Topology-Aware Reinforcement Learning for Tertiary Voltage Control

Balthazar Donon, François Cubelier
Efthymios Karangelos, Louis Wehenkel
Department of EE & CS - Montefiore Institute
Université de Liège, Belgium
{balthazar.donon, f.cubelier}@uliege.be
{e.karangelos, l.wehenkel}@uliege.be

Laure Crochepierre, Camille Pache
Lucas Saludjian, Patrick Panciatici
Research & Development Department
RTE (Réseau de Transport d'Électricité), France
{laure.crochepierre, camille.pache}@rte-france.com
{lucas.saludjian, patrick.panciatici}@rte-france.com

***Abstract*—Transmission systems have experienced an increase in the occurrence frequency and intensity of high voltage events over the past few years. Since traditional approaches to optimal power flow do not scale well to real-life systems, it has become urgent to develop new methods to help operators improve tertiary voltage control. In this paper, we propose to train a graph neural network to choose voltage setpoints by interacting with a power grid simulator using reinforcement learning techniques. Moreover, we introduce the hyper heterogeneous multi graph formalism to account for topology variations of real-life systems (assets disconnection, bus-splitting, etc.). Our approach is validated on an artificial case study based on the case60nordic power grid.**

***Index Terms*—Graph Neural Networks, Reinforcement Learning, Tertiary Voltage Control**

I. Background & Motivations

In recent years, transmission systems have experienced an increase in the frequency and intensity of over-voltage events [1]. This worrying trend is due to multiple factors, including the increasing share of hard-to-predict renewable energies, new electricity uses and several temporary net load reductions. Voltage is controlled *inter alia* by generators that adjust their reactive power production to match a certain voltage setpoint (primary voltage control). Multiple generators can be pooled together to control the voltage at a so-called "pilot" bus (secondary voltage control). Tertiary voltage control consists in keeping all voltages across the grid in an acceptable range, firstly by choosing appropriate voltage setpoints (as input to a primary or a secondary voltage control), but also by opening transmission lines, connecting and disconnecting shunts, controlling tap changers and starting synchronous condensers. The growing complexity of tertiary voltage control pushes Transmission System Operators (TSOs) to develop real-time decision support tools to assist operators in their daily tasks. For each operating condition it is given, it should suggest an action to the operator, the latter being free to ignore it. Thus, this is an open-loop configuration, where the tool cannot take as input the actual system's response to its own actions. Furthermore, the tool should not rely on any expensive intermediate simulation so as to meet real-time constraints.

Tertiary voltage control can be cast as an AC Optimal Power Flow (ACOPF) problem [2]. Despite an extensive body of literature, none of the current traditional resolution methods scale well to real-life problems, which involve large decision spaces and prohibitive combinatorial aspects. Hoping for a more scalable approach, the Power Systems community investigates the use of neural networks [3], a class of highly expressive and trainable functions. Neural networks are extremely fast at inference time, although they require a computationally heavy off-line training phase. They have achieved tremendous successes in various domains (computer vision [4], natural language processing [5] and games [6]), always displaying a strong ability to solve complex problems that require a very high level of abstraction with regards to the data at hand. In the context of ACOPF, neural networks can either be trained by imitation (supervised learning) or by interaction with a simulator through Reinforcement Learning (RL) [7]. Recent work explores the application of deep neural networks to ACOPF [8], while others [9], [10], [11], [12] frame the ACOPF problem as a closed-loop RL problem.

All previously mentioned works involving neural networks are valid under the assumption that topology (*i.e.* the way elements that make up the grid are interconnected together) remains constant. Such a hypothesis does not hold in real-life power systems: lines get disconnected, generators regularly start up and shut down through the course of a day and buses within a substation get merged and split. As a matter of fact, real-life power grid data have a graphical structure that may change from one operating condition to the other. A specific class of neural networks called Graph Neural Networks (GNNs) [13], [14] has been especially designed for such a case, and has thus been extensively applied to a wide range of power systems problems [15]. More specifically, GNNs have been trained to solve the closed-loop RL version of the ACOPF [16], [17], [18], and to imitate traditional ACOPF solvers [19].

As a first step towards a real-time decision making tool for tertiary voltage control, we focus on the sole choice of generator voltage setpoints. Our approach consists in the training of a GNN-based policy by interaction with a black-box simulator. The main contributions of this work are:

1) the first RL training of GNNs to address open-loop tertiary voltage control;

2) the introduction of Hyper Heterogeneous Multi Graphs[1] (H2MGs), that seamlessly represent power grids;
3) a companion GNN architecture that relies on Neural Ordinary Differential Equations (NODEs) [21];
4) an experimental comparison of our approach against an ACOPF baseline on data with large load and topology variations;
5) an exploration of the ability of our method to generalize to operating conditions – in particular topologies – never encountered during training;
6) an open-source implementation of our method, and open-source datasets along with their generation script.

Furthermore, we emphasize the following properties, that will prove critical in the face of real-life systems:

- training is operated by interaction with a simulator, not by imitation of a traditional ACOPF solver;
- the simulator is considered as a black-box, thus enabling the use of various kinds of simulators, including dynamical ones.

We believe that the present work lays a sound foundation for the real-time optimization of real-life cyber-physical structures (*i.e.* where additional couplings are induced by automata), considering both their continuous and topological aspects. Still, the present paper assumes a context where traditional ACOPF resolution methods are competitive: this allows for a sound assessment of our method against a well understood baseline.

The remainder of the document is organized as follows. Section II states the tertiary voltage control problem, converts it into a specific kind of reinforcement learning problem and then details our proposed H2MG data formalism and its companion neural network architecture. Section III provides details about the experimental protocol (dataset generation, baseline, hyper-parameters), compares our proposed method to an ACOPF baseline, and explores its ability to generalize to out-of-distribution operating conditions. Section IV summarizes the present study and explores promising research avenues with regards to our long term objective. While the present paper is self-contained, additional details can be found in the supplementary material [22].

