INFO-H413

Université Libre de Bruxelles

École Polytechnique de Bruxelles

Heuristic Optimisation Implementation Exercise 2

Author:
Jauffret Oscar

Professor:
Stützle Thomas

May 13th, 2025

Abstract

The Permutation Flowshop Scheduling problem is an NP-hard problem, and many different heuristics have been proposed to tackle it. In this study, we analyze the performance of a Tabu Search and a Memetic algorithm for the PFSP with total computation time objective. Both algorithms are executed under equal time budgets on a benchmark of set instances with varying sizes (50, 100, and 200 jobs). Performance is assessed using the deviation from best-known solutions. Algorithm parameters were automatically tuned using `irace` to ensure a fair comparison and reach maximal performance. Interestingly, for small instances, the best configuration of the Memetic algorithm excluded local search, relying instead on a high mutation rate. For larger instances, incorporating Iterative Improvement as a local search method proved beneficial. Overall, the Memetic algorithm consistently outperformed Tabu Search, reaching average deviations within 1-3% of best-known solutions, compared to 2-4% for Tabu Search. These results highlight the effectiveness of population-based strategies for solving the PFSP, especially under tight computational constraints.

Project repository available at: `https://github.com/OscarJauffret/FlowshopSLS`

Contents

1 Introduction **4**

2 Material and methods **4**
- 2.1 Algorithms implementation . . . 4
- 2.2 Tabu Search implementation . . . 4
  - 2.2.1 Tabu list . . . 5
- 2.3 Memetic algorithm implementation . . . 5
  - 2.3.1 Crossover operation . . . 5
  - 2.3.2 Mutation operation . . . 5
  - 2.3.3 Local search . . . 6
- 2.4 Experimental setup . . . 6
  - 2.4.1 Parameter tuning . . . 6

3 Results **7**
- 3.1 Performance Overview . . . 8
- 3.2 Statistical tests . . . 8
  - 3.2.1 Tabu Search vs. Memetic algorithm . . . 9
  - 3.2.2 Tabu Search vs. Variable Neighborhood Descent . . . 9
  - 3.2.3 Memetic algorithm vs. Variable Neighborhood Descent . . . 9
- 3.3 Correlation . . . 10

4 Conclusion **10**

1 Introduction

The *Permutation Flowshop Scheduling Problem* (PFSP) is one of the most studied optimization problems due to its relevance in manufacturing, logistics and production planning. It is known to be an NP-hard problem, and finding optimal solutions becomes computationally infeasible for very large instances. For this reason, heuristics have been developed to find near-optimal solutions in reasonable time.

In this paper, we study the performance of two Stochastic Local Search algorithms applied to the PFSP with the total flowtime minimization objective: a Tabu Search and a Memetic algorithm. Both algorithms are tuned using the `irace` package to ensure a fair comparison. The proposed methods are compared together and to a Variable Neighborhood Descent (VND) algorithm developed in a previous implementation exercise. The evaluation is conducted on the same instances and the termination criterion is defined by the average run-time of the VND algorithm. Performance is assessed based on average percentage deviation from best-known solutions, statistical tests, and a correlation plot. .

This study was conducted as part of a project for the *Heuristic Optimisation* course, for the second implementation exercise.

2 Material and methods

In this section, we describe how the algorithms are implemented and the experimental setup that was used to conduct the statistical tests. The results of these tests are discussed in Section 3.

2.1 Algorithms implementation

The algorithms were implemented in C++. Detailed documentation can be found in the `include/` source files, we briefly describe the overall structure here.

The two new algorithms were implemented in the same way as the previous Iterative Improvement and Variable Neighborhood Descent algorithms. We still use the `Instance` and `Solution` classes to hold data relevant to the instance, and provide methods to interact with the solution.

Two new solver classes are implemented: `FlowShopTabuSearch`, which applies Tabu Search, and `FlowShopMemetic`, which applies a memetic algorithm.

