Multi-Agent Reinforcement Learning for Power Grid Topology Optimization

Erica van der Sar, Alessandro Zocca, Sandjai Bhulai
Department of Mathematics, Vrije Universiteit, Amsterdam, The Netherlands
{e.t.van.der.sar, a.zocca, s.bhulai}@vu.nl

***Abstract*—Recent challenges in operating power networks arise from increasing energy demands and unpredictable renewable sources like wind and solar. While reinforcement learning (RL) shows promise in managing these networks, through topological actions like bus and line switching, efficiently handling large action spaces as networks grow is crucial. This paper presents a hierarchical multi-agent reinforcement learning (MARL) framework tailored for these expansive action spaces, leveraging the power grid's inherent hierarchical nature. Experimental results indicate the MARL framework's competitive performance with single-agent RL methods. We also compare different RL algorithms for lower-level agents alongside different policies for higher-order agents.**

***Index Terms*—Graph neural networks, Multi-agent reinforcement learning, Power grid reliability, Topology optimization, Transmission networks.**

I. Introduction

In 2019, the French TSO RTE launched the *Learning to Run a Power Network* (L2RPN) challenge [1], encouraging diverse researchers to use reinforcement learning (RL) for power network maintenance. While various solutions emerged, see [2]–[4] and more recently [5], [6], all faced challenges with the vast combinatorial action space, including line-switching and bus-splitting. This underscores the need for an advanced RL framework ensuring scalability and easy integration of actions like generation redispatch.

Multi-Agent Reinforcement Learning (MARL) seems a well-suited approach to address these challenges due to their scalability. However, its potential in this domain remains underexplored. A recent paper by [7] introduced a hierarchical reinforcement learning (HRL) framework where RL agents have roles based on their operational level. Still, a primary agent determines the best topological action.

A recent paper [2] utilizes a hierarchical framework that activates a soft actor-critic (SAC) agent [8] only when issues related to grid safety arise and uses graph attention layers to learn the intricate dependencies within the power network. While this SAC approach keeps a small action space due to its after-state implementation, its adaptability to network shifts is limited. To address this limitation, we transitioned to a soft actor-critic discrete (SACD) agent [9] that learns directly from actions rather than the after-state.

Although this adjustment does lead to a larger action space, we distribute tasks to multiple RL agents per substation instead of one global agent. This change brings benefits like reduced action space per agent, enhanced scalability, and easier integration of actions like generator redispatching, forming a cooperative MARL framework [10].

The simplest *independent learning* MARL treats each agent independently and considers the rest of the agents as part of the environment [11]. This approach avoids scalability and communication issues, however, the non-stationarity of the environment from each agent's perspective slows and possibly hinders learning. We use the *centralized training with decentralized execution* strategy [12], allowing agents to access additional information during training but not during execution.

This paper introduces various MARL strategies for power grid control, utilizing the problem's inherent hierarchical structure. In addition to SACD, we use another state-of-the-art RL algorithm, proximal policy optimization (PPO) [13].

The paper is organized as follows: Section II discusses Grid2Op and previous solutions on L2RPN; Section III details our RL methods; Section IV presents our findings; and Section V concludes and outlines future work.

II. Background and Related work

A. Controlling a power network: the Grid2Op environment

This section introduces Grid2Op [14], an open-source framework used for the *Learning to Run a Power Network* (L2RPN) competitions [1], [15], [16]. Designed for sequential decision-making in power systems, its aim is to ensure a safe power network by avoiding contingencies and ensuring constant connectivity. Grid2Op simulates real-world power grids, depicting them as graphs with nodes as substations and edges as transmission lines or transformers. Each substation connects various elements like generators, loads, and storage units, and connects to one of two *buses*. A consistent bus connection makes a substation a singular node; differing connections 'split' it, changing the local network topology. Grid2Op can compute the resulting power flow configurations from any topology change; see an example in Fig. 1. Modifications in a substation's bus configuration are termed *bus-switching actions*. Grid2Op simulates other actions like *line-switching* and *power redispatch actions*. Cost-effective for responding to contingencies, these *topological actions* redistribute line flows, mitigating issues without the need for pricier generators or load adjustments.

In this paper, we focus solely on bus-switching actions, and we use the realistic Grid2Op environment to develop a sequential decision-making RL model, termed *agent*, that keeps the power grid in a safe regime by choosing optimal bus-switching actions.