II. Methodology

In this Section, we first describe the optimization problem we wish to solve, and how we approach it as an RL problem. Secondly, we propose a data formalism for operating conditions, as well as a companion neural network architecture. Finally, we detail how to solve our RL problem with the said architecture.

A. Initial Optimization Problem

Let us denote a power grid operating condition by $x \in \mathcal{X}$, which encompasses both its topological structure and its numerical features[2]. This includes buses with their voltages bounds, loads with their active and reactive injections, generators with their active generation and reactive power bounds, lines and transformers with their admittances, etc. Let $y \in \mathcal{Y}(x)$ be a tertiary voltage control decision, which is contingent on the number and nature of voltage control assets available in x. In the context of this paper, y is the vector of generator voltage setpoints, whose size varies with the amount of generators present in x. We are looking for a value for y that minimizes a real-valued cost function c,

$$y^*(x) \in \underset{y \in \mathcal{Y}(x)}{\arg\min}\ c(x, y). \tag{1}$$

The long-term purpose of our research project is to address the problem of tertiary voltage control using both continuous (secondary voltage setpoints and ratio tap changers setpoints) and discrete (lines and shunts opening and synchronous condensers activation) control variables. As a first step towards this goal, we only consider generators voltage setpoints, which are continuous control variables.

When controlling voltages, operators actually aim at achieving multiple goals, each involving various time scales. As a proxy of this complex process, our cost function c includes losses c_L and penalizes voltages (resp. currents) that are too close or beyond their permanent limits using c_V (resp. c_I).

$$c(x, y) = c_L(x, y) + \lambda \left[c_V(x, y; \epsilon) + c_I(x, y; \epsilon)\right] \tag{2}$$

$$c_L(x, y) = \frac{\sum_{\ell \in \mathcal{E}_x^{branch}} P_\ell^L}{\sum_{n \in \mathcal{E}_x^{load}} P_n^d} \tag{3}$$

$$c_V(x, y; \epsilon) = \frac{1}{|\mathcal{E}_x^{bus}|} \sum_{n \in \mathcal{E}_x^{bus}} \max(0, \epsilon - v_n, v_n - 1 + \epsilon)^2 \tag{4}$$

$$c_I(x, y; \epsilon) = \frac{1}{|\mathcal{E}_x^{branch}|} \sum_{\ell \in \mathcal{E}_x^{branch}} \max\left(0, \epsilon - i_\ell, i_\ell - 1 + \epsilon\right)^2 \tag{5}$$

where $\mathcal{E}_x^{branch}$, $\mathcal{E}_x^{load}$, $\mathcal{E}_x^{bus}$ are respectively the sets of branches, loads and buses of x; P_ℓ^L is the power loss at branch ℓ; P_n^d is the power consumed by load n; $v_n = (V_n - \underline{V}_n)/(\overline{V}_n - \underline{V}_n)$ is the voltage magnitude of bus n normalized by its operational limits $(\underline{V}_n, \overline{V}_n)$; $i_\ell = (I_\ell - \underline{I}_\ell)/(\overline{I}_\ell - \underline{I}_\ell)$ is the current magnitude of branch ℓ normalized by its operational limits $(\underline{I}_\ell, \overline{I}_\ell)$ (taking the max normalized current over the two branch extremities); $\lambda > 0$ balances penalization terms c_V and c_I with regards to losses c_L; and $\epsilon > 0$ activates the said penalization before a violation occurs. Right-hand terms in equations (3), (4) and (5) indeed depend on x and y, as currents, voltages and losses are estimated by a power system simulator. Dependencies are not made explicit for the sake of readability. Notice that all intermediate costs (3), (4) and (5) are dimensionless, and do not scale with the power grid size. Pairs (x, y) that do not make the simulator converge are associated with a prohibitively large cost.

B. Reinforcement Learning Problem

Our goal is to train a deep neural network to solve the problem (1) not for just a single value of x, but instead for a whole distribution p of power grid operating conditions. Provided with a certain $x \sim p$, we have to return in one step a value for y that minimizes the black-box cost function

[1] H2MGs appeared in a prior PhD thesis [20], but not in published work.
[2] Assuming all binary and categorical features have been mapped to $\mathbb{R}$.

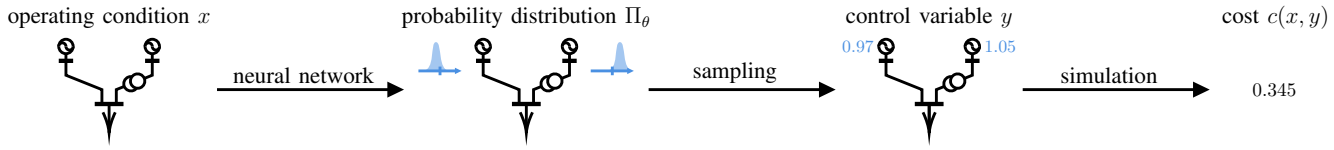

Fig. 1. A power grid operating condition x is passed to a neural network which produces a probability distribution $\Pi_\theta(\cdot|x)$ for each control variable (*i.e.* generator voltage setpoints). A value for y can thus be sampled, and passed to the cost function c for evaluation, which involves a simulation.

$c(x, \cdot)$. This is a case of *contextual bandit* problem (with the particularity of a deterministic reward), which is typically addressed by the RL literature [7]. A key idea in RL is to consider a probabilistic control policy Π_θ, which takes the form of a conditional probability distribution over $\mathcal{Y}(x)$ knowing a certain operating condition x, and parameterized by a vector θ. We are thus looking for a parameter value θ^* that minimizes c for the distribution p,

$$\theta^* \in \arg\min_{\theta\in\Theta} \mathbb{E}_{\substack{x\sim p(.)\\ y\sim\Pi_\theta(.|x)}} [c(x, y)] . \tag{6}$$

Once trained, the policy is used deterministically by choosing $\arg\max_y \Pi_\theta(y|x)$ as a decision variable for a given x. The pipeline from an operating condition x, to a probability distribution Π_θ, a sample y and then a cost $c(x, y)$ is illustrated by Figure 1.

