Evaluating Memetic and Genetic Algorithms for the Permutation Flow-Shop Scheduling Problem with Total Completion Time

Théo Desoil
000583396
Université Libre de Bruxelles (ULB)
Belgium
theo.desoil@ulb.be

Abstract

This study explores the effectiveness of two stochastic local search (SLS) algorithms applied to the Permutation Flow-Shop Scheduling Problem with Total Completion Time (PFSP-CT). We investigate the performance of a classic Tabu Search method and a Memetic Algorithm based on a Genetic Algorithm enhanced with local search. Both methods are applied to the Taillard benchmark instances used in the first implementation exercise. By leveraging insights from constructive and perturbative local search methods, each algorithm is designed to balance diversification and intensification. Experimental results demonstrate that both approaches significantly improve upon traditional local search baselines. Wilcoxon signed-rank tests and performance plots are used to assess and compare solution quality across problem sizes. Our findings highlight the benefits of hybrid metaheuristics for solving complex scheduling problems.

1 Introduction

The Permutation Flow-Shop Scheduling Problem with Total Completion Time (PFSP-CT) is a classical NP-hard optimization problem that arises in many industrial settings. In this problem, a set of jobs must be scheduled on a series of machines, each job visiting all machines in the same order. The objective is to minimize the total completion time of all jobs.

In the first part of this project, we focused on iterative improvement and Variable Neighborhood Descent (VND) algorithms, identifying the best local search configurations based on different neighborhood structures, pivoting rules, and initialization strategies.

This second part extends the analysis by introducing two stochastic local search (SLS) algorithms:

- a **Tabu Search**, which integrates memory structures to avoid cycles and escape local optima, based on move prohibition (tabu list) and aspiration criteria;
- a **Memetic Algorithm**, built upon a Genetic Algorithm enhanced with local search based on the insert neighborhood and first-improvement rule.

Both algorithms are tested on the same Taillard benchmark instances as in the previous study. To ensure fair comparison, the stopping criterion for both methods is aligned with a multiple of the average runtime of the VND algorithm for each problem size. Performance is assessed via the average percentage deviation from best-known solutions, total runtime, and statistical tests (Wilcoxon signed-rank test).

The following sections detail the problem formulation (Section 2), the implementation and experimental setup (Section 3), followed by results and discussion (Sections 4 and 5), and a conclusion summarizing the key findings.

2 Problem Description

The Permutation Flow-Shop Scheduling Problem with Total Completion Time (PFSP-CT) consists of scheduling n jobs on m machines. Each job must be processed on all machines in the same order, and no job can be interrupted or processed on more than one machine at a time.

Let p_{ij} be the processing time of job J_i on machine M_j. The goal is to find a permutation π of the jobs that minimizes the total completion time:

$$\min \sum_{i=1}^{n} C_{\pi(i)}$$

where $C_{\pi(i)}$ denotes the completion time of job $\pi(i)$ on the last machine.

Taillard benchmark instances are used in this study. These instances are widely adopted in the literature and vary in size, ranging from 50 to 200 jobs and involving 20 machines.

3 Materials and Methods

3.1 Algorithms Overview

Two stochastic local search algorithms were implemented and evaluated in this study:

- **Tabu Search (TS)**: A classical Tabu Search algorithm that explores the solution space using neighborhood moves while maintaining a tabu list to avoid cycles. The insert neighborhood is used, and aspiration criteria allow the acceptance of improving solutions even if they are tabu. The search is initialized using the simplified RZ heuristic, followed by an initial local search using the insert neighborhood and the first-improvement rule.
- **Memetic Algorithm (MA)**: This method is based on a Genetic Algorithm (GA) augmented with a local search step, making it a memetic algorithm. The GA uses tournament selection, order crossover (OX), and insert mutation. Each offspring is subsequently improved using a local search based on the insert neighborhood and the first-improvement pivoting strategy.

The design choices for the initialization method, neighborhood structure, and pivoting rule are grounded in the empirical results obtained in the first implementation exercise. Specifically:

- The **simplified RZ heuristic** consistently produced better initial solutions compared to uniform random initialization.
- The **insert neighborhood** outperformed both exchange and transpose in terms of solution quality, albeit at a higher computational cost.
- Although the **best-improvement** rule yielded slightly better solutions, the **first-improvement** strategy was chosen in this study due to its significantly lower computation time. This trade-off is particularly relevant under the fixed time budget imposed for each run.