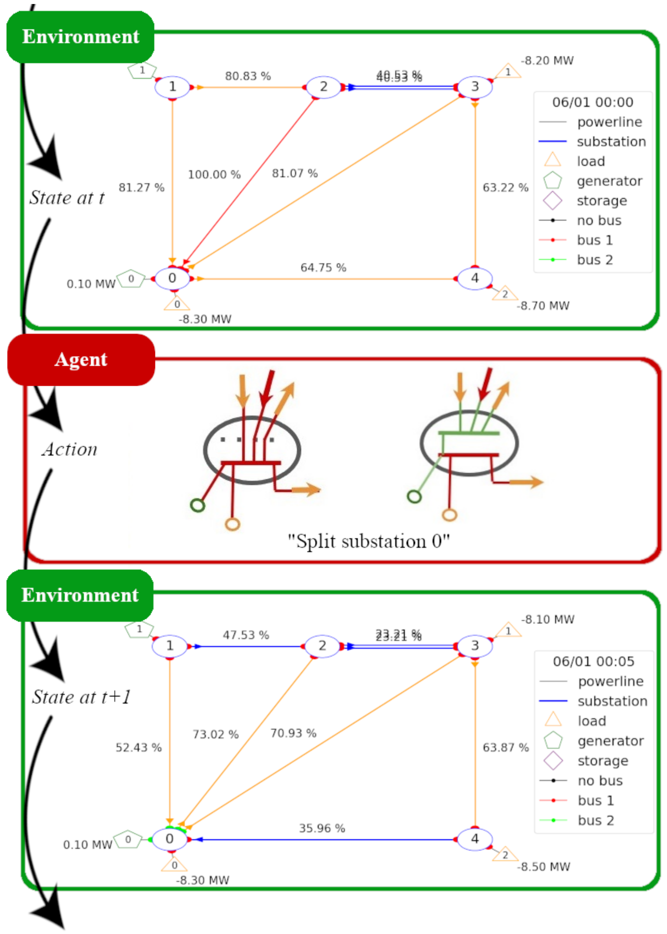

Figure 1: Example of a bus-switching action by an agent and the effect on the environment. Some elements of substation 0 are assigned to bus 2, which results in a split of substation 0, and a consequent power flow redistribution.

B. Previous solution approaches for L2RPN

The winning approach of L2RPN WCCI 2020 was the Semi-Markov Afterstate Actor-Critic (SMAAC) proposed by [2]. It acts only in dangerous states, transitioning from a Markov to a semi-Markov decision process. Instead of learning specific actions, SMAAC learns the optimal bus substation configuration, termed *goal topology*. Given that only one substation can be acted on at once in L2RPN, determining the action sequence is crucial. The authors of [2] used a two-tiered system: the “high-level” SAC-based policy sets the goal, while the “low-level” policy defines the sequence, with four rule-based options tested for the latter. SMAAC’s goal topology ensures the SAC algorithm learns fewer values. Yet, this leads to an agent that is largely unresponsive to environmental shifts. It tends to stick to one effective configuration without adapting as needed. Small actor output changes often do not influence the action unless they surpass a set threshold, complicating course correction during gradient descent.

Another recent paper [7] employs a three-tiered structure for optimal power network topology control. Like [2], the top level is rule-based. The intermediate-level agent selects the substation and utilizes PPO or SAC-trained policies. The lowest level uses either a brute-force method examining all configurations or an RL agent with an action mask aligned with the intermediate-level agent’s choice.

III. Methods

In this section, we provide details about the RL environment we used and our proposed RL architectures and algorithms.

A. Reinforcement learning environment: Grid2Op

The Grid2Op environment described in Section II-A is typically described as a Markov Decision Process (MDP) defined by $(\mathcal{S}, \mathcal{A}, p, r)$, where at each time step t an agent observes a state $s_t \in \mathcal{S}$ from the environment and takes an action $a_t \in \mathcal{A}$. The environment returns the next state $s_{t+1} \in \mathcal{S}$ to the agent with probability $p(s_{t+1}|s_t, a_t)$, which is the unknown state transition probability, and the agent receives an immediate reward $r(s_t, a_t) \in \mathbb{R}$. With this MDP formulation, reinforcement learning (RL) can be used to learn a (stochastic) policy $\pi(a_t|s_t)$ that optimizes the expected discounted reward $\mathbb{E}_\pi[\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$, where $\gamma \in (0, 1)$ is the discount factor. Within the Grid2Op environment, the state s_t that the agent can observe at time t consists of (i) the current generator and load states, including power production/consumption; (ii) the current topology configuration, comprehensive of line connections and current bus configuration at each substation; and (iii) the current load $\rho_\ell \in [0, 1]$ on each line ℓ, measured as the fraction of the capacity of that power line.