As a matter of fact, we do not assume that we have a direct access to the probability distribution p, but only to a dataset $\mathcal{D}_{train}$ sampled from it. The policy Π_θ is trained over $\mathcal{D}_{train}$ in the hope that it will perform well on another dataset $\mathcal{D}_{test}$ independently sampled from p.

Readers from the RL domain may find useful to think of c as the opposite of a *reward*, of x as a *state* and of y as an *action*.

Our control variables being continuous, we choose Π_θ to be a multivariate Gaussian distribution defined as

$$\Pi_\theta(\cdot|x) = \mathcal{N}(f_\theta(x), \sigma^2 \mathbb{1}), \tag{7}$$

where $\sigma > 0$ is a fixed parameter, $\mathbb{1}$ is the identity matrix of suitable dimension, and f_θ is a trainable function parameterized by a vector θ (see Subsection II-D). Notice that the covariance matrix could also be output by a neural network, which should be investigated in future work.

C. Hyper-Heterogeneous Multi Graphs (H2MGs)

Let us now introduce our Hyper Heterogeneous Multi Graph (H2MG) formalism to represent operating conditions.

1) Topological Variations: In practice, power grid operating conditions display a wide variety of topological configurations. New transmission lines, shunts or generators may be built, while some preexisting ones can get temporarily or definitively disconnected. Moreover, operators can perform bus-splitting operations that deeply alter interconnection patterns. We are thus facing a distribution p of operating conditions x where both numerical features and the topology can vary, as illustrated in Figure 2. To address this challenge, it is essential to represent operating conditions in a suitable fashion.

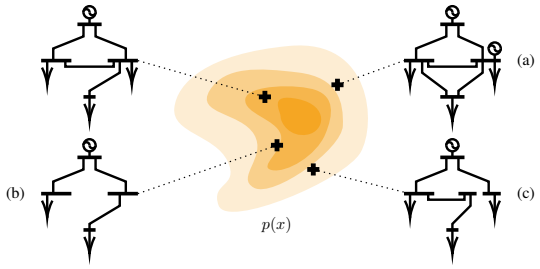

Fig. 2. In actual power grid distributions, topology varies from one operating condition to the other: (a) new assets can be built, (b) preexisting ones may be disconnected, and (c) operators may perform bus-splitting operations.

A naive approach consists in framing operating conditions as *standard* graphs. This however requires to aggregate together generator, load, shunt and bus features into a *node*. Transmission lines and transformers on the other hand are cast as *edges* without distinction, as shown in Figure 3b. This pre-processing step is the cause of an information loss which may prove detrimental in the face of our long term objective.

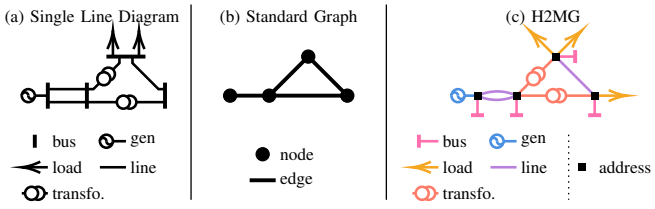

Fig. 3. Representation of a power grid as a single-line diagram (a) (at the busbar resolution), as a *standard* graph (b) and as a H2MG (c). *Standard* graphs are made of nodes and edges, and require to aggregate together objects of different natures, while our proposed H2MG formalism allows for a seamless representation of power grid operating conditions.

Our H2MG formalism arises from our need to represent untampered real-life operating conditions. It will also prove critical in the future when we will have to account for cyber-physical couplings induced by the secondary voltage control (or any other automaton), as well as switches and breakers that make up the detailed topology of substations. Readers who are familiar with power systems softwares will likely notice that H2MGs actually coincide with most power systems data formats.

2) Graphical Structure: We define the sets of elements of the structure of an operating condition x by[3]:

[3]For the sake of simplicity of the explanation and without limitation, we don't distinguish here among different types of lines (*e.g.* multi-terminal, DC, AC, ...), different types of transformers (*e.g.* voltage vs phase), different types of generators, loads, and shunts, and we do not consider switches and breakers, and automata such as secondary voltage controls.

- a set of buses $\mathcal{E}_x^{\text{bus}}$,
- a set of generators $\mathcal{E}_x^{\text{gen}}$,
- a set of loads $\mathcal{E}_x^{\text{load}}$,
- a set of shunts $\mathcal{E}_x^{\text{shunt}}$,
- a set of lines $\mathcal{E}_x^{\text{line}}$,
- a set of transformers $\mathcal{E}_x^{\text{transfo}}$.

Elements of such sets are called hyper-edges, and are interconnected via a set of *addresses* A_x to form the graphical structure of an operating condition x, as illustrated in Figure 3c. *Addresses* serve as interfaces between hyper-edges, and do not bear any numerical input feature.

Let us denote by $\mathcal{C} := \{\text{bus}, \text{gen}, \text{load}, \text{shunt}, \text{line}, \text{transfo}\}$ the set of all hyper-edge *classes*. For a given class $c \in \mathcal{C}$, all hyper-edges are of the same *order*, meaning that they are all connected to the same number of *addresses*.

For instance, transmission lines are of order 2: they have one "from" port, and one "to" port. On the other hand, loads are of order 1, since they are only connected to one *address*.

Instead of being composed of nodes and edges, graphical structures of H2MGs are made of a series of hyper-edges of various classes and orders, all interconnected through addresses. They lie at the intersection of:

- Hyper graphs – Hyper-edges of various orders (1, 2, etc.);
- Heterogeneous graphs – Multiple classes of hyper-edges;
- Multi graphs – Collocated hyper-edges of the same class.

