AI-Enhanced Integration of Production Planning and Vehicle Routing in Logistics Operations: A Hybrid Metaheuristic Framework

immediate

October 31, 2025

Abstract

Purpose - Integrating production planning on unrelated parallel machines with the vehicle routing problem in modern logistics operations presents significant challenges. This study addressed this integration to optimize operational efficiency and meet customer demands.

Design/Methodology/Approach -

This study presents a comprehensive solution framework that combines exact and heuristic methods. First, we formulate a Mixed Integer (MIP) Programming model and develop two tailored heuristics—a constructive heuristic and a neighborhood search heuristic—to generate high-quality solutions. Next, we propose a machine learning–based selection mechanism that predicts, for each problem instance, which heuristic will perform best. Finally, we introduce two novel metaheuristics: a randomized variable neighborhood descent (RVND) algorithm and a hybrid framework (PPO-VND) that integrates variable neighborhood descent (VND) with the Proximal Policy Optimization (PPO) reinforcement learning algorithm. Together, these contributions offer a robust, adaptive toolkit for solving complex instances across diverse settings.

Findings - Comparative tests demonstrate the superiority of our proposed methods over existing approaches, confirming their effectiveness in addressing the critical challenges posed by integrated production planning and vehicle routing in logistics operations.

Originality/Value - Our work highlights the key role of Artificial Intelligence (AI) in logistics optimization, promoting adaptability and resilience in dynamic operational environments.These innovations significantly improve solution quality and computational efficiency.

Keywords: Integrated Production Planning Problem; Parallel Machine Scheduling; Vehicle Routing Problem; Hybrid Metaheuristic; Unsupervised Learning; Reinforcement Learning

1 Introduction

In recent years, global supply chains have faced unprecedented disruptions, highlighting the vulnerability of logistics networks to external shocks such as pandemics, geopolitical conflicts, and fluctuations in demand. These challenges have underscored the urgent need for more resilient, adaptive, and integrated production and distribution systems, capable of maintaining high service levels under uncertainty.

Traditionally, production planning and distribution routing have been treated as separate optimization problems, often leading to suboptimal solutions that fail to capture the synergies of joint decision-making. However, with the advent of Industry 4.0 and the increasing digitalization of industrial processes, the boundaries between production and logistics are becoming increasingly blurred. Integrating these domains is now recognized as a key lever for achieving operational excellence, cost savings, and enhanced customer satisfaction in highly competitive markets.

Despite significant advances, existing approaches frequently rely on simplified models or heuristic methods that may not scale well or deliver the level of responsiveness required by decentralized, real-time logistics environments. Moreover, while metaheuristics have demonstrated success in tackling large-scale combinatorial problems, the potential of Artificial Intelligence—particularly machine learning and reinforcement learning—for orchestrating adaptive and intelligent logistics operations remains largely untapped.

Our study delves into the intricacies of decentralized production systems, where the seamless integration of production planning and distribution is paramount. Unlike conventional methodologies primarily focused on cost reduction and resource optimization, our research shines a spotlight on enhancing customer service levels. By prioritizing the minimization of total weighted delay in product delivery to customers, our study aligns with the evolving demands of contemporary logistics operations. Leveraging insights from JIT principles and AI-driven optimization, we navigate the complexities of Industry 4.0, fostering synchronized

supply chains and driving operational efficiency. Through a holistic approach that emphasizes service quality and responsiveness, our research contributes to the ongoing discourse on optimizing logistics operations within the dynamic landscape of Industry 4.0.

The potential contributions of this article are significant. First, we introduce a Mixed Integer Linear Programming (MILP) model that solves the integrated problem of production planning on unrelated parallel machines and the routing of capable vehicles. Furthermore, we present an initial solution algorithm and eight neighborhood search heuristics (NSH). The study also presents an innovative framework based on machine learning that can predict the most effective NSH heuristic to solve each instance. Furthermore, a hybrid metaheuristic combines the heuristic Variable Neighborhood Descent (VND) with the reinforcement learning algorithm Proximal Policy Optimization (PPO), significantly improving solution quality and computational efficiency. Finally, detailed comparisons demonstrate the superiority of the proposed AI approaches over the others presented, with the framework standing out better than NSH and the PPO-VND standing out in instances of greater complexity. These contributions demonstrate the relevance and innovation of optimizing logistics operations.

This paper is organized as follows: Section 2 presents a literature review, Sections 3 and 4 present the research problem and describe the mathematical model, respectively. Section 5 presents the Constitutive Heuristics, the Neighborhood Search Heuristics, the framework, the Proximal Policy Optimization, the metaheuristic Random Variable Neighborhood Search (RVND), and the hybrid metaheuristic PPO-VND. In Section 6, we present the computational experiments conducted for this research. Finally, Section 7 concludes the study.

2 Literature Review

Integrated production and distribution planning problems, extensively studied and exemplified by researchers such as (Chen, 2004); (Chen, 2010) and practical applications showcased by (Christian A Ullrich, 2013a), (Yağmur and Kesen, 2024), (Mohammadi *et al.*, 2020), (Boudia *et al.*, 2008), and (Tamannaei and Rasti-Barzoki, 2019), are at the forefront of addressing contemporary logistics complexities. In the context of Industry 4.0, where the Just in Time (JIT) system emerges as a solution to the challenges of operating without stock, seamless connectivity throughout the entire supply chain is imperative. The main idea of the JIT system is to create only what is necessary when it is necessary and with the appropriate resources. This approach helps minimize excess inventory and lowers storage costs. However, its mastery hinges upon the interconnectedness, alignment, and readiness of the entire supply chain to fulfill demands promptly and efficiently, underscoring the pivotal role of Artificial Intelligence (AI) in optimizing logistics operations within the Industry 4.0 framework.

The principles of Just in Time (JIT) production, exemplified by Toyota's pioneering adoption and subsequent success in the automotive industry (Ghinato, 1995); (Monden, 2011); (Sugimori *et al.*, 1977), underscore the tangible benefits of streamlined operations within manufacturing and assembly industries. As JIT systems minimize inventories, waste, and idle time while enhancing efficiency and operational effectiveness, they align seamlessly with the challenges of contemporary logistics highlighted in the preceding paragraph. In the era of Industry 4.0, where decentralized production systems demand interconnectedness and agility, JIT principles resonate deeply, emphasizing the need for synchronized supply chains and optimized processes. This symbiotic relationship between JIT methodologies and the logistics landscape underscores the vital role of Artificial Intelligence (AI) in orchestrating seamless operations, driving efficiency, and ensuring the timely delivery of goods and services amidst evolving complexities.

We provide an overview featuring articles related to integrating production planning with the vehicle routing problem in Table I. Drawing on the insights from various authors who have explored integrated production and distribution planning systems, our study delves into the challenges posed by decentralized production systems. (Nagano *et al.*, 2022) contribute to this discourse by investigating an integrated production and distribution problem within a flow shop system and capable vehicle routing. Their objective, to sequence orders and minimize the makespan, underscores the importance of streamlined operations and efficient resource utilization. In tandem with these endeavors, our research emphasizes optimizing customer service levels. By integrating insights from such studies with JIT methodologies and AI-driven optimization techniques, we strive to address the complexities of Industry 4.0, fostering agile and responsive supply chains.

Aligned with previous research endeavors, (Martins *et al.*, 2021) investigated an analogous integrated problem, focusing on a hybrid flow-shop configuration for production. Their study highlights makespan optimization using a mixed integer linear programming model and a biased random variable neighborhood descent (BR-VND) algorithm. Such insights contribute to the ongoing discourse on integrated production and distribution planning systems, paving the way for enhanced operational efficiency and resource utilization within contemporary logistics frameworks.

Table I: Overview of articles related to integrating production planning with the vehicle routing problem and similar to our work. The table includes columns specifying the machine configuration, type and number of vehicles in VRP, authors' names, and checkmarks indicating whether the article contains one or more mathematical models, heuristics, metaheuristics, and artificial intelligence.

Machine Configuration	Vehicle Routing Problem Type	Number	Author(s)	Math Model	Heuristic	Metaheuristic	IA
Flexible flowshop	Scheduling problem only		Zhu et al. (Zhu *et al.*, 2020)	yes	yes	yes	yes
Hybrid Flow Shop	Scheduling problem only		Nahhas et al. (Nahhas *et al.*, 2022)	no	no	no	yes
Vehicle Routing Problem only	Homogeneous	Single	Peng et al. ((Peng *et al.*, 2020)) Nazari et al. ((Nazari *et al.*, 2018))	no	no	no	yes
Flow shop	Homogeneous	Limited	Hou et al. ((Hou *et al.*, 2022))	yes	no	yes	no
Flow shop	Homogeneous	Single	Ta et al. (Ta *et al.*, 2015)	yes	yes	no	no
Permutation flow-shop	Homogeneous	Single	Naganoa et al. ((Nagano *et al.*, 2022))	yes	no	yes	no
Hybrid flow-shop	Homogeneous	Single	Martins et al. ((Martins *et al.*, 2021)), Wang et al. ((S. Wang *et al.*, 2020))	yes	no	yes	no
Flexible job-shop	Heterogeneous	Limited	Mohammadi et al. (Mohammadi *et al.*, 2020)	yes	no	yes	no
Single machine	Homogeneous	Limited	Tamannaei et al. (Tamannaei and Rasti-Barzoki, 2019), Liu et al. (Liu *et al.*, 2020)	yes	yes	yes	no
Single machine	Heterogeneous	Limited	Felix et al. (Felix and José E. C. Arroyo, 2020), Arroyo et al. (Félix *et al.*, 2023), Zou et al. (Zou *et al.*, 2018)	yes	yes	yes	no
Identical parrallel machines	Heterogeneous	Limited	Yagmur et al. (Yağmur and Kesen, 2024)	yes	no	yes	no
Parallel machines with machine-dependent ready times	Heterogeneous	Limited	Ullrich et al. (Christian A. Ullrich, 2013b)	yes	no	yes	no
Unrelated parallel machines with machine-dependent ready times	Heterogeneous	Limited	Araujo et al. (Araujo *et al.*, 2022)	no	yes	no	yes
Unrelated parallel machines with machine-dependent ready times	Heterogeneous	Limited	Our Approach	yes	yes	yes	yes

(Source: Authors own work)

Expanding on previous investigations, such as those conducted by (Martins *et al.*, 2021), and in line with the research by (Hou *et al.*, 2022) addressing an integrated problem within a distributed flow shop and multi-depot vehicle routing environment, Hou et al. present an enhanced brainstorming optimization (EBSO) algorithm to minimize total weighted advances and delays, comparing its results with various other algorithms, including the discrete artificial bee colony (DABC) algorithm (Duan *et al.*, 2018), iterative greedy algorithm (IG) (Mao *et al.*, 2021), genetic algorithm (GA) (X. Zhang *et al.*, 2020), memetic algorithm (MA) (Kurdi, 2020), and scatter search algorithm (SS) (Pan *et al.*, 2019). These approaches collectively contribute to advancing operational efficiency and optimization within contemporary logistics paradigms.

