ECOLE
POLYTECHNIQUE
DE BRUXELLES

MA1 - IRIF

HEURISTIC OPTIMIZATION

IMPLEMENTATION EXERCISE 2

---

https://github.com/iQuad427/PSPF_WT

Author

Quentin ROELS

Course

INFO-H413

Professor

Thomas STUTZLE

Christian CAMACHO VILLALON

Academic Year

2022-2023

Abstract

This report focus on solving the Permutation Flow-shop Scheduling Problem with Weighted Tardiness (PFSP-WT) using simple Iterative Improvement algorithms and comparing their efficiency and solution quality. The second part examine the results of two Variable Neighbourhood Descent algorithms and compare them with the previous results of II algorithms. Comparisons where made using the paired Wilcoxon test to show the statistical difference between the solutions of those algorithms.

Experimental results have shown that for single neighbourhoods algorithms solution quality mainly depends on the neighbourhood, the exchange one being the best one.

For the multiple neighbourhood algorithms tested, they were unexpectedly less good than expected, seemingly due to the chosen order of neighbourhoods causing the state to rapidly find a local optima far away from the best known-solution.

Table of contents

1 Introduction

1.1 Introduction

The Permutation Flow-shop Scheduling Problem (PFSP) is an active research subject with already many results. Many variants of this Operations Research classical problem exist, such as flow-time minimisation or total tardiness minimisation. One possible explanation of this popularity might be the direct application to the manufacturing industry and thus the underlying will to optimise the production lines. This problem being NP-Hard, there are no exact algorithms to solve it efficiently, allowing for continuous optimisation using new algorithms. Furthermore, due to the amount of existing results, it also makes it easy to compare the efficiency of different algorithms and the quality of the solution they provide.

1.2 Specifications

Focused on the Weighted Tardiness variant of the PFSP, the report investigate some of the simplest possible algorithm to optimise this kind of problem, i.e. iterative improvements algorithms. As stated earlier, the amount of previous results will allow for better comparison of the different algorithms solution. The paired Wilcoxon test has been used to show the statistical difference between the results using their deviation from the current best known solutions for given instances.

1.3 Objectives

This implementation exercise goal is to implement two SLS algorithms and compare them statistically using the paired Wilcoxon test to determine if one is better than the other. It is also ask to provide a correlation graph and eventually a Run-Time distribution of those algorithm on some given instances.

1.4 Method

As stated earlier, to show statistically the difference between the results of two algorithms, the paired Wilcoxon test is applied. All statistical tests are based on a null hypothesis that will either be rejected or not. For the Wilcoxon test, the null hypothesis is that "the median of the differences [between the two datasets] is zero". Thus, in this case, the test shows whether there is a significant difference between the results.

If the hypothesis is correctly rejected, it means that there is a statistical difference. This is why the test result in a p-value which indicates the probability of incorrectly rejecting the null hypothesis. This probability must be as low as possible to show that there is a difference.

Typically, a significance level α is chosen (a maximum allowable value) to compare the p-value with (here $\alpha = 0.05$); if the p-value is under this value, it means that this probability of falsely rejecting the null hypothesis is accepted.

In short, if a p-value is under 0.05 in this report, it means that, statistically, there is a significant difference between the values that are compared.

2 Algorithms

This section provide a clear description of the two implemented SLS algorithm, i.e. a Tabu Search (TS) and a Memetic Algorithm (MA). The goal is to explain how they work, what are their components and discuss the choice of the parameters as well as the choice of the main components such as the initialisation and iterative improvement algorithm.

2.1 Tabu Search

For the first SLS method, a Tabu Search was implemented. The Tabu Search is a "Simple" SLS Method that aims at increasing the solution quality by effectively escaping from local, unexpectedly to find better local minima or much better, the global optima.

2.1.1 Description

The Tabu Search is based on keeping some sort of memory during the algorithm execution. In this case, a given solution is marked as tabu status when chosen. Those states (here solutions) with a tabu status are then forbidden for a certain amount of time, called the tabu tenure. For the rest of the iteration, an admissible neighbourhood is determined, and the best-improving candidate is chosen for the next step and marked as tabu.

Tabu Tenure

The tabu tenure is the most important parameter of the tabu search. It has an impact on how rapidly a given solution can be visited again. If it is too high, the search steps are too restricted and the search is ineffective. If it is too low, the search steps does not allow to escape local minima, and the search stagnates.

