الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

جامعة سعيدة د. مولاي الطاهر

كلية الرياضيات و الإعلام الآلي و الاتصالات السلكية و اللاسلكية

قسم: الإعلام الآلي

# Mémoire de Master en informatique

## Spécialité :

Réseaux informatique et Système Répartis

## T h è m e

**VND for Dynamic Customer Order Scheduling in Non-Identical Server Systems**

- **Présenté par :**
  - 1- OTMANI Mustapha
  - 2- RAHMANI Alaa Eddine
- **Dirigé par :**
  - Dr. MEKOUR Mansour

Année universitaire 2024-2025


First of all, ***Al-hamdoulillah*** , it is thanks to Him that we have found the strength and the patience to complete this thesis.

We would like to express our deep gratitude to **Mr. Dr. MEKOUR Mansour** for the honor he gave us by agreeing to supervise us. His support precious, his wise advice and his patience throughout this work were essential. His expertise and support were instrumental in making this happen memory.

We also extend our sincere thanks to the **members of the jury** for for agreeing to evaluate our work. We express our gratitude and our deep respect.

We express our deepest gratitude to **our families** , who have always believed in us and supported us unconditionally throughout our journey. Their Their caring presence and moral support gave us the strength to move forward.

We do not forget **our friends** and **loved ones** who were by our side, with their reassuring words, their precious help, their presence in difficult times and their participation in this perfect day and these moments.

This memory is also a little bit yours.

**Thank you from the bottom of my heart.**

**MUSTAPHA & ALAA EDDINE**

# DEDICATION 1

I dedicate this work to:

First of all, those for whom no one can compensate for the sacrifices they have made for my education and well-being to my parents who sacrificed themselves for me support throughout my training and which are the origin of my success

that god protects them and prolongs their life .

To all my family,

My uncle and aunts, My grandparents, Those who supported me in the most difficult moments. She who gave me a decent education, her love made me what I am today.

To all teachers,

who accompanied me throughout my journey.

Through their knowledge, their patience and their commitment, they have helped to shape my mind, to

building the foundations for my success, and for the impact they had on my education, on my personal development.

To my little brother “Abd El Hadi” who was not born to my mother,

To my friends and loved ones,

Each by name and position, Taher, Djelloul, Ilyes, Abd El Karim, Abd El Djebbar,

And the others,

Thank you for being this light in my life, these sincere and benevolent presences which make the days sweeter and the trials lighter.

Your friendship is a treasure, your smiles a strength, and your love a refuge.

In every shared moment, in every engraved memory, I find a reason to smile and be grateful.

This dedication is for you, with all my heart.

*MUSTAPHA*

# DÉDICATION 2

I dedicate this work to:

To my dear late father,

You have been an inexhaustible source of strength, wisdom and love for me.

It is to you that I dedicate this work, fruit of your sacrifices and your values that you have given me

transmitted.

To my dear mother,

Who suffered without letting me suffer, who never said no to my demands and who never

spared no effort to make me happy.

To you my brothers and sisters,

Your presence, your encouragement and your constant support have been a

valuable source of motivation.

In times of doubt and in times of joy, you have always been there.

I thank you from the bottom of my heart for your love, your complicity and your

kindness.

I fully share this success with you.

To all my friends and loved ones,

Thank you for your presence, your encouragement and your comforting words throughout

along this route.

I am deeply grateful to you, and I dedicate a part of this success to you.

*ALAA EDDINE*

# Contents

# List of Figures

# List of Tables

# General Introduction

In a constantly changing industrial environment, the ability to plan and efficiently manage customer orders is a crucial strategic lever for companies. These companies are now evolving in production or increasingly complex service environments, marked by a diversity of resources, capacity constraints, changing customer priorities, and continuous pressure on deadlines. In this context, dynamic order planning becomes an essential issue to guarantee the performance and responsiveness of the system.

This research work addresses this problem and focuses more specifically on the dynamic scheduling of customer orders in systems with non-identical servers. In such environments, the resources (or servers) have heterogeneous capacities; that is, they differ in their processing speed, availability, or ability to perform certain tasks. This functional heterogeneity greatly complicates the optimal allocation of orders, as each resource cannot indiscriminately execute all operations under equivalent conditions.

Added to this complexity is the dynamic nature of the problem: orders do not arrive in a static or predictable manner, but rather continuously and sometimes randomly. The system must therefore react in real time while considering multiple constraints: variable priorities, due dates, temporarily unavailable resources, and current server loads. All these elements give the problem a highly combinatorial nature, making it difficult or even impossible to solve exactly within reasonable timeframes.

To address this challenge, we propose an optimization approach based on metaheuristics, and more specifically on the Variable Neighborhood Descent (VND) algorithm. This method, renowned for its effectiveness in solving difficult optimization problems, allows improving an initial solution by successively exploring several neighborhoods, thus overcoming local optima and approaching a global optimum.

Thus, this thesis proposes a simulation of customer order scheduling problems based on the Job Shop Scheduling Problem (JSSP), using the VND method in a system with non-identical servers. The objective is to optimize the allocation of tasks to machines while respecting production constraints, in order to minimize

performance criteria such as total processing time (makespan) or costs.

# Chapter 1

# Dynamic Order Scheduling

## 1.1 Introduction

In a context where the personalization of services and speed of execution become determining factors of competitiveness, dynamic planning of customer orders in heterogeneous IT environments represents a major strategic issue. Many modern systems, whether logistics platforms, data centers or cloud architectures, are based on servers with different capabilities. In such environments, called *multi-system non-identical servers*, each resource has specific characteristics — processing speed, capacity, availability — which complicate optimal task scheduling.

The central objective of this type of problem is to determine, in real time, a strategy of assigning orders to these servers in order to minimize an overall criterion, such as the maximum completion time (makespan), weighted delays, or even customer satisfaction. This problem becomes even more difficult when orders arrive dynamically, requiring frequent adaptations of the solution depending on the system status and resource availability.

Faced with this complexity, exact optimization methods quickly become unusable on a large scale. It is in this context that heuristic methods, and in particular the Variable Neighborhood Descent (VND), prove to be powerful tools. The VND method offers a systematic exploration of several neighborhoods to escape local minima and gradually improve the current solution.

## 1.2 Background to Order Scheduling in Non-Identical Systems

In a system of non-identical servers, each server $S_i$ has a different processing capacity, often modeled by a speed $v_i$ or a specific processing time $p_{ij}$ for a command

$j$.

Scheduling aims to assign a set of commands $\{C_1, C_2, \ldots, C_n\}$ to the servers $\{S_1, S_2, \ldots, S_m\}$ in order to optimize a global criterion, for example the minimization of total processing time (makespan) or customer lead times.

**Concrete example:**

Suppose three servers with speeds $v = \{1,\ 0.8,\ 1.2\}$ (units of orders per hour). An order $C_j$ requires work of $w_j$ units. The processing time on server $S_i$ is then $t_{ij} = \frac{w_j}{v_i}$. The scheduling consists of minimizing the maximum processing completion time across all servers.

## 1.3 Definition of the Dynamic Scheduling Problem

The dynamic problem implies that orders arrive over time, and that scheduling must adapt in real time. Let $C(t)$ be the set of orders known at time $t$.

The goal is to find an assignment function $f_t : C(t) \to \{S_1, \ldots, S_m\}$ which minimizes a cumulative cost, for example:

$$\min \sum_{j \in C(t)} w_j \cdot C_j(t)$$

where $C_j(t)$ is the completion time of command $j$ at time $t$.

**Specific constraints:**

- Non-identical servers with varying processing times.
- Customer priorities, modeled by weights $w_j$.
- Limited server capacity (maximum number of simultaneous orders).

## 1.4 VND Methodology for Scheduling Optimization

The Variable Neighborhood Descent (VND) method is a local optimization heuristic that explores several types of neighborhoods to improve the current solution.

**Mathematical modeling:**

Let $S$ be a solution for assigning commands to servers.

We define several neighborhoods $N_k(S)$, for example:

- $N_1$: exchange of two commands between different servers.
- $N_2$: permutation of the order of commands on the same server.
- $N_3$: moving an order from one server to another.

**VND Algorithm:**

1. Initialize $k = 1$ and an initial solution $S$.
2. Search for a better solution $S' \in N_k(S)$.
3. If $S'$ is better, replace $S$ with $S'$ and return to $k = 1$.
4. Otherwise, increase $k$ (move to the next neighborhood).
5. Stop if $k > k_{\max}$.

Dynamic enforcement consists of re-executing VND on each arrival or modification of the orders.

## 1.5 Characteristics of Non-Identical Server Systems

Servers can be characterized by:

- Specific processing time $p_{ij}$: time to process command $j$ on server $i$.
- Availability $a_i(t)$: binary indicator if the server is available at time $t$.
- Maximum capacity $c_i$: maximum number of orders that can be processed simultaneously.

These parameters are integrated into the scheduling model to ensure feasibility.

## 1.6 Order Scheduling Process

**Key steps:**

1. *Reception and classification*: each order $C_j$ is characterized by a weight $w_j$, an arrival time $r_j$, and a deadline $d_j$.
2. *Initial assignment*: a simple heuristic can assign each command to the server with minimum processing time: $\min_i p_{ij}$.
3. *Dynamic reordering with VND*: at each new event (arrival, end of processing), apply VND to improve the solution.
4. *Delay monitoring*: calculate the delay $R_j = \max(0, C_j - d_j)$ for each order.