The agent must choose a bus-switching action a_t from the fixed collection $\mathcal{A}$ of substation reconfigurations. These always include the trivial action, which we name *do-nothing action*, that does not change the current bus configuration. As mentioned in Section II, the objective is to keep the power network in a *safe* state cost-effectively throughout each episode. Congested lines auto-disconnect based on overload severity and have a reconnection delay. Any network disconnection or isolation of a load or generator results in a “game over” with a reward of -1.

To steer the agent towards learning how to maintain a safe power grid, it is important to have a proper reward function that penalizes whenever power lines are in overload. When lines are congested, it leads to an increase in energy loss within the network. Therefore, during training, we use the energy efficiency of the power grid as a reward function defined by rescaling the ratio of the total served load and the total generation, similarly to [2].

B. Reinforcement learning algorithms

Next, we introduce two RL algorithms: a discrete Soft Actor-Critic variant (SACD) and Proximal Policy Optimization (PPO). In both algorithms, the policy, indicated by π, will be parameterized using a neural network with learnable parameters θ, while the action-value function, denoted by $Q(s_t, a_t)$, and for PPO the state-value function $V(s_t)$ will be represented by a neural network with parameters ϕ.

1) Soft actor-critic discrete (SACD): Soft Actor-Critic (SAC) [8] is an off-policy reinforcement learning algorithm aiming for efficient and stable learning in continuous action spaces. The algorithm incorporates an entropy term into its reward function, effectively encouraging more exploratory behavior. This leads to improved state-space exploration and provides a degree of robustness in policy learning. The core advantage of SAC is its ability to balance exploration and exploitation efficiently. Soft actor-critic discrete (SACD) is an adjustment of SAC to make the algorithm applicable to discrete environments [9].
For the critic, we learn a soft q-function $Q_\phi(s,a)$ via off-policy temporal-difference learning by minimizing the critic-loss:

$$J_Q(\phi) = \mathbb{E}_{(s_t,a_t)\sim D}\Big[\frac{1}{2}\big(Q_\phi(s_t,a_t) - y(s_t,a_t)\big)^2\Big] \qquad (1)$$

$$\text{with } y(s_t,a_t) = r_t + \gamma\mathbb{E}[V_{\bar{\phi}}(s_{t+1})],$$

where D is a replay buffer of past experiences and $\bar{\phi}$ is the parameter for the target critic network, and the soft state value calculation is defined as

$$V_{\bar{\phi}}(s_{t+1}) = \pi_\theta(s_{t+1})^T\left[Q(s_{t+1}) - \alpha\log\left(\pi_\theta(s_{t+1})\right)\right],$$

where α determines the relative importance of the entropy term versus the reward and is called the temperature parameter.
The objective function of the policy, the actor-loss, is:

$$J_\pi(\theta) = \mathbb{E}_{s_t\sim D}\Big[\pi(s_t)\big[\alpha\log\left(\pi_\theta(s_t)\right) - Q_\phi(s_t)\big]\Big]. \qquad (2)$$

The objective function for the temperature parameter α is

$$J(\alpha) = \pi_\theta(s_t)^T\Big[-\alpha\Big(\log\left(\pi_\theta(s_t)\right) + \bar{H}\Big)\Big],$$

with $\bar{H}$ a constant vector equal to the hyperparameter representing the target entropy.
2) Proximal Policy Optimization (PPO): Proximal Policy Optimization (PPO) [13] is a policy optimization algorithm that improves upon traditional policy gradient methods by introducing a clipped surrogate objective function. This ensures that policy updates remain close to the original policy, effectively balancing the trade-off between exploration and exploitation.
For PPO, the policy loss is computed as

$$L_t^{\text{CLIP}}(\theta) = \mathbb{E}\Big[\min\Big(r_t(\theta)A_t, \text{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)A_t\Big)\Big],$$

with ϵ a clipping hyperparameter, $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\bar{\theta}}(a_t|s_t)$ denotes the probability ratio with $\bar{\theta}$ the parameters of the current policy, and A_t is the generalized advantage estimation (GAE) function defined as

$$A_t = \sum_{l=0}^{h}(\gamma\lambda)^l\left[r_{t+l} + \gamma V_\phi(s_{t+l+1}) - V_\phi(s_{t+l})\right],$$

where h is the number of time steps collected in each iteration.
The critic-loss is again a squared error loss

$$L_t^{\text{VF}}(\phi) = \left(V_\phi(s_t) - V_t^{\text{targ}}\right)^2 = \Big(V_\phi(s_t) - \big(A_t + V_{\bar{\phi}}(s_t)\big)\Big)^2.$$

Combining both losses, the following objective is obtained, which is minimized in each iteration:

$$L_t = \mathbb{E}_t\Big[-L_t^{\text{CLIP}}(\theta) + c_1 L_t^{\text{VF}}(\phi) - c_2 S\big[\pi_\theta(s_t)\big]\Big],$$

where c_1, c_2 are coefficients, and S denotes an entropy bonus.

