Università degli studi di Trieste

Dipartimento di Matematica e Geoscienze

Master's Degree in
Data Science and Scientific Computing

Optimizing fault search in power grid outages through Reinforcement Learning

Candidate:

Angela Carraro

Supervisor:

Antonio Celani, ICTP

Co-Supervisors:

Andrea Zancola, AcegasApsAmga

Luca Bortolussi, UniTS

Academic year 2020/2021

Abstract

Riassunto in italiano.

Contents

Acronyms

AAA	AcegasApsAmga S.p.A.. 1, 23
AWS	Amazon Web Services. 18
BFS	Breadth-first search. 28, 29
DFS	Depth-first search. 29
DTP	Decision-Theoretic Planning. 6
MDP	Markov Decision Process. 6, 7, 10, 11, 16
ML	Machine Learning. 5
POMDP	Partially Observable Markov Decision Process. 10–12, 16, 18, 24
RL	Reinforcement Learning. 5, 6, 9, 10

Chapter 1

Introduction

1.1 The problem

AcegasApsAmga S.p.A. (AAA) is a company based in Trieste and subject to the direction and coordination of Hera S.p.A., called also Hera Group, which is a multiutility company based in Bologna, Italy. Hera operates in the distribution of gas, water, energy, and waste disposal in some Italian provinces.

The project involves AAA Trieste power grid.

1.2 The power grid

Describe the power grid and all its elements.

Chapter 2

Data Management

2.1 Introduction

2.2 Raw to bronze

2.3 Bronze to silver

2.3.1 The circuit graph

2.3.2 The electrical graph

2.3.3 The substations graph

2.4 Bisection

2.5 Weighted bisection

Chapter 3

Reinforcement Learning

3.1 Introduction

As described in [1], Reinforcement Learning (RL) is a paradigm of Machine Learning (ML), along with others, like supervised learning and unsupervised learning. It analyzes how it is possible to learn the best course of action in an environment based on maximizing some given numerical reward. The learner is not told which actions to take, but learns empirically which are the ones that lead to a bigger reward. It is also possible that the current action influences not only the immediate reward, but also the next status of the environment, and, through that, all the other rewards to come. In this case, we speak of delayed reward, which, together with trial-and-error search, form the two distinctive features of RL.

In an RL problem we have a learning *agent*, the decision-maker, that interacts over time with the *environment* in which it is placed — which includes everything outside the agent — in order to achieve a *goal*, which is to maximize the total reward it receives over the long run. As we see in Figure 3.1, the agent can sense aspects of its environment through the *states* — which can also be seen as a representation of the environment itself, it can choose *actions* to influence the environment, and it receives a *reward* based on the outcome of its actions. We consider to be a reinforcement learning method any method that is well suited

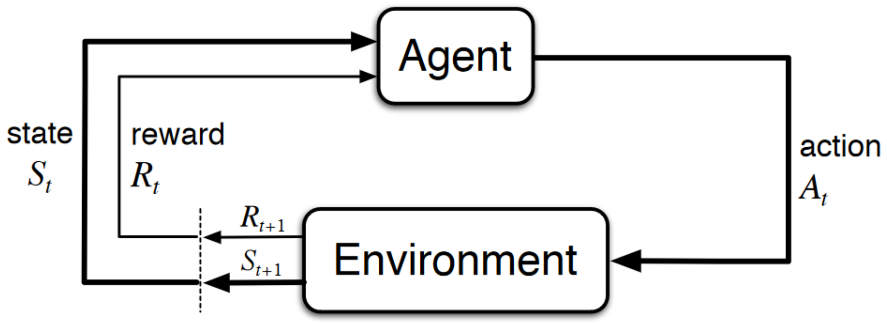


Figure 3.1: The agent-environment interaction [1]

to solving problems framed in this way. Other elements of the RL system are a *policy*, a *value function*, and a *model* of the environment. A *policy* describes the behavior of the agent at a given time, using a mapping from states to actions to be taken in those states. A *value function* determines the value of a state as the total amount of reward an agent can expect to accumulate over the future, starting from that state. Instead, a *model* of the environment reproduces the behavior of the latter, and it is used for planning actions before actually experiencing them. For example, given to the model a state and action, it might predict with a certain probability the resultant next state and next reward.

3.2 Finite Markov Decision Processes

The treatment of this topic will closely follow the one presented in [1].

Markov Decision Processes, or MDPs, are used to formalize sequential decision-making, where actions influence the given rewards and the next states of the system, and through the latter also future rewards can be affected. Thus, MDPs need to balance both immediate and delayed rewards. This formalism is used both in decision-theoretic planning (DTP), RL, and other learning problems in stochastic domains [2].

We assume that the process evolves through a sequence of discrete time steps, $t = 1, 2, 3, \ldots$, even if it is possible to extend to the continuous case. These steps do not need to reflect fixed intervals of real time, but they can refer to arbitrary successive stages of decision-making and acting. At each time step t, the agent detects the *state* of the environment, $S_t \in \mathcal{S}$, based on which it selects an action $A_t \in \mathcal{A}(s)$ (if the action set is the same in all states, we will write it simply as $\mathcal{A}$). Then, as a consequence, the agent receives, one time step later, a numerical reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$, and ends up in a new state S_{t+1}. Thus, given this sequential process, it emerges a sequence, or *trajectory*, given from

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, \dots \tag{3.1}$$

In a *finite* MDP we have that the set of states $\mathcal{S}$, actions $\mathcal{A}$, and rewards $\mathcal{R}$ have finite cardinalities, respectively $|\mathcal{S}|, |\mathcal{A}|$, and $|\mathcal{R}|$.

Since it is a *finite* MDP, the random variables R_t and S_t have well-defined discrete probability distributions, and since we are considering a *Markov* process, it holds the *Markov property*, also called *memorylessness property*, and thus the *dynamics* of the process through the space of states, or the *model* of the environment, depend only on the preceding state and action:

$$p(s', r \mid s, a) := \Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}\,, \tag{3.2}$$

for all $s', s \in \mathcal{S}$, $r \in \mathcal{R}$, and $a \in \mathcal{A}(s)$.

One could compute also the *transition probabilities*, defined (with a little abuse of notation, since we continue to use the letter p) as

$$p(s'|s, a) := \Pr\{S_t = s' \mid S_{t-1} = s, A_{t-1} = a\} = \sum_{r \in \mathcal{R}} p(s', r \mid s, a)\,. \tag{3.3}$$

The *expected rewards* for state–action–next-action triples can be computed as $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to R$,

$$r(s, a, s') := \mathbb{E}\left[R_t | S_{t-1} = s, A_{t-1} = a, S_t = s'\right] = \sum_{r \in \mathcal{R}} r \cdot p(s', r|s, a)\,. \tag{3.4}$$

The MDP can also be seen as the tuple $(\mathcal{S}, \mathcal{A}, p, r)$, where $r = r(s, a, s')$ is the expected immediate reward that the agent receives for taking action a in state s [3, 4].

To describe how likely it is for an agent to take any given action from any given state, we use the notion of policy. A *policy* is a class of probability distributions $\pi(a|s)$ over an action $a \in A(s)$ for each state $s \in \mathcal{S}$:

$$\pi(a|s) = \Pr\{A_t = a | S_t = s\} \,, \tag{3.5}$$

which describes the probability that the agent chooses that action being in that state at that time step t.

