Department of Mathematics and Computer Science
University of Southern Denmark, Odense

May 15, 2025
Marco Chiarandini

AI505 – Optimization

Sheet 11, Spring 2025

Solution:
Included.

Exercises with the symbol $^+$ are to be done at home before the class. Exercises with the symbol * will be tackled in class. The remaining exercises are left for self training after the exercise class. Some exercises are from the text book and the number is reported. They have the solution at the end of the book.

Exercise 1*

A manufacturing startup provides 3D printing services to both private and business customers. A single 3D printer is used to produce custom metal parts that must be printed without interruption. Once a part is printed and is allowed to cool down, it is ready for delivery and the machine becomes available to print the next part. Each part must be ready by a given deadline, which is set at the time the order is placed. Missing the deadline leads to a penalty that depends on the part, and is proportional to how late the part is completed. The time needed to print a given part, including the cooldown time, can be determined in advance.

The problem consists in determining the order in which a set of parts should be manufactured on the single metal 3D printer so as to minimize the total penalty incurred by the service provider.

Input The input contains $3n$ integers separated by whitespace (spaces, tabs, or newlines). The first n integers are the processing times p_j, the next n integers are the weights w_j, and the last n integers are the due dates d_j of the various parts $j = 1, ..., n$.

Solution evaluation A feasible solution is a permutation $\pi = (j_1, j_2, \ldots, j_n)$ of the integers $1, \ldots, n$, where j_i denotes the index of the ith part to be printed. Given the permutation π, the *completion time* of job j_i is

$$C_{j_i} = \sum_{k=1}^{i} p_{j_k} \qquad \text{for each } i = 1, \ldots, n$$

and the *tardiness* of job j is $T_j = \max\{C_j - d_j, 0\}$.

The cost of a solution π is the *total weighted tardiness*:

$$\sum_{j=1}^{n} w_j T_j$$

Output Print the part indices $j_1, j_2, \ldots, j_n$ of the n parts in the order in which they should be processed separated by whitespace.

Example Input:

```
26 24 79 46 32
1 10 9 10 10
80 220 180 50 100
```

Output (cost 67):

```
4 1 5 3 2
```

Problem instances A set of three problem instances can be found in this archive. The problem was posed by Carlos Fonseca and Alexandre Jesus at the SIGEVO Summer School 2024. It is an example of the scheduling model: *single machine total weighted tardiness problem* that captures the core of several real-life applications in manufacturing, services and computer processing.

Your Task Model the single machine total weighted tardiness problem as a search problem.
In the following you find some guiding questions.

Problem instance (known parameters) What (known) data is required to fully characterize a particular instance of the problem? This must be available in advance, and is not changed by the solver in any way.

Solution (decision variables) What (unknown) data is required to fully characterize a (feasible) candidate solution to a given problem instance? This is the data needed to implement the solution in practice, and will be determined by the solver during the optimization run.

Objective function How can the performance of a given candidate solution be measured? This depends only on the problem instance and the actual solution itself, and never on how the solution was actually found. Is the corresponding value to be minimized or maximized?

Neighborhood Structure for Constructive Search How can a solution be constructed piece by piece? If solutions are seen as subsets of a larger ground set of solution components, the construction process consists simply in successively adding components to the empty set according to some construction rule. The partial solutions generated during the construction process represent all feasible solutions that contain them, and their performance can be inferred from those sets in terms of suitable lower bounds (minimization) or upper bounds (maximization).

Neighbourhood Structure for Local Search What makes two given solutions similar to each other? This usually means that parts of the two solutions are somehow identical, but it should also happen that they exhibit similar performance in most cases. A candidate solution that performs at least as well as all solutions similar to it (its neighbours) is called a local optimum of the corresponding problem instance.

The choice of the neighbourhood structure is particularly important for the success of local-search algorithms. By ensuring that similar solutions tend to exhibit similar performance, one seeks to induce fewer local optima and large basins of attraction to those optima, although this can seldom be guaranteed. Furthermore, any two feasible solutions should be connected by a sequence of consecutive neighbours, so that the unknown global optimum can, at least in principle, be reached from any initial solution.
Similarly, the choice of ground set and construction rule, and the quality of the associated bounds, are very important for the success of constructive-search algorithms. In particular, it should be possible to construct all feasible solutions, and the inferred quality of partial solutions should be as accurate as possible so that the construction process can effectively rely on it.

Once suitable answers to the above questions are obtained, a more refined set of questions relative to the computer implementation of the model can be considered.

Problem instance representation How should the problem instance data be stored in a data structure so that the objective function and corresponding bounds can be easily computed?

Solution representation How should (possibly incomplete) solutions be represented, i.e., stored as a data structure, so that:

- Their performance can be evaluated efficiently through the objective function or related bounds?
- (Constructive search) Feasible solutions can be easily constructed by successively adding components?
- (Local search) Solutions can be easily modified to obtain neighbouring solutions?

Solution evaluation How can the objective function and/or corresponding bounds be computed given the instance data and the solution representation?

Move representation How can moves be represented, i.e.,

- (Constructive search) the addition or removal of components to/from a (partial) solution?
- (Local search) changes that, when applied to a solution, lead to a neighbouring solution?

Solution modification What are valid moves, and how are they applied to a solution?

Move evaluation How much would applying a given move to a (partial) solution change its performance? Can this effect be computed more efficiently without modifying the original solution than by evaluating it, applying the move and evaluating the result? How?