Autonomous Networking - Homework 2

A.Y. 2021-2022

Andrea Gasparini Edoardo Di Paolo Riccardo La Marca

1. Introduction

In this homework we addressed a drone routing problem using Reinforcement Learning techniques. We worked in a setting with *n* drones. All the drones collect information and they can transfer the packets to other drones. We have two homework's versions: in the first one each drone has a completely random path while in the second one they have a fixed path, that is different from the others. We had to implement two functions: *relay_selection* and *feedback*. The former is called at each timestep of the simulation, if needed, and here the drone decides whether to pass or not the packets; while the latter is called by each drone when a packet gets lost (in this case we have an *outcome* equals to *-1*, but it is barely common) or when a drone delivers packets to the depot (with *outcome* equalso to *1*).

2. Approach

In the general scenario a drone can perform three possible actions:

1. Transmit: the drone decides to transmit the packets to a *drone x* among its neighbours;
2. Not transmit: the drone decides to keep the packets;
3. Return to the depot: the drone decides to return to the depot.

How a drone decides to perform an action depends on the implemented algorithm (subsection 2.1).

2.1. Implemented algorithms

We implemented different algorithms but we always handled the *Q-table* as a dictionary with integers as keys and lists as values. The states are the cells' indexes in the environment and the lists have a length equals to $n + 1$ where the last element represents the action for the depot.

2.1.1. QLEARNING

In this algorithm the *relay_selection* function performs *exploration* or *exploitation*. Exploration is always chosen at the first iteration of the simulation and then again based on a probability ϵ. In the *exploration* case the drone selects an action according to this schema:

- **none**: if there are not neighbours or if the neighbours have less packets than itself;
- **a random drone** if it has more packets than the current drone, or if it is returning to the depot;
- **depot**: there are not neighbours returning to the depot and the current drone has ≥ 4 packets to deliver.

In the *exploitation* case, instead, the drone selects the best action according to this schema:

- **depot**: if the distance is ≤ 400 meters or if there are not neighbours and the drone has ≥ 2 packets in the buffer;
- **none**: if there are not neighbours;
- the **best drone** according to the max value in the *Q-table* and to the *buffer length* com.

These schemas concern the *relay_selection* function. About the *feedback*, we have seen that the *outcome* $= -1$ is barely common, so we just decided, in that case, to assign a reward equals to -2; otherwise we give a reward according to this function:

$$reward = \begin{cases} 2 \cdot \text{buffer\_len} + \frac{1}{energy_s} & \text{if } s_{dr} = d_{dr} \\ 2 \cdot \text{buffer\_len} & \text{otherwise} \end{cases} \quad (1)$$

where $energy_s$ is the energy spent to reach the depot and to return back to the mission, s_{dr} is the current drone and d_{dr} is the drone parameter in the *feedback* function.

At the end we update the Q-table as follow:

$$Q[c_{id}][d] \mathrel{+}= \alpha \cdot (r + \gamma \cdot (\max(Q[nc_{id}])) - Q[c_{id}][d]) \quad (2)$$

Autonomous Networking, Sapienza University of Rome, 1st semester A.Y. 2021-2022.

where c_{id} is the *cell index*, nc_{id} is the next cell index (the next state), d is the action selected, α is the learning rate and γ is the discount factor.

2.1.2. QLEARNING MGEO

This algorithm performs, as the subsubsection 2.1.1, *exploration* or *exploitation*. In the former case we select randomly a neighbour and if there are not neighbours the action will be *none*. In the *exploitation* we select a drone according to a drone score calculated as follow:

$$d_s = \begin{cases} -\frac{d_{dd}}{d_{sp}} \cdot d_{bf} \cdot Q[c_{id}][d_{id}] & \text{if not } d_{mr} \\ \frac{d_{dd}}{d_{sp}} \cdot d_{bf} \cdot Q[c_{id}][d_{id}] & \text{otherwise} \end{cases} \quad (3)$$

where d_{dd} is the distance of the current neighbour from the depot, d_{sp} is the drone's speed, d_{bf} is the # of packets that this drone has in its buffer. We just maximized this value in order to pick the best neighbour. Another action is to return to the depot, and we return only if we have not neighbours and our buffer has some packets.

2.1.3. DOUBLE QLEARNING

In this algorithm we have two Q-tables that we update randomly according to a policy that could be *exploration* or *exploitation*. In the *feedback* function we have a reward equals to -2 if the *outcome* is -1, otherwise it is $2 \cdot d_s$ where d_s is the drone's speed.