In addition to the previously mentioned studies, other researchers have explored integrated production and distribution problems under various configurations. For instance, (Christian A Ullrich, 2013a) examined the integration of machine scheduling and vehicle routing with time windows, highlighting the inherent complexity of coordinating shop floor operations with outbound logistics. This study underscores the importance of aligning production sequences with delivery constraints to achieve overall efficiency.

Building on this, (X. Wang and J. Li, 2013) addressed a related problem that integrates production planning with vehicle routing involving multiple trips, time windows, and uncertain travel times. Their work emphasizes the necessity of accounting for uncertainty and proposes robust optimization models to handle variability in logistical parameters, which is especially relevant in real-world operational settings.

Another critical and emerging line of research involves the incorporation of learning mechanisms into metaheuristics. (Iklassov *et al.*, 2024), for example, introduced a reinforcement learning framework for solving the stochastic vehicle routing problem with time windows. Their results reveal the significant potential of AI-driven algorithms to adapt to dynamic environments and learn efficient routing policies over time—directly inspiring the reinforcement learning component (PPO) employed in our hybrid metaheuristic.

(S. Wang *et al.*, 2020) addressed a three-stage hybrid flow shop scheduling problem incorporating product distribution. Their study involves Stage 1 utilizing a system of identical parallel machines with sequence-dependent setup times, Stage 2 employing dedicated machines, and Stage 3 focusing on the multitrip traveling salesman problem with capable vehicles, all aimed at minimizing the maximum delivery completion time. To tackle this challenge, Wang et al. proposed a mixed integer linear programming model alongside methods based on variable neighborhood search (VNS), a four-layer constructive heuristic method (C HVNS), and a hybrid heuristic method (CONSVNS) that combines the VNS and C HVNS methods. These strategies contribute to optimizing operational efficiency and logistics within complex production and distribution environments.

Our proposed approach leverages several advanced techniques to address the intricate challenges of integrating production planning on unrelated parallel machines with the vehicle routing problem. Firstly, we introduce a robust Mixed Integer Linear Programming Model (MILPM) to provide a solid mathematical foundation for optimization. Additionally, we deploy a Constructive Heuristic and Neighborhood Search Heuristics to explore solution spaces efficiently. To enhance adaptability and efficiency further, we introduce a novel framework integrating machine learning, enabling predictive analysis to determine the most effective Neighborhood Search Heuristics for a given instance based on its unique characteristics. Moreover, we present an innovative RVND metaheuristic alongside a Hybrid Metaheuristic (PPO-VND), which

integrates Variable Neighborhood Descent (VND) with a Reinforcement Learning Algorithm, Proximal Policy Optimization (PPO). These advancements offer tangible gains in solution quality and computational efficiency and highlight the significant advantages of incorporating Artificial Intelligence (AI) into logistics optimization, fostering adaptability and resilience in dynamic operational environments.

The Randomized Variable Neighborhood Descent (RVND) algorithm is a variant of the classical Variable Neighborhood Descent (VND), widely used in combinatorial optimization problems due to its ability to escape local optima. While the traditional VND explores neighborhoods in a fixed, deterministic order, RVND introduces randomness by selecting neighborhoods at random in each iteration (VASCONCELOS *et al.*, 2021). This mechanism allows for more effective diversification of the search, avoiding repetitive patterns and increasing the chances of reaching higher-quality solutions. Furthermore, when no improvement is found after exploring a neighborhood, it is temporarily discarded, allowing for dynamic and adaptive exploration of the solution space. This strategy has been successfully applied in various contexts, such as the vehicle routing problem with time windows, showing promising results in both solution quality and computational time (Vidal *et al.*, 2020). Due to its efficiency and flexibility, RVND has been widely adopted in hybrid metaheuristics, especially for integrated problems like production scheduling and vehicle routing, where combinatorial complexity demands robust intensification and diversification strategies.

These studies collectively highlight the field's evolution from deterministic, sequential methods to integrated, AI-enhanced, and learning-capable frameworks. In this context, our proposal contributes not only by addressing a novel configuration (unrelated parallel machines with integrated vehicle routing) but also by embedding intelligence into the search process itself. Through the use of predictive models to guide heuristic selection and reinforcement learning to adapt the local search dynamics, we offer a methodology that aligns with the demands of Industry 4.0—flexibility, scalability, and resilience in the face of complex, dynamic operational environments.

3 Problem Definition

This work addresses the integration of production planning on unrelated parallel machines with setup and ready times and the vehicle routing problem (VRP). This combined problem arises in real-world contexts where optimizing both production and distribution is critical for efficiency. It involves two interconnected stages: (i) assigning a set of jobs to parallel machines, and (ii) routing vehicles to deliver the finished products to customers. The integration of these stages introduces additional complexity, as decisions in one stage affect the other.

The problem is NP-hard, since it generalizes two known NP-hard problems: the unrelated parallel machine scheduling problem with setup and ready times, as demonstrated by Lin and Hsieh (2014), and the vehicle routing problem, as established by Lenstra and Kan (1981). Therefore, the integrated version naturally inherits this computational complexity.

Formally, a set of jobs $I = 1, 2, ..., n$ must be scheduled on m unrelated parallel machines $M = 1, 2, ..., m$ without preemption. Each job $j \in I$ is characterized by a machine-dependent processing time p_ij on machine i, a due date d_j, a tardiness weight w_j, and a size h_j, representing the space it occupies in a delivery vehicle. Additionally, transitioning from job j to job k on machine i requires a setup time s_ijk, which is generally asymmetric (i.e., $s\_ijk \neq s\_ikj$) (Araujo *et al.*, 2022).After processing, jobs are grouped into batches and delivered to customers using a fleet of l vehicles $L = 1, 2, ..., l$. The routing is modeled on a graph $G(V, A)$, where $V = 0, 1, ..., n$ is the set of nodes (with node 0 representing the depot), and $A = (j, k) \mid 0 \leq j, k \leq n, j \neq k$ is the set of arcs, each with an associated travel time t_jk. Each job corresponds to a customer node, and each vehicle $v \in L$ has a capacity q_v that must not be exceeded. The model assumes that each vehicle is used once and each customer is visited only once. The objective is to determine both the optimal job scheduling on machines and the vehicle delivery routes so as to minimize the total weighted tardiness (TWT) of all jobs (Araujo *et al.*, 2022).

Table II presents the parameters of an example instance with three machines ($m = 3$), six jobs ($n = 6$), and four vehicles ($v = 4$). The setup times between jobs are detailed in Table III. Figure 1 illustrates a solution for this instance, highlighting the completion times, delivery times, and vehicle routes for each job. In this specific instance, the completion and delivery for each job are respectively, Job I_1 21 and 44, job I_2 45 and 62, Job I_3 39 and 59, Job I_4 14 and 36, Job I_5 8 and 58, Job I_6 23 and 61.

In this example, the vehicle routes and schedules are defined as Vehicle V_1 starts at time 14 and delivers jobs I_4 and I_5, following the route: $I_0 \rightarrow I_4 \rightarrow I_5 \rightarrow I_0$, where I_0 represents the depot, Vehicle V_2 starts at time 25 and is responsible for jobs I_1 and I_6, with the route: $I_0 \rightarrow I_1 \rightarrow I_6 \rightarrow I_0$, Vehicle V_3 begins its route at time 45, delivering only job I_2 via: $I_0 \rightarrow I_2 \rightarrow I_0$, Vehicle V_4 starts at time 39 and handles the delivery of job I_3, following the route: $I_0 \rightarrow I_3 \rightarrow I_0$. Among all jobs, only job I_3 was delivered late—by four time units. Given that the tardiness penalty is three units per time unit, the total weighted tardiness (TWT) for this solution is twelve (12).

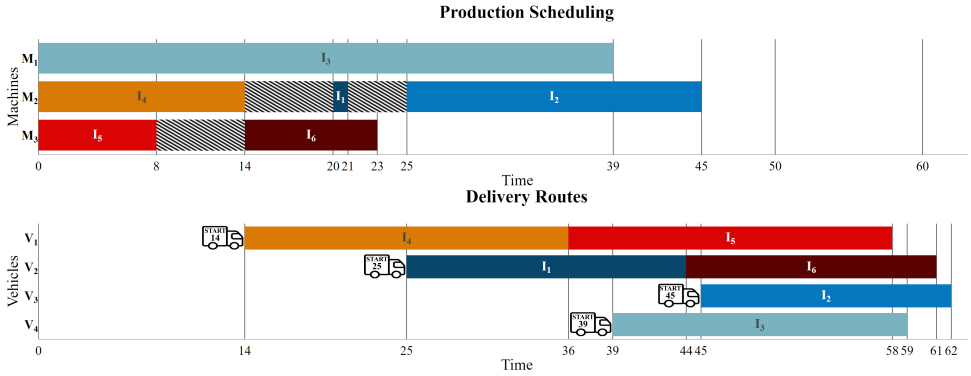


Figure 1: The scheduling of jobs and delivery routes. (Source: Authors' own work)

Table II: Parameters for an example involving three machines, six jobs, and four vehicles.

Jobs	Processing Time			Job Information			Distances							
	M_1	M_2	M_3	h_j	d_j	w_j	Jobs	I_0	I_1	I_2	I_3	I_4	I_5	I_6
I_0	0	0	0	0	0	0	I_0	0	19	17	20	22	22	16
I_1	36	1	43	24	47	10	I_1	19	0	18	34	41	36	17
I_2	46	20	32	27	62	9	I_2	17	18	0	18	33	39	29
I_3	39	24	37	24	55	3	I_3	20	34	18	0	21	38	36
I_4	41	14	32	20	62	2	I_4	22	41	33	21	0	22	33
I_5	7	5	8	5	62	9	I_5	22	36	39	38	22	0	19
I_6	31	45	9	19	61	16	I_6	16	17	29	36	33	19	0
Vehicle information														
Vehicle			V_1			V_2			V_3			V_4		
q_v			58			65			41			72		

(Source: Authors own work)

Table III: Setup time an example involving three machines, six jobs, and four vehicles.

