1 and the time objective w_2, on the average utility function scores produced by the agents of each algorithm. Three different sets of weights were used; the first set ($w_1 = 0.9, w_2 = 0.1$) prioritizes the security objective, the second set ($w_1 = 0.5, w_2 = 0.5$) aims on balanced solutions between the two objectives, while the third set ($w_1 = 0.1, w_2 = 0.9$) emphasizes minimizing the time objective. In all three cases, the MOQL algorithm was trained for 410,000 episodes, whilst the DQN algorithm was trained for 3200 episodes. Note that the number of episodes for each algorithm was selected based on the experimental analysis presented in Section 6.3.

Figs. 3a and 3b present the average utility function scores for each pair of w_1 and w_2 values when using the MOQL and the DQN algorithm, respectively. Note that the objective is to maximize security while minimizing time. The values of w_1 and w_2 determine the rewards given to the agents when selecting countermeasures during the training process, emphasizing either or both objectives. As w_2 increases, the reward function increasingly penalizes time-consuming actions, leading the agent to favor faster countermeasures (since the time objective is minimized), which explains the observed decrease in the average time score. When security is prioritized ($w_1 = 0.9, w_2 = 0.1$), both algorithms achieve a high average security score of approximately 38. However, this comes at the cost of a higher time objective, as the selected countermeasures, while effective in security, require significant time investment. In contrast, when time is prioritized ($w_1 = 0.1, w_2 = 0.9$), both algorithms achieve the lowest average time objective of 5, but at the expense of security, which is significantly reduced. Finally, when the objectives are balanced[1] ($w_1 = w_2 = 0.5$), both MOQL and DQN achieve decent performance, with joint utilities of 7.43 and 7.47, respectively.

The confidence intervals in Fig. 3b showcase the variability of the results across multiple runs of the same experiment with different random seeds. The 95% confidence intervals are narrow, typically below ± 0.3 utility units for both Security and Time scores, indicating that the algorithm produces highly consistent performance across runs. Confidence intervals are not shown in Fig. 3a because the Q-Learning results demonstrate significant stability, with deviations below ± 0.1 utility units across runs. This high consistency is to be expected, as unlike neural models, tabular Q-Learning is not sensitive to factors such as weight initialization.

Overall, both algorithms effectively adjust their performance based on the assigned weights, demonstrating their ability to prioritize objectives accordingly. Notably, DQN exhibits a slight advantage when security is emphasized ($w_1 = 0.9, w_2 = 0.1$), achieving a lower time objective without sacrificing security, compared to MOQL. For all subsequent experiments, the weight values are set to $w_1 = 0.5$ and $w_2 = 0.5$ to balance both objectives.

To evaluate the performance of the proposed approach, we compare it against two benchmark methods. Although the specific characteristics of our problem make direct comparisons with existing studies difficult, these benchmarks capture the most relevant aspects of security–cost trade-offs.

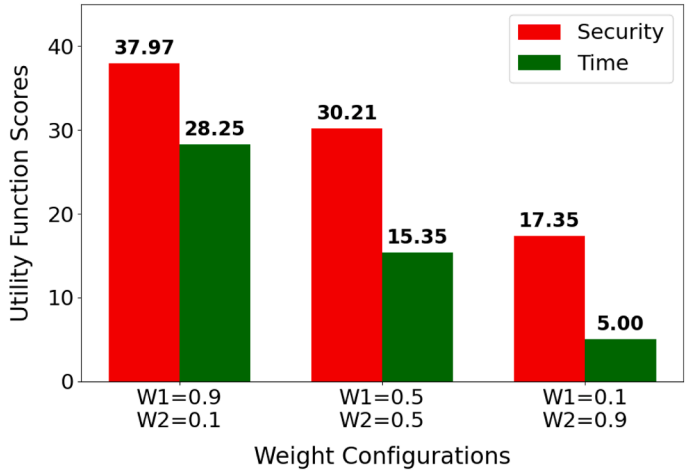
- SecCost: This benchmark is adapted from the study in [52], which emphasizes the relationship between security effectiveness and associated monetary costs. In this approach, countermeasures are prioritized and selected based on their combined impact on security and cost, using the following metric:

$$\frac{|\mathcal{W}(\theta_c)|\, P(\theta_c)}{T(\theta_c)} \tag{26}$$

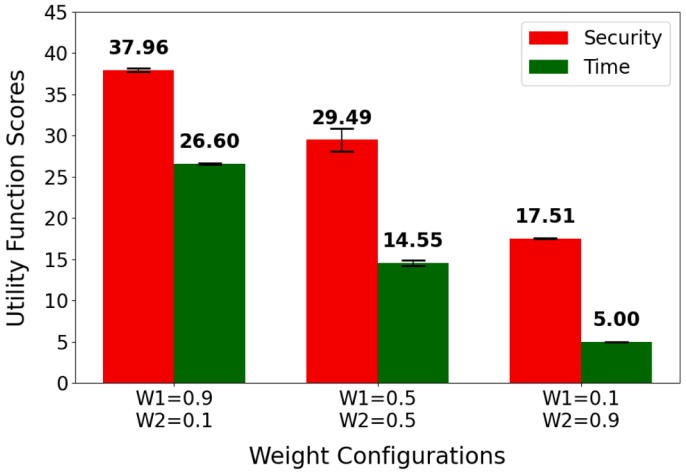
  where $P(\theta_c)$ denotes the probability of preventing an attack, normalized as a percentage. In this benchmark, countermeasures are ranked in descending order of effectiveness until all potential attacks are mitigated.
- Rule-Based: This baseline represents a classical selection strategy where countermeasures are chosen solely to maximize security, without considering cost or efficiency trade-offs.

Fig. 3c presents a comparison between the proposed algorithms (MOQL and DQN) and the two benchmark methods, SecCost and Rule. The benchmarks focus on a single objective, either cost or security, while our algorithms are designed to balance both objectives through w_1 and

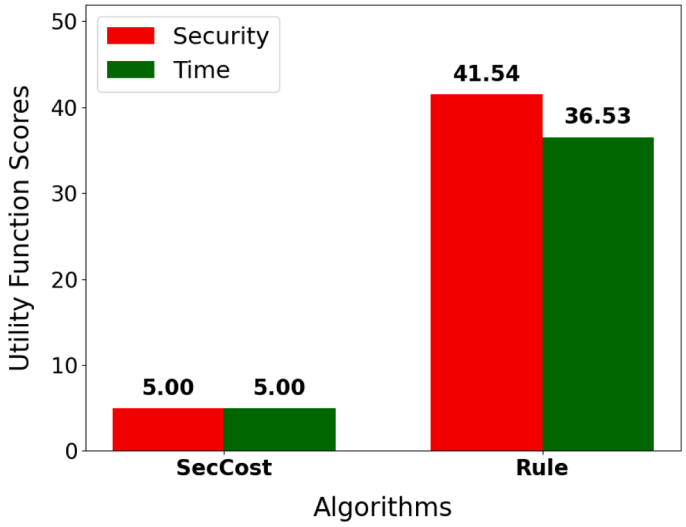
[1] Since (8) includes sequential and parallel terms, security and time values are not necessarily equal even with equal weights.



(a) MOQL algorithm



(b) DQN algorithm



(c) Comparison of MOQL and DQN with related work (SecCost & Rule)

Fig. 3. Impact of w_1 and w_2 on the joint utility for MOQL and DQN algorithms (top) and comparison with related work (bottom).

w_2. As shown in the figure, SecCost prioritizes cost reduction, leading to minimal time utility but at the expense of security performance. In contrast, the Rule-based approach maximizes security without considering cost efficiency. However, both MOQL and DQN agents are capable of adapting dynamically based on the assigned weights, thus achieving superior overall utility. Even when a single objective is emphasized, they produce more cost-effective and secure recommendations, maintaining high performance across different optimization priorities.

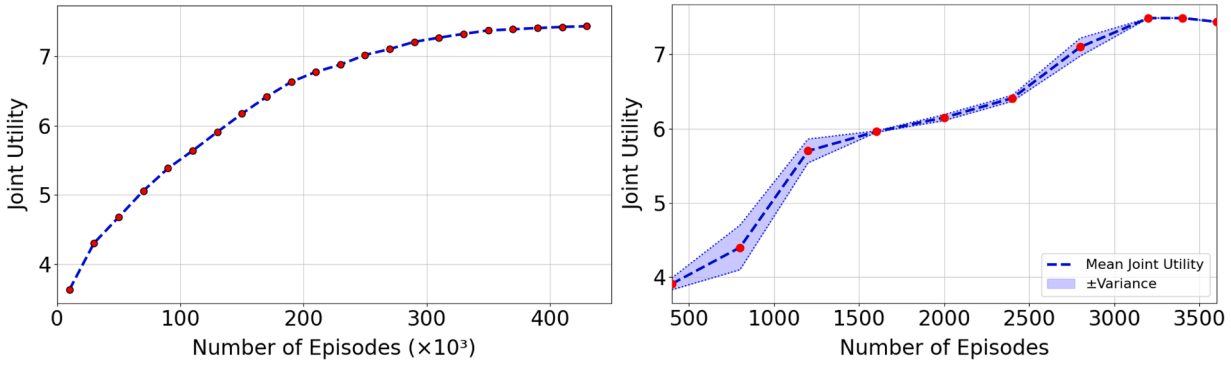
6.3. Impact of number of episodes on the overall joint utility

The effectiveness of both algorithms depends significantly on the fine-tuning of their hyperparameters. Specifically, the number of training episodes and the d decay rate have a great effect on the algorithms' performance. These parameters need to be optimized concurrently; as the number of training episodes increases, the decay rate of d needs to be reduced in order to manage the exploration-exploitation tradeoff.

