IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 17, 2021, accepted August 23, 2021, date of publication September 3, 2021, date of current version September 13, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3109856

Deep Multi-Task Conditional and Sequential Learning for Anti-Jamming

ROBERT BASOMINGERA[1], (Student Member, IEEE), AND YOUNG-JUNE CHOI[2], (Senior Member, IEEE)
[1]Department of Computer Engineering, Ajou University, Suwon 16499, South Korea
[2]Department of Software and Computer Engineering, Ajou University, Suwon 443-749, South Korea

Corresponding author: Young-June Choi (choiyj@ajou.ac.kr)

This work was supported by the Industrial Infrastructure Program for Fundamental Technologies funded by the Ministry of Trade, Industry and Energy (MOTIE), South Korea, under Grant N0002312.

ABSTRACT Multi-task learning provides plenty of room for performance improvement to single-task learning, when learned tasks are related and learned with mutual information. In this work, we analyze the efficiency of using a single-task reinforcement learning algorithm to mitigate jamming attacks with frequency hopping strategy. Our findings show that single-task learning implementations do not always guarantee optimal cumulative reward when some jammer's parameters are unknown, notably the jamming time-slot length in this case. Therefore, to maximize packet transmission in the presence of a jammer whose parameters are unknown, we propose deep multi-task conditional and sequential learning (DMCSL), a multi-task learning algorithm that builds a transition policy to optimize conditional and sequential tasks. For the anti-jamming system, the proposed model learns two tasks: sensing time and transmission channel selection. DMCSL is a composite of the state-of-the-art reinforcement learning algorithms, multi-armed bandit and an extended deep-Q-network. To improve the chance of convergence and optimal cumulative reward of the algorithm, we also propose a continuous action-space update algorithm for sensing time action-space. The simulation results show that DMCSL guarantees better performance than single-task learning by relying on a logarithmically increased action-space sample. Against a random dynamic jamming time-slot, DMCSL achieves about three times better cumulative reward, and against a periodic dynamic jamming time-slot, it improves by 10% the cumulative reward.

INDEX TERMS Ad hoc network, deep learning, jamming attack, multi-armed bandit, spectrum sensing.

I. INTRODUCTION

In wireless communication, network interference happens when nearby communicating nodes transmit at the same time with closer frequencies, resulting in a jamming attack if done intentionally. In a jamming attack, the attacker named jammer intentionally uses its electromagnetic energy to interfere with radio frequencies used by communicating nodes in the jammed area. Both jamming and network interference affect the communication between nodes by corrupting transmitted packets. Communicating nodes with the capability of dynamically operating the radio spectrum access may escape or reduce the impact caused by the jammer's activities by transmitting their data on a frequency different than the one affected by the jammer. A node's ability to properly switch between frequencies needs an efficient channel switching policy to adapt to the jammer's operating scheme. One way of developing such policy, especially against a dynamic jammer, is to use reinforcement learning (RL) algorithms [1], [2].

The convergence of RL algorithms to the optimal value in a reasonable time is the ultimate goal for all RL algorithms; however, not all of them always converge optimally. The reason can be deduced in the nature of the task (completely random vs. repeating tasks), the communication capabilities between learning agents, or the used algorithm itself. In terms of the algorithm used, the learning algorithm type may help speed up the learning process. For example, an offline learning algorithm generates a better performance, in most cases (not always guaranteed), than an online algorithm due to its functional approach. Offline learning uses a dataset of logged experiences; hence, does not require more interactions with the environment. In contrast, online learning

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan M. Abu-Mahfouz.

requires actively and continuously interacting with the environment. The continuous interactions result in online learning being deemed slower or converging to a minima value instead of the optimal value. Therefore, offline learning can speed up learning and improve performance; however, it is not easy or feasible to use offline learning for all kinds of tasks, since it is applicable in settings with only fixed batches of interactions with the environment, without further interactions [3]–[6].

Among other techniques that can be used to improve the performance of learning algorithms, especially in an anti-jamming system, is to use multi-task learning. When tasks are related or dependent, the model's performance can be significantly improved by learning multiple tasks instead of the single-task alone. Multi-task learning has a few disguises such as joint learning, predicting multivariate responses, learning to learn, and learning with auxiliary tasks [7]–[9]. In this work where anti-jamming is the study case, we consider tasks to be spectrum sensing (deciding sensing time) and data transmission (deciding the transmission channel), which are conditional and sequential because they always run in sequential order and the actions of the next task (deciding the transmission channel) are conditioned to its predecessor's choice (deciding sensing time) and performance. In multi-task learning, just as in single-task learning, the agent interacts with the environment within a series of observations, actions, and rewards. The ultimate objective is to maximize the cumulative reward by optimizing the action-selection policy.

This work proposes a multi-task learning algorithm that improves the model's long-term return by learning conditional and sequential tasks, compared to single-task learning. The proposed multi-task learning algorithm jointly learns both tasks through two connected steps:

- The first step exhaustively decides the sensing time (in a multi-arm bandit approach) from a continuous action-space (sensing time), which is discretized and continuously updated adaptively to the observed performance.
- The second step selects the transmission channel (in a deep-Q-network-like approach) from a finite action-space sample (transmission channels).

The main contributions of this work are summarized as:

- Through empirical analysis, we show that the performance of single-task learning-based anti-jamming systems suffers when their input data (channel status) are not perfectly collected.
- We design an algorithm of continuously updating the action-space to maximize the cumulative reward of the model.
- We propose a multi-task learning algorithm that offers better convergence of the anti-jamming model against a static and dynamic jammer.

Table 1 shows a summary of the symbols used in this paper. The rest of this paper is organized with related works in Section II. The proposed system model is explained in Section III before showing the simulation results in Section IV. Section V presents the conclusion.

TABLE 1. Definition of symbols.

Symbols	Description
$\mathcal{K}$	Tasks
$\mathcal{S}(i)$	State space for task i
$s_t(i)$	State for task i at time i
$\mathcal{A}(i)$	Action-space for task i
$a_t(i)$	Action for task i at time t
$\mathcal{R}$	Total reward for an episode
$\mathbf{r}_t$	Reward for an iteration t
$\varepsilon(i)$	Exploration rate for task i
$\varepsilon_{min}(i)$	Minimum Epsilon value for task i
$\bar{\varepsilon}(i)$	Decaying factor of ϵ of task i
lr	Learning rate
$\mathcal{T}_{min}$	Initial minimum action value
$\mathcal{T}_{max}$	Initial maximum action value
$\mathbf{\Phi}$	Initial action space-sample size
$\mathbf{d}$	Difference between two consecutive action values
$\mathcal{D}$	Replay memory for the DQN
$\mathcal{UCB}$	Upper-confidence-bound
$\mathcal{X}$	Performed iterations for $\mathcal{K}(1)$ while $\mathcal{K}(2)$ is explored
$\widehat{\mathcal{X}}$	Number of iterations a given action from $\mathcal{A}(1)$ has been explored and exploited
$\mathcal{V}$	Value (rewards) for each action from $\mathcal{A}(1)$
η	Population size of the action sample space
$\mathbf{P}^{(1+)}$	Probability that each action from the $\mathcal{A}(i)$ has been exploited at least once while $\mathcal{K}(2)$ is explored
κ	Iterations needed until probability $\mathbf{P}^{(1+)}$ is satisfied
$\mathcal{J}$	Counter to keep track of how many exploration for $\mathcal{K}(1)$ and exploitation for $\mathcal{K}(2)$ are done within an episode
$\mathbf{D}$	Deduplication value
$\mathbf{T}$	Total iterations for a single episode

II. RELATED WORKS

A. ANTI-JAMMING

Jamming defenses can be grouped into three categories [10], [11], based on nodes' strategy, as:

- Competition strategy: communicating nodes remain in the jammed area and keep transmitting or receiving. They rely on techniques such as null steering to cancel jammed signals or increase their transmission power to operate at a higher signal-to-interference-plus-noise ratio to outpower the jammer.
- Spatial retreat strategy: communicating nodes evacuate from the jammer's transmission range in order to be able to transmit or receive data.
- Spectral retreat strategy: communicating nodes do frequency hopping to retreat from the channel in which the jammer is operating.

Each defense strategy has drawbacks, benefits, and requirements, depending on elements such as the ability or resources held by communicating nodes, the jammer's strength, or the network environment. For each strategy, the communicating node can use heuristics, game theory, or machine learning techniques. This work focuses on learning-based anti-jamming for spectral retreat strategy against both jamming attacks and interference among nodes. Anti-jamming learning-based systems have been extensively studied in different approaches which, recently, focus on improving the learning algorithm's performance by solving

jamming in multi-dimensional or multi-layers [12]–[15], handling the lack of meaningful or clear jammer's information [16]–[18], or improving on the slowness of the algorithms [19]–[22].

In [12], they formulated a cross-layer scheme that simultaneously approaches routing selection, channel selection, and power allocation. In [13] they proposed a multi-dimensional design that combines frequency-motion-antenna to improve signal-to-interference power. Both schemes improve the performance provided by a single dimension or a single layer solution. However, the performance improvement comes with additional requirements, such as the need for mobility, which may be intricate if the jammer has a similar ability. In other works such as [15], they proposed a jamming-aware routing approach with a reinforcement learning-based cooperation framework allowing nodes to make decisions on whether to do data transmission, defend, or jam the network. In case the communicating nodes cannot implement multi-dimensional solutions, having access to the jammer's information or configurations may assist in mitigating the attacks. Unfortunately, it is not trivial for the communicating nodes to know the jammer's configuration early. Meanwhile, whenever communicating nodes have no information about the jammer, anti-jamming activities may require more strategies. Therefore, to deal with the lack of information about the jammer, [16] developed an approach that takes advantage of the jamming signal to achieve the mission where the communicating node, a drone, leverages the jammer signal. In [18], they study anti-jamming in an unknown and changing environment by utilizing the waterfall spectrum information directly; thus, constructing an anti-jamming Q-network. The user does a whole band sensing every 1ms and holds the spectrum data in 200ms. In [17], they used an evasion attack against the jammer itself so that communicating nodes do not give up their communication without a fight against the jammer. The transmitter builds a machine learning model to predict the channels' states. The attacker relies on deep learning to launch the attack. As a defense strategy, the transmitter then launches attacks on the attacker's cognitive engine to affect its convergence. Although by attacking the jammer communicating nodes spend additional resources, they can continue their communication if successful.

