Institut de Mathématiques d'Orsay
Université Paris-Saclay
RTE - Réseau de Transport d'Electricité

Possibilities of a decentralized power grid management in real time using multi-agent reinforcement learning

Farid Najar
Master Mathematics and Artificial Intelligence

Contents

1 Introduction **4**

2 Theoretical formalism **5**
2.1 Power grid management 5
2.2 Reinforcement learning 7
2.3 Game theory 12
2.3.1 POSG (Partially Observable Stochastic Games) 12
2.3.2 AEC (Agent Environment Cycle) 14
2.4 Multi-agent reinforcement learning 15

3 Environment **16**
3.1 Grid2Op 16
3.1.1 Power grid components 17
3.1.2 Power grid as a graph 18
3.1.3 Observations 19
3.1.4 Actions 20
3.1.5 Rewards 21
3.1.6 Markov property in Grid2Op 23
3.2 Multi-agent Grid2Op 23
3.2.1 Multi-agent Grid2Op as a Dec-MDP 24
3.2.2 Conflict handling 25

4 Experiments **26**
4.1 Do nothing, random dummy actions and parrot agents 27
4.2 Heuristic agents 27
4.2.1 Do nothing assumption 28
4.2.2 Guess/predict others' actions 28
4.2.3 Results 29
4.2.4 Possible improvements 30
4.3 Deep RL agents 34
4.3.1 Results 34
4.3.2 Possible improvements 35

5 Conclusion **35**

A Predictors' hyper parameters **38**

B RL agents hyper parameters **38**

Remerciements

Tout d'abord, je remercie ma famille qui m'a aimé, aidé et accompagné toutes ces années malgré les difficultés rencontrées.
Je remercie les organisateurs de notre master "Mathématiques de l'Intelligence Artificielle", en particulier, Monsieur Gilles Blanchard qui m'a accordé sa confiance en tant que responsable du master. Et durant ces deux dernières années, il m'a aidé et accompagné en tant que tuteur.
Je remercie bien évidemment RTE et toute l'équipe du pôle DS-IA qui m'ont accueilli durant ce stage. Je remercie d'autant plus mes encadrants, Messieurs Karim Chaouache, Benjamin Donnot et Clément Goubet qui m'ont accompagné et aidé patiemment.

Abstract

Power grid management is challenging because of the task's complexity and the importance of electricity as an energy source. This task is becoming increasingly complex and difficult as our power systems undergo an unprecedented transition to renewable energies. In recent years, the company in charge of the French power grid developed and held different researches using artificial intelligence and reinforcement learning.
In this work, we introduce a multi-agent environment that uses the original environment made by RTE, called Grid2Op.
We also explore different possibilities for a multi-agent operation. We show that using heuristic methods give us reasonable results, but that they are not suitable for larger grids.
On the other hand, deep learning methods seem unable to learn effectively in this new multi-agent environment, but they are faster for computations.
As a conclusion, we recommend a hybrid approach using both methods, in order to take advantage of the best of the two worlds.

1 Introduction

Electricity is a vital resource for us. Power grids are in an unprecedented transition towards clean energies. In addition, other low carbon efforts like electric vehicles (EV) and electrification projects (e.g. heat pumps) not only increase demand for electricity, but also increase the variability of the consumption. Due to their random nature, these changes add more complexity to the grid. Beside that, many households can produce electricity and inject the surplus to the grid. In other words consumers can become producers. Moreover, European power grids are more and more interdependent that requires more collaboration and communication between them.
The power grid can be partitioned in four categories; production, transmission (high voltage), distribution (low voltage) and consumption (see Figure 2). Here, we are interested by the transmission part. In France, RTE (Réseau de Transport d'Electricité) is the company responsible for this mission. It is responsible for the operation, maintenance and development of the French high-voltage transmission system, which at approximately 100,000 kilometres (62,000 mi), is Europe's largest.
With these novelties and difficulties, in order to manage the power grid, operators need more information, which means more data to process (see Figure 1). This increases the workload and fatigue among human operators, leading to possible errors.
Hence, RTE seeks to conceive different decision-making tools to help operators in the process. Some of these, based on classical optimization and supervised learning, are already deployed and used. Recently, the company aims to explore reinforcement learning that has shown impressive performances (see the example of controlling a fusion reactor [20]). To do so, we need a relevant environment in which we can train AI models, since we can't do it in the real world ! Grid2Op is a simulator that allows to perform power grid operations. The current Grid2Op is a classical reinforcement learning (RL) environment with only one agent operating in the grid. This configuration may work for small grids but for large grids, this approach faces serious issues because of the curse of dimensionality. Indeed, the number of possible actions grows exponentially with respect to the number of objects in the grid (see 3.1.4). Furthermore, having only one agent handling the entire grid raises resilience and security concerns. In this context, we would like to test many agents in collaboration to handle these large observation and action spaces as RTE seeks to scale up. Therefore, we need a multi-agent framework in which different agents can operate and collaborate together without (or with few) conflicts.
In the classical Grid2Op environment (single agent), it is possible to implement a multi-agent model by defining a master agent who manages regional agents (see [17] for more information). But, this is a task done by the user, and the regional agents can't interact directly with the environment and take decisions independently. Our objective is to ease this process by introducing a multi-agent version of Grid2Op.
In this internship work, we aim to design a relevant multi-agent version for Grid2Op, in phase with real world standards, that enables power grid distributed management in real time. Then, we analyze the behavior of some decentralized agents in the environment, especially their performance compared to the single agent case, and the collaboration

between them.

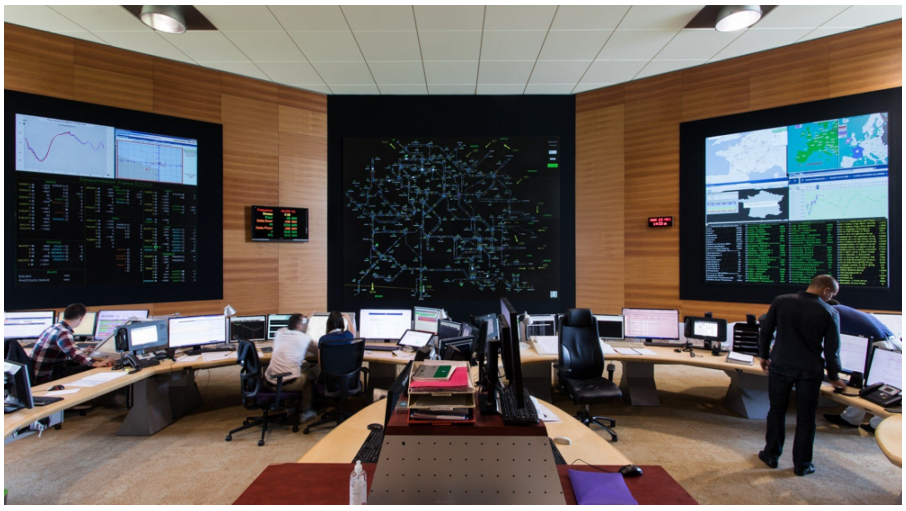


Figure 1: A power grid control room in France. Image from RTE.

2 Theoretical formalism

2.1 Power grid management

The power grid, an interconnected network for delivering electricity from producers to consumers (see Figure 2), has become an essential component of modern society. For a safe and reliable transmission of electricity, it is constantly monitored and managed by human experts in control rooms. Operators of electricity transmission and distribution networks are responsible for the safe and reliable operation of these networks. This task grows progressively more challenging due to the increasing presence of renewable generation and power-electronics-based resources on the grid. Both trends combine to make power flows in the network more variable, less predictable and sensitive to disturbances. Network control rooms are staffed by operators who rely on their experience to anticipate and resolve undesirable system behavior. Faced with an increasingly volatile network, continuing this mode of operation is likely to reduce system security, and/or greatly increase costs of operation. Therefore, there has been growing interest in automatically controlling and managing the power grid. Most of these methods are based on operations research and optimization, but, some researchers and system operators have proposed machine learning to anticipate risks, notably in the context of online dynamic security assessment [3][12].
Recently, *reinforcement learning (RL)* has emerged as a powerful approach to automated decision making. Through iterative experimentation, RL arrives at policies that aim to maximise a numerical reward signal. The basic RL framework is applicable to a tremendous range of settings including the control of electricity grids [16].

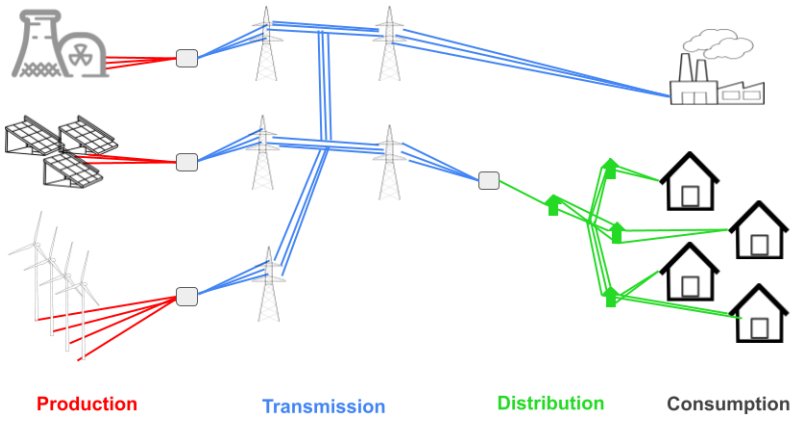


Figure 2: Simple scheme of power grids organization. The production is mixed between several power sources. The transportation consists in transmission (high voltage) and distribution (low voltage). Illustration from L2RPN2022 paper.

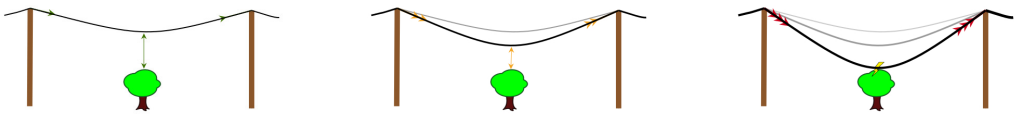


Figure 3: Overheating can cause accidents, since the heat lengthens the power lines.

2.2 Reinforcement learning

A way of learning and training individuals to find optimal behavior in an environment is by trial and error. This approach in machine learning is called *reinforcement learning (RL)*. RL is concerned with how agents ought to take actions in an environment to maximize their expected cumulative rewards (also called *return*). At each time step t, agent receives the state s_t (or observation if the entire state is not accessible) and a reward r_t, then agent takes a possible action a_t to receive the next state and reward (c.f. figure 4). The environment is typically stated in the form of a *Markov decision process (MDP)*, because many reinforcement learning algorithms for this context use dynamic programming techniques. A MDP is a discrete-time stochastic control process, and an extension of Markov chains, that provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. The main difference between the classical dynamic programming methods and reinforcement learning algorithms is that the latter do not assume knowledge of an exact mathematical model of the MDP and they target large MDPs where exact methods become infeasible. In RL, the agent has to find a balance between exploration (better knowing the environment) and exploitation (using current knowledge) [15].