Fig. 4a illustrates the average joint utility of the solutions generated by the MOQL agent when trained over a progressively increasing number of episodes, ranging from 10,000 to 430,000. Concurrently, the decay rate for each training process is slightly decreased as the number episodes rise, from an initial value of 0.1 to a value of 0.0001. The decay rate values were tuned to maximize the agent's joint utility, as for instance, a low decay rate with only 10,000 episodes leads to non-convergence. The data clearly demonstrates that the joint utility improves with the number of episodes, but it shows signs of leveling off as it approaches 400,000 episodes.

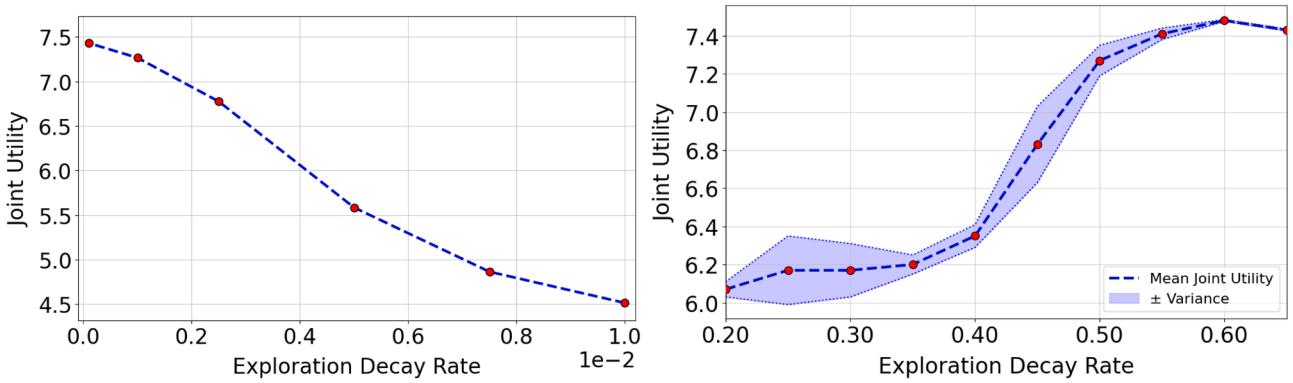
Fig. 4b demonstrates the performance of the DQN algorithm, in terms of the joint utility, as the number of episodes increase. The decay rate d is also decreased gradually from 0.7 to 0.6. Unlike MOQL, DQN requires significantly fewer episodes to achieve meaningful performance improvements, but each DQN episode takes considerably more time (as further discussed in Section 6.5). The plot shows a steep initial increase in joint utility during early training, followed by a more gradual rise. Optimal performance is reached at 3200 steps, after which the utility values converge and remain nearly constant.

The variance bands in Fig. 4b illustrate the stability of the learning process across multiple runs. As shown, the variance is noticeably higher at lower episode counts, which is expected since the agent has not yet sufficiently explored the environment. As training progresses and the



(a) MOQL algorithm (b) DQN algorithm

Fig. 4. Impact of number of episodes on the joint utility for MOQL and DQN algorithms.



(a) MOQL algorithm (b) DQN algorithm

Fig. 5. Impact of exploration decay on the joint utility for MOQL and DQN algorithms.

number of episodes increases, the variance steadily decreases, indicating that the agent's policy becomes more consistent.

6.4. Impact of decay rate on the overall joint utility

Similarly to the experiments of the previous section, we examine how the decay rate d affect the joint utility for both algorithms, while keeping the number of episodes fixed according to the experimental analysis in Section 6.3. In the case of Fig. 5a, the number of episodes is fixed to 410,000, while the decay rate d gradually rises from a value of 10^{-4} to 0.01. It is evident that there is a negative correlation between the decay rate and the joint utility. As the exploration decay rate increases, the joint utility decreases stressing the importance of finding the optimal exploration-exploitation tradeoff. A higher decay rate forces the algorithm to exploit learned policies too early, potentially preventing it from fully discovering better strategies.

Fig. 5b shows the effect of the exploration decay rate on the agent's joint utility. The joint utility initially increases gradually for lower decay rates, then exhibits a sharper rise as the decay rate approaches 0.45-0.5, before stabilizing at higher values. This pattern indicates that moderate decay rates enable a balanced trade-off between exploration and exploitation, allowing the agent to discover more effective policies before gradually focusing on exploiting the learned knowledge.