The agent's goal is to maximize the *expected return*, o *cumulative reward*, it receives in the long run given a certain policy π:

$$\mathbb{E}[G_t] = \mathbb{E}_{p(\cdot,\cdot|s_t,a_t),\pi}\left[\sum_{t=0}^{\infty} \gamma^t R_t\right] . \tag{3.6}$$

where $\gamma \in [0,1]$ is the *discount factor*, and it determines how much we value future rewards. With $\gamma = 0$ the agent considers only immediate rewards and discards future ones, while with $\gamma = 1$ the agent will take into account, in the same way, every reward it receives, considering a very long series of events.

Actually, we can take this average with respect to the distribution of rewards themselves, so as to ignore the dependency on the distribution of rewards. This is a distinctive property of the fact that we are working with a *known model of the environment*. So, since we are only interested in optimizing averages, we do not care any longer about what is the actual distribution of the rewards. Thus, we can rephrase the agent goal as

$$\mathbb{E}[G_t] = \mathbb{E}_{p(\cdot|s_t,a_t),\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s,a,s')\right] . \tag{3.7}$$

To reach its goal, the agent must estimate how good it is to be in a given state if it is following a certain policy. This value is stored in the *(state-)value function*. Formally, the value of state s under policy π is

$$V_\pi(s) := \mathbb{E}\left[G_t \mid S_t = s\right] = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^k R_t \,\middle|\, S_t = s\right] , \text{ for all } s \in \mathcal{S} \,, \tag{3.8}$$

which is the expected return when starting in state s and following policy π thereafter. If there exists a terminal state, which causes the process to terminate if reached, its value is always zero.

Similarly, the value of taking action a in state s under policy π is stored in the *(state-)action value function*, or *quality*, which is expressed with

$$\begin{aligned} Q_\pi(s,a) &:= \mathbb{E}\left[G_t \mid S_t = s, A_t = a\right] \\ &= \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^k R_t \,\middle|\, S_t = s, A_t = a\right], \end{aligned} \tag{3.9}$$

which is the expected return starting from state s, taking action a and following policy π thereafter.

We have that the following relation holds:

$$V_\pi(s) = \sum_a \pi(a|s) Q_\pi(s,a)\,. \tag{3.10}$$

The value function possesses a fundamental property, which is extensively used in RL: it satisfies the following recursive relation

$$\begin{aligned} V_\pi(s) &= \sum_a \pi(a|s) \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V_\pi(s')\right] \\ &= \sum_a \pi(a|s) \sum_{s'} p(s' \mid s, a)\left[r(s,a,s') + \gamma V_\pi(s')\right]\,, \end{aligned} \tag{3.11}$$

which is called *Bellman equation for* V_π.

Similarly, we have the *Bellman equation for* Q_π:

$$\begin{aligned} Q_\pi(s,a) &= \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')\right] \\ &= \sum_{s'} p(s' \mid s, a)\left[r(s,a,s') + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s',a')\right]\,. \end{aligned} \tag{3.12}$$

3.3 Partially Observable MDPs

The presentation of this topic will mainly follow the one in [5].

What happens if the agent does not know anymore with full certainty the state of the environment, maybe due to imperfections or limitations in the agent's sensors, or to errors in the interpretation of the environment, or simply to unknown aspects of the environment itself?

If some features of the state are hidden from the agent, and the latter can sense only a part of the real state of the environment, the resultant state signal will no longer be Markovian, breaking a key assumption of most RL methods, like the MDP formulation. Luckily, we can consider an extension of the (fully observable) MDP framework, that can deal with the uncertainty originating from the imperfect states perceived by the agent or the uncertain action effects.

A *partially observable Markov decision process*, or POMDP, is a framework that considers environments that are only partially observable to the agent, but allows anyway for optimal decision-making [3], contrary to the requirement of full knowledge of the environment of MDPs.

The fact that the environment is partially observable can derive mainly from two reasons:

- different states generate the same observation, due to the same sensor reading, caused by the agent's limited knowledge of the environment;
- sensor readings are noisy, so the same state can generate different observations due to different sensor readings.

Thus, we have that the state of the environment is not uniquely identified by the agent's observations, and situations that appear similar to the agent may require instead different actions.

Interestingly, we can consider fully observable MDPs as a special case of POMDPs, in which the observation function maps each state to its correct unique observation deterministically [6].

Since a POMDP is an extension of an MDP, they share many elements. Also in POMDPs the time is divided into different steps, and in each one of them the agent has to take an action. As for the MDPs, we will consider discrete and finite models, which are simpler than the continuous ones. So the environment is represented with a finite set of states $\mathcal{S} = \{s_0, s_1, s_2, \ldots, s_N\}$, and the finite set of possible actions is $\mathcal{A} = \{a_0, a_1, \ldots, a_K\}$. But instead of perceiving directly the state in which the environment is, the agent senses an *observation* of it — a signal that depends on the true state of the system but provides only partial information

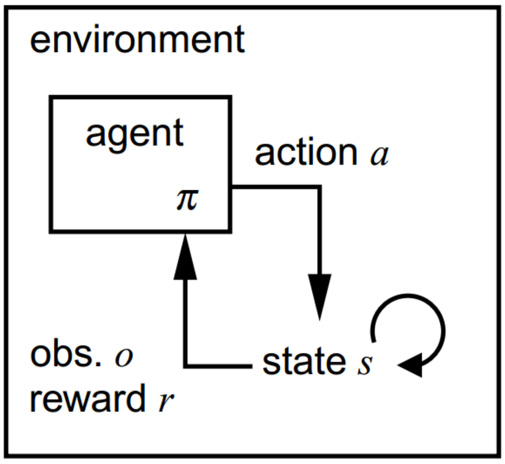


Figure 3.2: The agent-environment interaction in a POMDP [5]

about it. The set of observations is discrete and finite: $\mathcal{O} = \{o_0, o_1, \dots, o_M\}$, and represents all the possible sensor readings the agent can receive.

If the system is in state s and the agent performs action a, we have that the system transitions to state s' according to the transition probability $p(s'|s,a)$ as in a MDP, but the agent receives observation o' with probability

$$z(a, s', o') = \Pr(o'|a, s')\,, \tag{3.13}$$

using the notation of [6].

Figure 3.2 represents the interaction between these elements in the POMDP.

Thus, a POMDP can be described as the tuple $(\mathcal{S}, \mathcal{A}, p, r, \mathcal{O}, z)$, where $r = r(s, a, s')$ is the expected immediate reward, as in an MDP [3].

The goal of the agent is, once again, to find the optimal policy π that maximizes the expected return in (3.7). To solve both MDPa and POMDPs, one can use several different methods, which are classified according to the dimensions of the state and action spaces, as well as based on whether the model of the environment is available or not. In fact, there are *tabular methods* — which are exact and use arrays to store value functions, or *approximate solution methods* — which employ function approximators, using limited computational resources; then there are *model-based methods* — which make use of the known model of the environment and of planning to find the exact optimal policy, or *model-free methods* — which are trial-and-error learner.

Often finding an exact optimal policy for POMDPs can be computationally very expensive, so even for relatively small problems it is required to use approximate methods [7]. They are divided into two main classes: *value-function methods* — or *action-value methods* as in [1], that attempt to learn an approximate value function of the belief states (which are probability distributions over the states); and *policy-based methods*, that search for a good policy within a fixed class of parameterized policies. We will focus on the second ones, since it is what we will implement to solve our problem.