Definition 2.1. A *Markov decision process* M is defined as a tuple $(\mathcal{S}, s_0, \mathcal{A}, P, R)$ where :

- $\mathcal{S}$ is the state space (either a finite set or a measurable space with its Borelian σ-algebra).
- s_0 is the initial state.
- $\mathcal{A}$ is the action space for the agent (either a finite set or a measurable space with its Borelian σ-algebra).
- $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability for states (*i.g.* $P(s'|s, a)$ is the probability that the next state is s' starting from the state s and doing action a).
- $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is the reward function for the agent that takes the current state and the action chosen by the agent in this state. This function can be a random variable or be represented as an expected reward $R(s, a) = \mathbb{E}_{r \sim \nu(s,a)}[r]$ where $\nu(s, a)$ is the reward distribution being in state s and executing action a. In this report, we consider the second case.

Definition 2.2. A *policy* π is a function that takes a state $s \in \mathcal{S}$ and returns a probability distribution function over the action space $\mathcal{A}$. The policy can be deterministic as a special case, in this framework.

Definition 2.3. *State-value function (or value function)* $V^\pi(s)$ is defined as the expected return starting with state s, i.e. $s_0 = s$, and successively following policy π. Hence, roughly speaking, the value function estimates "how good" it is to be in a given state [15]. We have :

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s\right]$$

with $\gamma \in [0, 1)$ the discount rate. Events in the distant future are weighted less than events in the immediate future. If γ is near 1, it means that distant future events still have an importance to the agent. If γ is near 0, it's the opposite and most of the importance is given to actions in the immediate future.
Although state-values suffice to define optimality, it is useful to define a measure for state-action pairs. Given a state s an action a and a policy π, the *state-action value* of the pair (s, a) under π is defined by :

$$Q^\pi(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a\right]$$

This function is also called Q-value function or Q-function for short.

The core objective of RL is to find a policy with maximum expected discounted return. From the theory of MDPs it is known that, without loss of generality, the search can be restricted to the set of so-called stationary policies. A stationary policy is a policy that doesn't change over time. Meaning, the same function π is applied for every state $s \in \mathcal{S}$ to get the action distribution $\pi(s)$. Following theorems give us the theoretical background for this statement.

Theorem 2.1. Bertsekas [2007]
Consider an MDP with a finite action space and an initial distribution β over states such that $|\{s \in \mathcal{S} : \beta(s) > 0\}| < \infty$. For any policy π, let $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$

$$p_t^\pi(s, a) = \mathbb{P}(S_t = s, A_t = a_t); \;\; p_t^\pi(s) = \mathbb{P}(S_t = s)$$

Then, for any history based policy π there exists a Markov policy $\overline{\pi}$ such that $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ and $\forall t \in \mathbb{N}^*$

$$p_t^\pi(s, a) = p_t^{\overline{\pi}}(s, a); \;\; p_t^\pi(s) = p_t^{\overline{\pi}}(s)$$

Proof. The proof is based on induction.
For any $\pi = (d_0, d_1, \dots)$ with d_t a randomized history dependent decision rule, let $\overline{\pi} = (\overline{d}_0, \overline{d}_1, \dots)$ be a randomized Markov policy such that

$$\overline{d}_t(a|s) = \frac{p_t^\pi(s, a)}{p_t^\pi(s)}$$

For any s, $p_0^\pi(s) = p_0^{\overline{\pi}}(s)$ by definition, and

$$p_0^{\overline{\pi}}(s,a) = p_0^{\overline{\pi}}(s)\overline{d}_0(a|s) = p_0^{\overline{\pi}}(s)\frac{p_0^\pi(s,a)}{p_0^\pi(s)} = p_0^{\overline{\pi}}(s)\frac{p_0^\pi(s,a)}{p_0^{\overline{\pi}}(s)} = p_0^\pi(s,a)$$

The base case is verified.
Now, we assume for some $t > 0$ and any state s, $p_t^\pi(s,a) = p_t^{\overline{\pi}}(s,a)$ and $p_t^\pi(s) = p_t^{\overline{\pi}}(s)$. So with P the transition probability as defined in definition 2.1,

$$\begin{aligned}
p_{t+1}^{\overline{\pi}}(s_{t+1}) &= \sum_{s_t,a_t} p_t^{\overline{\pi}}(s_t,a_t)P(s_{t+1}|s_t,a_t) \\
&= \sum_{s_t,a_t} p_t^{\overline{\pi}}(s_t)\overline{d}_t(a_t|s_t)P(s_{t+1}|s_t,a_t) \\
&= \sum_{s_t,a_t} p_t^{\overline{\pi}}(s_t)\frac{p_t^\pi(s_t,a_t)}{p_t^\pi(s_t)}P(s_{t+1}|s_t,a_t) \\
&= \sum_{s_t,a_t} p_t^\pi(s_t,a_t)P(s_{t+1}|s_t,a_t) \\
&= p_{t+1}^\pi(s_{t+1})
\end{aligned}$$

We can show that $p_{t+1}^\pi(s_{t+1},a_{t+1}) = p_{t+1}^{\overline{\pi}}(s_{t+1},a_{t+1})$ with the same reasoning.

□

NB : In the proof, we considered finite state space, but the theorem is also true for continuous state space and the proof is analogous to the one given here.
The theorem below demonstrates that Markov policies are as expressive as history based policies, at least for the finite action space case.

Theorem 2.2. Bertsekas [2007]
Consider an MDP with a finite action space and an initial distribution β over states such that $|\{s \in \mathcal{S} : \beta(s) > 0\}| < \infty$. For any policy π, let $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$

$$\rho_\gamma^\pi(s,a) = (1-\gamma)\sum_{t\geq 0}\gamma^t\mathbb{P}(S_t = s_t, A_t = a_t)$$

$$\rho_\gamma^\pi(s) = (1-\gamma)\sum_{t\geq 0}\gamma^t\mathbb{P}(S_t = s_t)$$

called the discounted occupancy measure and state discounted occupancy measure respectively. Then, for any policy π, there exists a stationary policy $\overline{\pi}$ such that $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$ and $\forall t \in \mathbb{N}^*$,

$$\rho_\gamma^\pi(s,a) = \rho_\gamma^{\overline{\pi}}(s,a); \quad \rho_\gamma^\pi(s) = \rho_\gamma^{\overline{\pi}}(s)$$

Proof. We have, with P the transition probability as defined in definition 2.1,

$$\begin{aligned}
\rho_\gamma^{\overline{\pi}}(s) &= (1-\gamma)\sum_{t\geq 0}\gamma^t \mathbb{P}(S_t = s) \\
&= (1-\gamma)\beta(s) + (1-\gamma)\sum_{t\geq 1}\gamma^t \mathbb{P}(S_t = s) \\
&= (1-\gamma)\beta(s) + (1-\gamma)\gamma\sum_{t\geq 1}\gamma^{t-1}\sum_{s',a}\mathbb{P}(S_{t-1}=s', A_{t-1}=a)P(s|s',a) \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s'}(1-\gamma)\sum_{t\geq 1}\gamma^{t-1}\mathbb{P}(S_{t-1}=s')\sum_a \overline{\pi}(a|s')P(s|s',a) \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s'}(1-\gamma)\sum_{t\geq 1}\gamma^{t-1}\mathbb{P}(S_{t-1}=s')p^{\overline{\pi}}(s|s') \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s'}\rho_\gamma^{\overline{\pi}}(s')p^{\overline{\pi}}(s|s')
\end{aligned}$$

with $p^{\overline{\pi}}(s|s') = \sum_a \overline{\pi}(a|s')p(s|s',a)$. Using matrix formulations, with

$$[\boldsymbol{\rho}_\gamma^{\overline{\pi}}]_s = \rho_\gamma^{\overline{\pi}}(s)$$
$$[P^{\overline{\pi}}]_{s,s'} = p^{\overline{\pi}}(s'|s)$$

we have

$$\begin{aligned}
\rho_\gamma^{\overline{\pi}}(s) &= (1-\gamma)\beta(s) + \gamma\sum_{s'}\rho_\gamma^{\overline{\pi}}(s')p^{\overline{\pi}}(s|s') \\
\Rightarrow \boldsymbol{\rho}_\gamma^{\overline{\pi}} &= (1-\gamma)\beta(s) + \gamma P^{\overline{\pi}}\boldsymbol{\rho}_\gamma^{\overline{\pi}} \\
\Rightarrow \boldsymbol{\rho}_\gamma^{\overline{\pi}} &= (1-\gamma)\beta(s)(I-\gamma P^{\overline{\pi}})^{-1}
\end{aligned}$$

Since the eigenvalues of a stochastic matrix P^π belong to $[0,1]$, consequently, $-\gamma \notin span(P^\pi)$, thus, $(I-P^\pi)$ is invertible.
For any non-stationary policy π, we define a stationary policy $\overline{\pi}$

$$\overline{\pi}(a|s) = \frac{\rho_\gamma^\pi(s,a)}{\rho_\gamma^\pi(s)}$$

We have with analogous calculations as before

$$\begin{aligned}
\rho_\gamma^\pi(s) &= (1-\gamma)\beta(s) + \gamma\sum_{s',a}(1-\gamma)\sum_{t\geq 1}\gamma^{t-1}\mathbb{P}(S_{t-1}=s', A_{t-1}=a)P(s|s',a) \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s',a}\rho_\gamma^\pi(s',a)P(s|s',a) \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s',a}\overline{\pi}(a|s')\rho_\gamma^\pi(s')P(s|s',a) \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s'}\rho_\gamma^\pi(s')\sum_a\overline{\pi}(a|s')P(s|s',a) \\
&= (1-\gamma)\beta(s) + \gamma\sum_{s'}\rho_\gamma^\pi(s')p^{\overline{\pi}}(s|s')
\end{aligned}$$

So, with matrix formulations we have

$$\begin{aligned}
&\boldsymbol{\rho}_\gamma^\pi = (1-\gamma)\beta(s) + \gamma P^{\overline{\pi}} \boldsymbol{\rho}_\gamma^\pi \\
\Rightarrow &\boldsymbol{\rho}_\gamma^\pi = (1-\gamma)\beta(s)(I - \gamma P^{\overline{\pi}})^{-1} \\
\Rightarrow &\boldsymbol{\rho}_\gamma^\pi = \boldsymbol{\rho}_\gamma^{\overline{\pi}}
\end{aligned}$$

□

If two policies have the same discounted occupancy measure, they have the same value function because

$$\begin{aligned}
V^\pi(s) &= \mathbb{E}\left[\sum_{t\geq 0} \gamma^t R(s_t, a_t)\right] \\
&= \sum_{t\geq 0} \gamma^t \sum_{s,a} R(s,a)\mathbb{P}(S_t = s, A_t = a) \\
&= \frac{1}{(1-\gamma)} \sum_{s,a} (1-\gamma) \sum_{t\geq 0} \gamma^t R(s,a)\mathbb{P}(S_t = s, A_t = a) \\
&= \frac{1}{(1-\gamma)} \sum_{s,a} \rho_\gamma^\pi(s,a) R(s,a)
\end{aligned}$$

Hence, the previous theorem proves that stationary policies (with finite action spaces at least) can "generate" any value function (consequently any q-function). Therefore, the search can be restricted to the set of stationary policies.
NB : The same theoretical guarantees can be obtained for continuous state and action spaces by discretizing them.
Modern RL algorithms seek either to approximate the value or Q functions or directly find the best policy π_θ that maximize the discounted return by adjusting the parameters θ (typically neural nets' parameters).

