# Use of Augmented Random Search Algorithm for Transmission Line Control in Smart Grids – A Comparative Study with RNA-based Algorithms

**Antonio B. S. Rufino, Filipe Saraiva**

Instituto de Ciências Exatas e Naturais – Universidade Federal do Pará (UFPA) – Belém – PA – Brazil

antonio.rufino@icen.ufpa.br, saraiva@ufpa.br

**Abstract.** *Due to climate change challenges, countries are diversifying their energy sources to reduce carbon emissions and adopt cleaner alternatives. However, integrating these new energy sources into existing power grids poses challenges, such as increased intermittency. Prior studies have shown that active control of the power grid's topology can address these issues. This research aims to demonstrate the effectiveness of the Augmented Random Search (ARS) algorithm as a faster alternative to neural network-based rein- forcement learning algorithms. The ARS algorithm can achieve comparable results to neural networks in significantly less time, enabling a broader range of tests and reducing computational training costs.*

**Keywords:** Reinforcement Learning, Deep Learning, Augmented Randon Search, Micro-Grids, Active Topology.

## 1. Introduction

In the economic discourse of several countries, environmental concerns have become a central issue. This concern culminated in the 2015 Paris Agreement, signed by more than 190 countries. The aim of the agreement is to reduce emissions of polluting gases and to limit the global temperature increase to 2°C above pre-industrial levels through national contributions.

Emphasising the article 4 of the agreement which proposes a balance between carbon emissions and the capacity of natural sinks (forests), from this point on the signatory countries begin to look for ways to reduce their $CO_2$ emissions, including the incorporation of non-polluting energy sources, such as solar and wind. This decision directly affects the old energy matrix, mainly based on highly polluting fossil fuels, generating changes in the structure of electrical systems [He et al. 2021].

The problem with these new structures in the electricity network is that their high intermittency can damage the networks, which are unable to respond to this new model of energy production [Bhalshankar and Thorat 2016]. This is because they were not designed for situations in which the energy load passing through them can vary abruptly in a short period of time. To solve this problem, a new concept has emerged called smart grids or intelligent energy networks [Flick and Morehouse 2011].

Smart grids are systems that combine various elements such as energy storage, supply and demand management, and distributed generation, coordinated for maximum

efficiency and cost reduction [Pratt 2004]. Combining these elements with computational intelligence techniques and telecommunications demonstrated the potential for automation of controllers with intelligent software capable of predicting situations and making decisions based on data [Donnot et al. 2017]. Among the various artificial intelligence techniques applied to power system problems, reinforcement learning algorithms outperform other algorithms in controlling the status of transmission lines, there- by reducing damage to the electrical grid [Lan et al. 2019][ Marot et al. 2021].

Within reinforcement learning, the most prominent algorithms for smart grid control are Q-Learning [Sun et al. 2017][ Kofinas et al. 2018] and DQN [Marot et al. 2021][ Rocchetta et al. 2019]. However, Q-Learning has a limitation related to the state X action set, which increases the necessary processing as the set increases (known as combinatorial explosion)[ Van de Wiele et al. 2020]. On the other hand, neural network derivatives such as DQN, DDQN, PPO, etc. are highly sensitive to the hyperparameter set and also require a relative amount of training time, although less than the usual for Q-learning [Watkins and Dayan 1992].

In this perspective, this paper presents, as a contribution, the feasibility study of an RL algorithm capable of solving the problem of state control of power lines to prevent their collapse, in a shorter time and with less computational power. The Augmented Random Search (ARS) algorithm is proposed as a viable alternative to algorithms based on neural networks for the problem of transmission line state control. The present study utilized a competition platform grid2op [Marot et. al. 2021] for testing and compared ARS to 3 other RL algorithms based in neural networks - DQN, DDQN and PPO. The results point ARS can reach the same performance of those algorithms and 5 times faster.

The paper is structured as follows. Section 2 provides a brief overview of the state of the art in the use of reinforcement learning in smart grids. Section 3 describesthe transmission line overheating problem and demonstrates the studied environment. In Section 4, the algorithms implemented in this work are described. Section 5 outlines the proposed methodology for solving this problem using reinforcement learning and the Augmented Random Search algorithm. Section 6 discusses the obtained results, and Section 7 concludes the article and highlights its contributions.

## 2. State of the Art

This section is a list of some of the papers that are relevant to the topic under analysis in this document.

Lan et al. [Lan et al. 2019] present in their research a set of techniques used to ensure the stability of an energy system through control of the active topology. These techniques include DDQN algorithms combined with an early warning system to assist in the search for a more efficient action policy. Despite the excellent results, the article implements a solution highly focused on the single tested environment with a decision-making assistant agent, while the present work aims for a more general control with the entire responsibility of control placed on the algorithms, without the use of an auxiliary agent.

Rocchetta et al [Rocchetta et al. 2019] propose a way to increase energy profits while mitigating damage to the power system by using DQN to balance generator powers and maximise maintenance schedules in the system, thereby reducing fatigue. The

state action space consists of 3 adjustment actions for each generator and 2 preventive maintenance and correction actions for each line. The presented article has a small state-action set and also does not make comparisons with other algorithms, in addition to the control environment being excessively small when compared to the one proposed in the present research.