	Setup Time M_1							Setup Time M_2							Setup Time M_3							
Jobs	I_0	I_1	I_2	I_3	I_4	I_5	I_6	Jobs	I_0	I_1	I_2	I_3	I_4	I_5	I_6	Jobs	I_0	I_1	I_2	I_3	I_4	I_5	I_6
I_0	0	0	0	0	0	0	0	I_0	0	0	0	0	0	0	0	I_0	0	0	0	0	0	0	0
I_1	0	0	1	4	5	2	4	I_1	0	0	4	1	4	5	2	I_1	0	0	4	6	2	3	6
I_2	0	5	0	3	6	4	6	I_2	0	4	0	3	4	1	4	I_2	0	4	0	8	4	3	2
I_3	0	2	3	0	3	1	3	I_3	0	6	3	0	7	4	4	I_3	0	7	6	0	6	7	3
I_4	0	4	2	3	0	4	1	I_4	0	6	2	5	0	3	6	I_4	0	2	3	4	0	1	5
I_5	0	1	2	4	6	0	2	I_5	0	3	1	4	3	0	5	I_5	0	1	4	5	1	0	6
I_6	0	3	1	2	5	3	0	I_6	0	2	4	3	5	3	0	I_6	0	5	3	6	3	4	0

(Source: Authors own work)

Table IV: Parameters and the decision variables for a Mixed Integer Linear Programming (MILP) model

Indices	
i	Machine.
j, k	Jobs.
v	Vehicle.
O	Origin (Deposit).
Sets	
$M = \{1, 2, \ldots, m\}$	Set containing m machines.
$I = \{1, 2, \ldots, n\}$	Set containing n jobs.
$N = I \cup \{O\}$	Set containing jobs and a fictitious job from the source.
$L = \{1, 2, \ldots, l\}$	Set containing l vehicles.
Parameters	
p_{ij}	Processing time for job j on machine i.
s_{ijk}	Setup time for job k after job j on machine i.
t_{jk}	Travel time between location j and location k.
w_j	Delay penalty for job j.
d_j	Delivery date for job j.
h_j	Customer demand j (job size j).
q_v	Vehicle capacity v.
Decision variables	
Z_{vjk}	1 - If the vehicle v is used to travel between points j and k. 0 - Otherwise.
X_{ijk}	1 - If job j precedes job k on machine i, 0 - Otherwise.
Y_{ij}	1 - If job j is processed on machine i, 0 - Otherwise.
U_v	1 - If the vehicle v is used to make deliveries. 0 - Otherwise.
C_j	Completion time of job j.
S_v	Vehicle trip start time v.
T_j	Delivery delay for job j.
D_j	Delivery time for job j.

(Source: Authors own work)

4 Mixed Integer Linear Programming Model

This section introduces a Mixed Integer Linear Programming (MILP) model formulated to address the proposed research problem. The parameters and decision variables used in the model are summarized in Table IV.

The model incorporates the following additional parameters:

- MG is a sufficiently large constant: $MG = \sum_{i \in M} \sum_{j \in N} \sum_{k \in N} (t_{jk} + p_{jk} + s_{ijk})$ This value is used to ensure logical consistency in time-related constraints.
- $N = I \cup O$ represents the extended set of jobs, including a dummy job O that models the starting point (origin) of processing on each machine.

The complete mathematical model is presented below:

$$\min \text{ TWT} = \sum_{j \in I} w_j T_j \tag{1}$$

$$\sum_{v \in L} \sum_{k \in N, k \neq j} Z_{vjk} = 1, \quad \forall j \in I \tag{2}$$

$$\sum_{j \in N} \sum_{k \in N, k \neq j} Z_{vjk} h_j \leq q_v U_v, \quad \forall v \in L \tag{3}$$

$$\sum_{k \in N} Z_{v0k} = U_v, \quad \forall v \in L \tag{4}$$

$$\sum_{j \in N} Z_{vjh} = \sum_{k \in N} Z_{vhk} \forall h \in N, \quad \forall v \in L \tag{5}$$

$$\sum_{k \in I} X_{i0k} \leq 1, \quad \forall i \in M \tag{6}$$

$$\sum_{i \in M} Y_{ij} = 1, \quad \forall j \in I \tag{7}$$

$$Y_{ij} = \sum_{k \in N, k \neq j} X_{ijk}, \quad \forall i \in M, \forall j \in I \tag{8}$$

$$Y_{ik} = \sum_{j \in N, k \neq j} X_{ijk}, \quad \forall i \in M, \forall k \in I \tag{9}$$

$$C_k - C_j + MG(1 - X_{ijk}) \geq s_{ijk} + p_{ik}, \quad \forall j \in N, \forall k \in I, j \neq k, \forall i \in M \tag{10}$$

$$S_v \geq C_k - MG(1 - \sum_{j \in N, j \neq k} Z_{vjk}), \quad \forall v \in L, \forall k \in N \tag{11}$$

$$D_k - S_v \geq t_{0k} - MG(1 - Z_{v0k}), \forall v \in L, \quad \forall k \in N \tag{12}$$

$$D_k - D_j \geq t_{jk} - MG(1 - \sum_{v \in L} Z_{vjk}), \quad \forall j \in I, \forall k \in N, j \neq k \tag{13}$$

$$T_j \geq D_j - d_j, \quad \forall j \in I \tag{14}$$

The objective function, defined in Equation (1), aims to minimize the total weighted tardiness of all jobs. The constraints serve specific roles in structuring the solution. Equation (2) ensures that each customer is served exactly once and by a single vehicle, while Equation (3) enforces the vehicle capacity limits, considering that $h\_0 = 0$ for the dummy job. Equations (4) and (5) guarantee that any vehicle used must start and return to the depot. Constraint (6) ensures that only one job is scheduled immediately after the dummy job I_0 on each machine, and constraint (7) ensures that each job is assigned to exactly one machine.

Equations (8) and (9) require that each job has exactly one predecessor and one successor in the production sequence. Constraint (10) defines the completion time for each job, using the base case $C_0 = 0$. Constraint (11) sets the starting time for each vehicle's delivery route. Equations (12) and (13) calculate the arrival times at customer locations. Finally, constraint (14) determines job delays based on their respective due dates.

5 Proposed Algorithms

In this study, we propose two solution-decoding algorithms and a constructive algorithm, complemented by a set of neighborhood search heuristics. To enhance problem-solving efficiency, we develop a machine learning-based framework capable of selecting the most appropriate neighborhood search heuristic for a given problem instance, based on its specific characteristics.

Additionally, we introduce the Proximal Policy Optimization (PPO) algorithm within a Reinforcement Learning context. Building on this, we present the PPO-VND Hybrid Metaheuristic, which integrates the Variable Neighborhood Descent (VND) method with the PPO algorithm. This hybrid approach leverages the adaptive learning capabilities of PPO to guide the neighborhood search process dynamically. The following section provides a detailed explanation of these algorithms and their integration within the proposed solution framework.

5.1 Decoding Algorithms

Although the problem comprises two interdependent components—parallel machine scheduling and vehicle routing—the solution is represented as a pair, $sol = s, r$, where s refers to the job schedule on machines and r to the delivery routes. The construction of a solution relies on two job lists: $L1$ and $L2$. The list $L1$ determines the order in which jobs are assigned to machines using the NEH (Nawaz-Enscore-Ham) decoding algorithm. Upon completion of the NEH decoding, the job scheduling component s is defined. In parallel, the list $L2$ guides the insertion order of jobs into vehicle routes via the PIFH (Push Forward Insertion Heuristic) decoding algorithm, producing the routing component r. Thus, the decoding algorithms collectively yield a complete solution to the problem, denoted as $sol = s, r$. The decoding process is illustrated in Figure 2.

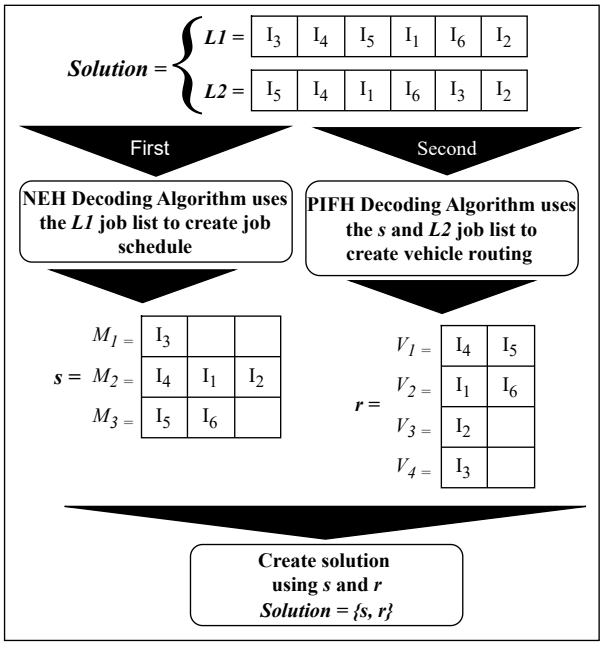


Figure 2: The solution decoding process. (Source: Author's own work)

The NEH heuristic, originally proposed by (Nawaz *et al.*, 1983), is a greedy algorithm widely used in scheduling problems. The NEH decoding algorithm applies this heuristic by sequentially inserting jobs from the $L1$ list onto machines. Each job is placed in the position that minimizes its completion time. Once all jobs are scheduled, the machine assignment s is finalized. The NEH decoding procedure is detailed in Algorithm 1.

Algorithm 1 NEH decoding algorithm.

```
1: function NEH(L1)
2:     s ← ∅                                          ▷ The job scheduling
3:     for i = 1 to n do
4:         Inserts job L1[i] in the best position in the sequencing s
5:     end for
6:     return s
7: end function
```

Solomon's PIFH heuristic (Solomon, 1987) is an efficient algorithm for constructing routes. It uses a greedy approach to sequentially insert jobs into routes. The PIFH decoding algorithm places jobs into routes based on the $L2$ list. In each iteration, a job is inserted in a way that minimizes the total weighted tardiness (wt). After inserting all jobs in the routes, the vehicle routes r are finally defined. The PIFH decoding algorithm is presented in Algorithm 2.

Algorithm 2 PIFH decoding algorithm.

```
1: function PIFH( L2, s )
2:     r ← ∅                                          ▷ Vehicle routes
3:     for i = 1 to n do
4:         Inserts job L2[i] in the best position in the route's r
5:     end for
6:     return r
7: end function
```

5.2 *Initial Solution*

To construct a solution for the problem ($sol = s, r$), the process begins with the definition of the job list $L1$. This list is formed by sorting the jobs in descending order based on the sum of their average processing and setup times. Once $L1$ is defined, the NEH decoding algorithm is applied to determine the job sequencing on machines, resulting in the schedule component s. Following the generation of s, the job list $L2$ is created using the Earliest Due Date (EDD) rule. This list is sorted in ascending order of the difference between each job's due date and its completion time ($d_j - C_j$), prioritizing jobs that are closer to or beyond their due dates. Once the $L2$ list is established, the PIFH decoding algorithm is applied to construct the delivery routes, completing the routing component r. Figure 3 illustrates the overall process of initial solution construction, while Algorithm 3 formally describes the procedure for generating the initial solution to the problem.