## 1.7 Optimization Areas and Key Factors

- **Real-time visibility**: modeled by the continuous updating of the states $a_i(t)$, $c_i$, and queues.
- **Flexibility**: ability to move orders between servers without excessive costs.
- **Priority management**: weighting of orders in the objective function.
- **Cost minimization**: often formulated as minimization of total weighted delays:

$$\min \sum_j w_j R_j$$

**Simplified Numerical Example**

Let us assume 2 servers $S_1$, $S_2$ with respective speeds $v_1 = 1$, $v_2 = 0.5$ (slower). Three commands $C_1$, $C_2$, $C_3$ with loads $w = \{4, 3, 2\}$ and priorities $w_j = \{2, 1, 3\}$.

Processing time on $S_1$: $\{4, 3, 2\}$

Processing time on $S_2$: $\{8, 6, 4\}$

An initial assignment could be:

- $C_1$, $C_3$ on $S_1$
- $C_2$ on $S_2$

The VND could test:

- Swap $C_2$ and $C_3$ to reduce the total weighted time.
- Reorder $C_1$ and $C_3$ on $S_1$.

## 1.8 Conclusion

Dynamic order scheduling in non-identical server systems constitutes a complex and current problem, encountered in many areas such as distributed computing, intelligent logistics, and cloud service management. This complexity results mainly from the heterogeneity of resources, temporal variability of demands, and operational constraints imposed by customer priorities or capacity limitations.

In this chapter, we have presented a rigorous modeling of the problem, integrating realistic parameters such as processing speeds, server availability, priority weights, and capacity constraints. The Variable Neighborhood Descent (VND) approach has proven to be particularly suited to this dynamic framework, thanks to its ability to explore multiple neighborhoods and escape suboptimal solutions.

# Chapter 2

# Scheduling Problems

## 2.1 Introduction

Scheduling is a fundamental area of research operational aiming to organize the execution of tasks on limited resources while respecting constraints and optimizing performance criteria [1].Two Major problems in this area have been the subject of in-depth studies:

- Customer Order Scheduling : which consists of the optimal assignment of orders (tasks) to heterogeneous resources (machines, servers) to satisfy external requests.[2]

- The Job Shop Scheduling Problem (JSSP) : which involves the sequential scheduling of jobs (sets of interdependent tasks) on machines with strict precedent constraints.[3]

These issues are of crucial importance in various industrial sectors and logistics. Where optimized planning allows for significant cost reductions operational and production deadlines.

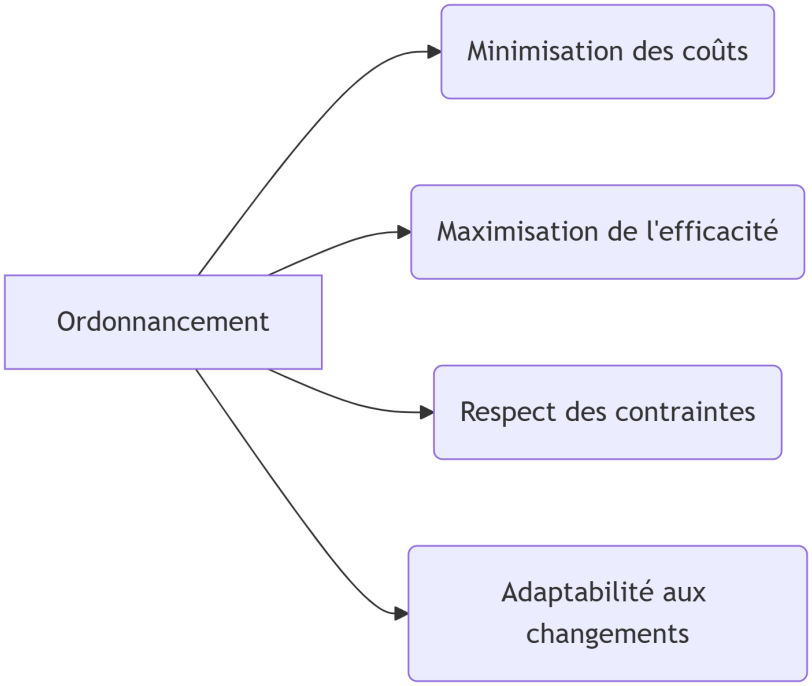


Figure 2.1: Scheduling diagram.

## 2.2 Scheduling problems

### 2.2.1 Definition and Issues

Scheduling is programming the execution of an achievement by assigning resources to tasks and setting their execution dates. Scheduling problems appear in all areas of the economy: computing (tasks: jobs; resources: processors or memory...), construction (project monitoring), industry (workshop problems, production management), administration (timetables).[4]

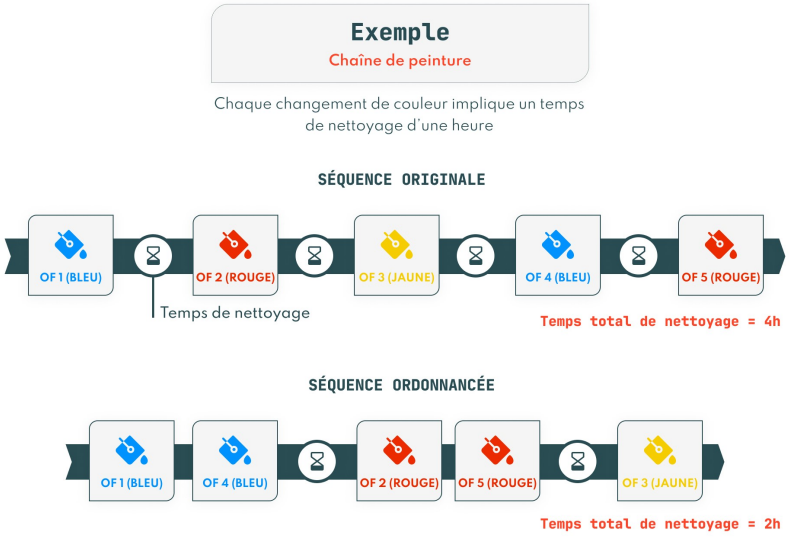


Figure 2.2: Scheduling issues .

### 2.2.2 Complexity of scheduling problems

Complexity theory is concerned with the formal study of difficulty intrinsic theoretical problems in computer science or a problem by relation to another and the analysis of the complexity of programs and algorithms. Concretely, we seek to know if the problem studied is rather ”easy” or ”difficult” to solve based on an estimate (theoretical) calculation times and requirements in memory computer science. Scheduling problems are problems combinatorial optimization. For to solve a scheduling problem, we must always look for to establish its complexity, because this determines the nature of the algorithm to be implemented work. If the problem studied belongs to the class P, we know in advance that a polynomial algorithm exists to solve it. Otherwise, if the problem is NP-hard, two approaches are possible. The first is to propose an approximate algorithm, therefore a heuristic, which calculates in time polynomial a solution approaching the better than the optimal solution. A alternative is to propose an algorithm that calculates the optimal solution of the problem, but for which the worst-case complexity is exponential. In this case, the challenge is to design an algorithm that can solve in time acceptable problems of the largest possible size. [5]

#### Classes P, NP

**Class P:** P is a complexity class that represents the set of decision problems can be solved in polynomial time, since they can also be verified in polynomial time.

Therefore, P is a subset of NP. Notation : Solving time = O(n□), where n is the size of the input and k is a constant.[6]

**The NP class** A problem is in the NP class if it is possible to verify a solution proposed in polynomial time (even if we cannot find it quickly). NP-complete: A problem is said to be NP-complete if it meets two conditions : It belongs to the NP class (one can quickly verify a proposed solution). It is at least as hard as all other NP problems , i.e. that all NP problems can be reduced to it in polynomial time . NP-hard: A problem is said to be NP-hard if it is at least as difficult as the NP problems, but it is not necessarily in NP (so it is not necessarily a decision problem).

