Université Libre de Bruxelles

ECOLE
POLYTECHNIQUE
DE BRUXELLES

Heuristic Optimization - INFO-H413

Implementation Exercice Report Iterative improvement Algorithm for PFSP

Student :
Bienfait Alexandre 513930

Professor :
STÜTZLE Thomas

April 2024

Academic Year 2024-2025

Abstract

This report presents the implementation and evaluation of iterative improvement algorithms to solve the Permutation Flow-Shop Scheduling Problem with Total Completion Time (PFSP-CT), a classic NP-hard optimization problem in production scheduling. The study systematically compares twelve algorithmic configurations combining different pivoting rules, neighborhood structures, and initial solution strategies. Experiments are conducted on Taillard benchmark instances of varying sizes, and the Wilcoxon signed-rank test is used to determine the statistical significance of performance differences. Additionally, the report explores Variable Neighborhood Descent (VND) as a way to improve solution quality by cycling through multiple neighborhoods. The results show that initialization with the simplified RZ heuristic, combined with the first-improvement pivoting rule and exchange neighborhood, offers the best balance between computational time and solution quality. VND further enhances results by achieving over 50% improvement in deviation across all instance sizes, highlighting its effectiveness in escaping local optima and producing robust solutions.

Table des matières

1 Introduction

Permutation Flow-Shop Scheduling with Total Completion Time (PFSP-CT) is a classic NP-hard optimization problem in manufacturing and production. The goal is to schedule a set of jobs across multiple machines in a way that minimizes the total completion time. Efficient scheduling is critical in real-world industries—getting it right can boost productivity, cut costs, and ensure on-time deliveries.

In this report, we implement and test a set of iterative improvement algorithms for solving PFSP-CT. These algorithms, based on local search, work by gradually refining solutions to find the best possible schedule within a given search space. We compare twelve different versions of the algorithm, combining :

Two pivoting rules (first-improvement vs. best-improvement)

Three neighborhood structures (transpose, exchange, and insert moves)

Two initialization methods (random permutation vs. the simplified RZ heuristic)

We also explore Variable Neighborhood Descent (VND), where multiple neighborhood structures are used together during the search process.

To evaluate performance, we run experiments on 30 benchmark instances from Taillard's dataset, covering problem sizes of 50, 100, and 200 jobs. We use the Wilcoxon test to check if performance differences are statistically significant. The results provide practical insights for designing better local search strategies, especially for large-scale scheduling problems.

2 Theoretical Background

2.1 Permutation Flow-Shop Scheduling Problem (PFSP)

The Permutation Flow-Shop Scheduling Problem (PFSP) is a classic problem in operations research and production scheduling. In this setting, we are given n jobs that need to be processed on m machines, with each job passing through the machines in the same fixed order. The objective in this exercise is to minimize the total completion time across all jobs, defined as :

$$\text{Total Completion Time} = \sum_{i=1}^{n} C_i$$

Here, C_i is the time at which job i finishes on the last machine. Each job i has a predefined processing time p_{ij} on machine j. Because the PFSP becomes computationally hard (NP-hard) for three or more machines, it often requires heuristic or metaheuristic methods to find good solutions in reasonable time.

2.2 Iterative Improvement Algorithms

Iterative improvement, or local search, is a simple yet powerful optimization approach. The algorithm starts from an initial solution and explores its neighborhood to find a better one. If an improved solution is found, it replaces the current one. This continues until no further improvement is possible—i.e., a local optimum is reached.

There are two main pivoting strategies considered :

- **First-Improvement :** As soon as a better neighbor is found, the algorithm moves to it.
- **Best-Improvement :** All neighbors are evaluated, and the best among the improving ones is chosen.

Each strategy can behave quite differently in practice, with trade-offs between speed and the quality of solutions.

2.3 Neighborhood Structures

A neighborhood defines how solutions are connected—specifically, how we can move from one solution to another. This project uses three types of neighborhood operators :

- **Transpose :** Swaps two adjacent jobs.
- **Exchange :** Swaps two jobs at any positions.
- **Insert :** Takes a job from one position and inserts it elsewhere in the sequence.

Each of these modifies the job permutation in different ways and affects how the search explores the solution space.