The variance bands highlight that at lower decay rates, the variance is relatively high due to excessive exploration. As the decay rate increases, the variance steadily decreases, reflecting improved policy convergence and consistent learning behavior. Beyond approximately 0.5, the variance becomes minimal, suggesting that the agent has reached a stable and well-optimized policy.

6.5. Time complexity of the proposed solutions

The aim of our final experiment is to examine the time complexity of the algorithms. Both algorithms are applied on variations of the security recommendation problem of different sizes, in terms of the number of countermeasures and attack types. Both algorithms are trained until they achieve a comparable level of average joint utility, ensuring a fair comparison of computational efficiency.

Fig. 6 demonstrates the training time (in minutes) of both algorithms as the problem size increases. For the MOQL algorithm, the training time remains relatively low for smaller problem sizes, requiring less than 2 minutes for configurations with fewer than 13 countermeasures/attacks. However, the time complexity increases exponentially, exceeding 9 minutes for the 15×15 problem size. It is evident that MOQL struggles to scale efficiently as the state space expands, likely due to its reliance on tabular Q-learning and value iteration over a large state-action space.

In contrast, the DQN algorithm exhibits significantly more stable time complexity across different problem sizes. Regardless of the number of countermeasures and attack types, DQN maintains a relatively constant training time, remaining around 3-4 minutes even for larger problem instances, demonstrating superior scalability than MOQL.

Fig. 7 illustrates the Pareto fronts of the MOQL algorithm when applied to the countermeasure selection problem, for different w_1, w_2 weight configurations. Each curve represents the tradeoff between security and time, with different weights assigned to each objective. The red curve corresponds to the 20k iteration setting, while the other curves represent increasing iteration counts (80k, 140k, and 410k iterations). The labeled points on the red curve indicate the corresponding w_1, w_2 values, showing how different weight assignments impact the trade-off.

