DOKUZ EYLÜL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

A SOLUTION APPROACH FOR THE DISTRIBUTED NO-IDLE FLOWSHOP SCHEDULING PROBLEM WITH DUE WINDOWS

by

Kasra MOUSIGHICHI

January, 2023

İZMİR

A SOLUTION APPROACH FOR THE DISTRIBUTED NO-IDLE FLOWSHOP SCHEDULING PROBLEM WITH DUE WINDOWS

A Thesis Submitted to the
Graduate School of Natural And Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in
Industrial Engineering Department

by

Kasra MOUSIGHICHI

January, 2023
İZMİR

M.SC THESIS EXAMINATION RESULT FORM

We have read the thesis entitled "**A SOLUTION APPROACH FOR THE DISTRIBUTED NO-IDLE FLOWSHOP SCHEDULING PROBLEM WITH DUE WINDOWS**" completed by **Kasra MOUSIGHICHI** under supervision of **Assoc. Prof. Dr. Mualla Gonca Avcı** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

.....

Assoc. Prof. Dr. Mualla Gonca Avcı

Supervisor

.....	.....
Assoc. Prof. Dr. Leyla Demir	Assoc. Prof. Dr. Fehmi Burçin Özsoydan
Jury Member	Jury Member

Prof. Dr. Okan Fıstıkoğlu
Director
Graduate School of Natural and Applied Sciences

To all people fled their home . . .

ACKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor, Mualla Gonca Avci, this thesis would not have been possible without your input, insightful feedback, and supervision. And my mother and brother, I cannot forget to thank you for all the unconditional support throughout my life, I would never have been where I am without you two.

Izmir, December 2022

Kasra Mousighichi

A SOLUTION APPROACH FOR THE DISTRIBUTED NO-IDLE FLOWSHOP SCHEDULING PROBLEM WITH DUE WINDOWS

ABSTRACT

This study addresses an extension of the Distributed Permutation Flowshop Scheduling Problem with no-idle and due window constraints. The Distributed No-idle Flowshop Scheduling Problem with Due Windows (DNIFSPDW) entails a number of jobs to be completed at a number of factories. The DNIFSPDW's goal is to identify the job assignments to the factories and the job sequences within each facility that yield the lowest total weighted early and late (TWET) penalties. The DNIFSPDW concerns the industries where setup operations are so expensive that reactivating the machines is not cost-effective. Therefore, any idle time between two consecutive jobs on a machine is prohibited. In addition, each job is associated with a due window indicating its earliest and latest completion times. In the related literature, there exists no study that proposes a mathematical formulation or a solution approach for DNIFSPDW to the best of our knowledge. In this thesis, three mathematical formulations are developed for the DNIFSPDW. Moreover, to tackle large-sized DNIFSPDW problems a hybrid iterated greedy-tabu search algorithm (IG-TS) is proposed. In the computational study, the performances of the proposed mathematical models were analyzed. Additionally, extensive numerical experiments were conducted to evaluate the components of IG-TS. Furthermore, the performance of IG-TS was compared with that of a basic iterated greedy algorithm with a local search (IG-LS). The results of the computational study indicate the effectiveness of the proposed IG-TS in solving the DNIFSPDW.

Keywords: The distributed permutation flowshop scheduling problem, no-idle constraints, due windows, total weighted earliness and tardiness, iterated greedy algorithm

