Heuristic Optimization: implementation exercise 1

Samuel Buchet

March 2017

1 PFSP problem

This first implementation exercise consists in working on the Permutation Flow-shop Scheduling Problem with two heuristic methods : iterative improvement and variable neighborhood descent.

2 Iterative improvement heuristic

2.1 Implementation details

The implementation has been done in c++ from the sample code. One improvement function has been written for each neighborhood. The improvement functions take a parameter that decides the pivoting rule (first or best improvement). For the first improvement rule, after a new improvement, the algorithm doesn't restart the search from the first neighbour, as seen in class.

One of the main difficulty of the implmentation was the computation of the cost of a solution (weighted completion times). The easiest way is to compute the entire completion time, starting from the first job. However, it takes a lot of time and it is not always necessary. For example, if only the last two jobs are modified in the solution, only the last two completion times need to be re-computed. To recompute only a part of the solution cost, it is necessary to keep the completion times of each job for each machine. These times are saved in a matrix in the *PfspInstance* class. Thanks to this matrix, two functions of the class PfspInstance are available to compute the completion time, one that computes the the total weighted sum and modify the matrix, and another which compute the sum without modifying the matrix.

These two functions are used in the iterative improvement procedure. One iterative improvement function have been written for each neighborhood. The procedures iterate over all neighbours. If the pivoting rule is the best improvement rule, the best neighbor is saved and the change is done at the end of the loop. If the pivoting rule is the first improvement rule, each time an improving neighbor is found, the change is done and the search continue with the new solution.

The transpose improvement procedure is the easiest in the sense that there is no nested loop. In this procedure, all the possible transpose moves are tried on the solution. As explained below, it is not necessary to recompute the wighted sum from the first job if the transpose move is not done at the begining of the solution. However, to do so, the matrix of the completion times of the *initial* solution has to be kept. Thats why the procedure used to compute the weighted sum of a neighbor doesn't modify the matrix. If the pivoting rule is the first improvement rule and the neighbor improve the solution, the cost is re-computed in order to apply the changes on the completion times matrix.

The exchange improvement procedure is very similar. The only difference is the nested loop. It is then obvious that this neighborhood is longer to compute than the transpose one. The insert procedure is also similar. However, the insertion is properly done only once. For a given element, the procedure insert it in the first position and the other positions are tried with transpose moves, wich is equivalent. However, the procedure try the original position of the element and nothing has been done to prevent this case.

All these improvement procedures return a boolean that indicates wether an improving solution has been found or not. This boolean is used in the iterative improvement procedure to stop the algorithm. The iterative improvement procedure also init the solution with the random permutation or the simplified rz heuristic.

2.2 Results

The following table report the average of the computation times and the average of the percentage deviation from the best known solution for each algorithm. bestImp means that the pivoting rule is the best improvement rule and firstImp corresponds to the first improvement rule. random means that the solution is initialized with a random permutation and srz means that the simplified rz heuristic has been used. The last parameter is the neighborhood used in the algorithm.

algorithm	average computation time	average percentage deviation
best imp./random/exchange	919.706	4.383
best imp./random/insert	2721.295	3.242
best imp./random/transpose	12.753	36.135
best imp./srz/exchange	239.501	3.157
best imp./srz/insert	872.557	2.231
best imp./srz/transpose	7.757	4.304
first imp./random/exchange	93.031	3.252
first imp./random/insert	292.007	2.676
first imp./random/transpose	1.963	36.163
first imp./srz/exchange	80.713	3.153
first imp./srz/insert	215.415	2.185
first imp./srz/transpose	7.216	4.306

We can see that some algorithms seem to be much worst. Indeed, the average percentage deviation is quite high for the algorithms using random and transpose parameters. We can also notice that the algorithms using the transpose neighborhood are much faster. The algorithms with the insert neighborhood are also the slowest.

2.3 Statistical tests

The following statistical tests have been done using R. The statistical test used is the Wilcoxon test with a significance level equal to 0.05.

2.3.1 Initial solution

The following table shows the result of the test applied to all pairs of algorithms which differ only between the initial solution.

algorithms compared	p-value	null hypothesis
best imp. ; exchange	6.173616e-09	rejected
best imp. ; insert	6.35434e-07	rejected
best imp. ; transpose	1.671329e-11	rejected
first imp. ; exchange	0.5339054	not rejected
first imp. ; insert	0.0005783928	rejected
first imp. ; transpose	1.671329e-11	rejected

For most of the tests, the null hypothesis is rejected, which means that there is a statistically significant difference between the solution quality. Moreover, with the srz heuristic, the average of the percentage deviation is lower for each algorithm. We can then conclude that the simplified rz heuristic gives better results than the random permutation.

