POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Dynamic state and action factorization for distributed reinforcement learning

Tesi di Laurea Magistrale in
Computer Science Engineering - Ingegneria Informatica

Andrea Fondacaro, 10725432

Advisor:
Prof. Marcello Restelli

Co-advisors:
Gianvito Losapio
Marco Mussi

Academic year:
2024-2025

Abstract: The handling of high-dimensional network-based systems is a key challenge in the real-world application of reinforcement learning algorithms. Standard methods are typically effective on small-scale problems but fail to scale to larger scenarios, such as those considered in the AI4REALNET project. In this work, we propose a dynamic clustering method based on mutual information to factorize the original problem into approximately independent sub-problems. We then develop a multi-agent reinforcement learning approach based on the Proximal Policy Optimization (PPO) algorithm, in which each sub-problem is tackled by a distinct agent. We demonstrate the effectiveness of this framework in two different environments. The first is a relatively small and simple setting involving power line management and load balancing. Through this environment we show that our approach significantly reduces training time while maintaining competitive performance comparing to a standard centralized approach. The second is a larger and more complex railway network simulation, which we used to show that our factorization approach can be applied across different domains.

Key-words: Multi-Agent Reinforcement Learning, Dynamic Factorization, Mutual Information

1. Introduction

In recent years, an increasing number of human tasks have been automated due to the rapid spread of AI and new technologies. The management of large networks is one of the activities that can be automated to enhance security and improve service quality: today, many operations needed to maintain these infrastructures are performed by engineers, but the rapid evolution of structures and technologies is making them increasingly difficult to manage, leading to a higher probability of human error, which can have serious consequences in safety-critical contexts such as railways and air traffic.
Network management involves a wide range of decision making processes, including monitoring and fault detection, resource allocation, routing and scheduling. These tasks are typically characterized by complex constraints and uncertainty in both the environment and the demand. Moreover, many of these systems can be represented as graphs, where nodes and edges represent model components and interactions, and where local decisions may propagate and produce effects on the adjacent components or the whole system. In this scenario, automated controllers must not only achieve good performances, but also be able to respond reliably to unexpected events like sudden changes in demand and partial observability.
Given this, many attempts have been made to apply existing technologies to large and complex infrastructures,

such as railway networks and power grids, and some of the most successful approaches leverage Reinforcement Learning (RL) techniques. RL has emerged as a powerful alternative to classic mathematical methods, which struggle with the computational issues resulting from having to deal with real-time operational problems. These approaches are learning-based, meaning that models learn from experience by trying different solutions to the same problem and identifying which option is most suitable for a given context. This makes RL well-suited to stochastic environments subject to frequent and unpredictable changes, as it targets sequential decision making and aims to optimize long-term objectives rather than isolated actions.
Despite this, applying RL to large-scale infrastructures remains challenging: as the size of the network grows, the dimensionality of the state and action spaces increases substantially (a phenomenon known as the curse of dimensionality), often resulting in slower training and less stable convergence, limiting the applicability of traditional RL methods. In addition to this, large network problems often involve long planning horizons and sparse or delayed rewards, all of which can reduce sample efficiency and make learning less stable in practice. To address these scalability limitations, the natural solution would be decomposing the problem into smaller subproblems, which can then be tackled with a multi-agent RL approach. Instead of learning a single global controller, the overall task is divided into multiple interacting components, each controlled by an agent operating on a reduced state and action space. This can significantly improve learning efficiency, but also introduces a new challenge: subproblems are not independent, and poor factorization can degrade performance and bring to convergence issues. Therefore, the effectiveness of a multi-agent approach depends heavily on how the factorization is performed. In our setting, agents are trained cooperatively by optimizing a shared global reward, encouraging coordination across clusters while preserving the benefits of a localized decision-making approach. In this work, we propose a new clustering technique based on Mutual Information (MI) to perform such a decomposition on large networks, dividing the infrastructure into clusters managed by different agents. The underlying intuition is that MI provides a statistical way to quantify dependence between components, allowing to have clusters that are highly dependent internally while minimizing dependencies across clusters. By constructing clusters in this way, we aim to reduce the complexity of the learning problem for each agent while keeping the information necessary for effective global control. This clustering principle is designed to be independent from the domain, making it applicable to various kinds of problems, despite their different dynamics and constraints. We evaluate the proposed approach on two types of network infrastructures with different characteristics: a smaller power grid control problem created by us and a larger scale railway routing problem. The large-scale railway scenarios were generated using the Flatland simulator. Flatland is a widely adopted benchmark environment for studying learning-based methods in railway traffic and routing problems, enabling controlled scalability experiments under increasing network size and traffic complexity. This allows us to study the effectiveness of our approach on a controlled benchmark and then on larger scenarios, highlighting how the same clustering principles can be applied across heterogeneous domains. The analysis focuses on the impact of clustering on training efficiency while increasing convergence stability and getting as close as possible to the optimal solution. This thesis is developed withing the context of the AI4REALNET project, funded by the European Union's Horizon Europe Research and Innovation programme. AI4REALNET focuses on critical infrastructures modeled as networks including power grids, railway systems and air traffic management, with the objective of developing AI solutions that support human operations in increasingly complex decision-making processes. In this perspective, our work aligns with the goal of improving the scalability and reliability of RL-based approaches for real-world network operation.

1.1. Thesis Structure

This thesis is structured in the following way: Section 2 contains the related work already done on the same subject. Section 3 provides a brief theoretical background on clustering, Mutual Information, Reinforcement Learning and Multi-Agent systems. In Section 4 presents a more technical and detailed description of the problem, while Section 5 proposes our solution, describing the algorithm and the testing environment. Section 6 reports the results of our experiments and Section 7 discusses these results and draws conclusions.

2. Related Work

Prior work has shown that reinforcement learning can be competitive for network critical-infrastructure control when realistic simulators and benchmarks are available. In the context of power-system operation, the Grid2Op ecosystem formalizes grid operations as a sequential decision-making problem and has been used to benchmark RL agents under changing operational constraints. Regarding this, in 2020 the french company Réseau de Transport d'Électricité (RTE) promoted a challenge called Learning to Run a Power Network (L2RPN), where several solutions like the one by Yoon et al. [1] were proposed. This approaches still had some limitations that were addressed in the Marot et al. [2] work. Regarding the context of train dispatching, there has been a similar

challenge based on Flatland simulator where the problems addressed are quite similar: as we can see in the Li et al. [3] work a global centralized control in Flatland is computationally unfeasible at scale. Even when using a multi-agent approach, having thousands of agents into the same network can make the computational runtime grows exponentially. We follow the path presented by Liu et al. [4] at IEE which already shows some promising results regarding factorization and multi-agent approaches. We try to solve the clustering problem by presenting a dynamic approach that is suitable for different contexts.
A common method used for network control is centralized training with decentralized execution (CTDE), where agents act on local information but are trained using the whole global state. Several Multi-Agent Reinforcement Learning (MARL) algorithms have been proposed in this setting, including approaches that leverage actor-critic methods and factorization of the values. Techniques such as Value Decomposition Networks (VDN) and QMIX learn a factorized representation of the global value function, allowing decentralized policies Sunehag et al. [5], Rashid et al. [6]. These methods solve some of the scalability problem, but assume a predefined decomposition without solving the problem of how to meaningfully partition the network.
Since our work relies on clustering, it is related to graph partitioning methods applied to infrastructures. In power systems, partitioning has already been used to define operational zones based on data, with the goal of localizing control decisions like in the work from Cotilla Cotilla-Sanchez et al. [7]. Similar ideas have been applied also to transportation networks like in the paper by Corman Corman et al. [8]. However all this approaches are driven by domain specific data, while our objective is to create a domain agnostic approach that can be used in different settings.