Competing against the jammer is meaningless if not done successfully within an acceptable time. One way to improve timing is by speeding the learning process. To address the slow-convergence of anti-jamming learning algorithms, [19], [20] accelerated the learning process using hotbooting transfer learning. In [20], the power control experiences from similar scenarios are used to initialize the Q-table, while in [19], the value of the quality function toward each action-state pair is initialized with the experiences from similar scenarios. Nevertheless, other works such as in [11], [21], instead of conventional reinforcement learning, used federated learning which allows training the model across multiple nodes; thus, making the model converging faster or better as it relies on several resources.

Most of these works, which fall into the spectral retreat strategy category, use frequency hopping. Frequency hopping consists of switching radio signals among different frequency channels to transmit data packets. For communicating nodes to switch between the transmission channels, it is customary and useful to first identify the status of channels (idle or busy). Channel status is defined by identifying the power level in the channel for a given time, which is not an instant action but rather an action executed for a period called sensing time. Meanwhile, sensing time is the time spent by communicating nodes sniffing on a given channel to identify whether the channel is busy or idle. Numerous studies such as [23]–[29] have studied about setting or using dynamic spectrum access/sharing in multichannel wireless networks. In [23], they designed an adaptive dynamic sensing schedule framework. The design is done by investigating the trade-off between sensing performance and achievable throughput regarding time-varying fading, based on adjusting the sensing time adaptively. Setting sensing time basing on fading is helpful for communication. However, it may not be enough in the presence of dynamic jammer nodes such as the ones designed in [30]–[33].

Three techniques may be used as strategies to assist channel hopping against a dynamic jammer. First, multi-dimensional solutions can be used to assist frequency hopping. In [14] for example, they proposed a two-dimensional anti-jamming mobile communication system. The communication system, to mitigate jamming attacks, uses both frequency hopping and user mobility. By using mobility, the proposed solution combines both spatial retreat strategy and spectral retreat strategy. This solution may be limited in some cases as not all communicating nodes have mobility capabilities. The second is when some details about the jammer's dynamism are known or may be observed. In that case, the anti-jamming solution may also be dynamic accordingly. In [22] for example, they proposed a pattern-aware anti-jamming approach against a dynamic multi-mode jammer. The communicating node has first to identify the current jamming pattern and then act accordingly with a different model for each jamming model. Third, the communicating node can try to adaptively define a proper spectrum sensing duration without prior information about the jammer's dynamism. However, no study yet handled the dynamic tuning of spectrum sensing duration in the presence of a jammer, without prior information of the jammer's jamming patterns. Thus, current learning-based anti-jamming systems that do frequency hopping against a jammer with different jamming time slot duration may face performance issues if:

- The learning environment's state is channel status, which is observed by a spectrum sensing duration that is arbitrarily decided; meanwhile, only channel selection is learned.
- There is no alternative solution-strategy, such as mobility, that can be relied on when frequency hopping fails.
- There are no prior information about the jammer that can be capitalized on to build the learning model.

B. ANALYSIS OF SINGLE-TASK ANTI-JAMMING

To mitigate jamming by using RL, channel status is defined as the environment's state. The agent observes the channel's status after using the sensing time sniffing the power level in the transmission channels. Meanwhile, channel status results from selected sensing time, despite not being among elements learned by the algorithm in a single-task learning approach. Single-task learning only decides on the transmission channel, and sensing time is decided manually. Concerning this anti-jamming design flow, we analyze a system where single-task learning is used to mitigate jamming attacks when the jammer applies a static and a dynamic jamming time-slot duration (We refer to jamming time-slot the length of the period within which the jammer sends its jamming signal. It can be static for an entire network time or dynamic where the jammer keeps changing the period length).

We formulate anti-jamming as single-task learning in which the transmitter agent interacts with the environment (made of receiver, jammer, and other transmitter nodes) in a sequence of state $\mathcal{S}$, action $\mathcal{A}$, and reward $\mathbf{r}$. The state-space $\mathcal{S}$ represents the status of each channel. The action-space $\mathcal{A}$ represents the actions that can be chosen by the agent at a given iteration, namely the number of transmission channels. The reward $\mathbf{r}$ is the value of a state-action pair, which is $+1$ for a successful transmission and -1 for a failed transmission; thus, define the iteration reward $\mathbf{r}_t$,

$$\mathbf{r}_t : \mathcal{S} \times \mathcal{A} \Rightarrow \{-1, +1\}. \tag{1}$$

Transition policy Π is the online policy that maps already tried actions with given states by using $\gamma = 0.95$ as the discount factor. By following a decaying[1] ϵ, for each iteration, the agent explores the transmission in the environment to learn the jammer activities or exploits the transmission in the environment with the higher expected reward based on previously computed statistics, solved by calculating the optimal state-action value function using deep Q-network (DQN). Both the jammer and transmitter abide by the Assumption 1.

Assumption 1: We assume that the jammer does not know the length of the sensing time used by the transmitter, and the transmitter does not know the length of the jamming time-slot used by the jammer.

Fig. 1 shows the simulation results where an agent uses different sensing times (50 ms, 100 ms, 150 ms, 225 ms, 500 ms, and 900 ms). Each simulation runs against a jammer using 150 ms as its jamming time-slot. The results show that when the sensing time is much lower or higher than the jamming time-slot, the cumulative reward converges to a lower value. When the sensing time is closer to the jamming time-slot, the cumulative reward is maximized, which is the case for 150 ms and 225 ms. Fig. 2a and Fig. 2b show an agent's results with 100 ms sensing time against a jammer with a periodic and random dynamic jamming time-slot, respectively. Accumulated rewards are significantly affected by the dynamism of jamming time-slot, which results in fluctuating cumulative rewards. Therefore, with Assumption 1, we derive Conjecture 2.

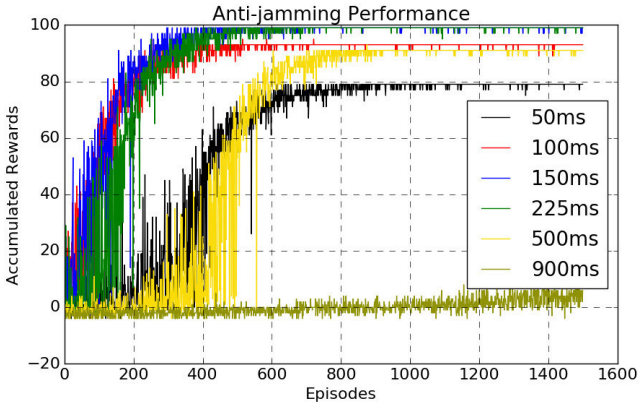


FIGURE 1. Performance comparison of multiple learning anti-jammers using static sensing time-slot lenght (50 ms, 100 ms, 150 ms, 225 ms, 500 ms, and 900 ms) against jammer using 150 ms as its jamming time-slot length.

Conjecture 2: When a transmitter agent does not know the jammer's internal working details, solving jamming attacks as a single-task RL problem, which only learns about channel hopping policy, without optimizing sensing time, does not always guarantee the optimal accumulated reward, especially against a dynamic jamming time-slot.

In contrast to the above single-task analysis, in this work, we propose a dynamically tuned sensing time, to optimize the performance of the model that relies on a sensed channel status. This is not a trivial problem because of the nature of sensing activity. Spectrum sensing is a passive activity that does not trigger any change in the environment's state whenever applied. It is only used to observe the environment. Meanwhile, it cannot claim a reward from the environment when it is the only considered action. This property makes it hard to calibrate sensing time with learning algorithms. However, spectrum sensing can run with other actions, such as data transmission. Therefore, spectrum sensing time can be adjusted along with other actions that allow it to claim a reward for a given selected sensing time. Multi-task learning is, therefore, a candidate to assist in adjusting sensing time.

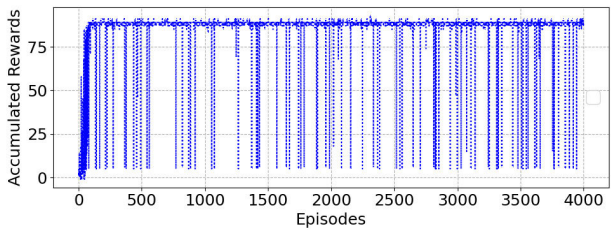
III. SYSTEM MODEL

A. PRELIMINARIES

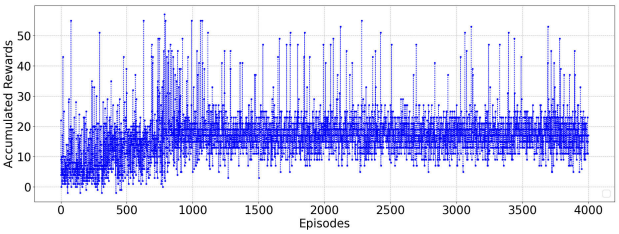
1) MULTI-TASK REINFORCEMENT LEARNING (MTRL)

A multi-task reinforcement learning model learns $\mathcal{K}$ tasks, with $\mathcal{K} > 1$. In this work, we study tasks with a dependency, which makes it possible to solve all of them sequentially and continuously, to yield better performance than single-task solutions. For this case, one task can be approached as a

[1] Epsilon ϵ is used to define the balance between exploration and exploitation of the agent. By following a decaying ϵ we start with the $\epsilon = 1$, and preset minimum epsilon ϵ_{min} and decaying factor $\overline{\epsilon}$. Whenever a reward is received by the agent, the ϵ value is updated as: $\epsilon = \max(\epsilon_{min}, \epsilon \times \overline{\epsilon})$. This results in exponential decay for the ϵ until ϵ value becomes equal or less than the ϵ_{min}.



(a) Periodically changing jamming time-slot.



(b) Randomly changing jamming time-slot.