Marot et al. [Marot et al. 2021] conducted a summary research on the 2019 L2RNP competition, in which the goal was to control the active topology of the grid using reinforcement learning. The authors provide an overview of the competition and show the different approaches taken by the winners, where all of them managed to overcome the initial control challenge.

Bollenbacher and Rhein [Bollenbacher and Rhein 2017] proposed a reinforcement learning-based system capable of selecting the best configuration for an energy hub, taking into account the power plants, customers, and energy storage. The aim was to reduce costs while fully meeting demand without changing consumption patterns. Your approach does not take into consideration renewable generators, which can beseen as problematic, as the integration of renewable generators into the grid can have a profound impact on its operation.

Hu et al. [Hu et al. 2021] introduces the use of multi-agent soft agent criticism (MASAC) based on policy gradient to solve the volt/VAR control (VVC). The MASAC method uses a multi-agent deep reinforcement learning (MADRL) approach to address the real-time VVC problem using PV and WT inverters regulation. The main difference between the proposed work and the present article is the use of multi-agents, which has not been implemented in the current study.

Liul J. et al. [Liul et al. 2022] proposed autonomous decentralized control of distributed generations (DGs) using multi-agent reinforcement learning improves scalability, privacy, and communication efficiency. This article also presents a multi-agent application, unlike the article presented here, which focuses on a central agent.

Bi W. et al. [Bi et al. 2020] employs a DQN architecture to perform a set of battery dispatch actions including charging, idling, or discharging batteries. The main difference between this study and the standard one here is that in addition to the control environment being much smaller comprising a solar panel, a battery and a wind generator, the study does not compare other algorithms.

Quakernack, L [Quakernack et al. 2022] proposes a DDQN approach with multiple agents to control the charging power of electric vehicles and the charge and discharge power of battery storage systems in a low-voltage grid. The simulations showed promising results in reducing transformer utilization rates and preventing reverse power flow. The works differ in applications, however, this work is an important indicator of the use of RL in power systems.

## 3. Problem Description

The overheating of transmission lines is an increasingly common problem in electrical power systems, especially due to the increased intermittency of renewable generators. With the growing use of renewable energy sources such as wind and solar power, electricity generation has become more unpredictable, which can lead to an increase in intermittency and fluctuations in the network's voltage [Yang et al. 2010]. As a result,

transmission lines may become overloaded and overheat, which can lead to failures and even blackouts.

This problem is particularly critical in regions with high penetration of renewable energy, where the transmission infrastructure may not be adequately sized to handle the variations in power generation [Sukumar et al. 2010]. In addition, the overheating of transmission lines can increase operation and maintenance costs, as well as safety risks for invol-ved workers.

### 3.1 Modeled Environment

The environment was simulated using a power system simulation framework designed for reinforcement learning called grid2op [Marot et. Al. 2021]. This library allows actions to be executed within the energy grid and monitor the effects that are perceived. The algorithms were submitted in two different environments described in Table 1.

**Table 1. Environments and their elements**

| Name | Lines | Loads | Substations | Generators | Renewable Generators |
|---|---|---|---|---|---|
| l2rpn_case14_sandbox | 20 | 11 | 14 | 6 | 3 |
| l2rpn_wcci_2020 | 59 | 37 | 36 | 22 | 12 |

The substations are interconnected through transmission lines, receiving or sending energy to each other, while each substation internally controls the generators and loads, functioning as a hub. The focus of the work is on balancing the energy that passes through these transmission lines and preventing them from going into energy overflow.

The environment also has some constraints that guide how the RL agent will operate within the system during a 5 minute time step. The ones that most affect the proper functioning of the system are:

1. The lines are disconnected after reaching their thermal limit.

2. A line is disconnected if it exceeds or reaches its thermal limit for 3-time step intervals.

3. A topological change can only be made in the same substation after 3-time step intervals.

4. A voltage line can reach up to 2 times its thermal limit, but if it exceeds that, it will be immediately shut down.

5. A line can remain disconnected for up to 12-time step intervals.

6. Only one substation and one transmission line can be modified in 2 consecutive time step intervals.

### 3.2 Rewards

Three different types of rewards were used for the simulations. The first type is related to training neural network algorithms, the second type is related to training the ARS algorithm, and the third type is related to the evaluation method of the algorithms.

The "LinesCapacityReward" was used to train the neural network algorithms. It re-turns 1 if no current is flowing through the line, and if all lines are in overflow, it re-turns 0. This allows the algorithm to find an appropriate middle ground for controlling the flow of energy passing between the lines.

ARS is an algorithm that is entirely focused on action policy [Gao et al. 2023], which allows it to not need to work with future states or check the actions taken like the DQN and DDQN algorithms, as long as the final policy is the most suitable. Based on this characteristic, "EpisodeDurationReward" is a more suitable reward function for ARS because it ignores individual actions and evaluates only the set of actions taken, something that could not be done with the other algorithms because it does not provide immediate feedback, which is necessary for the other three proposed algorithms.