3. Theoretical Background

3.1. Mutual Information

Mutual information is defined in probability theory as a measure of the mutual dependence between two variables. In our case, is what stands at the base of our algorithm since it allows us to quantify how much a node's state is statistically dependent on an action, and to determine whether it is important for a specific agent to observe its state. The logic behind this measure is to determine how much knowing one variable can reduce the uncertainty of another variable. When X and Y are discrete and their joint distribution is known (or well estimated), the mutual information can be computed using the formula:

$$I(X;Y) = \sum_{x\in\mathcal{X}} \sum_{y\in\mathcal{Y}} p(x,y)\, \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right) \tag{1}$$

In this context, we can also use the relation:

$$I(X;Y) \leq \min\{H(X),\, H(Y)\} \tag{2}$$

which tells us that the MI of two variables is always bounded by the minimum of the entropies of the two variables.
This is important since it allows us to normalize the computed values: without normalization, a value that seems high could in reality be low compared to the maximum obtainable value to that specific variable, while a low mutual information could be quite significant in some cases if it is bounded by a variable with very few possible values. For instance, if we consider a binary variable, the maximum obtainable mutual information (when the variables are perfectly dependent) is

$$\log(2) \approx 0.69 \tag{3}$$

so even a value of 0.4, which might seem small, would instead indicate a strong dependence.
This works for the case of discrete variables, but in a real world environment this is not always the case. In a continuous scenario, mutual information is defined by replacing sums with integrals over densities:

$$I(X;Y) = \int_{\mathcal{X}} \int_{\mathcal{Y}} p(x,y)\, \log\left(\frac{p(x,y)}{p(x)\,p(y)}\right)\, dx\, dy, \tag{4}$$

but the densities are often unknown and have to be estimated from the samples. To do this we can rely on **estimators** which provide a good estimate of the actual densities. In our specific case, we used a non-parametric estimator based on nearest-neighbor statistics named **Kraskov-Stögbauer-Grassberger (KSG)** estimator, introduced in *Estimating Mutual Information* by Kraskov et al. [9]. Given N samples $\{(x_i, y_i)\}_{i=1}^{N}$, for each point it computes the distance to its k-th nearest neighbor in the joint space. Typically the distance measure

used are Chebyshev or max norm as in the original formulation, but it can be adapted to different measures. This distance defines a neighborhood which is bigger in sparse regions and gets smaller in more dense ones. The radius of it is used to count how many samples fall into this spaces. One of the most common KSG forms, which is also the one used in our implementation, estimates mutual information as

$$\hat{I}_{\text{KSG}}(X;Y) = \psi(k) + \psi(N) - \frac{1}{N}\sum_{i=1}^{N}\Big(\psi(n_x(i)+1) + \psi(n_y(i)+1)\Big), \tag{5}$$

where $\psi(\cdot)$ is the digamma function.
We can control the parameter k to change the trade-off between bias and variance: a small value of k reduces bias, while a larger values smooth the estimate reducing variance.
Talking about complexity, the real issue with the KSG estimator is the k-nearest-neighbor search: it implies computing all pairwise distance between the elements, leading to a complexity of $O(N^2)$, which is not good for big scale applications. There are some techniques to reduce this cost using spatial indexing structures (like KD-trees or Ball-trees), which can usually achieve a computation of $O(DN \log N)$. However, as the dimension increases the advantages that these methods bring are less noticeable, so for high dimensionality contexts is still good practice to keep a low k value to limit runtime in exchange for some estimator variance.

3.2. Clustering

Clustering is a technique widely used in data analysis that aims to divide a set of elements into groups (clusters) such that elements within the same cluster share similar characteristics. It is one of the most common approaches in Unsupervised Learning, meaning that it does not require labeled data: instead, it tries to discover structure directly from observations by exploiting similarity measures or distance.
In our specific scenario, data are represented as a network: in these cases, clustering is often formulated as a graph partitioning problem where nodes represent components and edges represent physical interactions or possible routes. In this setting, the objective is to obtain clusters with strong internal connectivity while limiting inter-cluster coupling, so that each sub-network can be controlled more independently and efficiently. In power systems applications, partitioning has been applied to divide the complete grid into areas using both static and dynamic criteria, as in the work by Zhang et al. [10]. Very similar principles can be applied to transportation networks when the problem is too difficult to be solved as a single unit and needs to be factorized.
Clustering methods can be grouped depending on how similarity is defined and how the clusters are formed:

- **Partitioning methods** like k-means assume that data points can be represented in a metric space and the objective is to minimize the dispersion around a centroid; they are quite efficient but have a big drawback: the number of clusters has to be fixed in advance.
- **Hierarchical methods** build tree of clusters to organize data based on similarity. It can have two different approaches: the agglomerative one start from small clusters and aggregates them, while the divisive approach starts from a single big cluster and splits it.
- **Density-based methods** identify clusters as high density regions separated by low density regions.
- **Model-based methods** assume that data are generated from a finite mixture of underlying probability distributions and identifies clusters estimating the parameters of these distributions, rather that relying on distance metrics.

In cases where data can be represented as a graph, it is usually more convenient to use a **graph-based** approach. These methods are good for network domains since they can incorporate both the topology of the system and edge weights.
In our case we followed a graph-based perspective, but the edge weights are based on the data coming from our MI matrix. A similar approach with MI-weighted graphs and thresholding of weak dependencies was proposed in the works from Butte and Kohane [11] and Margolin et al. [12]

3.3. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns how to make decisions by interacting with an environment. At each time step t the agent observes the current state of the environment S_t, selects an action A_t from the available action space, receives a reward R_{t+1} and transitions to a new state S_{t+1}. The agent's objective is to learn which are the best actions to choose to maximize the expected reward over time. Sutton and Barto [13]
A standard formalization of RL is the **Markov Decision Process (MDP)**, defined by the tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma, \rangle, \tag{6}$$

where:

- $\mathcal{S}$ is the **state space**, a defined set of states
- $\mathcal{A}$ is the **action space**, a defined set of actions
- $P(s' \mid s, a)$ is the **transition probability function**, which determines the probability of moving to state s' from state s taking action a. In many real-world problems P is unknown and can only be sampled through multiple interactions.
- $R(s, a, s')$ is the **reward function**, which returns a scalar value to the agent after a transition. Having a robust reward function is fundamental since it represents the final objective and guides the agent towards the desired behavior.
- $\gamma \in [0, 1)$ is the **discount factor** that determines how future rewards are weighted with respect to the immediate ones. Smaller values encourage short-term gains, while values closer to 1 aim towards long-term rewards.

The discounted return from time t can be computed as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{7}$$

and the goal is to maximize the expected return.
A **policy** π is what Agents have to rely on to select the action to take given the current state, and it can be deterministic or stochastic. In the stochastic case, a policy is seen as a probability distribution over actions conditioned on the state. Formally, it can be defined as:

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \qquad (s, a) \mapsto \pi(a \mid s), \tag{8}$$

such that

$$\sum_{a \in \mathcal{A}} \pi(a \mid s) = 1 \qquad \forall s \in \mathcal{S}. \tag{9}$$

The policy that leads to the maximum possible return in the given environment is defined as the optimal policy π^*. Considering the definition of G_t given above, we can see the optimal policy as

$$\pi^* \in \arg\max_{\pi} \mathbb{E}_{s \sim \mu}[G_t \mid S_t = s, \pi] \tag{10}$$

where μ denotes a distribution over the states.
To evaluate how good a policy is we can use **value functions**. The state-value function describes the expected return obtained when following policy π from state s and is defined as

$$V^{\pi}(s) = \mathbb{E}[G_t \mid S_t = s, \pi] \tag{11}$$

The action-value function instead describes the expected return obtained when starting from a state s, taking action a and then following policy pi, and it is defined as

$$Q^{\pi}(s, a) = \mathbb{E}[G_t \mid S_t = s, A_t = a, \pi]\,. \tag{12}$$

3.4. Multi-Agent Reinforcement Learning

Multi-Agent Reinforcement Learning (MARL) is a specific RL paradigm in which multiple agents interact with the same environment at the same time. Each agent has its own policy and follows it to interact with the environment, meaning that at each time step the next state is going to be determined by the combination of the agent's actions. Depending on how the system has been designed, they can cooperate towards a common objective, have conflicting objectives or operate in a mixed setting as seen in the work of Busoniu et al. [14].
The possible approaches in MARL are different and based on the degree to which learning and actions are centralized. In fully centralized learning, for example, the system is seen as a single decision-maker that observes the global state and selects a joint action for all the agents through a single policy. This approach presents some significant scaling problems and, in many cases, is difficult to apply since agents do not have access to the global state. On the other hand, a fully decentralized approach assumes that each agent learns its own policy independently using only local observations. This is much more similar to a real world application, but can lead to convergence problems since the state seen by an agent could change because of the actions taken by a different agent. There are also some solutions that sit in the middle, like centralized training with decentralized execution, where during training agents are given additional centralized information, but then they act only based on their local observation.
Although MARL is surely a common approach to improve scalability, it introduces some problems that we

don't have in classic single agent RL. One of the main issues is **non-stationarity**: since every agent updates its policy at the same time in the learning process, the environment from the perspective of a single agent is non-stationary, which brings to more fluctuations during training. Another issue is **partial observability**: since each agent sees only its local observation, it is harder for it to take decisions according to what the other agents do in the environment. The last big issue that happens in cooperative problem is **coordination and credit assignment**: when agents have a shared reward, it is very hard to determine which agent's actions contributed the most to improvement or failure, making it harder to understand which policy has to be modified.