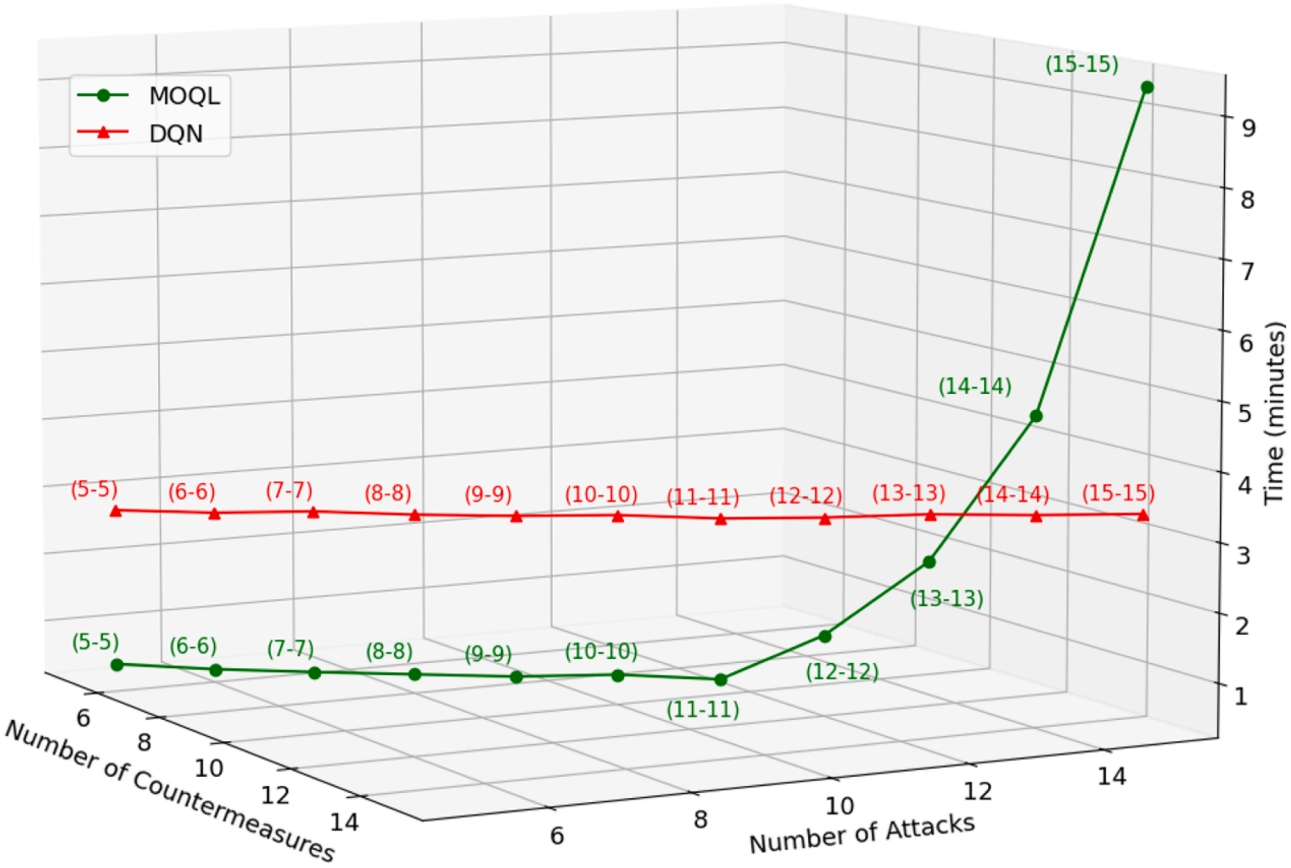


Fig. 6. Time complexity of both algorithms as the problem's size increases.

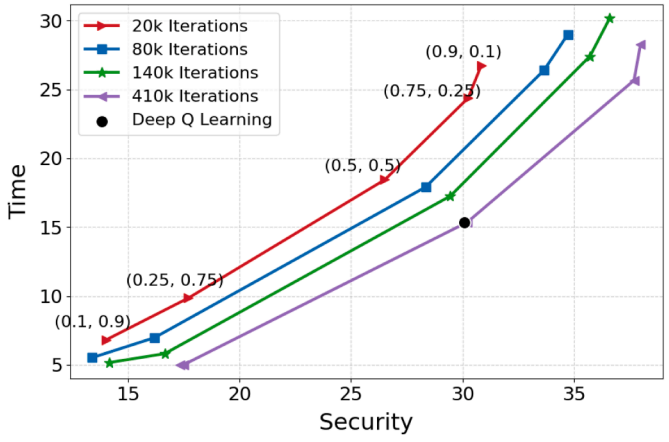


Fig. 7. Pareto fronts for different number of iterations when using the MOQL algorithm.

Additionally, the plot includes a black dot representing the DQN algorithm's performance (average joint utility) when $w_1 = w_2 = 0.5$. This visualisation complements the time complexity analysis, as it demonstrates that MOQL achieves comparable levels of security in approximately one-third of the time required by DQN when applied to the problem. Consequently, MOQL is more efficient for security recommendation problems of small to medium scale, while DQN is better suited for large-scale security recommendation tasks, where its neural network-based approach offers greater scalability and adaptability.

6.6. Discussion

The proposed IRS can be implemented as a modular engine within current network architectures. It can comprise several components, including an event reception and scheduling module, an optimisation controller (the IRS core), and a database module that stores countermeasure information and its effectiveness against different attack types. Complementary engines, such as IDS, can operate as SAs deployed as network functions within the 5G infrastructure to identify and classify attacks. The deployment of the IDS components within SAs has been detailed in our earlier work [5].

Once an attack is detected, the IDS can communicate security-related information–such as severity, probability, affected components, and QoS parameters–to the IRS through a messaging system, such as a Kafka bus. The IRS, encoded as a Python module and containerised as a Virtual Machine, can operate under the Management and Orchestration (MANO) framework of the 5G core. Upon receiving alerts, the IRS can perform the proposed Security-Time-QoS optimisation to determine the most appropriate countermeasures. These results can then be sent to the Security Orchestrator, which translates the optimisation output and interacts with the MANO Orchestrator for deployment. Finally, MANO can apply the selected countermeasures in the underlying 5G network via REST APIs. Such an integrated deployment of both the IDS and IRS through the MANO framework in a 5G environment has been implemented and experimentally validated within the EU SANCUS project [35].

Regarding the limitations and main challenges, the MOQL algorithm relies on a tabular representation, which limits its scalability and makes it difficult to train for large or complex network topologies. As the size of the state-action space increases, the memory requirements and convergence time of MOQL grow significantly. Although this issue is miti-

gated by the DQN algorithm through the use of function approximation, DQN introduces its own challenges, such as sensitivity to hyperparameter configurations, neural network architecture, and training stability.

7. Conclusion

In this work, we addressed the challenge of dynamically selecting countermeasures in IRSs by jointly considering both security effectiveness and QoS costs. Our contributions are threefold. First, we modelled the countermeasure selection process by supporting both sequential and parallel execution of atomic and non-atomic countermeasures, enabling a realistic representation of IRS operations. Second, we formulated a joint Security-vs-QoS optimization problem and proved its NP-hardness by mapping it to the WSCP, highlighting the complexity of achieving optimal decisions. Third, we introduced a novel MOQL-based solution, used for the first time in this context, and compared it against a DQL approach and two other benchmarks through extensive simulations using project-driven experimental data. The results demonstrate that MOQL is particularly effective for small- to medium-scale problems, while DQL shows superior scalability and adaptability in large-scale scenarios.