Implementation

In this implementation, the tabu tenure is a count down, at each step, we had the new candidate found to the list of "visited solutions" with an integer equivalent to the tabu tenure. Then all the other candidate in the list of previously visited solutions see their tabu tenure decrease by one. All of those having their tabu tenure dropping to zero are removed from the list, and thus no more forbidden.

2.1.2 Components

Iterative Improvement

In the tabu search, the iterative improvement used is by construction the best improvement since we chose "the best improving neighbour of the non-tabu neighbourhood". However, we take the best one of the neighbourhood, which does not mean that it will improve the current solution, only that it will be the best out of those present. It is thus more of a Best Neighbour than a Best Improvement step.

Initialisation

For the initialisation of the algorithm, a choice has to be made between a simple RZ greedy heuristic or a random initialisation. When one (srz) typically gives better starting point, closer to a good local minima, the other (random) make good use of the stochastic nature of the algorithm when launched multiple times. In this case, since the algorithm is run 5 times per instance, it takes better advantage of the random initialisation than the simple RZ heuristic, which would make the starting state deterministic.

2.1.3 Parameters

Tabu Tenure

The tabu tenure being the most import parameter of the Tabu Search, one should give the utmost attention when deciding on its value. When studying the behaviour of the algorithm on different tabu tenure, the best value showed to be extremely instance dependent. While really low tabu tenure had good results for some instances, it has the opposite effect on other ones. Showing how much of a difficulty it is to decide on a certain tabu tenure.

To decide on this parameter, multiple short runs of 10 seconds were launched with different tabu tenure and the same random seed to see how well each tenure perform on each instance. With the final goal to either determine the best one or a range of possible tenure for a "Robust Tabu Search" implementation (where the tabu tenure is chosen at random in a given interval).

Tenure	Deviation - 50	Deviation - 100	Deviation - total
0	-26.7961659639938	-14.816584031242	-20.8063749976179
3	-32.7290047046137	-15.331706843973	-24.0303557742934
5	-34.606043258001	-15.4910710080547	-25.0485571330278
7	-33.8723851448887	-15.1608144723296	-24.5165998086092
10	-29.1201163812212	-15.6431680653801	-22.3816422233007
20	-34.2278025070033	-16.5772124786501	-25.4025074928267
30	-32.3342336369326	-16.1828537680901	-24.2585437025114
40	-33.537060594633	-15.164988265722	-24.3510244301775
50	-35.5465232804056	-15.8098906438052	-25.6782069621054
60	-34.7277256776625	-16.0400789638668	-25.3839023207647
70	-33.1534683992162	-15.9502193381691	-24.5518438686927
80	-30.882198563658	-17.0397054835077	-23.9609520235829
90	-32.6788029342852	-15.7153135251517	-24.1970582297185
100	-37.1256548039155	-17.01126637617	-27.0684605900427
110	-37.3119072734863	-15.9131246161303	-26.6125159448083
120	-33.7205004173247	-16.6330633559888	-25.1767818866567
130	-37.3092095959711	-15.7270401844709	-26.518124890221
140	-34.4790270482156	-15.0408604658147	-24.7599437570151
150	-33.7330048074581	-15.5317195668581	-24.6323621871581
200	-32.9034946891771	-14.4497996793472	-23.6766471842622

Table 2.1: Deviation from best-known solution w.r.t. Tabu Tenure and Instance Size

Note : the best-known values being out-dated, a negative deviation implies a better solution than what was previously found. The more its negative, the better it is.

The previous experiment shows that best tenures with regard to the instance size are; 110 for instances of size 50, 80 for instances of size 100 and 100 overall. The best tenure overall being approximately the same as for the two instances size, a tabu tenure of 100 was used.

Maximum Run Time

For the maximum run time, which is in this case the termination criterion, it was asked to take a sufficient time for the algorithm to do its magic. To avoid creating losses due to the algorithm running for too much time, the Tabu Search returns the best solution found during the whole execution. We then take a way too long period of time to be sure that the algorithm has done what it could.

2.2 Memetic Algorithm

For the second SLS method, a Memetic Algorithm was implemented. The Memetic Algorithm is Population-based SLS Method, which means that it uses a "population" of candidate solution instead of one candidate at a time for the Tabu Search (and other SLS methods). This simple extension allow for way better search diversification.