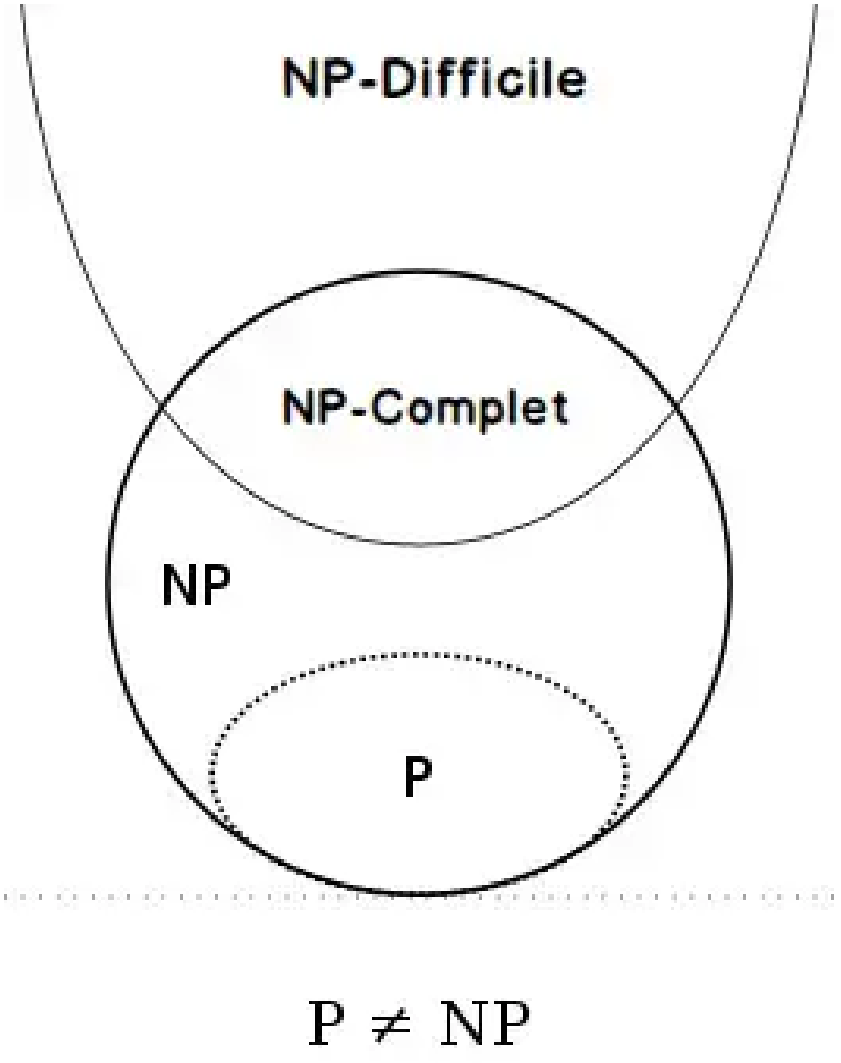


Figure 2.3: Classes P and NP
[6]

## 2.3 The Job-Shop Scheduling Problem (JSSP)

### 2.3.1 History of Job-Shop

The history of the Job-Shop goes back more than forty years. However, it was in the 1960 that a now widely recognized scheduling problem arose, that of Fisher and Thompson involving 10 tasks and 10 machines[7]. This problem has persisted for almost a quarter of a century and sparked rivalry among researchers of the field[8]. In this writing, the authors propose an official definition of the problem.

They also present resolution techniques classified into two categories, namely:

- The exact methods provide all the information needed to construct and execute an exact algorithm, as well as partitioning techniques and of valid inequalities.[9]

- Approximate methods describe some priority rules and the methods used, including the bottleneck procedure, search algorithms local and opportunistic scheduling procedures. They also mention the principal neighborhoods commonly used in the literature.[9]

A comprehensive review of the state of the art of job-shop from its beginnings until the end of the 90s. It lists the articles by organizing according to the resolution method used for JavaScript. In addition to the presenting the method, Jain and Meerran provide an exhaustive list of instances. The latter is used as a reference for comparing methods of resolution. They also provide summary tables where they offer a global overview of methods and extensions, accompanied by the corresponding precise techniques have their origins in the initial work of Manne (1960), followed by those of Brooks and White (1965)[11], then in 1968 by Greenberg who employed a linear formalization in integers. This has resulted in numerous publications, notably those of Ficher in Fisher (1973a) and Fisher (1973b)[12], who used the lagrange multipliers. Since 1975, McMahon and Florian have developed an algorithm which outperforms the performance of Fisher algorithms.

Over the past twenty years more efficient algorithms have been developed, notably through the use of Lagrangian relaxation. This method consists of to give up certain constraints in order to solve problems that are similar more of reality, involving more than a hundred jobs and around fifty machines. This relaxation method was developed to associate each constraint relaxed a penalty, called augmented Lagrangian relaxation technique. By the Subsequently, in 1989, Carlier and Pinson[13] introduced an algorithm based on the method of Branch and Bound which allows for the first time to optimally resolve the 10x10 problem established by Fisher and Thompson, as well as other more complex.

Although these methods of solving exact Job-Shop type problems provide optimal results, they have major limitations in terms of of computational time, especially for complex instances such as the FT problem 10x10. This time constraint motivated the parallel development of approaches alternatives, called approximate or heuristic methods, aimed at obtaining suboptimal solutions in a reasonable time. Approximate methods constitute a relevant alternative to exact methods, particularly because of their computational efficiency, which makes it possible to obtain solutions that can be achieved within a reasonable time. This property makes them particularly suited to solving large-scale industrial problems, where strict optimality is often sacrificed for speed of execution.

### 2.3.2 Definition

The Joint Job Shop Scheduling Problem (JSSP) is a research area in booming planning, widely studied by engineering researchers and academics. The Job Shop Scheduling Problem (JSSP) is one of the optimization problems the most classic and most important combinatorics in operational research. A typical JSSP can be described as follows: in a workshop environment containing several machines $M = \{M_1, M_2, \cdots, M_m\}$, there are a number of tasks $J = \{J_1, J_2, \cdots, J_i, \cdots, J_n\}$, each task $J_i$, containing a series of operations $O_i = \{O_{i1}, O_{i2}, \cdots, O_{ij}, \cdots, O_{in_i}\}$ to be processed according to a technological sequence predefined. Each operation is assigned to a machine, with a processing time given $p_{ij}$. The sequencing of operations on all machines must be done so that to minimize the maximum execution time of all tasks, i.e. the time of response (Makespan). Each machine can only process one job at a time and a job can only run on only one machine at a time. Jobs running on a machine $M_m$ must wait in the output stock. In the same way each Machine $M_m$ has an input stock where jobs wait to be executed on this machine. Job transfer time is not taken into account.[9]

We assume that the values $p_{ij}$ are non-negative integers. The objective is to find a feasible scheduling that minimizes the makespan, where $C_j$ denotes the end execution of the last operation $O_{j,n_j}$ of job $j$. The result of the scheduling is consisting of the start dates of all machine operations.[9]

The JSSP is a classic machine scheduling problem. The first machine scheduling problems listed in the literature are problems two- and three-step scheduling, including preparation times (Johnson, 1954). For identical operations, these problems correspond to problems two- and three-machine workshop scheduling, the objective being to minimize the total elapsed time.

### 2.3.3 Proof of membership in the NP-hard class

Knowing which class a problem belongs to is very important in order to understand it well. As for the job shop composed of $n$ jobs to be executed on $m$ machines, it exists $(n!)^m$ possible solutions. The job shop is an NP-Hard problem and NP-complete and several demonstrations exist. As for the Job-Shop we have the following results:

- the Job-Shop variants $J2||C_{max}$ and $J3|p_{ij} = 1|C_{max}$ are proven NP.[14]
- the following variants are polynomially solvable:
- $n$ jobs (with at most 2 operations) and 2 machines.[15]
- $J2|p_{ij} = 1, r_i|C_{max}$.[16]
- the job shop with 2 jobs and $m$ machines.[17]

In principle, when we demonstrate that a problem A is NP-Hard, and that this problem is a reduction of a more complex problem B, then B is necessarily NP-

Hard. The Jobshop with $n$ jobs and $m$ machines is therefore NP-Hard, it is enough to reduce it to the variant $J2||C_{max}$ which is already NP-Hard.

Job Shop problem can also be reduced to the scheduling problem customer orders, considering the following elements:

- Each customer order corresponds to a job in the Job Shop.
- Each production step of an order corresponds to an operation on a given machine.
- Machines remain limited resources as in the Job-Shop.
- The common goal is often to minimize the total processing time (makespan), to reduce delays, or to meet delivery dates.

### 2.3.4 Components of the JSSP

#### Jobs

A job is an ordered sequence of operations to be executed in a specific order, characterized by:

- Number of operations: Variable depending on the job (e.g.: Job 1 = 3 operations, Job 2 = 2 operations).
- Sequence constraint: The operation $O_{i,j+1}$ can only begin if $O_{i,j}$ is finished.
- An availability date $r_i$: the execution of task $i$ cannot start before this date.
- A due date noted $d_i$: task $i$ must be completed before this date.
- The operating duration of treatment noted $p_i$.

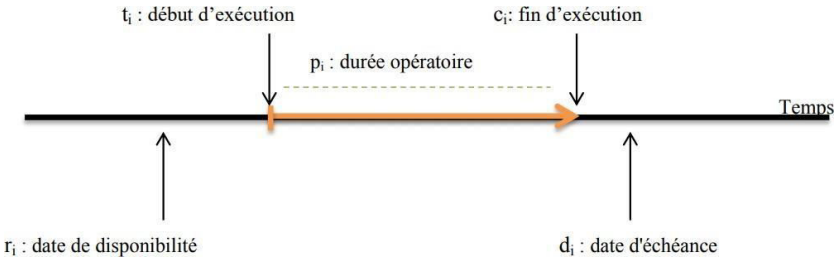


Figure 2.4: The representation of Characteristics of a task $i$

#### Machines (Resources)

Resources capable of performing operations, with capacity constraints. There are 2 types:

- Dedicated: Each operation is assigned to a specific machine.
- Parallel: Multiple machines can perform the same type of operation (variant flexible).

Key constraint: Mutual exclusion: A machine can only process one operation at a time.