FIGURE 2. Single-task dynamic jamming time-slot. Performance of a learning anti-jammer with 100 ms sensing time-slot against a jammer with changing jamming time-slot.

single-task; however, the other one (sensing time selection) cannot be solved as a single-task because it is a passive action with no reward scheme. The model's overall reward depends on both tasks' performance. A multi-task model's objective is to solve and optimize each task's performance while exploiting similarities and dependencies between them all. Therefore, it is multi-task learning when the model is optimizing multiple loss functions. The task is a piece of works or steps needed (and related among themselves) to achieve the agent's ultimate goal. In the anti-jamming case, the tasks differ based on input and action distributions, and their loss functions.

2) CONDITIONAL AND SEQUENTIAL TASKS

We refer to conditional and sequential tasks when all the tasks run in a predefined order. During the exploitation of a later task, the next task's actions are conditioned by its predecessor task results. This means, in a pool of $|\mathcal{K}| = \{\mathcal{K}(1), \mathcal{K}(2), \mathcal{K}(3), \ldots\}$ tasks, $\mathcal{K}(1)$ runs before $\mathcal{K}(2)$. Furthermore, $\mathcal{K}(2)$ actions are based on the output of the $\mathcal{K}(1)$ task. The underlying tasks in this work are sensing time selection and data transmission channel selection. From the above conditional and sequential definition, we note that data transmission has to run after spectrum sensing and that data transmission is done on a channel identified as appropriate after spectrum sensing. For simplicity, in the rest of the paper, we refer to sensing time selection as the first task and data transmission channel selection as the second task.

B. MULTI-TASK LEARNING ALGORITHM

The proposed multi-task algorithm is designed as a composite of two sub-models; the first sub-model handling the first task and the second sub-model handling the second task, but both being connected. The algorithm uses a multi-armed-bandit (MAB) algorithm as the first sub-model and a slightly modified version of DQN as the second sub-model. The dyad model makes the deep multi-task conditional and sequential learning algorithm (DMCSL). The choice of sub-models is based on their performance and the nature of tasks under consideration. Specifically, for the first task, we consider a task that has continuous and passive actions. The MAB model is more suitable for sensing duration selection because it can learn to decide which arm (action) to select without relying on the environment's state. MAB decides by analyzing the reward connected with each arm at each iteration from previous iterations. For the second task, we use DQN due to its capability to converge to the optimal policy in due time, even with a nonlinear approximator, to estimate the action-value function.

DMCSL modifies DQN in two ways, to allow it to adapt to connection with MAB to learn multi-tasks. First, the transition stored in the replay memory $\mathcal{D}$ includes the action taken by the first sub-model, because this action has impact on the reward obtained by the whole model; hence, the action of the first sub-model acts as a state member of the state-space of the second sub-model. The transition in the replay memory then becomes $(a_t(1), s_t(2), a_t(2), \mathbf{r}_t, a_{t+1}(1), s_{t+1}(2))$ instead of $(s_t(2), a_t(2), \mathbf{r}_t, s_{t+1}(2))$. Second, for each iteration, Q-learning updates are applied on samples of $(a_t(1), s_t(2), a_t(2), \mathbf{r}_t, a_t(1)', s_t(2)') \sim \mathcal{D}_t$ experience so that the sequence of loss functions becomes:

$$\begin{aligned}\mathcal{L}_t(\theta_t) &= \mathbb{E}_{(a_t(1), s_t(2), a_t(2), \mathbf{r}_t, a_t(1)', s_t(2)')} \\ &\sim \mathcal{D}_t\Big[\Big(r + \gamma \max_{a_t(2)'} Q\Big(a_t(1)', s_t(2)', a_t(2)'; \theta_t'\Big) \\ &\quad - Q\Big(a_t(1), s_t(2), a_t(2); \theta_t\Big)\Big)^2\Big], \end{aligned} \tag{2}$$

where θ_t' are weights of Q-network at iteration t, and θ_t are weights used to estimate the target at iteration t. The θ_t' weights are updated with θ_t. The weights update is not done for every iteration, but rather after every certain number of iterations, similarly as in the original DQN. DMCSL, as illustrated in Fig. 3 and later described in Algorithm 1, works in action-state-action-reward (ASAR) sequence as:

- Action: the agent selects action $a_t(1)$ for task $\mathcal{K}(1)$, which is the duration of spectrum sensing.
- State: the agent observes environment state $s_t(2)$, which includes $a_t(1)$ and channel status.
- Action: the agent selects action $a_t(2)$ for task $\mathcal{K}(2)$, which is the channel for data transmission.
- Reward: the agent observes reward $\mathbf{r}_t$ from the environment, which represents the success or failure of the transmission.

DMCSL develops a transition policy Π in order to maximize the long-term cumulative reward. For every iteration, the model receives a positive iteration reward $\mathbf{r}_t$ if the transmission is successfully, and a negative reward $\mathbf{r}_t$ otherwise,

$$\mathbf{r}_t : \mathcal{A}(1) \times \mathcal{S}(2) \times \mathcal{A}(2) \Rightarrow \{-, 0, +\}. \tag{3}$$

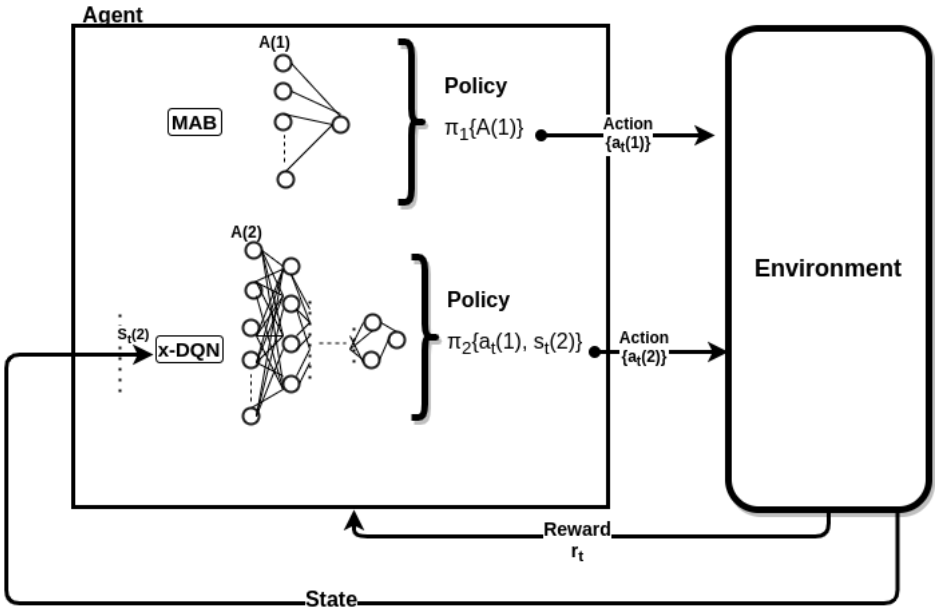


FIGURE 3. DMCSL architecture. The agent works in an action-state-action-reward sequence using two models, MAB and an extended DQN. MAB creates a transition policy Π to decide sensing time $a_t(1)$, then the agent observes the state of the environment $s_t(2)$. Both $s_t(2)$ and $a_t(1)$ are used as input of the x-DQN to create a transition policy that selects the action of transmitting or not, $a_t(2)$. After the action is applied, a reward $\mathbf{r}_t$ is observed by the agent.

The agent cannot see the environment's internal working scheme. It can only observe the interaction between itself and the stochastic environment to identify and use the optimal actions that will lead to maximum long-term rewards. The DMCSL algorithm uses two decaying exploration rates: $\varepsilon(1)$ and $\varepsilon(2)$, which are for $\mathcal{K}(1)$ and $\mathcal{K}(2)$, respectively. For every iteration, they are continuously updated using a decaying rate $\overline{\varepsilon}(i)$ and comparing with the preset minimum epsilon $\varepsilon(i)_{min}$ as:

$$\varepsilon(i) = max\Big(\varepsilon(i)_{min}, \varepsilon(i) \times \overline{\varepsilon}(i)\Big), \tag{4}$$

but, for $\varepsilon(1)$, it is only updated if $a_t(2)$ is not randomly selected (exploitation).

As described in Algorithm 1, [*line* 5 $\sim$ 8] for every iteration in a given episode, with probability $\varepsilon(1)$, the agent selects randomly action $a_t(1)$ from $\mathcal{A}(1)$; otherwise, it selects an action with a higher upper-confidence-bound $\mathcal{UCB}$ as:

$$a_t(1)^* = \mathcal{A}\,[argmax(\mathcal{UCB})], \tag{5}$$

where $\mathcal{UCB}$ is calculated by Equation (6) from [3]:

$$\mathcal{UCB} = \mathcal{V} + \left(2 \times \sqrt{\frac{\ln(\mathcal{X})}{\widehat{\mathcal{X}}}}\right), \tag{6}$$

with $\mathcal{X}$ being the total number of iterations (both exploration and exploitation) within $\mathcal{K}(1)$, $\widehat{\mathcal{X}}$ being the total number of iterations a given action from $\mathcal{A}(1)$ has been explored and exploited, and $\mathcal{V}$ being the value (reward-like) for each action from the action-space $\mathcal{A}(1)$. The agent then uses the selected action $a_t(1)$ to observe the state $s_t(2)$ [*line* 9 $\sim$ 10]. After observing the state, with probability $\varepsilon(2)$, the agent selects randomly the action $a_t(2)$ from $\mathcal{A}(2)$; otherwise, it uses a greedy policy (Equation (7)) to select optimal action from $\mathcal{A}(2)$ as:

$$a_t(2)^* = \arg\max_{\mathcal{A}(2)} Q(\emptyset(a_t(1), s_t(2)), \mathcal{A}(2); \theta), \tag{7}$$

where θ is the weight of the Q-network [*line* 11 $\sim$ 14].