2.2 Tabu Search implementation

The Tabu Search implementation is the based on the method proposed by Tseng and Lin [1] as part of their memetic algorithm, which uses the tabu mechanism from Nowicki and Smutnicki [2].

The algorithm iteratively improves a solution until the execution time reaches a fixed time defined by the average time a run of the VND algorithm took to complete. At each iteration, the best insertion operation that is not tabu or that improves the current best solution (aspiration criterion) is selected. The neighborhood is restricted by a parameter α, which defines how far a job can be inserted from its original position. The best move found is added to the tabu list, and if it provides a better solution, it is applied to the current best solution, and a new iteration starts. Otherwise, a `stuck` counter is incremented. If no improvement occurs for more than `maxStuck` iterations, a perturbation step is performed, which replaces the current candidate solution by one of the best solutions found by the insertion operator, regardless of tabu status.

2.2.1 Tabu list

As mentioned above, the tabu mechanism follows the approach from Nowicki and Smutnicki [2]. Let π be a permutation of the jobs. In their implementation, a move (from, to) is added to the tabu list in the following way:

- If $\text{from} < \text{to}$, then add $(\pi[\text{from}], \pi[\text{from} + 1])$ to the tabu list.
- Otherwise, add $(\pi[\text{from} - 1], \pi[\text{from}])$ to the tabu list.

A move (from, to) from a permutation β has tabu status if one of the following condition is satisfied

- If $\text{from} < \text{to}$, (from, to) is tabu if $(\beta[j], \beta[\text{from}])$ is in the tabu list with $j = \text{from}+1, ..., \text{to}$.
- Otherwise, (from, to) is tabu if $(\beta[\text{from}], \beta[j])$ is in the tabu list with $j = \text{to}, ..., \text{from} - 1$.

2.3 Memetic algorithm implementation

The implemented Memetic Algorithm is also based on the work of Tseng and Lin [1]. It combines evolutionary operators with the previously implemented Tabu search, or any other local search. This design enables a direct comparison between the standalone Tabu Search and the Memetic approach, and highlights the potential benefits of incorporating population-based strategies.

2.3.1 Crossover operation

The crossover operation splits each parent into N segments, and attempts to recombine in a way to find the best offspring. However, to avoid evaluating all 2^N combinations, an orthogonal array (OA) $L_{N+1}(2^N)$ is used. These orthogonal arrays are constructed using the method introduced by Plackett and Burman [3].

Each row in the OA indicates, for every segment, whether it should be taken from parent 1 or parent 2[1]. The quality of each combination is evaluated based on the main effects of segments, and a `TaguchiChild` is constructed accordingly. Duplicate jobs in the children are removed and replaced by missing jobs to ensure a valid permutation. The best child is chosen as the offspring.

2.3.2 Mutation operation

The mutation operation also relies on orthogonal arrays. It consists of multiple generations, which start by randomly selecting several insertions, and choosing the best combination using an orthogonal array (in the same way as with the crossover operation). If the mutated solution is better than the current one, it becomes the base for a new generation. Otherwise, a stagnation counter is incremented. If the mutation operation remains stuck for `maxStuck` generations, but

[1]If a row is defined as $\text{row}_i = a_1, a_2, ..., a_j, ..., a_N$, $a_j = 0$ means it should take segment j from parent 1, and $a_j = 1$ means it should take it from parent 2

the mutated solution is better than a `replaceRate` factor times the current solution, then the current solution is replaced.

2.3.3 Local search

As discussed earlier, this algorithm can use any solver as its local search. For Tabu Serach, all parameters were set according to the values recommended by Tseng and Lin in their original study. For Iterative Improvement and Variable Neighborhood Descent, the parameters were set according to the best parameters found in the previous implementation exercise.

2.4 Experimental setup

Both SLS algorithms were run under time constraints proportional to the average execution time of the Variable Neighborhood Descent algorithm. Specifically, for instances with 50 or 100 jobs, the time limit was set to 500 $\times$ avgVNDTime, and for instances with 200 jobs, the time limit was reduced to 100 $\times$ avgVNDTime to avoid excessively long run times. The resulting execution times per instance size are shown in Table 1.