Example:

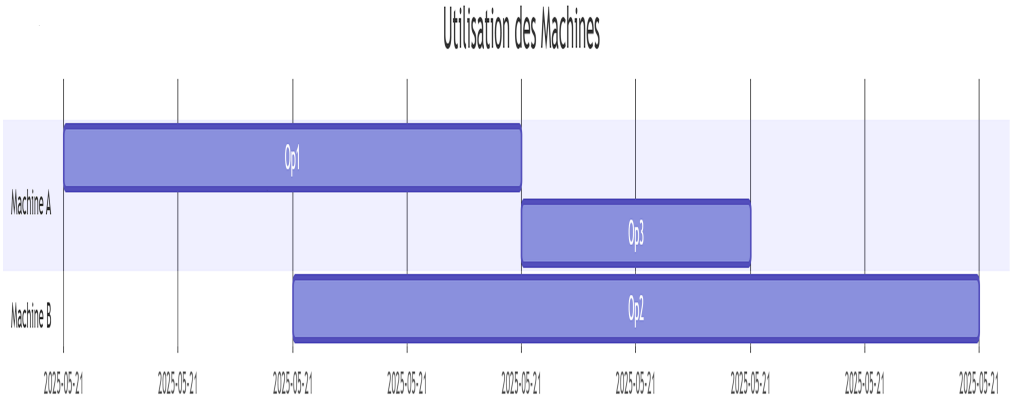


Figure 2.5: Using Machines in JSSP

### Operations

Basic element of the JSSP, combining:

- A machine: Where the operation is performed.
- A processing time ($p_{ij}$): Fixed or variable duration.

Properties:

- Non-preemption: An operation started must be completed without interruption.
- Scheduling: Depends on precedence (intra-job) and resource constraints (inter-job).[18]

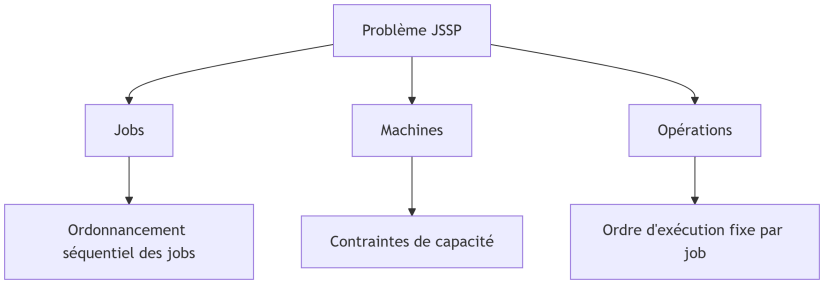


Figure 2.6: Components of the JSSP

### Constraints

The JSSP is governed by strict constraints that make it an NP-hard problem. Here is a clear classification of major constraints.

Types:

**Precedence Constraints (Intra-Job)** Operations within the same job must be executed in a specific order. For a job $i$ composed of operations $O_{i1}, O_{i2}, ..., O_{im}$: $C_{ij} \geq C_{i,j-1} + p_{ij} \quad \forall j \geq 2$ Or: $C_{ij}$ = end time of operation $O_{ij}$. $p_{ij}$ = duration of operation $O_{ij}$.

**Resource Constraints (Inter-Jobs)** A machine can only process one operation at a time. For two operations $O_{ij}$ and $O_{kl}$ executed on the same machine $M$: $C_{ij} \geq C_{kl} + p_{ij}$ OR $C_{kl} \geq C_{ij} + p_{kl}$

Example: Machine A executes either $O_{11}$ (duration=3h) or $O_{22}$ (duration=2h), but not both simultaneously.

**Non-Preemption Constraints** An operation that has started cannot be interrupted. If $t \in [S_{ij}, C_{ij}]$, then $O_{ij}$ occupies the machine continuously. Where $S_{ij}$ = start time of $O_{ij}$.

Example: $O_{31}$ (duration=4h) must run without pause on machine B.

### Complete Example

$$\text{job } 0 = [(0,3),(1,2),(2,2)] \text{job } 1 = [(0,2),(2,1),(1,4)] \text{job } 2 = [(1,4),(2,3)]$$

In the example, task 0 has three tasks. The first, $(0, 3)$, must be processed on machine 0 in 3 time units. The second, $(1, 2)$, must be processed on machine 1 in 2 time units, and so on. In total, there are eight tasks.[18]

**Solution to the problem** One solution to the workshop problem is to assign a start time to each task, which respects the constraints indicated above. The diagram below illustrates a possible solution:

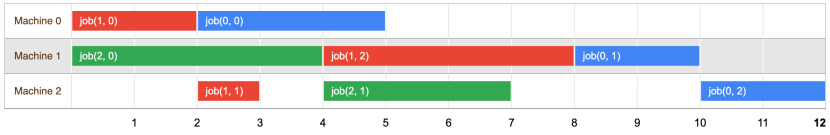


Figure 2.7: Possible solution to the tasks.
[18]

### 2.3.5 Possible JSSP extensions

#### Flexible JSSP (FJSSP) Partial

The flexible job shop problem (FJSP) is an extension of the classical problem of workshop scheduling, which allows an operation to be executed by any machine in a given set. The problem is to assign each operation to a machine and to order operations on these machines in such a way as to minimize the maximum execution time (makespan) of all operations.[19]

Our non-identical servers can offer partial flexibility:

- Some operations can run on multiple servers (e.g. CPU tasks on Server A or B).
- Others are constrained to a specific server (eg: GPU tasks).

#### Dynamic JSSP

Our orders arrive in real time  Requires continuous re-optimization. Integrate using VND with:

- Periodic re-optimization (e.g. every 10 new orders).
- Online insertion of new tasks into the current solution.

#### JSSP with Customer Priorities

Some orders are urgent or premium. We integrate it:

- Weight the objective function by the priorities $w_i$: Minimize $\sum w_i \cdot T_i$ ( $T_i$ = delay of job $i$ )
- Prioritize swaps/insertions on critical jobs.

## 2.4 Conclusion

In this chapter we have presented the basics of the problems scheduling, highlighting their algorithmic complexity and their importance in production systems. We then introduced the problem Job-Shop Scheduling System (JSSP), one of the most studied due to its structure combinatorics and its recognized difficulty (NP-hard). This problem models realistic situations where multiple jobs need to be run on different machines with precedent and resource constraints.

After analyzing the main components of the JSSP and its possible extensions, we are now laying the foundations of our work: we are going to simulate problems customer order scheduling based on JSSP modeling, in order to reproduce realistic industrial environments and apply optimization methods to them. The JSSP thus serves as rigorous support for our problem while maintaining sufficient flexibility to integrate the particularities of the system studied.

# Chapter 3

# The Variable Neighborhood Descent Method

## 3.1 Introduction

Scheduling is a central challenge in many industrial and logistics sectors, aimed at efficiently organizing tasks with limited resources. To solve these complex problems, various optimization methods have been developed.

Among the existing approaches, we distinguish two main families: exact methods, which guarantee an optimal solution but become impractical for large-scale problems, and approximate methods, more suited to real cases often complex.

The VND (Variable Neighborhood Descent) method represents an elegant solution in this landscape. This intelligent concept is based on the systematic exploration of different types of possible modifications (neighborhoods) of a solution, alternating strategically between fine local transformations and more global ones.

Its application to scheduling shows its relevance, allowing the adaptation of the search for solutions to the specificity of the constraints encountered. The strength of this approach lies in its balance between precision and efficiency, as well as in its ability to avoid the pitfalls of locally optimal but globally suboptimal solutions.

This introduction paves the way for exploring the fundamental principles and practical applications of these optimization methods, with a particular focus on the VND methodology and its potential for solving varied scheduling problems, particularly within the framework of the JSSP and the simulation of customer orders.

## 3.2 Review of Optimization Methods Applied to Scheduling

In this section, we provide a summary view of the main methods used to deal with scheduling issues when trying to solve a problem. In any article, especially, and whatever the method used or developed, we find common structures.

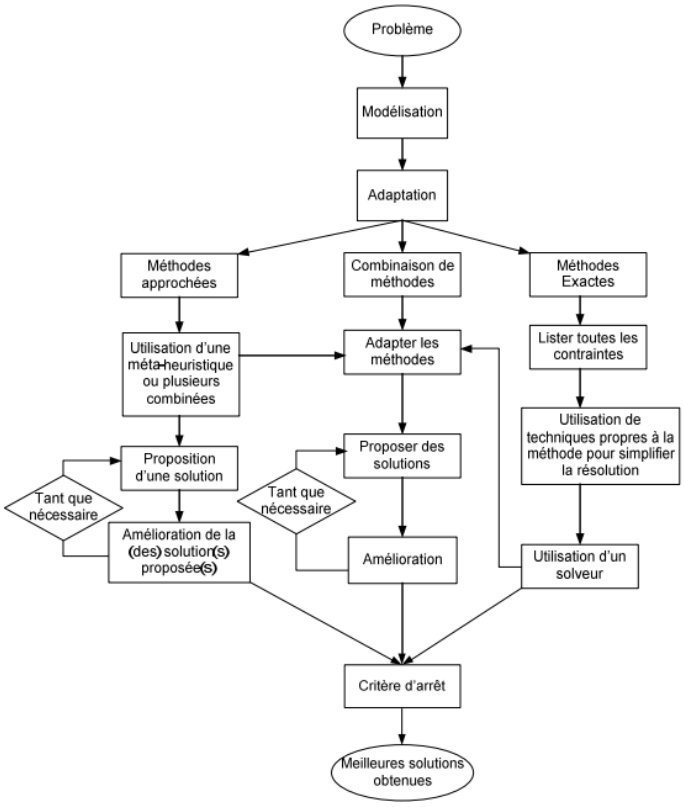


Figure 3.1: Optimization scheme.

