Evaluating Iterative Improvement and VND 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 investigates the performance of iterative improvement and Variable Neighborhood Descent (VND) algorithms for the Permutation Flow-Shop Scheduling Problem with Total Completion Time (PFSP-CT). We explore how pivoting rules, neighborhood structures, and initialization methods affect solution quality and computation time.

Twelve iterative algorithms are tested by combining two pivoting strategies (first- and best-improvement), three neighborhoods (transpose, exchange, insert), and two initializations (random and simplified RZ). Additionally, two VND algorithms using different neighborhood orders are implemented.

Experimental results on 30 Taillard instances show that RZ initialization leads to better starting points. First-improvement converges faster, while best-improvement reaches slightly better solutions. The insert neighborhood yields the best performance in solution quality. VND methods consistently outperform single-neighborhood searches, with the transpose-insert-exchange order performing best overall.

Wilcoxon signed-rank tests confirm the statistical significance of performance differences.

Keywords: Permutation Flow Shop, Total Completion Time, Iterative Improvement, Variable Neighborhood Descent, Heuristic Optimization

1 Introduction

The Permutation Flow-Shop Scheduling Problem (PFSP) is a classical NP-hard combinatorial optimization problem arising in manufacturing and production systems. In its total completion time (TCT) variant, the goal is to find a permutation of jobs that minimizes the sum of their completion times across multiple machines.

Due to its computational complexity, exact methods become impractical for large instances. As a result, heuristic and metaheuristic approaches are commonly employed to provide high-quality solutions in reasonable time. Among them, Iterative Improvement (II) algorithms and Variable Neighborhood Descent (VND) are simple yet effective local search strategies.

This study focuses on implementing and analyzing the performance of II and VND algorithms for PFSP-CT. We examine the impact of different pivoting rules, neighborhood structures, and initialization methods on solution quality and computational efficiency. Our goal is to identify the most effective combinations and assess whether combining multiple neighborhoods through VND can improve performance over traditional single-neighborhood searches.

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

Experimental Setup

We evaluated twelve Iterative Improvement (II) algorithms generated by combining:

- Two pivoting strategies: *first-improvement* and *best-improvement*,
- Three neighborhood operators: *transpose*, *exchange*, and *insert*,
- Two initialization methods: *random* and a simplified RZ heuristic.

Each algorithm starts from an initial solution and explores neighboring solutions iteratively until a local optimum is reached. In first-improvement, the first improving neighbor is accepted; in best-improvement, the best neighbor in the neighborhood is selected.

We also implemented two Variable Neighborhood Descent (VND) algorithms:

- **VND1**: transpose → exchange → insert,
- **VND2**: transpose → insert → exchange.

Both VND algorithms use the first-improvement strategy and are initialized using the simplified RZ heuristic.

All methods were tested on 30 Taillard benchmark instances (TA51–TA60, TA81–TA90, TA101–TA110), representing problems

with 50, 100, and 200 jobs across 20 machines. Each algorithm was executed 10 times per instance using different random seeds. Performance was evaluated based on:

- The average percentage deviation from the best-known solution,
- The total computation time (in seconds).

The Wilcoxon signed-rank test was used to assess the statistical significance of performance differences.

Implementation Details

All algorithms were implemented in **C++** for performance and executed via automated scripts. Outputs (one CSV file per configuration) were post-processed using **Python** scripts to:

- Merge and clean results,
- Compute average deviations and execution times,
- Generate summary tables for reporting.

All statistical analyses (Wilcoxon signed-rank tests) were performed using dedicated **R scripts**, applied to paired performance vectors extracted from the CSV logs.

The entire experimental pipeline, including source code, data folders, scripts, and documentation, is publicly available on GitHub. A detailed README file provides step-by-step instructions to reproduce all results. The link to the repository is provided at the end of this paper.

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

4.1 First Part

4.1.1 Iterative Improvement Algorithms

Twelve iterative improvement (II) algorithms were evaluated by combining:

- Two pivoting rules: *first-improvement* and *best-improvement*,
- Three neighborhood operators: *transpose*, *exchange*, and *insert*,
- Two initialization strategies: *random* and *simplified RZ*.

Each algorithm was executed 10 times per instance over 30 Taillard benchmarks (10 each for sizes 50, 100, and 200 jobs). Rather than presenting individual results per algorithm, Table 1 summarizes the average percentage deviation from best-known solutions and total computation time, aggregated by initialization method and problem size.

The complete raw results for all combinations of pivot rules, neighborhoods, and initializations are provided in Appendix A (Table 8).