2.4 Initial Solution Strategies

The starting point of the local search can have a big impact on its performance. In this project, two ways of generating an initial solution are used :

- **Random Uniform Permutation :** Jobs are shuffled randomly to create a starting point.
- **Simplified RZ Heuristic :** A simple construction method based on a dispatching rule that tends to produce better starting solutions than random ones.

    1. **Order Jobs** : Start by ordering the jobs in ascending order of their total processing times. The total processing time for job J_i is calculated as :

    $$\text{Total Processing Time} = \sum_{j=1}^{m} p_{ij}$$

    where p_{ij} is the processing time of job i on machine j.

    2. **Construct Solution** : Build the solution incrementally by inserting one job at a time. At each step, insert the next job in the position that minimizes the total completion time (CT) of the partial schedule.

The goal is to see how much the choice of the initial solution influences the final results after local search.

2.5 Variable Neighborhood Descent (VND)

Variable Neighborhood Descent (VND) is a more advanced local search method that tries to overcome the limitations of using a single neighborhood. It starts with one neighborhood and, when no further improvement is found, switches to another. This way, the algorithm can escape local optima that are "local" only in one neighborhood structure.
In this assignment, two sequences of neighborhoods are considered :

1. Transpose → Exchange → Insert
2. Transpose → Insert → Exchange

By rotating through multiple neighborhoods, VND increases the chances of finding better solutions compared to a standard iterative improvement with just one neighborhood.

3 Results

3.1 Exercice 1.1 - Applying 12 I.I algorithm

Algorithm	**Initial Solution**	**Average Deviation (%)**	**Total Time (s)**
best-exchange	random	25.67	19.06
best-exchange	srz	5.98	16.69
best-insert	random	25.79	16.40
best-insert	srz	5.69	14.39
best-transpose	random	26.06	1.06
best-transpose	srz	5.98	4.27
first-exchange	random	12.92	13.90
first-exchange	srz	5.89	9.33
first-insert	random	14.58	13.66
first-insert	srz	5.81	11.20
first-transpose	random	22.74	1.52
first-transpose	srz	5.97	4.34

Table 1 : Results for Instance Size 50.

Algorithm	Initial Solution	Average Deviation (%)	Total Time (s)
best-exchange	random	25.22	157.30
best-exchange	srz	7.08	125.74
best-insert	random	25.77	133.14
best-insert	srz	7.03	127.65
best-transpose	random	26.27	2.92
best-transpose	srz	6.96	29.69
first-exchange	random	14.18	108.37
first-exchange	srz	6.99	78.52
first-insert	random	16.58	99.46
first-insert	srz	7.14	86.91
first-transpose	random	23.24	8.07
first-transpose	srz	6.95	29.81

TABLE 2 : Results for Instance Size 100.

Algorithm	Initial Solution	Average Deviation (%)	Total Time (s)
best-exchange	random	26.12	940.08
best-exchange	srz	6.52	702.54
best-insert	random	26.24	847.06
best-insert	srz	6.16	758.83
best-transpose	random	26.91	10.41
best-transpose	srz	6.26	218.15
first-exchange	random	15.13	888.41
first-exchange	srz	6.23	497.63
first-insert	random	17.34	749.58
first-insert	srz	6.32	641.13
first-transpose	random	23.91	49.06
first-transpose	srz	6.24	217.13

TABLE 3 : Results for Instance Size 200.

Note : the Total Time values do not reflect the real time taken knowing that multi-processing has been used to fasten the computations. For this reason, the values are correct but the time required to run the benchmarks isn't the sum of all time values.

The experiments conducted across three instance sizes (50, 100, and 200 jobs) reveal several consistent trends in the performance of the algorithmic configurations :

- **Impact of Initial Solution :** Across all configurations and sizes, the `srz` initial solution consistently yields lower average percentage deviations from the best-known solutions compared to the `random` initial solution. The improvement is particularly significant when combined with Best pivoting.
- **Pivoting Rules :** The `best` pivoting rule typically produces slightly better solutions than `first` pivoting, especially when paired with `srz`. However, this comes at the cost of higher computation times, especially for larger instances.
- **Neighborhood Structures :**

- `Transpose` provides the fastest results but at the expense of solution quality.
- `Exchange` offers a good balance between computation time and solution quality, particularly effective with `first` pivoting.
- `Insert` performs similarly to `exchange` in terms of quality but tends to require slightly more time.