During the training of the ARS algorithm, the "EpisodeDurationReward" was used as the reward function, which is simply the division of the number of timesteps the algorithm takes by the number of timesteps available in the episode the algorithm is running. Therefore, the maximum reward is 1 and the minimum reward is 0, with 1 rep-resenting the algorithm's complete success and 0 representing an agent that was unable to perform any acceptable control within the environment.

Finally, the test reward, which is the same reward for all agents, is based on the operation costs of a grid, therefore, a calculation is made to evaluate the costs of the operations within the grid, which includes failures in operation and losses to consumers. The equation 1 that determines this reward is given by:

$$R = \sum Closs(t) + \sum Load(t) * p(t) * beta \quad (1)$$

Where Closs(t) is the cost of energy loss due to the joule effect in the line. The operation cost is then added to the cost of a blackout, if it occurs. This cost is equal to the load that is not supplied (Load(t)) multiplied by the energy cost at the moment (p(t)) multiplied by a factor beta > 1.

### 3.3 State X Action

The chosen state-action set was the same for all algorithms. The elements described in the list below represent the set of observed states that were used in the algorithms, including observations of consumers, voltage line states, and the voltage passing through the voltage lines.

- Load_p: It is a vector that represents the value of the active power of each load.
- Load_v: It is the vector that represents the voltage on the bar for each connected load.
- V_or: It is a vector that represents the voltage at the bus where the source end of each line is connected.
- Rho: It is a vector that represents how much of the thermal capacity of eac line isbeing used, with overflow being considered from 1.0.

- P_ex: It is a vector that represents the active power flow at each end of the voltage lines.
- Q_ex: It is a vector that represents the reactive power flow at each end of the voltage lines.
- Q_or: It is a vector representing the reactive power flow at the source ends of each voltage line.

The set of observed states for the "l2rpn_case14_sandbox" environment is a 122-position vector and the set of actions is a 20-position vector that indicates whichline should be modified. For the "l2rpn_wcci_2020" environment, the state set has a size of 428 and the action set has a size of 59.

The action used in the environment is "line_change_status", which changes the current status of the line based on the line number. For example, if line 2 is selected and it is currently off, the action will turn it on, and vice versa if it is already on.

## 4. Algorithm Descriptions

This section describes the algorithms applied to the problem in this study. They are four: DQN, DDQN, ARS, and PPO.

### 4.1 DQN

The DQN (Deep Q-Network) algorithm is a deep reinforcement learning technique that uses neural networks to approximate a Q-function, which determines the expected reward for each action in a given state. The goal of the algorithm is to learn an optimal policy to solve a specific Markov decision process control problem [Fan et al. 2019].

The basic structure of DQN consists of an agent that interacts with an environment, receiving observations of the current state and choosing actions to maximize an expected future reward. The agent uses a deep neural network to approximate the Q function, updating its weights at each step based on an objective function that minimizes the difference between the actual rewards and those estimated by the network [Roderick et al. 2017].

$$Q(s,a) = R + \gamma * \max(Q(s',a')) \quad (2)$$

The Equation 2 represents the Bellman update function, where Q(s,a) is the value of action a in state s, R is the reward received for executing action a in state s, s' is the resulting state from transitioning from states to states after executing action a, a' is the action that maximizes the Q value in state s', γ is a discount factor that determinesthe relative importance of immediate and future rewards, and max Q(s',a') is the maxi- mum Q value in state s' for all actions a'.

### 4.2 DDQN

The DDQN (Double Deep Q-Network) algorithm is an extension of DQN that aims to mitigate the problem of overestimating Q, which can occur when DQN uses a single neural network to approximate the Q function. DDQN [Van Hasselt et al. 2016] uses two neural networks instead of one: a main neural network and a target neural network,

and updates both networks using a soft update approach. It updates the weights using the same Equation 2.

The difference in DDQN is that the target neural network Q' is updated using a soft update approach, rather than a full weight update at each training step. Instead of completely replacing the weights of the target neural network with the weights of the main neural network, the soft update is performed by gradually updating the weights of the target neural network towards the weights of the main neural network.

$$\Theta' = t * \Theta + (1-t) * \Theta' \quad (3)$$

The Equation 3 represents the soft update function, where θ represents theweights of the main neural network, θ' represents the weights of the target neural net- work, t is a smoothing parameter that controls the update speed of the target neural net- work.

### 4.3 ARS

Augmented Random Search (ARS) is an optimization algorithm that uses an augmented random search approach to find the optimal policy in a continuous action space. ARS uses a numerical gradient approach to update policy parameters without explicitly cal- culating the objective function gradients. Instead, ARS uses information about the varia-tion of the objective function in different directions obtained from random samples to update policy parameters.

$$\theta \leftarrow \theta + \alpha * (1/m) * \sum(\delta f/\delta\theta) * \Delta\theta \quad (4)$$

The equation 4 describes the parameter update of ARS [Gao et al. 2023], where θ represents the policy parameters, α is the learning rate, m is the number of training samples, δf/δθ is an estimate of the objective function gradient with respect to policy parameters θ, and Δθ is a random search direction. The gradient estimate δf/δθ is ob-tained from two training samples: a positive sample, where the policy is evaluated with a small deviation in the parameters, and a negative sample, where the policy is evaluat-ed with the same deviation, but with the opposite sign. The difference between the two samples is used as an estimate of the objective function gradient with respect to the pol-icy parameters.