3.4 Policy gradient methods

A *policy gradient method* is a policy-based method that attempts to optimize the parameters of a parametrized policy using either *gradient ascent* or *gradient descent* in the parameters' space, based on the expression of the agent's goal. In fact, it tries to maximize (or minimize) the expected return of a policy inside the policy class dictated by the parametrization. This method then uses directly the policy to choose the actions, instead of consulting a value function, which however might still be used to learn the parameters.

Policy gradient methods present some perks that make them highly valuable: the parametrized policy allows for direct incorporation of potential environment insights, they can naturally switch from the discrete setting to the continuous one, and they converge to at least a locally optimal policy, even if, as a drawback, the convergence can be very slow and finding the global optimum instead of a local one can be hard [8]. Another distinctive trait of these methods is that they can also naturally handle partial state information. In fact, if a state variable can not be observed, then we can choose the policy parametrization such that the parameters do not depend on that state variable. Thus, we can directly apply these methods to POMDPs, without having to make any changes.

We will denote the policy's parameters' vector with $\boldsymbol{\theta} \in \mathbb{R}^{d'}$, and we will write, as in (3.5),

$$\pi(a|s,\boldsymbol{\theta}) = \Pr\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}\,, \tag{3.14}$$

for the probability of taking action a being in state s at time t with parameters $\boldsymbol{\theta}$ [1]. The policy can be parametrized in any way, as long as it is differentiable with respect to its parameters, which means that the column vector of partial derivatives of π w.r.t. the components of the parameters' vector $\boldsymbol{\theta}$, $\nabla_{\boldsymbol{\theta}}\pi(a|s,\boldsymbol{\theta})$, exists and is finite for all $s \in \mathcal{S}, a \in \mathcal{A}$ and $\boldsymbol{\theta} \in \mathbb{R}^{d'}$. In particular, as described in [8], for discrete problems is often used the (exponential) soft-max distribution (i.e., Gibbs or Boltzmann distribution)

$$\pi(a|s,\boldsymbol{\theta}) = \frac{e^{\phi(s,a)^\top \boldsymbol{\theta}}}{\sum_{b\in\mathcal{A}} e^{\phi(s,b)^\top \boldsymbol{\theta}}}\,, \tag{3.15}$$

while for continuous problems it is used the Gaussian distribution

$$\pi(a|s,\boldsymbol{\theta}) = \mathcal{N}(\phi(s,a)^\top \boldsymbol{\theta}_1, \boldsymbol{\theta}_2)\,, \tag{3.16}$$

where $\boldsymbol{\theta}_2$ is an exploration parameter, and $\phi(s,a) \in \mathbb{R}^{d'}$ is a feature vector characterizing state s and action a [9].

To learn the policy parameters $\boldsymbol{\theta}$ we will use the gradient of some scalar *performance measure* $J_\pi(\boldsymbol{\theta})$ — which can also be directly the expected return — with respect to the policy parameters. If the performance $J_\pi(\boldsymbol{\theta})$ measures the rewards that the agent receives, we want to maximize it, so we update the parameters approximating gradient *ascent* in $J_\pi(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta}_k)\,; \tag{3.17}$$

instead, if the performance measures the costs that the agent encounters, we seek to minimize it, so we use approximate gradient *descent* in $J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \alpha \nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta}_k)\,; \tag{3.18}$$

where, in both cases, $\alpha > 0$ is a *step-size parameter* or the *learning rate.*

As done before, we will treat the episodic case, for which we define the *performance measure* $J_\pi(\boldsymbol{\theta})$ as the average value of the initial states of the process:

$$\begin{aligned} J_\pi(\boldsymbol{\theta}) &= \sum_s \rho_0(s) V_{\pi_{\boldsymbol{\theta}}}(s) \\ &= \sum_s \rho_0(s) \sum_a \pi(a|s) Q_{\pi_{\boldsymbol{\theta}}}(s,a)\,. \end{aligned} \tag{3.19}$$

where we define as $\rho_0(s')$ the probability that an episode begins in state $s' \in \mathcal{S}$.

Let us define $\eta_\pi(s')$ as the average number of time steps that the agent spends in state s' before the process dies. Time is spent in a state s' if episodes start in that state s', or if the environment transitions into s' from a previous state s in which time is spent. Thus, we have that the formula for η_π is

$$\eta_\pi(s') = \rho_0(s') + \sum_s \eta_\pi(s) \sum_a \pi(a|s) p(s'|s,a), \text{ for all } s' \in \mathcal{S}. \tag{3.20}$$

This is a *linear system* of equations, one for each state $s' \in \mathcal{S}$, and it can be solved for the expected number of visits $\eta_\pi(s)$.

The *policy gradient theorem* provides an analytic expression for the gradient of the performance J with respect to the policy parameters, which is what we need to approximate for gradient ascent in (3.17) (or descent in (3.18)):

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_s \eta_\pi(s) \sum_a Q_\pi(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|s). \tag{3.21}$$

See [1] for the demonstration in the episodic case, which involves just elementary calculus and re-arranging of terms in the expression of the value function V_π.

Thus, solving the expression for (3.21) and using it in equations (3.17) – (3.18) allows optimizing the parameters of the policy, building an optimal policy in the chosen parametrization class. Note that to find both η_π and Q_π we need to solve two linear systems, (3.12) and (3.20), and, moreover, they depend on the policy π, thus they must be solved at every iteration.

Chapter 4

The model

In this chapter, we are going to formulate our problem using the frameworks that we saw in chapter 3: we will define the basic elements, and then we will develop the formulas and the algorithm that we need to solve it.

4.1 The mathematical model

After a fault occurs and after the technician, together with the remote-control room, restricts it to a limited number of non-remotely-controlled substations, the problem consists in visiting, and thus reconnecting, these substations in an order that minimizes the cost of the fault. We define the *cost of the fault* as the amount of time each underlying user of each substation remains disconnected.

Thus, we are given a set of initially disconnected substations $\mathcal{C}$, with cardinality $|\mathcal{C}| = N$, between two remotely controlled substations, where these last ones will not be included in the set, since they are already reconnected. Looking at the electrical schematics of the Trieste power grid, we notice that the number of substations between two remotely controlled ones is always less than 20, so we can say that in our problem $N < 20$. Luckily, this restricts the dimension of the problem.

4.1.1 Elements of the POMDP

Given that we don't know the position of the fault, but we have to find it while we reconnect the substations, we cannot use an MDP to model this problem; instead, we will use a POMDP. The *agent* is the technician that has to decide which substation to visit at each step, while the *environment* includes everything else, among which the possible positions of the fault, and the set of disconnected substations, and their positions and distances.