These elements were integrated into both algorithms to exploit the most promising configurations observed in earlier experiments, while respecting the runtime constraints specified for this second part.

3.2 Implementation Details

Both algorithms are implemented in C++ and reuse key components from the first implementation exercise (e.g., instance reader, local search operators, SRZ initialization). The Memetic Algorithm builds on the genetic framework but adds a local search phase to each newly generated offspring.

For the Tabu Search, the tabu tenure is fixed to 7 iterations, and the tabu list is managed as a hash map of forbidden moves. The best admissible neighbor is selected in each iteration, and the process continues until a predefined time limit is reached.

3.3 Experimental Setup

Experiments were conducted on the same 30 Taillard benchmark instances as in Part 1, grouped according to problem size: 50, 100, and 200 jobs (with 20 machines in all cases). Each algorithm was executed multiple times per instance using different random seeds to account for stochasticity.

To ensure a fair comparison, the stopping criterion was defined based on the average runtime of the VND algorithm for each instance size, multiplied by a fixed factor:

- For instances with 50 and 100 jobs, the time budget was set to **500 × VND average time**, corresponding approximately to 153 seconds and 2725 seconds respectively.
- For instances with 200 jobs, due to their significantly higher computational cost, the time budget was reduced to **100 × VND average time**, resulting in a limit of approximately 9000 seconds per run.

Regarding the number of independent executions:

- For 50- and 100-job instances, each algorithm was run **10 times per instance**.
- For 200-job instances, each algorithm was run **3 times per instance**, to maintain computational feasibility while still allowing statistical analysis.

All executions were performed using the same hardware and software environment to avoid bias from system-related variability. This setup allows a consistent comparison of algorithmic performance across problem sizes.

3.4 Evaluation Metrics

Each algorithm is evaluated based on:

- **Average Percentage Deviation (APD)** from best-known solutions, computed as:

$$\text{APD} = 100 \times \frac{C - C^*}{C^*}$$

  where C is the total completion time found, and C^* is the best-known value.
- **Total runtime**, averaged over all runs for each instance size.
- **Wilcoxon signed-rank test** is used to assess the statistical significance of differences between the two algorithms, separately for each problem size.

3.5 Parameter Tuning and Reproducibility

To ensure a fair and reproducible configuration of the Memetic Algorithm (MA), we conducted a systematic parameter tuning phase. The tuning focused on three key hyperparameters of the Genetic Algorithm (GA) component:

- Population size $\in \{20, 50, 100\}$
- Mutation rate $\in \{0.05, 0.1, 0.2, 0.4\}$
- Tournament size $\in \{2, 3, 5\}$

The tuning experiments were automated using a custom shell script (`run_tuning_genetic.sh`), which launches all configurations in parallel across a defined set of Taillard instances (`ta051`, `ta081`, `ta101`) and seeds. For each parameter combination, the script records the completion time and total runtime in a central CSV file (`tuning_genetic.csv`).

The use of controlled random seeds and fixed instance subsets ensures the reproducibility of results across executions. The script also enforces a maximum number of concurrent jobs to stabilize CPU load and avoid variability due to system noise.

Following the analysis of the resulting data (`Tuning_GA_Jobs_*.csv`), the best-performing parameters per job size were selected based on average completion time:

Table 1: Best parameter configurations selected for the Memetic Algorithm

Jobs	Tournament Size	Population Size	Mutation Rate
50	5	20	0.05
100	5	100	0.2
200	5	100	0.2

This process guarantees a rigorous and reproducible experimental setup, while adapting the algorithm's configuration to the complexity of each problem size.

3.6 Run-Time Distribution Setup

To evaluate the reliability and convergence speed of the algorithms, we measured qualified run-time distributions (RTDs) for both Tabu Search and the Memetic Algorithm. These distributions characterize how quickly each algorithm reaches solutions of targeted quality levels.

Experiments were conducted on four selected instances:

- **50 jobs**: TA51 and TA52

- **100 jobs**: TA81 and TA82

For each instance, 25 independent runs were executed using different seeds. Success was defined as reaching a solution whose cost is at most $\{0.1\%, 0.5\%, 1.0\%, 2.0\%\}$ above the best-known value. The cutoff time was reduced to 10× the average VND runtime, resulting in approximate limits of 3.1 seconds for 50-job instances and 54.5 seconds for 100-job instances.