C. Hierarchical architecture

Building on suggestions from [2] and [7], power network control naturally fits a hierarchical scheme due to the tiered nature of topological actions. Decisions range from determining if an action is needed, selecting substations to act upon, to finalizing their busbar configuration. We introduce a three-level hierarchical reinforcement learning framework, detailed subsequently.
1) Highest level: At the highest level, a single agent determines whether to act at each time step, often doing nothing in safe environments. We use a rule-based method, similar to prior L2RPN implementations. The agent checks for any critically loaded line, with a load ρ exceeding a threshold ρ_{thresh}. When the environment is *unsafe*, that is $\max_\ell \rho_\ell > \rho_{\text{thresh}}$, the highest-level agent activates the mid-level one. It remains passive while lower agents work, evaluating environment safety after their actions conclude.
2) Mid level: The intermediate level has a single agent that, upon activation, selects the substations requiring action. Since network operators typically prefer to intervene at a single substation at a time, the mid-level agent is also tasked with establishing the order in which the selected substations should act. Taking a rule-based approach, we apply the CAPA policy from [2], known for its effectiveness in the context of goal topology. This policy prioritizes substations with higher line loads. The mid-level agent then sequentially activates the low-level agents based on the chosen order.
3) Lowest level: The lowest level features *substation-specific agents*, each responsible for selecting bus assignments for their substation's elements. These agents have a predefined discrete action space, further reduced by excluding symmetric actions. Unlike rule-based higher-level agents, these agents use reinforcement learning to determine optimal policies, aligning with the Multi-Agent Reinforcement Learning (MARL) framework. Our unique approach of multiple substation-specific agents for topology decisions distinguishes our method from prior hierarchical strategies. The subsequent section delves into the MARL architectures we explore.

D. Multi-agent reinforcement learning (MARL)

As mentioned in Section III-C3, for the lowest level agent, we use a Multi-Agent Reinforcement Learning (MARL) framework where multiple agents function in a shared environment.
We explored two MARL variants with independent and dependent agents. Independent agents each learn their own policy, viewing others as part of the environment. This makes the environment non-stationary for each agent due to others' learning, eliminating convergence assurances. However, independent agent structures have empirically shown effective performance. In contrast, dependent agents, collaborating and sharing information, can produce more coherent and effective multi-agent policies. Yet, handling inter-agent dependencies can increase computational costs and scaling difficulties. Hence, for the dependent agents, we embrace the Centralized Training with Decentralized Execution (CTDE) paradigm, a favored approach in cooperative MARL [17], [18]. In this paradigm, agents train

using global state information for collective learning, but make decisions via decentralized local policies.

We evaluate two primary model-free RL algorithms, Proximal Policy Optimization (PPO) and the Soft Actor-Critic with a discrete action space (SACD) from Section III-B, resulting in four distinct MARL strategies:

1) Independent agents SACD (ISACD);
2) Independent agents PPO (IPPO);
3) Dependent agents SACD (DSACD);
4) Dependent agents PPO (DPPO).

In the next subsections, we elaborate on each strategy. Regardless of the chosen approach, the mid-level agent's activation sets a sequence for the low-level agents. Although the sequence persists even if a low-level agent takes no action, we omit such actions to save time and reduce "game over" risks.

1) Independent SACD (ISACD): In the independent MARL version, each agent has its own critic and actor, which are updated based on their own state-action pairs. For each agent i, we update its state-action value-function Q^i_ϕ and its policy π^i_θ using (1) and (2) respectively, based on the transitions stored in D^i. Thus, to compute the critic-loss for agent i, we have:

$$J_{Q^i}(\phi) = \mathbb{E}_{(s_t,a_t)\sim D^i}\left[\frac{1}{2}\big(Q^i_\phi(s_t,a_t) - y(s_t,a_t)\big)^2\right] \quad (3)$$

with

$$y(s_t,a_t) = r_t + \gamma\mathbb{E}\big[V^i_{\bar\phi}(s_{t+1})\big],$$

where the soft state-value function of the next state is calculated based on the agents' i own policy and critic:

$$V^i_{\bar\phi}(s_{t+1}) = \pi^i_\theta(s_{t+1})^T\Big[Q^i(s_{t+1}) - \alpha\log\big(\pi^i_\theta(s_{t+1})\big)\Big].$$