We define the *state* s of the environment as the tuple

$$s = (x_g, v_k, \{v\}) \tag{4.1}$$

where x_g is the position of the fault, $v_k \in \mathcal{C}$ is the substation in which the technician is, and $\{v\}$ is the set of substations still disconnected after the technician operates in the current substation v_k. The set $\{v\}$ can also be chosen to be the set of substations already reconnected, since they are complementary w.r.t. the set $|\mathcal{C}|$, but we chose to use the disconnected substations to ease the notation. Since the position of the fault is unknown, the variable x_g is *hidden*, while the variables v_k and $\{v\}$ are *observable*. When the fault occurs, the technician can be everywhere: at home if it happens in the middle of the night, at the company, or on the go. So we introduce an extra dummy substation, called substation 0, which is the position of the technician when the fault occurs. Given this, we have that the *initial state* is always of the form $s_0 = (x_g, 0, \mathcal{C})$, thus we have different initial states, one for every possible position of the fault. Instead, a *terminal state* is of the form $s_t = (x_g, v_k, \varnothing)$. If the fault is localized on an electrical cable, then v_k is one of the two substations at the ends of that faulty cable, so we have two different terminal states; while if the fault is in a substation, v_k is that exact substation, so the terminal state is only one. We notice that the initial cost has a random component, which depends on the position of the technician when the fault occurs, so on the position of the substation 0. To remove this randomness, we could also impose that the technician position is always at the company, but we chose not to do that, in order to be closer to what really happens.

We then define the *observation* o that the agent senses from the environment as

$$o = (v_k, \{v\})\,, \tag{4.2}$$

and we will also write the state as $s = (x_g, o)$, where $o = (v_k, \{v\})$ is the observation itself. We define the observable o to be a function of s:

$$\begin{array}{rccc} o(s): & \mathcal{S} & \rightarrow & \mathcal{O} \\ & s = (x_g,\, o = (v_k, \{v\})\,) & \mapsto & o = (v_k, \{v\}) \end{array}\,. \tag{4.3}$$

For different states $s = (x_g,\, o = (v_k, \{v\})\,)$ that differ only by the position of the fault x_g, this function associates the same observable $o = (v_k, \{v\})$. Thus, we have that o is an equivalence class for s. For brevity, we will often keep this dependency implicit, and write simply o instead of $o(s)$, meaning that $o = o(s)$. In particular, the observation of an initial state $s_0 = (x_g, 0, \mathcal{C})$ is $o_0 = (0, \mathcal{C})$.

We define the *action* a that the agent can do as the choice of the specific substation the technician will visit as the next step, thus we have that $a \in \mathcal{C}$. Actually, since the technician visits only disconnected substations, we have that $a \in \{v\}$, if we are in state $s = (x_g, v_k, \{v\})$.

We said that this is a problem with terminal states, which occur when we reconnect all the substations. A terminal state will always be reached, since with every action we visit a substation and can reconnect it, so at the very least we remove that substation from the set of disconnected substations. Actually, if we are lucky, each time we can remove half of the substations from the set of disconnected substations. We are therefore positive that the process terminates. So, for this specific problem, it doesn't make sense to introduce a discount factor γ (therefore, we have that $\gamma = 1$ in the formulas of chapter 3).

Given all that, we have that the *next state* s' of the environment is

$$s' = (x_g, v_{k+1} = a, \{v'\}), \tag{4.4}$$

where $\{v'\}$ is the set of disconnected substations after the technician operates in the substation v_{k+1}. Since the technician can always at least reconnect the substation they visit, the set of disconnected substations decreases after each action, so we have that $\{v'\} \subseteq \{v\} \backslash a$.

Finally, we define the *expected reward* as the *cost* of going to a certain substation (as the time, in seconds, it takes to go there from where the technician is) multiplied by the number of disconnected users. Let's define as $d_{v_k,v_{k+1}}$ the time in seconds to go from the substation v_k to the next substation v_{k+1}, and as n_k the number of users still disconnected *before* operating in the substation v_{k+1}. So if we are in a state $s = (x_g, v_k, \{v\})$, we make an action a, and we end up in a state $s' = (x_g, v_{k+1} = a, \{v'\})$, we have that the number of disconnected users is

$$n_k = \sum_{v \in \{v\}} u_v, \tag{4.5}$$

where u_v is the number of users underneath the substation v. So the expected reward has the following formula:

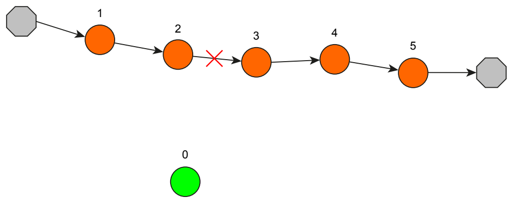
$$r(s, a, s') = d_{v_k,a} \cdot n_k = d_{v_k,a} \cdot \sum_{v \in \{v\}} u_v \,, \tag{4.6}$$

even if it actually depends only on the previous state s and the action taken a, but we will keep the term s' to be consistent with the formulas of [1]. For now, in the cost we will ignore the cost of establishing if the fault is before or after the substation in which the technician is, which is complicated and might raise the total cost significantly. This is due to a lack of data. To improve the computation of the cost, we need to carefully take note of the operations the technicians perform when a fault occurs, and then expand the model. To carry out the data collection, we will implement a serverless Telegram bot using AWS.

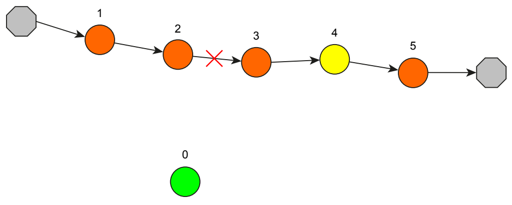
In Figure 4.1 we can see (part of) a trajectory (we miss the rewards) of our POMDP, using a set of fictional substations.

The environment is *deterministic*, so given an admissible action a, we will surely perform it and end up in the state to which that action leads. We can say that there are no execution errors, and we will always do what we want to do. This means that the next state s' is a function of the previous state s and the action a performed: $s' = \sigma(s, a)$. So, mathematically, we have that the *transition probability* is

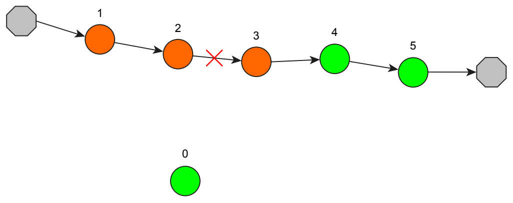
$$p(s' \mid s, a) = \mathbb{I}\big(s' = \sigma(s, a)\big) = \delta_{s',\sigma(s,a)} = \begin{cases} 1 & \text{if } s' = \sigma(s, a) \\ 0 & \text{otherwise} \end{cases} , \tag{4.7}$$



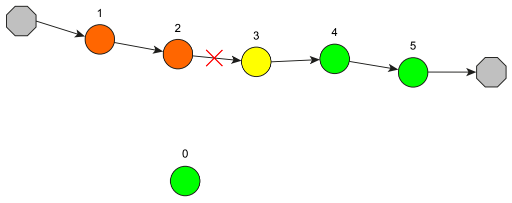
(a) A fault has occurred in 2-3. We are in substation 0 and all the substations are disconnected (orange). Initial state: $s_0 = (2\text{-}3, 0, \mathcal{C} = \{1, 2, 3, 4, 5\})$.



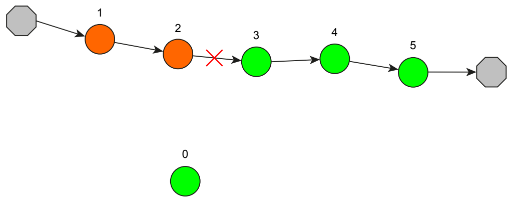
(b) We visit substation 4 (yellow). Action: $a_0 = 4$.



