IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 January 2023, accepted 1 February 2023, date of publication 8 February 2023, date of current version 14 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3243549

RESEARCH ARTICLE

A Comparative Study of Reinforcement Learning Algorithms for Distribution Network Reconfiguration With Deep Q-Learning-Based Action Sampling

NASTARAN GHOLIZADEH[1], NAZLI KAZEMI[1], (Graduate Student Member, IEEE), AND PETR MUSILEK[1,2], (Senior Member, IEEE)

[1]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB T6G 2R3, Canada
[2]Department of Applied Cybernetics, University of Hradec Králové, 50003 Hradec Králové, Czech Republic

Corresponding author: Nastaran Gholizadeh (nastaran@ualberta.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada under Grant ALLRP 549804-19; and in part by the Alberta Electric System Operator, AltaLink, ATCO Electric, ENMAX, EPCOR Inc., and FortisAlberta.

ABSTRACT Distribution network reconfiguration (DNR) is one of the most important methods to cope with the increasing electricity demand due to the massive integration of electric vehicles. Most existing DNR methods rely on accurate network parameters and lack scalability and optimality. This study uses model-free reinforcement learning algorithms for training agents to take the best DNR actions in a given distribution system. Five reinforcement algorithms are applied to the DNR problem in 33- and 136-node test systems and their performances are compared: deep Q-learning, dueling deep Q-learning, deep Q-learning with prioritized experience replay, soft actor-critic, and proximal policy optimization. In addition, a new deep Q-learning-based action sampling method is developed to reduce the size of the action space and optimize the loss reduction in the system. Finally, the developed algorithms are compared against the existing methods in literature.

INDEX TERMS Distribution network reconfiguration, reinforcement learning, deep Q-learning, data-driven control, soft actor-critic, proximal policy optimization.

I. INTRODUCTION

Large-scale penetration of electric vehicles and integration of renewable energy resources have increased the complexity of power distribution networks [1]. As a result, more advanced control and optimization techniques are required to keep the system within operational constraints, reduce losses, and supply demands [2]. Distribution network reconfiguration (DNR) is the process of connecting and disconnecting different distribution lines in a way that the system loss is minimized and the voltage level is maintained. The non-linearity of AC power flow and the network radiality constraint make DNR optimization an NP-hard, mixed-integer, nonlinear problem [3]. Therefore, heuristic optimization algorithms [4] and mathematical simplifications [5] are typically used for DNR optimization in the literature.

Majority of DNR studies are based on heuristic optimization algorithms. To this end, a new social beetle swarm optimization algorithm considering two social behaviors is developed in [6] to solve the multiobjective DNR problem which minimizes network loss, load balance index, and maximum voltage deviation. The same method is coupled with grey target decision-making in [7] to improve the process of selecting the best beetle and solve the problem of conflicting objectives. Chaos disturbed beetle antennae search (CDBAS) [8] and Levy flight and chaos disturbed beetle antennae search (LDBAS) [9] algorithms are combined with grey target decision-making technique to solve the DNR problem while minimizing the load balancing index, active power loss, and the maximum node voltage deviation.

The associate editor coordinating the review of this manuscript and approving it for publication was Emilio Barocio.

A modified multi-objective Bayesian learning-based evolutionary algorithm is proposed in [10] to balance the voltage stability and the absorption rate of wind energy which is then combined by a technique for order preference by similarity to an ideal solution (TOPSIS) to solve DNR. In [11], a θ-modified bat algorithm is developed to solve the DNR problem while the uncertainties associated with multiobjective DNR are handled via cloud theory constructed based on fuzzy theory combined with the concept of probability. However, to solve the DNR problem, all of these studies utilize heuristic or meta-heuristic algorithms which are time-consuming, have high computational cost, and do not guarantee finding the optimal solution.

In a different set of studies, mathematical models are utilized to solve the DNR optimization [12]. For example, a distributionally robust chance-constrained DNR approach for a three-phase unbalanced distribution network is proposed in [13]. It optimizes the switching cost and the expected power supply cost from an upstream grid. Later, the model is reduced to a mixed-integer, linear programming problem. A risk-averse two-stage mixed-integer conic programming model is presented in [14] for grid reconfiguration where the seasonal reconfiguration decisions of microgrids (MG) are made in the first stage, and validated under stochastic islanding scenarios in the second stage. In [15], Benders decomposition framework is combined with mixed-integer quadratic programming (MIQP) to solve DNR and reduce voltage volatility. MIQP and mixed-integer linear programming are used in [16] for operation mode adjustment minimizing the active power loss and for service restoration maximizing restored loads. The main drawback of these methods is that they simplify the powerflow equations. Hence, the obtained solutions are neither accurate nor optimal.

There have been other studies which use algorithmic or fuzzy logic approaches. Switch opening and exchange (SOE) method is developed in [17] for multi-hour stochastic DNR which is based on sequential opening of switches until no loops remain in the network and modifying the status of obtained branches. In [18], a multiagent weight-based self reconfiguration algorithm with distributed generators is presented for load sharing and reducing congestion in lines. An adaptively tunable fuzzy logic controller is proposed in [19] for network reconfiguration during power system restoration. Finally, intra-day dynamic reconfiguration [20] is solved using time period reduction and decimal coding strategy to maximize the accommodation revenue of distributed generation (DG) units and to minimize the operation cost of distribution network. These methods can improve the quality of the solutions, but as the system size grows, the number of decision variables increases and they cannot guarantee optimality. In addition, their execution time becomes exponentially longer with the increase in system size.

Recently, machine learning has also been successfully applied for DNR. By combining original neural network algorithm with chaotic local search and quasi-oppositional-based learning approaches, a new algorithm is developed in [21] to simultaneously solve DG allocation and DNR optimization. However, artificial intelligence algorithms used for optimization purpose still face the problems of poor convergence and falling into local optima [9]. A deep neural network is developed in [3] to adaptively learn the reference joint probability distributions (PD) of DG outputs and loads from historical data. Later, three-phase unbalanced DNR is solved using a modified column-and-constraint generation method under the worst-case PD scenarios. In this study, neural network is only developed for forecasting the PD of DG outputs. In [22], a deep convolutional neural network (CNN) is developed for DNR. It learns the relationship between network topology and short-term voltage stability performance from historical data. Since the model is only trained on historical network topologies, it is unable to perform new reconfigurations that result in unperceived topologies. Hence, the optimality of the reconfiguration actions is not guaranteed. In [23], a hybrid data-driven and model-based DNR framework is proposed which uses long short-term memory (LSTM) network to learn the mapping mechanism between load distribution and optimal reconfiguration strategies. The model presented in [24] was similar except that it also considered the switching cost in addition to the system loss in the objective function. The main disadvantage of these two models is that the dimension of the neural network needed for training grows with the size of the network and the computational cost increases exponentially. In addition, they do not search for optimal reconfiguration strategies for different load values as they are only trained on a few reconfiguration topologies.

