Autonomous Networking - Homework 3

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 the packets or not to; 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 depots (with *outcome* equalso to *1*). In our scenario we have 2 depots one at the bottom and one at the top.

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 with different strategies. In general, we used a *Q-table* (except for the *Gradient Bandit* approach) where the states are the cells' indexes and the actions are the drones plus the two depots. We tried to give the rewards immediately (i.e. in the *relay_selection*) and with a "delay" (i.e. in the *feedback*).

2.1.1. AI Routing

In this algorithm the *relay_selection* function performs either *exploration* or *exploitation*. Exploration is always chosen at the first iteration and then again based on an a probability ϵ. In the *exploration* case if the drone has at least one neighbour it randomly selects one of them, otherwise it selects itself.
The *exploitation* case has two possible *sub-cases*:

1. no neighbours and at least one packet in the buffer;
2. at least one neighbour.

In the first case, the drone will take in account the *mean* expressed in Equation 1 (where c_s is the current timestamp and n_s is the packet's creation time) and, based on this value, it will perform an action.

$$\text{avg\_pkts} = \frac{1}{N} \cdot \sum_{n=1}^{N} (c_s - n_s) \tag{1}$$

If the mean is greater than a value w (`AVG_PCKT_THRESHOLD`) then the drone checks whether it can deliver or not the packets to the depot. In the second case, in which the mean is less than w, the drone will consider the time t to reach the nearest depot from its *next target* and the time o to reach the next target from its current position; if the sum $t + o$ is greater than a value `TOTAL_TIME_AVG_PCKT_THRESHOLD`, then the drone will do the same as in the case that the mean is $> w$, otherwise it waits. The function that a drone evaluates to understand if it can return to a depot or not, namely `calculate_best_depot`, is defined as follows: it takes the distance in Equation 2

$$\min\{\Delta_{fd_n}, \Delta_{sd_n}, \Delta_{fd_c}, \Delta_{sd_c}\} \tag{2}$$

where Δ_{fd_n} is the distance from the next target to the first depot, Δ_{sd_n} the distance from the next target to the $2nd$ depot and similarly for the last two distances but for the

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

current position. If the min distance is one of the first two, then the drone waits, otherwise it checks if this minimum value is < 550 and > 200, it returns to the best depot immediately.

If the drone has some neighbours, it is then going to consider the *Q-table*. The drone takes the neighbours' values from the *Q-table*, and if the $\max$ is > 0, it selects the drone with that value. If the $\max = 0$, then the best drone is selected based on a *drone score*, defined in Equation 3.

$$d_s = \frac{(d_{sp} + \ln d_{bl})}{\Delta_{dt}} \tag{3}$$

In Equation 3 the d_{sp} is the *drone's speed*, the d_{bl} is the *drone's buffer length* and the Δ_{dt} is the distance from the best depot; for this Δ_{dt} the current drone calculates the distance from the next target if the neighbour is not going to the depot, otherwise it estimates the distance from the current position. If this d_s is $\geq$ than the actual drone score, then the drone selects this neighbour as the best one.

The reward is defined as follows (Equation 4):

$$\begin{cases} -2 & \text{if outcome} = -1 \\ \frac{2}{\text{delay}} & \text{otherwise} \end{cases} \tag{4}$$

2.1.2. Gradient Bandit

In the `Gradient Bandit` algorithm we use a different approach. It does not involve a *Q-table*, but a list H such that, for each action a (that can be either pass or keep or returns to the nearest depot) the value $H_t(a)$ represent a numerical preference for that specific action in the step *t*. Initially the value of the list is $H_1(a) = 0 \; \forall a$, while $\forall t > 1$ let A_t be the action chosen (Equation 5)

$$\begin{cases} H_{t+1}(A_t) = H_t(A_t) + \alpha_t(R_t - \bar{R}_t)(1 - \pi_t(A_t)) \\ H_{t+1}(a) = H_t(a) - \alpha_t(R_t - \bar{R}_t)\pi_t(a), \quad \forall a \neq A_t \end{cases} \tag{5}$$

where R_t is the reward given at the step *t* for the action A_t and $\bar{R}_t$ is the average of all the given rewards up to step *t*: $\bar{R}_t = \bar{R}_{t-1} + \frac{1}{N}(R_t - \bar{R}_{t-1})$. Finally, α is a exponential learning rate, *i.e.* $\alpha_n = \alpha_0 e^{-dn}$ where d is the decay and n is the step of the simulation (given by `simulator.cur_step`).

While in the other algorithms the used policy is the ϵ-greedy, here we decide to use a different one: $\pi_t(a)$ is the probability of choosing the action a computed according to the Softmax function and the preference's list. The formula is defined in Equation 6:

$$\pi_t(a) = \Pr[A_t = a] = \frac{e^{H_t(a) - \max_a H_t(a)}}{\sum_{b \in A} e^{H_t(b) - \max_b H_t(b)}} \tag{6}$$

So, using this policy each drone will choose one of its neighbours based on a probability given by π_t. However, in order to do not choose a "bad" (*i.e.*, with a small preferences) neighbour, we compute a *drone score* (see Equation 3) taking all the near drones with a value greater than mine. At the end, we have a subset of drones on which we apply the Softmax policy. Note that, for actions -1 and -2 we don't compute the drone score, but we use the procedure `calculate_best_depot`. If it returns the action "keep the packets", then no any other actions is added to the list of best actions, otherwise we add -1 or -2 according to the procedure's output.