2.2.1 Description

The Memetic Algorithm is built on applying iteratively genetic operators such as mutation, recombination and selection to mimic the principles that take place in evolution. The Memetic Algorithm differs from the simple Evolutionary Algorithm in the way that it apply a subsidiary local search in between each population modification to allow for better search intensification than its counter-part.

2.2.2 Components

Initialisation

Being a perturbative population-based algorithm, the Memetic Algorithm requires an input population to perform perturbations on. The initialisation step mainly consists in giving a starting point to the algorithm.

Subsidiary Local Search

The subsidiary local search is the main difference of the Memetic Algortihm with the standard Genetic Algorithm. Its goal is to improve the quality of the population before each evolutionary step, with the expectation that it will lead to a better population on the next iteration. This component of the Memetic Algorithm provide an intensification processus.

Recombination

Recombination is the main source of perturbation in the Memetic Algorithm, it aims at creating new individuals based on the current population. The more the diversity in the population the more powerful is the perturbation of the recombination step. In this implementation, the recombination is a one-point crossover applied randomly to one of the best individuals (top 25%) and an other individual taken randomly in the population.

Mutation

Mutation is the second source of perturbation in the algorithm. The goal of this component is to avoid the stagnation of the algorithm in a local optimum. In fact, the evolutionary behaviour of the algorithm tend to reduce the overall diversity of the population by keeping only the best individuals. Thus the mutation is an extremely important operation to allow perturbative moves in a population lacking diversity.

In this implementation, a random exchange move is performed on each individual with a probability equal to the mutation rate.

Selection

Selection is the main component of any evolutionary algorithm. It mimics natural selection by keeping only hte best individuals of each generation to populate the next generation. Only those who have been selected will have a chance to reproduce (by recombination), so only the best solutions will give their intrinsic quality to the next generations.

In this implementation, the N best individuals (taken from all those generated at the current generation and those from the previous generation) are kept as the population for the next iteration, N being the population size.

2.2.3 Parameters

Initialisation

In this case, using a deterministic constructive heuristic causes approximately zero genetic diversity, which is counter-productive. Thus, the initialisation is the aggregation of N individuals generated randomly, N being the population size (discussed later in this section).

Subsidiary Search

That parameter choice depend on the conclusions of the previous implementation exercise. To limit the execution time and optimise the solution quality, an Iterative Improvement SLS algorithm have been used with first improvement. The neighbourhood was chosen by running the memetic algorithm on each of the three possibilities and chosing the neighbourhood that returned the best solutions.

Instance	exchange	insert	transpose
DD_Ta051	27668	31348	45971
DD_Ta052	25865	28880	54842
DD_Ta053	33915	38577	56681
DD_Ta054	33019	37701	42873
DD_Ta055	21908	26477	46222
DD_Ta056	16992	19089	35823
DD_Ta057	25001	27197	48494
DD_Ta058	22762	25803	52073
DD_Ta059	28534	31984	47328
DD_Ta060	64993	69068	80787

Table 2.2: Best Solutions found for each instance w.r.t. Neighbourhood

This experiment clearly shows the superiority of the exchange neighbourhood. Added to that, the time required to run the subsidiary local search with

Population Size

Due to the random initialisation, the population size is the primary source of genetic diversification. The bigger the population, the more we have material genetic at our disposal to create new individuals, but also, the bigger the computation overhead for the genetic operators. Even more in our case, with the subsidiary local search taking close to 5 seconds per individual for large instances. For that reason, preliminary experiments made for the memetic algorithm are only based on instances of small sizes to rapidly obtain relatively good parameters.

Population Size	Deviation - 50
10	-48.0055087320058
20	-48.0061024453452
50	-46.8773991842017
100	-48.1425358565567
200	-48.2107069355664

Table 2.3: Deviation from best-known solution w.r.t. Population Size

From this experiment we can see that, the bigger the population, the best is the result. This parameter will thus be chosen such that the computation overhead is reasonable.

Mutation Rate

The mutation rate is the secondary source of genetic diversification. To determine the optimal mutation rate, 11 experiments were launched for different values of mutation rate (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0). The experiment results can be found in the Table 3.2, from those we can see that the best mutation rate is at 0.7.