In more detail, both MOQL and DQN effectively balanced the two objectives when equal weights were assigned, achieving joint utility scores of 7.43 and 7.47, respectively. For the DQN algorithm, the 95% confidence intervals remained within ± 0.3 utility units for both security and time objectives, while the MOQL algorithm showcased even smaller variations, consistently below ± 0.1 utility units. These results demonstrate that both methods are highly stable across multiple runs, showing minimal sensitivity to random initialization and training variations. Moreover, both algorithms significantly outperformed the benchmark approaches, confirming their ability to adapt effectively even when the optimization emphasizes a single objective.

In the future, we aim to extend the model by incorporating additional objectives such as monetary cost and energy consumption. We also plan to integrate this optimization with the Intrusion Detection System, enabling joint decision-making between detection and response. Another promising direction is Security Agent placement, which influences both detection coverage and system resilience. We can explore how their positioning and allocated resources affect the overall detection capability across the network. Beyond these extensions, the framework can evolve toward AI-native 6G networks and ultra-dense IoT environments, where autonomous and distributed security management becomes essential. In such systems, countermeasure selection could rely on multi-agent reinforcement learning to adapt policies dynamically, while lightweight edge agents collaborate through federated optimization to maintain Security-QoS balance under limited resources. An interesting direction for future work is the integration of the two Q-learning approaches. A hybrid framework could exploit the interpretability and computational efficiency of MOQL in smaller or well-defined state spaces, while dynamically switching to the DQN model in larger or more complex environments. This adaptive design would allow the system to maintain interpretability where feasible and scalability where necessary, effectively combining the complementary strengths of both algorithms.

CRediT authorship contribution statement

Arash Bozorgchenani: Writing – review & editing, Writing – original draft, Supervision, Project administration, Methodology, Formal analysis, Conceptualization; **Dimitris Manolakis:** Writing – original draft, Visualization, Validation, Software, Methodology, Data curation; **Antonios Lalas:** Writing – review & editing, Supervision, Formal analysis, Data curation.

Data availability

Data will be made available on request.

Declaration of competing interests

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Antonios Lalas reports financial support was provided by Horizon Europe. If there are other authors, they declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme, in the frame of the NATWORK project (Net-Zero self-adaptive activation of distributed self-resilient augmented services) under Grant Agreement No 101139285. The security recommendation problem addressed in this work was originally defined during the SANCUS project (Grant Agreement No 952672), which also contributed a realistic dataset simulating 5G core-related attacks. This foundation enabled the refinement of the problem using the CERTH testbed and the adaptation of Q-learning and Deep Q-learning solutions within the NATWORK framework.