3.5. Proximal Policy Optimization

The Proximal Policy Optimization (PPO) algorithm is a policy gradient algorithm, meaning that its objective is to optimize the policy to increase the expected return as much as possible. It is designed to be used in both discrete and continuous action spaces, making it very versatile, and is based on an **actor-critic** structure: the actor policy $\pi_\theta(a \mid s)$ is optimized to maximize the expected return, while the critic value function $V_\phi(s)$ is trained to approximate the expected return and it is important to minimize the variance of the policy-gradient estimate.

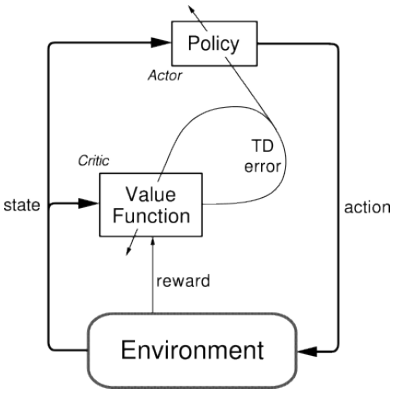


Figure 1: Actor-critic architecture

The main idea behind PPO is to prevent large policy updates optimizing a clipped surrogate objective, which is based on the probability ratio between the old policy and the new one. In practice, it is trained with a combined objective that also includes a loss function for the critic and often an entropy bonus to encourage the exploration of different policies. A common formulation of this is:

$$L(\theta, \phi) = L^{\text{CLIP}}(\theta) - c_1 \, \mathbb{E}_t \left[\left(V_\phi(s_t) - \hat{R}_t \right)^2 \right] + c_2 \, \mathbb{E}_t \left[\mathcal{H}\big(\pi_\theta(\cdot \mid s_t)\big) \right], \tag{13}$$

where $\hat{R}_t$ is an empirical return, $\mathcal{H}$ is the entropy and c_1, c_2 are the weighting coefficients. The training process is iterative, performing multiple epochs of gradient updates on the same data.
Here, there is a pseudocode version of the algorithm as in the original formulation by Schulman et al. [15]:

Algorithm 1 Proximal Policy Optimization (PPO)

Require: Discount factor γ, GAE parameter λ, clip parameter ϵ, number of steps per rollout T, number of epochs K, minibatch size M, coefficients c_1, c_2, learning rate α.

Initialize policy parameters θ and value-function parameters ϕ
for iteration $= 1, 2, \ldots$ **do**
  Set $\theta_{\text{old}} \leftarrow \theta$
  Collect rollout $\{(s_t, a_t, r_{t+1})\}_{t=0}^{T-1}$ by running $\pi_{\theta_{\text{old}}}$ in the environment
  Query $V_\phi(s_T)$ for the terminal bootstrap value
  Compute value predictions $V_\phi(s_t)$ for all $t \in \{0, \ldots, T-1\}$
  Compute advantage estimates $\hat{A}_t$ via GAE and return targets $\hat{R}_t$ for all t
  for epoch $= 1, 2, \ldots, K$ **do**
    Shuffle indices $\{0, \ldots, T-1\}$ and split into minibatches of size M
    for each minibatch $\mathcal{B}$ **do**
      Compute importance ratios for $t \in \mathcal{B}$:

$$\rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

12: Compute clipped surrogate objective:

$$L_{\mathcal{B}}^{\text{CLIP}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \min \Big(\rho_t(\theta)\, \hat{A}_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\, \hat{A}_t \Big)$$

13: Compute value loss:

$$L_{\mathcal{B}}^{\text{VF}}(\phi) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \Big(V_\phi(s_t) - \hat{R}_t \Big)^2$$

14: Compute entropy bonus:

$$S_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \mathcal{H}\big(\pi_\theta(\cdot \mid s_t)\big)$$

15: Update actor parameters by gradient ascent:

$$\theta \leftarrow \theta + \alpha\, \nabla_\theta \Big(L_{\mathcal{B}}^{\text{CLIP}}(\theta) + c_2\, S_{\mathcal{B}}(\theta) \Big)$$

16: Update critic parameters by gradient descent:

$$\phi \leftarrow \phi - \alpha\, c_1\, \nabla_\phi L_{\mathcal{B}}^{\text{VF}}(\phi)$$

17: **end for**
18: **end for**
19: **end for**

4. Problem formulation

The environments treated in this work are network-based systems that can be naturally represented as graphs because of their structure, where nodes are physical components and edges are their connections. The objective is usually global (power balancing, train delay reduction, etc.), but the interactions are controlled locally.
These systems are modeled as MDPs, as we have seen before, whose dimensionality grows drastically with the size of the graph, making a fully centralized approach almost unfeasible in many cases. For this reason, the main objective of our work is to obtain a **factorization** of the original MDP into smaller sub-networks that can be handled on reduced state/action spaces while preserving as much performance as possible. One of the main drawbacks of this approach is that we are giving up the Markovian assumption: even if the global process is Markovian in the full state, the local observation available to an agent is only part of the global state and is generally not sufficient to predict future transitions without additional information. Moreover, each agent could see the environment as non-stationary, even if it is stationary when considered as a whole: from a single agent perspective, its state space dynamics are influenced by other agents actions that he does not observe.
To factorize the problems as we desired, we had to create an MDP formulation for both to then test our approach.

4.1. Power Grid problem

The first environment we created is a simplified power-balancing problem designed to test our approach in a simple but scalable environment. The network is represented as a chain graph with N generators connected to $N+1$ power lines. Each generator $i \in \{0, \dots, N-1\}$ injects power into the network split between the two adjacent lines i and $i+1$. The load on each line depends on the split decision of the adjacent generators, creating strong coupling between the close components along the chain.
Both generator outputs and actions are discrete, which allows us to use the mathematical formulas presented in the theory part regarding MI without the introduction of estimators. The output of generators is defined as

$$G_t = \big(G_t^{(0)}, \dots, G_t^{(N-1)}\big), \tag{14}$$

where each $G_t^{(i)}$ evolves as a random walk with a different set of steps and range. The splitting among the lines is controlled by a set of admissible ratios, which is

$$\mathcal{P} = \{(100, 0), (80, 20), (60, 40), (40, 60), (20, 80), (0, 100)\}\,. \tag{15}$$

Each generator is assigned a specific split index that describes how much power is sent to the adjacent lines. We see the environment as an MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, where the state is defined as

$$S_t := (L_t,\, G_t,\, U_t), \tag{16}$$

where L_t are the current line loads, G_t is the generator output, and U_t is the split configuration of the generators. The actions instead are index adjustements among the possible splits

$$A_t := \big(A_t^{(0)}, \ldots, A_t^{(N-1)}\big), \qquad A_t^{(i)} \in \{-1, +1\}, \tag{17}$$

so that each action can redistribute a generator's output on its adjacent lines. The transition is given by the evolution over time of the generator outputs and the deterministic recomputation of line loads given by the splits.
The objective of each generator is to balance the loads across its two adjacent lines. The cost of the generator is then defined as the squared deviation of the loads from their local average

$$c_t^{(i)} = \Big(L_t^{(i)} - \bar{L}_t^{(i)}\Big)^2 + \Big(L_t^{(i+1)} - \bar{L}_t^{(i)}\Big)^2, \qquad \bar{L}_t^{(i)} = \frac{L_t^{(i)} + L_t^{(i+1)}}{2}. \tag{18}$$