Algorithm implementations were modified to:

- Dynamically check whether the target quality was reached;
- Log the corresponding CPU time;
- Immediately stop execution upon success to minimize runtime.

Execution was automated via a dedicated Bash script (`run_rtd.sh`), which ran multiple configurations in parallel. RTD results were collected in `rtd_results.csv`, and visualized using a Python script (`plot_rtd.py`).

3.7 Hardware and Execution Environment

All experiments were conducted on a machine equipped with an **Intel® Core™ i7-14700K** processor:

- **20 cores / 28 threads**, base frequency: 3.40 GHz,
- **32 GB of physical RAM**

The system runs **Windows 11 Pro (64-bit)** with the experiments executed under a **WSL (Windows Subsystem for Linux)** environment, ensuring compatibility with UNIX-based compilation and scripting tools.

4 Results

This section presents a detailed comparison of the two stochastic local search (SLS) algorithms applied to the PFSP-CT: the Tabu Search (TS) and the Memetic Algorithm (MA). The results are reported in terms of average percentage deviation from best-known solutions, correlation with instance size, and statistical significance tests.

4.1 Average Deviation from Best Known Solutions

Table 2 reports the average percentage deviation for each algorithm, grouped by instance size. The Memetic Algorithm consistently outperforms Tabu Search across all instance sizes, with lower deviations for small, medium, and large instances.

Table 2: Average Percentage Deviation from Best Known Solutions by Instance Size

Jobs	Memetic Algorithm (%)	Tabu Search (%)
50	0.82	2.60
100	1.74	3.34
200	2.27	3.12

4.2 Correlation with Instance Size

To visualize the correlation between the two algorithms, Figure 1 plots the average percentage deviation obtained by the Memetic Algorithm against those from the Tabu Search, for each instance. Instances are color-coded by problem size. Points below the $y = x$ line indicate cases where the Memetic Algorithm performed better.

4.3 Statistical Significance (Wilcoxon Test)

To determine whether the observed differences are statistically significant, we applied the Wilcoxon signed-rank test to the average deviation results per instance. Table 3 reports the p-values obtained per instance size. In all cases, the p-value is well below the common significance threshold $\alpha = 0.05$, confirming that the Memetic Algorithm significantly outperforms Tabu Search across all problem sizes.

Table 3: Wilcoxon Signed-Rank Test Results (Memetic vs Tabu Search)

Instance Size (Jobs)	p-value
50	0.001953
100	0.001953
200	0.001953

4.4 Run-Time Distribution Analysis

To analyze the convergence behavior of the two SLS algorithms under time constraints, we measured qualified run-time distributions (RTDs) on four representative instances: TA51 and TA52 (50 jobs), TA81 and TA82 (100 jobs). For each instance, we tested whether the algorithms could reach high-quality solutions within a restricted time budget of 10× the average VND time.

The quality targets were defined as reaching a solution within $\{0.1\%, 0.5\%, 1.0\%, 2.0\%\}$ above the best-known values. Each algorithm was executed 25 times per instance and target. A run was considered successful if it reached the target and terminated early, recording the time required.

Figure 2 shows the cumulative success rate (y-axis) over time (x-axis) for all successful configurations of the Memetic Algorithm (Tabu Search is not shown, as it never met the target in time).

The results reveal that:

- Only the **Memetic Algorithm** succeeded in reaching the 2% target on three configurations: TA51, TA52, and TA82.
- For TA51 and TA52 (50 jobs), the Memetic Algorithm succeeded in over 85% of the runs within 8 seconds.
- For TA82 (100 jobs), the algorithm only succeeded once out of 25 runs, requiring more than 50 seconds, indicating poor reliability under strict cutoffs.
- No success was observed for any other targets (0.1%, 0.5%, or 1.0%) for any instance.
- **Tabu Search failed to reach any of the target thresholds** in the allocated time for all tested instances.

These results highlight that while the Memetic Algorithm shows some promise on small instances with relaxed thresholds, none of the methods are reliable within such limited time for tighter quality targets.

