Computational Game Theory / Learning Dynamics Multi-Agent Learning Assignment 3

Deadline: August 17, 2025, 23:59

Submission Instructions

- We have organized a Github classroom for this assignment. Use this link `https://classroom.github.com/a/tKMU7Lnv` to accept the assignment and to create inside the classroom your own private repository. Commit there your code and report (in PDF form), before the deadline.
- Follow this naming convention for your PDF report: *Lastname_Firstname_studentNo_affiliation.pdf.*
- Late submissions will be penalized (-1 point/day and 0 after 4 days).
- This is an **individual** assignment, sharing code is considered plagiarism and will be punished. Any other online sources used should be declared in the report. Plagiarism will not be tolerated.
- Your code for the environments of the assignment should strictly follow the PettingZoo API, which is the multi-agent extension of the industry standard Gymnasium API for single-agent reinforcement learning. Use the documentation of PettingZoo to create your environment. Numerous examples are available in the documentations to help you.
- Very important, make sure your code is legible, clean, and well-commented. The same goes of course for your report.
- If you have any questions send an email to: `raphael.avalos@ai.vub.ac.be` with the subject `[CGT2024][Assignment 3]`.

1 Learning with sparse interactions

For this assignment, you are going to implement the sparse interaction learning agents from the paper *'Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems'*, by Melo & Veloso (AAMAS, 2009). In many applications the interactions between the different agents coexisting in the same environment are not very frequent. This makes it reasonable to create agents that learn the situations in which they need to coordinate, and in which they don't have to. The approach seeks to exploit local interactions, allowing the agents to act independently whenever possible, and takes advantage of the assumed sparse interactions in domains where the multiple agents are loosely coupled. Concretely, each agent must learn from experience which situations require coordination. The setting is one of partial observability for the individual agents. The authors introduce an extension of Q-learning, in which they augment the action space of each agent with a coordination (pseudo-)action that attempts to use information from the other agents to decide the correct action. The authors consider for each agent k that the reward r_k is decomposable as

$$r_k(x, a) = r_k^I(x_k, a_k) + r^C(x, a),$$

where r_k^I is the individual component of the reward function, that depends only on the local state and action of agent k, and r^C is a common component of the reward function, that depends on the overall state of the game and on the actions of all agents. In a sense, the individual component encodes the individual goal of agent k while the common component determines the situations in which interaction must occur. The authors assume that the agents know nothing about the transition probabilities or about any of the reward functions r_k. Each agents must therefore learn from experience not only its "individual goal" but also in which states it should coordinate with the other agents.

The main algorithm which you have to implement (see the section on algorithms), and for which you can find the full details in the paper of Melo & Veloso, is the following:

Algorithm 1 Learning algorithm for agent k

```
1:  Initialize $Q_k^*$ and $Q_k^C$;
2:  Set $t = 0$;
3:  while (FOREVER) do
4:      Choose $A_k(t)$ using $\pi_e$;
5:      if $A_k(t) =$ COORDINATE then
6:          if $ActivePercept =$ TRUE then
7:              $\hat{A}_k(t) = \pi_g(Q_k^C, X(t))$;
8:          else
9:              $\hat{A}_k(t) = \pi_g(Q_k^*, X_k(t))$;
10:         end if
11:         Sample $R_k(t)$ and $X_k(t+1)$;
12:         if $ActivePercept =$ TRUE then
13:             QLUpdate$(Q_k^C; X(t), \hat{A}_k(t), R_k(t), X_k(t+1), Q_k^*)$;
14:         end if
15:     else
16:         Sample $R_k(t)$ and $X_k(t+1)$;
17:     end if
18:     QLUpdate$(Q_k^*; X_k(t), A_k(t), R_k(t), X_k(t+1), Q_k^*)$;
19:     $t = t + 1$;
20: end while
```

Figure 1: Learning algorithm for the agent k

You are strongly advised to read the paper, since it contains most of the information to finish this assignment.

2 Environments

You have to implement the following three environments (the details are in the paper as well), and can disregard the other environments from the paper. Do not forget to follow the PettingZoo API to implement your environments.