Most DNR algorithms discussed so far are model-based controllers. This means that they need accurate values of distribution network parameters which is difficult to obtain due to the expansive structure of distribution networks and seasonal weather changes [25]. On the other hand, the computational burden of these algorithms increases exponentially with the size of the network [26] which makes them impractical for real-time control. Reinforcement learning (RL) is an area of machine learning which uses various algorithms to train agents for taking best actions in an environment [27]. This method allows building a model-free control approach for DNR since the agent does not need to directly interact with the distribution network and it does not need to know the transition probablities between different state and action pairs. Only very few studies have used RL for DNR process. To this end, deep Q-learning and NoisyNet deep Q-learning network (DQN) with automatic exploration is applied to DNR in [28] and [29], respectively. However, in neither of these two studies, the scalability of the proposed method is demonstrated on larger test systems. Moreover, these studies compare the performance of only several DQN methods. DQN was also applied to the DNR problem in [30]. However, DQN performance was only compared with the brute-force search and genetic algorithms, but not with any other RL algorithm. Batch-constrained soft actor-critic RL algorithm

is developed in [31]. It learns a control policy from a finite historical operational dataset without interacting with the distribution network. The main drawback of this approach is that the agent is only trained on historical operational dataset which means that it does not search for new and more optimal reconfiguration strategies.

This study develops an RL model for DNR that samples from a very large action space. Hence, it is not only based on historical operation. In addition, this article compares the performance of five popular RL algorithms for DNR including DQN, policy gradient methods, and actor-critic methods. Although there were studies that implemented DQN and soft actor-critic (SAC) for DNR, no previous study implemented dueling DQN, DQN with prioritized experience replay, and proximal policy optimization (PPO) for this purpose. In addition, no study presented a comprehensive comparison of these algorithms. The main contributions of this paper are as follows:

- Proposal of a new action search algorithm to find the feasible action space for RL algorithms.
- Development of a new DQN-based action sampling method to reduce the size of the action space and improve the optimality of the obtained DNR solutions by RL.
- Implementation of five RL algorithms for DNR (including DQN, dueling DQN, DQN with prioritized experience replay, SAC, and PPO) and comparison of their performances.

II. PROBLEM FORMULATION

This section covers the problem formulation for DNR as well as the RL algorithms.

A. PRELIMINARIES

There are two main components in a standard RL structure: environment and agent. As shown in Fig. 1, the agent selects an action a_t according to the environment state s_t and receives the reward r_{t+1} and next state s_{t+1} from the environment. The main goal of RL algorithm is to train the agent to select the actions in the environment that maximize its rewards. The environment for RL is modeled as a Markov decision process (MDP). An MDP is denoted by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma, \mathcal{T})$ and consists of a state space $\mathcal{S}$ (where, $s_t \in \mathcal{S}$), action space $\mathcal{A}$ ($a_t \in \mathcal{A}$), transition probability function $\mathcal{P}$, reward function $\mathcal{R}$ ($r_t \in \mathcal{R}$), discount factor $\gamma \in [0, 1)$, and a time horizon $\mathcal{T}$.

The agent finds the optimal control policy π for its environment by maximizing the expected discounted return $G = \sum_{t=0}^{\mathcal{T}} \gamma^t r_{t+1}$. By calculating the action-values using Bellman equation, the agent decides which action to select in a given state. Bellman equation, starting from state s_t and taking action a_t while following policy π, is determined as

$$Q_\pi(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim \pi}[r(s_t, a_t) + \gamma Q_\pi(s_{t+1}, \pi(s_{t+1}))], \tag{1}$$

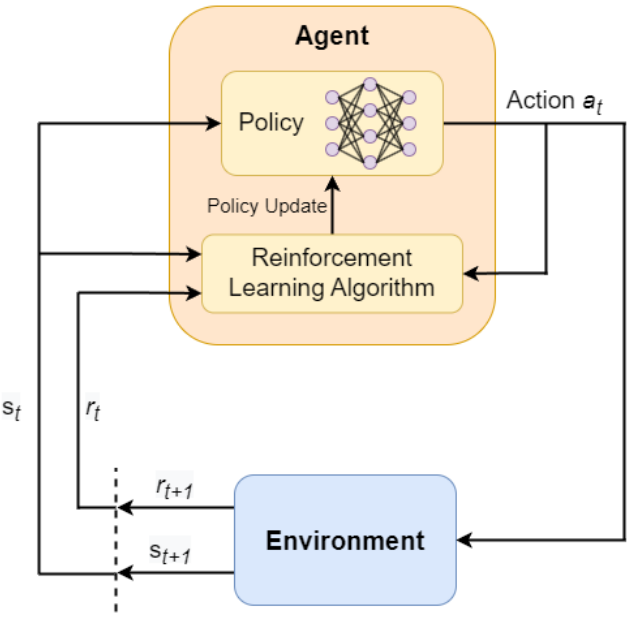

FIGURE 1. Reinforcement learning framework.

B. DNR ENVIRONMENT

To apply RL algorithms to the DNR problem, DNR should be modeled as an MDP. For this purpose, the switching actions are defined as $a_t = [1, 1, 0, \ldots, 1]$ where the open and close commands of the lines are represented by zeros and ones, respectively. In this study, it is assumed that all lines have switches which can be opened or closed. Therefore, the action dimension is equal to the number of lines in the system. The state of the system is the switch states, active, and reactive power consumption at different buses represented by $s_t = [p_t, q_t, \alpha_t]$. Here, $\alpha_t = [\alpha_{1t}, \alpha_{2t}, \ldots, \alpha_{mt}]$ is the switch states, $p_t = [p_{1t}, p_{2t}, \ldots, p_{nt}]$ is the active power, and $q_t = [q_{1t}, q_{2t}, \ldots, q_{nt}]$ is the reactive power of different buses. Parameter n stands for the number of busses, and m for the number of lines in the system. The transition probability between time-steps, $\mathcal{P}$, is assumed to be the random process of power injections.