2.1.4. AI ROUTING

In this algorithm we have a reward that is equal to -2 if the *outcome* is -1, otherwise it is $2 \cdot \frac{e_d}{d}$ where e_d is the *event_duration* variable and d is the delay. Also in this case, the algorithm performs *exploration* and *exploitation*. In the *exploration* case we select a random drone with a certain probability and only if we have exactly 1 packet in the buffer while, if we have more than 2 packets, we select a random drone that is returning to the depot (if there is) otherwise we return. In the *exploitation* case we return to the depot if we have not neighbours and we have more than 2 packets, otherwise we select a best drone based on the distance from the depot, the drone's speed, the buffer length and the value in the Q-table for that drone.

2.1.5. OPTIMISTIC AI ROUTING

This algorithm is similar to the subsubsection 2.1.4 but with the Q-table initialized with optimistic values.

3. Experiments and results

In the plots shown in Appendix C we can observe the behaviour of our routing algorithm compared to other on some metrics. The results shown for our algorithm have been obtained after an hyperparameter tuning phase on α, γ, and ϵ, which respectively are the learning rate, the discount factor and the probability of performing exploration.

The tested values for the hyperparameters are in Table 1, along with the final values we used highlighted in bold. Figure 13 shows the different scores obtained during the hyperparameter tuning.

We also tested our algorithms setting the configuration variable `SWEEP_PATH` both to `True` and `False`, in order to change the drones' path from fixed to random. Appendix C includes plots for both the configurations.

We are delivering the subsubsection 2.1.1 algorithm since it has a good *packet delivery ratio* (Figure 9), a very good *routing time over the mission time* (Figure 5) and a very good *energy move routing* (Figure 1) with an high number of events to the depot (Figure 7). However, it has an *event mean delivery time* slightly higher than the other algorithms due to its policy.

4. Contributions

The algorithms' implementations have been developed jointly with meetings on a VoIP software. However, each of us contributed in the following way:

1. **Andrea Gasparini**: hyperparameter tuning, AI Routing, Optimistic AI;
2. **Edoardo Di Paolo**: AI Routing, QLearning, Double QLearning;
3. **Riccardo La Marca**: AI Routing, QLearning, Double QLearning.

Appendices

A Tables

Hyperparam.	Tested values
α	$\{0.4, \mathbf{0.6}, 0.8, 1.0\}$
γ	$\{0.4, 0.6, \mathbf{0.8}, 1.0\}$
ϵ	$\{0.05, 0.10, 0.15, \mathbf{0.20}\}$

Table 1. The tested hyperparameters (the best ones are highlighted in bold)

B Other approaches' source code

In the following the source code of other approaces:

- **AI Routing**;
- **Double QLearning**;
- **Optimistic AI Routing**;
- **QLearning MGEO**.