Table 1: Aggregated performance of iterative improvement algorithms by initialization and problem size

Initialization	Jobs	Avg. Dev. (%)	Total Time (s)
Random	50	9.20	11.17
Random	100	9.52	179.77
Random	200	10.00	3535.47
SRZ	50	4.35	0.33
SRZ	100	5.00	5.95
SRZ	200	4.49	104.00

4.1.2 Impact of Initialization, Pivoting Rule, and Neighborhood

To assess the influence of various algorithmic components, we conducted **Wilcoxon signed-rank tests** on the experimental results. The following three tables summarize the statistical significance of:

- the impact of the initialization strategy (random vs. simplified RZ),
- the comparison between first- and best-improvement pivoting rules,
- and the relative performance of different neighborhood structures.

All tests are paired and performed per instance. Results show that the initialization strategy and pivoting rule both significantly affect performance in most cases. Similarly, the insert neighborhood consistently outperforms exchange and transpose.

Tables 2–4 report the associated p-values.

Table 2: Wilcoxon test: Random vs. SRZ initialization

Configuration	p-value
First + Insert	$< 2.2 \times 10^{-16}$
First + Exchange	$< 2.2 \times 10^{-16}$
First + Transpose	$< 2.2 \times 10^{-16}$
Best + Insert	$< 2.2 \times 10^{-16}$
Best + Exchange	$< 2.2 \times 10^{-16}$
Best + Transpose	$< 2.2 \times 10^{-16}$

Table 3: Wilcoxon test: First vs. Best pivoting

Configuration	p-value
Insert (Random)	$< 2.2 \times 10^{-16}$
Exchange (Random)	$< 2.2 \times 10^{-16}$
Transpose (Random)	$< 2.2 \times 10^{-16}$
Insert (SRZ)	6.29×10^{-5}
Exchange (SRZ)	2.08×10^{-5}
Transpose (SRZ)	0.0376

Table 4: Wilcoxon test: Neighborhood comparisons

Comparison	p-value
Best + Rand: Exchange vs Transpose	$< 2.2 \times 10^{-16}$
Best + Rand: Insert vs Exchange	3.58×10^{-4}
Best + Rand: Insert vs Transpose	$< 2.2 \times 10^{-16}$
Best + SRZ: Exchange vs Transpose	1.86×10^{-9}
Best + SRZ: Insert vs Exchange	1.86×10^{-9}
Best + SRZ: Insert vs Transpose	1.86×10^{-9}
First + Rand: Exchange vs Transpose	$< 2.2 \times 10^{-16}$
First + Rand: Insert vs Exchange	$< 2.2 \times 10^{-16}$
First + Rand: Insert vs Transpose	$< 2.2 \times 10^{-16}$
First + SRZ: Exchange vs Transpose	1.86×10^{-9}
First + SRZ: Insert vs Exchange	5.72×10^{-7}
First + SRZ: Insert vs Transpose	1.86×10^{-9}

4.2 Second Part

4.2.1 VND Algorithms: Summary and Comparison

We evaluated two Variable Neighborhood Descent (VND) algorithms:

- **VND1**: transpose → exchange → insert
- **VND2**: transpose → insert → exchange

Both algorithms use the first-improvement pivoting rule and are initialized with the simplified RZ heuristic. Table 5 shows the average deviation from the best-known solutions and average computation time for each number of jobs.

Table 5: Performance of VND algorithms by instance size

Algorithm	Size	Avg. Deviation (%)	Avg. Time (s)
VND1	50	2.65	1.33
	100	2.17	6.62
	200	1.58	111.73
VND2	50	2.33	1.32
	100	2.01	6.77
	200	1.59	118.65

To better understand the benefit of using multiple neighborhoods, we computed the average percentage improvement of each VND algorithm over single-neighborhood local searches using either *insert* or *exchange*.

Table 6: Average improvement of VND over single-neighborhood searches

VND Algorithm	Size	Compared to	Improvement (%)
VND1	50	exchange	29.68
	100	exchange	20.46
	200	exchange	18.67
	50	insert	0.30
	100	insert	1.08
	200	insert	1.28
VND2	50	exchange	33.08
	100	exchange	21.45
	200	exchange	18.75
	50	insert	1.52
	100	insert	1.83
	200	insert	1.31

4.2.2 Statistical Comparison of VND Algorithms

To determine whether there is a statistically significant difference between the solution quality of the two VND algorithms, we applied the Wilcoxon signed-rank test to their results across all instances.

The null hypothesis assumes that there is no difference in the median performance between VND1 and VND2. The test result is shown in Table 7.

Table 7: Wilcoxon test: VND1 vs. VND2

Comparison	p-value
VND1 vs VND2	0.1191

With a p-value of 0.1191, we cannot reject the null hypothesis at the 5% significance level. Therefore, we conclude that there is no statistically significant difference in solution quality between VND1 and VND2.

Summary of Results

These experimental findings provide a comprehensive comparison between iterative improvement strategies and VND algorithms. We now analyze their implications and answer the guiding questions in the following Discussion.

5 Discussion

5.1 First Part – Iterative Improvement Algorithms

The results of Part 1 highlight the influence of different algorithmic components on solution quality and computational cost.