The reward function for this RL problem is the negative of network loss which needs to be maximized so that the loss will be minimized. It is defined as

$$\mathcal{R}(s_t, a_t) = -C^l p_t^l(s_{t+1}) - C^v(s_{t+1}), \tag{2}$$

where $p_t^l(s_{t+1})$ is the total line losses of the system at state s_{t+1} and C^l is the penalty for line losses in [1/kW]. The variable $C^v(s_{t+1})$ is the penalty of violating the network voltage constraint defined as

$$C^v(s_{t+1}) = \begin{cases} \lambda & \text{if } V^{\max} < \max(V_{t,n}) \text{ or} \\ & \min(V_{t,n}) < V^{\min} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where λ is a large constant number to impose a highly negative reward for actions that cause voltage violation in the

Algorithm 1 Structure Identification

1 **Input:** Topology information of the network
2 **Output**: Feasible action space
▷ Construct the network with graph analysis tools
3 Close all lines
4 Find all loops in the network $l \in \{1, 2, \ldots, N\}$
5 Find the set of all switches S_l that open each of these loops
6 Find the total number of combinations, M, between S_l from each loop
▷ Find all combination of lines for opening loops
7 **for** $i \in \{1, 2, \ldots, M\}$ **do**
8 $flag = 0$
9 **while** $flag = 0$ **do**
10 **for** $j \in \{1, 2, \ldots, N\}$ **do**
11 **if** *loop j exists* **then**
12 Open a line from loop j
13 **if** *Network is disconnected* **then**
Close the line and open a different line
14 **if** *the line is already open* **then**
15 Open a different line from loop j
16 **if** *Network is disconnected* **then**
17 Close the line and open a different line
18 Find all the loops l in the network
19 **if** $l = 0$ **then**
20 $flag = 1$
21 Save the current structure of the network as a feasible action
22 **else**
23 Repeat 5-21 for the new loops

Algorithm 2 Deep Q-Learning

1 **Input:** Learning rate α, number of episodes N, discount factor γ,
batch size, environment
2 **Output**: Optimal policy
▷ Initialization
3 Initialize target network with parameters θ_{targ} and q-network with parameters $\theta_{targ} \leftarrow \theta$
4 Initialize replay buffer B
5 Initialize greedy policy π and ϵ-greedy policy π_ϵ
▷ Training
6 **for** *Episode* $\in \{1, 2, \ldots, N\}$ **do**
7 Reset the environment and record s_0
8 **for** $t \in \{0, 1, .., T-1\}$ **do**
9 Select action a_t using π_ϵ
10 Execute action a_t and receive s_{t+1}, r_{t+1}
11 Add transition $(s_t, a_t, r_{t+1}, s_{t+1})$ into buffer B
12 Set $s_t = s_{t+1}$
13 Choose a batch, b, of transitions from buffer B
14 Compute loss function for the selected batch

$$y_i = \begin{cases} r_i & \textbf{For } \textit{terminal state} \\ r_i + \gamma \max(Q(s_i, a_i|\theta_{targ})) & \textbf{For } \textit{non} \\ & -\textit{terminal state} \end{cases}$$

$$L(\theta) = \frac{1}{|b|} \sum_{i=1}^{|b|} [y_i - Q(s_i, a_i|\theta)]^2$$

15 Perform gradient descent for q-network and update its parameters θ
16 Every M episodes synchronize $\theta_{targ} \leftarrow \theta$

system. Load balance and radiality constraints are fulfilled while selecting the action space $\mathcal{A}$.

The complete action space for DNR is very large. For example, a system with 20 lines has a total of 2^{20} actions. However, most of these actions cause loops or disconnections in the system and hence, are infeasible to be performed in a real distribution system. To address this issue, Algorithm 1 is used to select only the feasible actions as the action space for RL. In this algorithm, first, all lines are closed to find all loops in the network (lines 3-4). Then, all of the switches that open these loops are found (line 5). Finally, all combinations between these switches are found in a way that no loops remain in the network and no disconnections occur (lines 6-23). It should be noted that action space is a pool from which the agents choose their actions. Algorithm 1 only changes the action space and it does not alter the action selection strategy of the agents in any of the RL algorithms.

C. RL ALGORITHMS

RL algorithms are divided into two main categories: on-policy and off-policy. In on-policy RL, the behavior policy which agent uses to select its actions and the target policy which it tries to learn are the same. In off-policy RL, they are different. Therefore, on-policy algorithms are more likely to converge to a sub-optimal policy. This study implements and tests the performance of DQN, dueling DQN, DQN with prioritized experience replay, and SAC as off-policy algorithms, and PPO as an on-policy algorithm.

Algorithm 2 summarizes the DQN algorithm. First, the target neural network, q-network, replay buffer, and greedy policy are initialized (lines 3-5). Then, for N number of episodes, the environment is reset, for T number of steps, an action is chosen by the policy and executed, the transitions are added to the replay buffer, and finally, the loss over a batch of transitions is computed and gradient descent is performed on the q-network to update its parameters and the policy (lines 6-15). Every M episodes, the target and q-network are synchronized (line 16). It should be noted that the q-network outputs actions values. The policy chooses the action with the highest value.

Dueling DQN is an improvement over DQN [32]. The key motivation behind this architecture is that, in some states, performing an action will not change the obtained reward so it is unnecessary to know the value of each action at every

time step. An example of such condition is the Atari game Enduro where taking an action will not change the obtained reward until collision is imminent. In general, the action value is defined as

$$Q_\pi(s,a) = \mathbb{E}_\pi[\sum_{t=0}^{T-1} \gamma^t r_t | s_t = s, a_t = a], \quad (4)$$

and the state value is defined as

$$V_\pi(s) = \mathbb{E}_\pi[\sum_{t=0}^{T-1} \gamma^t r_t | s_t = s]. \quad (5)$$

The advantage value shows how advantageous selecting an action is relative to the other actions at a given state. The advantage function is obtained by subtracting the state value from the action value

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \quad (6)$$

By separating $Q_\pi(s,a)$ into $V_\pi(s)$ and $A_\pi(s,a)$, the dueling architecture learns the state value without having to learn the effect of each action for each state. $V_\pi(s)$ and $A_\pi(s,a)$ are combined into one equation as follows

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|A|}\sum_{a'} A(s,a;\theta,\alpha)). \quad (7)$$

DQN with prioritized experience replay was first introduced by Google DeepMind in 2015 [33]. The basic idea behind this algorithm is to prioritize the transitions in the replay buffer that are rare but more helpful for the learning process. The metric to measure the importance of each transition is the magnitude of a transition's time difference (TD) error, δ. However, an offset value should be added to prevent the importance to become zero, i.e.

$$p_i = |\delta_i| + \epsilon. \quad (8)$$

Using the priority, the probability of selecting each transition is defined as

$$P_i = \frac{(p_i)^a}{\sum_{i=1}^N (p_i)^a}, \quad (9)$$

where a is a factor that determines how much prioritization is used. Prioritized replay introduces bias in training since it selects some transitions more frequently. To prevent this, the neural network update step is changed to αw_i instead of α, where w_i is defined as

$$w_i = (\frac{1}{N}.\frac{1}{P_i})^b. \quad (10)$$

The exponent b is used to control the bias correction which is more important later during the training than at the beginning. Therefore, b starts at a low value and gradually increases to 1 over time.