At the end, we give a reward based on Equation 7

$$R_t = \begin{cases} -2, & \text{outcome} = -1 \\ \frac{d_{ds} \cdot d_{bl}}{\Delta_{dt} \cdot \text{avg\_pkts}}, & A_t = d \notin \{-1, -2\} \\ \frac{d_{ds} \cdot d_{bl}}{\Delta_{dt} \cdot \ln \gamma_t}, & \text{otherwise} \end{cases} \tag{7}$$

where γ_t is the energy spent, up to step *t*, of the drone that has delivered to the depot.

2.1.3. HW2 Routing

This algorithm is simply an adjustment of the $2nd$ homework with minor changes.

3. Experiments and results

In the plots showed in Appendix C we can observe the behaviour of our routing algorithms compared between each other on some metrics.

We are delivering the algorithm defined in subsubsection 2.1.1 since it has **the best** packet delivery ratio and, at the same time, it uses **less energy** than the others (except for the `HW2ROUTING`, which, however, has a very high delivery time) with a **good mean delivery time**.

The results showed for our algorithm have been obtained after an hyperparameter tuning phase on ϵ, `AVG_PCKT_THRESHOLD` and `TOTAL_TIME_AVG_PCKT_THRESHOLD` (with the values in Table 1) on a total of **3360** (Figure 13) experiments with $2, 5, 10, 20, 30, 40$ drones. We also tested our algorithms setting `SWEEP_PATH` both to `True` and `False`; in the Figure 11 we can see that our AI Routing has a score better than other all the algorithms when the paths are completely random. In Figure 14 are reported only the results for $\epsilon = 0.01$ and $\epsilon = 0.02$ that are the best.

The final values used for the hyperparameters of our algorithm depends on the number of drones and are defined in Table 2. In the Figure 15 there is the reward's convergence after 1 million steps in the simulation with 20 drones.

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**: AI Routing, HW2 Routing and hyperparameters tuning;
2. **Edoardo Di Paolo**: AI Routing, HW2 Routing, Gradient Bandit and final experiments;
3. **Riccardo La Marca**: AI Routing, Gradient Bandit and hyperparameters tuning.

Appendices

A Other approaches' source code

In the following the source code of other approaches:

1. **Gradient Bandit**;
2. **HW2 AI Routing**.

B Tables

Hyperparameter	**Tested values**
ϵ	{0.01, 0.02, 0.05, 0.1, 0.32, 0.62, 0.8}
`AVG_PCKT_THRESHOLD`	{400, 500, 600, 800}
`TOTAL_TIME_AVG_PCKT_THRESHOLD`	{600, 800, 1000, 1200}

Table 1. Tested values for the tuned hyperparameters

N. Drones	ϵ	**AVG_PCKT_THRESHOLD**	**TOTAL_TIME_AVG_PCKT_THRESHOLD**	**Score**
2	0.01	800	1000	2395.618
5	0.01	400	1000	2258.924
10	0.01	400	800	1889.479
20	0.01	500	600	1481.37
30	0.02	500	800	1070.698
40	0.02	600	1000	1012.86

