Implementation exercises for the course

Heuristic Optimization

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

Implementation exercise sheet 1

Implement iterative improvement algorithms for the permutation flow-shop scheduling problem with weighted tardiness (PFSP-WT). Information on the FSP-WT is provided in the accompanying introduction to the implementation exercise. Apply the implemented algorithms to all instances of 50 and 100 jobs available on teams. We provide 20 instances taken from [1]. The format of each instance is the following:

- *#jobs #machines*
- processing times matrix p_{ij}
  (*the processing time p_{ij} is preceded by the number of the machine j*)
- text string "`Reldue`"
  (*meaning Release due, but irrelevant otherwise)*
- `-1` due date `-1` priority weight
  (*The values* `-1`, *in the first and third column, are placeholders, also irrelevant for the exercises*.)

An example of instance file is the following:

```
20 10
  0 25  1 77  2  2  3  2  4 86  5 68  6 47  7  2  8 91  9 64
  0 14  1 67  2 27  3  8  4 85  5 82  6 44  7 51  8 66  9 49
  0 99  1 89  2 92  3 28  4 80  5 27  6 33  7 72  8 32  9  4
  0  5  1 40  2  7  3  1  4 21  5  7  6 15  7 39  8 65  9 45
  0 68  1  6  2 64  3 17  4 79  5 47  6 31  7 67  8 75  9 21
  0 98  1  5  2 44  3 76  4 59  5 59  6 53  7 57  8 89  9 56
  0 72  1 85  2 84  3 43  4 67  5 96  6 44  7 19  8 82  9 19
  0 39  1 62  2 76  3 79  4 27  5 94  6  8  7 11  8 11  9 75
  0 79  1 19  2 43  3 88  4 29  5 65  6 22  7 18  8 82  9  3
  0 13  1 73  2 18  3 13  4  5  5 19  6 88  7 75  8 33  9 96
  0 24  1 55  2 27  3  3  4 13  5 82  6 30  7 24  8 64  9 97
  0 14  1 24  2 91  3  2  4  8  5 44  6  1  7 96  8 82  9 17
  0 82  1 18  2 23  3 39  4 84  5 33  6 24  7 61  8 53  9 53
  0 90  1 64  2 17  3 39  4 42  5 13  6 81  7 68  8 67  9 38
  0 28  1 41  2 84  3 16  4 15  5 88  6 83  7  1  8 21  9 25
  0 50  1 55  2 90  3 95  4 18  5 93  6 96  7 35  8 24  9  7
  0 75  1 37  2 98  3 40  4 13  5 78  6 10  7 21  8 44  9 25
  0 86  1 44  2 93  3 13  4 18  5  4  6 86  7  9  8 59  9 24
  0 57  1 45  2 96  3 27  4 75  5 57  6 15  7 27  8 39  9 82
  0  4  1  5  2 56  3 73  4 23  5 44  6 98  7 63  8 84  9 53
Reldue
-1        1039        -1        6
-1        3499        -1        2
-1        3638        -1        7
-1        1667        -1        4
-1        3240        -1        5
-1        4027        -1        2
-1        3512        -1        1
-1        3892        -1        4
-1        1054        -1        8
-1        1806        -1        5
```

As variance reduction technique for the experiments make sure that the algorithms on a same instance start from the same initial solution. This can be ensured by using, for each of the executions on a same instance, the same random number seed. Also available on TEAMS is a "skeleton" in `C++` with a few sample routines that you can use for your implementation.

The deadline for the implementation exercise is: **April 14, 2023 (23:59)**

Exercise 1.1

Implement iterative improvement algorithms with either first-improvement or best-improvement pivoting rule for each of the following three neighborhoods: transpose, exchange, and insert.

As a starting solution for iterative improvement consider two possibilities. The first is to generate a random job permutation, that is, to use the method called *random uniform permutation* (see the slides of lecture). The second is to use the *simplified RZ* heuristic (again: see the slides of lecture for details).

1. Apply the 12 resulting iterative improvement algorithms (all combinations of two pivoting rules, three neighborhoods, and two possible initial solutions) to solve the 20 instances of the problem. Perform experiments with the 12 algorithm by executing five times each algorithm on each instance. Compute the following statistics for each combination of algorithms.

   - the average percentage deviation from best-known solutions,
   - the total computation time across all instances of the same size.

   Present the above information in a table grouped by algorithm and differentiating only between the initial solutions (i.e., random uniform permutation and RZ simplified heuristic) and the instance size (i.e., 50 and 100 jobs).

2. Determine by means of statistical tests (in this case, the Wilcoxon test), whether there is a statistically significant difference between the solutions generated by the different perturbative local search algorithms. With the results of this experimental comparison answer the questions: (i) which initial solution is preferable? (ii) which pivoting rule generates better quality solutions and which is faster? (iii) which neighborhood generates better quality solution and what computation time is required to reach local optima? Justify your answers concisely. Moreover, to answer these questions, do not simply do all possible comparisons, which would be 66 in total, but *pair* the comparisons appropriately targeting the specific question. For example, for the first question one would do six comparisons, one for each case where the algorithms differ between the initial solutions but all other features remain the same in a comparison. In practice, this means that you should compare an algorithm using uniform random permutation, first-improvement and the insert neighborhood to an algorithm that uses simplified RZ, first-improvement and the insert neighbourhood. **Note**: For applying the statistical test, the `R` statistics software can be used. `R` can be downloaded from `http://www.r-project.org/`. A short introduction to the most important commands for executing the tests was given in the introduction to the implementation exercise (see the slides of lecture).

Exercise 1.2

Implement variable neighborhood descent (VND) algorithms. For these algorithms, consider two reasonable orderings of the neighborhood relations:

- transpose, exchange, insert
- transpose, insert, exchange

Implement the VND algorithms only for the iterative first-improvement algorithms. Consider as starting solutions the ones obtained by the simplified RZ heuristic.

a) Compute the following statistics for each combination of algorithms and generation of initial candidate solution and for each number of jobs.

- average percentage deviation from the best known solutions;
- average computation time for each number of jobs;
- percentage improvement over the usage of a single neighborhood, in particular, the exchange and the insert one.

b) Apply again the statistical tests to compare the solution quality reached by the two VND algorithms.

c) With the results of the experiments quantify the improvement of the VND over a local search in a single neighborhood and examine which ordering of the neighborhoods in the VND is preferable. Justify your answers concisely.

Additional information:

- In order to pass the exam, you have to successfully complete this 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, and results of statistical tests), interprets concisely the observed results, and answers the questions posed above;
  (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 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.
- The algorithms implemented in this exercise will be re-used in the second implementation exercise.

1 Bibliography

References

[1] Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.