The agent uses the selected action $a_t(2)$ and a reward $\mathbf{r}_t$ is received from the network. The cumulative reward for a given episode is obtained as:

$$\mathcal{R} = \sum_{t=1}^{\mathbf{T}} \mathbf{r}_t, \tag{8}$$

with $\mathbf{T}$ being the total iterations for a single episode [*line* 15 $\sim$ 17]. After the reward is received, if the exploitation was done on $\mathcal{K}(2)$, the agent updates the $\varepsilon(1)$, and increases by 1 the total_times $\mathcal{X}$ and action_times $\widehat{\mathcal{X}}$. The action_values $\mathcal{V}_t$ at time t for the action $a_t(1)$ is updated with Equation (9):

$$\mathcal{V}_t[a_t(1)] = lr \times (\mathbf{r}_t - \mathcal{V}_{t-1}[a_t(1)]), \tag{9}$$

with lr being the learning rate. Otherwise, if the exploration was done for $\mathcal{K}(2)$, the agent does not update the total_times $\mathcal{X}$, the action_times $\widehat{\mathcal{X}}$, and the action_values $\mathcal{V}$, because, if $a_t(2)$ is randomly selected, the reward obtained is not directly due to the selected $a_t(1)$. On [*line* 18 $\sim$ 25] the algorithm keeps counting whenever exploration is done for $\mathcal{K}(1)$ and exploitation is done for $\mathcal{K}(2)$, by increasing the counter $\mathcal{J}$.

Also, still, after the reward is received (whether exploration or exploitation was done for $\mathcal{K}(2)$), the agent updates the

epsilon $\varepsilon(2)$ and the replay memory $\mathcal{D}$. The agent computes the Q-learning targets according to the fixed Q-network weight. A gradient descent step is then performed to optimize the mean square error between Q-network and Q-learning targets [*line* 26 $\sim$ 28]. These steps abide by the flow of the original DQN, with the main difference being on the transitions stored in the replay memory and the loss function that changes at each iteration (We omit some explanatory details about DQN in this paper and invite interested readers to check the work [34]). The agent also checks for two events: if the episode stopping criteria is met (by using a predefined condition, if any is available) [*line* 32 $\sim$ 34], and, if the $\mathcal{A}(i)$ update criteria is satisfied (by using Lemma 11 described in subsection III-C) [*line* 29]. If the update criteria is met, the action-space $\mathcal{A}(1)$ is updated as described in subsection III-D.

C. ACTION-SPACE UPDATE CONDITION

As $\mathcal{K}(1)$ has action-space with infinite action values, it is impossible to supply the model with all the possible actions. Alternatively, supplying the model with as many actions as possible would lead to divergence or much-delayed convergence. Consequently, the necessity of a strategic continuous update of the action-space, where the initial sample size is relatively small. The model is given more options to pick from by continuously updating the action-space, allowing converging the cumulative reward to the optimal value in the long run. The action-space update condition is probabilistically decided by using the probability of *random sampling with replacement* (In statistical analysis, sampling is a process to take a specific number of elements from a given population. *Random sampling with replacement* is when any member of the population can be taken more than once from the population (elements repetition is allowed). All elements in the sample have an equal probability of being selected every time).

The update criteria is satisfied when the probability $\mathbf{P}_x^{(1+)}$ that any action from the action-space sample has been used at least once is greater or equal than a preset threshold value. Probability $\mathbf{P}_x^{(1+)}$ is calculated with Equation (10),

$$\mathbf{P}_x^{(1+)} = 1 - \frac{(\eta - 1)^{\kappa}}{\eta^{\kappa}}, \tag{10}$$

where η is the population size, which is the action-space sample size, and κ is how many times sampling with replacement is done (which is how many times exploration is done for $\mathcal{K}(1)$ while exploitation is done for $\mathcal{K}(2)$). The value κ is obtained by Lemma 11 and compared against a counter number $\mathcal{J}$ that is continuously incremented. Whenever the counter $\mathcal{J}$ is greater or equal to the value κ, the action-space sample of $\mathcal{K}(1)$ is ready to be updated.

Lemma 3: Given an action-space of size η, the number κ of exploration required until the probability to have each action tried at least once reaches $\mathbf{P}_x^{(1+)}$ is:

$$\kappa = \frac{\ln(1 - \mathbf{P}_x^{(1+)})}{\ln(1 - \frac{1}{\eta})}. \tag{11}$$

Algorithm 1 Deep Multi-Task Conditional and Sequential Learning Algorithm

Initialize : $\mathcal{D}$, γ, lr, $\mathcal{A}(2)$, $\varepsilon_0(1)$, $\varepsilon_{min}(1)$, $\overline{\varepsilon}(1)$, $\varepsilon_0(2)$, $\varepsilon_{min}(2)$, $\overline{\varepsilon}(2)$, $\eta = len(\mathcal{A}(1))$, κ, $\mathcal{A}(1)$ with Equation (18), $\mathcal{V} = \eta * [0]$, $\widehat{\mathcal{X}} = \eta * [0]$, $\mathcal{X} = 0$, $\mathcal{J} = 0$, Q, Q^{-}
for episode=1,...,Episodes **do**
  $\mathbf{R} = 0$, $\mathcal{J} = 0$
  for t=1,...,**T do**
    With probability $\varepsilon(1)$
      choose $a_t(1)$ randomly
    Otherwise
      choose $a_t(1)^*$ with Equation (5)
    Execute action $a_t(1)$
    Observe $s_t(2)$
    With probability $\varepsilon(2)$
      choose $a_t(2)$ randomly
    Otherwise
      choose $a_t(2)^*$ with Equation (7)
    Execute action $a_t(2)$
    Observe reward $\mathbf{r}_t$
    Update cumulative reward with Equation (8)
    if Exploitation done for $\mathcal{K}(2)$ **then**
      Update $\mathcal{K}(1)$ epsilon, $\varepsilon(1)$, with Equation (4)
        Increment $\widehat{\mathcal{X}}$ and $\mathcal{X}$ by 1
        update $\mathcal{V}[a_t(1)]$ with Equation (9)
      if Exploration done for $\mathcal{K}(1)$ **then**
        Increment the counting number $\mathcal{J}$ by 1
      end if
    end if
    Store and sample transition $a_t(1)$, $s_t(2)$, $a_t(2)$, $\mathbf{r}_t$, $a_{t+1}(1)$, $s_{t+1}(2)$ in replay memory
    Compute Q-learning targets according to network weights and Optimize MSE using variant of stochastic gradient descent // following normal DQN process
    Update $\mathcal{K}(2)$ epsilon, $\varepsilon(2)$, with Equation (4)
    if $\mathcal{J} >= \kappa$ **then**
      Update $\mathcal{A}(1)$ with Algorithm 2
      Re-compute η as size of $\mathcal{A}(1)$, Re-update κ with Lemma 11
      Reset $\mathcal{J}$ to 0, Resize $\widehat{\mathcal{X}}$ and $\mathcal{V}$ lists
    end if
    if *episode_stopping_criteria* **then**
      End the episodes
    end if
  end for
end for

Proof of Lemma 11: Let η denote the population size of $\mathcal{A}(k)$ (action-space), that during every sampling the number of possibilities is η; meanwhile, if the population is sampled κ times, the total number of ordered options is:

$$\eta \times \eta \times \eta \times \ldots \times \eta = \eta^{\kappa}. \tag{12}$$

Each of these η^{κ} options is a random sample since all elements have an equal selection probability. We are interested in

how to find the probability $\mathbf{P}^{(1+)}$ that any random element has been used at least once. That is the probability that a random element is in a random option among the η^{κ} options.

In reference to the probability *complementary rule*, $\mathbf{P}^{(1+)}$ can be calculated as:

$$\mathbf{P}_x^{(1+)} = 1 - \mathbf{P}_x^{(0)}, \tag{13}$$

with $\mathbf{P}_x^{(0)}$ the probability that a random element has not been selected at all. The total number of options that do not contain a given random element is:

$$(\eta - 1)\times(\eta - 1)\times(\eta - 1) \times \ldots \times (\eta - 1) = (\eta - 1)^{\kappa}. \tag{14}$$

Hence, the probability that an element is not sampled at all is equal to the number of options without the element over the number of all options as:

$$\mathbf{P}_x^{(0)} = \frac{(\eta - 1)^{\kappa}}{\eta^{\kappa}}. \tag{15}$$

Referring to Equation (13), the probability that any element is sampled at least once is:

$$\begin{aligned} \mathbf{P}^{(1+)} &= 1 - \frac{(\eta - 1)^{\kappa}}{\eta^{\kappa}} \Longleftrightarrow 1 - \mathbf{P}^{(1+)} = \frac{(\eta - 1)^{\kappa}}{\eta^{\kappa}}, \\ \Rightarrow \kappa &= \log_{\frac{\eta-1}{\eta}} (1 - \mathbf{P}_x^{(1+)}), \\ \Rightarrow \kappa &= \frac{\ln (1 - \mathbf{P}_x^{(1+)})}{\ln (1 - \frac{1}{\eta})}. \end{aligned} \tag{16}$$

D. ACTION-SPACE UPDATE ALGORITHM

Action-space of first task $\mathcal{A}(1)$ is a continuous pool with infinite values that needs to be converted into discrete values. It is initialized by defining initial minimum action value $\mathcal{T}_{min}$, initial maximum action value $\mathcal{T}_{max}$, and the number of options Φ needed initially. The initial action-space list $\mathcal{A}_0(1)$ is a list made of Φ actions defined between $\mathcal{T}_{min}$ and $\mathcal{T}_{max}$, as shown by Equation (18):

$$\mathcal{A}_0(1) = \Big[range\Big(\mathcal{T}_{min}, (\mathcal{T}_{max} + 1), round(\mathbf{d})\Big)\Big], \tag{17}$$

$$\mathcal{A}_0(1) = \Big[\mathcal{T}_{min}, \mathcal{T}_{min} + \mathbf{d}, \mathcal{T}_{min} + (2 \times \mathbf{d}), \ldots, \mathcal{T}_{min} + ((\Phi - 1) \times \mathbf{d})\Big], \tag{18}$$

with spacing $\mathbf{d}$ the difference between two consecutive action values defined by:

$$\mathbf{d} = \frac{\mathcal{T}_{max} - \mathcal{T}_{min}}{\Phi - 1}. \tag{19}$$

After the initial action-space sample is created, the sample is updated continuously whenever the *action-space update condition* (explained in section III-C) is met. Since $\mathcal{K}(1)$ has infinite action-space values, by continuously updating the action-space sample, the model is given more choices to pick from. It may allow converging the cumulative reward to the optimal value in the long run. To update the action-space sample, as described in Algorithm 2, the element with higher upper-confidence-bound at a given time $a_t(1)^*$ is selected from $\mathcal{A}(1)$, then new $\mathcal{T}_{min}$ and $\mathcal{T}_{max}$ are calculated [*line* 3]:

$$\begin{aligned} \mathcal{T}_{min} &= a_t(1)^* - \mathbf{d}, \\ \mathcal{T}_{max} &= a_t(1)^* + \mathbf{d}. \end{aligned} \tag{20}$$

The updated values may include elements higher and lower than the initial minimum and maximum values. This means the initial minimum and maximum values do not have a critical effect on the whole process, as they can be overwritten. With these new values, the spacing $\mathbf{d}$ value is updated with Equation (19) [*line* 4]. Then a new list of actions is generated with Equation (18) [*line* 5], and cleaned by removing negative numbers and those already in the action-space. The new list of actions is then concatenated to the action-space list.