Table 2. Best values for number of drones

C Figures

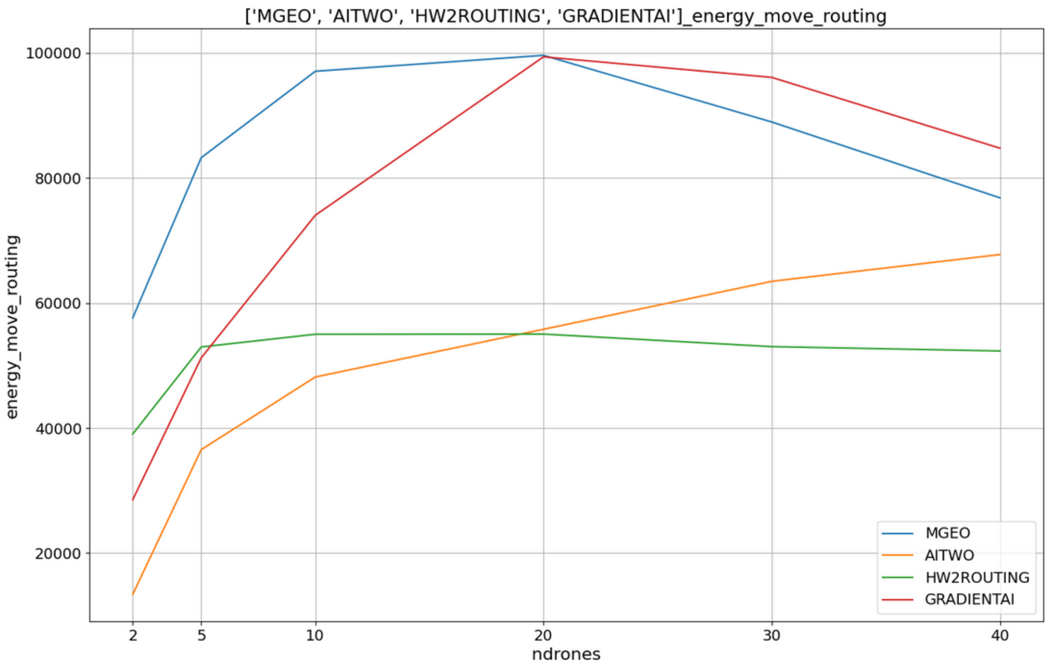

Figure 1. Energy move routing with `SWEEP_PATH False`

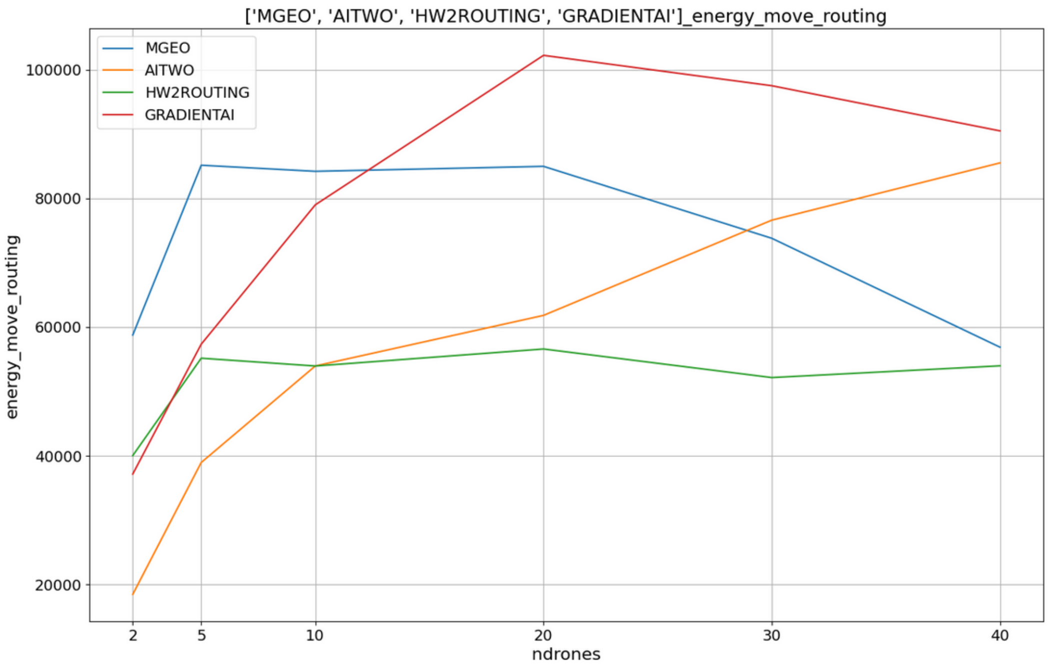

Figure 2. Energy move routing with `SWEEP_PATH True`

