SFWRENG 3XB3

L2 – Bin Packing Lab

McMaster University
ENGINEERING
Computing & Software

GENERAL INFORMATION

- Author: Dr. Sébastien Mosser mossers@mcmaster.ca

- Start date: 17/10/2022
- Delivery date: 06/11/2022
- Weigh in final grade: 20%

LEARNING OBJECTIVES

The first week is guided in a step-by-step way. Then, you'll be more autonomous. At the end of this three weeks assignment, students should:

- **Know and understand:**
  - How heuristics can help to solve a NP-hard optimization problem
  - The difference between benchmarking an algorithm and evaluating a solution
- **Be able to:**
  - Implement non-trivial bin packing algorithms
  - Provide evaluation metrics for solutions
  - Provide execution benchmark using state-of-practice tooling

WHAT IS BIN-PACKING?

Bin-packing is an optimization problem, known to be NP-hard. A NP-hard problem is informally defined as a problem being "***as hard as the hardest NP problem***". In a nutshell, the principle of bin-packing is… to pack things into bins while respecting some capacity constraints. There are several variants of this problems, as the notion of "things" can be quite diverse. In this assignment, we will only consider 1D bin-packing, i.e., we only consider as important the weight of each thing we have to pack.

To solve this problem, we can use an infinite number of bins, each one being of fixed capacity. Objects are then used to fill each bin, and the global objective is to use a minimal number of bins. Consider the following situation[1]: you have bins that can hold up to 10 pounds, and you have the following objects to pack, characterized by their weights: {4, 8, 1, 4, 2, 1}. We can pack these objects using two bins: the first one will contains objects {4, 4, 2}, and the second one the remaining {8, 1, 1}. In this case, the solution is complete, i.e., each bin is completely filled.

Bin-packing algorithms are widely used, from balancing cargo trunks in planes to allocating resources in cloud environment. For example, Google HashCode 2017 (code competition organized by google) was about allocating YouTube videos on cache servers to minimize bandwidth consumption.

In this assignment, we consider two classes of algorithms:

[1] https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/

1. **Offline algorithms** work by knowing all the objects to be allocated into bins since the beginning. It allows one to globally optimize the weight allocation. This is classical when filling a cargo trunk: you already know all the objects that needs to be stored, and want to fill the trunk in a way that balance the weight to reduce fuel consumption.
2. **Online algorithms** discover the objects on-the-fly. It usually leads to less optimal solutions, as it cuts out the possibility to reason globally on the set of objects. This situation is typical when allocating on-demand resources, as you cannot know the demand in advance.

We provide a reference dataset. The dataset contains different reference cases. The file oracle.xlsx binds each case to its proved optimal solution (as number of bins used foir this given case). We will also compare our work with a baseline implementation of bin-packing algorithm available in python (https://pypi.org/project/binpacking/).

MAIN CHANGES WIRTH RESPECT TO LAB #1

- You asked for more guidance at the code level, thus we provide a more structured code base to start with for this lab.
- You found the planning provided for Lab #1 terrifying, so for this lab we do not provide any hints. It is your responsibility to plan accordingly, and deliver your work in an iterative and incremental way.
- You asked for rubrics instead of grading scheme. Please read carefully the grading rubrics at the end of this document.
- The lab workload was sized according to the following protocol: the instructor implemented a solution to the different parts of the assignment. It took 5 hours. You have 12h of supervised work per team member (so 24h in total), non withstanding personal (unsupervised) work.

PREAMBLE

You are not allowed to modify the Pipenv file (except for the python version, among 3.8, 3.9 and 3.10). The only libraries that you can uses are the one available in python core library, and matplotlib for graphing your results. Typically no NetworkX, panda, scipy, solvers, …

PROJECT PREPARATION

1. Find a teammate in your lab section.
    a. If your lab section contains an odd number of students, then one (and only one) group can be composed of three students. It'll require manual action on Avenue, please check the situation with your TA.
2. One member has to fork the "L2 – bin Packing Lab" project on the GitLab interface:
    a. URL: https://gitlab.cas.mcmaster.ca/sfwreng-3xb3-fall-22/l2-bin-packing
    b. Make sure your project is "private"
    c. Add your teammate as "maintainer" of the project
    d. Add your TA and the instructor as a "reporter"
3. **Declare your group on Avenue by the end of your first lab**
    a. Check that you have access to the submission interface

WORK TO DO

The assignment is designed as a set of independent tasks. Your job is to work on these tasks, and report your results in the Jupyter Notebook named `report.ipynb` in the repository. You will be evaluated on the quality of your code (business logic and tests) as well as the quality of your reports.