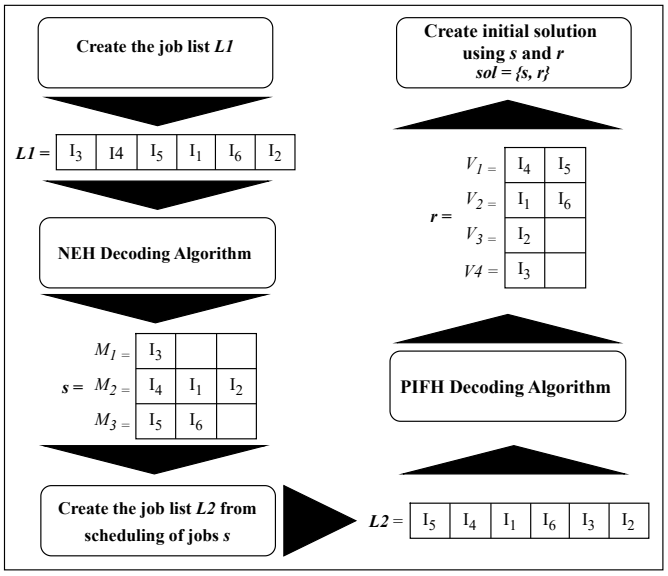


Figure 3: Initial solution construction process. (Source: Authors' own work)

Algorithm 3 Initial Solution.

```
1: function INITIAL SOLUTION( )
2:     Create job list L1
3:     s ← NEH(L1)
4:     Create job list L2 from scheduling of jobs on the machine's s
5:     r ← PIFH(L2, s)
6:     solution ← {s, r}
7:     return solution
8: end function
```

5.3 Neighborhood Search Heuristics

Initially, we developed two distinct algorithms for list manipulation: the Swap Algorithm (Algorithm 4) and the Insertion Algorithm (Algorithm 5). The Swap Algorithm operates by exchanging the positions of two elements within a given list to generate a modified list. In contrast, the Insertion Algorithm functions by removing an element from the list and subsequently reinserting it into an alternative position, thereby creating a new list configuration.

Algorithm 4 Swap Algorithm.

```
1: function SWAP( list, i, j )
2:     aux ← list[i]
3:     list[i] ← list[j]
4:     list[j] ← aux
5:     return list
6: end function
```

Algorithm 5 Insertion Algorithm.

```
1: function INSERTION( list, i, j )
2:     aux ← list[i]
3:     list.remove(aux)
4:     list.insert(j, aux)
5:     return list
6: end function
```

To further improve solution quality, we developed the Neighborhood Search Heuristics (NSH), detailed in Algorithm 6. The NSH iteratively explores potential improvements by applying either swap or insertion movements within the job lists ($L1$ or $L2$). Each generated solution is evaluated, and the algorithm ultimately returns the best solution discovered throughout its execution. The NSH takes two job lists ($L1$ and $L2$) and a set of parameters as input, which dictate its behavior. The movement parameter specifies whether the algorithm will utilize exchange or insertion movements. The list parameter determines which job list ($L1$ or $L2$) these movements will be applied to. Lastly, the restart parameter controls whether the search should restart if a superior solution is identified during the algorithm's execution. These parameters allow for the definition of eight distinct Neighborhood Search Heuristics, with the specific variations and parameter values outlined in Table V.

The NSH algorithm (Algorithm 6) starts by constructing an initial solution from the provided $L1$ and $L2$ job lists. The list and movements parameters then dictate which list and movement type the algorithm will use. If an exchange or insertion is applied to the $L1$ job list, the NEH decoding algorithm generates a new job sequence on the machines, denoted as s'. This s' is then used to create a new $L2'$ job list. Subsequently, the PIFH algorithm utilizes $L2'$ to define the job delivery routes (r'). These new s' and r' ultimately form a new solution.

Conversely, if the exchange or insertion is applied to the $L2$ job list, the initial job sequence on the machines (s) remains unchanged, and the PIFH algorithm defines the delivery routes (r'). In this scenario, s and r' combine to form a new solution. Each newly generated solution is then compared to the current best solution. If the new solution is superior, it replaces the current best. Finally, the restart parameter dictates whether the search should be reset. Upon completion, the algorithm returns the overall best solution identified during its execution.

Table V: Name of Neighborhood Search Heuristics and Parameters.

Name	Parameters		
	Movements	List	Restart
NSH 1	Swap	L2	False
NSH 2	Swap	L2	True
NSH 3	Insertion	L2	False
NSH 4	Insertion	L2	True
NSH 5	Swap	L1	False
NSH 6	Swap	L1	True
NSH 7	Insertion	L1	False
NSH 8	Insertion	L1	True

(Source: Authors own work)

Algorithm 6 Neighborhood Search Heuristics.

function NSH($L1, L2, movements, list, restart$)
  The decoders determine the $solution = \{s, r\}$ from jobs lists $L1$ and $L2$.
  $best\_L1 \leftarrow L1$
  $best\_L2 \leftarrow L2$
  if $list =' 1'$ **then** ▷ Value '1' indicates that the algorithm uses the L1 Job List.
    $L \leftarrow L1$
  else
    $L \leftarrow L2$
  end if
  $i \leftarrow 0$
  while $i \leq len(L)$ **do**
    $Best\_FO$ =F($solution$)
    $j \leftarrow i + 1$
    while $j \leq len(L)$ **do**
      if $movements = SWAP$ **then**
        $L' \leftarrow swap(L, i, j)$
      else
        $L' \leftarrow insertion(L, i, j)$
      end if
      if $list =' 1'$ **then** ▷ Value '1' indicates that L1 Job List is used.
        $s' \leftarrow$ NEH (L')
        Create job list $L2'$ from scheduling of jobs on the machines s'
        $r' \leftarrow$ PIFH ($s', L2'$)
        $solution' \leftarrow \{s', r'\}$
      else
        $r' \leftarrow$ PIFH (s, L')
        $solution' \leftarrow \{s, r'\}$
      end if
      if F($solution'$) <F($solution$) **then**
        $solution \leftarrow solution'$
        if $list =' L1'$ **then**
          $Best\_L1 \leftarrow L$
          $Best\_L2 \leftarrow L2'$
        else
          $Best\_L1 \leftarrow L1$
          $Best\_L2 \leftarrow L$
        end if
      end if
      $j+ = 1$
    end while
    $i+ = 1$
    if $restart$ =TRUE AND $i = len(L)$ AND $Best_FO \neq$F($solution$) **then**
      $L1 \leftarrow Best\_L1$
      $L2 \leftarrow Best\_L2$
      $i \leftarrow 0$
    end if
  end while
  return $solution, Best\_L1, Best\_L2$
end function

5.4 *Framework*

Effectively addressing combinatorial optimization problems often necessitates identifying the algorithm that consistently yields the most favorable outcomes across diverse instances. Traditional approaches, while successful for specific problem instances, may exhibit inconsistent performance when faced with varied inputs. To overcome this limitation, we propose a novel framework that integrates machine learning (ML) to dynamically select the most effective heuristic for a given problem instance. The proposed framework is visually represented in Figure 4.

Our methodology commences with the application of a clustering algorithm to group instances exhibiting similar characteristics. In this work, we employ the k-means algorithm, which has partitioned the instances into 15 distinct categories. Following this clustering, each of the eight Neighborhood Search Heuristics (NSH) was executed on every instance within the training set. For each instance, the heuristic that produced the superior solution was subsequently designated as the representative heuristic for its respective cluster.

Upon analyzing the underlying patterns and rules within the training data, a machine learning model is developed. This model's objective is to predict the optimal heuristic for solving a new problem instance based on its intrinsic characteristics. The model incorporates various instance-specific factors: the number of jobs (n), the number of machines (m), the average processing time as defined by Equation 15, the average setup time as defined by Equation 16, the average travel time as defined by Equation 17, the average demand as defined by Equation 18, and the average vehicle capacity as defined by Equation 19. Utilizing these inputs, the ML model predicts the most suitable NSH for a given instance. The training objective for the machine learning model is to learn a function $h(x) : x \rightarrow Y$ that accurately maps instance characteristics (x) to the corresponding optimal heuristic (Y). Post-training, the model yields a predictor $h(x)$ that, based on the characteristics of a given instance, identifies the most effective NSH for that specific problem instance (Araujo *et al.*, 2022).

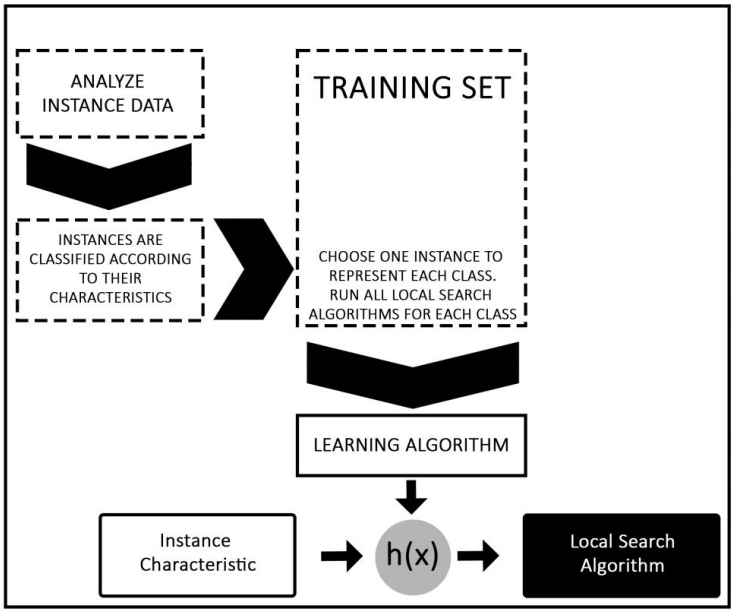


Figure 4: The framework scheme. Source: Araujo *et al.*, (2022).

$$\overline{p} = \frac{\sum_{i \in M, j \in I} p_{ij}}{m * n} \tag{15}$$

$$\overline{s} = \frac{\sum_{i \in M, j \in I, k \in I, j \neq k} s_{ijk}}{(m * n^2) - (m * n)} \tag{16}$$

$$\overline{t} = \frac{\sum_{j \in N, k \in N, j \neq k} t_{jk}}{(n+1)^2 - (n+1)} \tag{17}$$

$$\overline{h} = \frac{\sum_{j \in I} h_j}{n} \tag{18}$$

$$\overline{q} = \frac{\sum_{v \in L} q_v}{l} \tag{19}$$

The core of our predictive framework is a Multilayer Perceptron (MLP) neural network. This MLP is composed of sequential linear layers, with each layer computing its output as a linear combination of its inputs. For training, we employed the Adam algorithm (Kingma and Ba, 2014), a first-order, gradient-based optimization method particularly effective for stochastic objective functions due to its adaptive estimation of lower-order moments. Network performance was evaluated using Mean Squared Error (MSE), which measures the discrepancy between the target output and the network's predicted output (Araujo *et al.*, 2022). While training the neural network required approximately four hours, once trained, the model can recommend the optimal algorithm for any given instance in a remarkable 0.1 seconds.