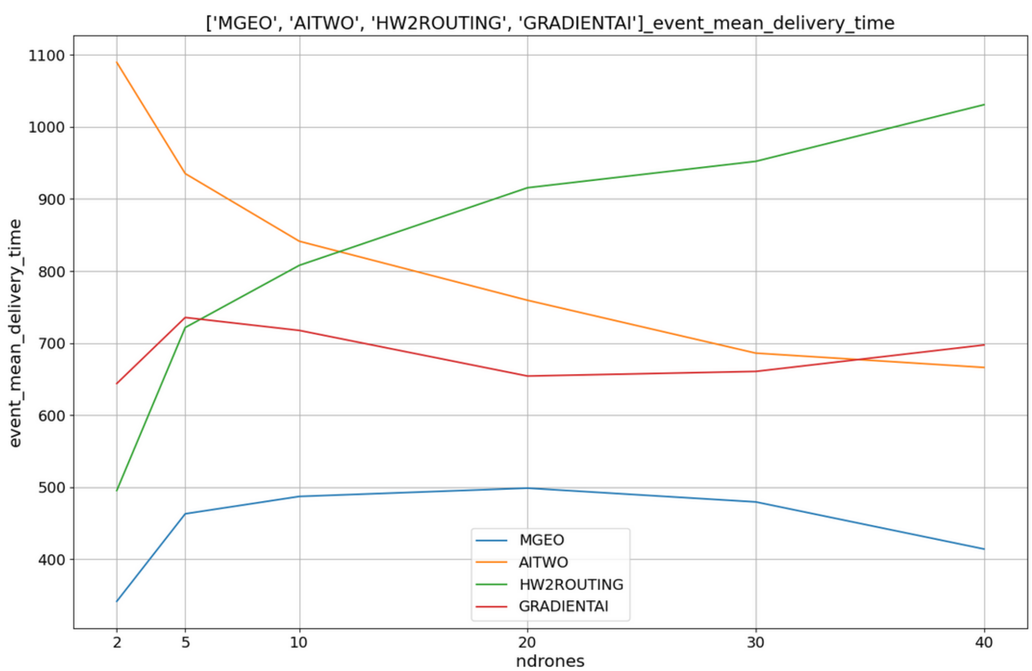

Figure 3. Event mean delivery time with `SWEEP_PATH False`

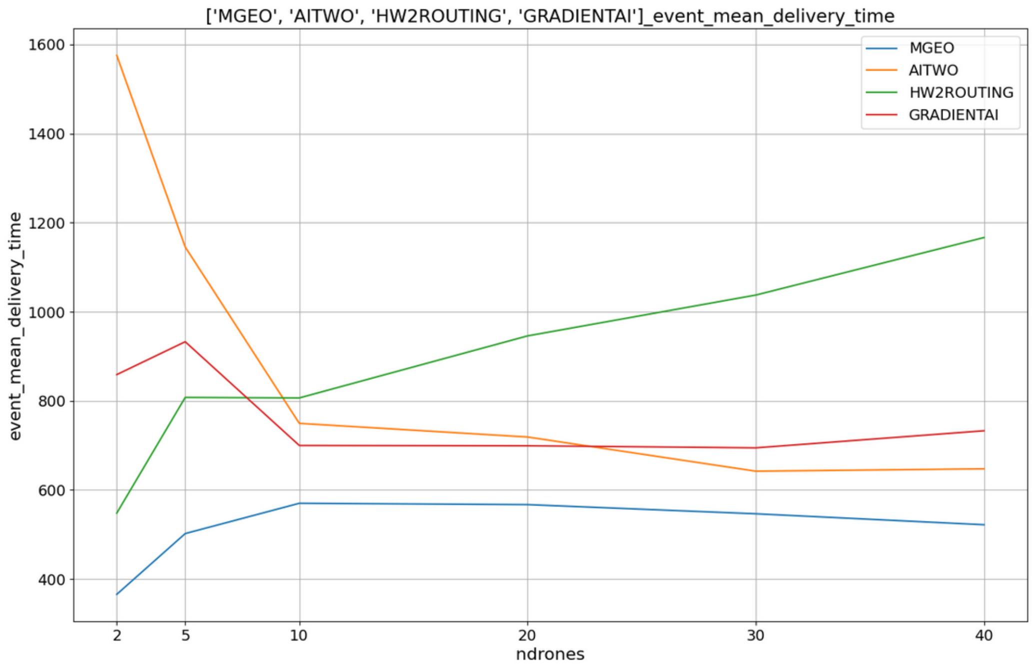

Figure 4. Event mean delivery time with `SWEEP_PATH True`

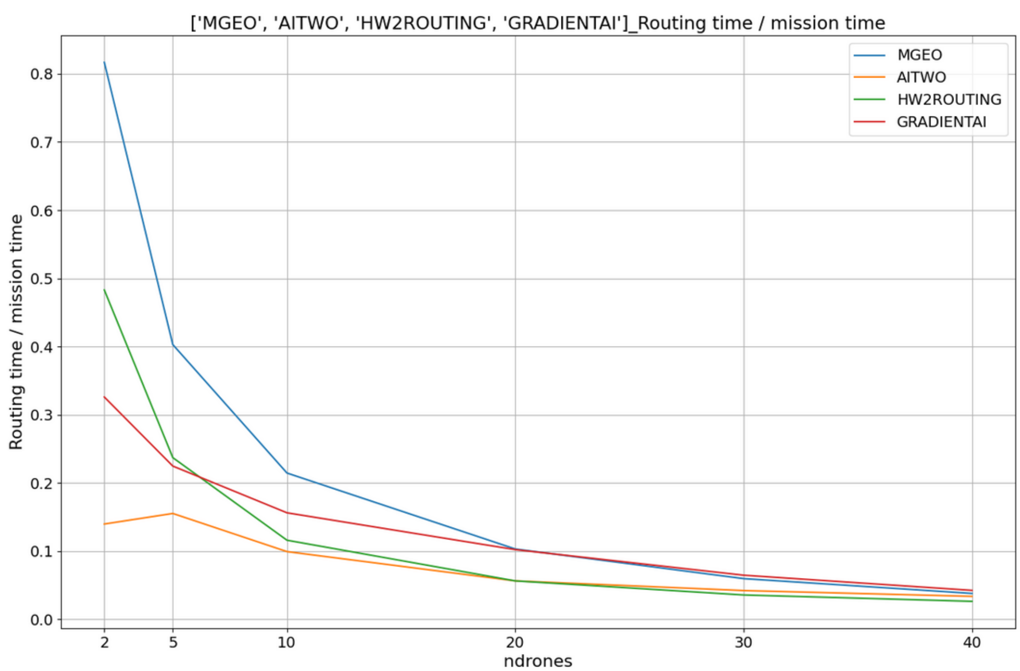