SAC is an off-policy reinforcement learning algorithm whose objective is not only to maximize the rewards but also to increase the entropy. Entropy is the likeliness of the algorithm to explore new actions. A high-entropy algorithm prevents premature convergence to a bad local optima and encourages the state space exploration improving the collected transition data. Therefore, the policy is trained to maximize the following objective

$$J(\theta) = \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\theta}}[r(s_t,a_t) + \alpha\mathcal{H}(\pi_\theta(.|s_t))], \quad (11)$$

where $\mathcal{H}(.)$ is the entropy measure, and the parameter α represents the entropy temperature and controls the randomness of the policy versus the reward. In general, there are three functions that SAC needs to learn: 1) the policy with parameter θ, 2) soft Q-value function with parameter w, and 3) soft state value function with parameter ψ. Soft Q-value and state value are determined, respectively, by

$$Q(s_t,a_t) = r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V(s_{t+1})], \quad (12)$$
$$V(s_t) = \mathbb{E}_{a_t\sim\pi}[Q_{s_t,a_t} - \alpha log\pi(a_t|s_t)], \quad (13)$$

To obtain the soft state value function parameters, the following mean squared error is minimized

$$J_V(\psi) = \mathbb{E}_{s_t\sim\mathcal{D}}[\frac{1}{2}(V_\psi(s_t) - \mathbb{E}[Q_w(s_t,a_t) - log\pi_\theta(a_t|s_t)])^2], \quad (14)$$

where $\mathcal{D}$ is the replay buffer. The soft Q function parameters are obtained by minimizing the soft Bellman residual

$$J_Q(w) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}[\frac{1}{2}(Q_w(s_t,a_t) - (r(s_t,a_t) + \gamma\mathbb{E}_{s_{t+1}\sim\rho_\pi(s)}[V_{\overline{\psi}}(s_{t+1})]))^2], \quad (15)$$

where $\overline{\psi}$ is the target value function. Finally, the policy is optimized by minimizing the KL-divergence

$$J_\pi(\theta) = \mathbb{E}_{a_t\sim\pi}[log\pi_\theta(a_t|s_t) - Q_w(s_t,a_t) + logZ_w(s_t)], . \quad (16)$$

Parameter Z is the partition function to normalize the distribution. Finally, the SAC algorithm is implemented as Algorithm 3 [34]. First, the state value function, Q-value function, policy, target value function, and replay buffer parameters are initialized (lines 3-4). Then, for each episode, an action is selected by the policy and executed and the transition is added to the buffer (lines 5-9). If it is time to update, state value function, Q-value function, policy, and target value function are updated using the previously introduced equations (lines 10-14).

As previously mentioned, PPO is an on-policy RL algorithm. It is a type of policy gradient method that was introduced in 2017 [35] as an improvement over Trust Region Policy Optimization (TRPO). PPO uses two neural networks, an actor network whose input is the state and output is the list of probabilities for each action, a critic network, whose input is the state and the output is state value. The objective function of vanilla policy gradient methods is defined as

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log\pi_\theta(a_t|s_t)\hat{A}_t], \quad (17)$$

Algorithm 3 Soft Actor-Critic

Input: Learning rate α, number of episodes N, discount factor γ,
batch size, environment, entropy temperature α
Output: Optimal policy
▷ Initialization
Initialize parameters θ, w, ψ, and $\overline{\psi}$
Initialize replay buffer $\mathcal{D}$
▷ Training
for *Episode* $\in \{1, 2, \ldots, N\}$ **do**
  for $t \in \{0, 1, .., T-1\}$ **do**
    Select action a_t using policy $\pi_\theta(a_t|s_t)$
    Execute the action a_t and receive s_{t+1}, r_{t+1}
    Add transition $(s_t, a_t, r_{t+1}, s_{t+1})$ into buffer $\mathcal{D}$
  for *each gradient update step* **do**
    Update state value function with parameter ψ
    Update Q-value function with parameter w
    Update policy with parameter θ
    Update target value function with parameter $\overline{\psi}$

where $L^{PG}(\theta)$ is the policy loss, $log\pi_\theta(a_t|s_t)$ is the log of probabilities from the actor network, and $\hat{A}_t$ is the advantage function.

To prevent large policy updates, the policy gradient step is restricted by $r_t^s(\theta)$ in TRPO, defined as the probability ratio between the action under the current policy and the action under the previous policy

$$r_t^s(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{18}$$

As a result, the objective function in TRPO is defined as

$$\begin{aligned} &\text{Maximize } \hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t] \\ &\text{subject to } \hat{\mathbb{E}}_t[KL[\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)]] \leq \sigma \end{aligned} \tag{19}$$

Similarly, to restrict the update step in PPO, clipped surrogate objective is defined as

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t^s(\theta)\hat{A}_t, \text{clip}(r_t^s(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)], \tag{20}$$

To minimize the value function error and ensure enough exploration of the agent, the final objective function is defined as

$$L^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}} + c_2 S[\pi_\theta](s_t)], \tag{21}$$

where L_t^{VF} is value function error, $S[\pi_\theta](s_t)$ is entropy bonus, and c_1 and c_2 are hyperparameters. PPO algorithm is given in Algorithm 4. First, the replay buffer, actor, and critic networks are initialized (lines 3-4). Then, for N number of episodes, each actor runs the policy for T time steps, stores the transitions, and computes the advantage functions (lines 5-9). Finally, for a batch of transitions every K epochs, the objective function in Eq. (21) is optimized and the actor and critic parameters are updated.

Algorithm 4 Proximal Policy Optimization

1 **Input:** Learning rate α, number of episodes N, discount factor γ,
batch size, environment, number of parallel agents N^a
2 **Output**: Optimal policy
▷ Initialization
3 Initialize actor and critic networks
4 Initialize replay buffer B
▷ Training
5 **for** *Episode* $\in \{1, 2, \ldots, N\}$ **do**
6 **for** *actor* $\in \{1, 2, \ldots, N^a\}$ **do**
7 Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
8 Store transitions in the replay buffer
9 Compute advantage estimates $\hat{A}_1, \hat{A}_2, \ldots, \hat{A}_T$
10 Optimize $L^{CLIP+VF+S}$ w. r. t. θ, with K epochs and batch size $M \leq N_a T$
11 $\theta_{old} \leftarrow \theta$

D. ACTION REDUCTION

The action space in DNR is quite large. Presented RL algorithms either would not converge to the optimal solution with such a large action space, or they would require a very long training time and computational resources. Therefore, the size of the action space needs to be reduced while keeping only feasible actions.