2) Independent PPO (IPPO): Similarly to the independent MARL version for SACD, in IPPO, the policy and critic-loss for each agent i are calculated based on their own replay buffer D^i. For each agent i we, will update the policy loss with

$$L^{\text{CLIP}}_t(\theta^i) = \mathbb{E}\Big[\min\Big(r_t(\theta^i)A^i_t, \text{clip}\big(r_t(\theta^i), 1-\epsilon, 1+\epsilon\big)A^i_t\Big)\Big],$$

where $r_t(\theta^i) = \pi_{\theta^i}(a_t|s_t)/\pi_{\bar\theta^i}(a_t|s_t)$ and the GAE becomes

$$A^i_t = \sum_{l=0}^{h}(\gamma\lambda)^l\delta^i_{t+l} \text{ with } \delta^i_t = r^i_t + \gamma V^i(s_{t+1}) - V^i(s_t). \quad (4)$$

The critic-loss and combined loss are computed per agent i.

3) Dependent SACD (DSACD): The independent learning method, while simple, often struggles with non-stationarity and may not achieve the optimal policy due to limited information sharing during training. A proposed solution is using a centralized critic, incorporating all agents' actions $Q(s, a^1, \ldots, a^n)$. However, this approach risks the curse of dimensionality since we can have $\prod_{i=1}^n |A^i|$ number of action combinations.

The L2RPN challenge environment's structure ensures agents do not act simultaneously within a time step. As detailed in Section III-C2, lower-level agents activate sequentially based on substation loads. There is a certain probability p_{ij} that the lower-level agent j acts after agent i, based on the load distribution in this substation. These distributions are not known a priori, so they are estimated during the training phase. These probabilities p_{ij} are stored in matrix $\hat\pi_{\text{mid}}$, which is used to update the value functions of the critics, making the update of agent i dependent on the current value function of the other agents. We still use (3) to compute the critic-loss for agent i, but modify the expression y into $y(s_t,a_t) = r_t + \gamma\mathbb{E}[\hat V(s_{t+1})]$. Here, $\hat V(s_{t+1})$ is the average soft state-value function accounting for the likelihood of each agent to act after agent i, i.e.,

$$\hat V(s_{t+1}) = (\hat\pi^i_{\text{mid}})^T\begin{bmatrix}V^1(s_{t+1})\\ \vdots\\ V^n(s_{t+1})\end{bmatrix},$$

where $\hat\pi^i_{\text{mid}} = [p_{i1}\ldots p_{in}]^\top$ is a vector whose j-th entry is the empirical probabilities of activating agent j after agent i using the current mid-level agents' policy π_{mid}.

4) Dependent PPO (DPPO): Like the DSACD agent, the DPPO agent will use the probability matrix $\hat\pi_{\text{mid}}$, but now to update the GAE. All equations remain the same, except for the temporal difference in (4), which changes to

$$\delta^i_t = r^i_t + \gamma(\hat\pi^i_{\text{mid}})^T\begin{bmatrix}V^1(s_{t+1})\\ \vdots\\ V^n(s_{t+1})\end{bmatrix} - V^i(s_t)$$

to reflect a dependent update for each agent i.

IV. Results

In this section, we present the results obtained using both single-agent and multi-agent architectures in the L2RPN context.

A. Experimental setup

We apply our RL-based design to the *IEEE case 5* environment, a power grid with 5 substations, 8 lines, 2 generators, and 3 loads, shown in Fig. 1. This environment has 20 episodes, or *chronics*, per Grid2Op documentation. Each episode has 2016 time steps, representing 5-minute intervals, showcasing varying demand and supply patterns. For diverse training starts, episodes are divided into five overlapping sub-episodes of 864 time steps, equaling three days. We allocated 18 episodes for training, one for testing, and one for validation.

Each of the four RL architectures was trained for 10,000 interactions across five model seeds, counting only steps where the low-level agents were active. For every 100 interactions, agent performance was assessed using test sub-episodes and the L2RPN Codalab competition's rescaled score function. This score differs from the reward function used during training, but it captures the agents' performance well. Using this function, the agent receives a score between -100 and 0 when the agent dies before the Do-Nothing baseline agent would, a score above 0 when it outperforms the baseline, and a score between 80 and 100 if the agent is able to finish the episode.