The global reward is instead the negative total cost

$$R_{t+1} = -\sum_{i=0}^{N-1} c_{t+1}^{(i)}. \tag{19}$$

In addition to the fully discrete formulation described above, we also implemented a continuous version of the same problem to test the performance of the clustering approach when both state and action variables are not discrete. In this context, the generator output $G_t^{(i)}$ evolves with a continuous random walk

$$G_t^{(i)} = \mathrm{clip}\Big(G_{t-1}^{(i)} + \varepsilon_t^{(i)},\ \mathit{low},\ \mathit{high}\Big), \qquad \varepsilon_t^{(i)} \sim \mathcal{N}(0, \sigma_i), \tag{20}$$

where each generator is assigned a different sigma to introduce different dynamics across them. The discrete index representation of the splits is instead replaced by a continuous split ratio for each generator

$$U_t = \big(U_t^{(0)}, \ldots, U_t^{(N-1)}\big), \qquad U_t^{(i)} \in [0, 1], \tag{21}$$

where $U_t^{(i)}$ is the output of the generator i in line i and $1 - U_t^{(i)}$ in line $i+1$. The outputs can still be deterministically computed at each time step with the formula

$$L_t^{(j)} = \sum_{i \in \{j-1, j\} \cap [0, N-1]} p_{\to j}^{(i)}\, G_t^{(i)}, \tag{22}$$

In this case, actions are a continuous adjustment of the split ratios

$$A_t = \big(A_t^{(0)}, \ldots, A_t^{(N-1)}\big), \qquad A_t^{(i)} \in [-\eta, +\eta], \tag{23}$$

applied as

$$U_{t+1}^{(i)} = \mathrm{clip}\Big(U_t^{(i)} + A_t^{(i)},\ 0,\ 1\Big), \tag{24}$$

where η is the maximum change in each step, which in our case was set to 10%. The rewards are kept the same, so the dynamics of the system are exactly the same, the difference is only in the state and action spaces.
We created three different versions of the system with two, four and eight generators to test scalability; this is an example of a four generators scenario

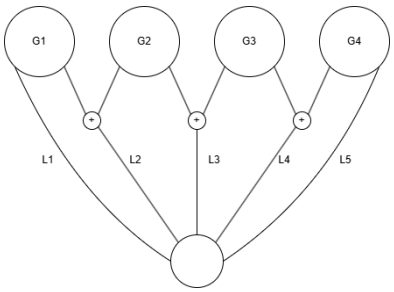


Figure 2: Power grid architecture example

4.2. Flatland problem

Flatland is a multi-agent railway simulator designed to study train dispatching and rescheduling problems on networks represented as a two dimensional grid. The time steps are discrete and at each of them every train, which in the standard formulation is an agent, selects an action and advances by one step.
The global state can be seen as a combination of two main factors: the railway structure (switches, rails, dead-ends, etc.) and the status of each train, such as the position, direction, target destination, and so on. Flatland provides three different ways in which the agent can observe this state:

- **Global observation:** a grid representation of the whole environment
- **Local grid observation:** a representation of the grid section around the target. This is useful for scaling better on large grids, but the target of an agent may fall out of its observable space, which could be a problem.
- **Tree observation:** built by spanning a four branched tree from the current position of the train. Each branch follows the allowed transitions until a cell with multiple allowed transitions is reached.

For each agent, the action space is defined as:

$$\mathcal{A}^i = \{\texttt{NOOP}, \texttt{LEFT}, \texttt{FORWARD}, \texttt{RIGHT}, \texttt{STOP}\} \tag{25}$$

but the actual possible actions at a specific position are often limited by the railway structure.
Flatland may be formulated as a centralized MDP with a single controller that considers the joint action of all N trains. However, because of scalability problems and the fact that trains usually have access only to local information, it is generally treated as a multi-agent problem, where each agent learns a local policy and then they all aim at a shared objective, like limiting the delay times overall.

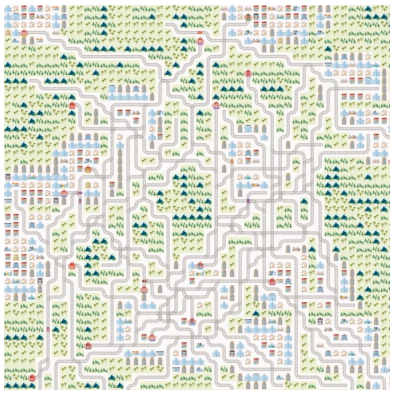


Figure 3: Flatland railway example

4.2.1 SwitchFL

SwitchFL is a custom library built on top of Flatland simulator [16]. It provides a switch-centered formulation of the railway control problem: instead of having the trains as agents, the decisions are taken by the railway switches. Each of these switches is treated as a single agent that becomes active when a train is approaching and it has to take a decision on where to send it. This is closer to a real world representation, since it is not the trains taking decisions but the dispatchers, and in our formulation it allows us to use the graph representation and cluster switches that are close and have high coupling, while having trains that move around the graph would have created more issues.
The MDP formulation is then way different: the environment is defined over a set of switch agents $v \in \mathcal{V}$, each associated with a local observation that is a summary of the traffic conditions around that switch. A typical observation includes information such as:

- The current occupation status and the status of outgoing direction.
- The desired target of the incoming train
- Local scheduling indicators such as delay

The action space also becomes switch-dependent: each agent can decide which internal routing to apply among the feasible ones. We can then represent the action space of a switch v as:

$$\mathcal{A}^v = \{0, \dots, m_v - 1\}, \tag{26}$$

where m_v depends on the switch type (not every switch can route on any direction). The types of switches are:

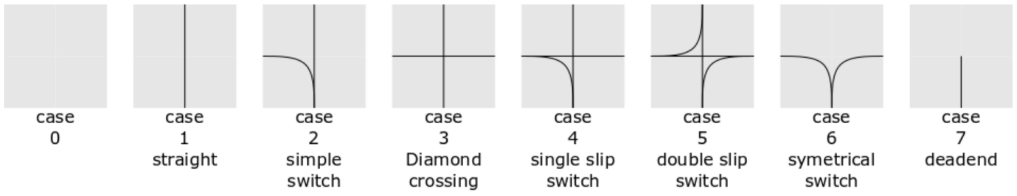


Figure 4: Set of the possible types of switches

but we noticed that some of them are way less likely to be present and so some actions are taken very few times. We can represent the action space of a switch v as

$$\mathcal{A}^v = \{0, \dots, m_v - 1\} \tag{27}$$

where m_v depends from the switch type but has a maximum value of 7, with only the *double slip switch* being able to perform all of them.
This formulation leads to a transition structure: a decision is taken at a switch anytime a new train approaches.

5. Algorithm proposal

To factorize the original problem into approximately independent sub-problems, we propose a data-driven clustering method based on MI. Given a transitions dataset $\{(S_t, A_t, S_{t+1})\}$, we define the input vector $Z_t = (S_t, A_t) \in \mathbb{R}^{n_z}$ and compute a **transition MI matrix** $M \in \mathbb{R}^{n_z \times n_s}$, where each cell is

$$M_{ij} \;=\; I\Big(Z_t^{(i)};\, S_{t+1}^{(j)}\Big)\,. \tag{28}$$

Since MI is highly dependent on variable cardinalities, we noticed that in cases where we had a small set of possible values (like the actions in the power grid problem presented above) the values were bounded to very low values. To avoid this, we introduced a normalization process in which we use the relation $I(X;Y) \leq \min\{H(X), H(Y)\}$ to define an upper bound with the equation

$$M_{ij} \;=\; I\Big(Z_t^{(i)};\, S_{t+1}^{(j)}\Big)\,. \tag{29}$$

so that $\bar{M}_{ij}$ provides a consistent dependence score.
This works for the discrete version of the problem, but can't be applied to the continuous version of it since both generators and actions transition distributions are unknown. In this case, we use the KSG algorithm presented in section 3.1 to estimate mutual information. This also means that the relation $I(X;Y) \leq \min\{H(X), H(Y)\}$ doesn't hold anymore, since the entropy of the variables can't be computed, so we don't apply normalization in this case.
The idea at the base of the algorithm is that each agent should be able to observe every state in which it can perform an action. Given this, we start from actions to create the clusters and we create as many cluster as the number of actions we have by placing in each cluster every state that has a MI value higher than a given threshold. At this point, we evaluate any possible merges between the clusters using our evaluation function, which penalizes high coupling relations that are not considered and low coupling elements placed in the same cluster.