Definition 2.4. A policy $\pi_{\theta'}$ is *better* than π_θ (noted $\pi_{\theta'} \geq \pi_\theta$), iff we have $\forall s \in \mathcal{S},\ V^{\pi_{\theta'}}(s) \geq V^{\pi_\theta}(s)$.

Definition 2.5. A policy π_* is *optimal* iff $\forall \pi,\ \pi_* \geq \pi$

Generally, policy based RL algorithms seek to *improve* the agent's policy and to maximize the associated value function for every state s. On the other hand, value or Q function based algorithms seek to estimate the true value or Q function, most of the time by bootstrapping techniques (see [15]).

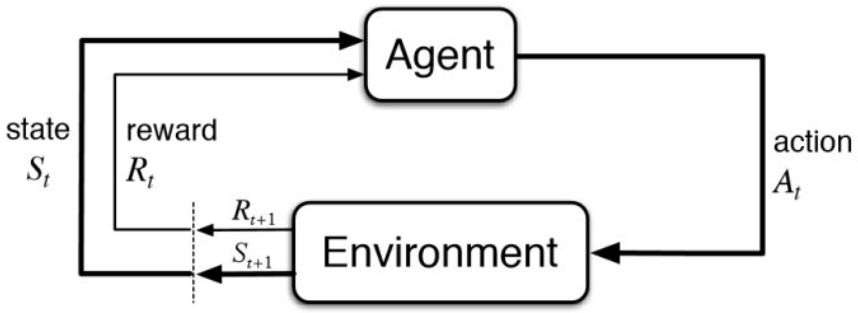


Figure 4: The typical framing of a Reinforcement Learning (RL) scenario: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.

2.3 Game theory

Game theory is the study of mathematical models of strategic interactions among rational agents. Originally, it addressed two-person zero-sum games, in which each participant's gains or losses are exactly balanced by those of the other participants. It has been extended to N-player games in which players are not necessarily in competition and can collaborate to maximize their respective gains.
The two most prominent mathematical models of games are *Stochastic Games* [1] and *Extensive Form Games (EFGs)* (see Figure 5) [7]. We are in particular interested by stochastic games. A stochastic game, introduced by Lloyd Shapley in the early 1950s, is a repeated game played by one or more players (or agents in the RL literature). The game is played in a sequence of stages or steps in RL. At the beginning of each step the game is in some state. The agents select their actions simultaneously and each player receives a reward that depends on the current state and the chosen actions. The game then moves to a new random state whose distribution depends on the previous state and the actions chosen by the agents. The procedure is repeated at the new state and play continues for a finite or infinite number of steps. The total reward to a player is often taken to be the discounted sum of the step rewards or the limit inferior of the averages of the step rewards.
Stochastic games generalize Markov decision processes to multiple interacting decision makers.

2.3.1 POSG (Partially Observable Stochastic Games)

In many use cases, agents cannot observe the entire state of the environment. So, we are in a partially observable configuration. We will always assume that our games are Markovian, *i.e.* the transition depends solely on the current state and the action profile. In a POSG, all agents step together, observe together, and are rewarded together.

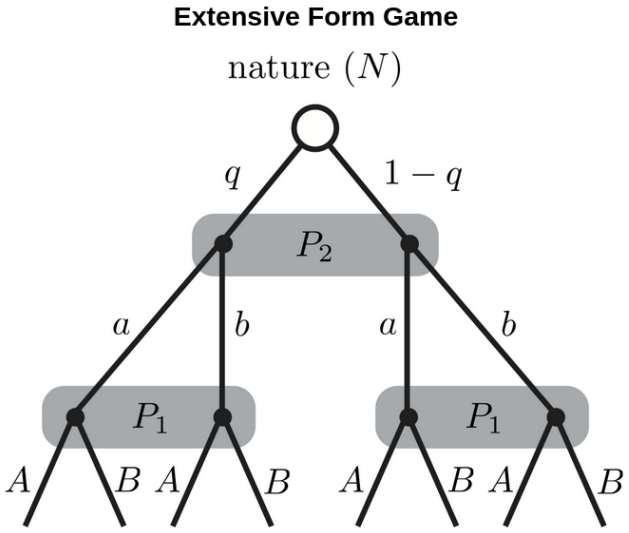


Figure 5: An extensive for game's representation with the nature (player 0) in the root and other players playing round by round with actions represented as edges. Illustration from [19]

Definition 2.6. A *Partially Observable Stochastic Game* (POSG) G is defined as a tuple $\big(N, \mathcal{S}, s_0, (\Omega_i)_{i\in[N]}, (O_i)_{i\in[N]}, (\mathcal{A}_i)_{i\in[N]}, P, (R_i)_{i\in[N]}\big)$, where :

- N is the number of agents. The set of agents is $[N]$.
- $\mathcal{S}$ is the state space (either a finite set or a measurable space with its Borelian σ-algebra).
- s_0 is the initial state.
- Ω_i is the observation space for agent i (either a finite set or a measurable space with its Borelian σ-algebra).
- $\mathcal{A}_i$ is the action space for agent i (either a finite set or a measurable space with its Borelian σ-algebra).
- $P : \mathcal{S} \times \prod_{i\in[N]} \mathcal{A}_i \times \mathcal{S} \to [0,1]$ is the transition probability for states.
- $O_i : \mathcal{A}_i \times \mathcal{S} \times \Omega_i \to [0,1]$ is the observation probability($i.g.$ $O_i(\omega|s, a_i)$ is the probability that agent i observes ω in the state s after action a_i).
- $R_i : \mathcal{S} \times \prod_{i\in[N]} \mathcal{A}_i \to \mathbb{R}$ is the reward function for agent i. This function can be a random variable or be represented as an expected reward $R_i(s, a_i, a_{-i}) =$

$\mathbb{E}_{r\sim\nu(s,a_i,a_{-i})}[r]$ where $\nu(s, a_i, a_{-i})$ is the reward distribution in the state s and action a_i by the agent i and the action profile a_{-i} which represents actions by other agents except i. Here, we consider the second case.

This model has made it much easier to apply single agent RL methods to multi-agent settings. However, there are two immediate problems with this model:

1. Supporting strictly turn-based games like chess requires constantly passing dummy actions for non-acting agents (or using similar tricks).
2. Changing the number of agents for agent death or creation is very awkward, as learning code has to cope with lists suddenly changing sizes.

2.3.2 AEC (Agent Environment Cycle)

POSG models have some practical issues. As mentioned before, it's difficult to implement turn-based games. In addition, agents do not observe other agents' actions since, they act all at the same time. Another important issue is that rewards are hard to track as these combined rewards are often the composite rewards from the actions of other agents.
To tackle this issues, Agent Environment Cycle (AEC) games [19] have been introduced. In addition, it's also possible to define EFGs as AEC games.

Definition 2.7. An *Agent Environment Cycle* (AEC) game G is defined as a tuple $\big(N, \mathcal{S}, s_0, (\Omega_i)_{i\in[N]}, (O_i)_{i\in[N]}, (\mathcal{A}_i)_{i\in[N]}, P, (T_i)_{i\in[N]}, (R_i)_{i\in[N]}, \nu\big)$, where :

- N is the number of agents. The set of agents is $[N]$. There is also an additional "environment" agent, denoted as agent 0. We denote the set of agents along with the environment by $\mathcal{N} := [N] \cup 0$.
- $\mathcal{S}$ is the state space (either a finite set or a measurable space with its Borelian σ-algebra).
- s_0 is the initial state.
- Ω_i is the observation space for agent i (either a finite set or a measurable space with its Borelian σ-algebra).
- $\mathcal{A}_i$ is the action space for agent i (either a finite set or a measurable space with its Borelian σ-algebra).
- $P : \mathcal{S} \times \mathcal{S} \to [0, 1]$ is the transition function for the environment. State transitions for environment steps are stochastic: $P(s, s')$ is the probability that the environment transitions into state s' from state s.
- $T_i : \mathcal{S} \times \mathcal{A}_i \to \mathcal{S}$ is the transition function for agents. State transitions for agent actions are deterministic.

- $O_i : \mathcal{A}_i \times \mathcal{S} \times \Omega_i \to [0,1]$ is the observation probability(*i.g.* $O_i(\omega|s,a_i)$ is the probability that agent i observes ω in the state s after action a_i).
- $R_i : \mathcal{S} \times \mathcal{N} \times \prod_{i\in[N]} \mathcal{A}_i \times \mathcal{S} \times \mathcal{R}_i \to \mathbb{R}$ is the reward function for agent i. $\mathcal{R}_i \subseteq R$ denotes the set of all possible rewards for agent i (which we assume to be finite). R_i is the reward function for agent i. $R_i(s,j,a,s',r)$ is the probability of agent i receiving reward r when agent j takes action a while in state s, and the game transitions to state s'.
- $\nu : \mathcal{S} \times \mathcal{N} \times \mathcal{A}_i \times \mathcal{N} \to [0,1]$ is the next agent function. This means that $\nu(s,i,a,j)$ is the probability that agent j will be the next agent permitted to act given that agent i has just taken action a in state s. This should attribute a non-zero probability only when $a \in \mathcal{A}_i$.

AEC games aim to be an alternative to POSGs in some cases. So, we need some theoretical guarantees. By prooving that any AEC game can be defined as a POSG and vice-versa, we have the same guarantees as POSGs by equivalence.

Theorem 2.3. AEC games are equivalent to POSGs, i.e., any AEC game can be defined as a POSG and vice-versa.

Proof. [19] □

2.4 Multi-agent reinforcement learning

In the multi-agent reinforcement learning (MARL for short), we are in a POSG framework. Every agent has its own reward function R_i and it seeks to maximize the cumulative discounted rewards associated to this function. Hence, for every agent i, we define

$$V_i^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_i(s_t,a_t)|s_0 = s\right]$$

and

$$Q_i^\pi(s,a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R_i(s_t,a_t)|s_0 = s, a_0 = a\right]$$

with $\pi = (\pi_1,\dots,\pi_N)$ the policy profile for all agents and $a_t = (a_t^1,\dots,a_t^N)$ the action profile for all t, such that $\forall i \in [N],\ a_t^i \sim \pi_i$. Here, we considered (will consider) same discount rate γ for all agents, but we can go further and consider agents with different discount rates γ_i.
In this case, multiple agents try to learn with others. If we put ourselves in a local perspective and count other agents as part of the environment, we have an non-markovian environment [8] that poses challenge to the standard approach of learning from exploration [15]. Agents can easily mistake other agents' exploration as environment randomness [5]. For example, an agent's "good" action can be masked by another agent's "very

bad" action, hence, the reward will be poor. In this context, agents must optimize their policy conditionally to other agents' policies.
In some cases, especially in collaborative games, mediation and/or communication mechanisms can improve greatly agents' performance[9]. For example, in our case we could have a control center that sends guidelines to agents and they take actions based on those. In this scenario, agents can also communicate among themselves to avoid incompatible actions.