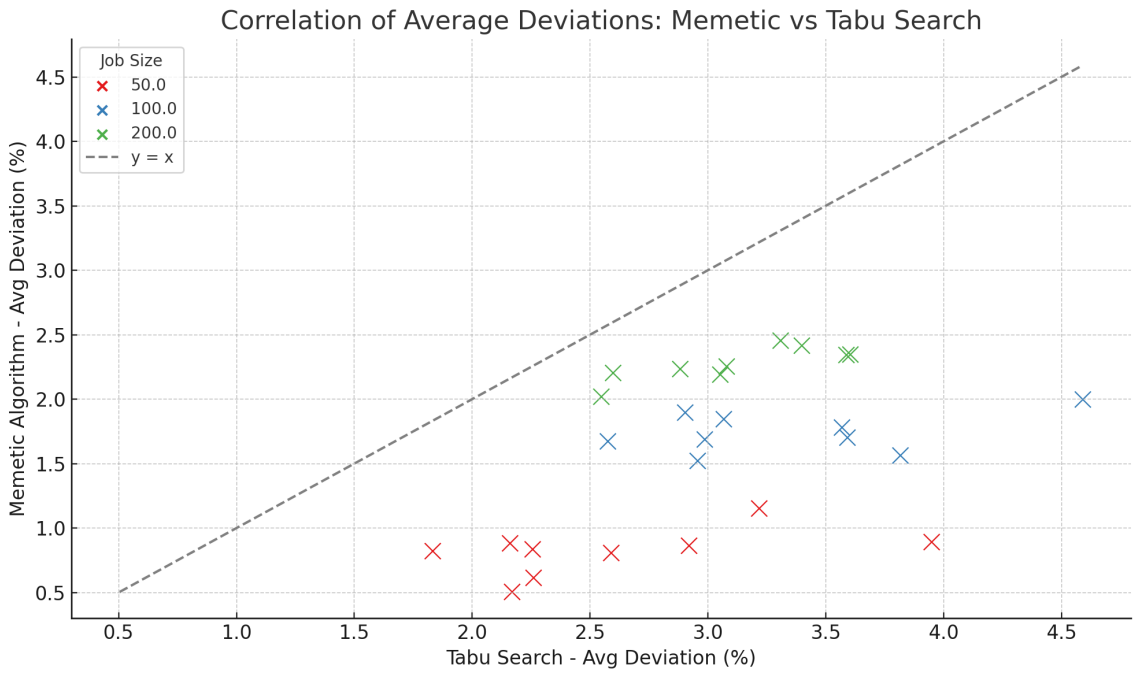

Figure 1: Correlation of average percentage deviation between Memetic Algorithm and Tabu Search, grouped by instance size.

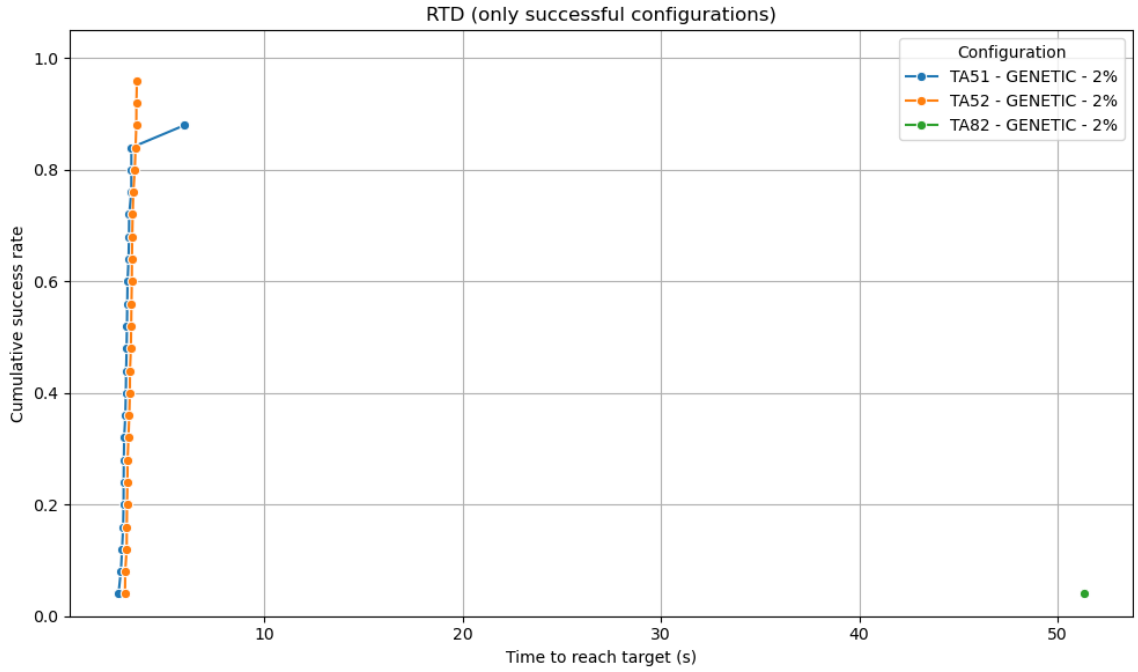

Figure 2: Run-time distribution (RTD) curves for the Memetic Algorithm reaching the 2% target.