Algorithm 2 Evaluation function

procedure EvaluateMerge(M, rows, cols, shape_1, shape_2, α, δ)
  $fullScore \leftarrow 0$; $sepScore \leftarrow 0$
  $B \leftarrow M[\text{rows}, \text{cols}]$ ▷ submatrix induced by the candidate merged cluster
  for $i = 0$ **to** $\text{rowCount}(B) - 1$ **do**
    for $j = 0$ **to** $\text{colCount}(B) - 1$ **do**
      if $B[i,j] < \delta$ **then**
        $fullScore \leftarrow fullScore + \alpha\,(1 - B[i,j])$
      end if
      if $i < \text{shape}_1^{(r)}$ **and** $j < \text{shape}_1^{(c)}$ **and** $B[i,j] < \delta$ **then** ▷ inside top-left block (cluster 1)
        $sepScore \leftarrow sepScore + \alpha\,(1 - B[i,j])$
      else if $i \geq \text{rowCount}(B) - \text{shape}_2^{(r)}$ **and** $j \geq \text{colCount}(B) - \text{shape}_2^{(c)}$ **and** $B[i,j] < \delta$ **then** ▷ inside bottom-right block (cluster 2)

```
            sepScore ← sepScore + α (1 − B[i, j])
        else if (i ≥ shape_1^(r) ∧ j < colCount(B) − shape_2^(c)) or (i < rowCount(B) − shape_2^(r) ∧ j ≥ shape_1^(c))
    then                                          ▷ off-diagonal region between the two blocks
            if B[i, j] ≥ δ then
                sepScore ← sepScore + (1 − α) B[i, j]
            end if
        end if
    end for
  end for
  return (sepScore ≥ fullScore, fullScore)
end procedure
```

This allows to understand if two agents are highly dependent from each other and if it could be beneficial to merge them into a single cluster. At this point, we consider all the leftovers states, which are states that have no relevant coupling with any actions. However, they could have strong coupling with other states and be useful to be observed by an agent, so we evaluate if it is beneficial to add them to one or more of the previously created clusters.

We allow states to be placed inside multiple clusters since a state that an agent does not control may still be very important to observe to get to a good policy. Our first idea to keep Markov assumption and have optimal result was to merge clusters with states in common analyzing dependencies between clusters, but this often led to a very low number of agents, which did not bring any benefits in terms of convergence and training time. We then decided to give away this property and allow multi clustering to ensure good performances while keeping the benefits of a multi-agent approach.

The full algorithm procedure is the following

Algorithm 3 Mutual-information clustering of transition dependencies

```
procedure AnalyzeMatrix(M, α)
  n_s ← col-count of M; n_a ← row-count of M − n_s
  clusters ← [ ]
  for i = n_s to n_s + n_a − 1 do                          ▷ initial discovery
    C ← {i}
    for j = 0 to n_s − 1 do
      if M[i, j] > threshold then
        C ← C ∪ {j, j + n_s + n_a}
      end if
    end for
    clusters ← clusters ∪ {C}
  end for
  while true do                                            ▷ greedy merging
    s* ← +∞; bestPair ← null
    for (C_i, C_j) in clusters × clusters where i ≠ j do
      C' ← Deduplicate(C_i ∪ C_j)
      (b, s) ← EvaluateMerge(M, C', shape(C_i), shape(C_j), α)
      if b = true and s < s* then
        s* ← s; bestPair ← (C_i, C_j)
      end if
    end for
    if bestPair = null then
      break
    end if
    clusters ← MergeClusters(bestPair, clusters)
  end while
  for s = 0 to n_s − 1 do                                  ▷ add unclustered states
    if s ∉ any cluster then
      singleton ← {s, s + n_s + n_a}
      for C in clusters do
        (b, sc) ← EvaluateMerge(M, C∪ singleton, shape(C), (1, 1), α)
        if b = true then
          C ← C∪ singleton
        end if
      end for
```

end if
end for
return clusters
end procedure

5.1. Computational complexity

Let N be the number of transitions in the dataset, n_s the state dimension, n_a the action dimension, and $n_z = n_s + n_a$ the size of $Z_t = (S_t, A_t)$. Building the transition MI matrix $M \in \mathbb{R}^{n_z \times n_s}$ requires estimating $n_z\, n_s$ pairs of mutual information.
In the discrete scenario, each MI term can be computed in $O(N)$ time, since it only requires one pass for each sample, reaching an overall cost of $O(n_z\, n_s\, N)$.
In the continuous case, KSG introduces a higher complexity: for each element (i, j) we need to perform a k-nearest-neighbor search. As we have seen, this depends on the k value, which we decided to keep quite low (5 in the current implementation) since it was already good at estimating and we did not want to risk to have the time grow a lot. In the best case scenario this part takes $O(n_z\, n_s\, N \log N)$, and in the worst one $O(n_z\, n_s\, N^2)$ as we have seen in the 3.1 section.
Once we computed the MI matrix, the clustering stage is dependent on the number of actions: the initialization of the clusters costs $O(n_a\, n_s)$, the greedy merging step is instead dependent to the number of clusters created in the first step and therefore on the number of actions. Having at most $n_a - 1$ merges, the number of evaluation needed will be at most $O(n_a^2)$.
In practice, this means that if the number of actions is limited, the runtime is higher for the MI estimation step, while with a big number of actions the clustering part is going to take more time. Given that usually the number of actions is way more limited compared to the number of states, it is a good thing that the algorithm complexity scales with it.

6. Experimental results

In this section we will report the empirical results obtained on the toy Power Grid environment, comparing a centralized single-agent PPO baseline with our clustered multi-agent approach. We considered both the discrete and the continuous versions of the problem analyzing both the quality of the clustering and the impact on training speed and stability.

6.1. Discrete Power Grid results

Starting from the discrete version, we chose to present our results using a graph representation where state variables are drawn as square nodes and action variables as circular nodes, while colored boundaries represent the clusters returned by our algorithm. Our initial representation was a matrix version of this, but that did not allow to represent intersections between different clusters, that are very visible with this visualization. The resulting factorization tends to group actions with the states that they influence the most.

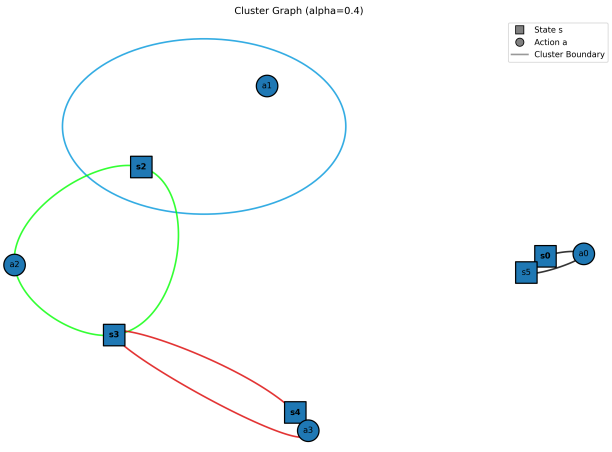
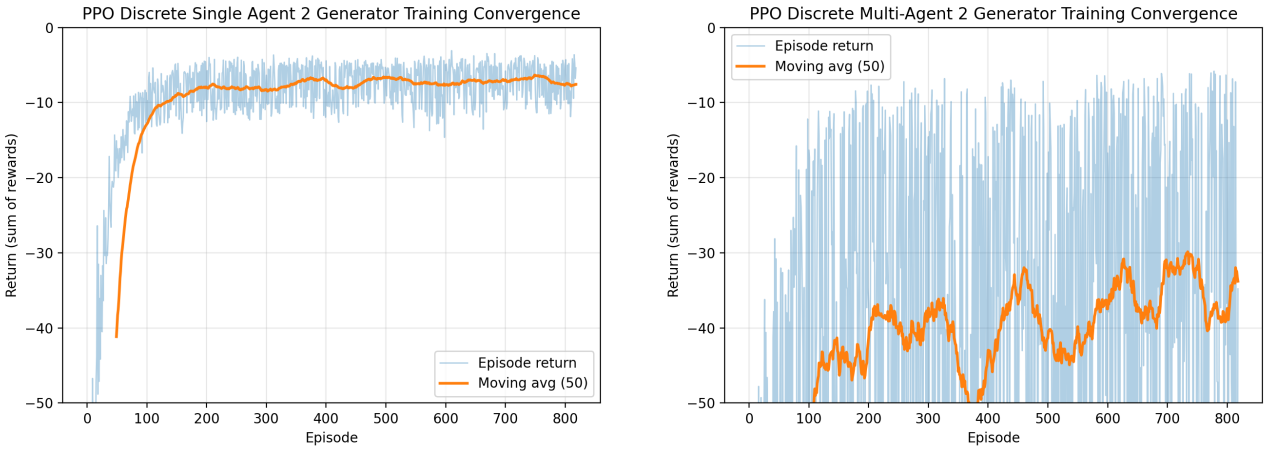