Algorithm 2 Action-Space Update Algorithm

1: $Results \leftarrow \mathcal{A}_t(1)$
2: $Data \leftarrow a_t(1)^*, \mathbf{d}, \Phi$
3: $\mathcal{T}'_{min} = a_t(1)^* - \mathbf{d}$, $\mathcal{T}'_{max} = a_t(1)^* + \mathbf{d}$
4: $\mathbf{d} = \frac{\mathcal{T}'_{max} - \mathcal{T}'_{min}}{\Phi - 1}$ // update spacing value $\mathbf{d}$
5: $new\_list = \Big[range\Big(\mathcal{T}'_{min}, (\mathcal{T}'_{max} + 1), round(\mathbf{d})\Big)\Big]$ // create a new list with Equation (18)
6: $\mathcal{A}_t(1) = \mathcal{A}_t(1) + clean(new\_list)$

E. ACTION-SPACE INCREASE ANALYSIS

With an increasing action-space, DMCSL is at risk of diverging rather than converging. An analysis of how the action-space is increasing may shed light on the model's convergence in the long run. The Φ defines the initial action-space size and specify the number of values to be added on the action-space sample every time it is updated based on the probability $\mathbf{P}^{(1+)}$. The action-space sample updates happen at different iterations of $\{t_0, t_1, t_2, t_3, t_4, \ldots, t_x\}$. If we know when the updates happen, we can infer the action-space's estimation size since at every update, there are Φ actions added. The update t_1 (the first time the sample is updated) will happen at iteration of $t_1 = t_0 + \kappa$, which is $t_1 = \kappa$ as $t_0 = 0$. By considering the effect of exploration of $\mathcal{K}(1)$ and exploitation of $\mathcal{K}(2)$, t_1 becomes,

$$t_1 = t_0 + \Big(\kappa \cdot (\frac{1}{(1 - \overline{\varepsilon}(1)) \cdot (\overline{\varepsilon}(2))})\Big); \tag{21}$$

which, if we replace κ by Equation 11 and take into account that the first action-space size is equal to Φ (and continuously updated in its multiples), t_1 becomes:

$$t_1 = t_0 + \Big(\frac{\ln (1 - \mathbf{P}_x^{(1+)})}{\ln (1 - \frac{1}{\Phi})} \cdot (\frac{1}{\xi})\Big), \tag{22}$$

with $\xi = (1 - \overline{\varepsilon}(1)) \cdot (\overline{\varepsilon}(2))$; hence, updates happen at iterations as follows:

$$t_1 = t_0 + \Big(\frac{\ln (1 - \mathbf{P}_x^{(1+)})}{\ln (1 - \frac{1}{\Phi})} \cdot (\frac{1}{\xi})\Big), \tag{23}$$

$$t_2 = t_1 + \Big(\frac{\ln(1-\mathbf{P}_x^{(1+)})}{\ln(1-\frac{1}{\mathbf{2}\cdot\Phi})}\cdot(\frac{1}{\xi})\Big), \tag{24}$$

$$t_3 = t_2 + \Big(\frac{\ln(1-\mathbf{P}_x^{(1+)})}{\ln(1-\frac{1}{\mathbf{3}\cdot\Phi})}\cdot(\frac{1}{\xi})\Big), \tag{25}$$

$$\vdots$$

$$t_x = t_{(x-1)} + \Big(\frac{\ln(1-\mathbf{P}_x^{(1+)})}{\ln(1-\frac{1}{x\cdot\Phi})}\Big)\cdot(\frac{1}{\xi})\Big), \tag{26}$$

$$\Rightarrow t_x = \ln(1-\mathbf{P}_x^{(1+)})\cdot\sum_{i=1}^{x}\frac{1}{\ln(1-\frac{1}{i\cdot\Phi})\cdot(\xi)}. \tag{27}$$

To consider the fact that during the increase, duplicates occur and get discarded, let us introduce a deduplication **D** value in the expression 27 with the logic that:

- Initially, **D** is set to 0.01, equivalent to almost no duplication in new values.
- **D** is continuously increased by a deduplication rate $\hat{\mathbf{D}}$ as $\mathbf{D} = \frac{\mathbf{D}}{\hat{\mathbf{D}}}$, for every iteration.
- The maximum **D** value is set to 0.99, equivalent to huge duplication in new values.

Equation 27 then becomes:

$$t_x = \ln(1-\mathbf{P}_x^{(1+)})\cdot\sum_{i=1}^{x}\frac{1}{\mathbf{D}\cdot\ln(1-\frac{1}{i\cdot\Phi})\cdot(\xi)}, \tag{28}$$

which, with different deduplication rate results in an algorithmic increase as shown in Fig. 4 and Fig. 5. Thus, using the proposed action-space update algorithm provides a logarithmic increase of the action-space. A logarithmic increase of the action-space means that the growth is more manageable than a linear or exponential increase. However, not all logarithmic increases are smooth in the same way. From the analysis of action-space increase, as shown in Figures 4 and 5, in order to maintain a smooth (not too quick and not too large) action-space increase, the threshold probability $\mathbf{P}_x^{(1+)}$ should be as higher as possible, but less than 1. This means a number like 0.99, for example.

F. COMPLEXITY ANALYSIS

DMCSL algorithm complexity is relative to the number of episodes and time-steps per each episode **T**. It also depends on the complexity of both DQN and MAB. DQN complexity in each time-step depends on the complexity of obtaining Q-approximation as well as training the weights of the DQN, which is dependent on the architecture of DQN along with different hyper-parameters used on it. As demonstrated in [35], the computational complexity per each DQN layer in fully-connected layers (same used in DMCSL) can be reduced to $\mathcal{O}(\mathbf{n}\cdot\log(\mathbf{n}))$, with **n** being the number of units for each layer. Therefore, the complexity of obtaining Q-approximation from **y** layers becomes:

$$\mathcal{O}(\mathbf{y}\cdot\mathbf{n}\cdot\log(\mathbf{n})). \tag{29}$$

MAB complexity in each time-step is the complexity of identifying the best arm from the collection of Φ arms. In DMCSL, MAB uses $\mathcal{UCB}$ to identify the best arm for each time-step. The complexity of identifying the best arm by $\mathcal{UCB}$ depends on the number of arms and the current time-step. That is $\mathcal{O}(\Phi\cdot\frac{T}{2})$ to calculate $\mathcal{UCB}$ for each arm, and $\mathcal{O}(\Phi)$ for identifying the arm with the maximum $\mathcal{UCB}$. The resulting run-time is:

$$\mathcal{O}(\Phi\cdot\frac{\mathbf{T}}{2}+\Phi)\Rightarrow\mathcal{O}\Big(\Phi\cdot(\frac{\mathbf{T}}{2}+1)\Big). \tag{30}$$

By taking into account that the number of actions increasingly changes logarithmically, expression (30) becomes:

$$\mathcal{O}\Big(\Phi\cdot\ln(\frac{\mathbf{T}}{2})\cdot(\frac{\mathbf{T}}{2}+1)\Big). \tag{31}$$

DMCSL complexity per time-step is then obtained from expressions (29) and (31) as:

$$\mathcal{O}\Big(\mathbf{y}\cdot\mathbf{n}\cdot\log(\mathbf{n})+\Phi\cdot\ln(\frac{\mathbf{T}}{2})\cdot(\frac{\mathbf{T}}{2}+1)\Big). \tag{32}$$

For a single episode of **T** time-steps, it is:

$$\mathcal{O}\Big(\mathbf{T}\cdot\Big(\mathbf{y}\cdot\mathbf{n}\cdot\log(\mathbf{n})+\Phi\cdot\ln(\frac{\mathbf{T}}{2})\cdot(\frac{\mathbf{T}}{2}+1)\Big)\Big). \tag{33}$$

Meanwhile, DMCSL slows down over time comparably to the DQN complexity; however, with the effect of action-space increasing, it becomes a little more slowly than DQN.

IV. SIMULATION

A. ATTACK MODEL

Jammers use different strategies to interfere with communicating nodes by either being intelligent or elementary [1] in order to interfere in either frequency domain such as:

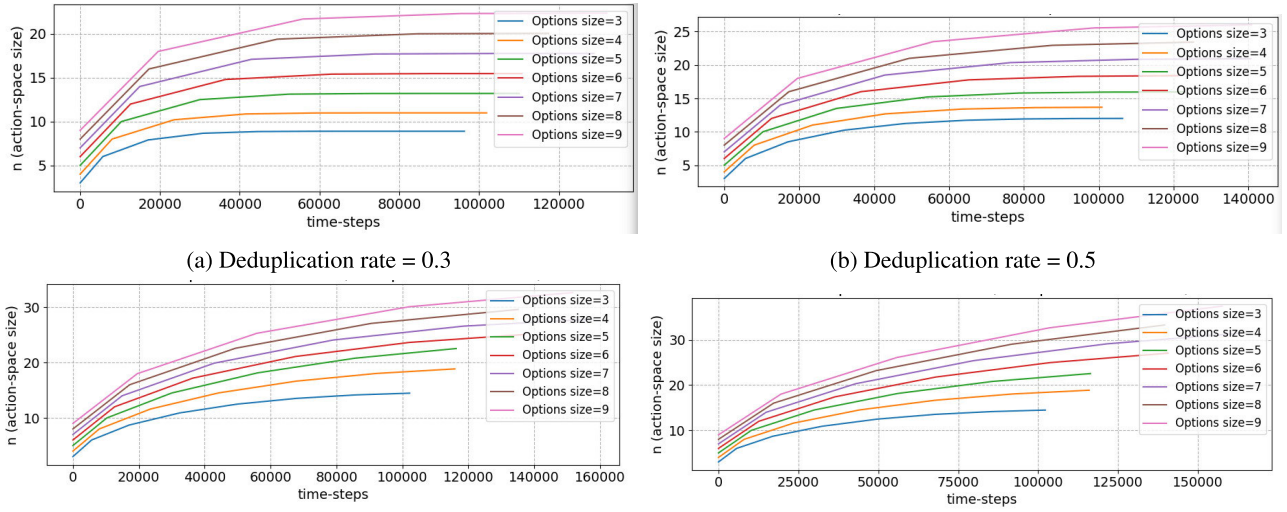
- Channel-hopping/sweep jammer: proactively hops among several channels at the same time.
- Spot jammer: uses its entire transmitting power on a single frequency.
- Follow-on jammer: hops among all channels very frequently, hence jamming each for a short time-slot.