Substations with less than three connected elements are not as crucial to control, since most actions either disconnect elements or resemble line-switching actions. We limit agents to substations of size larger than 3, resulting in three low-level agents in the MARL setup.

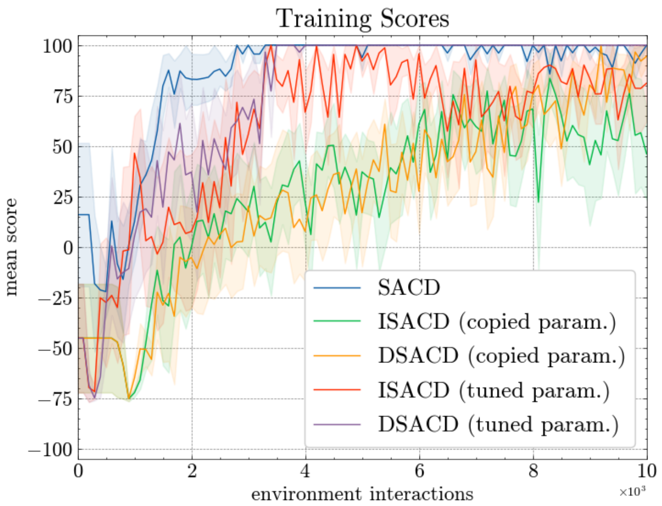

Figure 2: Training progress of the single-agent SACD, and two versions of multi-agents ISACD and DSACD, using the original and tuned parameters, respectively (cf. Table I).

B. Hyperparameters

Both PPO and SACD algorithms have multiple hyperparameters that influence the progress of training. For single-agent algorithms, we used default parameters both for PPO and SACD based on the literature [2], [9], [13]. However, as mentioned in [19], multi-agent RL may require different hyperparameters compared to single-agent RL for optimal performance. Therefore, in the MARL context, we tuned the hyperparameters using Optuna, an open-source hyperparameter optimization framework [20]. Table I summarizes the hyperparameters we used both in the single-agent and multi-agent settings.

Parameters	PPO	MAPPO	SACD	MASACD
(Mini-)Batch size	4 x 32	2 x 32	64	16
Update start			4	3
Discount (γ)	0.95	0.996	0.995	0.998
Learning-rate	0.003	0.002	5×10^{-5}	0.0002
VF coeff. (c_1)	0.5	0.5		
Entropy coeff. (c_2)	0.01	5×10^{-5}		
Clipping param. (ϵ)	0.2	0.12		
GAE param. (λ)	0.95	0.85		
Target entropy scale			0.98	0.98
Tau			0.001	0.002

Table I: Parameters used for the different RL algorithms. The two multi-agent versions of PPO (IPPO and DPPO) use the same parameters, reported in the column MAPPO. Similarly, the column MASACD reports the parameters used for both the multi-agent versions of SACD (ISACD and DSACD).

We did not perform any hyperparameter tuning for the design of the actor and critic networks. For training, we used the Adam optimizer. In both single- and multi-agent SACD, we used 3 GNN blocks in the shared layer with a dimension of 128. For the actor, we use 3 GNN blocks again, and for the critic 1 GNN block. Furthermore, in both the single- and multi-agent PPO we used 3 GNN blocks in both the critic and the actor.

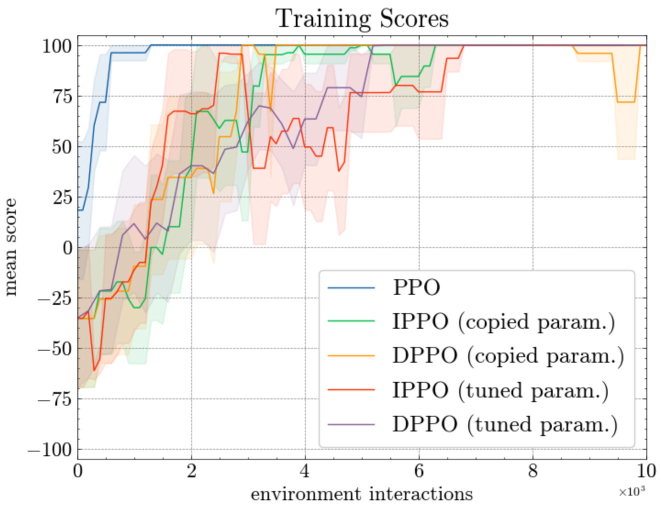

Figure 3: Training progress of the single-agent PPO, and two versions of multi-agents IPPO and DPPO, using the original and tuned parameters, respectively (cf. Table I).