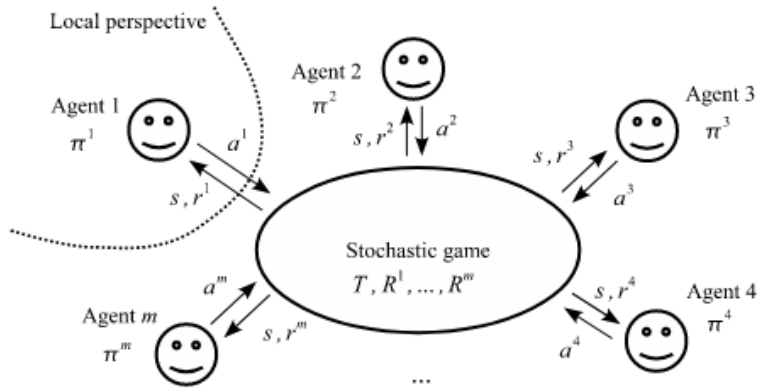


Figure 6: The local perspective, i.e. counting other agents as part of the environment, is not markovian. Illustration from [8]

3 Environment

In a RL framework, the environment's model is central. It must verify some essential properties (e.g. Markov) to be relevant and ensure RL algorithms convergence. In particular, current state and agents' actions should be sufficient for the environment's transition.
In this section, Grid2Op and its multi-agent version are presented. We will also argue about Grid2Op (resp. multi-agent Grid2Op) being a MDP (resp. POSG).

3.1 Grid2Op

The Grid2Op framework[18], which is open-source and easily re-usable, allows users to define new environments with its companion GridAlive ecosystem. Grid2Op relies on existing non-linear physical power network simulators and let users create a series of perturbations and challenges that are representative of two important problems: a) the uncertainty resulting from the increased use of unpredictable renewable energy sources, and b) the robustness required with contingent line disconnections.

3.1.1 Power grid components

In a power grid (see Figure 7 for an example of power grid represented in Grid2Op), there are in particular loads (power consumers e.g. houses, factories, etc.), generators (power producers e.g. nuclear plants, renewable energies, etc.), storage units (any type of energy storage facility e.g. batteries, pumped water, etc.) and power lines. These objects are connected together in some places called *substation* (see Figure 8). Substations are connected together by power lines (see Figure 8). Substations have *bus bars* (see black lines in Figure 9). It entails that we can "split" a substation into different independent "buses". A *bus* is a term from the power system community meaning that: "if two objects are on the same bus, then there exist a direct electrical path in the substation connecting them". In 3.1.4 we will see in depth the usefulness of substations and possible actions on bus bars.

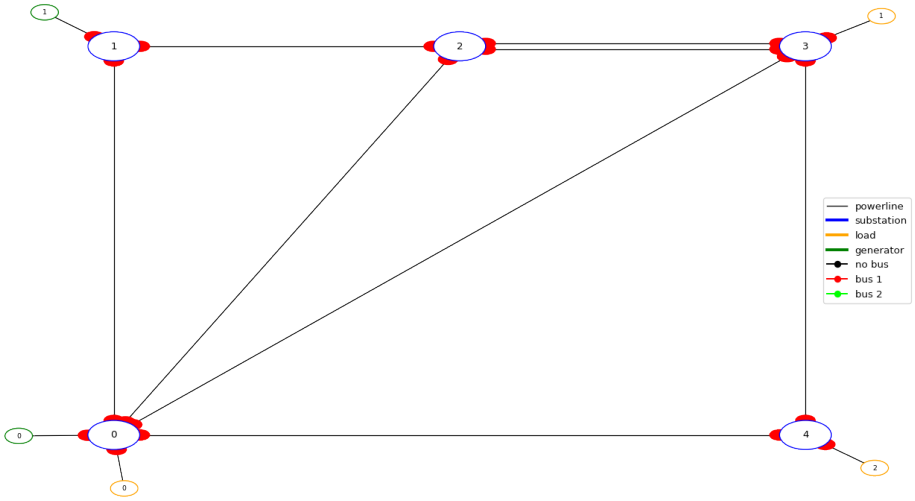


Figure 7: A simple power grid represented in Grid2Op.

Figure 8: Substations and power lines in reality. Source : Grid2Op notebooks in GitHub.

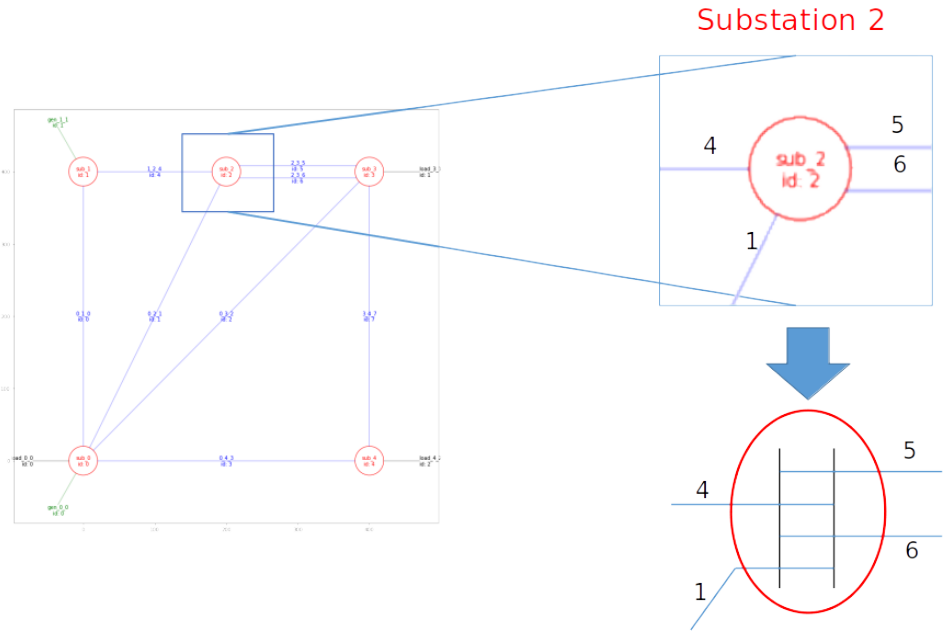


Figure 9: In Grid2Op, substations have 2 bus bars, but in reality, substations can have up to 6 bus bars. This simplification is because the number of possible actions grows exponentially with the number of bus bars in substations.

3.1.2 Power grid as a graph

A power grid can be seen as a connected graph with components as nodes and power lines as edges. Another representation is by taking connections as nodes only, meaning each bus can become a node as long as it has at least one power line connected (see Figures 9 & 10). Considering that, the power grid operation becomes a complex temporal graph problem. Instead of inferring a given graph properties as in well-known graph problems, we try to find the right graphs over time, among all possible topologies (see Figure 12 for different possible topologies in a substation), given the properties that would optimize the flows for safe network operations such as no line overflow.

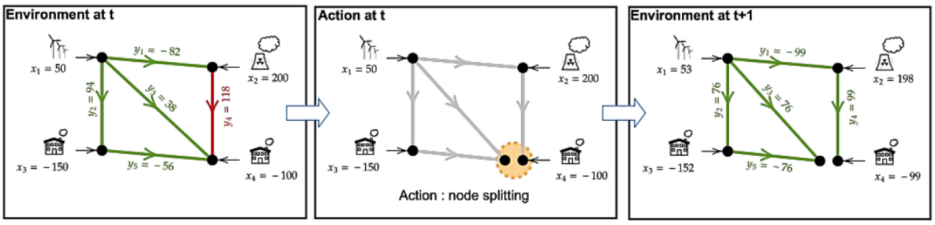


Figure 10: A topological action is a change on power grid's graph. Illustration from [18].

3.1.3 Observations

In order to make relevant decisions, agents need relevant observations over the grid. These observations can be *complete*, *incomplete* or *noisy*. We will only consider complete observations to stay in the MDP case.
These observations contain the following data:

- General information about the grid : number of objects (load, line, etc.) and number of elements connected to each substations.
- Temporal information : observations have calendar data (day of week, minute of hour, hour of day, etc.) to represent time which is very important in consumption predictions (more consumption implies more possible overloads in the grid).
- Physical information : active/reactive production/consumption, current, voltage, capacity (ρ), etc. The capacity is defined as the ratio between the current going through the line (in Ampere) and the maximum current allowed for this line. In a nominal situation rho must remain under 1.
- Connections : substations' and bus's id to which each object is connected, connectivity matrix, etc.
- Restrictions and future disruptions/changes : unavailable substations/lines because of a recent action, future maintenance on a power line, etc.

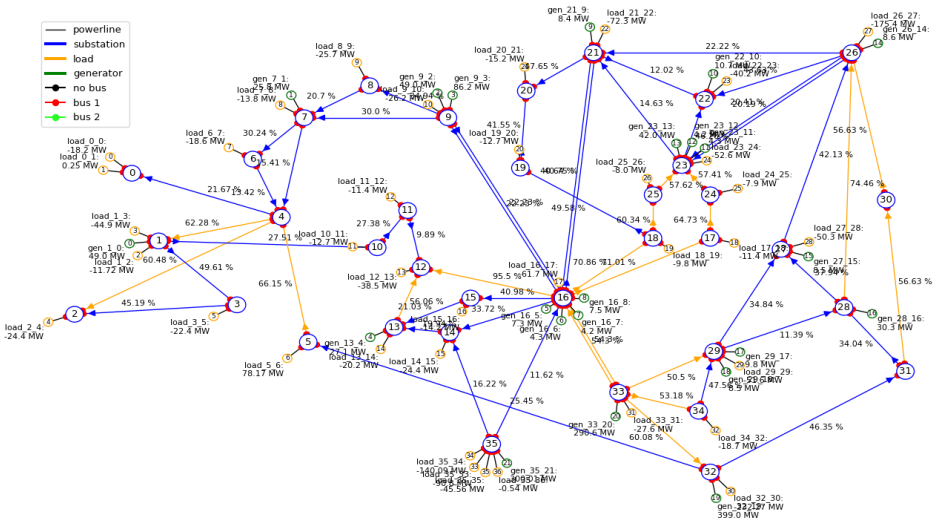


Figure 11: A representation of a power grid showing power lines' capacity with loads and generators' active flows.