- **Scalability :** As the instance size increases, the computation time increases considerably, especially for algorithms using the `best` pivoting rule. However, the `transpose` neighborhood remains computationally efficient even on large instances, despite its lower solution quality. The worst computational time is obtained with the best-exchange algorithm and random initialization (940.08 s for all 10 run of 10 instances of size 200). This may be counterintuitive given the time required to construct the initial solution for **srz** initialisation, which should increase the computational time. But since the solution is often close to a local optimum, the iterative improvement algorithm converges much faster than random initialisation

3.2 Exercice 1.1 - Using Wilcoxon test to detect statistically significant difference

The Wilcoxon test is a non-parametric statistical test used to compare two paired groups or to assess whether a set of samples comes from a distribution with a specific median.
The Wilcoxon test is particularly useful when the data does not meet the assumptions required for parametric tests, such as the t-test. It is based on the ranks of the data rather than the data values themselves, making it robust to outliers and suitable for ordinal data.

3.2.1 Question 1 : Which initial solution is preferable ?

The Wilcoxon test was performed to compare two types of initial solutions : **SRZ** and **Random**, across various pivoting rules and neighborhoods.

Pivoting Rule	Neighborhood	p-value	Better Initial Solution	Significance
Best	Transpose	6.08×10^{-51}	SRZ	Yes
Best	Exchange	6.08×10^{-51}	SRZ	Yes
Best	Insert	6.08×10^{-51}	SRZ	Yes
First	Transpose	6.08×10^{-51}	SRZ	Yes
First	Exchange	6.08×10^{-51}	SRZ	Yes
First	Insert	6.08×10^{-51}	SRZ	Yes

TABLE 4 : Wilcoxon Test for Initial Solutions (Deviation %)

Conclusion : The SRZ initial solution significantly outperforms the Random initialization in all tested configurations. All p-values are well below 0.05, indicating strong statistical significance.

3.2.2 Question 2 : Which pivoting rule generates better quality solutions and which is faster ?

This section compares the **Best** and **First** pivoting rules with respect to both solution quality (Deviation %) and computational time (Time (s)).

Initial Solution	Neighborhood	p-value	Better Pivoting Rule	Significance
Random	Transpose	1.18×10^{-44}	First	Yes
SRZ	Transpose	1.25×10^{-44}	First	Yes
Random	Exchange	6.08×10^{-51}	First	Yes
SRZ	Exchange	5.56×10^{-12}	First	Yes
Random	Insert	6.08×10^{-51}	First	Yes
SRZ	Insert	7.70×10^{-4}	Best	Yes

TABLE 5 : Wilcoxon Test for Pivoting Rules (Deviation %)

Initial Solution	Neighborhood	p-value	Faster Pivoting Rule	Significance
Random	Transpose	6.08×10^{-51}	Best	Yes
SRZ	Transpose	6.08×10^{-51}	Best	Yes
Random	Exchange	4.73×10^{-10}	First	Yes
SRZ	Exchange	1.57×10^{-21}	First	Yes
Random	Insert	2.33×10^{-17}	First	Yes
SRZ	Insert	1.72×10^{-27}	First	Yes

TABLE 6 : Wilcoxon Test for Pivoting Rules (Time in Seconds)

Conclusion : The **First** pivoting rule generally provides better solution quality, except for the insert neighborhood with SRZ where **Best** slightly outperforms. In terms of time efficiency, the **First** rule is faster in most cases except for the transpose neighborhood, where **Best** is faster.

3.2.3 Question 3 : Which neighborhood generates better quality solutions and what computation time is required ?

Neighborhoods **Transpose** and **Exchange** were compared for each pivoting rule and initial solution configuration.