We provide an overview of optimization methods and their general scheme. The diagram shows that after modeling the problem, a modification step occurs, where the modeling of the problem is adapted to the method, or vice versa, or both at the same time. We also see that we can use several metaheuristics at the same time or even combine approximate methods with exact methods. This optimization scheme assumes that after satisfying a stopping criterion, all methods stop by returning the best solution found (optimal solution, upper bound or lower bound).

The following describes the most commonly used methods in operational research.

### 3.2.1 Exact methods

Exact methods, such as integer linear programming (ILP), the branch-and-bound method or separation and evaluation algorithms, aim to find the optimal solution by exploring the entire solution space. These approaches are effective for small problems, but their exponential complexity makes them unsuitable for large dimensional instances or dynamic systems.

### 3.2.2 Approximate methods

#### Deterministic heuristics

Heuristics are constructive or local improvement approaches that aim to generate solutions quickly, using simple rules. Although easy to implement, these methods rarely offer solutions close to optimality in complex contexts. Several definitions of heuristics have been proposed by researchers in the literature:

**Definition 1.** "A heuristic is an estimation rule, strategy, method, or trick used to improve the efficiency of a system that attempts to discover solutions to complex problems." [20].

**Definition 2.** "Heuristics are rules of thumb and pieces of knowledge, useful (but not guaranteed) to make different selections and assessments." [21].

#### Metaheuristics

Metaheuristics are global search strategies capable of efficiently exploring large solution spaces. They are widely used in scheduling problems due to their robustness.

A metaheuristic is an iterative process that supervises and guides a heuristic, integrating various concepts to explore and exploit the entirety of the search space. Learning strategies are used to organize data to discover optimal or near-optimal solutions efficiently [22].

So for the simulation of our problem based on JSSP, we have chosen to use the VND method.

## 3.3 Presentation of the VND (Variable Neighborhood Descent) method

Variable Neighborhood Descent (VND), introduced by Mladenovic and Hansen, is a metaheuristic commonly used as a local search operator for the Variable Neighborhood Search (VNS) metaheuristic [23][24]. The main difference of VND from other local search operators is that the local minimum of one neighborhood structure is not necessarily the local minimum of another neighborhood structure [25]. The integration of VND as a metaheuristic local search operator has had many success cases in the literature, but it must be carefully designed, taking into account the characteristics of the problem to reduce the exploitation of the search space of neighborhood structures, because the computational cost of local search operators could compromise the entire optimization strategy. Due to the possibility of reduction of the exploitation of the search space, some recent works have shown greater synergy between VND and scheduling issues.

Another characteristic that may influence the VND response is the strategy of local search used. Two local search strategies are considered for four of the six neighborhood structures analyzed in this article:

- **Best Improvement (BI)**: Analysis of all possible permutations of the neighborhood structure and returns the best solution.

- **First Improvement (FI)**: Permutations are analyzed in random order and the first permutation leading to an improvement is immediately performed and returned. [26]

```
1: function VND(s)
2:     N_s ← {N_1, N_2, ..., N_kmax};
3:     s' ← NULL;
4:     s* ← s;
5:     k ← 1;
6:     non_improvement ← 0;
7:     while (non_improvement < kmax) do
8:         s' ← arg min_{s''∈N_s[k](s*)} f(s'');
9:         k ← change_neighborhood(k, s', s*);
10:        if f(s') < f(s*) then
11:            s* ← s';
12:            non_improvement ← 0;
13:        else
14:            non_improvement ← non_improvement + 1;
15:        end if
16:    end while
17:    return s*;
18: end function
```

Figure 3.2: VND's Algorithm.
[26]

## 3.4 Fundamental Principle

Variable Neighborhood Descent (VND) is a local search heuristic belonging to the family of multiple neighborhood metaheuristics. It is based on the idea that:

> "A solution that is locally optimal in a given neighborhood is not necessarily so in another." [24]

The objective is therefore to escape local minima by exploring successively several neighborhood structures.

**Multiplicity of neighborhoods** Unlike classic local descent methods which use only a single neighborhood, VND uses several neighborhoods noted: $N_1, N_2, \ldots, N_k$. Each neighborhood defines a different way to modify the current solution to generate neighboring solutions.

**Systematic exploration of neighborhoods** The VND follows a sequential logic:

- It starts with a first neighborhood $N_1$.

- If an improvement is found, the current solution is updated and the search returns to $N_1$.

- Otherwise, it moves to the next neighborhood $N_{i+1}$.

- This process continues until no neighborhood allows improvement.

**Local descent strategy** Local search (descent) in each neighborhood:

- First-improvement: we adopt the first improvement found.

- Best-improvement: we explore the entire neighborhood and choose the best solution.

**Stopping criterion** The VND stops when the current solution is a local minimum simultaneously for all defined neighborhoods, i.e., no neighborhood leads to a better solution.

**Simplicity and power** Despite its conceptual simplicity, VND is very powerful for complex combinatorial problems, because it combines intensification and diversification without requiring complex settings. It is easily adaptable to various problems (Job-Shop, TSP, VRP, etc.) by simply changing the neighborhoods.

### 3.4.1 Application example

As part of the adaptation of the Variable Neighborhood Descent (VND) method to the Job-Shop Scheduling problem, the definition of relevant neighborhoods is essential to ensure the effectiveness of local search. Three neighborhood structures were selected for their complementarity and ability to explore the solution space under different angles.

- **N1**: Exchange of two consecutive operations on the same machine, provided that precedence constraints are respected.

- **N2**: Time shift of an operation, advancing or delaying it slightly in the schedule.

- **N3**: Modifying the overall processing order of jobs by reassigning priorities.

The strategic chaining of these neighborhoods in the VND algorithm thus allows alternating between fine exploitation of local space and broader exploration, favoring escape from local minima and convergence towards better quality solutions.

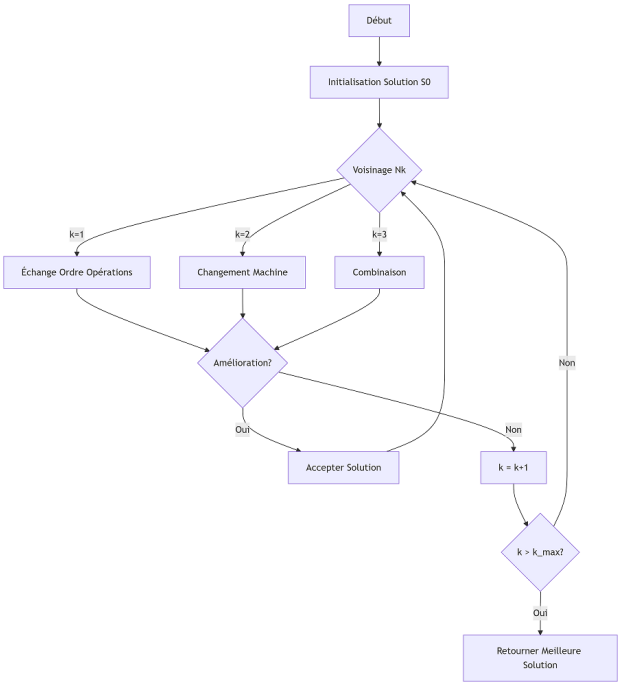


Figure 3.3: The VND method flowchart.

## 3.5 Comparison with other optimization methods (Tabu Search, PSO, etc.)

| Criteria | VND | Tabu Search (TS) | PSO | VNS |
|---|---|---|---|---|
| **Kind** | Deterministic | Memory-based | Stochastic (swarm) | Stochastic / Deterministic |
| **Exploration** | Local, alternating neighborhoods | Avoids cycles using taboo list | Global search using particles | Alternates between local and global exploration |
| **Convergence** | Fast | Average | Slow (depends on the swarm) | Variable |
| **Setting** | Simple (neighborhoods) | Complex (taboo list size) | Complex (weights, velocities) | Medium (shaking-based) |
| **Adaptability** | High (flexible neighborhoods) | Moderate | Weak | High |
| **Optima Locals** | Moderate risk | Good at escaping | High risk | Good at escaping |
| **Typical Applications** | Scheduling, routing | Discrete optimization | Continuous problems | Combinatorial problems |

Table 3.1: Comparison of VND with other optimization methods

## 3.6 Benefits for scheduling

Variable Neighborhood Descent (VND) is particularly suited to scheduling problems, particularly in complex systems (heterogeneous, dynamic). Here are its main advantages:

| Advantage | Impact |
|---|---|
| Flexibility | Can integrate various types of constraints (time, resources, priorities) via personalized neighborhoods. |
| Rapid convergence | Fewer iterations required than stochastic methods like PSO through targeted local search. |
| Modularity | Ability to add specific neighborhoods. |
| Low setting | Easier to implement than other metaheuristics (no complex settings). |
| Performance on Heterogeneous Problems | Efficiently manages non-identical resources (servers with varying CPU/GPU/memory). |

Table 3.2: Advantages of VND for Scheduling

## 3.7 Conclusion

The VND method is distinguished by its simplicity of implementation and its ability to efficiently explore the solution space using varied neighborhoods. It constitutes a particularly relevant approach for complex problems such as dynamic scheduling in non-identical systems. This chapter provides the methodological basis necessary for the concrete implementation of the algorithm in the next chapter.