The random search direction Δθ is obtained from a multivariate normal distribu-tion centered on the previous gradient direction. This distribution is parameterized by a diagonal covariance matrix, which is updated at each training step with information about the variations of the objective function in different directions.

### 4.4 PPO

Proximal Policy Optimization (PPO) is a policy optimization algorithm that seeks to maximize the expected reward in reinforcement learning problems. It uses an ascending gradient approach to adjust policy parameters and incorporates a penalty to avoid large changes in the parameters that may lead to significant drops in policy performance.

The objective of PPO is to maximize the objective function J(θ), which is given by the weighted sum of the expected rewards of a policy parameterized by a vector of parameters θ over a set of episodes. Equation 5 describes the objective function as:

$$J(\theta) = E[R(\tau,\theta)] \quad (5)$$

Where R(τ,θ) is the cumulative reward for an episode τ given the parameter vector θ. The parameter vector update is performed through an ascending gradient approach that uses a first-order gradient estimate based on samples.

$$\theta' = \text{argmax}\_\theta\ E[\min(\Omega(\theta) * A(\tau,\theta), \text{clip}(\Omega(\theta),1-\varepsilon,1+\varepsilon)A(\tau,\theta))] \quad (6)$$

Equation 6 describes the parameter vector update, where Ω(θ) is the ratio between the new policy and the old policy, A(τ,θ) is the estimated advantage of an episode τ under the current policy, and ε is a clipping parameter that limits the change in the policy ratio. Respecting Equation 6, the update is based on a loss function that includes terms for the clipping penalty and the entropy penalty, with 'clip()' being responsible for restricting the change in parameters θ.

$$L(\theta) = E[\min(\Omega(\theta) * A(\tau,\theta), \text{clip}(\Omega(\theta),1-\varepsilon,1+\varepsilon) * A(\tau,\theta))] - c * H(\pi(\cdot|s)) \quad (7)$$

Where $H(\pi(\cdot|s))$ is the entropy of the probability distribution of the policy $\pi(\cdot|s)$ for a state s, and c is a parameter that controls the weight of the entropy penalty. The loss function is optimized using a stochastic gradient descent method.

## 5. Methodology

The "grid2op" framework was used to simulate the control environment [Marot et al. 2021], the Python programming language was used to build the agents, and the free Google Colab environment was used as computational enviroment for the simulations. All algorithms underwent parameter tuning to determine which architecture was best suited to the problem. Each algorithm was exposed to a set of different randomly selected architectures and trained in the same environment with random data sets. The reward functions for training neural network based algorithms were presented in Section 3.2.

Table 2 illustrates the configuration of the architectures employed. In this representation, each "DENSE" parameter signifies a hidden layer of the neural network that is densely interconnected with another layer, as denoted by the sequential number following the term "dense." In cases where a layer does not contain any neurons, it is denoted as "NONE" within the table.

The set of parameters for DQN differs essentially in the network architecture, with 3 different sets of neurons described by the "DENSE" parameter (Table 2), while all share the same set of hyperparame- ters, such as a learning rate of 0.001, a memory size of 210 with a microbatch of 70, a discount factor γ of 0.95, and a randomness decay rate of 0.995. All architectures were tested with 4 different types of epochs, ranging from 500 to 2000 with a difference of 500 between them. The neural network update also had a total of 3 learning epochs. Theactivations between the dense layers are "ReLu" and the activation in the output layer is "Linear".

**Table 2. Number of neurons in each layer of each tested architecture**

| Algorithm | DENSE 1 | DENSE 2 | DENSE 3 |
|---|---|---|---|
| DQN | 50 | 50 | NONE |
| DQN | 320 | 150 | NONE |
| DQN | 530 | NONE | NONE |
| DDQN | 500 | 500 | NONE |
| DDQN | 1000 | 750 | 800 |
| DDQN | 8050 | NONE | NONE |
| PPO | 64 | 64 | NONE |
| PPO | 189 | 189 | 189 |
| PPO | 550 | NONE | NONE |

The set of parameters for DDQN also differs in the network architecture described by "DENSE" (Table 2), with the hyperparameters being the same among the 3 tested. The discount factor γ remains at 0.95, however, the decay rate of the randomness has increased to 0.9999, resulting in the need to increase the number of training epochs to 20000. Each architec- ture was tested in 4 different epochs, starting from 5000 and going up to 20000 in in- crements of 5000. The value of t (Equation 3) is 0.1 and the number of training epochs for each dataset was equal to 3. The activation functions in the dense layers were "ReLu", and in the output layer, it was "Linear".