Figure 5: Example of clustering graph for the discrete Power Grid problem with 4 generators

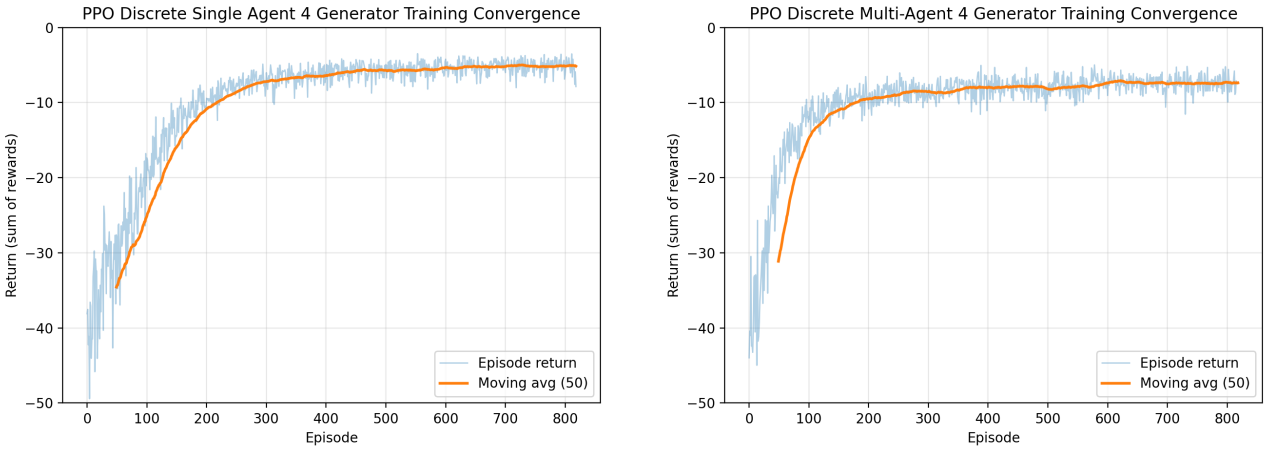
Every action is clearly present in the graph due to the way we designed the algorithm, but not every state is: some of them may in fact be not strongly related to any other state or actions, being irrelevant for the system dynamics, and so can be ignored.
To evaluate the effects of this factorization, for each configuration $N \in \{2, 4, 8\}$, we compare a single-agent PPO trained on the full state/action space against a clustered multi-agent PPO controller, where each agent acts only on the reduced set of the corresponding cluster. For the $N = 2$ version of the problem, the multi-agent version exhibits convergence and stability issues. We expected a similar behavior since clustering a problem this small with only two actions and four states provides no advantage: the task is very simple and the factorization just introduces coordination problems between the different agents. In the $N = 4$ version we already see some promising results: both approaches converge, but the multi-agent version reaches a stable value in less episodes. The final performance at this stage is still slightly worse than the centralized baseline, suggesting that at this scale factorization already accelerates learning but may still introduce a small approximation error. At $N = 8$ is where the difference becomes clearly noticeable: the multi agent approach converges way faster than the centralized one and reaches a stable policy many episodes before. This proves the main motivation of the proposed method: as the dimensionality increases, the clustered approach become more effective reducing the complexity of the learning problem and decreasing training time.



(a) Single agent, $N = 2$.

(b) Multi-agent, $N = 2$.

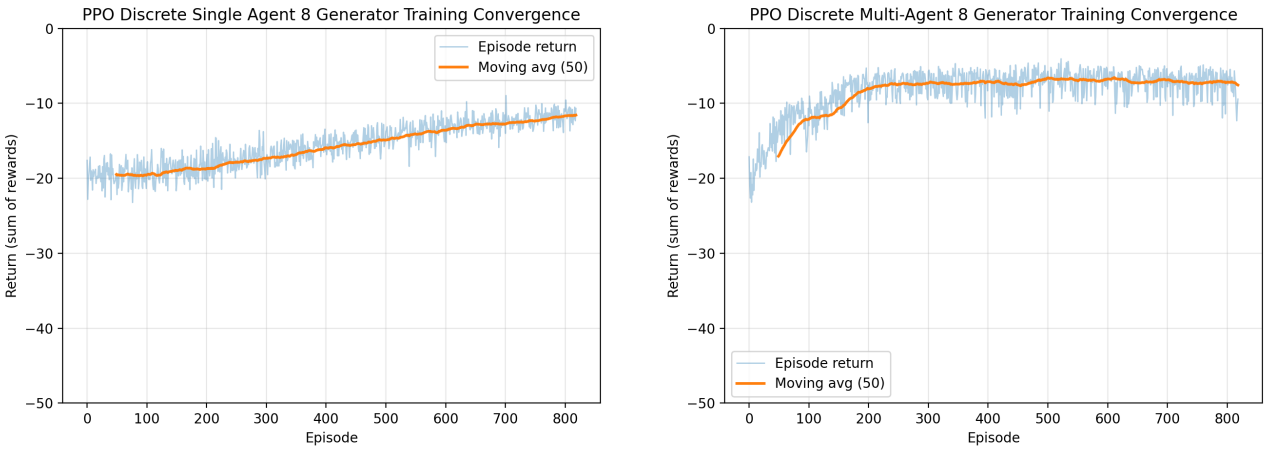
Figure 6: Discrete Power Grid training curves for $N = 2$ generators. The multi-agent setting shows instability and does not provide an advantage at this scale.



(a) Single agent, $N = 4$.

(b) Multi-agent, $N = 4$.

Figure 7: Discrete Power Grid training curves for $N = 4$ generators. Both approaches converge, while the clustered multi-agent version converges faster but to a slightly worse final return.



(a) Single agent, $N = 8$.

(b) Multi-agent, $N = 8$.

Figure 8: Discrete Power Grid training curves for $N = 8$ generators. The clustered multi-agent approach converges significantly faster than the centralized baseline, showing the scalability advantage of the factorized version.