5 Discussion

The results of this experimental study provide several insights into the relative performance of the two SLS algorithms tested on the PFSP-CT.

5.1 Memetic vs Tabu Search

The Memetic Algorithm (MA) consistently outperformed Tabu Search (TS) in terms of average percentage deviation from best-known solutions across all instance sizes. The differences were especially notable on small instances (50 jobs), where MA reached an average deviation below 1%, compared to 2.6% for TS. This confirms the benefit of integrating local search within a population-based approach, which allows the MA to combine global exploration through crossover and mutation with strong local exploitation via first-improvement search.

5.2 Scalability with Problem Size

While the performance gap remains statistically significant for larger instances (100 and 200 jobs), the difference between MA and TS becomes smaller. This may be attributed to the increasing computational complexity of local search steps on longer permutations, which affects both algorithms but reduces the relative advantage of MA. Still, the Memetic Algorithm maintains a consistent edge.

5.3 Statistical Significance and Robustness

The Wilcoxon signed-rank tests confirm that the performance improvements observed for the Memetic Algorithm are statistically significant at $\alpha = 0.05$ for all instance sizes. This highlights the robustness of the observed trend and suggests that the results are not due to random variation. The correlation plot further supports these findings: nearly all points fall below the $y = x$ line, indicating better results for MA on nearly every instance.

5.4 Run-Time Distribution (RTD) Analysis

The RTD results provide additional insights into the robustness and reliability of the two algorithms under strict time constraints. While the Memetic Algorithm managed to reach the 2% target within the time budget on small instances (TA51 and TA52), Tabu Search consistently failed to meet any target thresholds on all tested instances.

Notably, the Memetic Algorithm succeeded in over 85% of runs for 50-job instances, but only once for TA82 (100 jobs), highlighting its limited reliability as instance size increases. No success was recorded for stricter targets (0.1%, 0.5%, or 1.0%), regardless of the algorithm.

5.5 Interpretation and Recommendations

These results demonstrate that hybrid algorithms combining global and local search, such as memetic methods, can offer substantial benefits for complex scheduling problems such as PFSP-CT. However, the computational cost of such hybridization should be weighed against the runtime constraints of the application. In contexts where computation time is more restricted, the Tabu Search remains a viable and relatively effective alternative.

6 Conclusion

This study evaluated the performance of two stochastic local search (SLS) algorithms applied to the Permutation Flow-Shop Scheduling Problem with Total Completion Time (PFSP-CT): a classical Tabu Search and a Memetic Algorithm based on a Genetic Algorithm enhanced with local search.

Building on the best configurations identified in the first implementation exercise, both algorithms were designed using the insert neighborhood, simplified RZ initialization, and the first-improvement pivoting rule. This ensured that comparisons focused on the higher-level algorithmic strategies rather than low-level tuning choices.

Experimental results on 30 Taillard benchmark instances demonstrate that the Memetic Algorithm consistently outperforms Tabu Search in terms of average percentage deviation from best-known solutions. The difference is particularly marked for smaller problem sizes, and remains statistically significant across all instance groups, as confirmed by Wilcoxon signed-rank tests.

These findings highlight the value of hybridizing population-based metaheuristics with effective local search mechanisms. While Tabu Search remains a competitive baseline with lower implementation complexity, Memetic Algorithms offer superior solution quality when computation time allows.

Future work could explore the integration of adaptive mechanisms (e.g., dynamic neighborhood selection or restart strategies), as well as alternative crossover or mutation operators tailored to PFSP-specific structures. Investigating parallel or distributed versions of these algorithms would also be a promising direction to address scalability challenges on large-scale instances.

6.1 Acknowledge

The code is in the following github link : https://github.com/Teytey2002/Heuristic_Project_Part2.git