To improve the quality of the DNR solutions found by RL algorithms, in this study, the size of the action space is reduced in two stages. At the first stage, random sampling is used to define an action space for RL. Then, using the selected actions, DQN algorithm is implemented. At the end of the training, the agent has learned the action values. Therefore, it is used to only select the actions with the highest q-values to reduce the size of the action space at the second stage.

III. SIMULATION AND RESULTS

The proposed RL method is first applied to the IEEE 33-node system and then to a 136-node distribution system to demonstrate the scalability of the developed method. The data and parameters used for simulations are presented in this section, along with the obtained results.

A. EXPERIMENTAL DATA SETUP

The tuned parameters of the five described RL algorithms are given in Table 1. Since the number and range of hyperparameters for RL algorithms are very large, the hyperparameter tunning is performed manually, by changing the value of one hyperparameter, observing the result, and then deciding the next value. The daily load data is generated similar to [17] and then, considering a normal distribution and 15% standard deviation to the generated daily IEEE 33-node and 136-node

TABLE 1. Parameters of RL algorithms.

Algorithm	Parameters	Values
DQN	learning rate	10^{-3}
	number of hidden units	$\{1024, 512\}$
	batch size	512
	discount factor	0.99
	synchronize every	10
Dueling DQN	learning rate	10^{-3}
	number of hidden units	$\{1024, 512\}$
	batch size	512
	discount factor	0.99
	synchronize every	10
DQN with Prioritized Experience Replay	learning rate	10^{-3}
	number of hidden units	$\{1024, 512\}$
	batch size	512
	discount factor	0.99
	synchronize every	10
	parameter a	0.5
	parameter b	0.4
SAC	learning rate	10^{-4}
	batch size	512
	discount factor	0.99
	number of hidden units	$\{1024, 512\}$
	initial temperature parameter	1
	soft update interpolation factor	0.05
PPO	learning rate	$3 \cdot 10^{-4}$
	discount factor	0.99
	number of epochs (K)	512
	batch size	256
	clip rate	0.2
	number of hidden units	$\{1024, 512\}$
	L2 regularization coefficient	0.5
	entropy coefficient	10^{-3}
	entropy coefficient decay	0.99

network loads, the load data is generated for the required number of days. The minimum and maximum voltage limits are considered to be 0.95 p.u. and 1.05 p.u., respectively. Parameter C^l is set to 10,000 and λ to 100,000 based on empirical performance. The neural networks are coded using PyTorch and in Python environment. All RL algorithms use a four-layer neural network structure, with 1024 neurons in the first hidden layer and 512 neurons in the second hidden layer. NetworkX package is used as the graph analysis tool in Algorithm 1 and the power network is modeled using pandapower package.

Algorithm 1 finds 14,401 feasible actions for 33-node system and 56,996 actions for the 136-node system. The initial sampling randomly selects 300 and 1,000 actions for the 33- and 136-node systems, respectively. After applying the initial DQN, the convergence curve and action values for 33-node and 136-node systems are given in Fig. 2 and Fig. 3, respectively. Among these actions, the top 80 actions with the highest q-values are chosen as the action space for the 33-node test system and the top 400 are chosen as the action space of the 136-node system.

B. 33-NODE TEST SYSTEM

The 33-node test system is depicted in Fig. 4. It is assumed that all lines can be opened and closed using switches and the dashed lines are initially open.

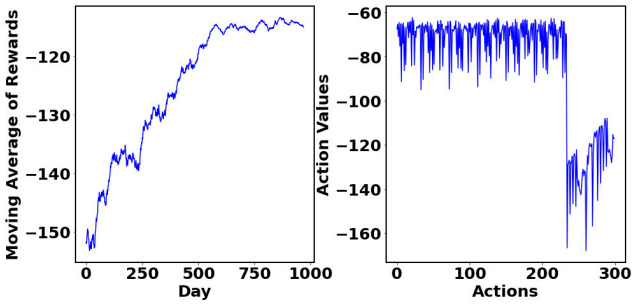

FIGURE 2. 30-step moving average of daily rewards and action values for 33-node system with 300 actions.

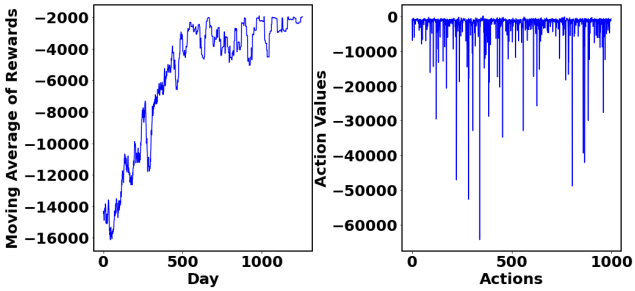

FIGURE 3. 30-step moving average of average daily rewards and action values for 136-node system with 1000 actions.

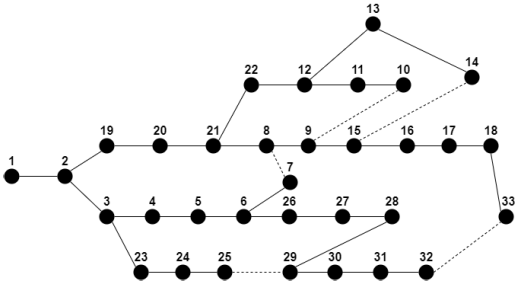

FIGURE 4. 33-node test system.

Average daily rewards and their moving averages for the five RL algorithms are illustrated in Fig. 5 and Fig. 6, respectively. As can be seen, SAC has the fastest convergence. However, the final obtained rewards are lower compared to DQN and dueling DQN. DQN and dueling DQN have very similar performance and PPO has the weakest performance among these algorithms.

Since in on-policy RL algorithms the agent uses the same behavior and target policy, these algorithms are more likely to become trapped in a local minimum. As a result, PPO, which is the only on-policy algorithm in this study, has the worst performance among other algorithms. Policy gradient methods such as SAC and PPO search directly for the optimal policy instead of estimating the action-value. These methods require fresh samples from the environment obtained with the current policy. However, value methods such as DQN can take advantage of the old data obtained from an older policy stored in the memory. Therefore, policy methods usually require more interaction with the environment to reach the

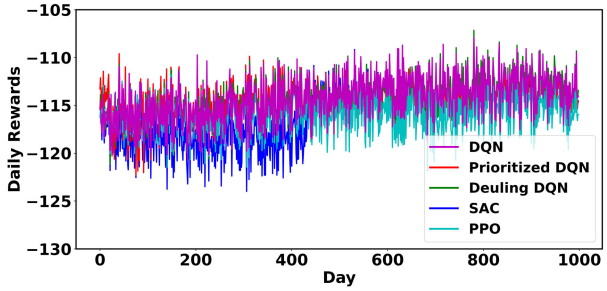

FIGURE 5. Average daily rewards for 33-node system during training using five RL algorithms.