Figure 5. Routing time over mission time with `SWEEP_PATH False`

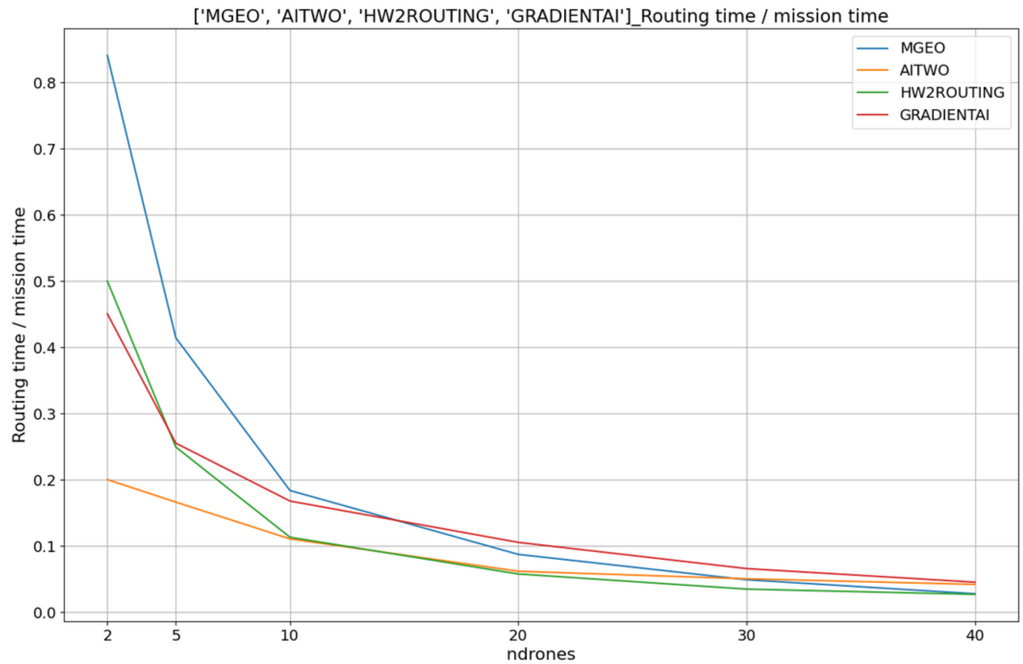

Figure 6. Routing time over mission time with `SWEEP_PATH True`

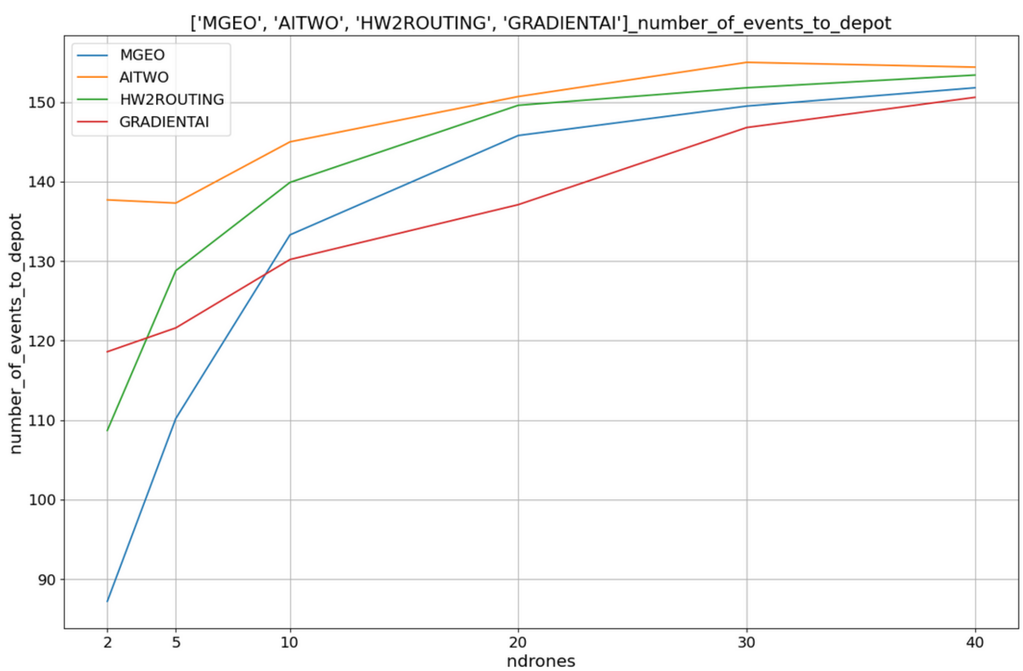

Figure 7. Numbers of events to the depot with `SWEEP_PATH False`