2.3.2 Pivoting rule

In this section, the test has been applied to all pairs of algorithms which differ only between the pivoting rule. This first table contains the results of the tests applied to the average of the percentage deviation.

algorithms compared	p-value	null hypothesis
random ; exchange	6.147428e-08	rejected
random ; insert	0.001416562	rejected
random ; transpose	0.7100708	not rejected
srz ; exchange	0.6481444	not rejected
srz ; insert	0.463876	not rejected
srz ; transpose	0.9878667	not rejected

We can see that there is a statistically significant difference only with two algorithms (which don't use the best initial solution). Moreover, for these two pairs of algorithms, the first improvement rule seems to give better results.

The table below contains the result od the statistical tests applied to the computation times.

algorithms compared	p-value	null hypothesis
random ; exchange	1.671329e-11	rejected
random ; insert	1.671329e-11	rejected
random ; transpose	1.671329e-11	rejected
srz ; exchange	5.278219e-11	rejected
srz ; insert	1.671329e-11	rejected
srz ; transpose	3.163469e-06	rejected

For each pair of algorithms, there is a statistically significant difference beteween the computions times. The average of the computation times is lower for the first improvement algorithm for each pairs. We can conclude that the first improvement algorithms are faster and there is no big differences between the quality of the solutions.

2.3.3 Neighborhood

In this section, the different neighborhood are compared. Comparisons are done for each pair of neghborhood (transpose/exchange, transpose/insert, exchange/insert). The p-value returned by the test is indicated in the tables below.

Comparison between the transpose neighborhood and the exchange neighborhood :

algorithms compared	relative percentage deviation	computation time
first imp. ; random	1.671329e-11	1.671329e-11
first imp. ; srz	5.282725e-11	1.671329e-11
best imp. ; random	1.671329e-11	1.671329e-11
best imp. ; srz	1.138656e-10	1.671329e-11

Comparison between the transpose neighborhood and the insert neighborhood :

algorithms compared	relative percentage deviation	computation time
first imp. ; random	1.671329e-11	1.671329e-11
first imp. ; srz	1.671329e-11	1.671329e-11
best imp. ; random	1.671329e-11	1.671329e-11
best imp. ; srz	1.671329e-11	1.671329e-11

Comparison between the exchange and the insert neighborhood :

algorithms compared	relative percentage deviation	computation time
first imp. ; random	0.0006803447	1.671329e-11
first imp. ; srz	2.115717e-09	1.671329e-11
best imp. ; random	1.345635e-06	1.671329e-11
best imp. ; srz	7.033074e-10	1.671329e-11

We can see that for each comparison, the null hypothesis is rejected. There is a statistically significant difference between the neighborhood in terms of quality and computation time.

For each pairs of algorithms, we can see on the table in 2.2 that in average the transpose neighborhood is faster but the quality of the solution found by the exchange neighborhood is better.

If we compare the transpose and the insert neighborhoods, we can see that once again, the transpose neighborhood is the faster one but its quality is worst than the insert neighborhood in average.

The comparison of average values between the exchange neighborood and the insert neighborhood indicates that the insert neighborhood is slower than the exchange neighborhood but the quality of its solution is better.

To conclude, the insert neighborhood is the one that generates the best solution in terms of quality. However, the transpose is the faster neighborhood. The exchange neighbohood is a good tradeoff because it is faster than the insert one and its quality is better than the transpose one.

3 Variable Neighborhood Descent heuristic

3.1 Implementation details

The implementation of the VND algorithm mainly uses the procedures created for the iterative improvement algorithm. The algorithm has been implmented in the *VNDHeuristic* procedure. It first init the solution with the chosen initialization algorithm. Then, it applies successively the different iterative improvement procedures following the chosen order of the neighbohoods. This step is reapeted until there is no improvement in the last neighborood.

3.2 Results

instances,avg ctime,avg rdp

algorithm	average compution time	avg percentage deviation	avg imp exchange	avg imp. insert
random ; ex. first	382.526	2.634	0.006	0.0003
random ; ins. first	307.123	2.091	0.011	0.006
srz ; ex. first	275.016	2.129	0.010	0.0005
srz ; ins. first	283.463	2.161	0.010	0.0002

4 Compilation and execution