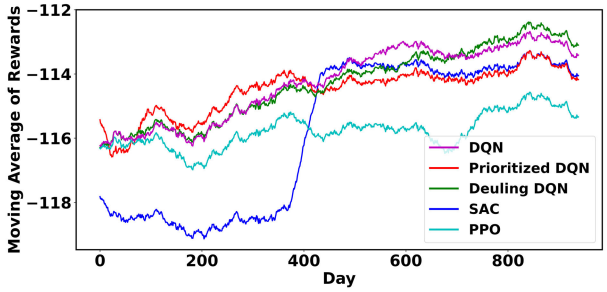

FIGURE 6. 60-step moving average of daily rewards for 33-node system using five RL algorithms.

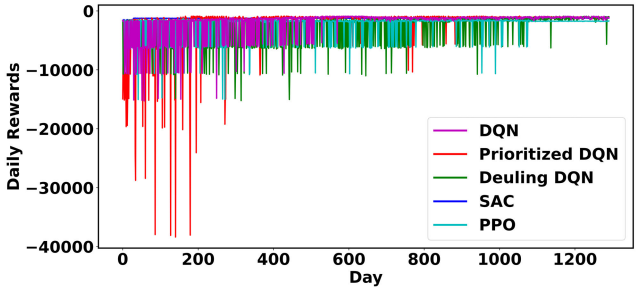

FIGURE 7. Average daily rewards for 136-node system during training using five RL algorithms.

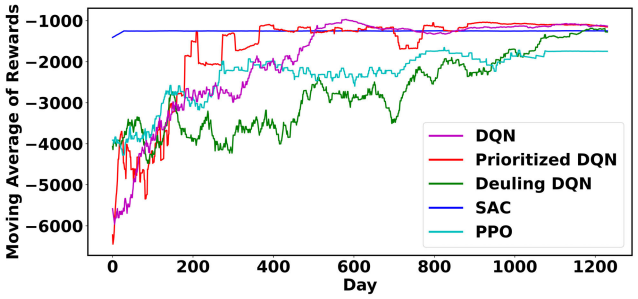

FIGURE 8. 60-step moving average of daily rewards for 136-node system using five RL algorithms.

TABLE 2. System loss obtained by single-hour DNR.

Algorithm	Test System	
	33-node	136-node
DQN	6.201	51.606
Dueling DQN	6.188	72.401
Prioritized DQN	6.273	144.373
SAC	6.273	72.384
PPO	6.342	100.411
SOE [17]	6.740	51.651
GA [36]	6.740	51.911
AACO [37]	6.740	51.655
CSA [38]	6.740	-
Action Sampling DQN	6.266	90.197

TABLE 3. Execution time comparison.

Algorithm	Execution Time [s]	
	33 bus	136 bus
PPO	0.206	0.776
DQN	0.207	0.786
Dueling DQN	0.502	1.911
SAC	0.453	1.916
Prioritized DQN	0.457	1.964
SOE [17]	1	16.9
AACO [37]	0.3	894.20
MP [17]	19	1236
Approximate RL Training Time	5727	25190

most optimal solution. On the other hand, since they search directly for the optimal policy, they might converge faster.

C. 136-NODE TEST SYSTEM

The 136-node distribution system has a relatively complex structure, as depicted in Fig. 9. The average daily rewards and their moving averages for the 136-node system are shown in Fig. 7 and Fig. 8, respectively. Similar to the 33-node system, SAC has the fastest convergence. However, the final obtained rewards are lower compared to DQN and DQN with prioritized experience replay which obtain similar rewards at the end of the training. However, DQN with prioritized experience replay converges slightly faster. By comparison to the 33-node system, it can be seen that for larger systems, the prioritized experience replay is more effective and converges faster.

D. COMPARISON OF RESULTS

Table 2 shows the network loss after a single-hour DNR using each of the five RL algorithms, the SOE method [17], genetic algorithm (GA) [36], adaptive ant colony optimization (AACO) method [37], and cuckoo search algorithm (CSA) [38] from the literature. Among the compared algorithms, DQN attains the lowest network loss, outperforming other published methods. It can also be inferred that the DQN-based action reduction approach was effective and that the resulting action space was sufficient to reduce the system loss in the DQN, Dueling DQN, and SAC algorithms. It can be concluded that, in terms of stability and optimality, DQN is the best-performing algorithm. In terms of convergence speed, SAC performs the best.

The computational times of the RL methods are compared against SOE, AACO, and mathematical programming (MP) methods (including mixed-integer linear and mixed-integer nonlinear programming) [17] in Table 3. As it can be inferred from this table, RL methods are at least eight times faster than SOE, AACO, and MP methods for the 136-node system. The computational speed of the RL algorithms is even more signified as the size of the system grows. However, RL algorithms