Or in time domain such as:

- Constant jammer: continuously jams the network.
- Random jammer: alternates between sleeping and jamming.
- Reactive jammer: stays quiet when the communication channel is idle and jams only when transmission activities are sensed on the channel.

We considered both domains during the simulation by implementing a sweep jammer that operates as a constant and random jammer in three sweep jammer variants as follows:

- Static jammer: the jammer has a static (non-changing) jamming time-slot duration of 100 ms.
- Dynamic pattern jammer: the jammer changes the jamming time-slot in a predefined pattern of 50 ms, 100 ms, 150 ms, 500 ms, 1000 ms.
- Dynamic random jammer: the jammer changes the jamming time-slot in a random sequence.



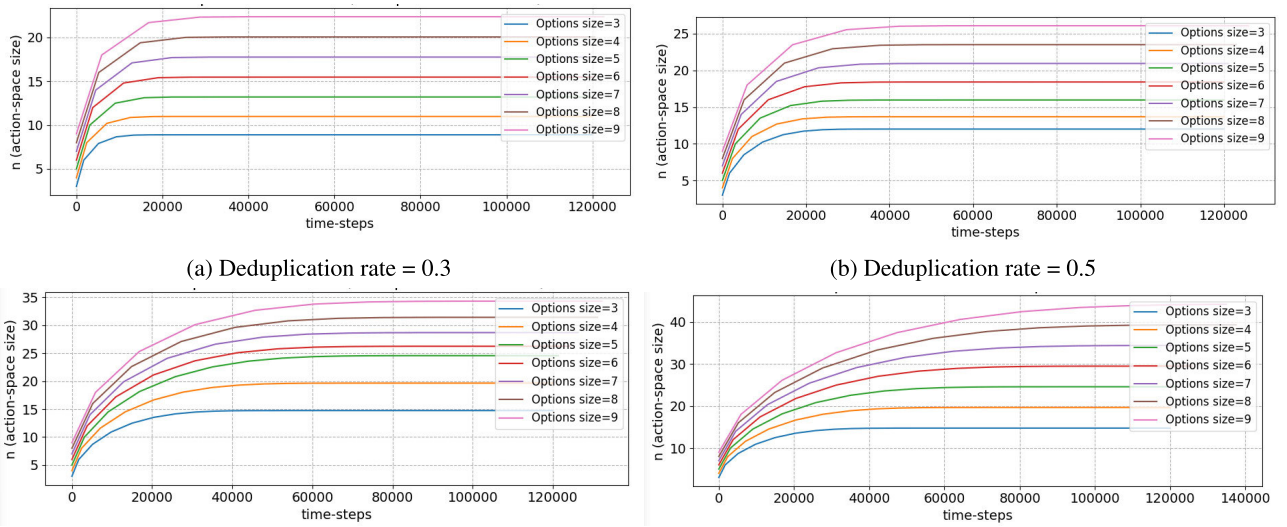
(a) Deduplication rate = 0.3

(b) Deduplication rate = 0.5

(c) Deduplication rate = 0.7

(d) Deduplication rate = 0.9

FIGURE 4. Illustration of a logarithmic increase of the action space over time with a threshold probability $\mathbf{P}_x^{(1+)}$ (probability that any action from the action-space sample has been used at least once) preset to 0.9 on different deduplication rates.



(a) Deduplication rate = 0.3

(b) Deduplication rate = 0.5

(c) Deduplication rate = 0.7

(d) Deduplication rate = 0.9

FIGURE 5. Illustration of a logarithmic increase of the action space over time with a threshold probability $\mathbf{P}_x^{(1+)}$ (probability that any action from the action-space sample has been used at least once) preset to 0.5 on different deduplication rates.

B. SIMULATION SETUP

The simulation aims to answer two main questions when the transmitter and jammer nodes abide by Assumption 1. First, find if solving jamming attacks using a multi-task algorithm allows the agent to adjust on unknown jamming time-slot and yield better performance than a single-task algorithm. Second, find if multi-task learning grants the optimal reward against a jammer whose jamming time-slot is dynamic or static. We set up network environment with *ns3-gym* by [36] and uses *TensorFlow* for DMCSL, and some basic simulation parameters are shown in Table 2. The DQN model is simulated with hyperparameters initialized as follows: $\overline{\varepsilon}(1) = 0.999$, $\overline{\varepsilon}(2) = 0.9999$, $\varepsilon_{min}(i) = 0.001$, $\varepsilon_0(i) = 1$, $\gamma = 0.95$, $lr = 0.1$. The hyperparameters were calibrated by a repeated grid search approach. For simplicity, we set reward $\mathbf{r}_t$ as 1 to a successful transmission and -1 on a failed transmission for every iteration.

$$\mathbf{r}_t = \begin{cases} +1, & \text{if successful transmission} \\ 0, & \text{if no transmission} \\ -1, & \text{if failed transmission} \end{cases} \tag{34}$$

Moreover, since the reward is given based on the packet transmission status, which is affected by both jamming and interference, the cumulative reward is comparable to the packet delivery ratio.

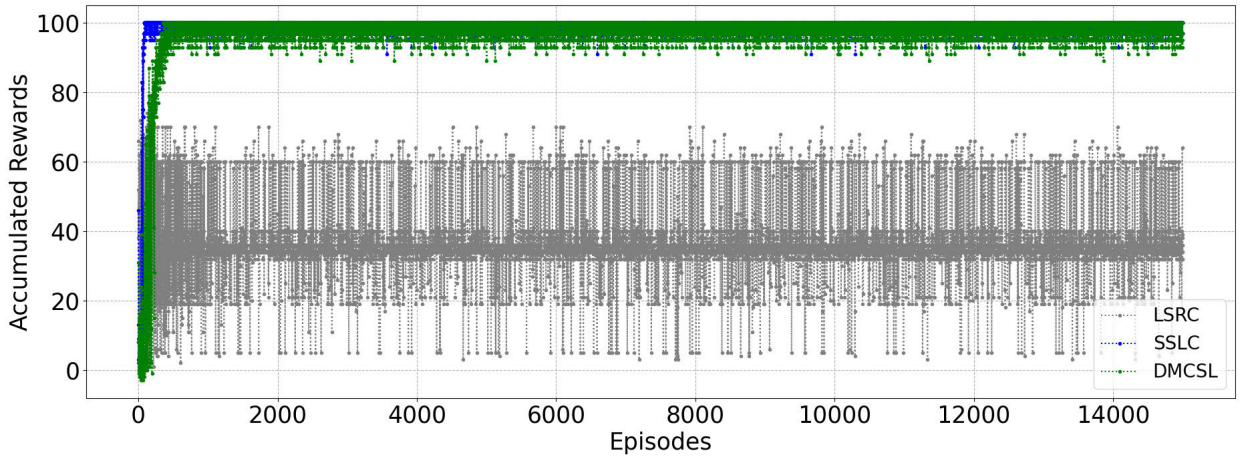


FIGURE 6. Learning anti-jammers with dynamic and static sensing time against jammer with jamming time-slots of 100 ms.

TABLE 2. Simulation parameters.

Parameter	Value
Simulation time	15000 episodes (100 iterations each)
Initial min sensing time $\mathcal{T}_{min}$	50 ms
Initial max sensing time $\mathcal{T}_{max}$	1000 ms
Options $\mathbf{\Phi}$	5
Transmission range	250 ms
Nodes' mobility	Random waypoint
Routing/ mac protocol	AODV/ IEEE 802.11b
Fading model	Nakagami propagation loss model
Threshold probability $\mathbf{P}^{(1+)}$	to 0.9

Network devices' communication decisions are formulated as a Markov decision process in a discrete-time stochastic control process where the transmitter node improves communication performance over time based on prior transmissions. The transmitter uses RL techniques to achieve optimal communication via multiple trial-and-errors. We assume a slotted-time multi-channel ad hoc network system where each agent can access at most one channel at a given slot-time. The number of available orthogonal and independent channels is limited to only four. For a baseline scenario, we set up a network of four communicating nodes (only one is relying on DMCSL to decide the transmission channel) and one sweep jammer node. The action-state spaces are:

- Sensing time: The agent decides how long a sensing time is at a given time.
- Transmission channel: Upon the spectrum sensing results, the agent observes the channel status and decides which channel to use for data transmission.

C. SIMULATION RESULTS

In the simulation, we analyzed the DMCSL performance by comparing it with two more channel hopping strategy systems:

- Static sensing learning channel (SSLC): when a single-task is used with an arbitrary sensing time and the model only learns the transmission channel.
- Learning sensing random channel (LSRC): when single-task learning is used to learn sensing time. However, channel transmission is selected randomly among identified idle channels.

The SSLC is adopted by several related works. Its main issue is the lack of an adaptive mechanism to determine the appropriate spectrum sensing duration. By comparing DMCSL to SSLC, we can observe whether DMCSL is better in setting the proper sensing time. Hence, improving channel mitigation for the spectral retreat defense strategy. A further comparison will be needed to analyze the improvement provided over competition and spatial retreat defense strategies.