3.1.4 Actions

There are different types of action possible in power grids. They are presented below even if we will not use them all in the policies investigated.

- Topological and power line actions : the most important and the most used type of action on power grids. It consists of changing the topological configuration of the power grid. In other words, we change the power grid's graph with these actions (see 3.1.2 and Figure 10). These are done by switching bus bars in substations that allow different configurations (see Figure 12). These actions are considered cheap and fast, since there is no need to change the electricity generation which can be costly and slow (think of changing a nuclear plant's production). However, the number of all possible topologies in a power grid grows exponentially with respect to number of substations, objects and bus bars (see Table 1). Nevertheless, these grids are quite small compared to the real French power grid with more than 10,000 substations. Fortunately, we can reduce these numbers. By adding some operational constraints, like only one substation and power line can be changed per time step, and omitting symmetrical actions, the total number of actions are reduced drastically. Thus, we obtain the Table 2's figures.

- Production actions : This class of action intends to change the production level (in MW) of a generating plant (ask for more or less power). All these actions are continuous. There are two types for production actions. The first one is redispatching that consists to ask a dispatchable generator to increase or reduce its production. Dispatchable generators are all those that can vary their productions, e.g. nuclear plants, hydropower, etc. Of course, each generator has its limit that can not be surpassed. As it is mandatory to ensure that production and consumption are equal at the grid level, any change in the production level of a plant must be caught up by other generators to re-balance the production. This process can be done manually by the user or automatically by the environment. These actions are costly and slow. Each generator has its limits in term of changes per time step, so required modifications can take many steps to be fully executed. Plus, starting up or shutting down a plant has consequent economical costs. In this work, we do not consider these costs in the reward function, but they are highly considered in operators' decision making process and should be integrated in future AI decision makers as well.

  The second type of production actions is curtailment. It consists of limiting the production on a renewable source of energy that is not dispatchable (solar, wind, etc.). As mentioned in the introduction, RTE seeks to prepare itself for the green transition and contribute in the process. Hence, limiting the green energy generation is not really desired. CO2 emissions are not taken into account in this work, but it is part of RTE's mission to reduce it.

  To conclude, production actions are sensitive and should not be executed very often. In the power grid operation, they are used when there are no alternatives.

Actions must be legal and unambiguous to be executed, otherwise, the environment will simply ignore them. Beside constraints, in terms of number of changed power lines and substations per time step, an action should not divide the grid's graph into two (or more) connected graphs. Furthermore, an action should not isolate a generator or a load. Plus, changes on production side must be clear and meet physical capacities.

Number of possible actions by environment without constraints				
Environment's name	Substations	Power lines	Total number of topologies	Total number of power line actions
rte case5 example	5	8	31,320	256
l2rpn case14	14	20	1,397,519,564	1,048,576
l2rpn wcci 2020	36	59	1.88 e+21 (*)	5.76 e+17 (*)
IEEE case 118	118	186	3.88 e+76 (*)	9.81 e+55 (*)

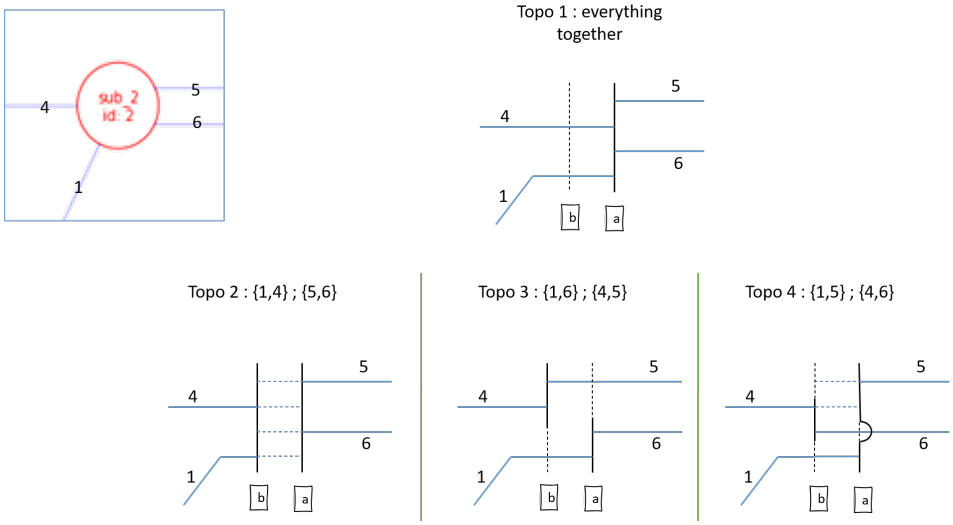
Table 1: (*) based on approximations. Source : Grid2Op notebooks in GitHub.

Number of possible actions by environment with constraints				
Environment's name	Substations	Power lines	Total number of topologies	Total number of power line actions
rte case5 example	5	8	117	8
l2rpn case14	14	20	179	20
l2rpn wcci 2020	36	59	66,811	59
IEEE case 118	118	186	72,150	186

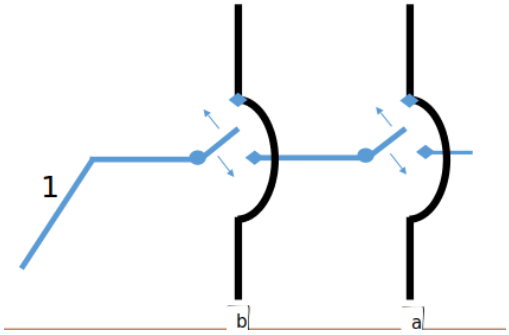
Table 2: By adding operational constraints, like only one substation and power line can be changed per time step, and omitting symmetrical actions, the total number of actions are reduced drastically. Source : Grid2Op notebooks in GitHub.

3.1.5 Rewards

There are multiple rewards defined in the Grid2Op environment. Here, we only consider two of them.
The first is the survival. For every time step in which the power grid is operational, the agent receives a reward of one. Being operational for a power grid means that we do not have a black out and every consumer has the energy load it needs. Plus, the power grid must follow some rules. For example, the power grid's graph must be connected. Although this is a classical reward in the RL literature, it is not very informative in our case. Hypothetically, we can imagine two actions. One reduces the overflow in the grid and the other overload all power lines. With the survival reward function, they both have the same reward, even if it is clear that first action is way better.



(a)



(b)

Figure 12: Different possible *topologies* for this particular substation (a). These topologies are created by switching power lines in the substation (b).

This is the reason why we introduce a second reward function. It is defined as the sum of squared margins on each power lines. For a line l, the margin $m_l(s)$ in the state s is defined as $m_l(s) = (1 - \rho_l(s))_+$ with ρ_l the line's capacity. So the reward function is

$$\begin{aligned} R(s,a) &= \mathbb{E}_{s' \sim P(.|s,a)} \left[\sum_l m_l(s')^2 \right] \\ &= \sum_{s' \in \mathcal{S}} \sum_l m_l(s')^2 P(s'|s,a) \end{aligned}$$

In the remainder of this document, we take this reward function by default.

3.1.6 Markov property in Grid2Op

In our case, each observation is the grid's state that contains all information needed to solve the physical equations integrated within the environment for a transition towards the next state. The consumption data are generated from a time series that follows some scenario (called chronic). Since the time is already integrated in the observation (or state), we abusively assume this data generation does not depend on past states or the dependence is weak. In addition, past actions are not needed for the transition neither. Indeed, instantaneous actions (topological actions and curtailments) are already present in the state via connections and power generation data. And if an action lasts on many iterations (e.g. redispatching), future changes (or remaining changes to apply) are represented in the state.
Hence, we can conclude that environment's transition does not depend on past states and actions and the Markov property holds in Grid2Op.

3.2 Multi-agent Grid2Op

We aim to design a suitable multi-agent environment. In our use case, defining a POSG is the most relevant. In reality, operators take their actions independently in their region, and do not wait to see other operators' actions. In addition, computing and resolving physical equations in the grid is computationally intensive. Nevertheless, AEC might be interesting in the training phase, as it is designed for a better reward tracking. In this report, only POSG is treated.
Multi-agent Grid2Op is a *wrapper* that takes a classical Grid2Op environment and agents' domains to add additional multi-agent functionalities. The domain is defined as the set of all substations under agent's control. To avoid conflicts and ambiguities, it's forbidden to have any intersection between domains, and every substation must be assigned to an unique agent. In other words, domains $\mathcal{D}_i$ form a partition of the set of all substations in the grid (see Figure 13). Here, we just considered a spatial partitioning, but other options are possible. An interesting alternative to this may be task partitioning. For example, one agent can be in charge of topological actions, while the other is in charge of production actions. Note that finding a relevant partition is a

hard task that deserves an entire internship for itself. In RTE, there has been multiple researches on the matter using various techniques, including machine learning [13]. Each agent i's actions space $\mathcal{A}_i$ is induced and created using the global action space $\mathcal{A}$

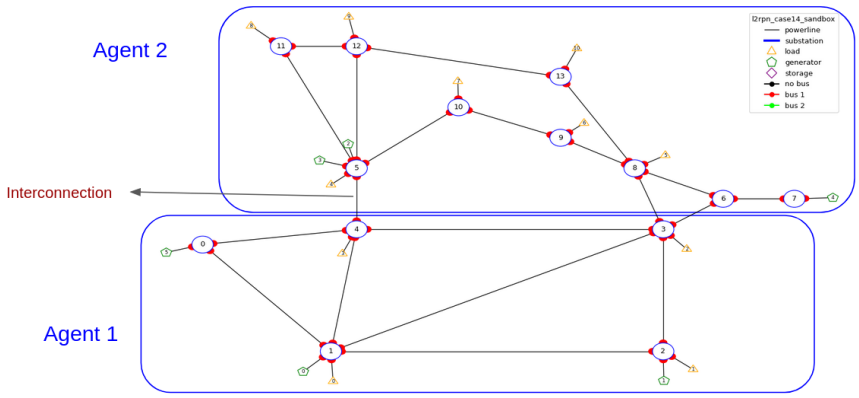


Figure 13: An example of grid partitioning. Power lines that connect two domains are called *interconnections*.

and agent's domain $\mathcal{D}_i$. The same for observation spaces, if we use local observations. In this report, we only consider global and complete observations. Hence, all agents observe the state s_t at each time step t. A direct consequence of this partitioning is the emergence of boundaries. These are power lines that connect two substations in two different domains. As they influence both domains, they could be source of conflicts as it is discussed in 3.2.2.