As illustrated in Figure 3, they allow for a much more natural and exhaustive representation of operating conditions than *standard* graphs, since they do not require to aggregate together assets into nodes and edges.

We denote by $\mathcal{O}^c$ the ordered set of ports of class $c \in \mathcal{C}$. A port $o \in \mathcal{O}^c$ is a mapping from a hyper-edge to an address, $o : \mathcal{E}_x^c \mapsto A_x$. For instance, two winding transformers are connected to exactly two addresses, and they have two distinct ports ("high voltage" and "low voltage") which are not interchangeable.

We define the neighborhood of address $a \in A_x$ as

$$N_x(a) = \{(c, e, o) | c \in \mathcal{C}, e \in \mathcal{E}_x^c, o \in \mathcal{O}^c, o(e) = a\}. \quad (8)$$

This notion of neighborhood of address a basically tells:

- which hyper-edges of x are connected to address a;
- to which classes they belong;
- through which ports they are connected to a.

3) Feature Vectors: All hyper-edges of an operating condition x bear real-valued features: loads are defined by their active and reactive power, transmission lines by their resistance, reactance, etc. We denote by x_e^c the feature vector born by hyper-edge $e \in \mathcal{E}_x^c$ of class $c \in \mathcal{C}$. All hyper-edges of the same class bear feature vectors of the same dimension.

D. Neural Network Architecture

Our tertiary voltage control policy Π_θ relies on a trainable function f_θ to compute the mean of a multivariate Gaussian distribution, as stated in equation (7). The said function should take an operating condition x as input, and return one voltage setpoint value per available generator. Let us remind that the number of connected generators may vary from one operating condition to the other.

As motivated in Section I, f_θ is chosen to be a GNN [13], [14], [15]. Moreover, our architecture borrows from the NODE literature [21], and is thus called Hyper Heterogeneous Multi Graph Neural Ordinary Differential Equation (H2MGNODE). Its overall process is illustrated in Figure 4.

1) Latent Variables: Our proposed architecture relies on the use of a series of latent vectors $(h_a)_{a \in A_x}$ located at each address of the input operating condition x. The dimension of h_a is a hyper-parameter.

We introduce the simplifying notation $h_e = (h_{o(e)})_{o \in \mathcal{O}^c}$, which is the concatenation of latent vectors located at all ports of a hyper-edge e. For instance, if we consider a transmission line e, connected to address 45 on its "from" port, and to address 78 on its "to" port, then $h_e = (h_{45}, h_{78})$.

2) Architecture Equations: Our proposed H2MGNODE architecture is defined by the following equations,

$$\forall (c, e), \quad \tilde{x}_e^c = \Xi_\theta^c(x_e^c), \quad (9)$$

$$\forall a, \quad h_a(0) = [0, \dots, 0], \quad (10)$$

$$\forall a, \quad \frac{dh_a}{d\tau} = \nu \left(\sum_{(c,e,o) \in N_x(a)} \Phi_\theta^{c,o}(\tilde{x}_e^c, h_e(\tau), \tau) \right), \quad (11)$$

$$\forall (c, e), \quad \mu_e^c = \Psi_\theta^c(\tilde{x}_e^c, h_e(1)). \quad (12)$$

where ν is a bounded monotonic and smooth element-wise function, and illustrated by Figure 4. Functions Ξ, Φ and Ψ are basic Multi Layer Perceptrons (MLPs). Notice that the parameter vector θ can be split in multiple parts, each being associated with a specific neural network block.

a) Encoding: Equation (9) describes the embedding of the input operating condition into a latent space, whose dimension is a hyper-parameter. The same MLP Ξ^c is applied to all hyper-edges of class $c \in \mathcal{C}$. Input features are normalized beforehand, as detailed in the supplementary material [22].

b) Interaction: Latent variables $(h_a)_{a \in A_x}$ obey a dynamical system defined by equations (10 – 11), where τ denotes time. Interactions follow the graphical structure of the input operating condition x. Addresses a and a' directly interact only if there exists a hyper-edge connected to both of them. They may also indirectly interact through common neighbors. Inference is obtained by solving the dynamical system forward in time from $\tau = 0$ to 1, while back-propagation (*i.e.* the estimation of $\nabla_\theta f_\theta(x)$) is based on solving it backward in time [21]. Coupling mappings $\Phi_\theta^{c,o}$ are shared across all ports o of hyper-edges of class c.

c) Decoding: The latent variables' values obtained at $\tau = 1$ are then converted into actually meaningful quantities at hyper-edges through equation (12). The same MLP Ψ^c is applied to all hyper-edges of class $c \in \mathcal{C}$. In our case, this step simply returns a scalar value μ_e^c for each controllable device e of class c connected in the operating condition x. This value is then used as the mean of the normal distribution defined in equation (7). The resulting output may be scaled to a reasonable range of values by an affine transformation.