Pivoting Rule	Initial Solution	p-value	Better Neighborhood	Significance
Best	Random	2.82×10^{-4}	Exchange	Yes
Best	SRZ	1.25×10^{-8}	Transpose	Yes
First	Random	6.86×10^{-51}	Exchange	Yes
First	SRZ	2.17×10^{-19}	Transpose	Yes

TABLE 7 : Wilcoxon Test for Neighborhoods (Deviation %)

Pivoting Rule	Initial Solution	p-value	Faster Neighborhood	Significance
Best	Random	6.08×10^{-51}	Transpose	Yes
Best	SRZ	6.14×10^{-51}	Transpose	Yes
First	Random	1.24×10^{-3}	Exchange	Yes
First	SRZ	5.25×10^{-33}	Exchange	Yes

TABLE 8 : Wilcoxon Test for Neighborhoods (Time in Seconds)

Conclusion : There is no universally superior neighborhood. **Exchange** performs better for random initialization, while **Transpose** yields better results with SRZ. Regarding time, **Transpose** is generally faster under Best pivoting, while **Exchange** is faster with First pivoting.

3.3 General summary

- **Initial Solution :** SRZ significantly outperforms Random across all cases.
- **Pivoting Rule :** First is usually better in solution quality and time, with a few exceptions.
- **Neighborhood :** Performance varies based on pivoting rule and initial solution ; Exchange is better with Random, Transpose is better with SRZ.

4 VND Results

We implemented two Variable Neighborhood Descent (VND) algorithms based on different orderings of neighborhoods :

- VND1 : Transpose $\rightarrow$ Exchange $\rightarrow$ Insert
- VND2 : Transpose $\rightarrow$ Insert $\rightarrow$ Exchange

Both algorithms use the **first-improvement** pivoting rule and start from an initial solution generated using the simplified RZ heuristic.

(a) Aggregated Statistics per Instance Size

- **Average Percentage Deviation from Best-Known Solutions :**
  - 50 jobs : 5.37%
  - 100 jobs : 6.46%
  - 200 jobs : 5.94%
- **Average Computation Time :**
  - 50 jobs : 0.39 s
  - 100 jobs : 3.82 s
  - 200 jobs : 31.22 s
- **Percentage Improvement over Exchange and Insert Neighborhoods :**
  - 50 jobs : 56.04%

- 100 jobs : 51.75%
- 200 jobs : 55.55%

(b) Statistical Comparison of VND Algorithms
A Wilcoxon signed-rank test was conducted to compare the performance of VND1 and VND2 in terms of average deviation. The test yielded the following results :

- **Wilcoxon test statistic :** 17.0
- **p-value :** 0.0923

Although the result is not statistically significant at the 5% level, it shows a slight preference for VND2 :

- VND1 Average Deviation : 5.93%
- VND2 Average Deviation : 5.91%

(c) Conclusions
Both VND variants significantly outperform the use of a single neighborhood (either exchange or insert), with an average improvement of over 50% across all instance sizes. This demonstrates the benefit of systematically exploring multiple neighborhoods in sequence. However, the computation time is significantly longer with up to 31.22 seconds required to run a single run on the biggest instance where it would take a maximum of 9.4 s for single neighborhood (940 s for 10 runs of the 10 instances) Between the two variants, VND2 achieves slightly better results in terms of solution quality, although the difference is not statistically significant. Given this and its consistent performance, VND2 is considered the preferable neighborhood ordering among the two tested.

5 Conclusion

The empirical analysis of the 12 algorithmic configurations highlights the critical impact of the initial solution strategy, pivoting rule, and neighborhood structure on both solution quality and computation time.

- The SRZ heuristic as an initial solution significantly improves solution quality across all settings and should be favored over random initialization.
- The Best pivoting rule generally provides slightly better-quality solutions but incurs higher computational cost. In contrast, First pivoting offers faster execution with competitive results, especially when paired with SRZ and exchange-based neighborhoods.
- Among neighborhoods, Exchange demonstrates the best balance between time and quality, while Transpose is suitable when minimal runtime is critical, despite its higher deviation.

In conclusion, the combination of SRZ initial solution, First pivoting rule, and Exchange neighborhood emerges as the most balanced approach, offering high-quality solutions within reasonable computational times, making it a strong default choice for practical applications.

VND Perspective : The application of Variable Neighborhood Descent (VND) further validates the advantage of using multiple neighborhood structures sequentially. Both VND1 and VND2 significantly outperformed single-neighborhood strategies, with average deviations improving by over 50% across all instance sizes. While the difference between VND1 and VND2 was not statistically significant, VND2 showed slightly better performance and is therefore the preferred configuration. Despite increased computation time, the enhanced solution quality makes VND particularly attractive for scenarios where robustness and optimization depth are prioritized over runtime.