C. Comparison of Single-Agent RL vs. Multi-Agent RL

In this section, we present the main results for single-agent and multi-agent RL architectures. In all plots, we show the mean episode score (solid line) and the corresponding standard error (shaded area) averaged over different model seeds to give insight into the training progress.

1) SACD algorithm: Fig. 2 shows the results of the single- and multi-agent versions of SACD. The single-agent version is able to achieve the optimal score but seems to remain unstable until the end. Fig. 2 also reports the scores of ISACD and DSACD deployed either (i) using hyperparameters identical to those used in the single-agent SACD (cf. the SACD column in Table I) or (ii) using ad-hoc hyperparameters obtained using Optuna. It is clear that hyperparameters optimized for the single-agent SACD version exhibit suboptimal performance when directly applied to their multi-agent counterparts. With optimized parameters, ISACD achieves peak scores but suffers from instability, performing significantly worse than SACD, as expected in Section III-D3. The DSACD agent with tuned parameters achieves an optimal score after a number of environment interactions comparable to those needed by the single agent. However, this agent is much more stable and, in fact, maintains a perfect optimal score until the training ends.

2) PPO algorithm: Fig. 3 illustrates the results for the single- and multi-agent versions of PPO. Even if the single-agent version converges more quickly, all agents are able to find the optimal solution. In terms of hyperparameters, the score difference between multi-agent algorithms with or without optimized parameters is less significant than when using SACD. This can be explained by the fact that the difference between the hyperparameters in PPO and MAPPO is modest and by the fact that PPO is less sensitive to hyperparameter changes. In the single-agent case, PPO outperforms SACD, but looking at the multi-agent setting, they achieve the maximum score

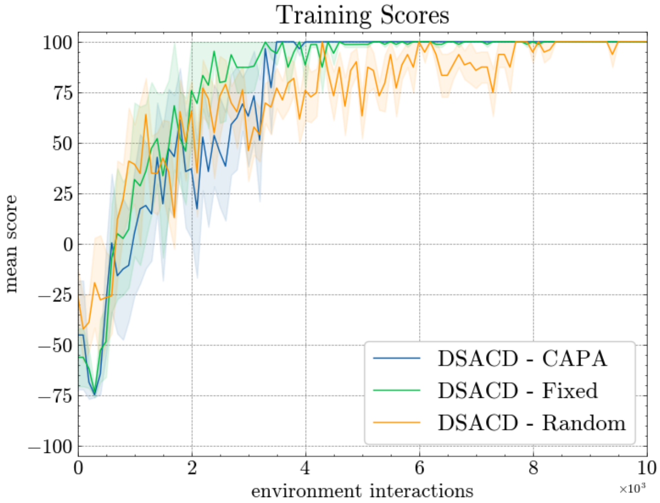

Figure 4: Training progress using different policies for the mid-level agent for DSACD.

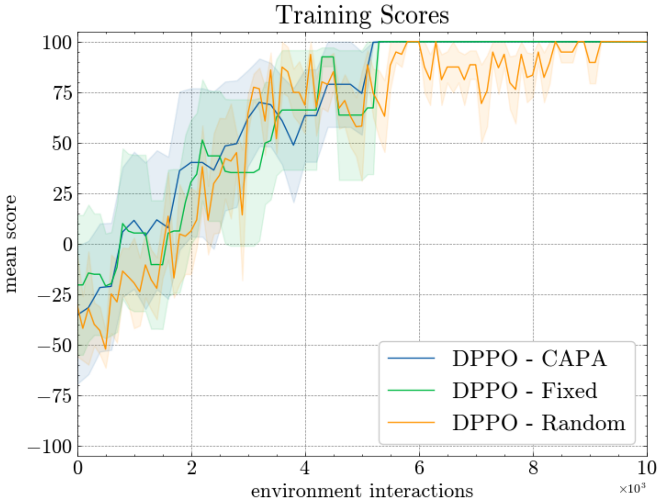

Figure 5: Training progress using different policies for the mid-level agent for DPPO.

after a similar number of environment interactions. Note that the independent version of PPO is not as unstable as ISACD.