T_1: UNDERSTAND THE CODE BASE

- Look at the code and dataset provided in the initial repository.
  - Take a particular look at how the "import" mechanism is used (using . and .. to navigate between modules) to avoid having to manually edit "sys.path" (which is a bad practice).
  - Pay attention to the type aliasing used to emphasize readability (Solution, WeightSet and WeightStream)
- For the code, explain how it follows SOLID principles.
- For the dataset, explain the different dimensions used, and why they are important for evaluating algorithms. Each dataset contains a file thet links to its source, where you can find explanation of its structure.
- To be more familiar with the code base, implement:
  - A new DatasetReader, to support the cases stored in the jburkardt directory. Integrate it in the class hierarchy.
  - The most terrible online bin-packing algorithm that can ever exist: this algorithm uses one bin per object. Integrate it in the class hierarchy.

T_2: IMPLEMENT AND BENCHMARK ALGORITHMS

Now that you have understood the provided source code, we can start by doing some bin-packing. The Wikipedia page for bin-packing

- Provide an implementation of the following algorithms:
  - Online[2]: First Fit, Best Fit, Worst Fit
  - Offline[3]: First Fit Decreasing, Best Fit Decreasing, Worst Fit Decreasing
- The number of bins used by a solution is important. But how to compare solutions that uses the very same number of bins?
  - Implement a way to evaluate a given solution, by measuring KPIs related to number of bins used, and any other things that can be useful to characterize a solution.
- Using pyperf, implement benchmarks to measure the execution time of your different implementations. Your report must mention which part of the dataset you used for benchmarking, and why is your benchmarking protocol relevant.
- In your report:
  - Plot the results of your benchmarking protocol in your report
  - Analyse the different algorithms

T_3: MEASURE IMPROVEMENT MARGIN

As we are using reference dataset, we can leverage the fact that, for each case, we know the minimal number of bins that was proved to be the optimal one. Thus, we can evaluate our algorithms against this value to measure our margin of improvement.

- Implement an experiment that compares all your algorithms and the provided baseline with the optimal number stored in the `oracle.xslx` file (you can transform these data into CSV file for example).

---

[2] For the first three ones, you can adapt the code available here for example: https://www.geeksforgeeks.org/bin-packing-problem-minimize-number-of-used-bins/. The main difference is that we want a solution, not only a number of bins.
[3] It is legit to separate concerns and sort the objects first, then call the online version. Like we did for Next Fit.

- Measure the margin of improvement with respect to the optimal solution. You can evaluate this margin in two different ways:
    - A discrete one: the optimal solution is found, or not found.
    - A continuous one: the optimal solution is "n" bins smaller than this solution.
- Plot your results, and analyse it:
    - Which cases (or classes of cases) are problematic? Why?

T_4: ADD SMARTER ALGORITHMS

The algorithms we have implemented are of the AnyFit class: they consider each object independently. If we normalize the weights of objects, we can use heuristics to classify the objects into categories (e.g., large, medium, small), and characterize the bins by their remaining space to decide how to fill. A good example is the Refined First Fit algorithm[4].

- Provide a way to normalize the dataset cases when reading them;
- Implement a version of the Refined First Fit algorithm
- Add it into your benchmark, plot the results and analyse them

T_5: FROM FIXED CAPACITY TO FIXED BINS

The algorithms we have implemented to date consider an infinite amount of bins. If we think about bin-packing as a way to allocate resources (e.g., connect websites' visitors to web server to balance the load and avoid outages), the number of bins (here the number of servers) is fixed. For example, Reddit allocates all its traffic among 9 webservers[5].

Thus, we have to consider a variant of the bin-packing problem where the objective is not to use less bin as possible, but to balance the load over a fixed number of bins. This problem is known under the name "Multiway Number Partitioning"[6]

- Modify the `macpacking` library to support this new variant of the problem
    - Integrate a baseline by using the "`to_constant_bin_number()`" method available in the binpacking library
- Are your KPIs adequate to evaluate such new objective for a solution?
    - More generally, what is reusable in your code to support this?
- Implement at least one algorithm, and compare it with the baseline (benchmark, plot and analyse)
- Compare these results with the one for the classical bin packing algorithms.

BONUS TASK (UP TO 10 BONUS POINTS)

The bonus task will only be considered if the five tasks are delivered and falls into the "Meet" or "Exceed" categories of the grading rubrics. Maximum grade is still 100.