DQN and DDQN were limited to a maximum of 200 episodes. The PPO algorithm followed the same method, changing only the neural network architectures (Table 1) while keeping the same set of parameters. The learning rate was set to 0.003, c (equation 7) for the policy network was 0.5, while for the value network it was 0.01, and ε (equation 6) was set to 0.2. Gamma had a value of 0.99. The activation functions for the dense layers were "ReLU", and for the policy network output layer, it was "Softmax". Overall, they were trained on three different epochs, ranging from 250 to 1000.

Only two architectures were tested for the ARS algorithm. The first experiment had a learning rate of 0.003 and 50 perturbations, with only two updating the target matrix. There were 1 episodes tested per matrix. A learning rate of 0.001, 16 perturbations, 4 updating the target matrix, and 3 episodes tested per matrix were found for the second architecture.

After defining the parameter tuning architectures, the algorithms were trained in the "l2rpn_case14_sandbox" environment. The trained models were then tested on the first 20 episodes of seed 0 for each environment. The tests collected data on the number of steps taken, cost per action based on Equation 1, and the percentage of time when at least one power line was at 90% or more of its thermal limit.

The best architecture for each algorithm was chosen based on the average number of steps taken. The algorithms with the highest average were selected. Only the best

algorithms from the first scenario were tested in the "l2rpn_wcci_2020" scenario. A training was conducted for this scenario, followed by a test on the first 20 episodes of seed 0, collecting the same information as in the first scenario.

Using the selected best algorithms and the collected data, a comparison was made in terms of training speed, solution quality (ability to keep all lines below 90% thermal limit), amount of time control is performed (information and result processing delay), and the number of steps taken in each environment.

**Table 3. Result of the best architectures**

| Algo-rithm | Dense 1 | Dense 2 | Dense 3 | lr | Epochs |
|---|---|---|---|---|---|
| DQN | 50 | 50 | NONE | 0.001 | 2000 |
| DDQN | 1000 | 750 | 800 | 0.001 | 20.000 |
| PPO | 189 | 189 | 189 | 0.003 | 250 |

Table 3 presents the results of the best neural network-based architectures which are described by the "DENSE" parameters in Table 3, where each "DENSE" number represents the quantity of neurons per layer that are densely connected. The "lr" column indicates the learning rate of each algorithm, and the "epochs" column rep- resents the number of training epochs required to achieve the result. For the ARS algo- rithm, the best architecture consists of 50 perturbations, 15 update perturbations, a learning rate of 0.003, and 8 training epochs.

## 6. Results and Discussions

Table 4 presents the training time for each algorithm, with ARS being up to 45% faster than DDQN, 72% faster than DQN, and 69% faster than PPO.

**Table 4. Training time of the best architectures**

| Algorithm | Time (minutes) |
|---|---|
| ARS | 293.38 |
| DQN | 1048.68 |
| DDQN | 537.025 |
| PPO | 886.18 |

The result of the amount of time the algorithm stayed above 80% of its thermal limit (Figure 1) demonstrates that ARS and DQN were more efficient in scenario 1, achieving the same results, while in scenario 2 ARS also achieved maintain network balance. The advantage is that the ARS took much less time to be trained (Table 4) and in scenario 2 it achieved the best balance between number of steps taken (Figure 3) and voltage line control, outperforming all other algorithms.

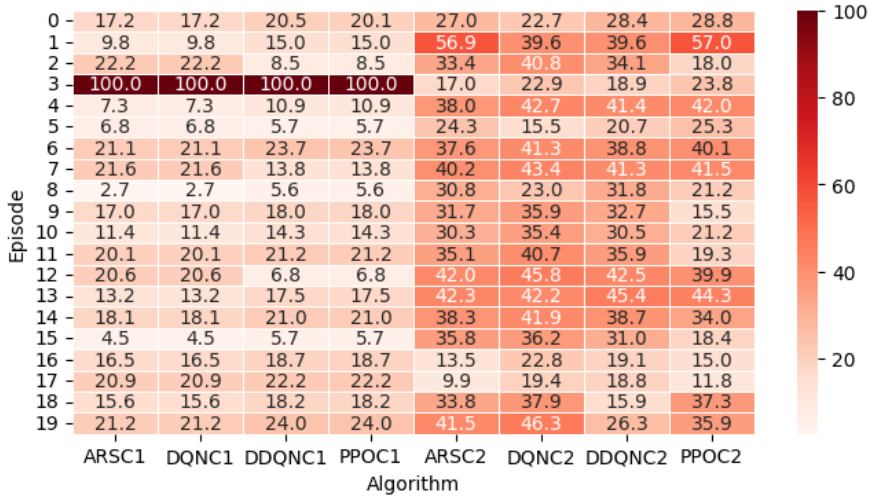


**Figure 1. Heatmap indicating the percentage of steps that each algorithm was above 80%of the thermal threshold. The "C1" at the end of each caption reads l2rpn_case14_sandbox and the "C2" reads l2rpn_wcci_2020**

Figure 1 also illustrates the effectiveness of each algorithm in managing individual episodes. The side bar indicates the duration for which the algorithm maintained at least one voltage line above 80%, with shades of red indicating varying levels of performance. The darker the red, the worse the performance. This enhancement provides a clearer perspective on the individual outcomes.