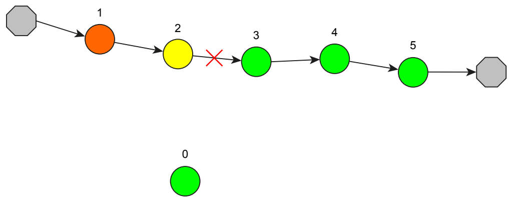
(c) We reconnect substations 4 and 5 (green). State: $s_1 = (2\text{-}3, 4, \{1, 2, 3\})$.



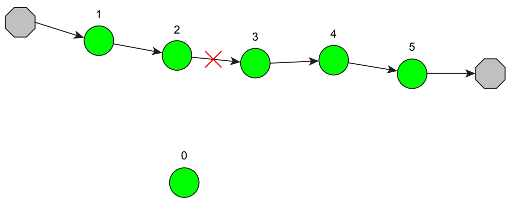
(d) We visit substation 3 (yellow). Action: $a_1 = 3$.



(e) We reconnect substation 3 (green). State: $s_2 = (2\text{-}3, 3, \{1, 2\})$.



(f) We visit substation 2 (yellow). Action: $a_2 = 2$.



(g) We reconnected substations 1 and 2 (green). All the substations are reconnected. Terminal state: $s_3 = (2\text{-}3, 2, \varnothing)$.

Figure 4.1: Fictional example of the sequence of actions of the technician, from the occurrence of the fault to its resolution. The fault is in the electrical cable between substations 2 and 3, and the gray octagonal substations are the remotely controlled ones, while the round substations are the disconnected ones, except substation 0, which is the initial position of the technician.

where $\mathbb{I}$ is the characteristic function of a set, and δ is the Kronecker delta. In our specific case, the transition probability is equal to 1 only when, starting from state $s = (x_g, v_k, \{v\})$, the new substation v_{k+1} of the next state $s' = (x_g, v_{k+1}, \{v'\})$ is equal to the action $a \in \{v\}$ that we took, so:

$$p(s'|s,a) = \begin{cases} 1 & \text{if } v_{k+1} = a \\ 0 & \text{if } v_{k+1} \neq a \end{cases} . \tag{4.8}$$

4.1.2 Policy parametrization

Given that we are in a situation of partial observability, as we don't know the position of the fault, the *policy* depends only on the observables and not on the state, so it doesn't know where the fault is. Let's define a parameterized policy using a simple *tabular* parameterization, in which we have a parameter $\theta_{o,a}$ for each observable o and action a pair:

$$\boldsymbol{\theta} = (\theta_{o,a})_{o\in\mathcal{O},a\in\mathcal{A}} = \begin{pmatrix} \theta_{o_1,a_1} & \theta_{o_1,a_2} & \cdots & \theta_{o_1,a_N} \\ \theta_{o_2,a_1} & \theta_{o_2,a_2} & \cdots & \theta_{o_2,a_N} \\ \vdots & & & \vdots \\ \theta_{o_{|O|},a_1} & \theta_{o_{|O|},a_2} & \cdots & \theta_{o_{|O|},a_N} \end{pmatrix} \tag{4.9}$$

(where we recall that N is the number of substations initially disconnected, so the number of possible actions).

Then we use the soft-max, or Boltzmann, distribution of (3.15) to construct the policy:

$$\pi\Big(a \mid o(s) = (v_k, \{v\}), \boldsymbol{\theta}\Big) = \frac{e^{\theta_{o,a}}}{\sum_{b\in\{v\}} e^{\theta_{o,b}}} . \tag{4.10}$$

So we have that also the policy is a matrix:

$$\pi\Big(a \mid o(s), \boldsymbol{\theta}\Big) = \begin{pmatrix} \pi(a_1|o_1,\boldsymbol{\theta}) & \pi(a_2|o_1,\boldsymbol{\theta}) & \cdots & \pi(a_N|o_1,\boldsymbol{\theta}) \\ \pi(a_1|o_2,\boldsymbol{\theta}) & \pi(a_2|o_2,\boldsymbol{\theta}) & \cdots & \pi(a_N|o_2,\boldsymbol{\theta}) \\ \vdots & & & \vdots \\ \pi(a_1|o_{|O|},\boldsymbol{\theta}) & \pi(a_2|o_{|O|},\boldsymbol{\theta}) & \cdots & \pi(a_N|o_{|O|},\boldsymbol{\theta}) \end{pmatrix} . \tag{4.11}$$

The policy cannot depend on the position of the failure, otherwise we would automatically have solved the problem: the policy would suggest going in the substation in which the fault is, or in the two substations at the ends of the faulty electrical cable.

Note that what matters in (4.10) is not the absolute value of the parameters, but only the relative value of a parameter over another one: if we add a constant $c \in \mathbb{R}$ to all the parameters, there is no effect on the action probabilities, since the constant cancels out:

$$\frac{e^{\theta_{o,a}+c}}{\sum_{b\in\{v\}} e^{\theta_{o,b}+c}} = \frac{e^{c}\cdot e^{\theta_{o,a}}}{e^{c}\sum_{b\in\{v\}} e^{\theta_{o,b}}} = \frac{e^{\theta_{o,a}}}{\sum_{b\in\{v\}} e^{\theta_{o,b}}} = \pi(a \mid o(s), \boldsymbol{\theta})\,. \tag{4.12}$$

Initially, all parameters are the same and depend neither on the observation nor on the action (e.g., $\boldsymbol{\theta} = \mathbf{0}$, thus $\theta_{o,a} = 0$ for all $o \in \mathcal{O}, a \in \mathcal{A}$), so that all actions have an equal probability of being selected. This is called the *random policy*, and in our case is

$$\pi\left(a \mid o(s), \boldsymbol{\theta}\right) = \frac{e^{\theta}}{\sum_{b\in\{v\}} e^{\theta}} = \frac{1}{\sum_{b\in\{v\}} 1} = \frac{1}{|\{v\}|}\,, \tag{4.13}$$

which means that we randomly choose a substation to visit, since they all have the same uniform probability. Actually, this is true for every choice of $\boldsymbol{\theta}$ which doesn't depend on the observation-action pair, so also any other constant $c \in \mathbb{R}$ would do (as we can see from (4.12)), but in practice, to construct a uniform policy, one usually takes $\boldsymbol{\theta} = \mathbf{0}$.

One of the advantages of using the soft-max distribution to parameterize policies is that, in the learning phase, we are able to choose a random action without introducing and manually tuning a ε term, which we would otherwise need to perform some exploration alongside the exploitation. Instead, the stochasticity is intrinsic, since we don't perform a deterministic maximization over the actions, but we choose it according to its probability. In particular, if the optimal policy is stochastic, an argmax would never be able to approximate it, not even with a ε parameter, while the soft-max can do it by construction. On the contrary, if an action is deterministic, the soft-max can approach it as close as possible; in fact, the approximate policy can approach a deterministic policy without any problem.