5.5 The Proximal Policy Optimization

Sequential decision-making problems are commonly addressed through Reinforcement Learning (RL), wherein an agent interacts with a stochastic environment to learn an optimal mapping from states to actions, thereby maximizing cumulative rewards. Such RL problems can be formally modeled as a Markov Decision Process (MDP), defined as a tuple $\{S, A, p, r(s, a)\}$. In this formulation, S denotes the set of states, A represents the set of actions, and p is the transition function, which specifies the conditional probabilities of state changes (ranging from 0 to 1). Furthermore, the reward function $r(s, a)$ assigns a numerical value to each state-action pair. The primary objective of the agent within this process is to discover an optimal policy that associates each state with a corresponding optimal action, ultimately maximizing the sum of rewards accumulated over time (Alves and Mateus, 2020).

The Proximal Policy Optimization (PPO) algorithm, introduced by (Schulman *et al.*, 2017), is an on-policy method rooted in the Policy Gradients (PG) concept, designed to optimize agent policies. PPO leverages a Deep Neural Network (DNN) to map states to actions, enabling Deep Reinforcement Learning (DRL) agents to directly learn the optimal policy while maximizing the cumulative reward within the environment. This distinguishes PPO from alternative methods that first learn a value function and then derive the policy.

In actor-critic methods, which are a class of PG algorithms, both the policy (learned by the actor) and the value function (learned by the critic) are approximated. The critic evaluates the current policy's performance, and minimizing variation during training significantly accelerates the learning process. PPO strikes a balance between implementation complexity, parameterization, and sample efficiency. Its core objective is to constrain the magnitude of policy updates, ensuring that subsequent policies remain sufficiently similar to the current one. This constraint is critical because overly extensive updates can introduce substantial variance into the learning process, potentially leading to suboptimal performance (Alves and Mateus, 2020). For our implementation, we utilize the PPO algorithm as provided by the stable-baselines3 library in Python (Raffin *et al.*, 2021). The pseudocode for the PPO algorithm is presented in Algorithm 7.

Algorithm 7 Proximal Policy Optimization

```
Initialize Θ and Φ                          ▷ policy and value-function parameters, respectively
for i = 0, 1, 2, ... do
    for actor = 1, 2, ..., N_actors do
        Run policy π(Θ) for T steps and collect the trajectories.
        Calculate advantage estimates Â_1, ..., Â_T based on value-function V_Φ.
    end for
    Form a batch, of size NT , with collected trajectories and advantages.
    for epoch = 1, 2, ..., K do
        Shuffle batch and split into minibatches
        for all minibatches do
            Update Θ wrt objective function via stochastic gradient.
            Update Φ wrt mean-squared error of value-function V_Φ.
        end for
    end for
end for
```

5.5.1 State, Action, and Reward Function

In our Reinforcement Learning (RL) framework, the state comprehensively captures the current environment the agent interacts with. This state is represented as a feature vector that describes the environment at a

specific moment, encompassing the number of machines (m), the number of jobs (n), the list of each job's completion times ($C_j, \forall j \in N$), the list of each job's delivery dates ($D_j, \forall j \in N$), and the list of each job's tardiness penalties ($w_j T_j, \forall j \in N$).

The action signifies the agent's decision at a given state, representing the next neighborhood to be explored by the local search. The model's output is an optimal solution policy derived from job lists $L1$ and $L2$, following the agent's chosen action.

The reward function is derived directly from the objective function, calculated using Equation (20). This calculation considers both a lower bound (LB) and an upper bound (UB). The LB is determined using Equations (21). To compute the UB, we make several assumptions: that jobs are distributed identically across machines and vehicles, that job j is the last to be executed on a machine and delivered on a route, and that all jobs have the longest possible processing, setup, and travel times.

$$\text{Reward Function} = -1 * \frac{(\sum_{j \in I} w_j T_j) - LB)}{(UB - LB)} * 100 \tag{20}$$

$$LB = \sum_{j \in I} w_j * Max((Min(p_{ij} \forall i \in M) + t_{0j} - d_j), 0) \tag{21}$$

5.5.2 The Environment Model

An agent's interaction with its environment necessitates a robust environmental model that accurately represents the problem. In this context, our environment models the integrated production and distribution planning problem. The model's input consists of two job lists, $L1$ and $L2$, which are used to construct the problem's solution, denoted as $solution = \{s, r\}$. The model's output is the objective function value of this solution. The agent's overarching goal is to discover a policy that maximizes cumulative rewards over time, thereby yielding superior solutions within the integrated problem. It was essential to create a distinct environmental model for each group of instances characterized by n values of 10, 15, and 30, as the environmental model is intrinsically tied to the number of jobs.

The NEH heuristic sequentially inserts jobs from the $L1$ list into the solution. In each iteration of the algorithm, a job $j \in L1$ is strategically inserted into the machine sequencing to minimize the machine's completion time. This process ultimately defines the final sequencing of machines (s). Subsequently, the PIFH heuristic sequentially inserts jobs from the $L2$ job list with the objective of minimizing the job's weighted delay (wt). Upon completion of this algorithm, the vehicle routes (r) are definitively established. The complete Environment Model is detailed in Algorithm 8.

Algorithm 8 Environment Model

1: $solution$ = New solution$\{s, r\}$.
2: Sort the job sequence as $L1$ in descending order of the sum of average processing and setup times.
3: $s \leftarrow$NEH($L1$)
4: Create job list $L2$ from scheduling of jobs on the machines s (EDD rule)
5: $r \leftarrow$ PIFH ($L2, s$)
6: Create $solution$ with s and r
7: Return $f(solution)$

5.6 PPO - VND

The Variable Neighborhood Descent (VND) algorithm (Hansen and Mladenović, 2001) is a deterministic local search method that systematically explores a predefined set of k neighborhood structures, $V = V_1, V_2, ..., V_k$, until a locally optimal solution S is reached. We introduce a hybrid metaheuristic, the PPO-VND, presented in Algorithm 9, which integrates Proximal Policy Optimization (PPO) to dynamically select and define the Neighborhood Search Heuristics (NSH) at each iteration. The PPO-VND scheme is further illustrated in Figure 5.

The algorithm commences with an initial solution. In each subsequent iteration, the PPO component selects a specific NSH. Upon executing the chosen NSH, a local optimal solution is returned. The state is then updated, and action masking is applied. If the newly found solution is superior to the current best solution, it is updated accordingly; otherwise, the solution is perturbed. This iterative process continues

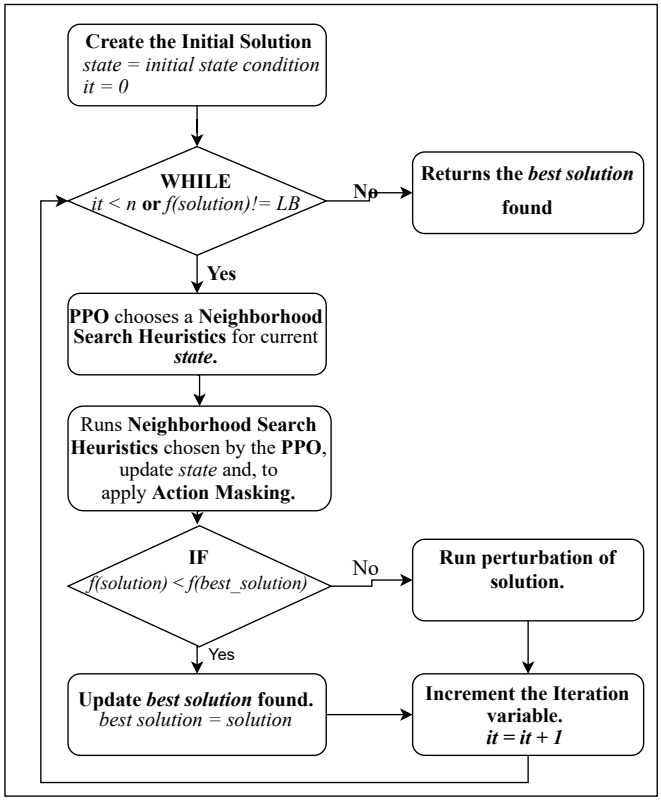


Figure 5: The PPO-VND scheme. (Source: Authors' own work)

until either the number of iterations equals the number of jobs in the instance (n) or the obtained solution matches the established Lower Bound (LB).

Solution perturbation is crucial in this context as it prevents the PPO-VND from becoming trapped in local optima, thereby enabling a more extensive exploration of the search space. This perturbation involves randomly exchanging and inserting movements within 30% of the jobs in either $L1$ or $L2$ (the list being chosen randomly). Specifically, 30% of the jobs are randomly removed and reinserted to create new lists. Subsequently, the NEH and PIFH algorithms utilize these perturbed $L1$ and $L2$ job lists to define a new solution, which is then employed in the subsequent steps of the PPO-VND algorithm. The 30% disturbance parameter was determined through a meticulous calibration process and systematic analysis using the Optuna library (Akiba *et al.*, 2019). This specific value has been shown to offer an optimal balance between exploration and exploitation, facilitating consistent improvements in solution quality without compromising computational efficiency.

6 Computational Experiments

This section outlines the experimental procedures and presents the results used to evaluate the performance of the proposed methods. The Mixed-Integer Linear Programming (MILP) model was implemented in C++ and solved using the CPLEX solver. All other solution methods were developed in Python 3. Computational experiments were performed on a machine equipped with an Intel Core i7-4790K (4.0 GHz) processor and 32 GB of RAM, with all algorithms executed using a single processing thread to ensure consistency and comparability.