References

[1] U.K. Dutta, M.A. Razzaque, M. Abdullah Al-Wadud, M.S. Islam, M.S. Hossain, B.B. Gupta, Self-Adaptive scheduling of base transceiver stations in green 5G networks, IEEE Access 6 (2018) 7958–7969.
[2] S.B. Prathiba, G. Raja, A.K. Bashir, A.A. AlZubi, B. Gupta, SDN-Assisted Safety message dissemination framework for vehicular critical energy infrastructure, IEEE Trans. Ind. Inf. 18 (5) (2022) 3510–3518.
[3] M.A. Ferrag, L. Maglaras, A. Argyriou, D. Kosmanos, H. Janicke, Security for 4G and 5G cellular networks: a survey of existing authentication and privacy-preserving schemes, J. Netw. Comput. Appl. 101 (2018) 55–82.
[4] Christopher J. Alberts, Audrey J. Dorofee, James F. Stevens, Carol Woody, OCTAVE-S Implementation Guide, 2005, https://doi.org/10.1184/R1/6575852.v1.
[5] A. Bozorgchenani, C.C. Zarakovitis, S.F. Chien, T.O. Ting, Q. Ni, W. Mallouli, Novel modeling and optimization for joint cybersecurity-vs-Qos intrusion detection mechanisms in 5G networks, Comput. Netw. 237 (2023) 110051.
[6] S.A. Zonouz, H. Khurana, W.H. Sanders, T.M. Yardley, RRE: A game-Theoretic intrusion response and recovery engine, IEEE Trans. Parallel Distrib. Syst. 25 (2) (2014) 395–406.
[7] A. Bozorgchenani, C.C. Zarakovitis, S.F. Chien, Q. Ni, A. Gouglidis, W. Mallouli, H.S. Lim, Intrusion response systems for the 5G networks and beyond: a new joint security-vs-Qos optimization approach, IEEE Trans. Network Sci. Eng. (2024) 1–14.
[8] P. Nespoli, F.G. Mármol, J.M. Vidal, A bio-Inspired reaction against cyberattacks: AIS-Powered optimal countermeasures selection, IEEE Access 9 (2021) 60971–60996.
[9] Nespoli, Pantaleone and Gomez Marmol, Felix and Kambourakis, Georgios, AISGA: Multi-Objective parameters optimization for countermeasures selection through genetic algorithm, in: Proceedings of the 16Th International Conference on Availability, Reliability and Security, ARES 21, Association for Computing Machinery, New York, NY, USA, 2021.
[10] Y. Guo, H. Zhang, Z. Li, F. Li, L. Fang, L. Yin, J. Cao, Decision-Making for intrusion response: which, where, in what order, and how long?, in: ICC 2020 - 2020 IEEE International Conference on Communications (ICC), 2020, pp. 1–6.
[11] Li, Xuan and Zhou, Chunjie and Tian, Yu-Chu and Qin, Yuanqing, A dynamic decision-Making approach for intrusion response in industrial control systems, IEEE Trans. Ind. Inf. 15 (5) (2019) 2544–2554.
[12] F. Li, Y. Li, S. Leng, Y. Guo, K. Geng, Z. Wang, L. Fang, Dynamic countermeasures selection for multi-path attacks, Comput. Secur. 97 (2020) 101927.
[13] B. Fila, W. Wideł, Exploiting attack-defense trees to find an optimal set of countermeasures, in: 2020 IEEE 33Rd Computer Security Foundations Symposium (CSF), 2020, pp. 395–410.
[14] O. Stan, R. Bitton, M. Ezrets, M. Dadon, M. Inokuchi, Y. Ohta, T. Yagyu, Y. Elovici, A. Shabtai, Heuristic approach for countermeasure selection using attack graphs, in: 2021 IEEE 34Th Computer Security Foundations Symposium (CSF), 2021, pp. 1–16.
[15] S. Iannucci, O.D. Barba, V. Cardellini, I. Banicescu, A performance evaluation of deep reinforcement learning for model-Based intrusion response, in: 2019 IEEE 4Th International Workshops on Foundations and Applications of Self* Systems, 2019, pp. 158–163.
[16] S. Iannucci, V. Cardellini, O.D. Barba, I. Banicescu, A hybrid model-free approach for the near-optimal intrusion response control of non-stationary systems, Future Generat. Comput. Syst. 109 (2020) 111–124.
[17] Z. Luo, Fog node intrusion detection and response based on SVMIF and INSGA-II algorithm, Syst. Soft Comput. 7 (2025) 200330.
[18] M. Hamad, A. Finkenzeller, M. Kühr, A. Roberts, O. Maennel, V. Prevelakis, S. Steinhorst, REACT: Autonomous intrusion response system for intelligent vehicles, Comput. Secur. 145 (2024) 104008.

