Implementation exercises for the course

# Heuristic Optimization

C. L. Camacho-Villalón and T. Stützle
Université Libre de Bruxelles — 2023

**Implementation exercise sheet 2**

---

Implement two stochastic local search algorithms for the permutation flow-shop problem with weighted tardiness (PFSP-WT) building on top of the constructive and perturbative local search methods from the first implementation exercise. Apply your algorithms to the same instances as in the first implementation exercise. The SLS algorithms can be based on either simple, hybrid, or population-based methods chosen among those described in the lectures. The two SLS methods chosen must belong to two different classes. For example, the first one could be a tabu search (simple SLS method) and the second one could be an ACO algorithm (population-based SLS method). To get inspiration for ways how to generate solutions, operators etc., you may consider algorithms that have been proposed in the literature for the same or similar problems. In fact, the PFSP-WT has not been very frequently treated in the literature. For inspiration you may also check papers on the related PFSP with sum completion time (or flowtime) objective and weighted tardiness. Some references are given below.

1. The most complete review of algorithms for the sum completion time PFSP including descriptions of ILS and IG algorithms:
   Q.-K. Pan, R. Ruiz. Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1): 31–43, 2012.
2. Example of constructive and ILS heuristics for the PFSP with minimise total tardiness objective:
   V. Fernandez-Viagas, J. M. Framinan. NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research*. 60, 27–36, 2015.
   L. M. Liao, C. J. Huang. Tabu search for non-permutation flowshop scheduling problem with minimizing total tardiness. *Applied Mathematics and Computation*, 217(2), 557-567, 2010.
3. Examples of ACO algorithms:
   C. Rajendran and H. Ziegler. Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. *Computers & Industrial Engineering*, 48:789–797, 2005.
4. Examples of memetic algorithms:
   L. Y. Tseng and Y.-T. Lin. A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem. *International Journal of Production Economics*, 127:121–128, 2010.
   Y. Zhang, X. Li, and Q. Wang. Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, 196(3):869–876, 2009.

The two algorithms implemented should be evaluated on all instances provided for the first implementation exercise and be compared using statistical tests. Optionally, to get extra points in your implementation exercise, you will measure the run-time distributions of a few selected instances and use them to compare the two algorithms. The implementation should make use of appropriately selected iterative improvement algorithms from the first implementation exercise. Justify the choice of the iterative improvement algorithm you use from the ones you implemented for the first implementation exercise. Please note that the experimental comparison should be run under same conditions (programming language, compiler and compiler flags, similar load on computer etc.) for the two algorithms. In other words, the observed differences should be attributable to differences in the algorithms and not to differences in the experimental conditions.

The deadline for the implementation exercise is: **May 19, 2023 (23:59)**

## Exercise 2.1

1. Run each algorithm five times on each instance. Instances are available on TEAMS and are the same as used in the first implementation exercise. As termination criterion, for each instance, use the average computation time it takes to run a full VND (implemented in the first exercise) on instances of the same size (50 or 100) and then multiply this time by 500 (to allow for long enough runs of the SLS algorithms).
2. Compute the average percentage deviation from the best known solutions of each SLS algorithms on each instance. Also compute the average percentage deviation across instances of the same size (i.e. across all instances of 50 jobs and across all instances of 100 jobs).

3. Produce correlation plots of the average relative percentage deviation for the two SLS algorithms (see lectures of Prof. Stützle), separating between the instance sizes, that is, either two separate plots or one plot where differences are clear between the two instance sizes.

4. Determine, using statistical tests (in this case, the Wilcoxon test), whether there is a statistically significant difference between the mean relative percentage deviations reached by the two algorithms for each instance size. Note: For applying the statistical test, the R statistics software can be used. The software can be download from `http://www.r-project.org/`.

5. [Optional] For each of the two implemented SLS algorithms and on the first 5 instances of size 50 jobs, measure the qualified run-time distributions to reach sufficiently high quality solutions (e.g., the best-known solutions available or some solution value close to the best-known one such as 0.1, 0.5, 1, 2best-known solutions). Measure the run-time distributions across 25 repetitions using a cut-off time of 10 times the termination criterion above.

6. Produce a written report on the implementation exercise:

   - Include a clear description of the implemented SLS algorithms, mentioning how they work, what are their algorithm components, and what were the computational results obtained by each of them. Justify also the choice of the parameter settings that are used and the choice of the method for generating (i) initial solutions and (ii) the iterative improvement algorithm.
   - Present the results as in the previous implementation exercise using tables and statistical tests.
   - Present the performance correlation plots and [optionally] present graphically the results of the analysis of the run-time distributions.
   - Interpret appropriately the results (statistical tests, run-time distributions, correlation plots) and make conclusions on the relative performance of the algorithms across all the benchmark instances studied.

**Additional information:**

- In order to pass the exam, you have to successfully complete this second implementation exercise.
- You need to do the implementation exercise by yourself — cooperation is forbidden.
- You have to submit the following items in **a zip folder with your name** via TEAMS:
  - (i) a report in `pdf` format with scientific article structure (see examples provided on TEAMS) that concisely explains the implementation, reports the above mentioned tasks (averages, standard deviations, run-time distributions, correlation plots and results of statistical tests), and interprets concisely the observed results;
    *(Note: some of the criteria to be evaluated in the report are: the use of a scientific article structure including abstract, intro, problem description, material and methods, results, conclusion; the use of tables and plots to present the obtained results; and the use of statistical tests to draw conclusion about the algorithms performance.)*
  - (ii) the source code of the implementation together with a `README` file explaining how to compile and/or execute the algorithms from a command line in Linux/MacOS;
    *(Note: some of the criteria to be evaluated in code are: compilation/execution without errors, the use of structures, code efficiency, indentation and comments.)*
  - (iii) a simple `.txt` file with the raw data that was used for statistical testing.

  It is important to stress that your code should compile/run on Linux/MacOS without errors and produce the requested outputs when executed on the provided instances; otherwise the implementation exercise will be considered insufficient.
- As programming language, you may use preferably `C`, `C++`, or `Java`. While using `Python` for this exercise is also possible, we recommend against it because it is much more slower than the alternatives. The sample routines are available only in `C++`, but you are free to adapt them at your convenience. Please make sure that your code is properly commented and indented, and that the `README` file mentions the exact commands for its compilation and execution.
- Happy coding :-D