6.2. Continuous Power Grid results

Considering the continuous variant of the problem, the main difference is that mutual information now has to be computed using the KSG estimator [9] rather than the classic discrete approach.
From the clustering point of view, which should have been the hardest part, the results are encouraging since the estimator worked quite well and we were able to group states and actions in a coherent way, as you can see in the following graph

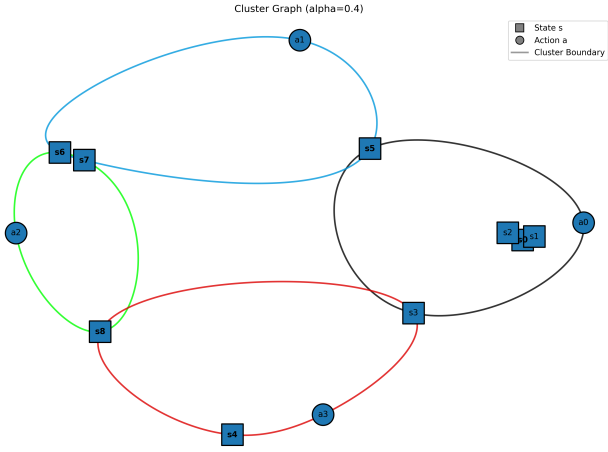
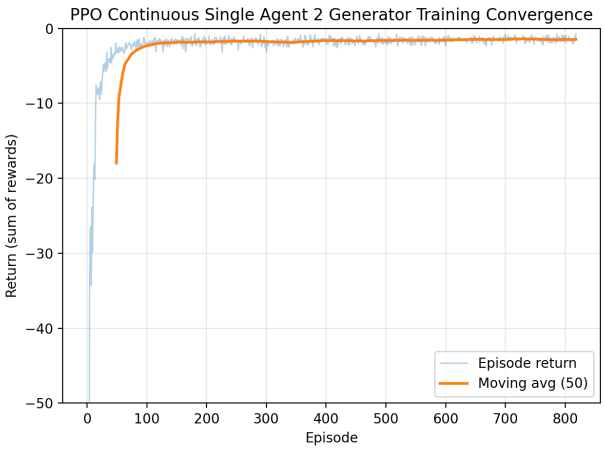
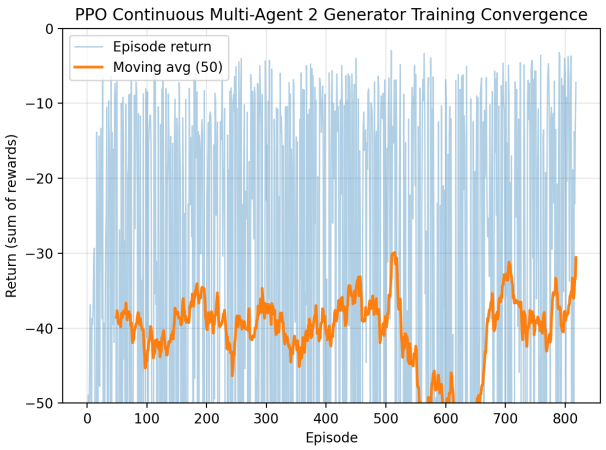


Figure 9: Example of clustering graph for the continuous Power Grid problem with 4 generators

We can see that it considers a few more state than the discrete version, but it should not be a problem since we have seen that it still converges quite fast.
Talking about training performances, the continuous version of the single-agent PPO version has really fast converges on its own, making it very hard to spot any possible improvement due to our approach. In the $N = 2$ and $N = 4$ versions of the problem the single-agent approach reaches better results, while for the $N = 8$ version the performance is very similar. This suggests that, even if the clustering remains coherent even when using MI estimators, the benefit of it are dependent on the difficulty of the learning problem: at the tested scale the centralized approach for the continuous version is still better, but we expect the multi-agent approach to become better as dimensionality increases.

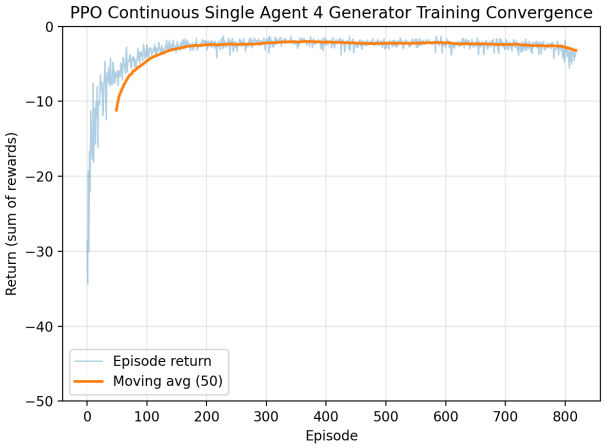


(a) Single agent, $N = 2$.

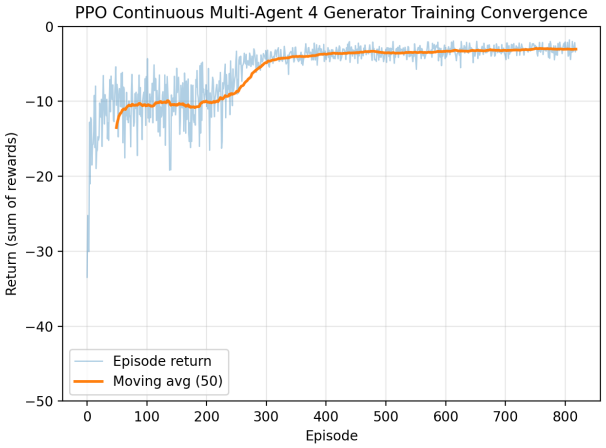


(b) Multi-agent, $N = 2$.

Figure 10: Continuous Power Grid training curves for $N = 2$ generators. The single-agent baseline performs better at this scale.



(a) Single agent, $N = 4$.



(b) Multi-agent, $N = 4$.

Figure 11: Continuous Power Grid training curves for $N = 4$ generators. The single-agent baseline achieves similar final performance, while the multi-agent factorization just takes more time to converge.

(a) Single agent, $N = 8$.

(b) Multi-agent, $N = 8$.

Figure 12: Continuous Power Grid training curves for $N = 8$ generators. The two approaches show similar behavior at this scale, suggesting that the advantage of factorization may become clearer only for larger instances.

6.3. Flatland results

While SwitchFL uses an asynchronous interface based on transitions that happen only when a train approaches, we need a time aligned transition dataset for our MI creation. For this reason we exported an observation buffer and an action buffer from the environment and we manipulated the data to be usable for our solution. Each observation row encodes the identity of the acting node, defined by a tuple of coordinates $(x, y) =$ (`row[0]`, `row[1]`), together with the target station identifier $(u, v) =$ (`row[3]`, `row[4]`)of the incoming train, the delay in `row[5]` and the simulation timestamp `row[7]`. At each observation row corresponds an action row with the action taken by the corresponding switch.

We had to reconstruct the transitions so we started from the set of unique stations $\{(u, v)\}$ observed, obtaining a fixed dimension n_s for the states. We also enumerated the set of unique switch nodes $\{(x, y)\}$ defining the action dimension n_a, then we grouped all transitions by their timestamp. We then obtained a state matrix $S_t \in \mathbb{R}^{n_s}$ where each value is the delay for the current station, and an action matrix $A_t \in \mathbb{R}^{n_a}$ where the values are the actions taken by the corresponding switch at time t. Since not all the switches act at each time step, many of the A_t values are set as `NaN` and only the observed actions contribute to the estimate for that time step.

From the resulting sequence $\{S_t\}_{t=0}^{T}$ and $\{A_t\}_{t=0}^{T-1}$ we form the transition pairs (Z_t, S_{t+1}) with $Z_t = (S_t, A_t)$ and compute the transition MI matrix $M \in \mathbb{R}^{(n_s+n_a)\times n_s}$ where each entry is

$$M_{ij} = I\left(Z_t^{(i)};\, S_{t+1}^{(j)}\right).$$

In this setting all variables are discrete so we don't make use of any estimators. To reduce the error caused by different variable cardinalities, we normalize using an upper bound of $\min\{H(Z_t^{(i)}), H(S_{t+1}^{(j)})\}$ approximated as $\log(|\mathcal{X}|)$ from the unique observed values per component. Finally, we apply our greedy clustering method to the normalized matrix.

The reason why we used only a small number of states with station delay is that we wanted to group highly coupled switches around a station. This is what happened since switches close to the same station will present high MI with the same delay values, so the clusters created were quite reasonable. We started setting our parameters α and *threshold* at 0.4 and 0.15 as in the previous scenario, but we noticed that in this case where we started with a huge number of agents we wanted to promote cluster merging, so by tweaking the *alpha* parameter to a value of $\alpha = 0.25$ we reached a point where, starting from 15 stations and 286 switch nodes, we had 14 agents controlling the system. The clusters seem reasonable as there is a big main cluster of highly interconnected switches that observe the whole system, and then smaller clusters with fewer switches that are around the same subset of stations but don't present high coupling with all of them.