Instance size	**Allowed Execution Time (s)**
50	47.972
100	630.062
200	1916.883

Table 1: Allowed execution time per instance size

All experiments were run on an 11th Gen Intel Core i7-11700k CPU operating at 3.6 GHz.

2.4.1 Parameter tuning

To ensure a fair comparison between the algorithms and to optimize their performance, the `irace` configuration tool was used. Unlike the first implementation exercise, where all parameter combinations were tested, the current setup involved tuning over larger integer-valued spaces, justifying the use of an automated approach. Additionally, both algorithms were initialized with the simplified RZ heuristic that was proposed in the first implementation.

Tabu Search: For the Tabu Search, the tuned parameters were the insertion distance restriction α, the tabu tenure and the stagnation threshold `maxStuck`. The tuning was conducted separately for each instance size, and the `maxExperiments` parameter of `irace` was set to 3000 for instances with 50 jobs, 400 for 100 jobs, and 100 for 200 jobs.

The domain for each parameter was chosen around the values proposed in the paper.

- For instances with 50 jobs, the tabu tenure was chosen between 5 and 10, α between 0 and 50, and `maxStuck` between 3 and 20.
- For instances with 100 jobs, the parameter domains were slightly larger. The tabu tenure was chosen between 5 and 20, α between 0 and 100, and `maxStuck` between 3 and 30.
- For instances with 200 jobs, the domain were again slightly larger. The tabu tenure was chosen between 5 and 40, α between 0 and 200, and `maxStuck` between 3 and 50.

The best-performing parameters found by `irace` are presented in Table 2.

Instance size	**Tabu tenure**	α	**maxStuck**
50	8	24	14
100	18	68	30
200	7	130	41

Table 2: Best parameters per instance size for the Tabu Search according to `irace`

Memetic Algorithm: For the Memetic Algorithm, the tuned parameters included the population size, the mutation rate, and the choice of local search method. The local search options evaluated were Iterative Improvement (II), Variable Neighborhood Descent (VND), Tabu Search (TS), and no local search (i.e., a standard genetic algorithm).

The parameter settings for II and VND correspond to those that yielded to the best results in the first implementation exercise. For Tabu Search, the parameters were set to the ones described in the paper. Tuning was performed in the same way as for the Tabu Search algorithm, using the same values for `maxExperiments`.

The domain of each parameter was chosen around the value proposed by the paper from Tseng and Lin. The population size was allowed to vary from 5 to 300 for instances with 50 or 100 jobs, and up to 500 for instances with 200 jobs. The mutation rate was not constrained and could vary from 0 to 1, and the local search method was selected from the four predefined options (TS, II, VND, or none).

The best parameters found by `irace` are shown in Table 3. All `irace` logs can be found in the `results/` directory of the repository.

Instance size	**Population size**	**Mutation rate**	**Local Search**
50	272	0.8836	None
100	263	0.6916	None
200	299	0.2216	Iterative Improvement

Table 3: Best parameters per instance size for the Memetic Algorithm according to `irace`

Interestingly, the best-performing configurations for instances with 50 and 100 jobs did not include any local search method. This could be due to the time-constrained setting. Omitting the local search allows the algorithm to perform more generations, which may be more beneficial than investing significant computation time in local searches. Additionally, a high mutation rate was selected in these configurations, likely to compensate for the absence of local improvement by enhancing population diversity and exploration.

However, for instances with 200 jobs, the best configuration included Iterative Improvement as the local search method. This suggests that with a longer time limit, dedicating resources to local optimization becomes worthwhile. In this case, the mutation rate is noticeably lower, indicating that local search can partially take over the role of diversification otherwise provided by mutation.

Without the local search method, the memetic algorithm behaves as a standard genetic algorithm.

3 Results

In this section, we present and analyze the results obtained by algorithms. We compare Tabu Search to the Memetic Algorithm, and include the best performing VND algorithm for reference.