Figures 2 and 3 depict the average operational cost incurred by each algorithm while executing agents within the environments. Figure 2 specifically illustrates the average cost for each algorithm in the L2RPN_CASE14_SANDBOX environment. Across all algorithms, there is a nearly uniform average cost per execution. A slight increase is observed in Episode 2 for the DDQN and PPO algorithms. This phenomenon highlights the competitive operational cost of ARS within this environment as well.

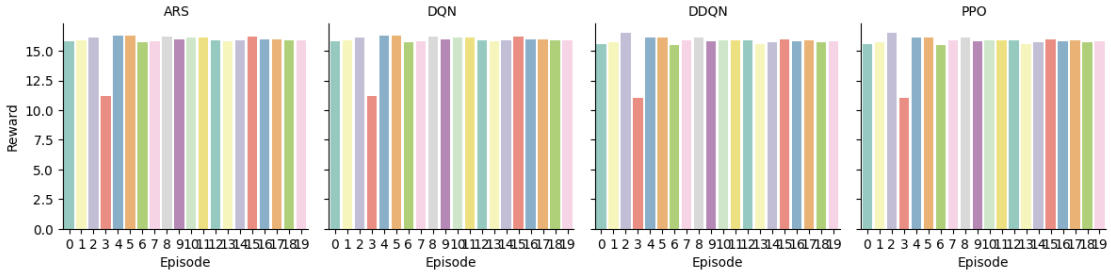


**Figure 2. Graph of average cost of action achieved by each algorithm in the environment L2RPN_CASE14_SANDBOX.**

Figure 3 portrays the costs associated with the L2RPN_WCCI_2020 environment. Notably, there is a significant increase in the overall average cost compared to the agents operating in the L2RPN_CASE14_SANDBOX environment. This divergence can likely be attributed to various factors, although the most plausible explanation involves alterations in the parameter "p(t)" of Equation 1 in Section 3.2. This parameter is dynamic and represents costs, defined within the backend of the application"p(t)" of Equation 1 in Section 3.2. This parameter is dynamic and represents costs, defined within the backend of the application.

In general, the algorithms maintain a consistent cost profile, with a slight decrease observed in Episode 5 for the ARS algorithm. Conversely, there is a marginal increase in costs during episodes 17 and 19 for the DDQN algorithm. The ARS algorithm also demonstrated strong competitiveness in comparison to the other algorithms, remaining within a similar cost range and even reducing operational costs in certain episodes.

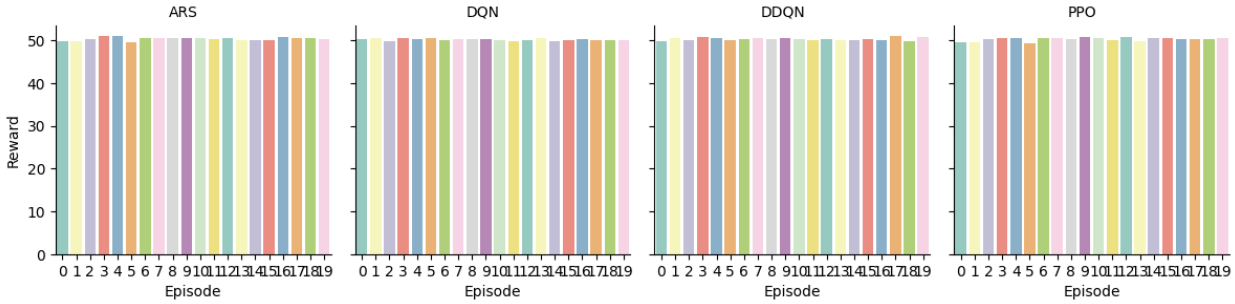


**Figure 3. Graph of average cost of action achieved by each algorithm in the environment L2RPN_WCCI_2020.**

ARS outperformed all algorithms in terms of the number of steps taken (Figure 4) in the "L2RPN_WCCI_2020" scenario, showing a significant lead in most episodes. The graphs of the ARS algorithm display a greater frequency of peaks and, on the whole, maintain higher average values. In the "L2RPN_CASE14_SANDBOX" scenario, both ARS and DQN achieved identical results. This demonstrates the robustness of ARS across different scenarios without requiring substantial hyperparameter adjustments.

Monitoring the progression of steps taken by the algorithms is particularly crucial to assess how effectively they balanced the overall policy of actions. As steps taken indicate the distance each algorithm managed to cover in each episode, they signify a longer duration of continuous control, ensuring a more prolonged network equilibrium.