For the computational experiments, we utilized the 540 problem instances proposed by (Araujo *et al.*, 2022). Each instance is characterized by two primary parameters: the number of jobs (n), taking values from the set $\{10, 15, 30\}$, and the number of machines (m), selected from $\{2, 4, 8\}$. For instance, "N_10_M_2" denotes an instance with 10 jobs and 2 machines.

The PPO agent was trained for 500,000 episodes using 36 randomly selected instances of varying sizes and complexities. The hyperparameters for the PPO agent were meticulously tuned using the RL Baselines3 Zoo library (Raffin, 2020).

Algorithm 9 PPO - VND

```
solution ← INITIALSOLUTION()
best_solution ← solution
state ← initial state condition
it ← 0
while it < n or f(solution)! = LB do
    u ← PPO.predict(state)                ▷ PPO chooses a NSH for current state.
    solution ← NSH u (solution)
    state ← update
    if F(solution) < F(best_solution) then
        best_solution ← solution
    else
        perturbation of the current solution
    end if
    it ← it + 1
end while
return best_solution
```

The performance of our proposed PPO-VND hybrid metaheuristic is benchmarked against the Random Variable Neighborhood Search (RVND) metaheuristic. RVND is a stochastic local search algorithm (Hansen and Mladenović, 2001) widely applied in various optimization domains, including production planning (Haddad *et al.*, 2014; Yildiz *et al.*, 2023) and vehicle routing problems (Subramanian *et al.*, 2013; Penna *et al.*, 2013). Furthermore, RVND has been successfully employed in solving integrated production and distribution planning problems (Félix *et al.*, 2023). The operational mechanism of RVND is similar to that of PPO-VND; the key distinction lies in RVND's random selection of Neighborhood Search Heuristics, as opposed to PPO-VND's learned selection.

The source code for all algorithms presented in this paper is publicly available at `https://github.com/matheusinhofreitas`. The processing time for the MILP model was set to a fixed duration of 600 seconds.

To quantitatively assess the quality of the obtained solutions, we employed the relative percentage deviation (GAP) from the best-known solution achieved among all methods. The GAP is formally defined by Equation 22.

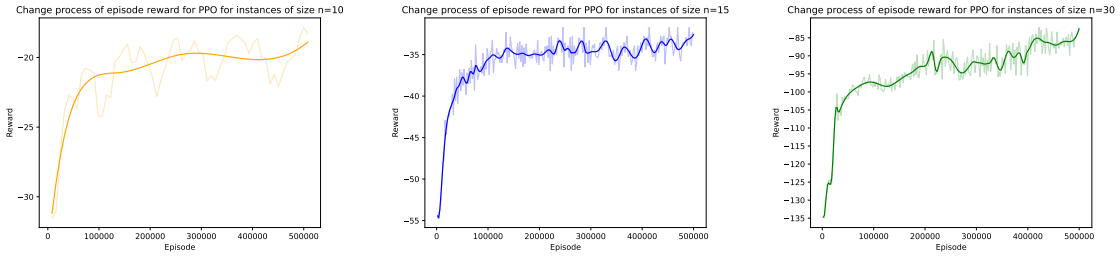
$$GAP = 100 \times \frac{(F_{method} - F_{best})}{F_{method}} \tag{22}$$

Where F_{best} represents the minimum TWT (total weighted tardiness of jobs delays) value obtained among all compared methods, and F_{method} is the TWT obtained with a specific method.

6.1 Computational Results

This section presents a thorough and comprehensive computational analysis, clearly demonstrating the significant advancements achieved by our proposed methodologies compared to existing approaches. The evaluated methods include the proposed Mixed-Integer Linear Programming (MILP) model, the PPO-VND metaheuristic, the RVND metaheuristic, the machine-learning-driven Framework, and the eight Neighborhood Search Heuristics (NSH). Given the inherent stochastic nature of PPO-VND and RVND, each of these algorithms underwent five independent executions per instance, ensuring robustness and reliability of the reported results.

Figures 6a, 6b, and 6c illustrate a consistent increase in the average reward during PPO training, signifying the effective learning process of the Deep Reinforcement Learning (DRL) agents in addressing the chosen problem. Specifically, the orange graph in Figure 6a represents the learning curve for instances with $n = 10$ jobs. Figure 6a demonstrates that the PPO algorithm achieves faster learning for smaller instances, with the reward stabilizing from approximately the first 100 thousand episodes onward. The blue and green graphs (Figures 6b and 6c, respectively) depict the learning curves for instances with $n = 15$ and $n = 30$ jobs. For these larger instances, it is evident that PPO exhibits greater difficulty in learning, requiring a higher number of episodes to stabilize the rewards. This indicates that the complexity of the instances directly correlates with the training time required for the PPO. Despite this increased training complexity, Table VII clearly shows the superior performance of PPO-VND when compared to other methods. Regarding training duration, PPO required approximately 16 hours for instances with $n = 10$, around 74 hours for $n = 15$, and approximately 45 days for $n = 30$. It is important to note that the PPO model's training time was not factored into our performance comparisons, as training is a one-time process.



(a) $n = 10$ (b) $n = 15$ (c) $n = 30$

Figure 6: Evolution of the episode reward during PPO training for different instance sizes. (Source: Authors' own work)

We rigorously confirmed the statistical significance of the performance disparities among algorithms using Analysis of Variance (ANOVA) (Zar, 1999). The resulting p-value was below 0.01, indicating highly significant differences. Figure 7 visually highlights these performance discrepancies by clearly illustrating the distributions of GAPs across all methods. This visualization effectively emphasizes PPO-VND's consistently superior performance, especially in more challenging scenarios.

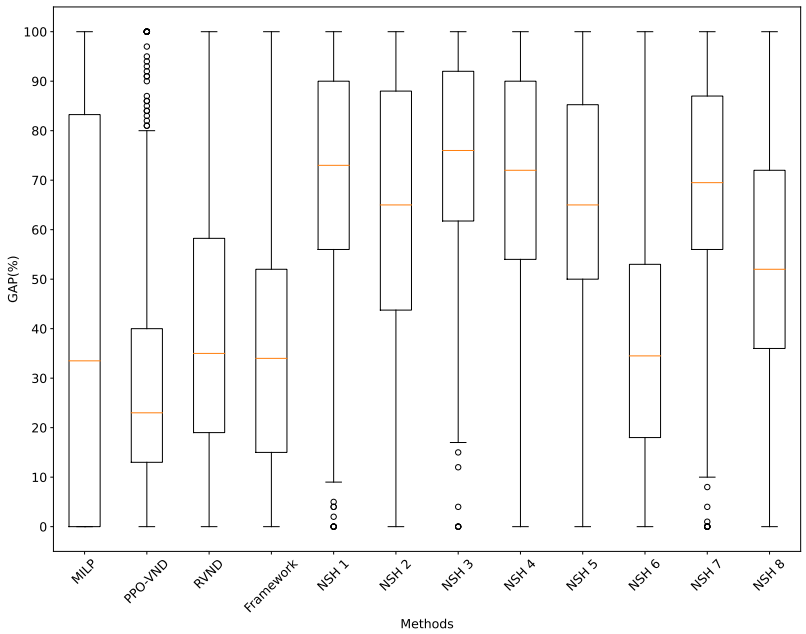


Figure 7: Boxplot of GAPs (%) for all methods. (Source: Authors' own work)

Table VI provides the average GAPs for the eight Neighborhood Search Heuristics. Meanwhile, Table VII displays the average GAPs for the other algorithms discussed in this work. Each row in these tables presents the average GAPs across 60 instances, categorized by the number of jobs and machines. The smallest average GAPs are highlighted in bold. The final row of each table summarizes the average GAPs for the entire set of instances.

As shown in Table VI, NSH 6 consistently delivered the best average performance among all Neighborhood Search Heuristics, achieving an average GAP of 37%. NSH 6's effectiveness stems from its strategy of applying switching movements to the $L1$ job list and restarting the search process whenever a better solution is discovered. When we compare only the NSH variants, NSH 6 found the best solutions for 386 instances, representing 71% of the total. NSH 8 followed with 139 instances (26%), while NSH 2 achieved the best results for 89 instances (16%). The remaining heuristics, NSH 5, NSH 4, NSH 7, NSH 1, and NSH 3, found the best values for a smaller percentage of instances: 47 (9%), 41 (8%), 37 (7%), 32 (6%), and 26 (5%) instances, respectively.

Table VII presents the GAPs achieved by the PPO-VND and RVND algorithms, showcasing their performance across instances of increasing complexity. For smaller instances (e.g., $n = 10$), the Mixed-Integer Linear Programming (MILP) model initially demonstrates superior average GAPs. However, as the

Table VI: Result for the Neighborhood Search Heuristics

Instances	Average GAP							
	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	83%	81%	85%	83%	78%	**49%**	79%	66%
N_10_M_4	73%	69%	76%	73%	65%	**49%**	69%	56%
N_10_M_8	46%	41%	54%	47%	41%	**29%**	44%	34%
N_15_M_2	88%	85%	89%	87%	81%	**38%**	85%	60%
N_15_M_4	71%	64%	75%	69%	65%	**40%**	69%	50%
N_15_M_8	52%	39%	58%	48%	47%	**30%**	54%	36%
N_30_M_2	90%	87%	91%	90%	89%	**33%**	90%	74%
N_30_M_4	75%	64%	77%	72%	70%	**33%**	72%	56%
N_30_M_8	60%	44%	63%	57%	54%	**30%**	58%	43%
Average	71%	64%	74%	70%	65%	**37%**	69%	53%

(Source: Authors own work)

Table VII: Result average GAP for the presented methods.

Instances	Average GAP					Best GAP		Worse GAP	
	MILP	PPO-VND	RVND	Framework	NSH 6	PPO-VND	RVND	PPO-VND	RVND
N_10_M_2	**7%**	41%	59%	49%	49%	34%	45%	46%	70%
N_10_M_4	**5%**	43%	45%	49%	49%	36%	32%	49%	57%
N_10_M_8	**3%**	22%	21%	29%	29%	16%	10%	27%	33%
N_15_M_2	38%	**34%**	65%	36%	38%	22%	51%	40%	75%
N_15_M_4	**21%**	24%	36%	39%	40%	14%	17%	32%	51%
N_15_M_8	37%	26%	**20%**	28%	30%	13%	5%	37%	33%
N_30_M_2	79%	**26%**	69%	31%	33%	4%	56%	43%	77%
N_30_M_4	80%	**24%**	34%	32%	33%	10%	15%	35%	47%
N_30_M_8	90%	**19%**	22%	30%	30%	7%	9%	26%	31%
Average	40%	**29%**	41%	36%	37%	**17%**	27%	**37%**	53%

(Source: Authors own work)

instance complexity escalates due to an increased job count, MILP's effectiveness noticeably diminishes. In contrast, algorithms that leverage machine learning, such as PPO-VND and our overarching Framework, exhibit improved performance as instance sizes expand.