ZAMAN PENCERELİ DAĞITIK BEKLEMESİZ AKIŞ TİPİ ÇİZELGELEME PROBLEMİ İÇİN BİR ÇÖZÜM YAKLAŞIMI

ÖZ

Bu çalışma dağıtık permütasyon akış tipi çizelgeleme probleminin boşta olmama ve zaman penceresi kısıtları ile genişletilmiş bir halini dikkate almaktadır. Zaman Pencereli Dağıtık Beklemesiz Akış Tipi Çizelgeleme Problemi (ZPDBATÇP), bir fabrika kümesinde işlenecek bir dizi işi içerir. ZPDBATÇP'nin amacı, toplam ağırlıklı erken tamamlanma ve gecikme cezalarını minimum kılacak iş-fabrika atamalarını ve her fabrikadaki iş sıralamalarını belirlemektir. ZPDBATÇP, ayar işlemlerinin çok pahalı olduğu ve makineleri durdurup yeniden çalıştırmanın maliyet açısından etkin olmadığı sektörlerle ilgilidir. Bu nedenle, bir makinede birbirini izleyen iki iş arasındaki herhangi bir boş zamana izin verilmemektedir. Ek olarak, her iş, en erken ve en geç tamamlanma zamanlarını gösteren bir zaman penceresiyle ilişkilendirilir. İlgili literatürde, bilgimiz dahilinde ZPDBATÇP için matematiksel bir formülasyon veya çözüm yaklaşımı öneren herhangi bir çalışma bulunmamaktadır. Bu tezde, ZPDBATÇP için üç matematiksel formülasyon geliştirilmiştir. Ayrıca, büyük boyutlu ZPDBATÇP örneklerini çözmek için bir hibrit yinelemeli açgözlü-tabu arama algoritması (HYA-TA) önerilmiştir. Hesaplamalı çalışmada, önerilen matematiksel modellerin performansları analiz edilmiştir. HYA-TA'nın bileşenlerini analiz etmek için kapsamlı sayısal deneyler yapılmıştır. Ayrıca, HYA-TA'nın performansı, temel bir yerel aramalı yinelemeli açgözlü algoritmanın (HYA-YA) performansı ile karşılaştırılmıştır. Hesaplamalı çalışmanın sonuçları, önerilen HYA-TA'nın ZPDBATÇP'nin çözümündeki etkinliğini göstermektedir.

Anahtar kelimeler: Dağıtık permütasyon akış tipi çizelgeleme problemi, boşta olmama kısıtları, zaman pencereleri, toplam erken tamamlanma ve gecikme, yinelemeli açgözlü algoritma

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

Table 1.1 Problem abbreviation

Abbreviation	Definition
FSP	Flowshop Scheduling Problem
PFSP	Permutation Flowshop Scheduling Problem
DPFSP	Distributed Permutation Flowshop Scheduling Problem
DNIFSPDW	Distributed No-Idle Permutation Flowshop Problem with Due Windows
DAPFSP	Distributed Assembly Permutation Flowshop Scheduling Problem
TWET	Total Weighted Earliness and Tardiness

A multi-factory environment is becoming more and more crucial in today's decentralized and globalized economy. As a result, the distributed scheduling problem, or multi-factory production system scheduling, has gained an increasing amount of interest over the course of the last few years (De Giovanni & Pezzella, 2010; Ying et al., 2017) Distributed manufacturing has become a research avenue due to being both a theoretical research area and also having a common application in industry and other real-world settings. One of the key categories is the distributed permutation flowshop scheduling problem (DPFSP), which is a developed version of the classic permutation flowshop scheduling problem (PFSP). The DPFSP problem involves F same factories, which are made up of I machines organized in a flowshop. A solution to this problem comprises two major decisions: the assignment of jobs to each factory, and their production order. It should be noted that Table 1.1 summarizes the abbreviations related to the problem.

The distributed no-idle permutation flowshop problem with due windows (DNIFSPDW) is not investigated in the literature regarding the extensions of flowshop problems. The no-idle constraint prohibits machines from remaining idle after beginning the process of the first job in the sequence (Pan & Ruiz, 2014). This occurs in production settings when machines shut down after the initial setup is not economically feasible due to long setup periods or high running expenses, or even technological limitations may not allow idle time. The steppers used to manufacture integrated circuits using photolithography (Pan & Ruiz, 2014), the production of

ceramic frits (Pan & Ruiz, 2014), production of fiberglass (Kalczynski & Kamburowski, 2005) and foundries (Saadani et al., 2003) are some examples of that idle time is not permitted for machines in the production process.