C Figures

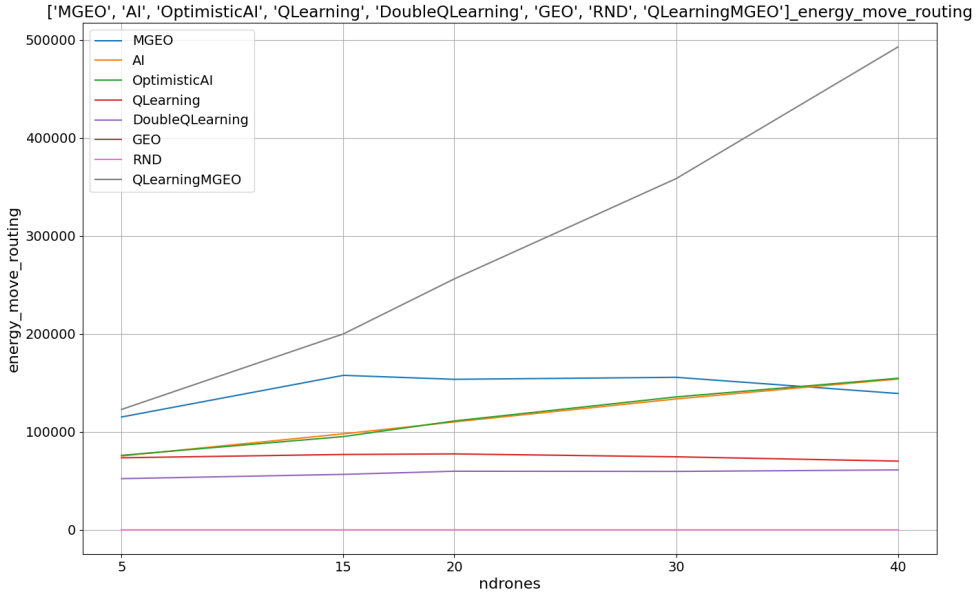

Figure 1. Energy move routing with `SWEEP_PATH False`

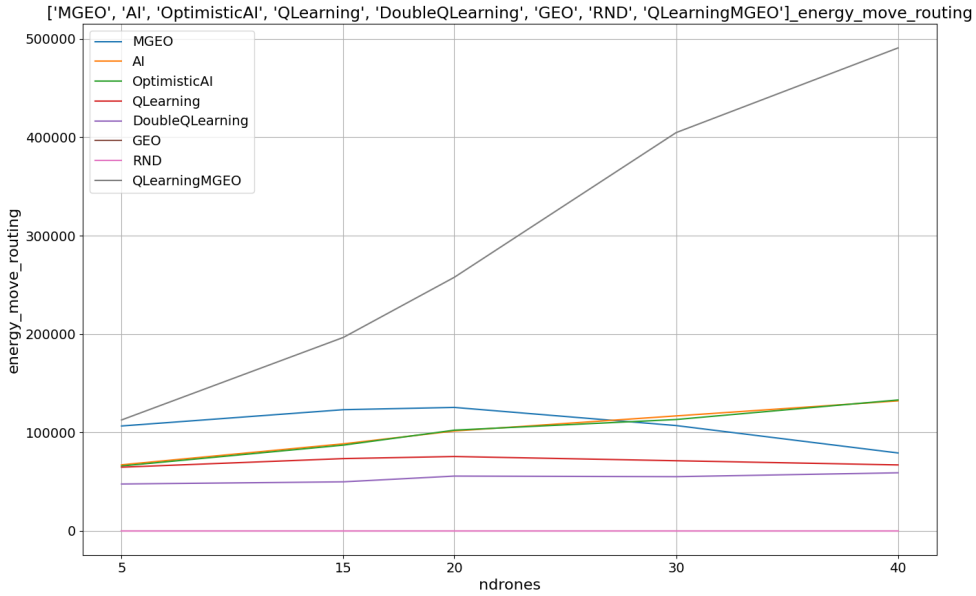

Figure 2. Energy move routing with `SWEEP_PATH True`