[19] M. Bashendy, A. Tantawy, A. Erradi, Intrusion response systems for cyber-physical systems: a comprehensive survey, Comput. Secur. 124 (2023) 102984.
[20] M. Lelarge, Coordination in network security games: a monotone comparative statics approach, IEEE J. Sel. Areas Commun. 30 (11) (2012) 2210 - 2219. Cited by: 29; All Open Access, Green Open Access.
[21] L.P. Rees, J.K. Deane, T.R. Rakes, W.H. Baker, Decision support for cybersecurity risk planning, Decis. Support Syst. 51 (3) (2011) 493–505.
[22] J. Simon, A. Omar, Cybersecurity investments in the supply chain: coordination and a strategic attacker, Eur. J. Oper. Res. 282 (1) (2020) 161–171.
[23] J.A. Paul, X.J. Wang, Socially optimal IT investment for cybersecurity, Decis. Support Syst. 122 (2019) 113069.
[24] M. Paul, Jomon A. and Zhang, Decision support model for cybersecurity risk planning: a two-stage stochastic programming framework featuring firms, government, and attacker, Eur. J. Oper. Res. 291 (1) (2021) 349–364.
[25] F. Uuganbayar, Ganbayar and Yautsiukhin, Artsiom and Martinelli, Fabio and Massacci, Optimisation of cyber insurance coverage with selection of cost effective security controls, Comput. Secur. 101 (2021) 102121.
[26] P. Nespoli, D. Papamartzivanos, F. Gómez Mármol, G. Kambourakis, Optimal countermeasures selection against cyber attacks: a comprehensive survey on reaction frameworks, IEEE Commun. Surv. Tutor. 20 (2) (2018) 1361–1396.
[27] M. Lyu, H. Habibi Gharakheili, V. Sivaraman, PEDDA: Practical and effective detection of distributed attacks on enterprise networks via progressive multi-stage inference, Comput. Netw. 233 (2023) 109873.
[28] J.R. Rose, M. Swann, K.P. Grammatikakis, I. Koufos, G. Bendiab, S. Shiaeles, N. Kolokotronis, IDERES: Intrusion detection and response system using machine learning and attack graphs, J. Syst. Archit. 131 (2022) 102722.
[29] T. Hussain, A. Beard, L. Chen, C. Nugent, J. Liu, A. Moore, From machine learning based intrusion detection to cost sensitive intrusion response, in: 2022 6Th International Conference on Cryptography, Security and Privacy (CSP), 2022, pp. 124–130.
[30] J. Chen, J. He, W. Li, W. Fang, X. Lan, W. Ma, T. Li, A hierarchical unmanned aerial vehicle network intrusion detection and response approach based on immune vaccine distribution, IEEE Internet Things J. 11 (20) (2024) 33312–33325.
[31] J. Lee, S. Park, S. Shin, H. Im, J. Lee, S. Lee, ASIC Design for real-Time CAN-Bus intrusion detection and prevention system using random forest, IEEE Access 13 (2025) 129856–129869.
[32] K. Vieira, F.L. Koch, J.B.M. Sobral, C.B. Westphall, L.J.L. de Souza, Autonomic intrusion detection and response using big data, IEEE Syst. J. 14 (2) (2020) 1984–1991.
[33] Y. Huang, L. Su, Design of intrusion detection and response mechanism for power grid SCADA based on improved LSTM and FNN, IEEE Access 12 (2024) 148577–148591.
[34] M. El-Hajj, AI-Powered Threat detection and response: leveraging machine learning for real-Time intrusion detection systems (IDS) using network traffic data, in: 2025 5Th Intelligent Cybersecurity Conference (ICSC), 2025, pp. 84–90.
[35] Sancus Project, Sancus project, 2023, [Online], https://sancus-project.eu/.
[36] C.-J. Chung, P. Khatkar, T. Xing, J. Lee, D. Huang, NICE: Network intrusion detection and countermeasure selection in virtual network systems, IEEE Trans. Dependable Secure Comput. 10 (4) (2013) 198–211.
[37] H.A. Kholidy, Autonomous mitigation of cyber risks in the cyber-physical systems, Future Generat. Comput. Syst. 115 (2021) 171–187.
[38] L. Golab, F. Korn, F. Li, B. Saha, D. Srivastava, Size-Constrained weighted set cover, in: 2015 IEEE 31St International Conference on Data Engineering, 2015, pp. 879–890.
[39] A. Bozorgchenani, D. Tarchi, W. Cerroni, On-Demand service deployment strategies for fog-as-a-Service scenarios, IEEE Commun. Lett. 25 (5) (2021) 1500–1504.
[40] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, W. Matusik, Prediction-Guided multi-Objective reinforcement learning for continuous robot control, in: Proceedings of the 37Th International Conference on Machine Learning, 2020.
[41] M. Langlois, R.H. Sloan, Reinforcement learning via approximation of the Q-function, J. Exp. Theor. Artif. Intell. 22 (3) (2010) 219–235.
[42] A. Bozorgchenani, F. Mashhadi, D. Tarchi, S.A. Salinas Monroy, Multi-Objective computation sharing in energy and delay constrained mobile edge computing environments, IEEE Trans. Mob. Comput. 20 (10) (2021) 2992–3005.
[43] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[44] P. Vamplew, J. Yearwood, R. Dazeley, A. Berry, On the limitations of scalarisation for multi-objective reinforcement learning of pareto fronts, in: W. Wobcke, M. Zhang (Eds.), AI 2008: Advances in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 372–378.
[45] K. Van Moffaert, M.M. Drugan, A. Nowé, Scalarized multi-objective reinforcement learning: novel design techniques, in: 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), 2013, pp. 191–199.
[46] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529-533.
[47] T.T. Nguyen, V.J. Reddi, Deep reinforcement learning for cyber security, IEEE Trans. Neural Netw. Learn. Syst. 34 (8) (2023) 3779–3795.
[48] A. Bozorgchenani, et al., Report on definition and modelling of MiU, 2023, Deliverable 4.2 EU SANCUS Project, https://sancus-project.eu/wp-content/uploads/2024/01/SANCUS_D4.2-Final.pdf.
[49] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015, Software available from tensorflow.org, https://www.tensorflow.org/.
[50] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-Level performance on imagenet classification, in: 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1026–1034.
[51] D. Kingma, J. Ba, Adam: a method for stochastic optimization, in: International Conference on Learning Representations (ICLR), San Diega, CA, USA, 2015.
[52] S. Tamjidi, A. Shamelli-Sendi, Intelligence in security countermeasures selection, J. Comput. Virol. Hack. Techni. 19 (1) (2023) 137–148. https://doi.org/10.1007/s11416-022-00427-8