If we search in this space of parametrized policies, this will give us a policy that doesn't depend on time, but only on the parameters, which we want to optimize. This means that, by construction, we have a *stationary policy.* The reason for this choice is that, in our problem, the time is not encoded like a problem of dynamic programming, in which we have a well-defined sequence of time steps. The structure of the states is already a measure of time, as the number of moves already done: the sequence of substations we already visited. So what is important is not to establish a policy at different time steps, but to create a policy with respect to the states (in our case w.r.t. the observables).

4.1.3 Value function and performance measure

In our problem, the equation for the *action value function* $Q_\pi(s,a)$ in (3.12), given that the state is $s = (\, x_g, o = (v_k, \{v\})\,)$, the action is $a \in \{v\}$ and the next state is $s' = (\, x_g, o' = (v_{k+1} = a, \{v'\})\,) = \sigma(s,a)$ (since the system is deterministic), becomes, thanks to (4.6) and (4.8),

$$\begin{aligned} Q_\pi(s,a) &= \sum_{s'} p(s' \mid s,a) \left[r(s,a,s') + \sum_{a'} \pi(a'|o(s'),\boldsymbol{\theta}) Q_\pi(s',a') \right] \\ &= \left(d_{v_k,a} \cdot n_k + \sum_{a' \in \{v'\}} \pi\Big(a' \Big| o\big(\sigma(s,a)\big), \boldsymbol{\theta}\Big)\, Q\big(\sigma(s,a), a'\big) \right) . \end{aligned} \tag{4.14}$$

Now, let us find the formula for $\rho_0(s')$, the probability of starting in state s'. Since we have no prior information on where the fault might be, ρ_0 doesn't depend on it, so it will be uniform in x_g. The fault can happen either in one of the substations, which are $|\mathcal{C}| = N$, or on one of the electrical cables, which are $N+1$, so the number of possible x_g is $N + (N+1) = 2N+1$. In an initial state, the current position of the technician v_k must be the dummy substation 0, which represents the position of the technician when the fault occurs, and the set of the disconnected substations must be equal to the set of all the substations $\mathcal{C}$. So ρ_0 must be 1 when the current substation is 0 and the set of the disconnected substations is equal to $\mathcal{C}$, and must be 0 for every other situation. In other words, we have that

$$\begin{aligned}\rho_0\Big(s = (\,x_g, o = (v_k, \{v\})\,)\Big) &= \Pr(x_g)\mathbb{I}(o = o_0 = (0, \mathcal{C})) \\ &= \frac{1}{2N+1}\mathbb{I}\big(v_k = 0, \{v\} = \mathcal{C}\big) \\ &= \frac{1}{2N+1}\delta_{o(s),o_0}\,,\end{aligned} \tag{4.15}$$

where o_0 is the initial observation, and in the last equivalence we used the Kronecker delta δ. Actually, according to the technicians of the AAA company, the majority of the faults happens on the electrical cables, usually when they are no longer perfectly isolated (e.g., the wire's insulation breaks down) and they cause a short circuit, being connected to the ground and allowing charge to flow through it. We will try to take advantage of this information at a later moment, but for now we will suppose that every component has the same probability of being damaged.

Moreover, the formula (3.20) for the function $\eta_\pi(s')$ in our case becomes, thanks to (4.8) and (4.15),

$$\begin{aligned}\eta_\pi\Big(s' = (\,x_g, o' = (v_{k+1}, \{v'\})\,)\Big) &:= \rho_0(s') + \sum_s \eta_\pi(s) \sum_a \pi(a|o(s), \boldsymbol{\theta}) p(s'|s,a) \\ &= \frac{1}{2N+1}\delta_{o',o_0} + \sum_{s\in \mathrm{pa}(s')} \eta_\pi(s)\pi(v_{k+1}|o(s), \boldsymbol{\theta})\,,\end{aligned} \tag{4.16}$$

where $\mathrm{pa}(s')$ indicates the parents of the node s' in the states' dependency graph, which represents all the possible sequences of states.

Finally, let us define the *performance measure* $J_\pi(\boldsymbol{\theta})$ as the sum of all the costs we incur, added up over time until the process is concluded. Actually, the time steps of the process are merely formal steps, since we don't keep track of the time passed, but we simply move from one substation to another. Thus, the actual physical time is in the costs, as the cost of going from one substation to another, which we measure using the time it takes to drive between them. Given the definition of $J_\pi(\boldsymbol{\theta})$ as in (3.19), in our case we have that

$$\begin{aligned} J_\pi(\boldsymbol{\theta}) &= \sum_s \rho_0(s) \sum_a \pi_{\boldsymbol{\theta}}(a|o(s),\boldsymbol{\theta})\, Q_{\pi_{\boldsymbol{\theta}}}(s,a) \\ &= \sum_s \frac{1}{2N+1}\delta_{o(s),o_0} \sum_a \pi_{\boldsymbol{\theta}}(a|o(s),\boldsymbol{\theta})\, Q_{\pi_{\boldsymbol{\theta}}}(s,a) \\ &= \sum_{x_g} \frac{1}{2N+1} \sum_a \pi_{\boldsymbol{\theta}}(a|o_0,\boldsymbol{\theta})\, Q_{\pi_{\boldsymbol{\theta}}}\big((x_g,o_0),a\big)\,, \end{aligned} \tag{4.17}$$

where in the second equivalence we used equation (4.15), and the last equivalence derives from the fact that the sum $\sum_s \delta_{o(s),o_0}$ is equivalent to a summation over every possible position of the fault for the initial observation o_0. In particular, (x_g, o_0) represents every possible initial state for different positions of the fault.

4.1.4 Policy gradient method

Since we know every aspect of the problem, and we have a model of the environment, we can solve it via a *model-based method*. Due to the partial observability, though, we cannot use *dynamic programming* methods like the ones in [10], because we would need a policy that depends on the entire state, thus also on the hidden variable x_g. However, if we know where the fault is, the optimal solution is straightforward: if it is on an electrical cable, we need to visit the two substations at the ends of it (we might have to choose the right order); if it is in a substation, we visit it directly. Thus, we need an algorithm that can work in a POMDP.

We will use one of the *policy gradient methods* of section 3.4 in the POMDP, since they can naturally handle partial observability. In particular, we will perform a *gradient descent* — since we want to *minimize* our *cost* $J_\pi(\boldsymbol{\theta})$ — on the parameters $\boldsymbol{\theta}$ of the policy $\pi_{\boldsymbol{\theta}}$, where we said that the latter depends only on the observations, and so will do the gradient.

The formula of the gradient in (3.21) in our case becomes

$$\nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta}) = \sum_s \eta_{\pi_{\boldsymbol{\theta}}}(s) \sum_a Q_{\pi_{\boldsymbol{\theta}}}(s,a) \nabla_{\boldsymbol{\theta}} \pi(a|o(s),\boldsymbol{\theta})\,. \tag{4.18}$$

So, given that $\boldsymbol{\theta}$ is a matrix $\boldsymbol{\theta} = (\theta_{o,a})_{o\in\mathcal{O},a\in\mathcal{A}}$, we have that also the gradient is a matrix:

$$\nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial\theta_{o_1,a_1}} J & \cdots & \frac{\partial}{\partial\theta_{o_1,a_N}} J \\ \frac{\partial}{\partial\theta_{o_2,a_1}} J & \cdots & \frac{\partial}{\partial\theta_{o_2,a_N}} J \\ \vdots & & \\ \frac{\partial}{\partial\theta_{o_{|O|},a_1}} J & \cdots & \frac{\partial}{\partial\theta_{o_{|O|},a_N}} J \end{pmatrix}, \tag{4.19}$$

where N is the number of initially disconnected substations.