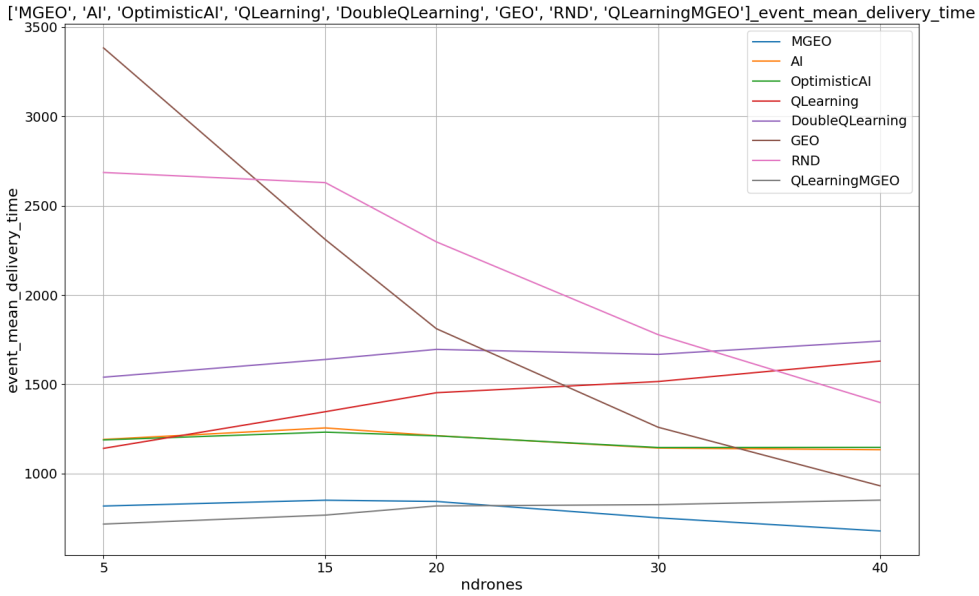

Figure 3. Event mean delivery time with `SWEEP_PATH False`

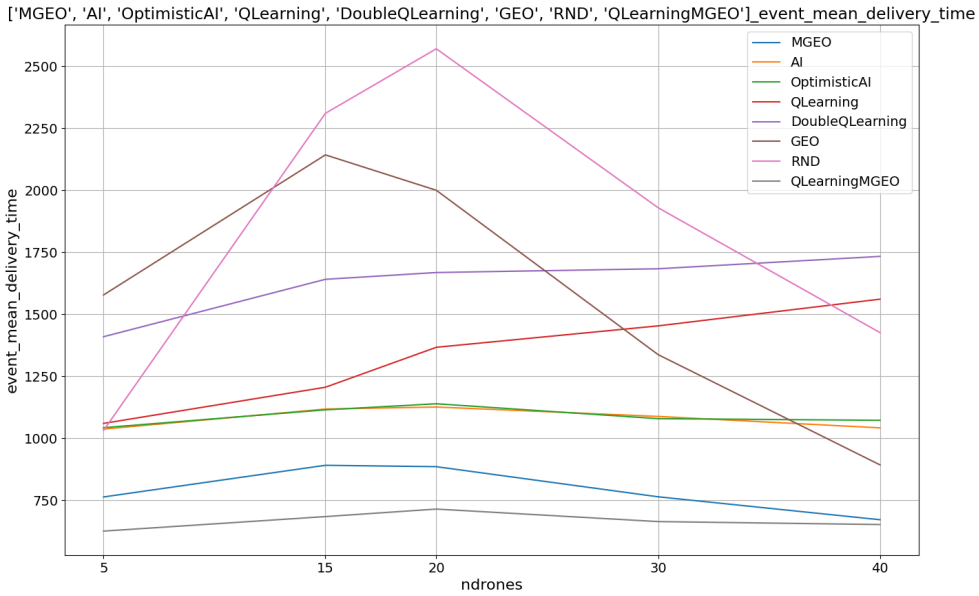

Figure 4. Event mean delivery time with `SWEEP_PATH True`

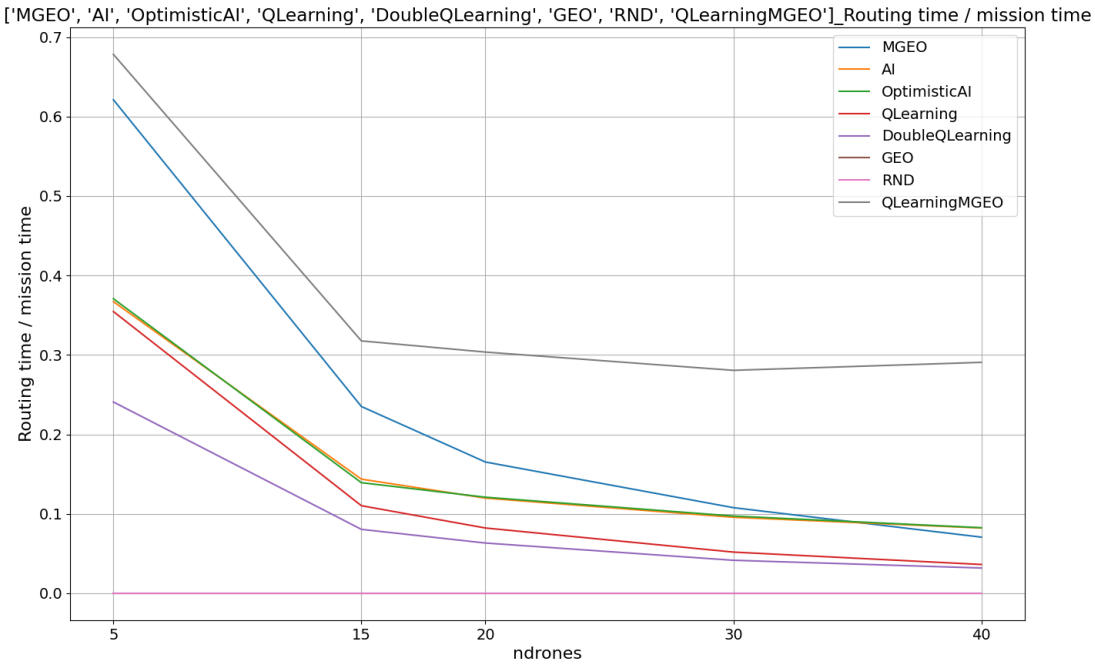