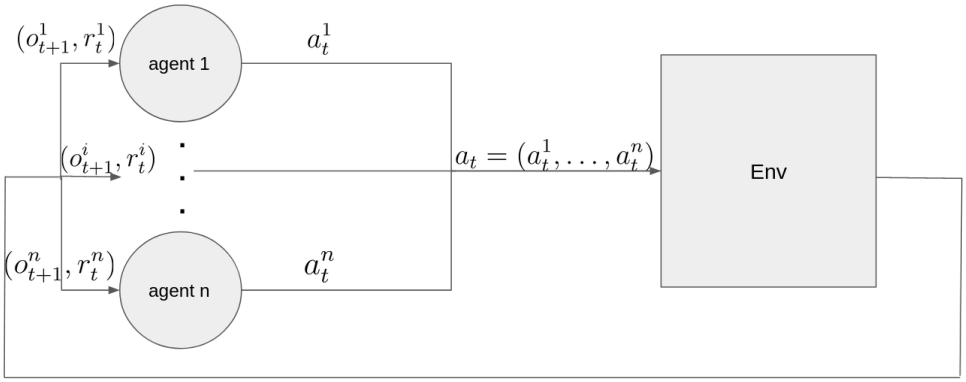


Figure 14: The execution cycle of a multi-agent environment.

3.2.1 Multi-agent Grid2Op as a Dec-MDP

Definition 3.1. A Dec-MDP is defined as a tuple $\big(N, \mathcal{S}, s_0, (\mathcal{A}_i)_{i\in[N]}, P, R\big)$ where :

- N is the number of agents. The set of agents is $[N]$.
- $\mathcal{S}$ is the state space (either a finite set or a measurable space with its Borelian σ-algebra).
- s_0 is the initial state.
- $\mathcal{A}_i$ is the action space for the agent i (either a finite set or a measurable space with its Borelian σ-algebra).
- $P : \mathcal{S} \times \prod_{i\in[N]} \mathcal{A}_i \times \mathcal{S} \to [0, 1]$ is the transition probability for states (*e.g.* $P(s'|s, a)$ is the probability that the next state is s' starting from the state s and doing action profile $a = (a_1, \ldots, a_N)$).
- $R : \mathcal{S} \times \prod_{i\in[N]} \mathcal{A}_i \to \mathcal{R} \subset \mathbb{R}$ is the reward function for the agent that takes the current state and the action profile chosen by agents in this state. This function can be a random variable or be represented as an expected reward $R(s, a) = \mathbb{E}_{r\sim\nu(s,a)}[r]$ where $\nu(s, a)$ is the reward distribution being in state s and executing action profile $a = (a_1, \ldots, a_N)$. In this report, we consider the second case.

Dec-MDP is a particular version of POSGs. Observations are complete and agents have the same reward function. Consequently, agents have an interest in collaboration.
So, for the first version of the multi-agent environment, we decided to design it as a Dec-MDP to guarantee the collaboration and to avoid divergent interests among agents. Hence, all agents have a complete and global observation, which is the state, and they have a common reward function. But, they act locally in their domain. Each agent takes an action conditionally to the state, then, the action profile is given to the environment. Afterwards, the environment combines local actions to create a Grid2Op global action that can be executed. Hence, our multi-agent environment is equivalent to a classical single agent environment (with some changes in rules and constraints like number of changed substations and power lines) in which the agent returns the latter action. This insures the Markov property in the multi-agent environment. Consequently, the latter is a Dec-MDP. Once the global action executed, the reward and the new state is sent to all agents (see Figure 14).

3.2.2 Conflict handling

Conflict handling is one of the most important topics in a multi-agent environment's design. Conflicts can provoke bugs and corrupt experiments which is an important issue to consider. More importantly, they can disturb the learning and decision making process by making illegal actions that are originally legal, or induce undesired behavior.

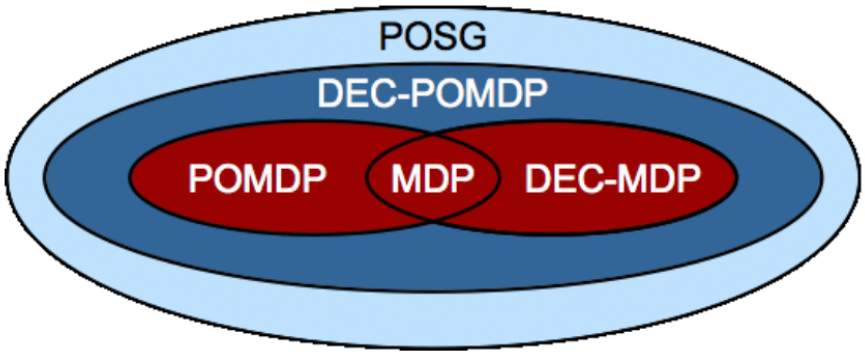


Figure 15: Different processes used in the RL literature.

For example, if an agent decides to disconnect an interconnection and the other decides to keep it connected, we have two contradictory actions and it is not obvious which one is right. In reality, operators can communicate with each other before any action, which is not the case, for now, in our environment. Hence, we need a conflict handling mechanism.
The first measure to avoid conflicts was the definition of domains previously described. The second measure is banning connection/disconnection actions on interconnections. These mechanisms may change as new features are added. For example, instead of banning connection actions on interconnections, we can think of a priority based system, in which the agent with higher priority has the last word in conflicts.

4 Experiments

Here, different experiments are presented. First experiments are done to verify the consistency of the freshly designed multi-agent environment. This step is very important, since the fact that having multiple agents operating in the grid instead of a single one, should not change the underlying dynamics and functionalities. We must also verify these new functionalities added are correctly working. It means local actions must be executed accordingly in the right substation and on the right objects.
Once the consistency is verified, we begin using different agents on the environment to compare their performances in single and multi agent cases. These are also compared to the "do nothing" policy (as a simple benchmark). Indeed, doing nothing may be a great strategy in power grids. In the past L2RPN competitions [1], many top performer algorithms intervene only in critical situations and do nothing the vast majority of time[18].
All experiments are done with the Grid2Op environment based on a grid configuration

[1]Since 2019, RTE has been organizing an annual AI competition called L2RPN (Learning to Run a Power Network) with the topic of developing an agent capable of operating an increasingly complex simulated power grid. See https://l2rpn.chalearn.org/ for the 2022 edition.

named *"l2rpn case14 sandbox"* with 14 substations. We kept this environment because it represents a power grid with a reasonable size, in which, operating is hard enough, but without too high computational costs.

4.1 Do nothing, random dummy actions and parrot agents

For instance, if we have only "do nothing" agents, no matter their head count, we should have the same observations, rewards and episode length in the classical Grid2Op environment and in its new multi-agent version. We checked that carefully before starting any experiment.
The second phase of verification was executing random dummy actions (over all possible actions) and see if they are correctly executed in the right domain and on the right object. We also check that illegal actions are not executed and replaced by do nothing. This phase was the random version of our previous unit tests when developing the multi-agent environment. Once again, these have passed.
Now we can have our first agents. The first type of agent is a "parrot agent" that executes a list of actions associated to some contexts. For example, if some power line is overloaded, it executes the action designed to treat this situation. The objective is to test an agent that takes an observation and makes a decision based on it. At the same time, we test if the observations are correctly treated, and relevant actions are executed based on them. After trying this agent on multiple episodes, we observed that parrot agents are not able to outperform the "do nothing" agent in both single and multi agent cases. But, we observed the wanted behavior in terms of actions executed.

4.2 Heuristic agents

Let $\mathcal{T}$ ($\mathcal{T}_i$ for every agent i) be the set of topological actions with $\mathcal{T} \subset \mathcal{A}$ ($\mathcal{T}_i \subset \mathcal{A}_i$). An agent i has the policy π_i is

$$\pi_i(o_i|a_{-i}) = \arg\max_{a_i \in \mathcal{T}_i} \widehat{R}_i(o_i, a_i, a_{-i})$$

with o_i the agent i's observation (which is equal to the state in our case), and a_{-i} other agents' action profile, and $\widehat{R}_i$ the simulated reward function (Grid2Op functionality). This agent tries and simulate every action and takes the one with highest simulated reward (see Figure 16). In other words, this agent is a greedy agent that aims to maximize the instantaneous reward with $\gamma = 0$.
Here, we only consider topological actions for computational reasons. Indeed, only one topological action per agent can be executed, so the search space is of cardinal $|\mathcal{T}_i|$. If we add dispatching and curtailment actions, the search space cardinal become $|\mathcal{T}_i| \times |\mathcal{D}_i|^{n_{dg}} \times |\mathcal{C}_i|^{n_{re}}$ with $\mathcal{D}_i$ and $\mathcal{C}_i$ discretized dispatching and curtailment spaces for agent i, and n_{dg} and n_{re} number of dispatchable and renewable energy generators in the agent's domain respectively. For example, if we have only one dispatchable generator and one renewable energy, and we discretize corresponding action spaces with 10 bins, the total actions to simulate is multiplied by 100 every single step. This has a significant impact in experiments and slow down the process.

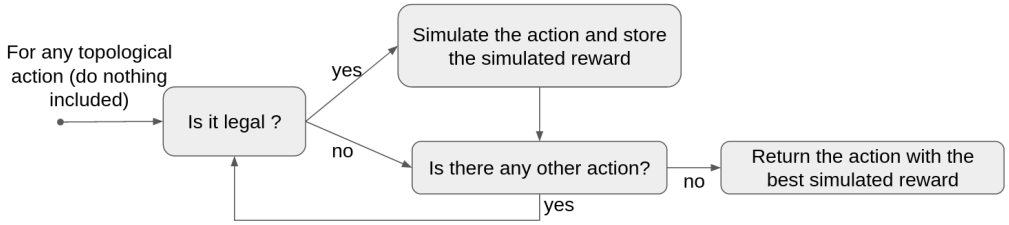


Figure 16: Heuristic agents' functioning

4.2.1 Do nothing assumption

However, this policy is conditional to a_{-i} which is unknown. Hence, every agent i needs to make assumptions on what other agents will do. First assumption that we can have in mind is what we call the "*do nothing assumption*", meaning that every agent will assume that others do nothing. So, the policy i is

$$\pi_i(o_i|a_{-i}=0) = \arg\max_{a_i \in \mathcal{T}_i} \widehat{R}_i(o_i, a_i, a_{-i})$$

with 0 the index of the do nothing action and the abusive notation $a_{-i} = 0$ that means all actions except a_i are zero (or do nothing for instance). For the rest of this report, we will keep this notation. This assumption is not far from reality since agents do nothing must of the time (see Figure 17).