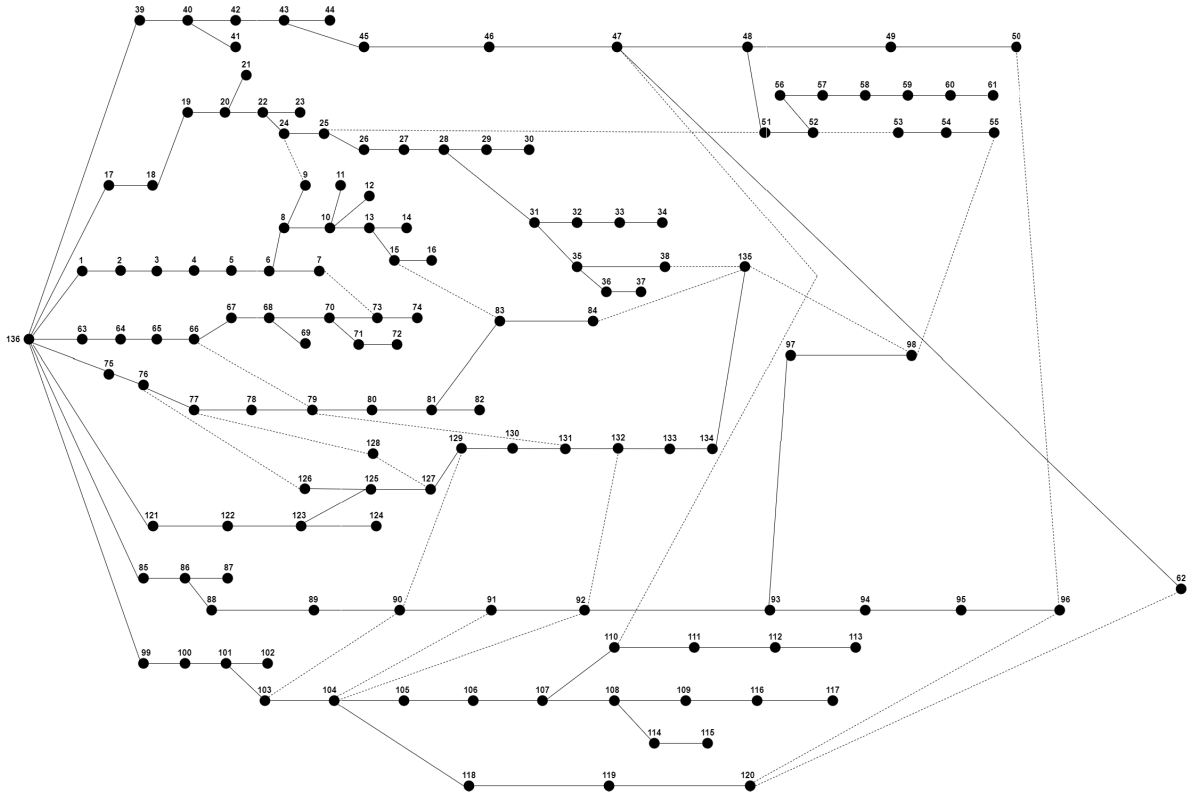

FIGURE 9. 136-node test system.

can have a very long training time, as presented in the same table.

IV. CONCLUSION

In this paper, the DNR problem is formulated as a Markov decision process to implement and compare five RL algorithms, including DQN, dueling DQN, DQN with prioritized experience replay, SAC, and PPO. The proposed RL-based approach was applied to the 33- and 136-node test systems. The results showed that among these algorithms, SAC had the fastest convergence, while DQN performed best in terms of stability and optimality. In addition, a new DQN-based action reduction method was developed to reduce the size of the action space. The effectiveness of the new reduction approach was tested by comparing the pre- and post-sampling results as well as the results from the literature.

Future studies will focus on developing new action space sampling methods for RL and performing sensitivity analysis for various action space selections. In addition, various objectives for the RL agent will be defined and tested. Finally, the effectiveness of the proposed method for increasing electric vehicle penetration will be investigated.

REFERENCES

[1] N. Gholizadeh and P. Musilek, "Distributed learning applications in power systems: A review of methods, gaps, and challenges," *Energies*, vol. 14, no. 12, p. 3654, Jun. 2021.

[2] T. Yang, Y. Guo, L. Deng, H. Sun, and W. Wu, "A linear branch flow model for radial distribution networks and its application to reactive power optimization and network reconfiguration," *IEEE Trans. Smart Grid*, vol. 12, no. 3, pp. 2027–2036, May 2021.

[3] W. Zheng, W. Huang, D. J. Hill, and Y. Hou, "An adaptive distributionally robust model for three-phase distribution network reconfiguration," *IEEE Trans. Smart Grid*, vol. 12, no. 2, pp. 1224–1237, Mar. 2021.

[4] E. Quintana and E. Inga, "Optimal reconfiguration of electrical distribution system using heuristic methods with geopositioning constraints," *Energies*, vol. 15, no. 15, p. 5317, Jul. 2022.

[5] H. Ahmadi and J. R. Marti, "Distribution system optimization based on a linear power-flow formulation," *IEEE Trans. Power Del.*, vol. 30, no. 1, pp. 25–33, Feb. 2015.

[6] Q. Chen, W. Wang, H. Wang, J. Wu, and J. Wang, "An improved beetle swarm algorithm based on social learning for a game model of multiobjective distribution network reconfiguration," *IEEE Access*, vol. 8, pp. 200932–200952, 2020.

[7] Q. Chen, W. Wang, H. Wang, J. Wu, X. Li, and J. Lan, "A social beetle swarm algorithm based on grey target decision-making for a multiobjective distribution network reconfiguration considering partition of time intervals," *IEEE Access*, vol. 8, pp. 204987–205013, 2020.

[8] J. Wang, W. Wang, Z. Yuan, H. Wang, and J. Wu, "A Chaos disturbed beetle antennae search algorithm for a multiobjective distribution network reconfiguration considering the variation of load and DG," *IEEE Access*, vol. 8, pp. 97392–97407, 2020.

[9] J. Wang, W. Wang, H. Wang, and H. Zuo, "Dynamic reconfiguration of multiobjective distribution networks considering DG and EVs based on a novel LDBAS algorithm," *IEEE Access*, vol. 8, pp. 216873–216893, 2020.

[10] T. Zhong, H.-T. Zhang, Y. Li, L. Liu, and R. Lu, "Bayesian learning-based multi-objective distribution power network reconfiguration," *IEEE Trans. Smart Grid*, vol. 12, no. 2, pp. 1174–1184, Mar. 2020.

[11] A. Kavousi-Fard, T. Niknam, and M. Fotuhi-Firuzabad, "A novel stochastic framework based on cloud theory and θ-modified bat algorithm to solve the distribution feeder reconfiguration," *IEEE Trans. Smart Grid*, vol. 7, no. 2, pp. 740–750, Jun. 2016.

[12] P. Akaber, B. Moussa, M. Debbabi, and C. Assi, "Automated post-failure service restoration in smart grid through network reconfiguration in the presence of energy storage systems," *IEEE Syst. J.*, vol. 13, no. 3, pp. 3358–3367, Sep. 2019.