In Fig. 6 and Fig. 7, static jamming time-slots of 100 ms and 897 ms are used, respectively. For both cases, the node using DMCSL converges to the maximum cumulative reward despite not having information about the jamming time-slot duration. For the SSLC, the node uses 100 ms as the sensing time. The convergence happens faster than on DMCSL for the 100 ms jamming time-slot; however, for the 897 ms jamming time-slot, the node does not manage to converge to the optimal value. The cumulative reward for the LSRC does not converge, which shows that learning sensing time alone is not a solution for consideration. Both DMCSL and SSLC performances fluctuate significantly in the initial episodes as the model is still quite exploring with the decaying epsilon approach. Moreover, despite that DMCSL reaches the optimal value in both cases, it later keeps showing a slight performance fluctuation, rather than a continuously optimized performance. This is because action-space is still updated, although at a slower pace. The action-space update would necessitate a stopping condition in order to ensure continuous optimal performance. In Fig. 8, a dynamic jamming time-slot is used, where the jammer changes the jamming time-slot, but in a predefined order of [50 ms, 100 ms, 150 ms, 500 ms, 1000 ms], namely, the jamming time-slot changes every time but the sequence of changes remains the same for the runtime. Simulation results show that DMCSL can

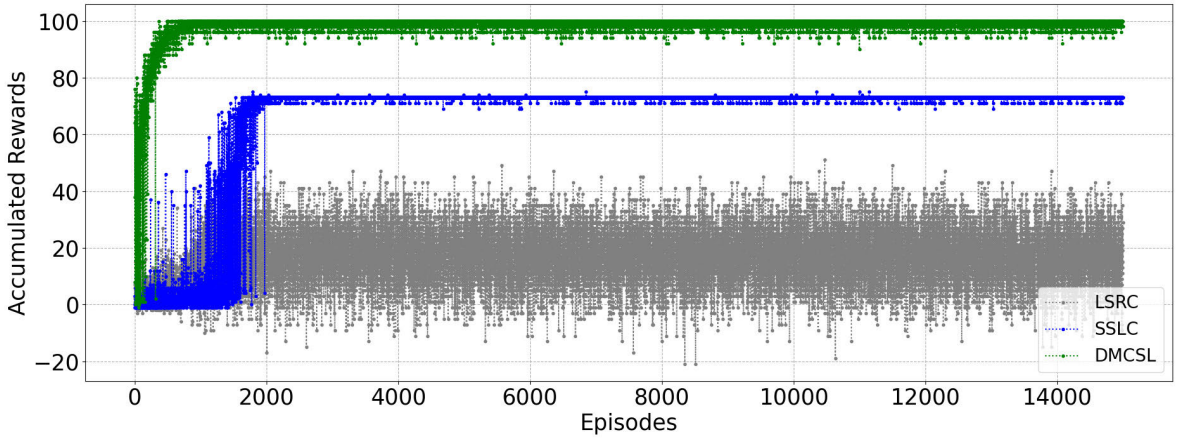


FIGURE 7. Learning anti-jammers with dynamic and static sensing time against jammer with jamming time-slots of 897 ms.

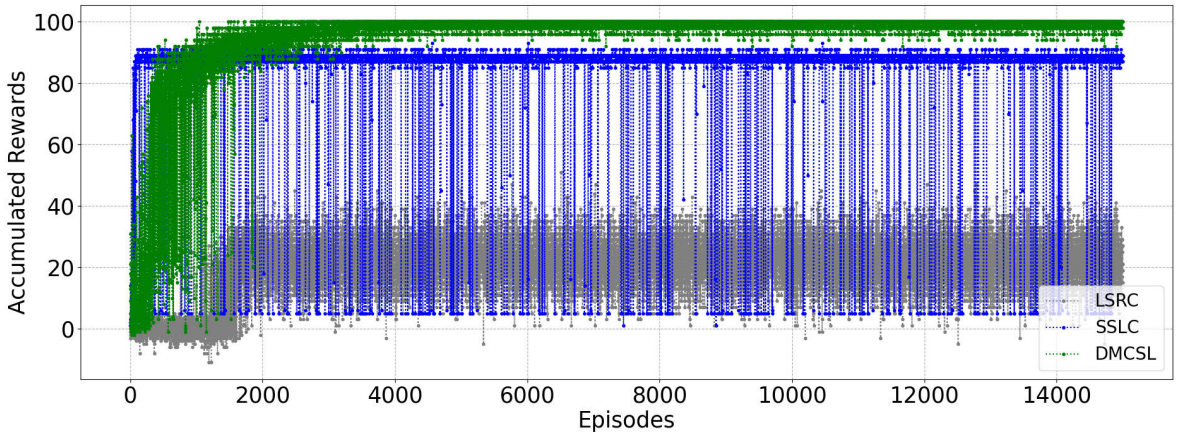


FIGURE 8. Learning anti-jammers with dynamic and static sensing time-slot against jammer with ordered dynamic jamming time [50 ms, 100 ms, 150 ms, 500 ms, 1000 ms].

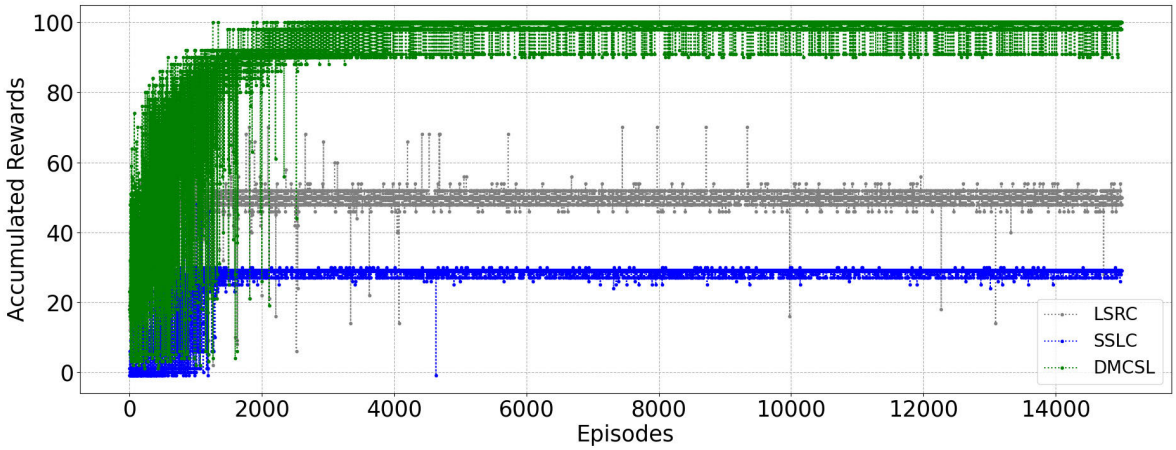


FIGURE 9. Learning anti-jammers with dynamic and static sensing time-slot against jammer with jamming time randomly picked with values between 100 ms and 150 ms.

adapt to the changes and optimize the cumulative reward in the long run. However, the convergence takes a longer time than when the jammer uses a single jamming time-slot length. A node using either SSLC or LSRC does not converge as the cumulative rewards show too many fluctuations.

In Fig. 9 and Fig. 10, an utterly random jamming time-slot length is used, but with different ranges. In Fig. 9, the jamming time-slot length is randomly picked with values between 100 ms and 150 ms, while in Fig. 10, the jamming time-slot length is picked with values between 100 ms and

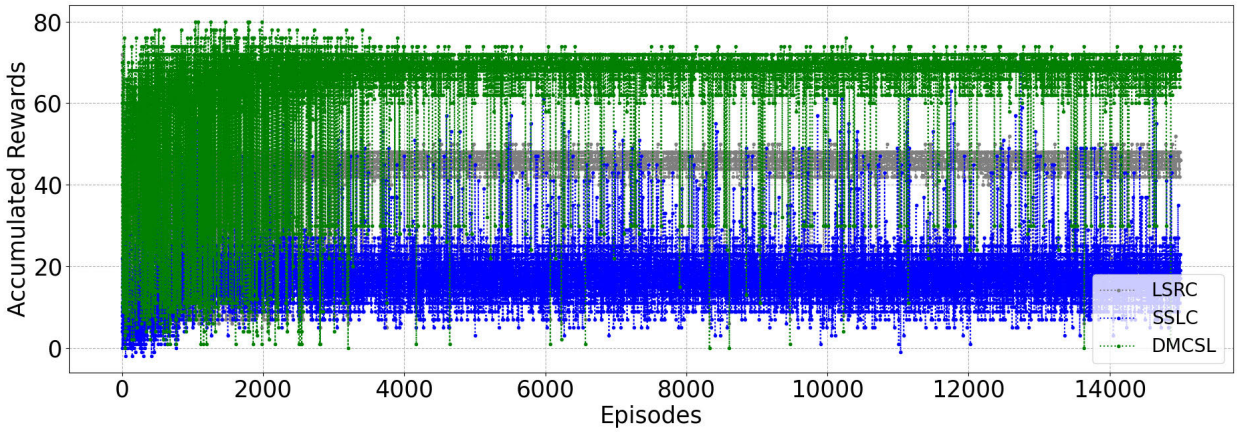


FIGURE 10. Learning anti-jammers with dynamic and static sensing time-slot against jammer with jamming time randomly picked with values between 100 ms and 1000 ms.

1000 ms. For Fig. 9, DMCSL converges to the optimal value but takes longer. Fig. 10 shows that the jamming time-slot lengths randomness affects the algorithms' performance as none of the algorithms manage to converge to the optimal value; however, the DMCSL outperforms the other anti-jamming schemes. From both Fig. 9 and Fig. 10, we can infer that randomness of the jamming time-slot generate higher impact when it is less predictable, as with a more extensive range, because that is the only moment where the DMCSL algorithm did not manage to converge within the 15000 episodes.

V. CONCLUSION

In this work, we investigate the performance of using a single-task learning algorithm to mitigate jamming attacks by using frequency hopping as the mitigation technique when the transmitter cannot know the internal working structure of the jammer, mainly the jamming time-slot length. The analysis showed that designing anti-jamming learning systems as single-task learning for the transmission channel selection does not always guarantee optimal performance in the long run if the sensing time used is not optimal. The proposed multi-task learning anti-jamming system provides better performance than single-task learning. It mainly guarantees the optimal cumulative reward against static jammer by developing a policy to handle the jammer with a dynamic jamming time-slot. When using DMCSL against static jammer time-slot, the performance is almost identical to learning anti-jamming with static sensing-time, if the sensing-time is identical to jammer time-slot. however, when both times are different, agents using DMCSL increase their cumulative rewards by 10%. The same agent using DMCSL against a random dynamic jamming time-slot achieves about three times better cumulative reward, while against a periodic dynamic jamming time-slot, it improves the cumulative rewards by 10% as well. The proposed DMCSL algorithm relies on MAB and DQN and has the potential of being used on an actual network such as drones, sensors, or vehicular networks. It can be extended with more design for future works, such as using double DQN, using another reward scheme, or testing against other types of jammers, such as intelligent or reactive jammer.