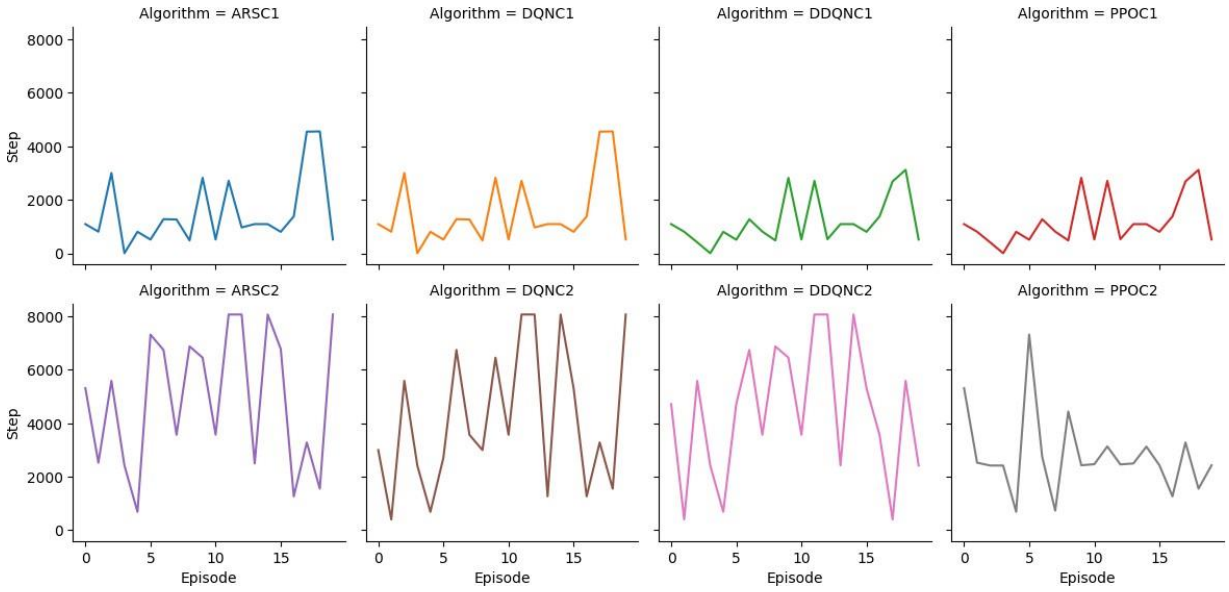


**Figure 4. Graph of the number of steps taken in each episode. The "C1" at the endof each caption reads L2RPN_CASE14_SANDBOX and the "C2" reads L2RPN_WCCI_2020**

One of the possible reasons for this difference is that the ARS, being simpler, is not as sensitive to changes in the environment when compared to neural network algorithms, which can also be understood as an extremely positive point for the ARS.

Finally, the last evaluative metric is the time each algorithm takes to process the data and deliver a response. In this case, the processing speed of a response is im- portant, as energy grid control requires a fast response rate that can keep up with the constant changes within the system.

**Table 5. Average processing time in milliseconds and memory usage in KB per processing**

| Algorithm | Time (ms) | Memory (KB) |
|---|---|---|
| ARS | 15 | 179.12 |
| DQN | 168 | 280.50 |
| DDQN | 42 | 282.99 |
| PPO | 96 | 281.35 |

ARS is the most efficient in terms of processing speed (Table 5) due to its simple architecture and easy implementation, unlike neural networks that require complex matrix calculations. This simplicity makes ARS easy to implement in simpler controls with limited memory available for operations, keeping implementation costs low. Additionally, the memory cost per processing is not cumulative in ARS, unlike in frame- works such as Keras and Pytorch. This feature allows the algorithm to be implemented in smaller controllers with more limited resources, making it a cost-effective solution.

## 7. Conclusion.

The evaluation of reinforcement learning algorithms for grid energy control showed that each algorithm has its strengths and weaknesses. The ARS and DQN algorithms performed similarly in terms of maintaining voltage levels, but the ARS had a faster training time. The PPO and DDQN algorithms were more cost-effective in terms of monetary expenditure; however, their control performance was deficient, experiencing losses in some episodes with a difference of nearly 1000 steps and almost 3% of line overflow time. On the other hand, ARS and DQN tied in all aspects except for training speed and processing speed. This allows the use of ARS in simpler controllers due to its simplicity, speed, and memory cost.

The ARS algorithm proved to be competitive compared to the other tested algorithms, showing significant superiority in terms of training and processing speed. This makes it more feasible for implementation, especially considering that complex neural networks can pose challenges in industrial applications where simple controllers may lack the required hardware support for their execution.

As suggestions for future work is to create an agent that combines taking no action with taking some action on the voltage lines. It is also suggested to increase the action space used, such as battery control or generator output control along with voltageline control, and also to combine the ARS algorithm with neural networks.

## 8. Acknowledgment.

Acknowledgments to CAPES for funding this research and to UFPA for its PPGCC program that guides cutting-edge academic research and opens the doors to the world of inquiry for those interested in shaping a better and more sustainable future.

## References

Bhalshankar, S. S. and Thorat, C. S. (oct 2016). Integration of smart grid with renewable energy for energy demand management: Puducherry case study. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. . IEEE. http://ieeexplore.ieee.org/document/7955498/.