First of all, let us compute the gradient of the policy π. Given the equation of the policy in (4.10), we have that its derivative is

$$\begin{aligned} \frac{\partial}{\partial\theta_{o',a'}} \pi\Big(a \mid o = (v_k, \{v\}), \boldsymbol{\theta}\Big) &= \frac{\partial}{\partial\theta_{o',a'}} \left(\frac{e^{\theta_{o,a}}}{\sum_{b\in\{v\}} e^{\theta_{o,b}}} \right) \\ &= \delta_{o',o} \cdot \frac{\delta_{a',a} \cdot e^{\theta_{o,a}} \cdot \sum_{b\in\{v\}} e^{\theta_{o,b}} - e^{\theta_{o,a}} \cdot e^{\theta_{o,a'}}}{(\sum_{b\in\{v\}} e^{\theta_{o,b}})^2} \\ &= \delta_{o',o} \cdot \left(\frac{\delta_{a',a} \cdot e^{\theta_{o,a}} \cdot \sum_{b\in\{v\}} e^{\theta_{o,b}}}{(\sum_{b\in\{v\}} e^{\theta_{o,b}})^2} - \frac{e^{\theta_{o,a}}}{\sum_{b\in\{v\}} e^{\theta_{o,b}}} \cdot \frac{e^{\theta_{o,a'}}}{\sum_{b\in\{v\}} e^{\theta_{o,b}}} \right) \\ &= \delta_{o',o} \cdot (\delta_{a',a}\pi(a|o) - \pi(a|o)\pi(a'|o)) \\ &= \delta_{o',o} \left(\delta_{a',a} - \pi(a'|o)\right) \pi(a|o)\,, \end{aligned} \tag{4.20}$$

since if $o \neq o'$ we have that $\theta_{o',a'}$ and $\theta_{o,a}$ are ultimately different parameters. In particular, for the random policy (so when $\boldsymbol{\theta} = \mathbf{0}$) we have that

$$\frac{\partial}{\partial\theta_{o',a'}} \pi\Big(a \mid o = (v_k, \{v\}), \boldsymbol{\theta}\Big)\Big|_{\boldsymbol{\theta}=\mathbf{0}} = \delta_{o',o} \left(\frac{1}{|\{v\}|}\delta_{a',a} - \frac{1}{|\{v\}|^2} \right) \tag{4.21}$$

Therefore, we have that also the gradient of the policy is a matrix:

$$\nabla_{\boldsymbol{\theta}} \pi(a|o,\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial\theta_{o_1,a_1}} \pi(a|o,\boldsymbol{\theta}) & \cdots & \frac{\partial}{\partial\theta_{o_1,a_N}} \pi(a|o,\boldsymbol{\theta}) \\ \vdots & & \vdots \\ \frac{\partial}{\partial\theta_{o_{|O|},a_1}} \pi(a|o,\boldsymbol{\theta}) & \cdots & \frac{\partial}{\partial\theta_{o_{|O|},a_N}} \pi(a|o,\boldsymbol{\theta}) \end{pmatrix} \tag{4.22}$$

Given this, we have that

$$\begin{aligned}\nabla_{\theta_{o',a'}} J_\pi(\boldsymbol{\theta}) &= \sum_s \eta_\pi(s) \sum_a Q_\pi(s,a) \nabla_{\theta_{o',a'}} \pi(a|o(s)) \\ &= \sum_s \eta_\pi(s) \sum_a Q_\pi(s,a) \delta_{o',o(s)} \Big(\left(\delta_{a',a} - \pi(a'|o(s)) \right) \pi(a|o(s)) \Big) \\ &= \sum_{x_g} \eta_\pi((x_g,o')) \sum_a Q_\pi((x_g,o'),a) \left(\delta_{a,a'} - \pi(a'|o') \right) \pi(a|o') \,,\end{aligned} \tag{4.23}$$

where, as we did in equation (4.17), the sum $\sum_s \delta_{o',o(s)}$ is equivalent to a summation over every possible position of the fault for the specific observation o' on which we are deriving. In particular, (x_g, o') is the state with a fault at some position x_g and with observation o'.

The idea of the algorithm is the following. We start from a certain policy, for example the random policy of equation (4.13), in which we choose randomly the substation to be visited. Then we compute the gradient for every parameter using (4.23). As we saw in (4.21), the gradients of the policy are simple computations (since we decided the parametrization of the policy, we were able to derive their formulas), while the computations for the objects Q_π and η_π have to be done indirectly: we have to solve the linear equations (4.14) and (4.16) for the current policy, which can be more or less computationally heavy, also depending on the method chosen.

Having done these steps, we have the value of the gradient for every parameter. Then we take a step in the parameters space, and we descend the gradient. We stop when the value of $J_\pi(\boldsymbol{\theta})$ doesn't change much in percentage.

In general, there is no guarantee that this is a convex problem in the parameters $\boldsymbol{\theta}$, on the contrary, we could have several minima. Therefore, we should perform different gradient descents starting with some policies different from the one which has $\boldsymbol{\theta} = \mathbf{0}$, to see if we can reach a different minimum, or if we always reach the same one. These are called random restarts. Of course, this doesn't guarantee finding the global minimum, but it is the best we can do to at least check that we are not stuck in a local minimum. For the random restarts, we chose the parameters $\boldsymbol{\theta}$ from a standard normal distribution $\mathcal{N}(0,1)$, in order to have both positive values and negative ones to start from.

If we have the intuition that there is a deterministic sequence of actions to be performed, we have that their parameters $\theta_{\cdot,a}$ tend to infinity. This is because the deterministic policies correspond to a one-hot vector in which we have 1 for only one action and 0 for all the other actions, and in the soft-max policy this happens when the parameter associated with this action becomes much bigger than the others, so that when we normalize it with the parametrization of the policy it becomes 1 while the others are so small that becomes 0. If it is so, the gradient descent never stops and there could be problems of overflowing on the values of the policy, since the parameters $\boldsymbol{\theta}$ can become very large, and the exponentials of the soft-max would explode. A smart thing to do when this happens is to rewrite the parametrization in the following way (the observation o is fixed):

$$\pi(a|o) = \frac{e^{\theta_{o,a}}}{\sum_b e^{\theta_{o,b}}} = \frac{e^{-(\max_{a'} \theta_{o,a'} - \theta_{o,a})}}{\sum_b e^{-(\max_{a'} \theta_{o,a'} - \theta_{o,b})}} \,. \tag{4.24}$$

The benefit of writing the policy in this way is that, since the exponents of the two exponentials are negative (the parts in the parenthesis, $\max_{a'} \theta_{o,a'} - \theta_{o,\cdot}$, are always positive), it never explodes. In fact, negatives with large exponents “saturate” to zero rather than infinity, so we have a better chance of avoiding NaNs. This is a simple numerical trick that allows improving the stability of the algorithm.