Common optimization targets in current research avenues involve total flow time (Mao et al., 2022), makespan (Zhao et al., 2022), and total penalty of tardiness (Alidaee et al., 2021), etc. On the other hand, the total weighted earliness\tardiness ($TWET$) as an essential criterion has only been explored seldom Jing et al. (2020). Product completion time in the real manufacturing environment might result in a financial burden. If a job is finished before its earliest due date, the firm will face additional inventory costs and financial burdens. If the firm violates the latest due date, it is required to pay liquidated damages, increasing the expenses (Jing et al., 2020). To the best of our knowledge, due windows have not been introduced into the distributed no-idle permutation flowshop problem. Therefore, this study aims to minimize the $TWET$ for the DPFSP with Due Windows with no-idle time (DNIPFSPDW for short).

The single machine flowshop problem with due windows is known to be NP-Hard (Pan et al., 2017); therefore the DNIPFSPDW can also be identified as an NP-Hard problem since there are F identical factories with I machines for each. For real-size instances, exact solution approaches demonstrate poor performance in terms of computational time. Therefore, metaheuristic algorithms are among the solution approaches with the most promising results for practical instances (Jing et al., 2020). As a metaheuristic solution approach, Iterated Greedy (IG) and its variants have proven to be the most successful of the various metaheuristic algorithms for solving different DPFSPs (Fernandez-Viagas & Framinan, 2015; Huang et al., 2020; Mao et al., 2021). This study contributes to the literature as follows:

- The distribution no-idle flowshop scheduling is addressed by introducing three mathematical models.
- A comprehensive benchmark is provided for this problem for the first time.

- A novel no-idle time adjustment approach is introduced.
- Devised a metaheuristic solution approach to solve large-scale instances of up to 500 jobs.

The remainder of this paper is structured as follows. Chapter 2 briefly explores relevant literature. Chapter 3 provides three MIP models, while Chapter 4 describes the Iterated greedy-based solution algorithm. Chapter 5 includes computational experiments and their respective results. Chapter 6 concludes with some suggestions for further study.

CHAPTER TWO
LITERATURE REVIEW

Table 2.1 Summary of related literature

Abbreviation	Definition	Abbreviation	Definition
IG	Iterated Greedy	COA	Collaborative Optimization Algorithm
MIG	Modified Iterated Greedy	EDA	Effective Estimation of Distribution Algorithm
ICG	Iterated Cocktail Greedy	DABC	Discrete Artificial Bee Colony
IRG	Iterated Reference Greedy	SS	Scatter Search
RIG	Referenced Iterated Greedy	CRO	Chemical Reaction Optimization
AIG	Adaptive Iterated Greedy	ILS	Iterated Local Search
VND	Variable Neighborhood Descent	CMA	Competitive Memetic Algorithm
EDA	Estimate of Distribution Algorithm	EA	Evolutionary Algorithm
BC	Branch-and-Cut	MDDE	Memetic Discrete Differential Evolution
TSMA	Two-Stage Memetic Algorithm	IG-TS	Hybrid Iterated Greedy-Tabu Search

Regarding the available literature, this study is the first to address DNIPFSPDW. Therefore, this section reviews the relevant studies concerning, the DPFSP, the PFSP with due windows, and the DPFSP with no-idle constraints. Table 2.1 indicates the used abbreviations and Table 2.2 summarizes the presented literature on DPFSP and its variants investigating the objective function and the proposed solution approach.