4.2.2 Guess/predict others' actions

Instead of assuming that others are doing nothing, agents can try to predict others' actions. To do so, every agent i needs a predictor f_{-i} that takes other agents' observations and returns their actions. As we assume all agents have the same complete observation o which is equal to the environment's state, we do not need to know their observations. So, we have $f_{-i} : \mathcal{S} \to \mathcal{T}_{-i} = \prod_{j \neq i} \mathcal{T}_j \subset \mathcal{A}_{-i}$. Agent i can have a predictor for every agent j in which case $f_{-i}(o) = (f_j(o))_{j \neq i}$ for $o \in \mathcal{S}$, with $f_j : \mathcal{S} \to \mathcal{T}_j$ agent j's predictor. Now, the policy is

$$\pi_i(o|a_{-i} = f_{-i}(o)) = \arg\max_{a_i \in \mathcal{T}_i} \widehat{R}_i(o, a_i, f_{-i}(o))$$

A possible option for predictors, is to learn agent's behavior. To do so, we can think of machine learning algorithms that are trained by learning previous multi-agent experiments with do nothing assumption. Multiple algorithms have been tested (e.g. random forest, gradient boosting, etc.) with different hyper parameters via grid search and cross validation to select the best estimator. The main problem that we meet here is the overwhelming presence of null actions (or do nothing actions) that represents more than 95% for agent 1's and 90% for agent 2's set of actions. In this situation, the constant predictor equal to zero has a precision of 95% and 90% respectively. This leads to a high

focus on these null actions as Table 3 shows. To resolve this issue, we need to reduce the share of these actions in datasets. This process is called *downsampling*. To do so, we keep all non-zero labeled data and we take randomly a selection of null actions. Here, the share of null actions is limited to 80%.
As shown in Table 3, this method led to a significant improvement in terms of precision for non-zero actions. Although, the precision overall and over null actions decreased. Nonetheless, it isn't a problem since those impressive performances were doped by the strong presence of null actions. In fact, while the best predictor is 2% better than the constant predictor for agent 1 before downsampling. The same predictor is 10% better than the latter after downsampling (the constant predictor's precision is 80%). For agent 2, the best predictor is 2% better than the constant predictor before, and nearly 4% better after downsampling. Hence, we can conclude that, overall, the downsampling process improved our predictors' precision and performance.
NB : Reducing furthermore this share does not improve accordingly the precision over non-zero labeled data according to our experiments.
NB2 : The agent 2's predictor's under-performance compared to the agent 1's can be explained by 2 facts. First, agent 2 has more possible topological actions and we do not have enough non-zero labeled samples (it is also the case for agent 1). Secondly, agent 2's non-zero actions are more diverse that means we have fewer samples per action which worsen the first problem. To resolve this problem, we need more samples, but data gathering is costly. For example, to run heuristic agents with do nothing assumption for 20 episodes, two days of computation[2] was needed because of the exhaustive search for the best action.

Prediction precision before and after downsampling				
	Before		After	
	Agent 1	Agent 2	Agent 1	Agent 2
Precision overall	**97.09%**	**91.98%**	87.99%	83.17%
Precision over do nothing actions	**99.91%**	**99.55%**	97.68%	97.91%
Precision over non-zero actions	37.44%	30.69%	**50.29%**	**37.53%**

Table 3: The performance of the best estimator (random forest) before and after downsampling. Non-zero actions are all topological actions except do nothing.

4.2.3 Results

Multiple combinations have been tried for heuristic agents :

- (1) Both agents are is using the do nothing assumption

[2]On a computer with a 12 core CPU and 32 GB of RAM.

- (2) Agent 1 is using the do nothing assumption and agent 2 tries to predict agent 1's action
- (3) Agent 2 is using the do nothing assumption and agent 1 tries to predict agent 2's action
- (4) Both agents try to predict other's action befire taking their own action

These combinations are compared to a "do nothing agent" and a single heuristic agent. All experiments were over 20 episodes with 20 different chronics (scenarios) with 2 agents for the multi-agent case. To be fair and without bias, all experiments were based on the same scenarios. We analyzed both rewards (see Figure 18) and actions (see Figure 17). In terms of rewards, best results in mean and median came from (1). We detected poor results for cases (3) and (4), which can be explained by the predictor's lack of accuracy (see Figure 3). As for (2), the performance had more variance than other methods.
In terms of actions, we observe a stable "do nothing rate" throughout different methods. Both agents act occasionally on the environment, between 3% and 6% for agent 1, and between 10% and 13% for agent 2. However, we acknowledge more interventions in the case (4). Another interesting fact is that when they both predict, they tend to act more in the environment. We can also see that actions are particularly concentrated in two substations 1 and 5 (see Figures 19 and 20). Indeed, the substation 1 is in charge of a nuclear plant (the most important generator in this grid). The substation 5 has a lot of connections and connected power lines are often overloaded. Overall, there is a strong correlation between the average overload of connected power lines in a substation and the number of actions made on that power line, which is natural.

4.2.4 Possible improvements

To improve agents with prediction, predictors should be improved. To do so, we need more data that is provided by past simulations with agents assuming do nothing. These simulations are costly in time, hence, to be feasible they should be optimized and parallelized, which we omitted in this work because of a lack of time.
In addition, for results to be statistically relevant, we need many more experiments than 20. Once again, the issue stated bellow needs to be resolved to have more experiments. Nonetheless, heuristic agents have some limitations, especially when it comes to scale up the grid. For a small grid of 14 substations, experiments took days to be completed, even with an optimized computation, these agents are hardly operational for grids with hundreds, even thousands of substations. Nevertheless, there is a possibility if the set of actions to be simulated is reduced to a set of "relevant actions". These actions can be selected by an "expert agent" using different methods (heuristic or AI). It is also possible to take a number of actions that have the best q-values that are given by a RL agent. A method that has been used by many participants in L2RPN challenge.

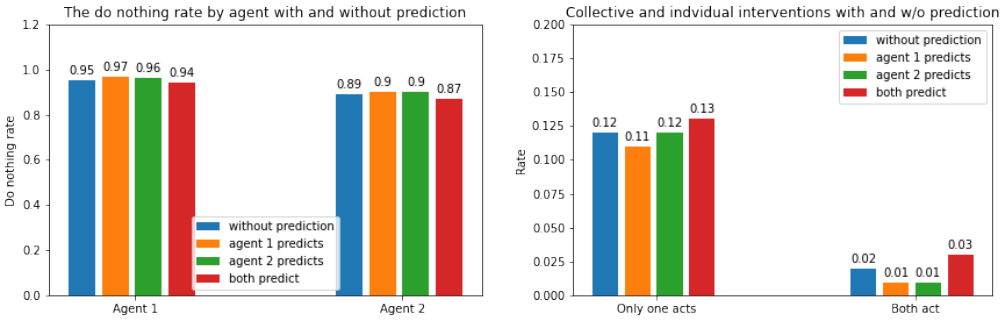


Figure 17: (Left) The do nothing rate per agent in different assumptions. (Right) The collective and individual intervention rate. The prediction of other agents' actions didn't have a significant impact on the do nothing rate and it remained stable through different experiments. On the other hand, when one agent tries to predict the other agent's action, it tends to act less often (collective actions are cut by half). But, when both try to predict, agents intervene more often and we acknowledge a significant rise in collective actions.

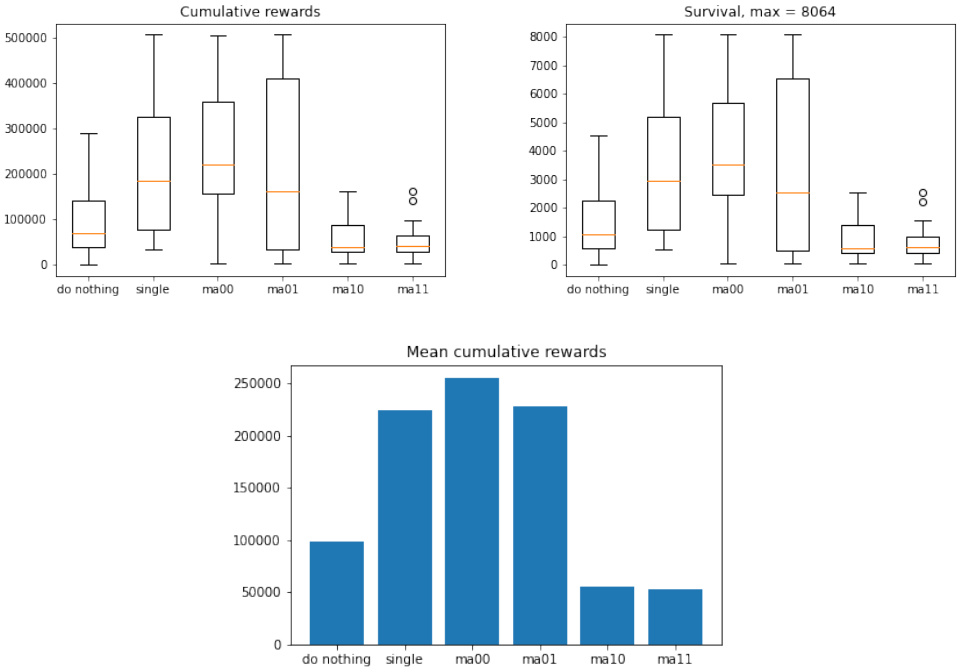
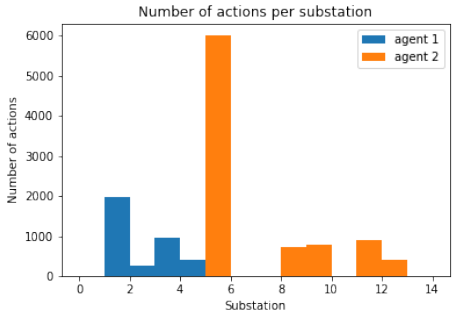
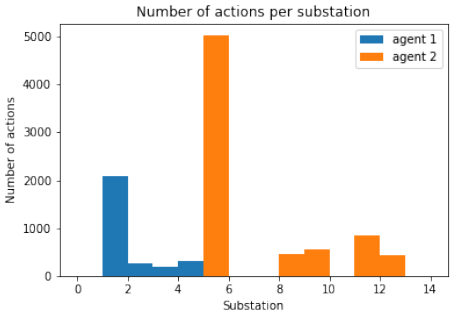


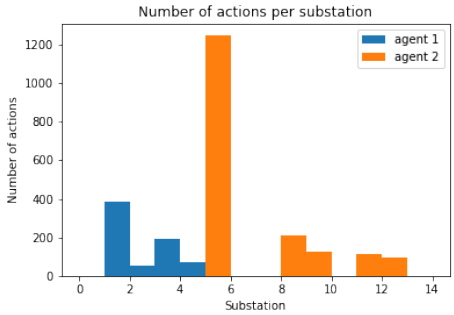
Figure 18: Results for heuristic agents. "ma00" means both agents are under do nothing assumption, "ma01" means agent 1 (see Figure 13) is under do nothing assumption and agent 2 tries to predict agent 1's action, "ma10" is the reverse of the latter, and "ma11" means both try to predict other's action.