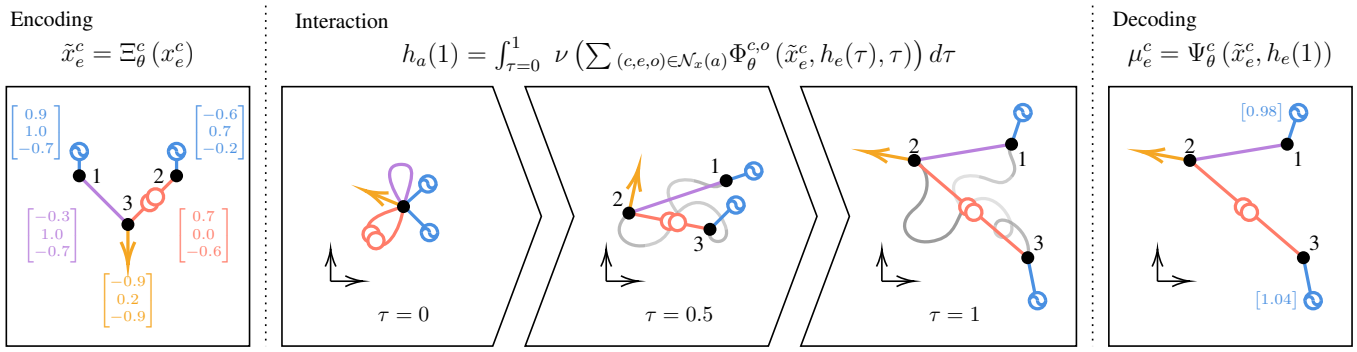

Fig. 4. Neural Network Architecture. The feature vectors of all hyper-edges are embedded into a latent space using class-specific encoders Ξ_θ^c. All addresses are associated with latent coordinates initialized at zero, and then follow a trajectory defined by a differential system whose second member involves couplings born by hyper-edges and defined by neural networks. Addresses interact until $\tau = 1$. Finally, hyper-edges exploit the final locations of their addresses to produce a meaningful prediction thanks to class-specific decoders Ψ_θ^c.

E. Training

Let us now define how to train our tertiary voltage control policy Π_θ. We have chosen an elementary RL algorithm called REINFORCE [7], which is quite suitable for our case (absence of sequentiality and deterministic cost function). It is based on the so-called "log-trick" ($\nabla_\theta \Pi_\theta = \Pi_\theta . \nabla_\theta \log \Pi_\theta$), which provides the following estimator of the gradient for a given x,

$$\nabla_\theta \mathbb{E}_{y\sim\Pi_\theta}\left[c(x,y)\right] = \mathbb{E}_{y\sim\Pi_\theta}\left[c(x,y).\nabla_\theta \log \Pi_\theta(y|x)\right]. \quad (13)$$

Depending on x, costs can vary on very different scales. Operating conditions x with larger cost variations induce a larger gradient, which hinders contributions of more common operating conditions. Standardizing costs per operating condition x (as detailed in Algorithm 1) has been observed to greatly improve the training speed and stability [23].

Recalling that Π_θ is a multivariate Gaussian distribution given by equation (7), we obtain the following:

$$\nabla_\theta \log \Pi_\theta(y|x) \propto (y - f_\theta(x)).\nabla_\theta f_\theta(x). \quad (14)$$

The policy training consists in iterating over operating conditions x, testing multiple tertiary voltage control values $y \sim \Pi_\theta(\cdot|x)$, and updating θ, as detailed in Algorithm 1.

Algorithm 1 REINFORCE Algorithm with standardized costs

Require: $\mathcal{D}_{train}$; $E, I, J \in \mathbb{N}^*$; $\sigma^2, \alpha \in \mathbb{R}^{+*}$
  Initialize θ randomly
  for $e \in \{1, \ldots, E\}$ **do**
    Sample I operating conditions $(x_i)_{i\in\{1,\ldots,I\}}$ from $\mathcal{D}_{train}$
    for $i \in \{1, \ldots, I\}$ **do**
      for $j \in \{1, \ldots, J\}$ **do**
        $y_{i,j} \sim \mathcal{N}(f_\theta(x_i), \sigma^2 \mathbb{1})$ ▷ Sample action
        $c_{i,j} \leftarrow c(x_i, y_{i,j})$ ▷ Compute cost
      end for
      $\hat{m}_i \leftarrow \frac{1}{J}\sum_j c_{i,j}$
      $\hat{v}_i \leftarrow \frac{1}{J}\sum_j (c_{i,j} - \hat{m}_i)^2$
      for $j \in \{1, \ldots, J\}$ **do**
        $\hat{a}_{i,j} \leftarrow \frac{c_{i,j} - \hat{m}_i}{\sqrt{\hat{v}_i}}$ ▷ Standardize costs
      end for
    end for
    $\theta \leftarrow \theta - \alpha \frac{1}{I\times J} \sum_{i,j} \hat{a}_{i,j}.(y_{i,j} - f_\theta(x_i)).\nabla_\theta f_\theta(x_i)$
  end for

III. Case Study

We validate our approach on the tertiary voltage control problem introduced at the beginning of Section II, by comparing it to an ACOPF solver baseline.

A. Dataset Generation

At first, three distinct variants[4] (namely *Standard*, *Condenser* and *Reduced*) were crafted from the *case60nordic* test case [24] as detailed in Figure 5, where all voltage limits are set to $\underline{V} = 0.9$ p.u. and $\overline{V} = 1.1$ p.u. Tuning parameters of the tertiary voltage control problem are set to $\lambda = 200$ and $\epsilon = 5\%$, and a simulator non-convergence is associated to a cost of 1.

For each of the three variants, exactly $100,000$ operating conditions are generated for the train set, $2,000$ for the validation set and $10,000$ for the test set. Instead of aiming for a so-called "realistic" dataset, we sample from a broad domain and then reject cases that do not meet certain criteria. Operating states vary in terms of topology (0 to 4 lines can be disconnected), total load (uniformly sampled between 50% and 120% of the initial value), individual load (sampled from a Gaussian distribution) and generation (defined by a DCOPF with random merit order), as detailed in the supplementary material [22]. Voltage setpoints are initialized at 1.0 p.u.. Resulting operating conditions being frequently in overvoltage in our context, we choose to disconnect all shunt capacitors and to keep connected all shunt inductors to reduce voltages. Each operating condition undergoes checks to reject cases with overflows, unrealistic bus voltages (outside of the [0.85 p.u., 1.15 p.u.] range), or non-convergences. Object disconnections (generators, transmission lines, north-east zone) are performed by completely removing the said objects from the network, which implies that the number of hyper-edges of a given class varies from one operating condition to the other. In the *Standard* test set, the number of generators varies