[13] A. Zhou, H. Zhai, M. Yang, and Y. Lin, "Three-phase unbalanced distribution network dynamic reconfiguration: A distributionally robust approach," *IEEE Trans. Smart Grid*, vol. 13, no. 3, pp. 2063–2074, May 2022.
[14] X. Cao, J. Wang, J. Wang, and B. Zeng, "A risk-averse conic model for networked microgrids planning with reconfiguration and reorganizations," *IEEE Trans. Smart Grid*, vol. 11, no. 1, pp. 696–709, Jan. 2020.
[15] Y. Song, Y. Zheng, T. Liu, S. Lei, and J. D. Hill, "A new formulation of distribution network reconfiguration for reducing the voltage volatility induced by distributed generation," *IEEE Trans. Power Syst.*, vol. 35, no. 1, pp. 496–507, Jan. 2020.
[16] H. Hong, Z. Hu, R. Guo, J. Ma, and J. Tian, "Directed graph-based distribution network reconfiguration for operation mode adjustment and service restoration considering distributed generation," *J. Mod. Power Syst. Clean Energy*, vol. 5, no. 1, pp. 142–149, Jan. 2017.
[17] J. Zhan, W. Liu, C. Y. Chung, and J. Yang, "Switch opening and exchange method for stochastic distribution network reconfiguration," *IEEE Trans. Smart Grid*, vol. 11, no. 4, pp. 2995–3007, Jul. 2020.
[18] R. K. Mishra and K. S. Swarup, "Adaptive weight-based self reconfiguration of smart distribution network with intelligent agents," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 6, pp. 464–472, Dec. 2018.
[19] N. Xia, J. Deng, T. Zheng, H. Zhang, J. Wang, S. Peng, and L. Cheng, "Fuzzy logic based network reconfiguration strategy during power system restoration," *IEEE Syst. J.*, vol. 16, no. 3, pp. 4735–4743, Sep. 2022.
[20] Z. Liu, Y. Liu, G. Qu, X. Wang, and X. Wang, "Intra-day dynamic network reconfiguration based on probability analysis considering the deployment of remote control switches," *IEEE Access*, vol. 7, pp. 145272–145281, 2019.
[21] T. V. Tran, B.-H. Truong, T. P. Nguyen, T. A. Nguyen, T. L. Duong, and D. N. Vo, "Reconfiguration of distribution networks with distributed generations using an improved neural network algorithm," *IEEE Access*, vol. 9, pp. 165618–165647, 2021.
[22] W. Huang, W. Zheng, and D. J. Hill, "Distribution network reconfiguration for short-term voltage stability enhancement: An efficient deep learning approach," *IEEE Trans. Smart Grid*, vol. 12, no. 6, pp. 5385–5395, Nov. 2021.
[23] N. Liu, C. Li, L. Chen, and J. Wang, "Hybrid data-driven and model-based distribution network reconfiguration with lossless model reduction," *IEEE Trans. Ind. Informat.*, vol. 18, no. 5, pp. 2943–2954, May 2022.
[24] X. Ji, Z. Yin, Y. Zhang, B. Xu, and Q. Liu, "Real-time autonomous dynamic reconfiguration based on deep learning algorithm for distribution network," *Electr. Power Syst. Res.*, vol. 195, Jun. 2021, Art. no. 107132.
[25] Y. Gao, B. Foggo, and N. Yu, "A physically inspired data-driven model for electricity theft detection with smart meter data," *IEEE Trans. Ind. Informat.*, vol. 15, no. 9, pp. 5076–5088, Sep. 2019.
[26] R. A. Pegado and Y. P. M. Rodriguez, "Distribution network reconfiguration with the OpenDSS using improved binary particle swarm optimization," *IEEE Latin Amer. Trans.*, vol. 16, no. 6, pp. 1677–1683, Jun. 2018.
[27] C. Huang, H. Zhang, L. Wang, X. Luo, and Y. Song, "Mixed deep reinforcement learning considering discrete-continuous hybrid action space for smart home energy management," *J. Mod. Power Syst. Clean Energy*, vol. 10, no. 3, pp. 743–754, 2022.
[28] O. B. Kundačina, P. M. Vidović, and M. R. Petković, "Solving dynamic distribution network reconfiguration using deep reinforcement learning," *Electr. Eng.*, vol. 104, no. 3, pp. 1487–1501, Jun. 2022.
[29] B. Wang, H. Zhu, H. Xu, Y. Bao, and H. Di, "Distribution network reconfiguration based on NoisyNet deep Q-learning network," *IEEE Access*, vol. 9, pp. 90358–90365, 2021.
[30] S. H. Oh, Y. T. Yoon, and S. W. Kim, "Online reconfiguration scheme of self-sufficient distribution network based on a reinforcement learning approach," *Appl. Energy*, vol. 280, Dec. 2020, Art. no. 115900.
[31] Y. Gao, W. Wang, J. Shi, and N. Yu, "Batch-constrained reinforcement learning for dynamic distribution network reconfiguration," *IEEE Trans. Smart Grid*, vol. 11, no. 6, pp. 5357–5369, Nov. 2020.
[32] Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 1995–2003.
[33] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.
[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1861–1870.
[35] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[36] J. C. Cebrian and N. Kagan, "Reconfiguration of distribution networks to minimize loss and disruption costs using genetic algorithms," *Electr. Power Syst. Res.*, vol. 80, no. 1, pp. 53–62, Jan. 2010.
[37] A. Swarnkar, N. Gupta, and K. R. Niazi, "Adapted ant colony optimization for efficient reconfiguration of balanced and unbalanced distribution systems for loss minimization," *Swarm Evol. Comput.*, vol. 1, pp. 129–137, Sep. 2011.
[38] T. T. Nguyen and A. V. Truong, "Distribution network reconfiguration for power loss minimization and voltage profile improvement using cuckoo search algorithm," *Int. J. Elect. Power Energy Syst.*, vol. 68, pp. 233–242, Jun. 2015.

NASTARAN GHOLIZADEH received the B.Sc. degree from the University of Tabriz, Iran, in 2017, and the M.Sc. degree from the Amirkabir University of Technology, in 2019. She is currently pursuing the Ph.D. degree in software engineering and intelligent systems with the University of Alberta. Her research interests include machine learning applications in power systems, reinforcement learning for grid control, and optimization of power systems.

NAZLI KAZEMI (Graduate Student Member, IEEE) received the B.Sc. degree in electrical and computer engineering from the Iran University of Science and Technology, Tehran, Iran, in 2018, and the M.Sc. degree in electrical and computer engineering from the University of Alberta, Edmonton, AB, Canada, in 2020, where she is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department. Her research interests include applications of machine learning in sensors, renewable energy systems, environmental sensing, monitoring, modeling, microwave active sensors, metamaterials, and biosensors.

PETR MUSILEK (Senior Member, IEEE) received the Ing. degree (Hons.) in electrical engineering and the Ph.D. degree in cybernetics from Military Academy, Brno, Czech Republic, in 1991 and 1995, respectively. In 1995, he was appointed as the Head of the Computer Applications Group, Institute of Informatics, Military Medical Academy, Hradec Králové, Czech Republic. From 1997 to 1999, he was a NATO Science Fellow with the Intelligent Systems Research Laboratory, University of Saskatchewan, Canada. In 1999, he joined the Department of Electrical and Computer Engineering, University of Alberta, Canada, where he is currently a Full Professor. Since 2016, he has been working as the Director of the Computer Engineering Program and the Associate Chair (Undergraduate). He is also the Associate Chair (Research and Planning). His research interests include artificial intelligence and energy systems. He developed a number of innovative solutions in the areas of renewable energy systems, smart grids, wireless sensor networks, and environmental monitoring and modeling.