In this case unfortunately a graph representation of the clustering is unfeasible since we should represent almost three hundred elements. This is a particular scenario that really tests our algorithm complexity since it present a really small number of states and a huge number of actions. Compared to the previous approach, it surely takes way more time since we go from seconds to a few minutes for the clustering process, but given the time benefits that it should bring with respect to a single agent approach, especially with this many actions to take,

in training it's still a feasible solution that should decrease the overall time needed for training.

7. Conclusions

The experiments on the toy Power Grid environment show that our mutual information based clustering can produce a factorization of the problem that, in the discrete version, translates into a clear scalability advantage for policy learning. In particular, from the clustering graphs we can see that actions are grouped with the states that they influence the most, and the agents share the states they control together, like power lines between different generators.
Considering the learning perspective, the comparison between single-agent and multi-agent approaches highlights a considerable advantage as the problem size grows: with only $N = 2$ generators the factorization is not beneficial and has scalability issues. With $N = 4$ both approaches converge, and even if the multi-agent version is slightly faster, it reaches a slightly worse result. With $N = 8$ the performance gap is clear, the convergence is much faster and more stable, achieving even better performances. All this data supports our theory that a multi-agent approach is more suitable for high dimensionality networks and shows that our clustering method allows to factorize network environments efficiently.
In the continuous version, the clustering procedure remains effective even when the KSG estimator is introduced. However, at the tested scale the single agent PPO performs really well and the multi-agent approach fails to decrease the training time. The only positive thing is that at $N = 8$ the performances become comparable, suggesting that with a more difficult problem or a larger scale we could still experience benefits even in a continuous scenario.
Finally, the Flatland experiments support the generality of our approach and that it can be applied not only to bigger problems, but also to multiple domains. Even when using a small state space based only on station delays, our clustering procedure was able to identify connected groups of switches: strongly coupled switches among small station groups were clustered together, while highly interconnected regions formed larger clusters that observed more stations. This setting also stresses a different scalability issue created by the huge action space. In this case the clustering computation is very expensive, but in the end it should still bring an advantage considering the time reduction in the training part.
Overall, these results show that MI-based clustering is a promising domain agnostic tool for factorization of network MDPs. When the original problem is sufficiently big, this factorization approach can significantly improve training efficiency while keeping competitive performance. These benefits depend on estimator cost, action/state dimensionality and the difficulty of the baseline problem.

8. Future Work

This work opens different directions for improvement, both on the clustering and reinforcement learning side. First of all the Flatland problem has to be studied deeper, measuring how convergence speeds up and how performance change when going from a centralized approach to our clustered multi-agent version. In particular, it would be important to quantify the convergence-time reduction in relation to the number of switches, to see its scalability, and to the number of clusters, to be able to tune the *alpha* parameter even better.
Another promising line could be adaptive clustering: clustering in our case is computed offline, but in a real world application dynamics can change over time. However, this would bring a lot of new issues in learning, changing clusters while the agents have already started learning on them.

9. Bibliography

References

[1] Deunsol Yoon, Sunghoon Hong, Byung-Jun Lee, and Kee-Eung Kim. Winning the L2RPN challenge: Power grid management via semi-markov afterstate actor-critic. In *International Conference on Learning Representations (ICLR)*, 2021. URL `https://openreview.net/forum?id=LmUJqB1Cz8`. Spotlight. OpenReview.

[2] Antoine Marot, Benjamin Donnot, Gabriel Dulac-Arnold, Adrian Kelly, Aidan O'Sullivan, Jan Viebahn, Mariette Awad, Isabelle Guyon, Patrick Panciatici, and Camilo Romero. Learning to run a power network challenge: a retrospective analysis. In Hugo Jair Escalante and Katja Hofmann, editors, *Proceedings of the NeurIPS 2020 Competition and Demonstration Track*, volume 133 of *Proceedings of Machine Learning*

Research, pages 112–132. PMLR, 2021. URL https://proceedings.mlr.press/v133/. Also available as arXiv:2103.03104.

[3] Jiaoyang Li, Zhe Chen, Yi Zheng, Shao-Hung Chan, Daniel Harabor, Peter J. Stuckey, Hang Ma, and Sven Koenig. Scalable rail planning and replanning: Winning the 2020 flatland challenge. In *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling (ICAPS)*, pages 477–485. AAAI Press, 2021. URL https://cdn.aaai.org/ojs/15994/15994-40-19487-1-2-20210517.pdf.

[4] Dingbang Liu, Fenghui Ren, Jun Yan, Guoxin Su, Wen Gu, and Shohei Kato. Scaling up multi-agent reinforcement learning: An extensive survey on scalability issues. *IEEE Access*, 12:94610–94631, 2024. doi: 10.1109/ACCESS.2024.3410318.

[5] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Stockholm, Sweden, 2018. URL https://www.ifaamas.org/Proceedings/aamas2018/pdfs/p2085.pdf. Extended abstract.

[6] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4295–4304. PMLR, 2018. URL https://proceedings.mlr.press/v80/rashid18a.html.

[7] Eduardo Cotilla-Sanchez, Paul D. H. Hines, Clayton Barrows, Seth Blumsack, and Mahendra Patel. Multi-attribute partitioning of power networks based on electrical distance. *IEEE Transactions on Power Systems*, 28(4):4979–4987, 2013. doi: 10.1109/TPWRS.2013.2263886.

[8] Francesco Corman, Andrea D'Ariano, Dario Pacciarelli, and Marco Pranzo. Dispatching and coordination in multi-area railway traffic management. *Computers & Operations Research*, 44:146–160, 2014. doi: 10.1016/j.cor.2013.11.011.

[9] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, 2004. doi: 10.1103/PhysRevE.69.066138. URL https://link.aps.org/doi/10.1103/PhysRevE.69.066138. Preprint: arXiv:cond-mat/0305641.

[10] Miao Zhang, Zhixin Miao, and Lingling Fan. Power grid partitioning: Static and dynamic approaches. In *2018 North American Power Symposium (NAPS)*, pages 1–6. IEEE, 2018. doi: 10.1109/NAPS.2018.8600609.

[11] A. J. Butte and I. S. Kohane. Mutual information relevance networks: Functional genomic clustering using pairwise entropy measurements. *Pacific Symposium on Biocomputing*, pages 418–429, 2000. doi: 10.1142/9789814447331_0040.

[12] A. A. Margolin, I. Nemenman, K. Basso, C. Wiggins, G. Stolovitzky, R. Dalla Favera, and A. Califano. Aracne: An algorithm for the reconstruction of gene regulatory networks in a mammalian cellular context. *BMC Bioinformatics*, 7(Suppl 1):S7, 2006. doi: 10.1186/1471-2105-7-S1-S7.

[13] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 2 edition, 2018.

[14] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008. doi: 10.1109/TSMCC.2007.913919.

[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

[16] Robin Uhrich, Marco Mussi, and Marcello Restelli. Switchfl: Switch-centered flatland environment for multi-agent reinforcement learning, 2025. URL https://github.com/RobinU434/SwitchFL.

Abstract in lingua italiana

La gestione di sistemi multi dimensionali basati su reti rappresenta una sfida chiave per l'applicazione di algoritmi di reinforcement learning al mondo reale. I metodi classici risultano solitamente efficaci su problemi di piccola scala, ma non si riescono ad applicare a scenari più ampi, come quelli considerati nel progetto AI4REALNET. In questo lavoro proponiamo un metodo di clustering dinamico basato sulla mutua informazione per fattorizzare il problema originale in sottoproblemi approssimativamente indipendenti. Successivamente abbiamo sviluppato un approccio multi agente di reinforcement learning basato sull'algoritmo Proximal Policy Optimization (PPO), in cui ciascun sottoproblema è gestito da un agente distinto. Abbiamo dimostrato l'efficacia di questo algoritmo in due diversi ambienti. Il primo è un contesto relativamente piccolo e semplice che coinvolge la gestione di linee elettriche e il bilanciamento del carico. Attraverso questo ambiente dimostriamo che il nostro approccio riduce significativamente il tempo di addestramento mantenendo prestazioni competitive rispetto ad un classico approccio centralizzato. Il secondo è una simulazione di una rete ferroviaria più ampia e complessa, che abbiamo utilizzato per mostrare come il nostro approccio possa essere applicato su domini diversi.

Parole chiave: Reinforcement Learning multi agente, fattorizzazione dinamica, mutua informazione