Mutation Rate	Deviation - 50
0	-46.8773991842017
0.1	-48.2530493600743
0.2	-48.1735998941202
0.3	-48.3040057124962
0.4	-47.4924660902126
0.5	-47.7752725259649
0.6	-48.3927223006719
0.7	-48.5791724414578
0.8	-48.3354412219973
0.9	-47.8503316696052
1	-48.0753397212418

Table 2.4: Deviation from best-known solution w.r.t. Mutation Rate and Instance Size

Maximum Run Time

Working on populations and using subsidiary local search is an explosive combo for large overheads in term of time to optimise each individual after each genetic operator. The biggest time dependency is right after the random initialisation where the biggest work has to be made on the candidate solutions. To allow the algorithm to start working on the genetic part we thus need to give him enough time after the initialisation. To do so, we start counting the maximum time of execution after the initialisation and first subsidiary local search, allowing for better estimation of the time spent in the genetic evolution. However, the time computed and stored in each file is the time to complete the whole algorithm, which let us compute the time spent for the initialisation by subtracting the maximum time given to the genetic evolution.

3 Comparison

3.1 Results

Instances	Size	Deviation - Tabu	Deviation - Memetic
DD_Ta051	50	-36,3040	-48,3032
DD_Ta052	50	-40,5857	-56,1668
DD_Ta053	50	-29,3206	-46,1303
DD_Ta054	50	-33,6242	-45,5343
DD_Ta055	50	-39,5028	-50,4020
DD_Ta056	50	-54,4281	-65,8857
DD_Ta057	50	-31,8540	-46,2175
DD_Ta058	50	-32,1895	-54,3474
DD_Ta059	50	-26,4889	-45,3543
DD_Ta060	50	-15,0299	-28,1274
DD_Ta081	100	-18,9194	-27,8165
DD_Ta082	100	-16,7808	-27,6660
DD_Ta083	100	-16,3841	-25,2304
DD_Ta084	100	-17,2850	-25,9190
DD_Ta085	100	-16,8373	-26,2530
DD_Ta086	100	-15,6300	-28,2569
DD_Ta087	100	-15,4816	-28,2398
DD_Ta088	100	-11,8666	-24,4940
DD_Ta089	100	-15,7166	-24,1147
DD_Ta090	100	-16,2177	-25,0102

Table 3.1: Deviation from best-known solution w.r.t. the Instance

3.2 Statistical Tests

Size	Tabu Search	Memetic Algorithm
50	-33,9328	-48,6469
100	-16,1119	-26,3001

Table 3.2: Deviation from best-known solution w.r.t. the Instance

The p-value returned by the paired Wilcoxon test is : 1.907349e-06. Which corroborates the difference between the results of both algorithms. From a statistical point of view, the Memetic Algorithm is better than the Tabu Search

3.3 Correlation Plot

Here under are the correlation plot made with the results of the previous comparison to the best-known solutions. To the correlation plot are added (in black) a line which is the linear regression (using least-square) of the dots. This line is supposed to be the best possible linear fit for the given data.

The correlation plot for instances of size 50 is quite straight forward, the dots are nicely distributed on an underlying line. However, the correlation is less visible for the instances of size 100. But once plotted with all the other dots, one could argue that the distribution is indeed linear

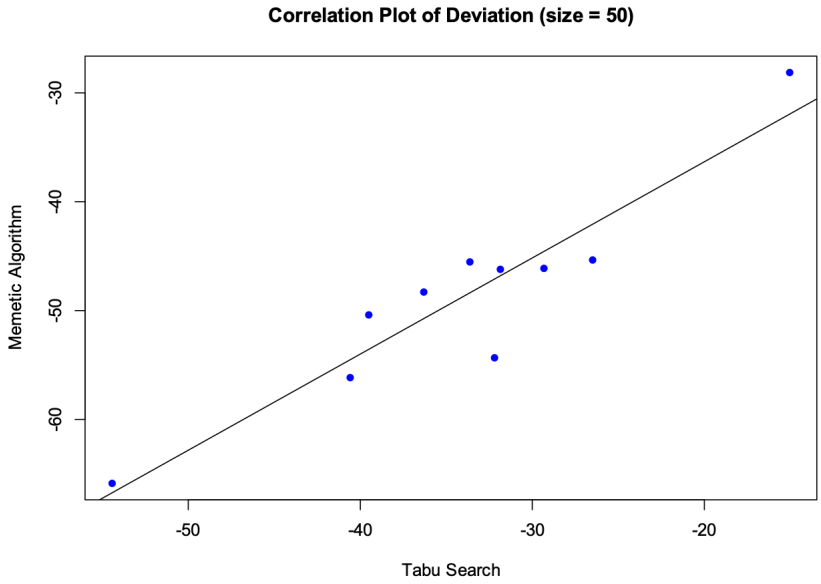

Figure 3.1: Correlation plot of the Deviation from best-known solution for instances of size 50

