Algorithms and Data Structures for Biology
21 April 2020

Ugo Dal Lago Guillaume Geoffroy

This assignment will be the second one to be marked. Please carefully read all the instructions below before starting to work on the assignment.

1 Instructions

This assignment must be completed individually, the deadline is on May 2nd 2020 at 23.59 CET. To complete the assignment, you need to upload the following files to the course's IOL page (where a specific link will be available):

- One Python file named `FIRSTNAME_LASTNAME.py`, with the code you have developed.
- One PDF file name `FIRSTNAME_LASTNAME.pdf`, preferably produced by way of LaTeX, including a description of the algorithms you designed, the relevant parts of the `Python` code you wrote, and the experimental results.

2 The Problem

Suppose you are a researcher who needs to move all her equipment to a new workplace on the other side of the world, where she just got a new job. Her research equipment consists of n objects, the i-th object having a weight equal to w_i kilograms. The way the researcher is supposed to send her equipment is by way of containers. Each container has enough *space capacity* to contain an arbitrary amount of objects, in principle all the n objects. There is however a constraint: each container is, for structural reasons, able to carry at most C kilograms, where C is typically smaller than $\sum_{i=1}^{n} w_i$, but higher than each of the w_i. In other words, the researcher needs more than one container, each of them costing a fixed amount p to be sent. The problem the researcher faces, then, is allocating the objects to as few containers as possible, this way minimizing the cost of shipping all the n objects to their final destination.

This assignment asks you to:

1. Model the problem described above as a combinatorial optimization problem (actually one we have encountered at some point in our course).
2. Give an exhaustive search algorithm solving the combinatorial problem from the previous point, and analyse the algorithm's complexity: give an upper bound to the worst-case computation time, in big O notation, in function of n.
3. Give a greedy algorithm that solves the same combinatorial problem from point 1 (or more precisely, that gives an approximate solution), and analyse the algorithm's complexity. Again, the bound should be a function on n. Give an upper bound to the approximation ratio of this algorithm (that is to say, the ratio of the cost of the solution it proposes to the cost of the optimal solution).
4. Make up two instances of the problem, one with $n = 4, C = 2$, the other with $n = 6, C = 2.5$, and both with the w_i between 0 and 1. Then,

- solve them by hand (*i.e.* find the optimal solutions),
- apply by hand your greedy algorithm on each of them (you may obtain sub-optimal solutions).

In your report, you do not need to give the details of your calculations: just describe both instances and give both sets of solutions.

5. Implement the two algorithms you designed as two `Python` functions. The two functions should have the same interface (*i.e.* they should have the same arguments and return their results in the same format). Please do not use external modules.

6. Write an automated testing routine that runs each function on each instance from point 4 and checks that they give the appropriate results.

7. Write a benchmarking routine that takes as argument either of the two functions (which is why they need to have the same interface) and runs it on randomly generated inputs in which

   - n is, successively, $4, 6, 8, 10, 12, 14, 16, 18, 20$,
   - the weights w_i are drawn uniformly and independently between 0 and 1.
   - the capacity C of each container is chosen as $\sqrt{n}$.

   The benchmarking routine should make use of the `random` module only.

8. Use the module `timeit` to experimentally test the running time of both functions. Give a graphical presentation of your results. Moreover, try to find the best-fit curves matching the two algorithms complexity by way of `numpy` and `scipy`.

9. Evaluate the (worst-case) approximation ratio of the solution given by your greedy program (with respect to the exact solution provided by your exhaustive program). Is it within the theoretical bounds you established in point 3?

As previously mentioned, the results of your work should be summarized in a PDF file. The report should be self-contained and allow the teachers to judge what you have done without delving into the details of your `Python` code or running the tests themselves.