Bi, W., Shu, Y., Dong, W. and Yang, Q. (oct 2020). Real-time Energy Management of Microgrid Using Reinforcement Learning. In 2020 19th International Symposium on Distributed Computing and Applications for Business Engineering and Science (DCABES). . IEEE. https://ieeexplore.ieee.org/document/9277821/.

Bollenbacher, J. and Rhein, B. (oct 2017). Optimal configuration and control strategy in a multi-carrier-energy system using reinforcement learning methods. In 2017 International Energy and Sustainability Conference (IESC). . IEEE. http://ieeexplore.ieee.org/document/8167476/.

Donnot, B., Guyon, I., Schoenauer, M., Panciatici, P. and Marot, A. (2017). Introducing machine learning for power system operation support.

Fan, J., Wang, Z., Xie, Y. and Yang, Z. (2019). A Theoretical Analysis of Deep Q-Learning.

Flick, T. and Morehouse, J. (2011). Securing the smart grid: next generation power grid security. Amsterdam ; Boston: Syngress.

Gao, Wei, Fan, R., Huang, R., et al. (mar 2023). Augmented random search based inter-area oscillation damping using high voltage DC transmission. Electric Power Systems Research, v. 216, p. 109063.

He, Y., Wu, S., Liang, Y., et al. (23 dec 2021). National Energy Demand And Carbon Emission Forecast Under The “Carbon peak and Carbon neutrality” Target Based On System Dynamic. In 2021 IEEE Sustainable Power and Energy Conference (iSPEC). IEEE. https://ieeexplore.ieee.org/document/9735833/.

Hu, D., Peng, Y., Yang, J., Deng, Q. and Cai, T. (8 dec 2021). Deep Reinforcement Learning Based Coordinated Voltage Control in Smart Distribution Network. In 2021 International Conference on Power System Technology (POWERCON). IEEE. https://ieeexplore.ieee.org/document/9697762/.

Kofinas, P., Dounis, A. I. and Vouros, G. A. (jun 2018). Fuzzy Q-Learning for multi-agent decentralized energy management in microgrids. Applied Energy, v. 219, p. 53–67.

Lan, T., Duan, J., Zhang, B., et al. (2019). AI-Based Autonomous Line Flow Control via Topology Adjustment for Maximizing Time-Series ATCs.

Liul, J., Xul, W., Liul, Z., et al. (1 nov 2022). Autonomous Decentralized Control of Distributed Generation using Multi-Agent Reinforcement Learning. In 2022 IEEE PES Innovative Smart Grid Technologies - Asia (ISGT Asia). IEEE. https://ieeexplore.ieee.org/document/10003595.

Marot, A., Donnot, B., Dulac-Arnold, G., et al. (2021). Learning to run a Power Network Challenge: a Retrospective Analysis.

Pratt, R. G. (2004). Transforming the U.S. electricity system. In IEEE PES Power Systems Conference and Exposition, 2004. .IEEE. http://ieeexplore.ieee.org/document/1397713/.

Quakernack, L., Kelker, M. and Haubrock, J. (10 oct 2022). Deep Reinforcement Learning For Autonomous Control Of Low Voltage Grids With Focus On Grid Stability In Future Power Grids. In 2022 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe). . IEEE. https://ieeexplore.ieee.org/document/9960416/.

Rocchetta, R., Bellani, L., Compare, M., Zio, E. and Patelli, E. (may 2019). A reinforcement learning framework for optimal operation and maintenance of powergrids. Applied Energy, v. 241, p. 291–301.

Roderick, M., MacGlashan, J. and Tellex, S. (2017). Implementing the Deep Q- Network.

Sukumar, S. R., Shankar, M., Olama, M., et al. (sep 2010). A methodology to consider combined electrical infrastructure and real-time power-flow impact costs in planning large-scale renewable energy farms. In 2010 IEEE Energy Conversion Congress and Exposition. . IEEE. https://ieeexplore.ieee.org/document/5617942/.

Sun, Q., Wang, D., Ma, D. and Huang, B. (nov 2017). Multi-objective energy management for we-energy in Energy Internet using reinforcement learning. In 2017 IEEE Symposium Series on Computational Intelligence (SSCI). . IEEE. http://ieeexplore.ieee.org/document/8285243/.

Van de Wiele, T., Warde-Farley, D., Mnih, A. and Mnih, V. (2020). Q-Learning in enormous action spaces via amortized approximate maximization.

Van Hasselt, H., Guez, A. and Silver, D. (2 mar 2016). Deep Reinforcement Learning with Double Q-Learning. Proceedings of the AAAI Conference on Artificial Intelligence, v. 30, n. 1.

Watkins, C. J. C. H. and Dayan, P. (may 1992). Q-learning. Machine Learning, v. 8, n. 3–4, p. 279–292.

Yang, W., Zhou, X. and Xue, F. (mar 2010). Impacts of Large Scale and High Voltage Level Photovoltaic Penetration on the Security and Stability of Power System. In 2010 Asia-Pacific Power and Energy Engineering Conference. IEEE. https://ieeexplore.ieee.org/document/5448930/.