REFERENCES

[1] X. Wei, Q. Wang, T. Wang, and J. Fan, "Jammer localization in multi-hop wireless network: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 765–799, 2nd Quart., 2017.
[2] F. Slimeni, B. Scheers, Z. Chtourou, V. Le Nir, and R. Attia, "Cognitive radio jamming mitigation using Markov decision process and reinforcement learning," *Procedia Comput. Sci.*, vol. 73, pp. 199–208, Jan. 2015.
[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[4] P. Jain, B. Kulis, I. S. Dhillon, and K. Grauman, "Online metric learning and fast similarity search," in *Advances in Neural Information Processing Systems*, vol. 21, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Red Hook, NY, USA: Curran Associates, 2009.
[5] R. Agarwal, D. Schuurmans, and M. Norouzi, "An optimistic perspective on offline reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 104–114.
[6] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, and M. Riedmiller, "Keep doing what worked: Behavioral modelling priors for offline reinforcement learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2020, pp. 1–21.
[7] S. Ruder, "An overview of multi-task learning in deep neural networks," 2017, *arXiv:1706.05098*. [Online]. Available: http://arxiv.org/abs/1706.05098
[8] A. Argyriou, T. Evgeniou, and M. Pontil, "Multi-task feature learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1–8.
[9] J. Zhang, Z. Ghahramani, and Y. Yang, "Learning multiple related tasks using latent independent component analysis," in *Advances in Neural Information Processing Systems*, vol. 18, Y. Weiss, B. Schölkopf, and J. C. Platt, Eds. Cambridge, MA, USA: MIT Press, 2006.
[10] W. Xu, K. Ma, W. Trappe, and Y. Zhang, "Jamming sensor networks: Attack and defense strategies," *IEEE Netw.*, vol. 20, no. 3, pp. 41–47, May 2006.
[11] N. I. Mowla, N. H. Tran, I. Doh, and K. Chae, "AFRL: Adaptive federated reinforcement learning for intelligent jamming defense in FANET," *J. Commun. Netw.*, vol. 22, no. 3, pp. 244–258, Jun. 2020.
[12] C. Han and Y. Niu, "Cross-layer anti-jamming scheme: A hierarchical learning approach," *IEEE Access*, vol. 6, pp. 34874–34883, 2018.
[13] J. Peng, Z. Zhang, Q. Wu, and B. Zhang, "Anti-jamming communications in UAV swarms: A reinforcement learning approach," *IEEE Access*, vol. 7, pp. 180532–180543, 2019.
[14] L. Xiao, D. Jiang, D. Xu, H. Zhu, Y. Zhang, and H. V. Poor, "Two-dimensional antijamming mobile communication based on reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 10, pp. 9499–9512, Oct. 2018.

[15] N. Abuzainab, T. Erpek, K. Davaslioglu, Y. E. Sagduyu, Y. Shi, S. J. Mackey, M. Patel, F. Panettieri, M. A. Qureshi, V. Isler, and A. Yener, "QoS and jamming-aware wireless networking using deep reinforcement learning," in *Proc. IEEE Mil. Commun. Conf. (MILCOM)*, Nov. 2019, pp. 610–615.
[16] P. Tedeschi, G. Oligeri, and R. Di Pietro, "Leveraging jamming to help drones complete their mission," *IEEE Access*, vol. 8, pp. 5049–5064, 2020.
[17] Y. E. Sagduyu, Y. Shi, and T. Erpek, "Adversarial deep learning for over-the-air spectrum poisoning attacks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 2, pp. 306–319, Feb. 2021.
[18] X. Liu, Y. Xu, L. Jia, Q. Wu, and A. Anpalagan, "Anti-jamming communications using spectrum waterfall: A deep reinforcement learning approach," *IEEE Commun. Lett.*, vol. 22, no. 5, pp. 998–1001, May 2018.
[19] L. Xiao, X. Lu, D. Xu, Y. Tang, L. Wang, and W. Zhuang, "UAV relay in VANETs against smart jamming with reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4087–4097, May 2018.
[20] G. Chen, Y. Zhan, Y. Chen, L. Xiao, Y. Wang, and N. An, "Reinforcement learning based power control for in-body sensors in WBANs against jamming," *IEEE Access*, vol. 6, pp. 37403–37412, 2018.
[21] N. I. Mowla, N. H. Tran, I. Doh, and K. Chae, "Federated learning-based cognitive detection of jamming attack in flying ad-hoc network," *IEEE Access*, vol. 8, pp. 4338–4350, 2020.
[22] S. Liu, Y. Xu, X. Chen, X. Wang, M. Wang, W. Li, Y. Li, and Y. Xu, "Pattern-aware intelligent anti-jamming communication: A sequential deep reinforcement learning approach," *IEEE Access*, vol. 7, pp. 169204–169216, 2019.
[23] M. Sun, X. Wang, C. Zhao, B. Li, Y.-C. Liang, G. Goussetis, and S. Salous, "Adaptive sensing schedule for dynamic spectrum sharing in time-varying channel," *IEEE Trans. Veh. Technol.*, vol. 67, no. 6, pp. 5520–5524, Jun. 2018.
[24] Q. Ye, W. Zhuang, S. Zhang, A.-L. Jin, X. Shen, and X. Li, "Dynamic radio resource slicing for a two-tier heterogeneous wireless network," *IEEE Trans. Veh. Technol.*, vol. 67, no. 10, pp. 9896–9910, Oct. 2018.
[25] O. Naparstek and K. Cohen, "Deep multi-user reinforcement learning for distributed dynamic spectrum access," *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 310–323, Jan. 2019.
[26] J. Hu, G. Li, D. Bian, J. Tang, and S. Shi, "Sensing-based dynamic spectrum sharing in integrated wireless sensor and cognitive satellite terrestrial networks," *Sensors*, vol. 19, no. 23, p. 5290, Dec. 2019.
[27] D. Zhao, H. Qin, B. Song, B. Han, X. Du, and M. Guizani, "A graph convolutional network-based deep reinforcement learning approach for resource allocation in a cognitive radio network," *Sensors*, vol. 20, no. 18, p. 5216, Sep. 2020.
[28] X. Liu, M. Jia, X. Zhang, and W. Lu, "A novel multichannel Internet of Things based on dynamic spectrum sharing in 5G communication," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 5962–5970, Aug. 2019.
[29] P. Wang, B. Di, H. Zhang, K. Bian, and L. Song, "Cellular V2X communications in unlicensed spectrum: Harmonious coexistence with VANET in 5G systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 8, pp. 5212–5224, Aug. 2018.
[30] S. Zhang, H. Tian, X. Chen, Z. Du, L. Huang, Y. Gong, and Y. Xu, "Design and implementation of reinforcement learning-based intelligent jamming system," *IET Commun.*, vol. 14, no. 18, pp. 3231–3238, Nov. 2020.
[31] Z. Yang, P. Cheng, and J. Chen, "Learning-based jamming attack against low-duty-cycle networks," *IEEE Trans. Depend. Sec. Comput.*, vol. 14, no. 6, pp. 650–663, Nov. 2017.
[32] T. Erpek, Y. E. Sagduyu, and Y. Shi, "Deep learning for launching and mitigating wireless jamming attacks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 5, no. 1, pp. 2–14, Mar. 2019.
[33] S. Amuru, C. Tekin, M. van der Schaar, and R. M. Buehrer, "Jamming bandits—A novel learning method for optimal jamming," *IEEE Trans. Wireless Commun.*, vol. 15, no. 4, pp. 2792–2808, Apr. 2016.
[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, and G. Ostrovski, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.
[35] Y. Wang, C. Ding, Z. Li, G. Yuan, S. Liao, X. Ma, B. Yuan, X. Qian, J. Tang, Q. Qiu, and X. Lin, "Towards ultra-high performance and energy efficiency of deep learning systems: An algorithm-hardware co-optimization framework," 2018, *arXiv:1802.06402*. [Online]. Available: http://arxiv.org/abs/1802.06402
[36] P. Gawłowicz and A. Zubow, "ns-3 meets OpenAI gym: The playground for machine learning in networking research," in *Proc. ACM Int. Conf. Modeling, Anal. Simulation Wireless Mobile Syst. (MSWiM)*, Nov. 2019, pp. 113–120.

ROBERT BASOMINGERA (Student Member, IEEE) received the B.S. degree from the College of Science and Technology, University of Rwanda, in 2014, the M.S. degree from the College of Engineering, Carnegie Mellon University, in 2017, and the Ph.D. degree in computer science and engineering from Ajou University, Suwon, South Korea, in 2021. His research interests include network security, distributed systems, and machine learning applications in information security.

YOUNG-JUNE CHOI (Senior Member, IEEE) received the B.S., M.S., and Ph.D. degrees from Seoul National University, Seoul, South Korea, in 2000, 2002, and 2006, respectively. From 2006 to 2007, he was a Postdoctoral Researcher with the University of Michigan, Ann Arbor, MI, USA. From 2007 to 2009, he was with NEC Laboratories America, Inc., Princeton, NJ, USA, as a Research Staff Member. In September 2009, he joined Ajou University, South Korea, as a Faculty Member, and founded the Intelligence Platform, Network, and Security (Intelligence PLANETs) Laboratory. He is currently a Professor with Ajou University. His research interests include beyond mobile wireless networks, radio resource management, and cognitive radio networks. He was a recipient of the Gold Prize from the Samsung Humantech Thesis Contest, in 2006, the Haedong Young Researcher Award, in 2015, and the Best Paper Award from the Journal of Communications and Networks, in 2015. He is an Executive Director of the Korean Institute of Communications and Information Sciences (KICS), the Director of the Korean Institute of Information Scientists and Engineers (KIISE), and the Founder and CEO of Coding Robot Laboratory, Inc.