D. *Comparing different mid-level policies*

For completeness, we investigate the effects of using other policies for the mid-level agent, which are different from the CAPA policy described in Section III-C2. More specifically, we consider a *fixed policy*, in which the mid-level agent orders the low-level agents based on the size of the corresponding substations, and a *random policy*, in which every time the low-level agents act in random order. The results of these tests are shown in Figs. 4 and 5. The random policy results in more unstable behavior than the other two. The difference between CAPA and the fixed policy is more difficult to appreciate. It makes sense that with random patterns in the order of the agents, the agents have more trouble learning the right actions compared to a more fixed order. We expect that when training both the mid-level and lowest-level agents simultaneously this will become a challenge.

V. Conclusion

This paper presents a hierarchical Multi-Agent Reinforcement Learning (MARL) framework for power network management through topological actions. Our architecture uses rule-based agents at the highest and mid levels, emphasizing varied MARL strategies for the lowest level. Future work will develop learning policies for these agents, addressing inter-level dependencies, and expand experiments to larger networks, further amplifying MARL's advantages over single-agent systems.

References

[1] A. Marot, B. Donnot, C. Romero, B. Donon, M. Lerousseau, L. Veyrin-Forrer, I. Guyon, Learning to run a power network challenge for training topology controllers, Electr. Power Syst. Res. 189 (2020) 106635.

[2] D. Yoon, S. Hong, B.-J. Lee, K.-E. Kim, Winning the l2rpn challenge: Power grid management via semi-Markov afterstate actor-critic, in: International Conference on Learning Representations, 2021.

[3] B. Zhou, H. Zeng, Y. Liu, K. Li, F. Wang, H. Tian, Action set based policy optimization for safe power grid management, in: Machine Learning and Knowledge Discovery in Databases. Applied Data Science Track, Springer International Publishing, 2021, pp. 168–181.

[4] M. Subramanian, J. Viebahn, S. H. Tindemans, B. Donnot, A. Marot, Exploring grid topology reconfiguration using a simple deep reinforcement learning approach, 2021 IEEE Madrid PowerTech (2021) 1–6.

[5] M. Dorfer, A. R. Fuxjäger, K. Kozak, P. M. Blies, M. Wasserer, Power grid congestion management via topology optimization with alphazero, arXiv preprint arXiv:2211.05612 (2022).

[6] A. R. R. Matavalam, K. P. Guddanti, Y. Weng, V. Ajjarapu, Curriculum based reinforcement learning of grid topology controllers to prevent thermal cascading, IEEE Trans. Power Syst. 38 (5) (2023) 4206–4220.

[7] Anonymous, Hierarchical reinforcement learning for power network topology control, Submitted to TMLR, https://openreview.net/forum?id=XgmAz5MSQH (2023).

[8] T. Haarnoja, A. Zhou, P. Abbeel, S. Levine, Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, in: ICML, PMLR, 2018, pp. 1861–1870.

[9] P. Christodoulou, Soft actor-critic for discrete action settings, arXiv preprint arXiv:1910.07207 (2019).

[10] A. Oroojlooy, D. Hajinezhad, A review of cooperative multi-agent deep reinforcement learning, Applied Intelligence 53 (11) (2023) 13677–13722.

[11] M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative agents, in: Proceedings of the tenth international conference on machine learning, 1993, pp. 330–337.

[12] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, Advances in neural information processing systems 30 (2017).

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, arXiv preprint arXiv:1707.06347 (2017).

[14] Grid2op 1.9.4 documentation, https://grid2op.readthedocs.io/en/latest/.

[15] A. Marot, B. Donnot, G. Dulac-Arnold, A. Kelly, A. O'Sullivan, J. Viebahn, M. Awad, I. Guyon, P. Panciatici, C. Romero, Learning to run a power network challenge: a retrospective analysis, in: Proceedings of the NeurIPS 2020, Vol. 133 of PMLR, 2021, pp. 112–132.

[16] A. Marot, B. Donnot, K. Chaouache, A. Kelly, Q. Huang, R.-R. Hossain, J. L. Cremer, Learning to run a power network with trust, Electr. Power Syst. Res. 212 (2022) 108487.

[17] R. Lowe, Y. WU, A. Tamar, J. Harb, O. Pieter Abbeel, I. Mordatch, Multi-agent actor-critic for mixed cooperative-competitive environments, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, 2017.

[18] J. Foerster, G. Farquhar, T. Afouras, N. Nardelli, S. Whiteson, Counterfactual Multi-Agent Policy Gradients, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 32, 2018.

[19] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. Bayen, Y. WU, The surprising effectiveness of PPO in cooperative multi-agent games, in: Advances in Neural Information Processing Systems, Vol. 35, 2022, pp. 24611–24624.

[20] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 2623–2631.