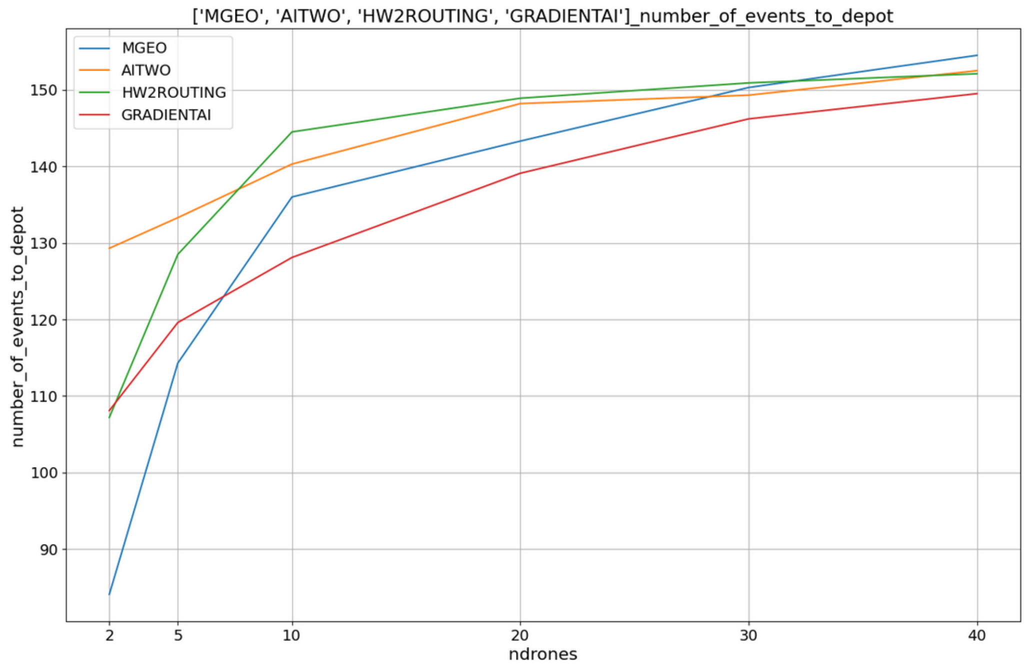

Figure 8. Numbers of events to the depot with `SWEEP_PATH True`

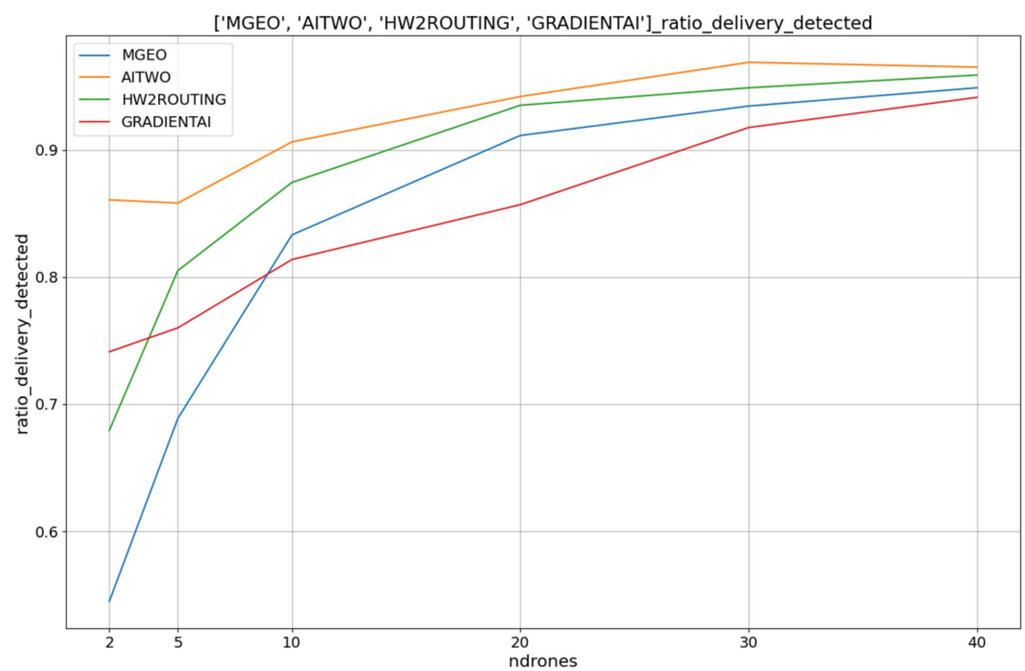

Figure 9. Packet delivery ratio with `SWEEP_PATH False`

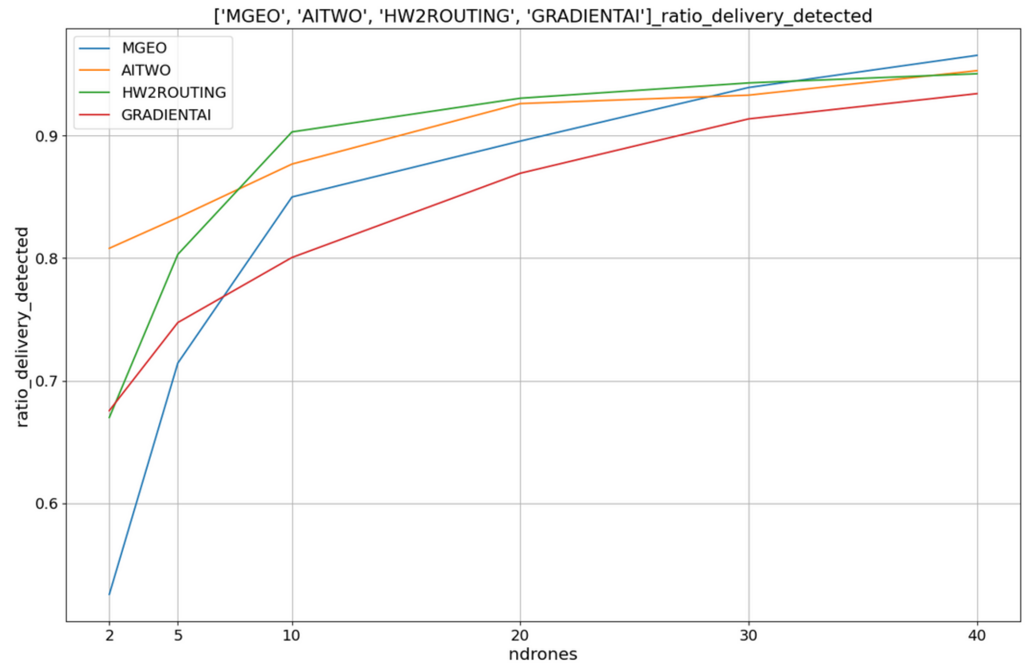

