Assignment 3: Matheuristics and deep Q-learning

Deadline: Dec 22 at 12h00 (late submissions are not tolerated)
Questions: during the dedicated lab of Dec 08 or via email (count two working days for an answer)

Guidelines

- Grading is transparent and is **entirely based on the marking scheme** available on Canvas.
- The assignment measures the same skills as the ones that were developed during the lectures and the workshops. Therefore, **you can use the lecture and workshop elements to get some inspiration** for your code, your graphs, and your answers.
- There is **no hard limit as far as the report length is concerned**. For indicative purposes, most reports will have between 7 and 13 pages (including graphs and tables), with an average of 10 pages.
- For assessing your work, we are **re-running your code**, or parts of your code at least. As a result:
  - Make sure that **your code is running** – the marking scheme is clear, it is 0 for the coding part if the code is not running;
  - Make sure that **your code is in its final version**: there should not be any major difference between the report and the Python implementation — major means, for example, a different reward system, not a different parameter value;
  - **We do not need your code to print every single information from the report**. It is okay to have some parts that are commented (with an indication within the code of what a commented part does if it is uncommented);
- If you realize in the middle of the report that there was a better representation for the problem and you do not have the time to start over, **stick with your first representation**. At the end of the report, you can mention your other (maybe better) ideas. **Do not mix representations and models in the middle of the report** as it makes things confusing.
- Make sure you **manage your time well**:
  - **Do not start the assignment at the last minute** since your agents will need some time to train;
  - Check the marking scheme and **focus first on the tasks that are given**;
  - If – and only if – you have some extra time and some extra interest in the topic, you can go the extra mile with advanced heuristic testing, evaluation of more than one agents' performance, fancy graphs, and other extra answers ... but be aware that **the extra effort might not be worth the reward**;
  - If you lack of time to finish the assignment, decrease the size of your training dataset. But be aware that this will also damage the accuracy and the performance of your agent;

Good luck with the assignment!

Exercise 1 [5.5 points]

Note: In this exercise, gathering the data to train your neural network can take some time as you need to solve an $\mathcal{NP}$-hard problem exactly in order to know if a variable is used in an optimal solution. Make sure to take that information into account before starting the exercise!

We study the 0-1 quadratic knapsack problem that can be defined as follows: given a knapsack with capacity c, a set of n items with weight w_i and profit $p_i (i = 1, \ldots, n)$, and a set of extra profit p_{ij} earned if both items i and j are selected ($i = 1, \ldots, n, j = i+1, \ldots, n$), the 0-1 Quadratic Knapsack Problem (0-1 QKP) consists in finding a set of compatible items with maximum profit that fits into the knapsack. See the paper of Billionnet and Soutif [1] for an introduction to the problem and an ILP formulation (check the linearized model (8)-(10) called **CLIN** in the paper and hereafter).

1. [1.0] Using a 0-1 QKP instance generator inspired (or not) by the one described in Section 5.1 of Billionnet and Soutif's paper, identify parameter values for which the generated instances cannot be solved by an ILP solver in less than 30 seconds (use Gurobi as ILP solver and **CLIN** for the formulation). Mention the optimality gap for the instances that could not be solved.

2. [0.5] According to you, what are the reasons why such instances cannot be solved to optimality by the ILP solver?

3. We now explore the possibility to solve model **CLIN** with a subset of variables determined by a neural network:

   (a) [1.0] Determine what variable features could serve as input layer and outline every relevant aspect of the training data set.

   (b) [0.5] Suggest an architecture for the neural network and a training scheme.

   (c) [2.5] Implement the proposed approach, train the neural network, determine a cut-off threshold, and test the matheuristic against the complete model (with a time limit of 30 seconds).

Exercise 2 [4.5 points]

The bin packing problem can be defined as follows: we are given a set of m items with weight $w_j (j = 1, \ldots, m)$ and an unlimited number of identical bins with capacity c, the Bin Packing Problem (BPP) consists in packing all the items into the minimum number of bins. See the paper from Delorme et al. [2] for an introduction to the problem and an ILP formulation. In this exercise, we are interested in the online version of the BPP in which item j ($j = 1, \ldots, m$) arrives and needs to be packed before item $j + 1$ (if any) arrives. Note that decision making is more difficult for online problems than for offline problems. Take for example an online BPP instance where $c = 10, m = 4, w_1 = w_2 = 4$:

- packing items 1 and 2 together leads to an optimal solution if $w_3 = w_4 = 7$ but not if $w_3 = w_4 = 6$,
- packing items 1 and 2 in separate bins leads to an optimal solution if $w_3 = w_4 = 6$ but not if $w_3 = w_4 = 7$.

However, one needs to decide whether items 1 and 2 are packed in the same bin before knowing w_3 and w_4.

1. [1.0] Describe and implement some greedy strategies for the problem.

2. [1.0] We now assume that we want to solve the problem using deep Q-learning. Propose a way to define the states and the actions. For example, an action can be a greedy move (e.g., pack the next item in bin i), one step of a greedy heuristic (e.g., pack the next item in the most loaded bin among the bins in which the item fits), or something else. Remember that the weights of subsequent items is not known by the agent when deciding the bin in which the current item should be packed. However, the total number of items m is known by the agent.

3. [1.0] Following the structure of "Knapsack_Env.py" and "MVC_Env.py", create a suitable environment "OBPP_Env.py". Pick a set of parameters for the instance generation, but keep in mind that the goal is to produce a deep Q-learning algorithm that is competitive with greedy strategies, so do not generate trivial instances.

4. [1.0] Train the DQN and save the model learnt by the agent. Provide in your reports meaningful graphics to show the evolution of the training.

5. [0.5] Create a file that calls the model learnt to find a good quality solution for an online BPP instance created by your instance generator. Compare the results obtained by the trained agent with those obtained by your greedy heuristics.

References

[1] Alain Billionnet and Éric Soutif. Using a mixed integer programming tool for solving the 0–1 quadratic knapsack problem. *INFORMS Journal on Computing*, 16(2):188–197, 2004.

[2] M. Delorme, M. Iori, and S. Martello. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research*, 255:1–20, 2016.