Figure 5. Routing time over mission time with `SWEEP_PATH False`

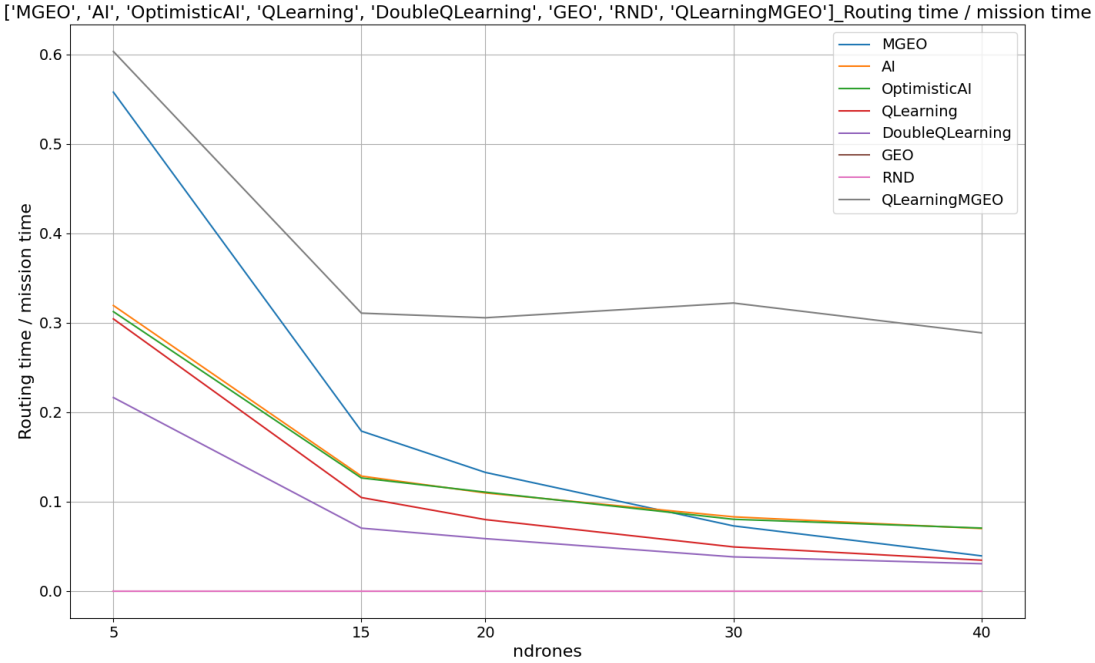

Figure 6. Routing time over mission time with `SWEEP_PATH True`

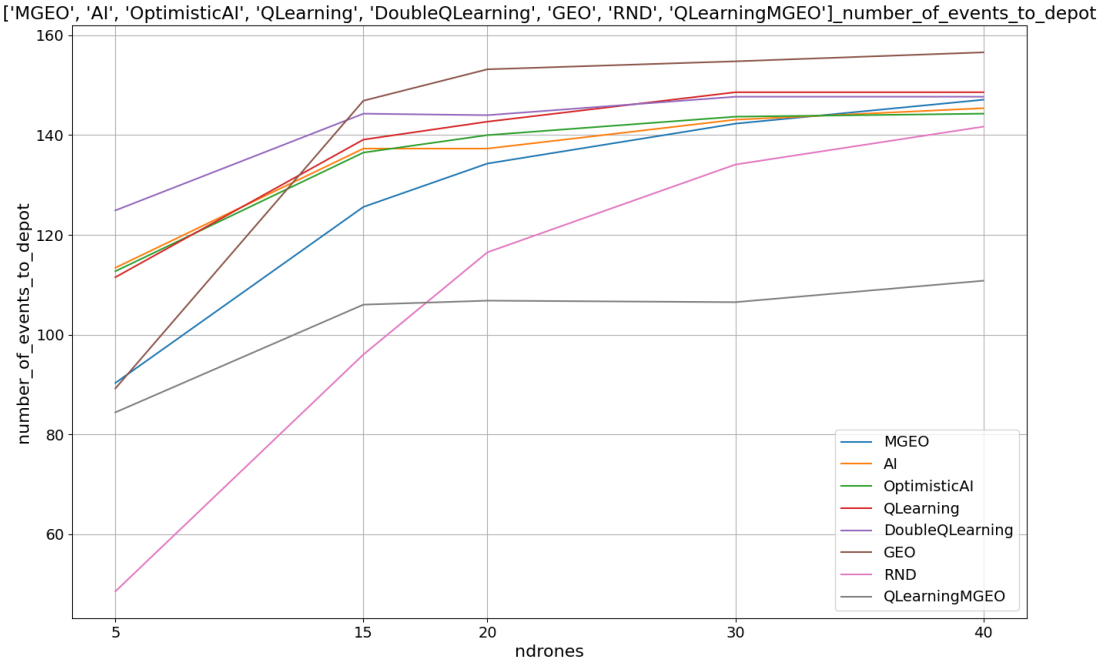