Figure 10. Packet delivery ratio with `SWEEP_PATH True`

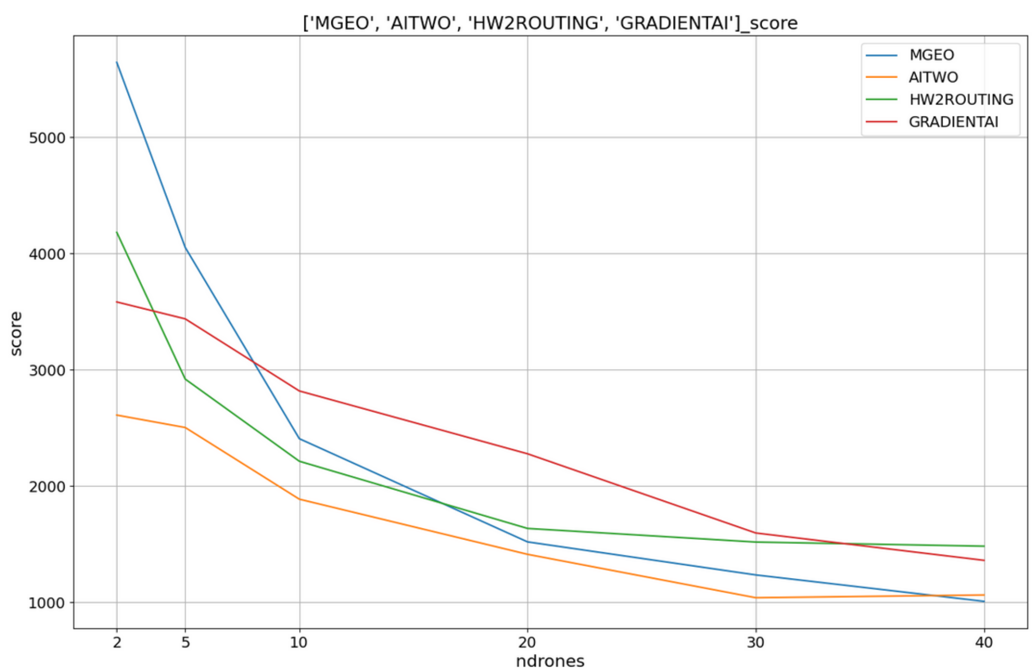

Figure 11. Score with `SWEEP_PATH False`

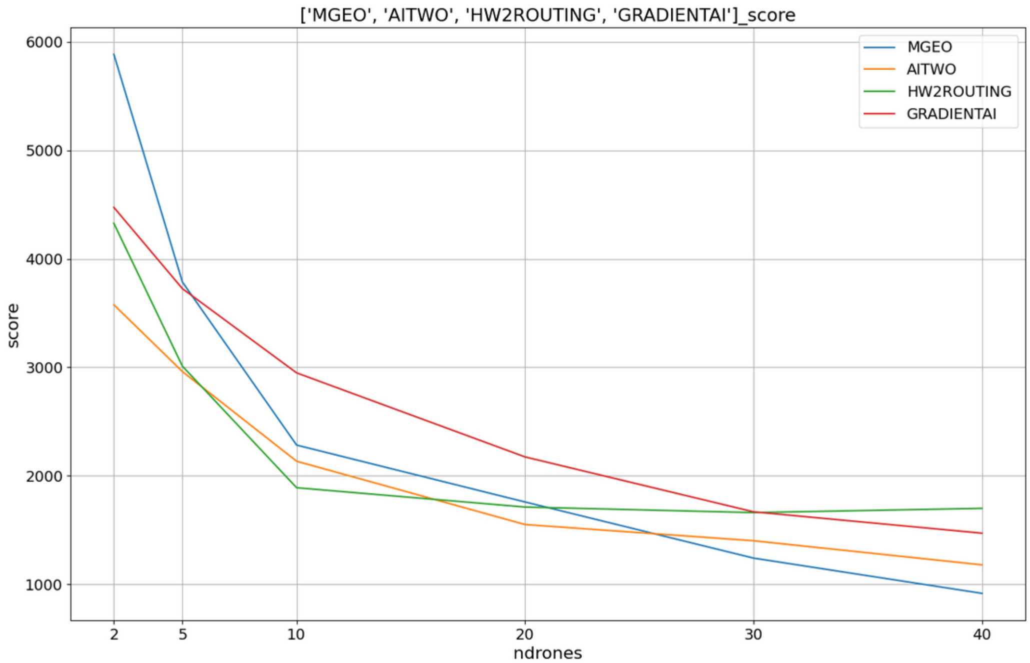

Figure 12. Score with `SWEEP_PATH True`

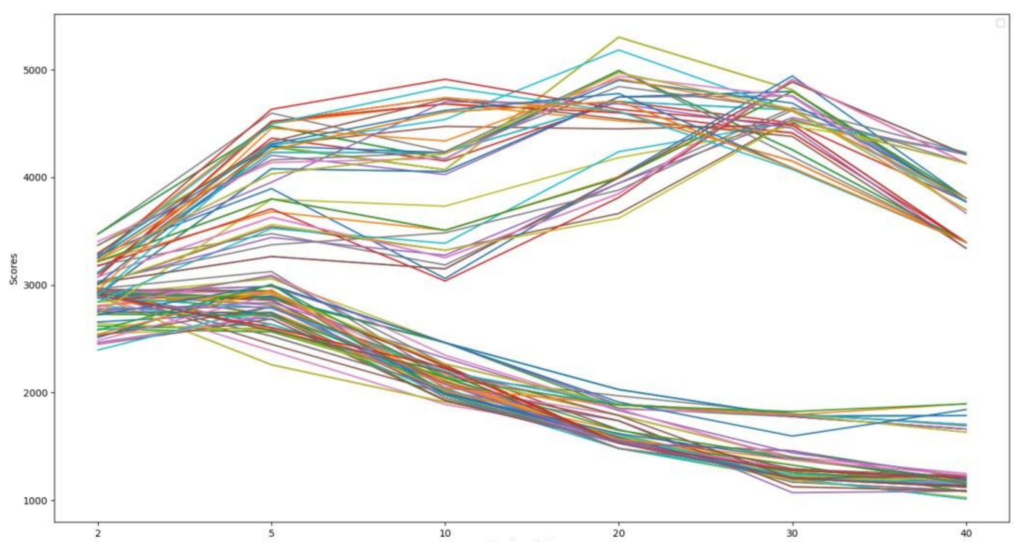