Since parameter tuning was already performed using `irace`, a single paired test per instance size is necessary to determine the best of the two SLS algorithms.

For all tests, a significance level (α) of 0.05 was used. In our case, using another α such as 0.01 would not change anything about the conclusion[2]. Also, the statistical tests were performed on the percentage deviation from best-known solutions.

3.1 Performance Overview

We begin by providing an overview of the distribution of solution quality across instance sizes, comparing Tabu Search, the Memetic algorithm and the VND. Figure 1 offers a visual summary of the performance landscape before delving into paired tests.

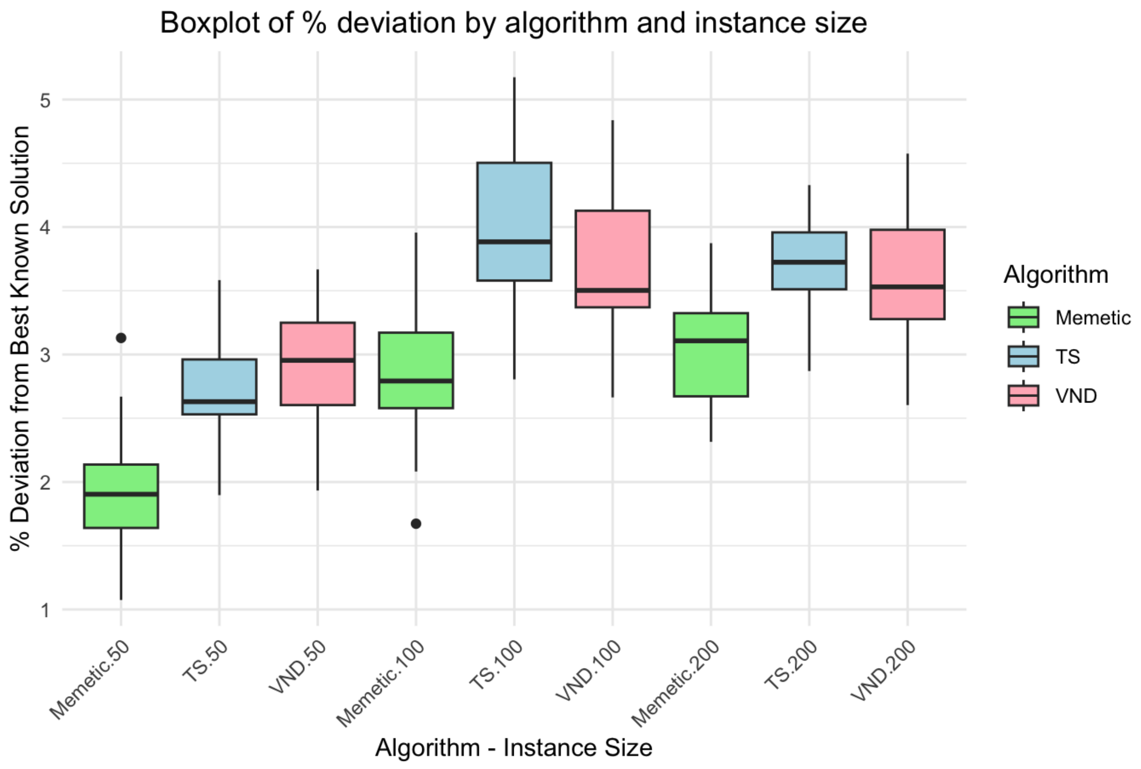

Figure 1: Boxplot of percentage deviation from best-known solutions by algorithm and instance size

From the figure, the Memetic algorithm seems better than both Tabu Search and VND across all instance sizes. Additionally, Tabu Search's performance is relatively similar to VND. These observations will be further analyzed and confirmed in the following statistical tests.

3.2 Statistical tests

In this section, we statistically compare the algorithms using Wilcoxon and t-tests, performed separately for each instance size. For clarity, when both tests agree, the results of the t-test are not shown in the tables. Full detailed results, including R scripts that were used to gather the statistics, are available in the `results/` directory of the repository. In all tables, whenever the p-value indicates that there is a significant difference between the two algorithms tested, the cell corresponding to the better-performing algorithm is highlighted in gray.