[4] available at github.com/bdonon/updating_case60nordic/tree/PSCC24

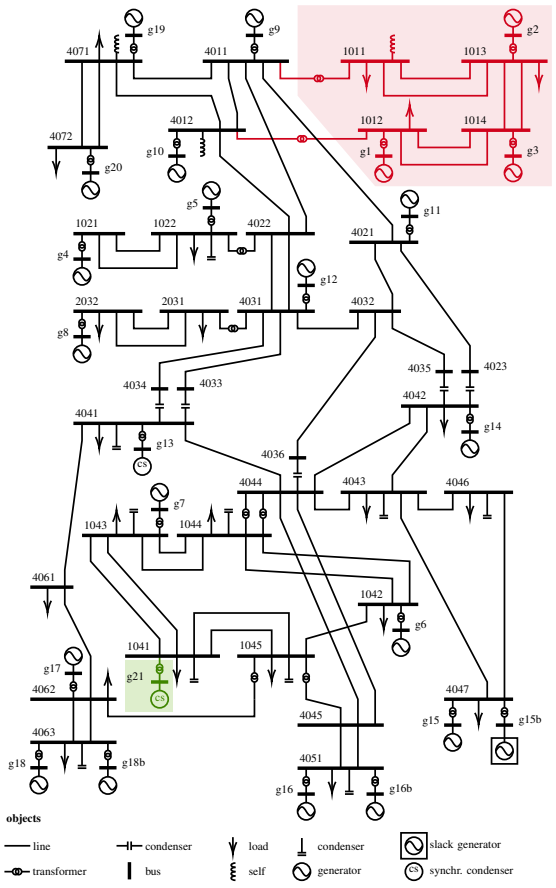

Fig. 5. Case60nordic test case. The *Condenser* dataset includes the whole system, the *Standard* dataset does not have the synchronous condenser g21 (green), and the *Reduced* dataset has neither the synchronous condenser g21 (green) nor the north-east region (red).

between 6 and 19 (while 23 are initially present), the number of lines varies uniformly between 53 and 57 (while 57 are initially present), the total load varies between 3 GW and 7 GW, and the initial number of voltage violations varies between 0 and 21. The supplementary material [22] provides a more comprehensive overview of the datasets' generation procedure and summary statistics.

A fourth dataset called *All* is composed of the union of the three previous datasets.

B. Optimization Baseline

In order to set an objective baseline for the proposed approach, we state and solve a relaxed version of the non-linear ACOPF problem in polar coordinates. All equality constraints have the standard form of the ACOPF equality constraints and have been reproduced from [2]. The relaxation concerns tolerating the violation of branch current magnitude and voltage magnitude inequality constraints, which is penalized in objective function (1) as per (2 – 5). The detailed formulation is included in the supplementary material [22].

Our implementation of the ACOPF baseline[5] uses the interior point method. It is developed in Julia [25] using the JuMP modeling language [26] and the PowerModels.jl framework [27] for data conversion. We rely on the KNITRO solver [28] for solving all non-linear optimization problem instances. Experiments are run on a MacBook Air M2 2022, and took approximately $0.13s$ per operating condition.

C. Experiment Settings

Our method[6] is implemented in Python [29] and JAX [30]. Multiple operating conditions are run in parallel, and power flow simulations use PandaPower [31]. In order to be compliant with the ACOPF formulation used by the baseline, we consider the normalized squared current $i_\ell = (I_\ell^2 - \underline{I}_\ell^2)/(\overline{I}_\ell^2 - \underline{I}_\ell^2)$ in equation (5). During preliminary studies, various hyper-parameter values have been tested. In the following, we report values used in our experiments.

1) Policy parameters: All latent variables are vectors of dimension 64. Encoders Ξ and decoders Ψ are MLPs with 2 hidden layers of size 128 and 64, with hyperbolic tangent non-linearities. Coupling mappings Φ are MLPs with 1 hidden layer of size 128 and hyperbolic tangent non-linearities. The function ν is a hyperbolic tangent. All MLPs are implemented in Flax [32].

ODEs defined by equations (10) and (11) are solved by a first-order explicit Euler scheme ($d\tau = 0.05$), implemented in Diffrax [33]. Back-propagation (that involves solving the system backwards in time) is performed using the same scheme.

The policy's standard deviation σ is set to 0.005 p.u..

2) Training algorithm: For each of the four datasets, we consider mini-batches of $I = 32$ operating conditions, and $J = 16$ different actions tested on each one. The gradient is processed by the optimizer Adam [34], with a learning rate of 3×10^{-4}, and with standard parameters. Each training lasts $E = 200,000$ iterations, which took 6 days. Power system simulations are run on AMD EPYC 7763 CPUs (16 Threads in parallel), while policy inference and back-propagation are run on NVIDIA RTX A5000 GPUs. Policies are evaluated over their respective validation sets every $1,000$ iterations, and only the best version is kept. A single training is performed per train set. Inference takes 0.26 s per operating condition.

Figure 6 shows the convergence of the four different training processes in terms of average cost over their respective validation sets.

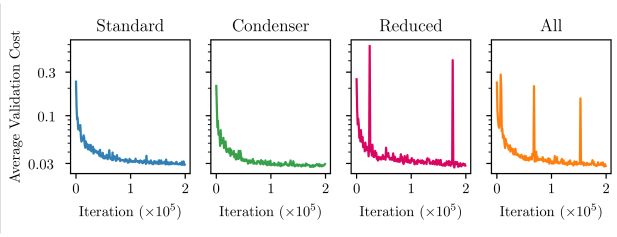

Fig. 6. Average validation cost during training of the four models. The vertical axis is in logarithmic scale.