Furthermore, MILP did outperform other methods in specific smaller instance groups, achieving better GAPs in four distinct categories. Specifically, MILP attained an average GAP of 7% for N_10_M_2 instances, 5% for N_10_M_4, 3% for N_10_M_8, and 21% for N_15_M_4. Moreover, MILP successfully identified optimal solutions for 34 instances, which accounts for 6% of all instances tested.

Conversely, the PPO-VND algorithm showcased exceptional scalability and adaptability, clearly differentiating itself as a superior choice for more demanding scenarios. It achieved outstanding GAP results, including 34% for instances n_15_m_2, 26% for n_30_m_2, 24% for n_30_m_4, and an impressive 19% for the largest and most complex scenarios (n_30_m_8). PPO-VND's superior adaptability is not merely theoretical; it is practically demonstrated, illustrating its ability to efficiently tackle large-scale optimization challenges.

Analyzing the data presented in Tables VI and VII, it is evident that both NSH 6 and the Framework consistently outperform the RVND meta-heuristic across four distinct instance groups. This observation underscores the significant advantage of selecting a proficient neighborhood structure over random selection, which is characteristic of RVND. Consequently, these findings highlight the pivotal role of intelligent neighborhood selection in the efficacy of these methodologies and their potential to surpass conventional approaches.

Interestingly, while NSH 6 emerged as the most effective individual Neighborhood Search Heuristic across all datasets, the Framework's performance rivaled or even surpassed NSH 6 in some cases. This provides compelling evidence that employing an algorithm that leverages machine learning to dynamically determine the best heuristic for a specific instance is highly efficient. The Framework's success rate in this experiment was 72%; in other words, for 391 instances (out of a total of 540), the Framework accurately selected the optimal NSH. The results achieved by the Framework were commendable, and in certain instances, they even surpassed those of the RVND meta-heuristic.

Table VIII presents the algorithm execution times across various problem instances, unveiling discernible performance trends. The MILP consistently exhibits lengthier execution times compared to other algorithms, implying its elevated computational complexity. In contrast, PPO-VND and RVND demonstrated commendable scalability with moderate execution times, even as instance complexity increased substantially. Notably, the Framework particularly distinguished itself through negligible runtime increases,

Table VIII: Algorithm execution time in seconds.

Instance	MILP	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	564.70	1.46	1.53	0.57	0.16	0.27	0.30	0.72	0.15	0.29	0.28	0.72
N_10_M_4	547.06	1.21	1.85	0.44	0.19	0.30	0.37	0.84	0.18	0.35	0.34	0.87
N_10_M_8	528.21	2.02	2.38	0.56	0.26	0.39	0.51	1.03	0.26	0.45	0.48	1.23
N_15_M_2	593.68	7.40	11.61	0.90	0.71	1.06	1.38	3.91	0.70	1.24	1.28	3.56
N_15_M_4	599.71	10.12	13.76	1.04	0.83	1.38	1.62	4.83	0.82	1.56	1.53	4.51
N_15_M_8	598.07	17.61	17.68	1.54	1.13	1.83	2.23	5.06	1.13	2.05	2.09	6.23
N_30_M_2	600.57	146.83	300.57	12.96	10.62	16.55	20.61	82.44	10.54	18.23	19.85	63.48
N_30_M_4	600.68	167.22	354.30	15.78	12.29	19.96	24.20	100.94	12.38	21.55	23.28	69.80
N_30_M_8	601.20	269.30	469.47	19.75	16.20	24.62	32.17	113.66	17.78	27.54	30.66	88.10
Averange	581.54	69.24	130.35	5.95	4.71	7.37	9.27	34.83	4.88	8.14	8.87	26.50

(Source: Authors own work)

Table IX: Result average RPI for the presented methods

Instance	Relative Percentage Improvement (RPI)										
	PPO-VND	RVND	Framework	NSH 1	NSH 2	NSH 3	NSH 4	NSH 5	NSH 6	NSH 7	NSH 8
N_10_M_2	**80%**	74%	76%	31%	36%	21%	30%	53%	76%	48%	67%
N_10_M_4	**70%**	69%	63%	35%	42%	25%	35%	51%	65%	43%	58%
N_10_M_8	57%	**59%**	51%	40%	45%	28%	37%	45%	53%	40%	49%
N_15_M_2	**85%**	78%	83%	27%	38%	18%	29%	55%	84%	46%	75%
N_15_M_4	**71%**	68%	65%	27%	39%	17%	29%	43%	65%	33%	57%
N_15_M_8	59%	**62%**	56%	36%	49%	26%	38%	43%	56%	35%	51%
N_30_M_2	**89%**	80%	87%	17%	36%	8%	22%	32%	87%	26%	70%
N_30_M_4	**73%**	70%	68%	17%	40%	9%	23%	35%	68%	29%	53%
N_30_M_8	**60%**	59%	54%	20%	43%	13%	25%	31%	54%	24%	44%
Average	**71%**	69%	67%	28%	41%	18%	30%	43%	67%	36%	58%

(Source: Authors own work)

underscoring its potential for efficient application in real-world contexts.

A holistic evaluation, considering overall GAP outcomes, decisively positions PPO-VND as the top-performing methodology, boasting an exceptional average GAP of 29%. The framework follows, achieving a GAP of 36%. NSH 6 ranks third with a GAP of 37%. The remaining algorithms are classified as follows: MILP (40%), RVND (41%), NSH 8 (53%), NSH 2 (64%), NSH 5 (65%), NSH 7 (69%), NSH 4 (70%), NSH 1 (71%), and finally, NSH 3 (74%).

To evaluate the percentage of improvement caused by the heuristic relative to the initial solution, the Relative Percentage Improvement (RPI) was calculated. The RPI metric is defined by the equation 23:

$$RPI(\%) = 100 \times \frac{(F_{s.i.} - F_{method})}{F_{s.i.}} \tag{23}$$

Where $F_{s.i.}$ is the TWT of the initial solution provided by the constructive heuristic, and F_{method} the TWT obtained with a specific method.

Table IX presents the RPI(%) values for all evaluated methods. The results show that the evaluated methods, PPO-VND, RVND, and Framework, achieved significant improvements compared to the initial solution. The PPO-VND method found the best solutions for the groups N_10_M_2 (80%), N_10_M_4 (70%), N_15_M_2 (85%), N_15_M_4 (71%), N_30_M_2 (89%), N_30_M_4 (73%), and N_30_M_8 (60%), followed by RVND, which found the best values for the groups N_15_M_8 (59%) and N_15_M_8 (62%). Overall, PPO-VND had the highest average RPI (71%), followed by RVND with 69%, Framework with 67%, and NSH. The results of PPO-VND highlight its consistent ability to significantly improve initial solutions, especially in more complex instances.

In conclusion, these computational results robustly confirm the remarkable effectiveness, adaptability, and scalability of PPO-VND and the Framework. The innovations presented herein significantly advance the state-of-the-art, establishing these methods as powerful tools for solving complex, integrated scheduling and routing challenges in logistics.

7 Conclusion

This study proposes a suite of innovative methodologies designed to address integrated production scheduling and distribution problems. The contributions include a Mixed-Integer Linear Programming (MILP) model, a hybrid heuristic known as PPO-VND, eight distinct neighborhood search heuristics, and a machine learning-based framework for selecting the most suitable Neighborhood Search Heuristic (NSH) for a given problem instance. Central to this approach is the use of the Proximal Policy Optimization (PPO) algorithm from

Reinforcement Learning (RL), which dynamically predicts the most effective NSH for each PPO-VND iteration. This integration offers an adaptive and data-driven solution strategy.

Computational experiments demonstrate that PPO-VND significantly outperforms traditional mathematical programming models and other heuristic algorithms in terms of both computational efficiency and solution quality. It is important to note that the training time for the PPO algorithm and the heuristic selection framework is excluded from the reported execution times, as training is performed only once. The results further reveal that while the MILP model performs well on small instances with limited search spaces, its scalability is limited. As instance complexity increases, MILP's performance deteriorates, making it less suitable for real-world applications characterized by complex and large-scale parameters. In contrast, PPO-VND consistently delivers superior performance in larger instances, particularly when the number of jobs increases to 15 and 30. This highlights the robustness, adaptability, and scalability of PPO-VND in solving complex production and distribution problems.

The proposed methodologies offer significant advantages for industrial environments, particularly in addressing complex production and distribution planning problems. The PPO-VND hybrid metaheuristic, with its exceptional scalability and adaptability, is well-suited to tackle the large-scale optimization challenges frequently encountered in real-world industrial settings. Unlike traditional methods that may falter with increasing instance complexity, PPO-VND demonstrates robust performance, even in highly demanding scenarios, as evidenced by its superior GAP results for larger problem sizes. This capability ensures that businesses can consistently achieve highly optimized solutions for their integrated planning needs, leading to more efficient resource utilization, reduced operational costs, and improved overall productivity. The emphasis on intelligent neighborhood selection, powered by machine learning, moves beyond rigid, predefined heuristics, allowing for more dynamic and effective problem-solving tailored to specific industrial instances.

Furthermore, the integration of machine learning into the decision-making process, as demonstrated by the Framework, offers a significant paradigm shift. By intelligently pinpointing the most effective heuristic for a particular problem instance based on its characteristics, the framework can streamline the optimization process. This eliminates the need for manual trial-and-error in selecting algorithms, saving valuable time and resources. While the initial training of the machine learning model might require some computational effort, its ability to recommend an optimal algorithm in approximately 0.1 seconds post-training highlights its immense potential for efficient application in dynamic industrial contexts. This rapid decision-making capability, coupled with the consistently strong performance across various instance types, translates directly into quicker responses to changing demands, enhanced operational agility, and a competitive edge in fast-paced industrial landscapes.

Disclosure statement:
No potential conflict of interest is reported by the authors

References

Akiba, Takuya *et al.*, (2019). "Optuna: A Next-generation Hyperparameter Optimization Framework", *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Alves, Júlio César and Mateus, Geraldo Robson (2020). "Deep reinforcement learning and optimization approach for multi-echelon supply chain with uncertain demands", *International Conference on Computational Logistics*. Springer, pp. 584–599.

Araujo, Matheus de Freitas, Arroyo, José Elias Claudio, and Nogueira, Thiago Henrique (2022). "Heuristics Assisted by Machine Learning for the Integrated Production Planning and Distribution Problem", *International Conference on Intelligent Systems Design and Applications*. Springer, pp. 120–131.