- Implement an algorithm of your choice capable of beating the baseline for the classical in-packing problem.
- Demonstrate how you beat the baseline in your report and explain why your algorithm provides better results.

[4] https://en.wikipedia.org/wiki/First-fit_bin_packing#refined
[5] Blatant advertisement for the CAS 735 grad course if you're interested by doing grad studies at Mac.
[6] https://en.wikipedia.org/wiki/Multiway_number_partitioning

- Benchmark the execution time of your algorithm with respect to the baseline.

DELIVERING YOUR ASSIGNMENT

MSAF POLICY

As this assignment is three weeks long, submitting an MSAF for the day of the deadline does not cancel it. When submitting an MSAF on Mosaic:

- An automatic 3-days extension is granted to this assignment on Avenue (for the group), and your GitLab repository will be automatically cloned at the new deadline.
- To avoid any domino effect with the upcoming code competition, the student who submitted an MSAF also receives a 3-days extension for the next code-competition, as a buffer to catch-up and recover.
- The instructor implements these relief actions when receiving the automated MSAF email from Mosaic. If your case is different from "classical" situations, please communicate with the instructor using direct message on Teams.

CODE AVAILABLE ON GITLAB

- Your GitLab repository will be automatically cloned at the deadline
- Your tests will be run using the following command:
  - `pipenv run python -m coverage run -m pytest -v`
- Flake8 will be run using the following command:
  - `pipenv run python -m flake8`
- Your report will be executed by running Jupyter and then restraing the kernel
  - `pipenv run python -m jupyter lab  report.ipynb`

SUBMIT YOUR REPORT ON AVENUE

- Print a PDF version of your Jupyter notebook.
  - Check that the whole text is in your PDF (and not just the first 80 characters of your code cells)
- Submit this PDF on Avenue

Late deliveries will not be considered. Too late is too late.

END OF ASSIGNMENT.

Appendix – Grading Rubrics*					
Category	Part	Below [0-49]	Marginal [50-69]	Meets [70-84]	Exceed [85-100]
T1 (10%)	*Analysis*	*No analysis*	*Shallow or incomplete analysis.*	*Provide an adequate analysis of the code.*	*Depth of reflection and analysis above and beyond*
	Code & design	*No integration with provided code base*		*Code integrated*	
T2 (20%)	*Code & design*	*Non-functional code*	*Functional code with flaky design*	*Design respects SOLID principles, encapsulation and visibility OK.*	*Use of OOP features and Design decisions above and beyond.*
	Benchmark	*No use of Pyperf*	*Benchmark data not justified*	*Benchmark code and used data are justified. Choice of plotting relevant for the benchmark*	*Justification and relevance of the benchmark of exceptional quality*
	Analysis	*No analysis*	*Shallow or incomplete analysis.*	*Provide an adequate analysis of the code and the benchmark results.*	*Depth of reflection and analysis above and beyond*
T3 (10%)	*Benchmark*	*No use of Pyperf*	*Benchmark data not justified*	*Benchmark code and used data are justified. Choice of plotting relevant for the benchmark*	*Justification and relevance of the benchmark of exceptional quality*
	Analysis	*No analysis*	*Shallow or incomplete analysis.*	*Provide an adequate analysis of the code.*	*Depth of reflection and analysis above and beyond*
T4 (20%)	*Code & design*	*Non-functional code*	*Functional code with flaky design*	*Design respects SOLID principles, encapsulation and visibility OK.*	*Use of OOP features and Design decisions above and beyond*
	Analysis	*No analysis*	*Shallow or incomplete analysis.*	*Provide an adequate analysis of the code.*	*Depth of reflection and analysis above and beyond*
T5 (20%)	*Code & design*	*Non-functional code*	*Functional code with flaky design*	*Design respects SOLID principles, encapsulation and visibility OK.*	*Use of OOP features and Design decisions above and beyond*
	Benchmark	*No use of Pyperf*	*Benchmark data not justified*	*Benchmark code and used data are justified. Choice of plotting relevant for the benchmark*	*Justification and relevance of the benchmark of exceptional quality*
	Analysis	*No analysis*	*Shallow or incomplete analysis.*	*Provide an adequate analysis of the code.*	*Depth of reflection and analysis above and beyond*
Misc. (20%)	*Tests*	*No tests*	*Flaky tests (e.g., no use of assertions)*	*Tests covers code and identify proper edge cases, good usage of pitest.*	*Test organisation is impressive and reusable.*
	Environment	*Flake8 reports full of error, no attention paid to report global quality*		*Attention paid to the global quality of the report and code*	*Above and beyond in terms of presentation.*
	Self-reflection	*No answers*	*Binary (or super short) answers*	*Answers describe the experience of the students adequately. Plan for action to support improvements*	*Depth in the self-reflection and plan for action*