Figure 7. Numbers of events to the depot with `SWEEP_PATH False`

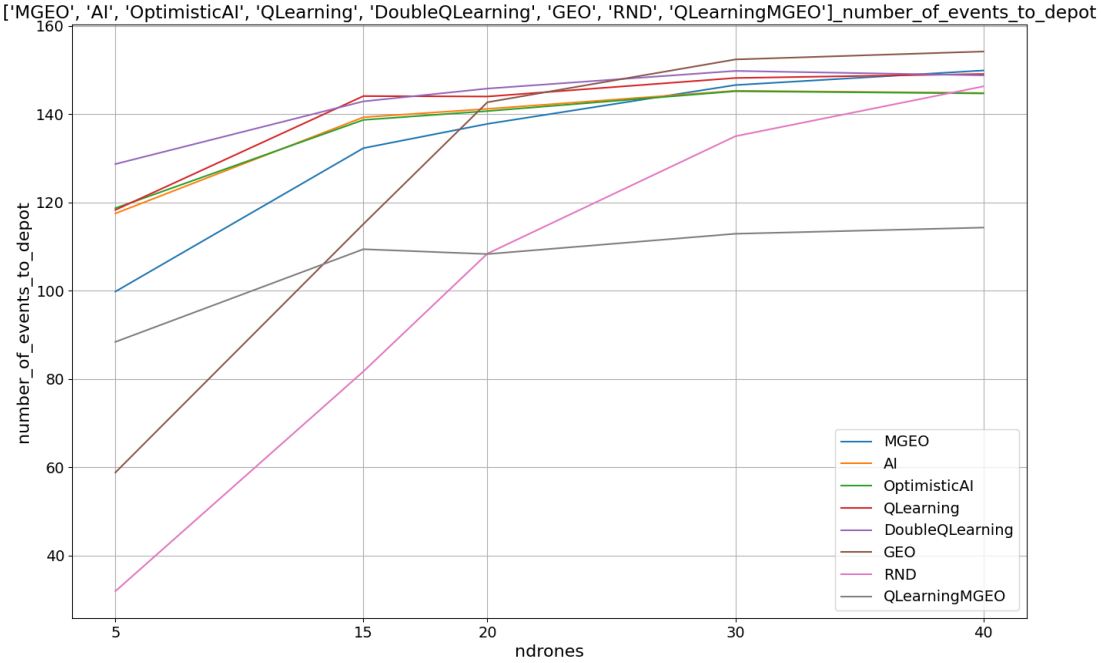

Figure 8. Numbers of events to the depot with `SWEEP_PATH True`

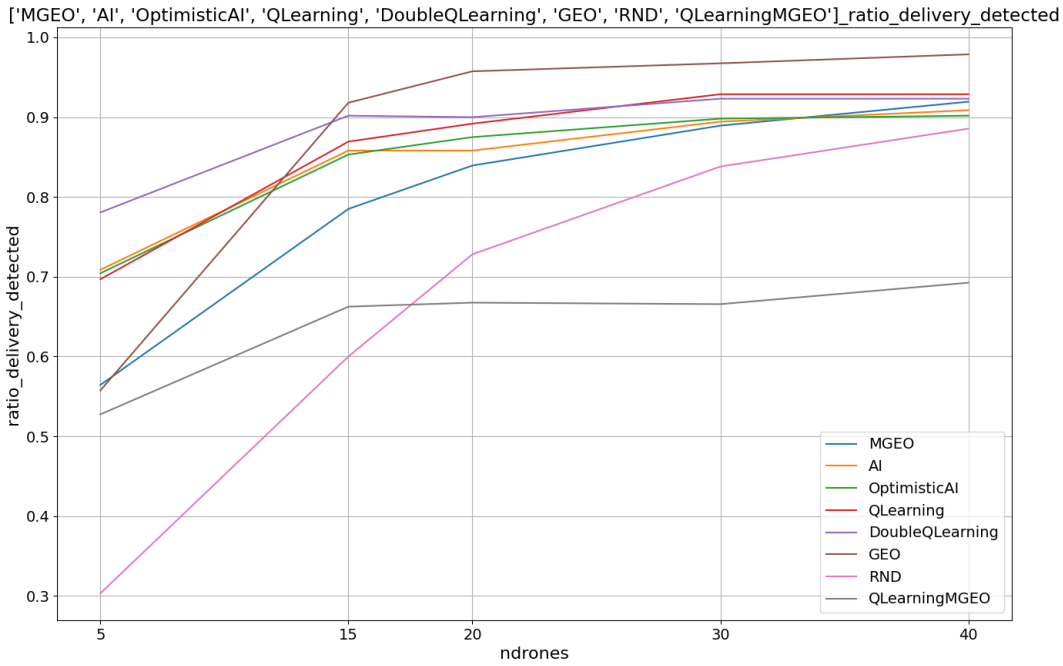