[2]There were no p-values between 0.05 and 0.01.

3.2.1 Tabu Search vs. Memetic algorithm

First, we compare Tabu Search and the Memetic algorithm. The results are presented in Table 4.

Instance Size	**Tabu Search (%)**	**Memetic (%)**	**p-value$_{\textbf{Wilcoxon}}$**
50	2.736	1.899	$< 2.2 \times 10^{-16}$
100	3.938	2.855	$< 2.2 \times 10^{-16}$
200	3.686	3.061	$< 2.2 \times 10^{-16}$

Table 4: Comparison between Tabu Search and Memetic algorithm across all tested instance sizes. Values are average percentage deviations from best-known solutions.

The results show that the Memetic algorithm consistently outperforms Tabu Search, with average percentage deviations sometimes reduced by over 1 point.

This improved performance may be attributed to the Memetic algorithm's greater exploration capability, especially on smaller instances where high mutation rates and the absence of local search enabled rapid traversal of the search space. The consistent superiority suggests that the evolutionary design, even in the absence of local search, provides a substantial advantage over single-solution methods like Tabu Search.

3.2.2 Tabu Search vs. Variable Neighborhood Descent

Now, we compare Tabu Search with VND which was one of the algorithms implemented in the first exercise. The goal is to evaluate if our Tabu search implementation can perform better than an iterative improvement approach. The results are presented in Table 5.

Instance Size	**Tabu Search (%)**	**VND (%)**	**p-value$_{\textbf{Wilcoxon}}$**
50	2.736	3.010	$< 2.2 \times 10^{-16}$
100	3.938	3.492	$< 2.2 \times 10^{-16}$
200	3.686	3.526	$< 2.2 \times 10^{-16}$

Table 5: Comparison between Tabu Search and VND across all tested instance sizes. Values are average percentage deviations from best-known solutions.

Variable Neighborhood Descent still outperforms Tabu Search on instances with 100 or 200 jobs. However, on small-size instances, Tabu Search provides better solutions.

The improvement on small instances may be explained by the Tabu Search's ability to avoid and escape local optima via the tabu and the perturbation mechanisms, allowing it to explore a broader portion of the search space. Tabu search may require a longer time on larger instances, whereas VND consistently improves its solution at every step, and does so very quickly.

3.2.3 Memetic algorithm vs. Variable Neighborhood Descent

Finally, we compare the Memetic algorithm to the VND. The results are presented in Table 6.

The results clearly highlight the superiority of the Memetic algorithm over VND. Its population-based structure, combined with evolutionary operators and occasional local search, allows for both broad exploration and effective intensification. This hybrid strategy consistently outperforms the purely local search-based approach of VND.

Instance Size	**Memetic (%)**	**VND (%)**	**p-value$_{\text{Wilcoxon}}$**
50	1.899	3.010	$< 2.2 \times 10^{-16}$
100	2.855	3.492	$< 2.2 \times 10^{-16}$
200	3.061	3.526	$< 2.2 \times 10^{-16}$

Table 6: Comparison between Memetic Algorithm and VND across all tested instance sizes. Values are average percentage deviations from best-known solutions.

3.3 Correlation

To complement the statistical tests, we present a correlation plot (Figure 2) comparing the performance of Tabu Search and the Memetic algorithm on a per-instance basis. Each point represents the average percentage deviation from the best-known solution over 10 independent runs[3].