Boudia, Mourad, Louly, Mohamed Aly Ould, and Prins, Christian (2008). "Fast heuristics for a combined production planning and vehicle routing problem", *Production Planning and Control*, Vol. 19 No. 2, pp. 85–96.

Chen, Zhi-Long (2004). "Integrated production and distribution operations", *Handbook of quantitative supply chain analysis*, pp. 711–745.

— (2010). "Integrated production and outbound distribution scheduling: review and extensions", *Operations research*, Vol. 58 No. 1, pp. 130–148.

Duan, Jun-Hua *et al.*, (2018). "An effective artificial bee colony for distributed lot-streaming flowshop scheduling problem", pp. 795–806.

Felix, Gabriel P. and Arroyo, José E. C. (2020). "HEURÍSTICAS PARA O SEQUENCIAMENTO DA PRODUÇÃO E ROTEAMENTO DE VEÍCULOS COM FROTA HETEROGÊNEA", *LII Simpósio Brasileiro de Pesquisa Operacional*,

Félix, Gabriel P., Arroyo, José E. C., and Freitas, Matheus de (2023). "Iterated Local Search Heuristic for Integrated Single Machine Scheduling and Vehicle Routing", *Third Congress on Intelligent Systems*. Ed. by Kumar, Sandeep *et al.*, Springer Nature Singapore: Singapore, pp. 223–235. ISBN: 978-981-19-9379-4.

Ghinato, Paulo (1995). "Sistema Toyota de Produção: mais do que simplesmente just-in-time", *Production*, Vol. 5, pp. 169–189.

Haddad, Matheus Nohra *et al.*, (2014). "AIV: A heuristic algorithm based on iterated local search and variable neighborhood descent for solving the unrelated parallel machine scheduling problem with setup times", *International Conference on Enterprise Information Systems*. Vol. 2. SCITEPRESS, pp. 376–383.

Hansen, Pierre and Mladenović, Nenad (2001). "Variable neighborhood search: Principles and applications", *European journal of operational research*, Vol. 130 No. 3, pp. 449–467.

Hou, Yushuang *et al.*, (2022). "Modelling and optimization of integrated distributed flow shop scheduling and distribution problems with time windows", *Expert Systems with Applications*, Vol. 187, p. 115827.

Iklassov, Zangir *et al.*, (2024). *Reinforcement Learning for Solving Stochastic Vehicle Routing Problem with Time Windows*, arXiv: 2402.09765 [cs.AI]. **available at**: https://arxiv.org/abs/2402.09765.

Kingma, Diederik P and Ba, Jimmy (2014). "Adam: A method for stochastic optimization", *arXiv preprint arXiv:1412.6980*,

Kurdi, Mohamed (2020). "A memetic algorithm with novel semi-constructive evolution operators for permutation flowshop scheduling problem", *Applied Soft Computing*, Vol. 94, p. 106458.

Lenstra, Jan Karel and Kan, AHG Rinnooy (1981). "Complexity of vehicle routing and scheduling problems", *Networks*, Vol. 11 No. 2, pp. 221–227.

Lin, Yang-Kuei and Hsieh, Feng-Yu (2014). "Unrelated parallel machine scheduling with setup times and ready times", *International Journal of Production Research*, Vol. 52 No. 4, pp. 1200–1214.

Liu, Ling *et al.*, (2020). "A coordinated production and transportation scheduling problem with minimum sum of order delivery times", *Journal of Heuristics*, Vol. 26 No. 1, pp. 33–58.

Mao, Jia-yang *et al.*, (2021). "An effective multi-start iterated greedy algorithm to minimize makespan for the distributed permutation flowshop scheduling problem with preventive maintenance", *Expert Systems with Applications*, Vol. 169, p. 114495.

Martins, Leandro do C *et al.*, (2021). "Combining production and distribution in supply chains: The hybrid flow-shop vehicle routing problem", *Computers & Industrial Engineering*, Vol. 159, p. 107486.

Mohammadi, S., Al-e-Hashem, S.M.J. Mirzapour, and Rekik, Y. (2020). "An integrated production scheduling and delivery route planning with multi-purpose machines: A case study from a furniture manufacturing company", *International Journal of Production Economics*, Vol. 219, pp. 347–359. ISSN: 0925-5273. **doi**: https://doi.org/10.1016/j.ijpe.2019.05.017. **available at**: https://www.sciencedirect.com/science/article/pii/S0925527319301987.

Monden, Yasuhiro (2011). *Toyota production system: an integrated approach to just-in-time*, CRc Press.

Nagano, M *et al.*, (2022). "Solution methods for the integrated permutation flowshop and vehicle routing problem", *Journal of Project Management*, Vol. 7 No. 3, pp. 155–166.

Nahhas, Abdulrahman, Kharitonov, Andrey, and Turowski, Klaus (2022). "Deep Reinforcement Learning Techniques for Solving Hybrid Flow Shop Scheduling Problems: Proximal Policy Optimization (PPO) and Asynchronous Advantage Actor-Critic (A3C)",

Nawaz, Muhammad, Enscore Jr, E Emory, and Ham, Inyong (1983). "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem", *Omega*, Vol. 11 No. 1, pp. 91–95.

Nazari, Mohammadreza *et al.*, (2018). "Reinforcement learning for solving the vehicle routing problem", *Advances in neural information processing systems*, Vol. 31.

Pan, Quan-Ke *et al.*, (2019). "Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem", *Expert Systems with Applications*, Vol. 124, pp. 309–324.

Peng, Bo, Wang, Jiahai, and Zhang, Zizhen (2020). "A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems", pp. 636–650.

Penna, Puca Huachi Vaz, Subramanian, Anand, and Ochi, Luiz Satoru (2013). "An iterated local search heuristic for the heterogeneous fleet vehicle routing problem", *Journal of Heuristics*, Vol. 19 No. 2, pp. 201–232.

Raffin, Antonin (2020). *RL Baselines3 Zoo*, https://github.com/DLR-RM/rl-baselines3-zoo.

Raffin, Antonin *et al.*, (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations", *Journal of Machine Learning Research*, Vol. 22 No. 268, pp. 1–8. **available at**: http://jmlr.org/papers/v22/20-1364.html.

Schulman, John *et al.*, (2017). "Proximal policy optimization algorithms", *arXiv preprint arXiv:1707.06347*,

Solomon, Marius M (1987). "Algorithms for the vehicle routing and scheduling problems with time window constraints", *Operations research*, Vol. 35 No. 2, pp. 254–265.

Subramanian, Anand, Uchoa, Eduardo, and Ochi, Luiz Satoru (2013). "A hybrid algorithm for a class of vehicle routing problems", *Computers & Operations Research*, Vol. 40 No. 10, pp. 2519–2531.

Sugimori, Yutaka *et al.*, (1977). "Toyota production system and kanban system materialization of just-in-time and respect-for-human system", *The international journal of production research*, Vol. 15 No. 6, pp. 553–564.

Ta, Quang Chieu, Billaut, Jean-Charles, and Bouquard, Jean-Louis (2015). "Heuristic algorithms to minimize the total tardiness in a flow shop production and outbound distribution scheduling problem", *2015 International conference on industrial engineering and systems management (IESM)*. IEEE, pp. 128–134.

Tamannaei, Mohammad and Rasti-Barzoki, Morteza (2019). "Mathematical programming and solution approaches for minimizing tardiness and transportation costs in the supply chain scheduling problem", *Computers & Industrial Engineering*, Vol. 127, pp. 643–656.

Ullrich, Christian A (2013a). "Integrated machine scheduling and vehicle routing with time windows", *European Journal of Operational Research*, Vol. 227 No. 1, pp. 152–165.

— (2013b). "Integrated machine scheduling and vehicle routing with time windows", *European Journal of Operational Research*, Vol. 227 No. 1, pp. 152–165. ISSN: 0377-2217. **doi**: https://doi.org/10.1016/j.ejor.2012.11.049. **available at**: https://www.sciencedirect.com/science/article/pii/S0377221712009010.

VASCONCELOS, AM, SOUZA, MJF, and SOUZA, SR DE (2021). "Algoritmo GVNS Híbrido Aplicado ao Problema das p-Medianas Capacitado", *Trends in Computational and Applied Mathematics*, Vol. 22 No. 3, pp. 453–473.

Vidal, Thibaut, Laporte, Gilbert, and Matl, Piotr (2020). "A concise guide to existing and emerging vehicle routing problem variants", *European Journal of Operational Research*, Vol. 286 No. 2, pp. 401–416. ISSN: 0377-2217. **doi**: https://doi.org/10.1016/j.ejor.2019.10.010. **available at**: https://www.sciencedirect.com/science/article/pii/S0377221719308422.

Wang, Shijin *et al.*, (2020). "Variable neighborhood search-based methods for integrated hybrid flow shop scheduling with distribution", *Soft Computing*, Vol. 24 No. 12, pp. 8917–8936.

Wang, X. and Li, J. (2013). "Detecting communities by the core-vertex and intimate degree in complex networks", *Physica A*. Vol. 392, pp. 2555–2563.

Yağmur, Ece and Kesen, Saadettin Erhan (2024). "Integrated production scheduling and vehicle routing problem with energy efficient strategies: Mathematical formulation and metaheuristic algorithms", *Expert Systems with Applications*, Vol. 237, p. 121586. ISSN: 0957-4174. **doi**: https://doi.org/10.1016/j.eswa.2023.121586. **available at**: https://www.sciencedirect.com/science/article/pii/S0957417423020882.

Yildiz, Seyda Topaloglu, Ozcan, Sel, and Cevik, Neslihan (2023). "Variable neighborhood search-based algorithms for the parallel machine capacitated lot-sizing and scheduling problem", *Journal of Engineering Research*, p. 100145.

Zar, Jerrold H (1999). "Biostatistical analysis. 4th", *New Jersey, USA*, p. 929.

Zhang, Xin, Li, Xiang-Tao, and Yin, Ming-Hao (2020). "An enhanced genetic algorithm for the distributed assembly permutation flowshop scheduling problem", *International Journal of Bio-Inspired Computation*, Vol. 15 No. 2, pp. 113–124.

Zhu, Jialin, Wang, Huangang, and Zhang, Tao (2020). "A deep reinforcement learning approach to the flexible flowshop scheduling problem with makespan minimization", *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*. IEEE, pp. 1220–1225.

Zou, Xuxia *et al.*, (2018). "A coordinated algorithm for integrated production scheduling and vehicle routing problem", *International Journal of Production Research*, Vol. 56 No. 15, pp. 5005–5024.