**Instructor reserves the right to adapt the marking scheme (while preserving its spirit) during the marking process. Parts contribute equally to each catego*

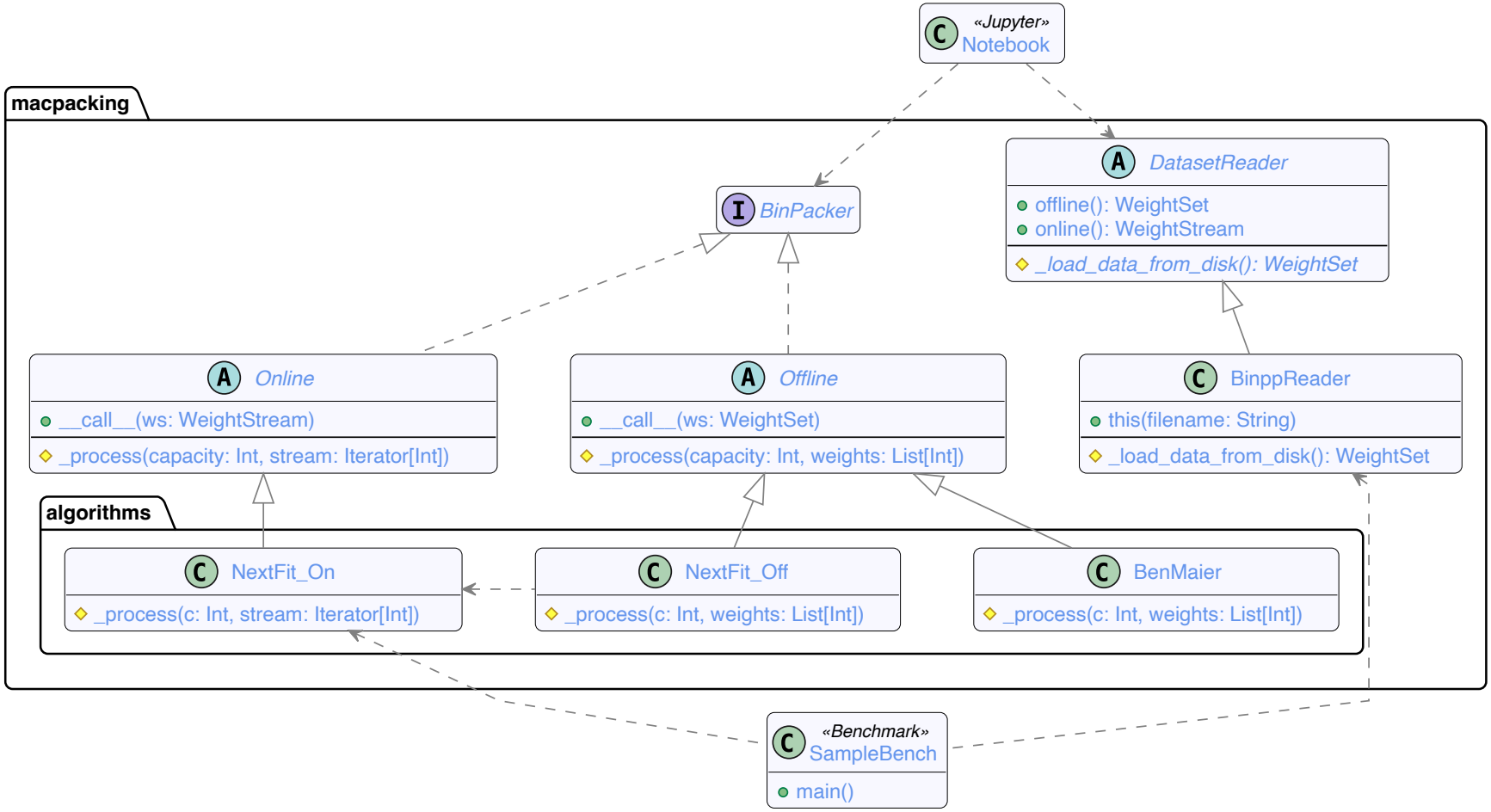

Figure 1. Class diagram of the provided code