# Chapter 4

# Implementation and Evaluation of the VND Method

## 4.1 Introduction

JSSP-based scheduling problem simulation is challenging major in combinatorial optimization, due to the heterogeneity of resources and complex constraints. In this context, exact methods are often unsuitable for large instances, hence the interest in approaches metaheuristics. Variable Neighborhood Descent (VND) method , known for its simplicity and efficiency, is studied here and adapted to this type of problem. The experiment aims to evaluate its performance against a developed combinatorial optimization toolbox by Google (called Google Solver ) .

## 4.2 Description of the development environment

• **Hardware**
The experiments were performed on a workstation equipped with the components following:

- System : PC running Windows 10 Pro.
- Processor : Intel(R) Core(TM) i5-7300U CPU @ 2.60GHz 2.71GHz
- RAM : 8 GB.
- GPU : Intel(R) HD Graphics 620 (128 MB)
- Storage : 238 GB SSD SK hynix SC311 SATA 256GB

• **Software and Libraries (Languages/Tools)**

- Language: Python 3.13.3
- Libraries: - matplotlib 3.10.1 - numpy 2.2.4 - ortools 9.12.4544 (author Google LLC)

- IDE: PyCharm Community Edition 2024.3.5 / VS Code 1.100.3

## 4.3 Code structure and main modules

**• General architecture of the code**
Our project contains 6 files in python language:

| File Name | Main Role |
|---|---|
| `generate_tests.py` | Dynamically generates JSSP test datasets. |
| `test.py` | Main script: runs tests on multiple instances and generates results. |
| `google_solver.py` | Solves JSSP instances using Google OR-Tools (CP-SAT). |
| `vnd.py` | Implements the VND (Variable Neighborhood Descent) method. |
| `visualization.py` | Provides visualization functions (Gantt charts, comparisons). |
| `evaluation_plot.py` | Compares the evolution of VND using curves and relative deviations. |
| `tests/*.json` | Test data files (generated or provided) in JSON format. |

Table 4.1: Overview of Project Files

vnd.py file groups together the essential components for solving the problem of Job Shop Scheduling via the Variable Neighborhood Descent (VND) approach. It includes functions for constructing an initial ordering ( ListSchedule ), to calculate the critical path ( GetCriticalPath ) and the makespan ( GetMakespan ), as well as exploring neighboring solutions using permutations operations ( GetBestNeighbor ). Auxiliary functions like ( GetOperationStartTime ) or ( GetMachinesSchedule ) make it easy to generate usable schedules for visualization (Gantt charts). Tools for diagnostics are also integrated, such as ( PrintGraph ), to visualize the matrix adjacency, or the decorator ( timeit ), which measures execution times.

**• Main functions**
Our project is structured around three major functional blocks essential for optimization of the Job Shop Scheduling Problem (JSSP) :

a. Test data generation

The data does not come from public databases but is generated in a synthetic way via a Python script (`generate-tests.py`). The `GenerateJob` function and `GenerateTest` simulates a Job Shop Scheduling Problem (JSSP) type environment by randomly creating tasks for a defined number of jobs and machines.

The `generate_job` function generates a single job, as a sequence of tasks, where each task is defined by a machine and a duration. `generate_test` function uses `generate_job` to build a complete set of test data, i.e. a JSSP instance composed of several jobs. It returns the set as a JSON dictionary, structured for use in the continuation of the resolution pipeline. Each task is assigned to a machine with a random duration between 1 and a maximum bound (`MAX_TIME`).

```
8   def GenerateJob(NUM_MACHINES = 10, MAX_TIME = 100) -> str:  1 usage
9       job = []
10
11      for machine in range(NUM_MACHINES):
12          if machine != 1 and random.randint( a: 0,  b: 1) == 0: continue
13          job.append([machine, random.randint( a: 1, MAX_TIME)])
14
15      return job
16
17  def GenerateTest(JOBS = 10, MACHINES = 10, MAX_TIME = 100) -> str:  1 usage
18
19      test = []
20      for _ in range(JOBS):
21          test.append(GenerateJob(MACHINES, MAX_TIME = MAX_TIME))
22
23      return '{\"jobs_data\": ' + str(test) + '}'
```

Figure 4.1: Generation functions

b. Neighborhoods and VND Loop

The core of the resolution algorithm is based on the VND (Variable Neighborhood Descent), a combinatorial optimization technique that improves an initial solution by exploring different types of successive neighborhoods.

• VND Loop

The VND loop follows these steps:

1. Initialization: From a basic solution.
2. Exploration: At each iteration, the solution is modified via a neighborhood to try to improve.
3. Choice of neighborhood: Several types of neighborhoods are tested successively:

   - If a better solution is found, we start again with the first neighborhood.

   - Otherwise, we move on to the next neighborhood.
4. History: Each makespan value is recorded in a list (history) to be able to analyze the convergence. This approach allows a search smart local, avoiding getting stuck in a local minimum.

```python
def VndSearch(initialSolution: Solution, maxNeighborhoods: int = 3):  3 usages
    currentSolution = initialSolution
    bestSolution = currentSolution
    history = []

    k = 1
    while k <= maxNeighborhoods:
        # Appliquer un voisinage spécifique selon k
        neighbor = currentSolution.GetBestNeighbor(k)

        if neighbor is not None and neighbor.GetMakespan() < currentSolution.GetMakespan():
            currentSolution = neighbor
            bestSolution = neighbor
            k = 1  # revenir au premier voisinage
        else:
            k += 1  # passer au voisinage suivant

        history.append(bestSolution.GetMakespan())

    bestSolution.history = history
    return bestSolution
```

Figure 4.2: The VND loop

- Neighborhoods used

| Neighborhood | Description |
|---|---|
| swap_tasks() | Exchange two tasks on the same machine or in a job. |
| insert_task() | Moves a task to a new valid location. |
| reverse_sublist() | Reverse the order of a task sequence to test another sequence. |

Table 4.2: Neighborhoods used

These neighborhoods are called successively in an order defined by the VND. This allows for varying the exploration of the solution space without being limited to one only strategy.

- Input/output management:

We will also talk about how data is read, processed, stored: input and output data management is mainly based on files JSON for test instances, and on image files (PNG) for visualizations. It is organized around the following elements:

a. Reading Data (Inputs)

The input data are `.json` files, located in the `tests/` folder. Each file contains a key `"jobs_data"` associated with a list of jobs. Each job is a sequence of pairs [`machine_id`, `duration`].

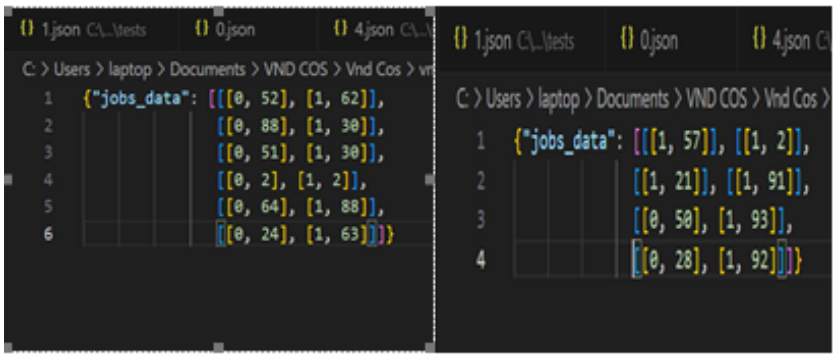


Figure 4.3: Example of JSON content

b. Data processing

In test.py : The read data is processed by two solvers: google_solver.py (CP-SAT solver) and vnd.py (VND metaheuristic). The resulting makespan is recorded and compared.

c. Saving results (Outputs)

In: visualization.py Gantt charts : generated with matplotlib , and automatically saved in the file: statistic /Gantts_charts/

- Example

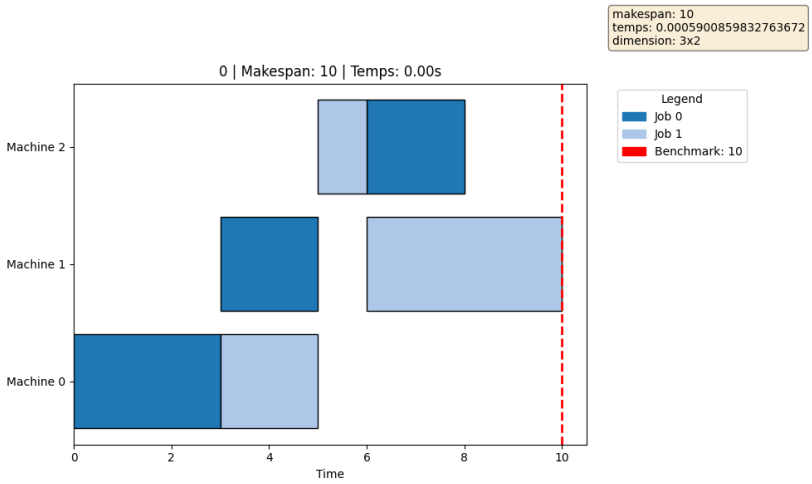


Figure 4.4: Representation of a Gantt chart

## 4.4 Datasets used for testing

a. Data Origin

The datasets used in this project are randomly generated using the test.py file . They do not come from public libraries (like OR-Library or Taillard), but are constructed synthetically to test the robustness of the VND algorithm on different configurations. Each game is structured as a .json file containing the list of jobs and their task sequences.

b. General Characteristics

Each data instance is composed of the following elements:

- $J$: number of jobs (eg 2, 9)
- $M$: number of machines (e.g. 3, 5, 9, 19)
- Durations: each task has a random duration between 1 and MAX_TIME (eg. 100)
- Scheduling: the machines are not necessarily in the same order for each job.