[5]available at github.com/montefiore-ms/ACOPF4TVC

[6]available at github.com/bdonon/PSCC2024

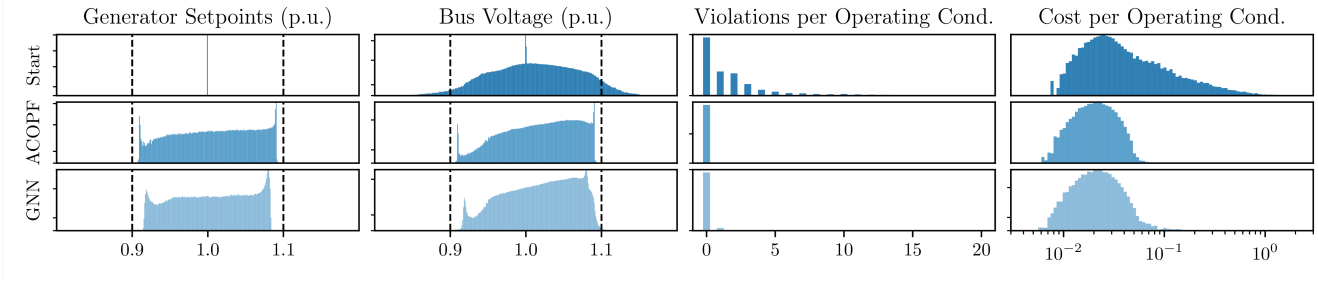

Fig. 7. Comparison of the data distribution before any modification (top row), after the application of the ACOPF solver baseline (middle row), and after the application of our policy trained on the *Standard* train set (bottom row). All data are from the *Standard* test set. The first column displays the histogram of all voltage setpoints (control variables) over all generators and all operating conditions. The second column shows the histogram of resulting bus voltage magnitudes over all buses and all operating conditions. The third column contains histograms of the violation count (current or voltage) per operating conditions. The fourth column presents histograms of costs for all operating conditions. The vertical axis is represented in *symlog* scale (*i.e.* $y \mapsto \ln(y+1)$). Both our trained policy and the ACOPF solver baseline manage to bring the distribution of voltages mostly within authorized bounds ($\underline{V} = 0.9$ p.u. and $\overline{V} = 1.1$ p.u.), although some marginal violations remain.

D. Results

We now investigate the performance of our four policies, each being trained on a different train set. More detailed results are available in the supplementary material [22]. The term violation encompasses both overflows and voltage violations. Both the ACOPF solver baseline and our policies cause approximately 0.2% of non-convergence over each test set (this rate goes up to 1% for some policies tested on the *Reduced* test set). In the following, non-convergences are counted as violations, but they are not displayed on histograms. Notice that among those operating conditions having a violation with our trained policies, 90% display a voltage violation, while only 20% display a current violation.

1) Standard Dataset Results: At first, let us analyse results of a policy (denoted by *GNN*) trained on the *Standard* train set, and tested on the *Standard* test set. Figure 7 compares its performance against the initial situation where all voltage setpoints are set to 1.0 p.u. (denoted by *Start*), and the ACOPF solver baseline (denoted by *ACOPF*).

Histograms of voltage setpoints issued by the ACOPF solver and our policy are quite similar: peaks can be observed near extreme values, and they both have an overall preference for high voltage setpoint values. The distribution of resulting voltages at all buses are also similar between the two approaches, although our GNN policy has more difficulty preventing overvoltages. The ACOPF solver brings the number of snapshots with at least one current or voltage violation from 58% to 0.98%, while our policy brings it to 5.2%. Still, both approaches manage to drastically reduce the severity of violations: they both bring the percentage of operating conditions with two or more violations close to zero, and violating values are reasonably close to limits.

2) Transfer Properties: A key property of our trained models is their ability to be compatible with any possible operating condition, regardless of its number of assets (generators, lines, etc.) and of the way they are interconnected. Table I displays the percentage of operating conditions in each test set with at least one violation while using each one of the four policies trained on the four training sets. In order to estimate the severity of violations, Table II considers an additional 5% tolerance: only voltages out of the $[0.89$ p.u., 1.11 p.u.$]$ range and currents above 105% of thermal limits count as a violation.

TABLE I
OPERATING CONDITIONS W/ VIOLATION.

	Standard test set	Condenser test set	Reduced test set
Start	58%	58%	59%
ACOPF baseline	0.98%	0.82%	1.0%
GNN trained on Standard	**5.2%**	5.8%	18%
GNN trained on Condenser	5.4%	**5.1%**	46%
GNN trained on Reduced	35%	34%	**4.6%**
GNN trained on All	4.5%	4.5%	4.7%

TABLE II
OPERATING CONDITIONS W/ VIOLATION – 5% TOLERANCE.

	Standard test set	Condenser test set	Reduced test set
Start	44%	44%	46%
ACOPF baseline	0.52%	0.49%	0.45%
GNN trained on Standard	**1.6%**	1.5%	6.0%
GNN trained on Condenser	1.9%	**1.4%**	20%
GNN trained on Reduced	21%	20%	**1.5%**
GNN trained on All	1.5%	1.3%	1.5%

We observe that policies trained on the *Standard* and *Condenser* train sets generalize decently to each other's test sets, while the policy trained on the *Reduced* train set does not generalize well to other situations. For each test set, the policy trained on the union of all train sets performs comparably to the policy trained on the corresponding train set. It has no difficulty dealing with a broader range of topological variations.

IV. Conclusion & Future Work

Our research project aims at developing a decision support tool for real-life and real-time tertiary voltage control. As an intermediate step towards this goal, we explore the training of

a GNN-based policy to optimize generators' primary voltage control setpoints, using the REINFORCE method together with a black-box simulator. We introduce the H2MG formalism to seamlessly represent power grid operating conditions, as well as a companion GNN architecture, the H2MGNODE. We experimentally showed on the *case60nordic* test case that our framework is able to reach competitive performances compared to an ACOPF baseline in a certain context. This confirms the relevance of our methodology, which we believe will scale better than traditional ACOPF solvers to larger systems and more complex problems. Furthermore, some of the trained policies were shown to generalize to out-of-distribution samples, which is an essential feature for practical applicability.