Figure 9. Packet delivery ratio with `SWEEP_PATH False`

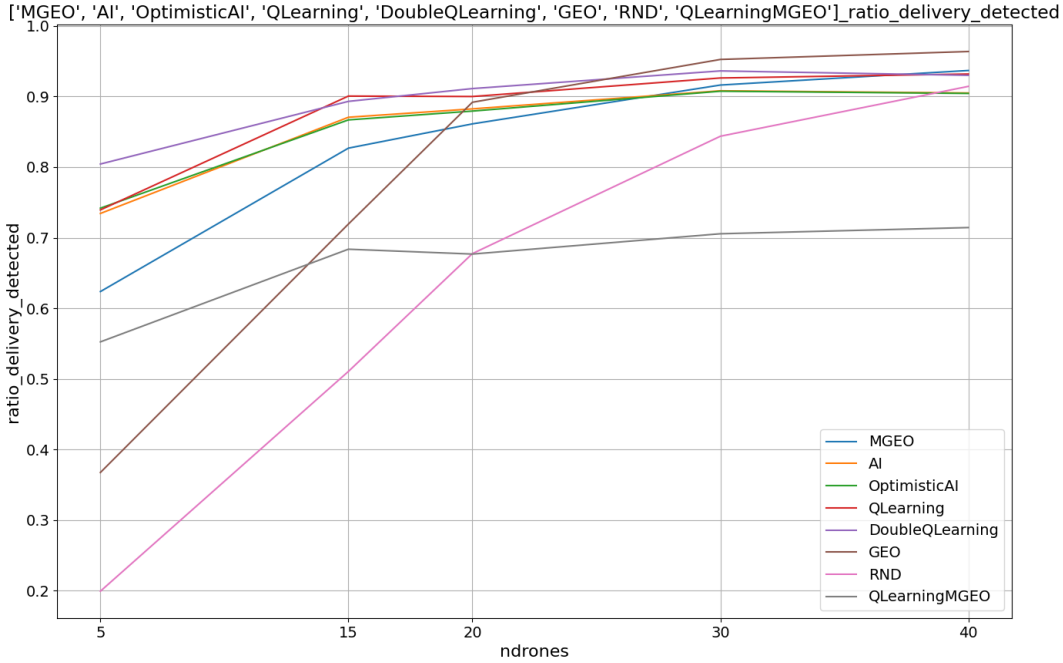

Figure 10. Packet delivery ratio with `SWEEP_PATH True`

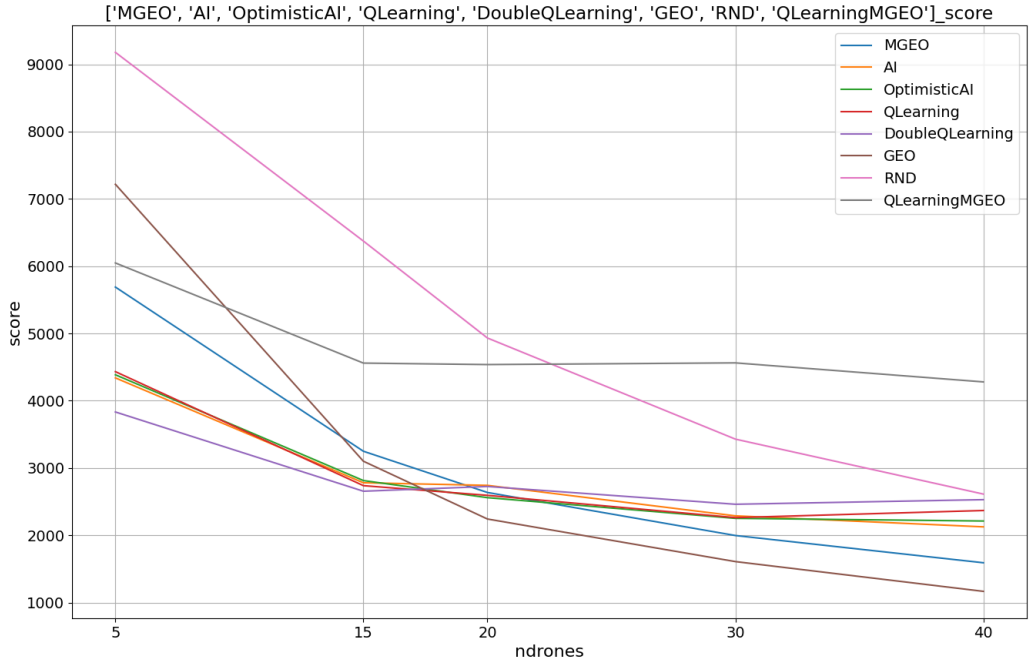