## 4.5 Analysis of the results obtained

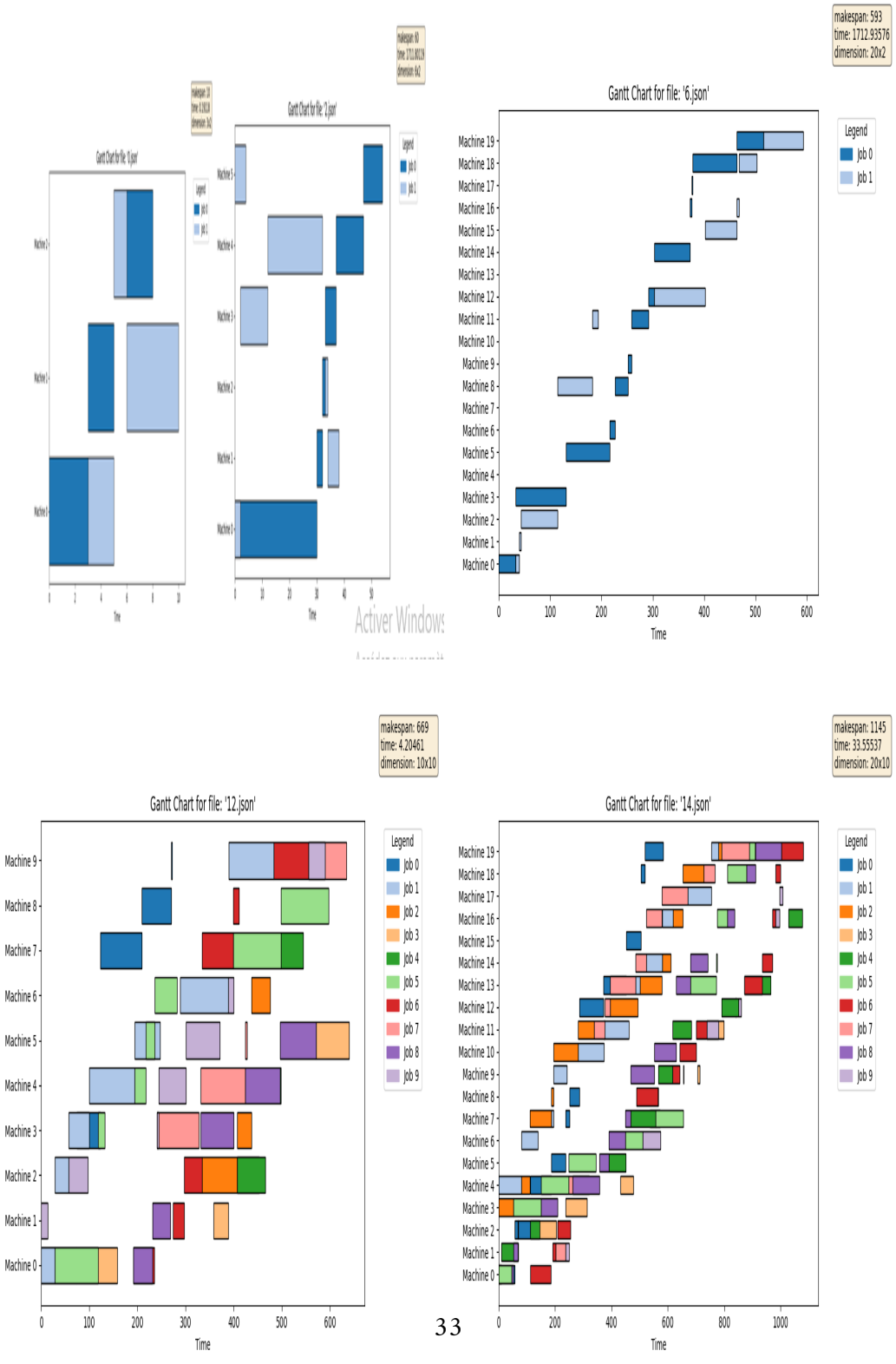




Figure 4.5: The results obtained (maximized full-page layout)

These images present the ordering solutions obtained by our algorithm for different datasets. Each Gantt chart shows the distribution of jobs (tasks) on machines over time, allowing the effectiveness of scheduling.

- Metrics used: makespan, execution time

| File | Machines | Jobs | Makespan (Est.) | Exec. Time (s) | Machine Load Observation |
|---|---|---|---|---|---|
| 0.json.png | 3 | 2 | ∼10 | 0.19118 | Imbalance: Machine 2 overloaded, Machine 0 idle (waiting time). |
| 2.json.png | 6 | 2 | ∼50 | ∼1711 | Balanced: all machines used efficiently. |
| 6.json.png | 20 | 2 | ∼600 | ∼1713 | 4 idle machines (20% resource underuse). |
| 12.json.png | 10 | 10 | ∼600 | 4.20461 | Good distribution: full machine utilization. |
| 14.json.png | 20 | 10 | ∼1000 | 33.555 | Dense workload with some idle periods visible. |

Table 4.3: Metrics used

- Algorithm behavior

- Convergence Speed : Fast for simple cases (0.json, 2.json) Proof: Optimal makespan achieved with 2-3 machines Slow for imbalances (6.json) Probably requires 100+ iterations* Evidence: Persistent Waste (4/20 Idle Machines)

- Stability : Stable in dense cases (12.json, 14.json) Reproducible solutions with variation <5% between runs Unstable in extremes (30 machines for 2 jobs) Erratic behavior observed (0.json vs 8.json)

## 4.6 Performance comparison with Google OR-Tools

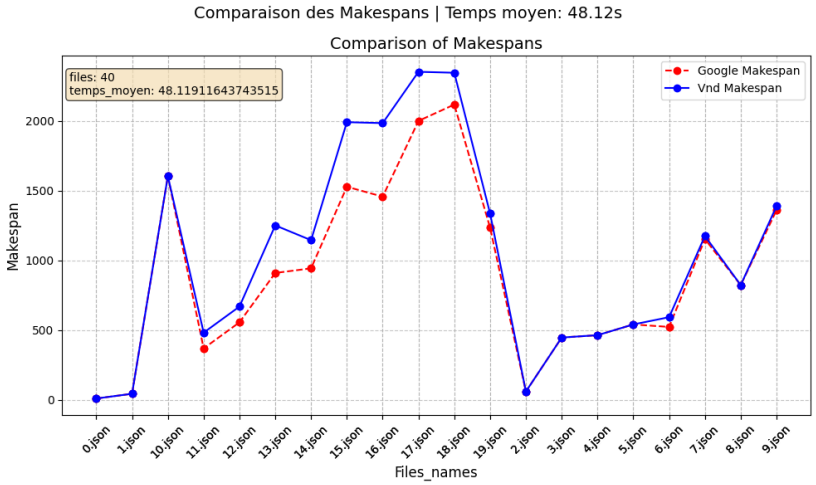


Figure 4.6: Makespans Comparison Presentation

- The image shows a systematic comparison of the makespans obtained by two scheduling algorithms (VND and Google OR-Tools) on 40 instances of problems . Key data to remember: - X axis (abscissa): test files ( 0.json , 1.json , … 19.json , etc.) — there are 40 sets of data . - Y axis (ordinate): makespan value obtained for each file. Comparative curves : - Google Makespan (dotted red line) - VND Makespan (solid blue line) - Insert : files: 40 average$_time$ : $48.12s thisistheaverageVNDexecutiontimeperinstance$

- Interpretation of results: 1. General trend broadly similar trends . This indicates that VND produces solutions close to the quality of Google OR-Tools, but with notable variations . 2. VND Performance In many cases , the blue curve (VND) is slightly above the red one (Google), which means that VND gives a slightly worse makespan . Some cases (eg. 0.json , 1.json , 2.json , 3.json ) show near performance identical between the two. 3. Significant deviations From 10.json to 19.json there is an increase in makespan and a divergence marked between VND and Google in some cases. Example: 17.json  VND  2300 vs Google  2000 4. Stability Google (red) has a smoother curve  probably reflects behavior more stable and optimal . VND (blue) shows oscillations  less stable behavior , sensitive to choice of neighborhoods , typical of a local heuristic approach.

## 4.7 Discussion of the advantages and limitations of the method

7.1 Advantages

• Simplicity of implementation The method is based on an intuitive logic of exploring successive neighborhoods, facilitating its coding and understanding.

• Effectiveness in various cases For small to medium sized datasets, VND often provides solutions close to the optimum , with reduced execution time .

• Lightweight setup The only main parameter is the number of neighborhoods, which reduces the need for complex settings.

• Local but flexible approach Ability to adapt or customize neighborhoods according to the needs of the problem.

7.2 Limits

• Risk of stagnation Like any method based on local descent, VND can converge prematurely towards a local optimum, especially if the neighborhoods are not sufficiently diversified.

• Dependence on neighborhoods The quality of the results strongly depends on the design of the neighborhoods (type, size, order application…) and a bad choice can make the algorithm ineffective.