Future work will focus on the following aspects:

- *Improving the training process.* This first study is based on a standard implementation of the REINFORCE algorithm, which could be improved in terms of sample efficiency. Moreover, it would be interesting to explore the impact of various integration schemes and parameters for our H2MGNODE model.
- *Real-life power systems.* Our long-term objective is to build a decision support tool for the French transmission system. This will require to extend the methodology to discrete variables to account for the switching of shunts, lines and synchronous condensers. Moreover, the cyber-physical structure of the secondary voltage control and the detailed substation topology will have to be modelled with the H2MG formalism. Finally, we will have to assess the scalability of our approach to the French system in terms of computational times.
- *Other applications.* The rather general nature of our proposed methodology makes it a sensible candidate to address other complex power systems problems, that may include for instance sequential decision making under uncertainty and distributed control.

Acknowledgements

The authors would like to thank Vincent Barbesant and Florian Benoit for insightful discussions about the tertiary voltage control problem, Rémy Clément and Marc Schoenauer for their help in formalizing the H2MG approach, and the operators of the Alan GPU cluster at the University of Liège on which all experiments were run.

References

[1] P. Mandoulidis, *et al.*, "Controlling long-term overvoltages in lightly loaded transmission systems," in *13th Mediterranean Conference on Power Generation, Transmission, Distribution and Energy Conversion (MEDPOWER 2022)*, vol. 2022, pp. 7–12, 2022.

[2] A. Castillo, *Essays on the ACOPF Problem: Formulations, Approximations, and Applications in the Electricity Markets*. PhD thesis, 2016.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[4] C. Szegedy, *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Los Alamitos, CA, USA), pp. 1–9, 2015.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," in *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[6] D. Silver, *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. MIT Press, 1998.

[8] X. Pan, M. Chen, T. Zhao, and S. H. Low, "DeepOPF: A Feasibility-Optimized Deep Neural Network Approach for AC Optimal Power Flow Problems," *IEEE Systems Journal*, vol. 17, pp. 673–683, 2023.

[9] B. L. Thayer and T. J. Overbye, "Deep Reinforcement Learning for Electric Transmission Voltage Control," in *2020 IEEE Electric Power and Energy Conference (EPEC)*, pp. 1–8, 2020.

[10] H. Hagmar, L. A. Tuan, and R. Eriksson, "Deep Reinforcement Learning for Long-Term Voltage Stability Control," in *11th Bulk Power Systems Dynamics and Control Symposium (IREP 2022)*, 2022.

[11] H. Zhen, *et al.*, "Design and tests of reinforcement-learning-based optimal power flow solution generator," *Energy Reports*, vol. 8, pp. 43–50, 2022.

[12] Y. Zhou, *et al.*, "A Data-driven Method for Fast AC Optimal Power Flow Solutions via Deep Reinforcement Learning," *Journal of Modern Power Systems and Clean Energy*, vol. 8, pp. 1128–1139, 2020.

[13] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," *IEEE Transactions on Neural Networks*, vol. 20, pp. 61–80, 2009.

[14] T. N. Kipf and M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *International Conference on Learning Representations (ICLR 2017)*, 2017.

[15] W. Liao, B. Bak-Jensen, J. R. Pillai, Y. Wang, and Y. Wang, "A Review of Graph Neural Networks and Their Applications in Power Systems," in *Journal of Modern Power Systems and Clean Energy*, vol. 10, 2022.

[16] A. Lopez-Cardona, G. Bernardez, P. Barlet-Ros, and A. Cabellos-Aparicio, "Proximal Policy Optimization with Graph Neural Networks for Optimal Power Flow," *preprint arXiv:2212.12470*, 2022.

[17] D. Owerko, F. Gama, and A. Ribeiro, "Unsupervised Optimal Power Flow Using Graph Neural Networks," *preprint arXiv:2210.09277*, 2022.

[18] J. Li, R. Zhang, H. Wang, Z. Liu, H. Lai, and Y. Zhang, "Deep Reinforcement Learning for Optimal Power Flow with Renewables Using Graph Information," *preprint arXiv:2112.11461*, 2022.

[19] F. Diehl, "Warm-Starting AC Optimal Power Flow with Graph Neural Networks," in *NeurIPS 2019 Workshop on Tackling Climate Change with Machine Learning*, 2019.

[20] B. Donon, *Deep statistical solvers & power systems applications*. PhD thesis, 2022.

[21] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural Ordinary Differential Equations," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[22] B. Donon, *et al.*, "Topology-Aware Reinforcement Learning for Tertiary Voltage Control - Supplementary Material," tech. rep., 2023. https://hdl.handle.net/2268/306778.

[23] A. Karpathy, "Deep Reinforcement Learning: Pong from Pixels," 2016. karpathy.github.io/2016/05/31/rl/.

[24] F. Capitanescu, "Suppressing ineffective control actions in optimal power flow problems," *IET Generation, Transmission & Distribution*, vol. 14, pp. 2520–2527, 2020.

[25] J. Bezanson, A. Edelman, S. Karpinski, and V. Shah, "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, pp. 65–98, 2017.

[26] I. Dunning, J. Huchette, and M. Lubin, "JuMP: A Modeling Language for Mathematical Optimization," *SIAM Review*, vol. 59, pp. 295–320, 2017.

[27] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "PowerModels.jl: An Open-Source Framework for Exploring Power Formulations," in *2018 Power Systems Computation Conference (PSCC)*, 2018.

[28] R. H. Byrd, J. Nocedal, and R. A. Waltz, "Knitro: An Integrated Package for Nonlinear Optimization," in *Large-Scale Nonlinear Optimization*, vol. 83, pp. 35–59, Springer US, 2006.

[29] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. 2009.

[30] J. Bradbury, *et al.*, "JAX: composable transformations of Python+NumPy programs," 2018.

[31] L. Thurner, *et al.*, "pandapower - an Open Source Python Tool for Convenient Modeling, Analysis and Optimization of Electric Power Systems," *IEEE Transactions on Power Systems*, 2018.

[32] J. Heek, *et al.*, "Flax: A neural network library and ecosystem for JAX," 2023.

[33] P. Kidger, *On Neural Differential Equations*. PhD Thesis, 2021.

[34] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," in *International Conference on Learning Representations (ICLR)*, 2015.