Figure 11. Score with `SWEEP_PATH False`

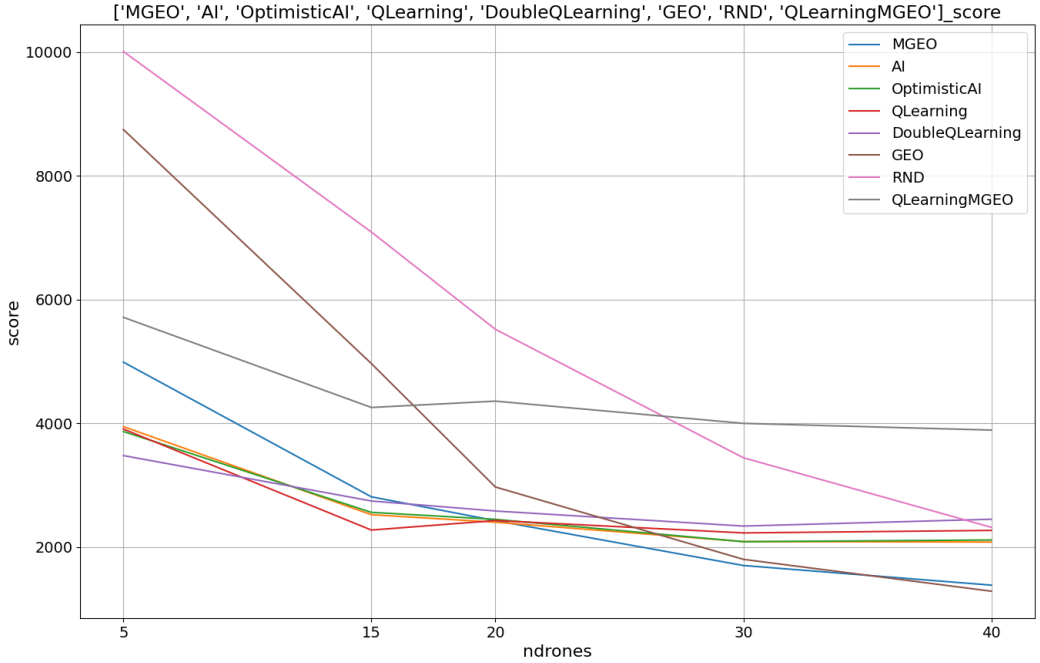

Figure 12. Score with `SWEEP_PATH True`

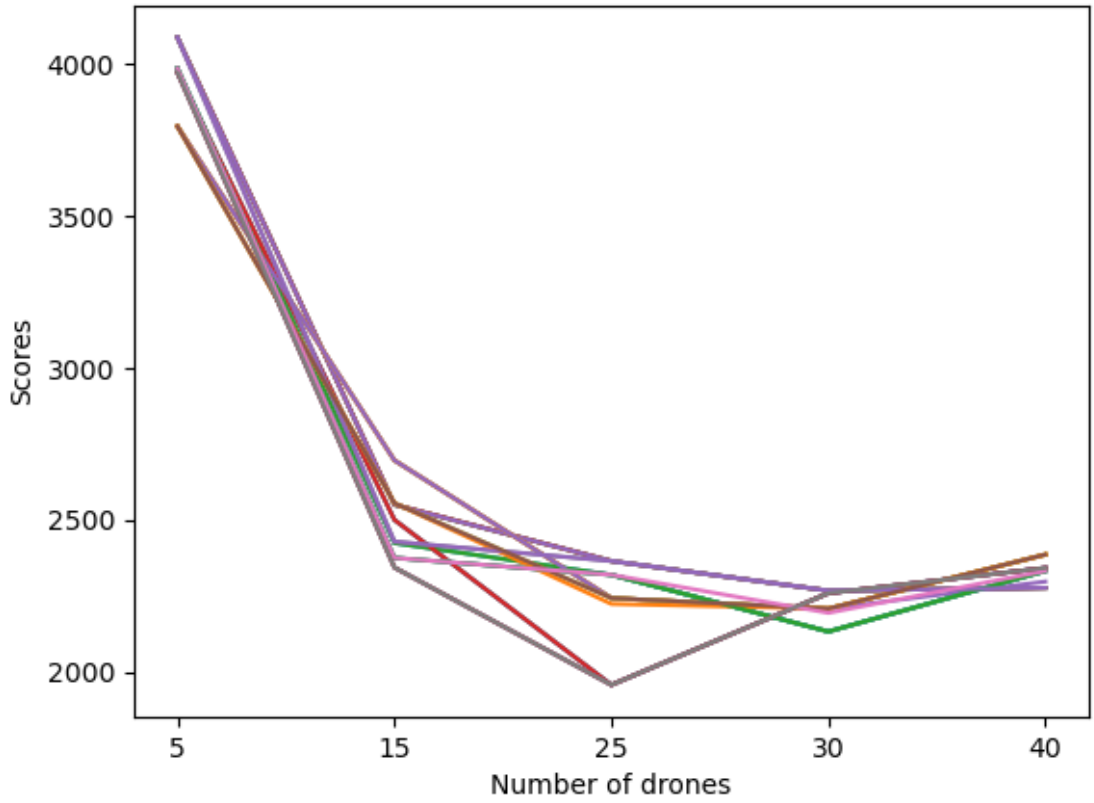

Figure 13. Hyperparameters tuning, **legend**