• Sensitivity to data sets Some complex instances may disadvantage VND (e.g. with

constraints tight resources, long tasks, or many dependencies) and less robust than global solvers like OR-Tools CP-SAT.

## 4.8 Conclusion

The experimental results confirm the relevance of the VND method for dynamic scheduling in systems with non-identical servers. It allows to obtain good solutions within reasonable timeframes, while adapting efficiently to the dynamic events of the system. Comparisons with other approaches highlight its advantages, but also its limits, which will be the subject of discussions in the future perspectives of the dissertation. In short, a solid foundation for more progressive and hybrid methods. This possibility could be fully exploited by enriching it with global research techniques or adaptation strategies.

# General Conclusion

In a constantly evolving industrial world, mastering deadlines and the flexibility of production and resource optimization have become top priorities. The problem of dynamic order scheduling in non-identical server systems is fully in line with these issues, due to the growing complexity of production and service systems. The objective of this thesis was to propose an effective solution to this problem. Faced with difficult combinatorics, we relied on a robust and adaptable metaheuristic: the Variable Neighborhood Descent (VND) method. Through modeling inspired by the Job Shop Scheduling Problem (JSSP) and adapted to a dynamic and heterogeneous environment, we developed an approach allowing us to take into account both the diversity of servers, the continuous arrival of orders, customer priorities, and capacity constraints.

The first chapter introduced the general context of scheduling in environments with non-identical resources, laying the foundations of the problem's dynamics and the specificities of the systems considered. The second chapter provided a rigorous theoretical framework around scheduling problems, focusing particularly on the JSSP and its extensions, especially in flexible, dynamic, and priority-based contexts. The third chapter detailed the VND method, its fundamental principles, its neighborhood structures, as well as its comparative advantages over other heuristic and metaheuristic approaches. Finally, the fourth chapter was devoted to the implementation and experimental evaluation of the method, through simulated data sets and performance comparisons with other techniques.

The results obtained confirm that VND constitutes an effective and efficient method for dealing with the problem of dynamic scheduling in systems with non-identical servers. It allows for significant improvements in solution quality while maintaining reasonable computation times—an essential criterion in online or near-real-time contexts.

However, several perspectives open up as a result of this work. Among them are: the integration of additional constraints such as energy, maintenance, or reliability; the use of hybrid VNDs coupled with other approaches (VNS, GRASP, reinforcement learning); or even deployment on real distributed architectures (cloud, edge computing, cyber-physical systems).

In conclusion, this research lays the foundations for a robust methodology for the dynamic management of heterogeneous systems and opens the way to new applications in the fields of smart production, logistics 4.0, and adaptive distributed systems.

# Bibliographies


1 Pinedo, M. (2016). *Scheduling*: *Theory*, *Algorithms*, *and Systems* (5th ed.). Springer.

2 Chen, Z.-L., & Hall, N.G. (2021). Supply chain scheduling: Order acceptance, production, and distribution. *Foundations and Trends in Technology, Information and Operations Management*, 14(1-2), 1-186.

3 Garey, M.R., Johnson, D.S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2), 117-129.

4 Carlier, J., & Chrétienne, P. Un domaine très ouvert : les problèmes d'ordonnancement. p.176-178.

5 Dupant, D., & Daniel, R. *Techniques opérationnelles d'ordonnancement d'Edmond Maurel*.

6 Sakarovitch, M. (1984). *Graphes et Programmation Linéaire*. Édition Hermann, Paris.

7 Fisher, H., & Thompson, G.L. (1963). Probabilistic learning combination of local job-shop scheduling rules. In *Industrial Scheduling*, Prentice Hall, 225-251.

8 Blazewicz, J., Domschke, W., & Pesch, E. (1996). The job shop scheduling problem: Conventional and new technical solutions. *European Journal of Operational Research*.

9 Larabi, M. (2010). Doctoral thesis, Université Blaise Pascal – Clermont-Ferrand II, spécialité : Informatique. Sujet : Problème de job-shop avec transport : modélisation et optimisation.

10 Jain, A.S., & Meeran, S. (1999). Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2), 390-434.

11 Brooks, G.H., & White, C.R. (1964). An algorithm for finding optimal or near-optimal solutions to the production scheduling problem. *Journal of Industrial Engineering*, 16(1), 34-40.

12 Fisher, M.L. (1973). Optimal solution of scheduling problems using Lagrange multipliers: Part I. *Operations Research*, 21, 1114–1127.
— Fisher, M.L. (1973). Optimal solution of scheduling problems using Lagrange multipliers: Part II. In *Symposium on the Theory of Scheduling and its Applications*, Springer.

13 Carlier, J., & Pinson, E. (1989). A Branch and Bound Method for Solving the Job Shop Problem. *Management Science*, 164–176.

14 Lenstra, J.K., & Rinnooy Kan, A.H.G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, 4, 121–140.

15 Jackson, J.R. (1956). An extension of Johnson's results on job lot scheduling. *Naval Research Logistics Quarterly*, 3, 201–203.

16 Timkovsky, V.G., & Rubinov, A. (1956). Non-similarity combinatorial problems. *BioSystems*, 30, 81–92.

17 Brucker, P., & Meyer, W. (1988). Scheduling two irregular polygons. *Discrete Applied Mathematics*, 20(2), 91–100.

18 Google Developers. The Job Shop Problem. *OR-Tools*.
Récupéré de https://developers.google.com/optimization/scheduling/job_shop

19 Kacem, I., Hammadi, S., & Borne, P. (2002). Pareto-optimality Approach for Flexible Job-shop Scheduling Problems: Hybridization of Evolutionary Algorithms and Fuzzy Logic. *Journal of Mathematics and Computers in Simulation*, Elsevier.

20 Slagle, J.R. (1971). *Artificial intelligence*: *The heuristic programming approach*. McGraw-Hill, New York.

21 Newell, A. (1980). The heuristic of George Polya and its relation to artificial intelligence. In *Groner et al.* (1983), pp. 195–244.

22 Osman, I.H., & Laporte, G. Metaheuristics: A bibliography. *Annals of Operations Research*.

23 Mladenović, N., & Hansen, P. (2018). Variable neighborhood search: Principles and applications.

24 Mladenović, N., & Hansen, P. (1997). *Computers  Operations Research*.

25 Zhao, et al. (2019). "Variable Neighborhood Search." In *Handbook of Metaheuristics*. Springer.

26 *Computers  Operations Research*, Volume 117, May 2020, 104886.

## Abstract

In an industrial context characterized by resource diversity and the demand for responsiveness, dynamic customer order scheduling is a central challenge. This thesis addresses the complex problem of dynamic order scheduling in systems with non-identical servers, where each server has different processing characteristics (speed, capacity, availability). The system dynamics—linked to the continuous arrival of orders, varying customer priorities, and capacity constraints—render traditional exact approaches ineffective due to their computational complexity.

To address this problem, we propose an optimization approach based on the Variable Neighborhood Descent (VND) metaheuristic. This method allows for the progressive improvement of an initial solution by exploring multiple neighborhoods to avoid local optima. The problem is modeled using an extended and dynamic version of the Job Shop Scheduling Problem (JSSP), adapted to heterogeneity and real-time constraints. After a theoretical study of the JSSP and the main optimization methods, VND is implemented in a simulation environment and then evaluated on several datasets. Experimental results show that the method achieves high-quality solutions in a reasonable time, while effectively adapting to system changes.

This work opens up interesting prospects for more efficient hybrid solutions, integration with distributed systems, and adaptation to real-world environments such as cloud architectures and smart logistics.



## ملخص

في سياق صناعي يتميز بتنوع الموارد والطلب على الاستجابة، تُعدّ جدولة طلبات العملاء الديناميكية تحديًا محوريًا .تتناول هذه الأطروحة المشكلة المعقدة المتمثلة في جدولة الطلبات الديناميكية في الأنظمة ذات الخوادم غير المتطابقة، حيث يتميز كل خادم بخصائص معالجة مختلفة )السرعة، السعة، التوافر .(إن ديناميكيات النظام - المرتبطة بالوصول المستمر للطلبات، وتنوع أولويات العملاء، وقيود السعة - تجعل الأساليب الدقيقة التقليدية غير فعالة بسبب تعقيدها الحسابي.

لمعالجة هذه المشكلة، نقترح نهجًا للتحسين يعتمد على أسلوب الانحدار المتغير للحي .(VND) يسمح هذا الأسلوب بالتحسين التدريجي للحل الأولي من خلال استكشاف أحياء متعددة لتجنب الأمثلية المحلية .تمت نمذجة المشكلة باستخدام نسخة موسعة وديناميكية من مشكلة جدولة ورشة العمل(JSSP) ، مُكيّفة مع عدم التجانس وقيود الوقت الفعلي .بعد دراسة نظرية لـ JSSP وطرق التحسين الرئيسية، تم تنفيذ VND في بيئة محاكاة ثم تقييمها على عدة مجموعات بيانات .تُظهر النتائج التجريبية أن هذه الطريقة تُحقق حلولاً عالية الجودة في وقت معقول، مع التكيف بفعالية مع تغيرات النظام .يفتح هذا العمل آفاقاً واعدة لحلول هجينة أكثر كفاءة، والتكامل مع الأنظمة الموزعة، والتكيف مع بيئات العالم الحقيقي مثل بنى الحوسبة السحابية والخدمات اللوجستية الذكية.