Figure 13. Complete hyperparameters tuning

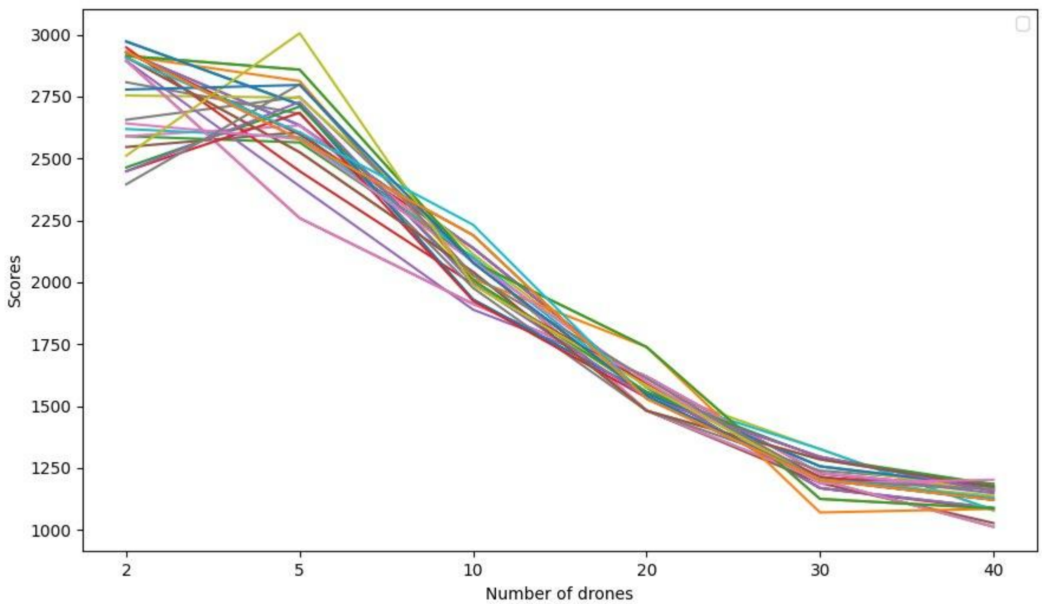

Figure 14. Hyperparameters tuning for $\epsilon = 0.01$ and $\epsilon = 0.02$

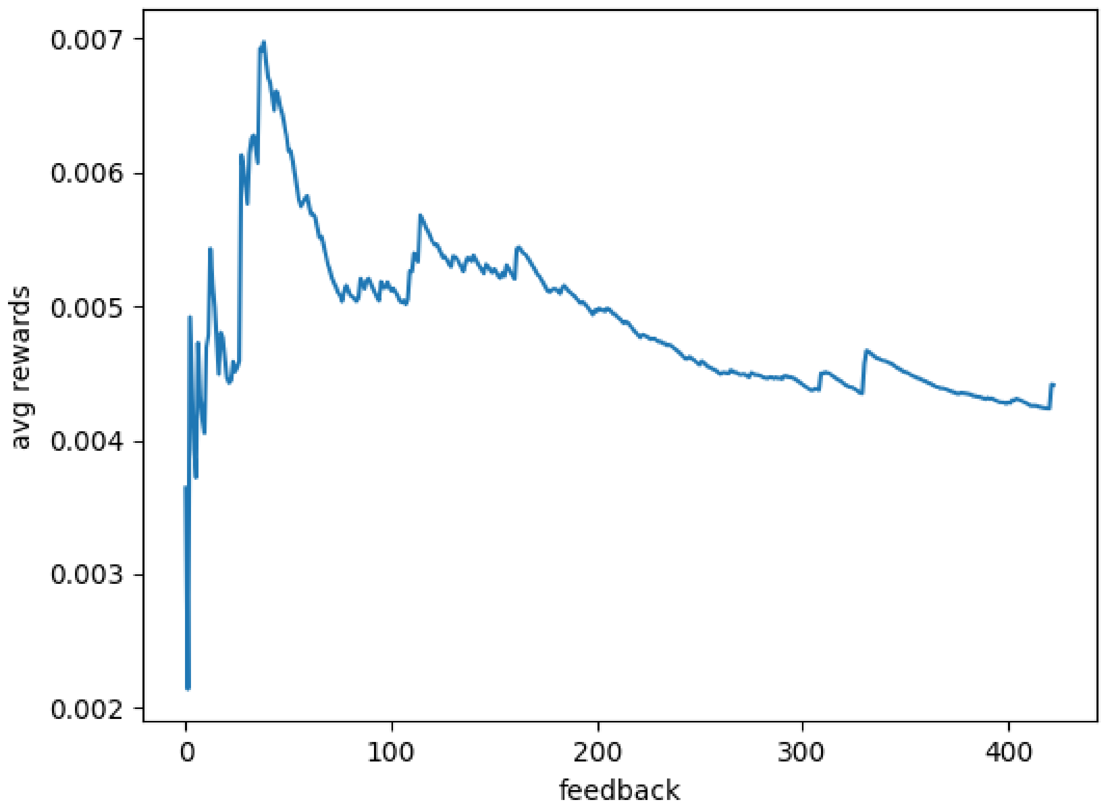

Figure 15. Average rewards over 1 million steps