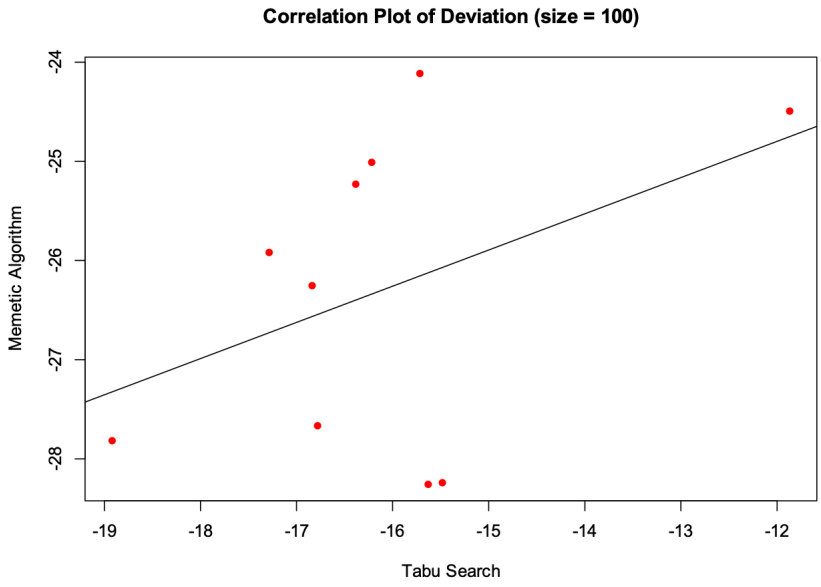

Figure 3.2: Correlation plot of the Deviation from best-known solution for instances of size 100

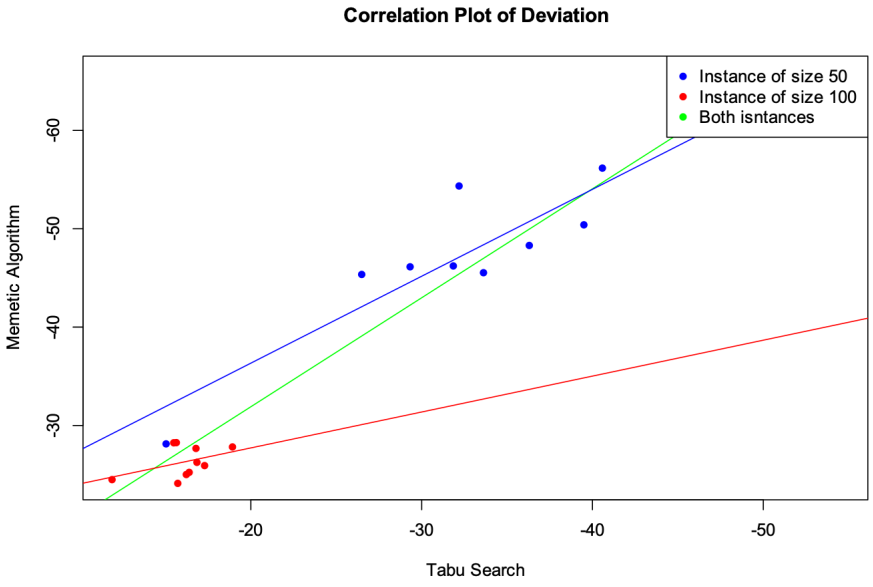

Figure 3.3: Correlation plot of the Deviation from best-known solution for both instances size

4 Conclusion

This report started by introducing two SLS techniques by explaining their components and main parameters. This introduction was followed by the identification of the best values for the parameters of each of them. Then,the two algorithms were tested with the previously found parameters to showcase their relative efficiency by comparing their output with the best-known solutions.

It turned out that the Memetic Algorithm is statistically better than the Tabu Search on the given instances and with the given configuration. Also, it was shown that there exist a correlation between the result of those two algorithms for the given instances.

It is important to note that the parameter selection was made to prevent needing to much computation time and be able to progress in the project. A more tedious study of those parameters would more than certainly lead to better results.