4.2 Implementation of the model

First of all, we generated all the states of the system using *backtracking*, a technique used to generate systematically all the admissible solutions of a problem (usually of combinatorial nature) [11]. It proceeds by building candidate solutions incrementally using recursion, checking at each step if it arrived at a complete and valid solution, and backtracking when it founds one — or when it discovers that the current one cannot become a valid solution — to explore other ones. Indeed, the function starts from a complete initial state, so, besides the current position of the technician and the set of disconnected substations, it requires also the position of the fault, and it determines all the possible actions that can be taken from that state. Then it computes all the subsequent states, determining each time which substations can be reconnected by doing that action (here is where we need to know the position of the fault), and it continues until it has reconnected all the substations. Looping on all the possible positions of the fault, so on all the possible initial states, we can reconstruct all the possible states of the system. While constructing the states, at the same time it creates the states' dependency graph, in which we can find, given a state, all the possible states in which we can end up doing an admissible action. This is actually a forest of trees: each graph is a tree because we can never return to a previous state, and it is a forest since they are grouped by the position of the fault: if we start in a given initial state we can never end up in a state with a different position of the fault.

From the list of states, we create also the list of observables, noting down also to which state they are associated with.

After having done that, we proceed to construct the matrix of parameters $\boldsymbol{\theta}$, initialized to zero, and the matrix of the policy $\pi_{\boldsymbol{\theta}}$, equal in dimensions to the previous one, using the soft-max parametrization.

Then we proceed with the computation of the matrix of Q_π. We constructed the matrix $R = (R[s,a])_{s\in\mathcal{S},a\in\mathcal{A}}$, which stores the immediate cost of being in state s and doing action a, which thanks to (4.6) is:

$$R[s,a] = r(s,a,s') = d_{v_k,a} \cdot \sum_{v\in\{v\}} u_v. \tag{4.25}$$

Then we use it to initialize the matrix which will store the values of Q_π. Recall that in the equation for Q_π, (4.14), we use the values of Q_π of the next states s' to compute the value of Q_π for the current state s. Thus, we need to use the states' dependency graph from the leaves to the roots to compute the matrix of Q_π bottom-up. We perform a personalized *breadth-first-search*, or BFS, on the states' dependency graph, in order to be able to add each next-state contribution to the value of Q_π for the current state. A breadth-first-search is an algorithm for exploring a graph, which visits every node and every edge of the latter. In a breadth-first-search, the nodes are visited in order of increasing distance from the source of the visit, where the distance among the source and a generic node is the minimum number of edges in a path among them [11]. In our modified BFS instead, we start from the terminal states in the leaves, and then we visit all the edges of the graph in a breadth-first-search order, with every edge traversed in the reverse direction in order to arrive at the initial state in the root, and with all the out-edges of a node visited before the visiting that node itself, so to have finished computing the Q_π value of a node, having added all the terms deriving from its children, before using it to compute another Q_π value.

After this, we proceed with computing the matrix of η_π. In equation (4.16) we have that to compute the value of η_π for a state s' we use the values of η_π of all its preceding states s. Thus, we will use the states' dependency graph from the roots to the leaves to compute the η_π matrix top-down. As we did for the matrix of Q_π, we perform a personalized BFS on the states' dependency graph, which visits all the edges and visits a node only after all its in-edges have been visited, in order to have added to the η_π value of that node all the terms deriving from its parents before using it to compute another η_π value.

We could not have used a depth-first search, or DFS, to compute Q_π and η_π. In a DFS, contrary to a BFS, after visiting a node the search proceeds as far as possible from it along a path, until it is reached a node whose adjacent nodes have all already been visited. Then the search goes back along the last edge and continues moving away along another path not yet visited [11]. Thus, using a DFS, it would be impossible to visit all the in-edges (or the out-edges in the reverse order) of a node before visiting the node itself.

Having the matrix of the policy $\pi_{\boldsymbol{\theta}}$ and the matrix of Q_π, we can compute the performance measure $J_\pi(\boldsymbol{\theta})$ using the last equivalence of (4.17). Instead, for the computation of the gradient of $J_\pi(\boldsymbol{\theta})$, $\nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta})$, we use also the matrix η_π, following equation (4.23).

How to choose the learning parameter α appropriately? We want the difference $\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k$ in (3.18) to be small, in particular, we want $\boldsymbol{\theta}_{k+1} - \boldsymbol{\theta}_k \ll 1$, thus it must be also $\alpha \, |\nabla_{\boldsymbol{\theta}} J_\pi(\boldsymbol{\theta})| \ll 1$. Given that we have $Q_\pi \sim 10^6$ and $\eta_\pi \sim 10^{-2}$, we choose $\alpha \sim 10^{-6}$. Actually, in later experiments we set an adaptive α which is $1/\max(Q_\pi)$, in order to speed up the convergence and avoid using a really small α w.r.t. the value of Q_π.

After having computed all the elements that we need, we proceed to perform the gradient descent, stopping when the percentage relative error among two subsequent values of $J_\pi(\boldsymbol{\theta})$ is less than 1%. Actually, since we have a very little value of α, we rescale for it.

Chapter 5

Results and conclusion

We can see that the trajectories of the parameters $\boldsymbol{\theta}$ are continuous and not with sudden turns.

FIGURE OF PARAMETERS TRAJECTORIES

Bibliography

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts (MA): MIT Press, 2018. URL: http://incompleteideas.net/book/the-book.html (cit. on pp. 5, 6, 12–14, 18).

[2] Marco Wiering and Martijn van Otterlo. *Reinforcement Learning: State of the Art*. Springer Verlag, 2012. ISBN: ISBN: 978-3-642-27645-3 (cit. on p. 6).

[3] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial Intelligence* 101.1 (1998), pp. 99–134. ISSN: 0004-3702. DOI: https://doi.org/10.1016/S0004-3702(98)00023-X (cit. on pp. 7, 10, 11).

[4] William Uther. "Markov Decision Processes". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 642–646. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_512 (cit. on p. 7).

[5] Matthijs T. J. Spaan. "Partially Observable Markov Decision Processes". In: *Reinforcement Learning: State of the Art*. Ed. by Marco Wiering and Martijn van Otterlo. Springer Verlag, 2012, pp. 387–414 (cit. on pp. 9, 11).

[6] Pascal Poupart. "Partially Observable Markov Decision Processes". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 776–776. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_642 (cit. on pp. 10, 11).

[7] D. Aberdeen and Jonathan Baxter. "Scaling Internal-State Policy-Gradient Methods for POMDPs". In: *International Conference on Machine Learning*. 2002 (cit. on p. 12).

[8] Jan Peters and J. Andrew Bagnell. "Policy Gradient Methods". In: *Encyclopedia of Machine Learning*. Ed. by Claude Sammut and Geoffrey I. Webb. Boston, MA: Springer US, 2010, pp. 774–776. ISBN: 978-0-387-30164-8. DOI: 10.1007/978-0-387-30164-8_640 (cit. on pp. 12, 13).

[9] Richard S Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems*. Ed. by S. Solla, T. Leen, and K. Müller. Vol. 12. MIT Press, 2000 (cit. on p. 13).

[10] Richard E. Bellman. *Dynamic Programming*. Princeton Landmarks in Mathematics and Physics. Princeton University Press, 1957. ISBN: 0-691-07951-X (cit. on p. 24).

[11] Alan Bertossi and Alberto Montresor. *Algoritmi e strutture dati*. Terza edizione. CittàStudi Edizioni, 2014. ISBN: 9788825173956 (cit. on pp. 27–29).

Index