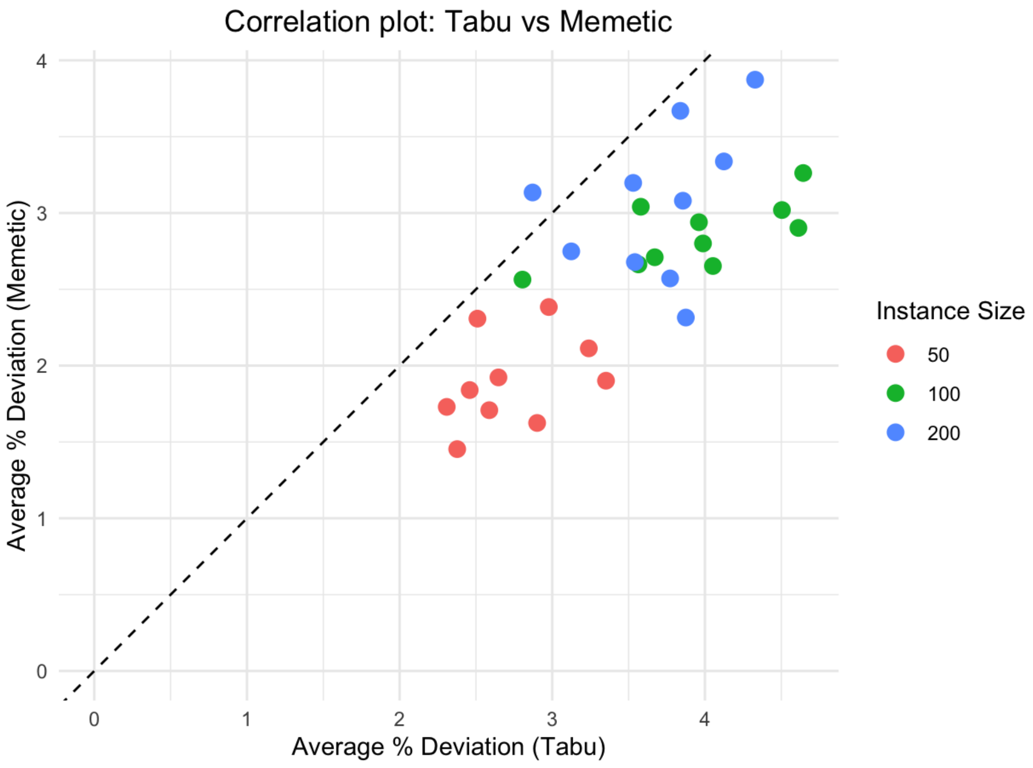

Figure 2: Correlation plot between the solution qualities of the Tabu Search and the Memetic algorithm. Each point represents an instance.

Unsurprisingly, the results mirror what was observed in the statistical tests: the Memetic algorithm outperforms Tabu search on most instances. While Tabu Search is better on one 200-job instance, the improvement over the Memetic algorithm is marginal and it does not reverse the overall trend.

4 Conclusion

In this report, we conducted an experimental analysis of a Tabu Search and a Memetic algorithm for solving the Permutation Flowshop Problem. We implemented state-of-the-art algorithms

[3]Only 3 runs for instances with 200 jobs

and tuned the parameters using `irace`, providing a fair comparison.

Between both algorithms, the Memetic was the best-performing, providing solutions within 1 to 3% of the best-known solutions.

The Tabu Search outperforms Variable Neighborhood Descent on small instances, but may require more time for larger instances.

On instance with 50 or 100 jobs, the Memetic was found to be better without any local search method, replacing it by a higher mutation rate. For instances with 200 jobs, an Iterative Improvement appeared as the best local search candidate.

Overall, this study highlights the superiority of population-based strategies for solving the Permutation Flowshop Problem in a time-constrained setting.

References

[1] L.-Y. Tseng and Y.-T. Lin, “A genetic local search algorithm for minimizing total flow-time in the permutation flowshop scheduling problem,” *International journal of production economics*, vol. 127, no. 1, pp. 121–128, 2010.

[2] E. Nowicki and C. Smutnicki, “A fast tabu search algorithm for the permutation flow-shop problem,” *European journal of operational research*, vol. 91, no. 1, pp. 160–175, 1996.

[3] R. L. Plackett and J. P. Burman, “The design of optimum multifactorial experiments,” *Biometrika*, vol. 33, no. 4, pp. 305–325, 1946.