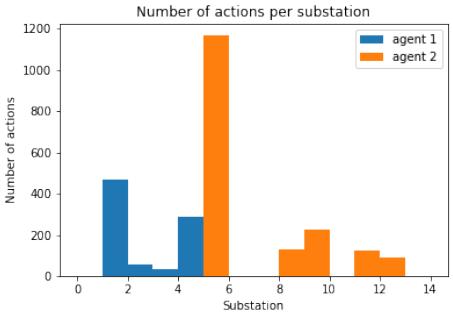
(a) Both with do nothing assumption



(b) Only agent 2 predicts

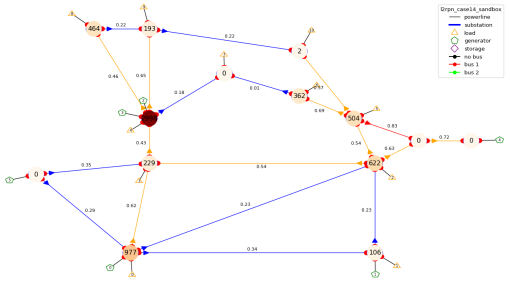


(c) Only agent 1 predicts

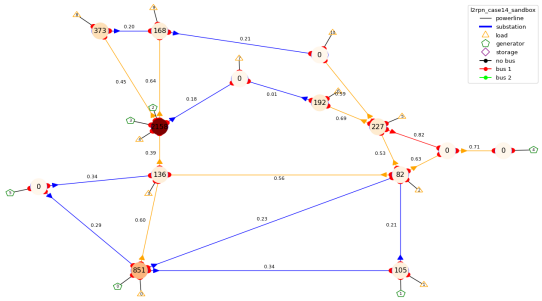


(d) Both predict

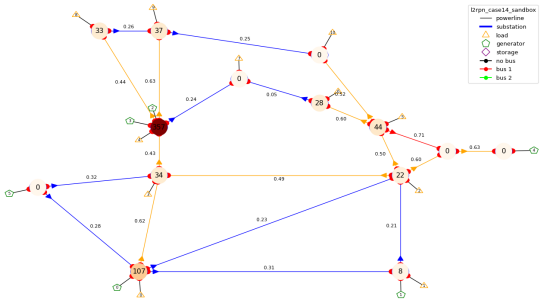
Figure 19: Number of actions per substation. Two substations, in particular, concentrate most of actions (1 & 5).



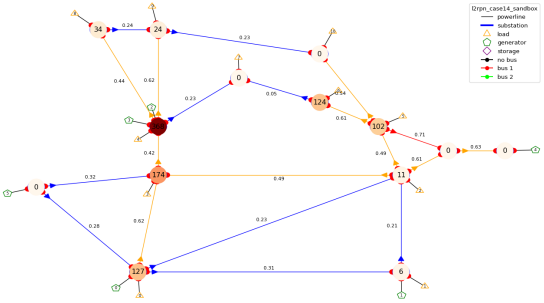
(a) Both with do nothing assumption



(b) Only agent 2 predicts



(c) Only agent 1 predicts



(d) Both predict

Figure 20: Number of actions per substation and average overload ρ per power line. Two substations, in particular, concentrate most of actions (1 & 5). Substation 5 has a lot of objects connected and its generators are renewable (generators 2 & 3). On the other hand, substation 1 is in charge of the only nuclear generator in the grid (generator 0).

4.3 Deep RL agents

Using deep learning methods scaled up RL methods performance. AlphaGo was one of first major agents using them, and since it has been used nearly in every RL agent for other use cases. It consists of using artificial neural networks (ANN or simply NN) to compute value and/or q and/or policy functions and to optimize some objective function. This class of functions is interesting because NNs are universal approximators[2][4][6], and thanks to GPUs, they are quite fast to compute since they use mainly matrix multiplications. We use PPO [11] for the single agent case, and for the multi-agent case, PPO and MADDPG[10].
However, using the term "deep" might be a bit abusive, since we use at most 3 hidden layers. We test different hyper parameters, in particular, the number of hidden layers and neurons in each layer. Nonetheless, because of the lack of time, we are not able to tune these hyper parameters to find the best.

4.3.1 Results

We train deep RL agents over 1000 episodes/epochs. As shown in Figure 21, single agents barely surpassed the do nothing threshold. We observe that, with 3 hidden layers, the learner have a remarkable performance before collapsing. With 2 layers, however, we have more stability but the learner doesn't improve significantly after the first 200 episodes.
On the other hand, their multi-agent versions had very bad performances. Indeed, they learned a bit at the beginning and they stopped learning, and apparently got stuck in a local optimum. This illustrates the difficulty of learning in a multi-agent environment, in which agents influence each other's actions, thus disrupting the learning process.

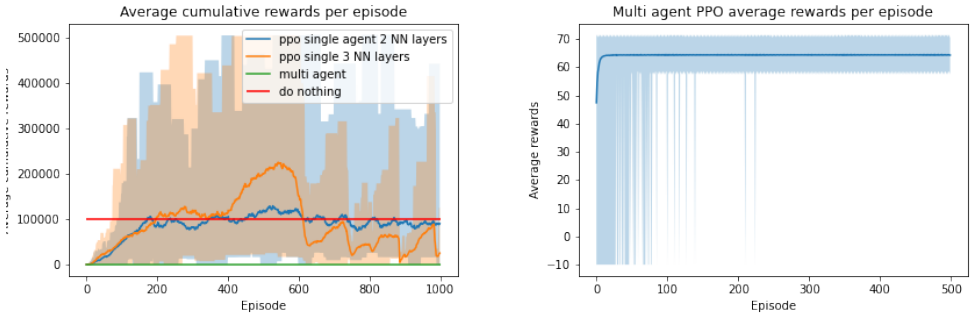


Figure 21: Average rewards per episode/epoch. The red line represents the average rewards obtained by doing nothing in the grid, an important threshold to surpass that justifies the use of the corresponding algorithm.

4.3.2 Possible improvements

Tuning hyper parameters could improve results, nevertheless, it is unlikely to change drastically multi-agent's case's performance. Other methods could be tested, e.g. [14].
As stated before, a hybrid approach federating heuristic and deep RL methods is more likely to succeed as they can learn and have some guarantees.
Using communication between agents might help to improve performance too, especially in cooperative games[9].

5 Conclusion

This work has been mainly exploratory. Indeed, multi-agent reinforcement learning has not yet been significantly tested for the operation of a power grid. These are the first experiments at RTE.
Power grid control is difficult, since actions can have a great impact on the grid, and in some cases, provoke blackouts in a domino effect. In this context, learning can be challenging and the convergence of the different methods and algorithms is far from being assured in this real world problem.

As our experiments (and many before) show, using heuristic methods require consequent computing power and is costly in time and energy, but yields some interesting results in terms of rewards well above the basic "do nothing" policy.
Heuristic agents (with a do nothing assumption, see 4.2.1), in a multi-agent case, outperform slightly the single agent case. By dividing the grid, we reduce the number of possible actions per agent and we can speed up consequently the computation by parallelization. Despite this possibility, as the number of actions grows greatly with number of elements and substations, all these optimizations might not be enough to have a viable solution for large power grids. Moreover, these heuristic algorithms do not learn (or learn hardly) in time, hence, they do not improve themselves by interacting with their environment (i.e. the grid2Op simulator).

On the other hand, a straightforward application of deep reinforcement learning algorithms resulted in rather poor performance compared to heuristic agents - and even the very basic "do nothing" agent (in the single agent and in the multi-agent setting) achieved a better score. Our experiments so far have shown that these "pure" reinforcement learning algorithms, despite their sophistication, did not learn much, as their reward improvement stopped just after a few episodes.

In this context, an interesting lead would be a hybrid approach with more sophisticated (domain / expert) rules than a simple end to end deep reinforcement learning. Furthermore, the design of a communication mechanism, learnable or not, between agents seems necessary and is highly recommended in future work.

References

[1] Lloyd S Shapley. "Stochastic games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[2] Kurt Hornik. "Approximation capabilities of multilayer feedforward networks". In: *Neural networks* 4.2 (1991), pp. 251–257.

[3] Louis Wehenkel and Mania Pavella. "Decision trees and transient stability of electric power systems". In: *Automatica* 27.1 (1991), pp. 115–134.

[4] Moshe Leshno et al. "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function". In: *Neural networks* 6.6 (1993), pp. 861–867.

[5] Caroline Claus and Craig Boutilier. "The dynamics of reinforcement learning in cooperative multiagent systems". In: *AAAI/IAAI* 1998.746-752 (1998), p. 2.

[6] Allan Pinkus. "Approximation theory of the MLP model in neural networks". In: *Acta numerica* 8 (1999), pp. 143–195.

[7] Ross Cressman, Christopher Ansell, and Ken Binmore. *Evolutionary dynamics and extensive form games*. Vol. 5. MIT Press, 2003.

[8] Guillaume J Laurent, Laëtitia Matignon, Le Fort-Piat, et al. "The world of independent learners is not Markovian". In: *International Journal of Knowledge-based and Intelligent Engineering Systems* 15.1 (2011), pp. 55–64.

[9] Jakob Foerster et al. "Learning to communicate with deep multi-agent reinforcement learning". In: *Advances in neural information processing systems* 29 (2016).

[10] Ryan Lowe et al. "Multi-agent actor-critic for mixed cooperative-competitive environments". In: *Advances in neural information processing systems* 30 (2017).

[11] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[12] Ioannis Konstantelos et al. "Using vine copulas to generate representative system states for machine learning". In: *IEEE Transactions on Power Systems* 34.1 (2018), pp. 225–235.

[13] Antoine Marot et al. "Guided machine learning for power grid segmentation". In: *2018 IEEE PES Innovative Smart Grid Technologies Conference Europe (ISGT-Europe)*. IEEE. 2018, pp. 1–6.

[14] Tabish Rashid et al. "Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning". In: *International conference on machine learning*. PMLR. 2018, pp. 4295–4304.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[16] Adrian Kelly et al. "Reinforcement learning for electricity network operation". In: *arXiv preprint arXiv:2003.07339* (2020).

[17] Siyuan Chen et al. “Active power correction strategies based on deep reinforcement learning—part II: A distributed solution for adaptability”. In: *CSEE Journal of Power and Energy Systems* (2021).

[18] Antoine Marot et al. “Learning to run a power network challenge: a retrospective analysis”. In: *NeurIPS 2020 Competition and Demonstration Track*. PMLR. 2021, pp. 112–132.

[19] Jordan K. Terry et al. “Pettingzoo: Gym for multi-agent reinforcement learning”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15032–15043.

[20] Jonas Degrave et al. “Magnetic control of tokamak plasmas through deep reinforcement learning”. In: *Nature* 602.7897 (2022), pp. 414–419.

A Predictors' hyper parameters

Hyper parameters	Agent 1	Agent 2
Type	Random forest	Random forest
Criterion	log loss	log loss
Min samples to split	6	8
Number of estimators	150	250

Table 4: All unspecified hyper parameters are default values on scikit learn.

B RL agents hyper parameters

Hyper parameters	PPO	MADDPG
Number of SGD iterations per training mini-batch	10	-
Hidden layers	[100, 50] (2 layers experiment) [200, 100, 100] (3 layers experiment)	[200, 100, 100]
Value function clip param	100	-
Actor learning rate	5e-5	5e-3
Critic learning rate	5e-5	5e-3

Table 5: All unspecified hyper parameters are default values on RLLib.