(i) Which initial solution is preferable? The simplified RZ heuristic systematically outperforms random initialization. The Wilcoxon signed-rank tests confirm this finding across all combinations of pivoting rules and neighborhoods ($p < 2.2 \times 10^{-16}$ in all cases). Therefore, the RZ-based initialization is strongly recommended.

(ii) Which pivoting rule generates better quality solutions and which is faster? Best-improvement generally provides better solution quality than first-improvement, but at the cost of significantly higher computation time. For example, using the insert neighborhood on instances with 200 jobs, best-improvement achieves slightly lower average deviation than first-improvement, but requires twice the computational effort. Thus, first-improvement is preferable when time is limited, while best-improvement may be chosen when solution quality is prioritized.

(iii) Which neighborhood generates better quality solutions and what computation time is required to reach local optima? The insert neighborhood consistently leads to better-quality solutions than exchange and transpose, regardless of the initialization or pivoting strategy. However, it also has the highest computational cost. Transpose is the fastest operator, but yields the worst results. This trade-off suggests that insert is best when quality matters most, while exchange or transpose may be appropriate when faster solutions are needed.

5.2 Second Part – Variable Neighborhood Descent (VND)

The use of VND significantly improves solution quality over single-neighborhood local searches.

(i) Improvement over single neighborhoods. Both VND1 and VND2 show large improvements over using exchange alone, especially on small instances (e.g., over 30% gain for 50 jobs). The gain over insert-only search is smaller but still present, with VND2 showing better improvements on small instances, and VND1 performing better as problem size increases.

(ii) Which VND ordering is preferable? While both strategies perform similarly, VND2 (transpose → insert → exchange) achieves slightly better results for 50-job problems, whereas VND1 (transpose → exchange → insert) becomes more effective on larger instances. This may be explained by the fact that applying the computationally intensive insert operator last in VND1 allows for deeper exploitation after faster moves have been exhausted.

(iii) Are the two VND algorithms statistically different? The Wilcoxon signed-rank test yields a p-value of 0.1191, indicating no statistically significant difference between VND1 and VND2. This suggests that both algorithms perform similarly in general, although specific configurations may yield slight variations depending on instance size.

6 Conclusion

This study provided a comparative analysis of twelve iterative improvement (II) algorithms and two Variable Neighborhood Descent (VND) algorithms for solving the permutation flow-shop scheduling problem with the total completion time criterion (PFSP-CT).

In the first part, we showed that the simplified RZ heuristic significantly outperforms random initialization. The choice of pivoting rule and neighborhood operator also plays a key role: best-improvement yields slightly better solutions but at a higher computational cost, while the insert neighborhood achieves the best quality overall but requires more time.

In the second part, both VND1 and VND2 clearly outperformed single-neighborhood searches, particularly when compared to exchange-based local search. While VND2 yielded slightly better results than VND1 across all tested instances, this difference was not statistically significant.

These findings highlight the importance of initialization strategies and neighborhood combination in local search methods. Future work may explore adaptive neighborhood ordering or integration with metaheuristics to further enhance solution quality and efficiency.

6.1 Acknowledge

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

7 Annexe

A Complete Summary Table

Table 8: Average deviation and total computation time for each algorithm, initialization method, and instance size

Algorithm	Init	Size	Avg. Deviation (%)	Total Time (s)
best + exchange	random	50	5.36	1.37
best + exchange	srz	50	4.40	0.04
best + insert	random	50	4.47	3.00
best + insert	srz	50	3.01	0.12
best + transpose	random	50	19.65	0.11
best + transpose	srz	50	13.84	0.02
first + exchange	random	50	5.83	1.03
first + exchange	srz	50	4.49	0.04
first + insert	random	50	4.88	1.52
first + insert	srz	50	3.42	0.11
first + transpose	random	50	21.89	0.07
first + transpose	srz	50	13.40	0.02
best + exchange	random	100	5.64	18.59
best + exchange	srz	100	4.87	0.78
best + insert	random	100	4.35	34.22
best + insert	srz	100	2.95	1.38
best + transpose	random	100	19.64	0.77
best + transpose	srz	100	13.72	0.21
first + exchange	random	100	5.67	12.71
first + exchange	srz	100	4.44	0.79
first + insert	random	100	4.64	19.46
first + insert	srz	100	3.41	1.14
first + transpose	random	100	21.26	0.59
first + transpose	srz	100	13.07	0.19
best + exchange	random	200	5.66	322.22
best + exchange	srz	200	5.03	13.90
best + insert	random	200	4.20	635.20
best + insert	srz	200	2.90	26.30
best + transpose	random	200	19.45	2.47
best + transpose	srz	200	13.77	0.73
first + exchange	random	200	5.67	224.60
first + exchange	srz	200	4.78	13.81
first + insert	random	200	4.59	424.55
first + insert	srz	200	3.50	23.95
first + transpose	random	200	21.42	1.70
first + transpose	srz	200	13.36	0.62