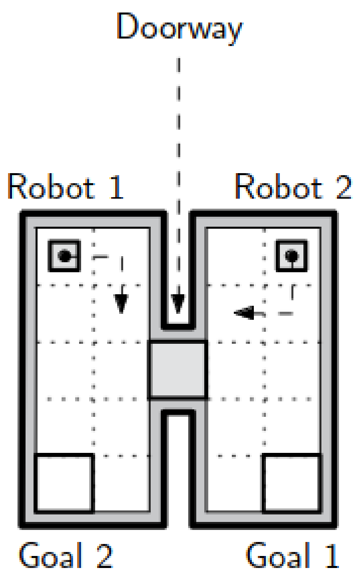

Figure 2

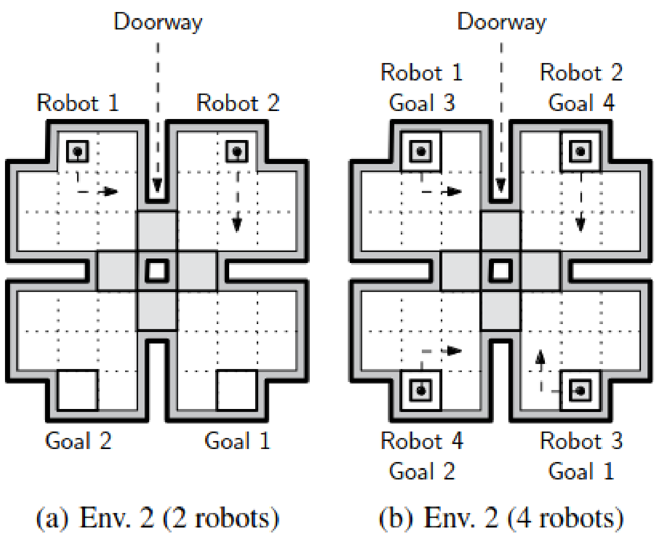

Figure 3

In the first environment, two robots must move from an initial location to their goal location in a gridworld consisting of 21 states. As depicted in Figure 2, each robot starts in one corner of the environment and must move to the opposite corner, receiving a reward of 10 for succeeding. If both robots simultaneously end up in the shaded state (the doorway) they both receive a penalty of -20 (a "miscoordination" penalty). Each robot can choose between 4 possible actions, each moving the robot in one of the 4 possible directions (North, South, East and West). These actions succeed with a 0.8 probability and fail with a 0.2 probability (meaning the action simply does not occur and the agents stand still). When agents try to occupy the same spot, they succeed, but in the doorways they do get the miscoordination penalty. The first environment in Figure 3 is similar to the previous one, only considering a different gridworld with 36 states. In this environment there are 4 doorways granting a penalty to both agents if both simultaneously end up in the same doorway. In the third environment there are 4 robots, again in the 36-state environment. In all scenarios, and in order to discourage the excessive use of the

COORDINATE action, each agent receives a penalty of -1 every time it chooses this action. Notice that the "excessive" use of the COORDINATE action impacts the sparsity of Q_k^C , which is undesirable for large problems.

You can create your environments however you like: a single-channel 2D grid where a certain ASCII character represents a certain kind of object or agent, or a binary grid with multiple channels in the third dimension (one binary channel for walls, one for agents, one for doors, etc.), or any representation that you think might be useful. But you must be able to have a function that prints your environment to a human-readable format at any time step in the console/terminal in ASCII format. **Include a print of all of your three environments in your report at the initial state in ASCII format.**

3 Algorithms

For this assignment, in addition to implementing the environments, you are required to implement two algorithms:

- Independent, self-interested Q-learning agents (so just regular Q-learning per agent, and run all agents simultaneously). Use an epsilon-greedy exploration strategy and do not forget to update the exploration rate at the end of every episode:

  ```
  epsilon = max(epsilon * decay_rate, epsilon_min)
  ```

  Tune your hyperparameters (reasonably, you do not need to acquire perfection; manual fine-tuning is fine, but you can use grid searches or other approaches if you like), such as the learning rate of your agents and their exploration parameters (hyperparameters are all uniform over agents). $\epsilon_{max} = 1$, but ϵ_{min} and the decay rate should be set so epsilon decreases slowly enough towards ϵ_{min}, more or less somewhere between 2/3rds or at the end of your training episodes, depending on your choice of ϵ_{min}. You should mention your final parameters in your report. Set the discount factor $\gamma = 0.99$.

- The sparse interaction algorithm of Melo & Veloso (Figure 1, and the aforementioned paper).

For all three environments, train each algorithm for 10 000 episodes. You will also evaluate your trained agents after every 1000 (including 0 and 10 000) training episodes, with each evaluation being an average of 32 episodes (so turn off exploration and see how your Q-tables fare greedily for 32 episodes, only then resume training) for all algorithms.

Plot both your training total (discounted) rewards per episode, averaged over the agents, as well as your total (discounted) evaluation rewards (averaged over agents, but also averaged over the 32 episodes). Now do all of this for 7 separate training runs, so you can plot standard deviations as error bars over these 7 training runs (to use shaded error regions for a non-clouded visibility of your training error bars, see this matplotlib documentation), and also error bars for every evaluation run after every 1000 episodes.

To be clear, you will have one plot per environment per algorithm, where each plot contains one line (with error bars) connecting all training episode rewards, and one other line (or scatter plot) representing all evaluation rewards (with x-values a 1000 episodes apart from each other on the x-axis).

Furthermore, besides keeping track total rewards, you will also keep count of the amount of miscoordinations per episode, and plot these in the same manner as for the total rewards.

Finally, also keep track of the amount of steps in which an episode ends, and plot them in the same manner again. An episode ends when all agents are in their goal state, or when they reach the limit of 100 time steps. You should now in total have 18 plots (2 agent algorithms x 3 environments x 3 kinds of variables you keep track of).

Next, discuss your results through your plots. Compare especially the results between algorithms, and talk about why you think these results might differ or not.
Finally, plot a typical trained episode in your report (i.e., a printout of the environment per time step until the agents reach their goal states) for the sparse interaction learners. Are there asymmetries between agents? Discuss.