The DPFSP was first presented by Naderi & Ruiz (2010) minimizing the makespan. For the DPFSP, they suggested a total of six different MIP models as well as two different factory-assignment rules referred to as the NEH1 and NEH2 heuristics. Additionally, the authors developed 14 heuristics and two variable neighborhood descent (VND) methods. In addition, they tested the efficacy of the proposed solutions on a total of 720 large instances. Later, Gao & Chen (2011) devised a hybrid Genetic Algorithm - Local Search for the DPFSP with makespan minimization. The authors were able to improve 692 out of 720 instances. In another study, Gao et al. (2013) investigated the known DPFSP by a combined tabu search-enhanced local search and updated the 472 solutions of Taillard instances. Lin et al. (2013) employed a modified IG (MIG) algorithm with makespan criterion. Moreover, the obtained solutions indicate that almost half of the Taillard instances were improved in a lower computational time. Using explicit probability distributions throughout the search process, Wang et al. (2013) devised an estimate of distribution algorithm (EDA) applying a local search. Naderi & Ruiz (2014) improved all 720 large-sized instances solutions by devising a scatter search algorithm. Further studies

are addressing different heuristics as solution approaches for DPFP like (Xu et al., 2014; Fernandez-Viagas & Framinan, 2015; Bargaoui et al., 2017; Ruiz et al., 2019)

As an extension to the DPFSP, Hatami et al. (2013) developed the distributed assembly permutation flow-shop scheduling problem (DAPFSP), which is divided into the stages of assembly and production. The authors developed the first MIP model for the DAPFSP with makespan minimization, three constructive heuristic approaches, and three VND methods to tackle large-size instances. Later, Lin & Ying (2016) extended the DPFSP considering no-wait constraints. The authors investigated the problem by proposing a MIP model with makespan minimization and devised an augmented variant of the IG, namely the Iterated Cocktail Greedy (ICG) algorithm, which employs a cocktail destruction operator and two automated tuning functions. In a recent study, Avci et al. (2022) investigated DPFSP with no-wait constraints devising alternative MIP formulations. As a solution approach, they proposed an exact branch-and-cut (BC) technique employing a heuristic approach for obtaining high-quality upper bounds. Pan et al. (2019) devised three constructive heuristics based on LR and NEH along with four solution approaches, namely, discrete artificial bee colony, scatter search, iterated local search, and iterated greedy, to minimize the total flowtime of the DPFSP. Jing et al. (2020) investigated DPFSP with due windows and proposed an IG algorithm to minimize *TWET* for the first time. Their IG includes an idle time insertion procedure to satisfy the due window constraints.

The Distributed No-idle FSP (DNIPFSP), which reduces the idle time between the execution of each pair of subsequent jobs on each machine, is another extension of the DPFSP. Ying et al. (2017) studied the DNIPFSP with makespan minimization for the first time. The authors proposed an Iterated Reference Greedy algorithm (IRG) to tackle large-size DNIPFSP instances. Afterward, Ling-Fang et al. (2018) employed a two-stage memetic algorithm (TSMA) for DNIPFSP. The proposed TSMA consists of two phases. In the first phase, various search procedures are utilized to categorize the solutions by considering the workloads of the factories. In the second phase, the best solutions lead the search in an effort to improve exploitation. Chen et al. (2019) examines a bi-objective DNIPFSP with the objective functions of minimizing

makespan and overall energy consumption. In addition, they have presented a collaborative optimization algorithm (COA) in which two heuristics are used collaboratively to ensure the diversity and quality of the initial population. Secondly, multiple search operators work to improve exploring the solution space in an adaptive manner. Thirdly, distinct local intensification techniques were devised for dominant and non-dominant individuals in order to increase exploitation. Finally, a speed-adjustment technique for non-critical operations is intended to reduce overall energy usage. Cheng et al. (2019) proposed the distributed mixed no-idle FSP with makespan criteria and an IG-based algorithm was devised as the solution approach. Li et al. (2021) expands the DNIPFSP investigating the mixed no-idle constraints (DMNIPFSP) with total flowtime minimization proposing a mathematical formulation and an Adaptive IG (AIG) algorithm as a solution approach. Furthermore, they have used swap-based local search techniques to enhance the quality of the solutions produced by the suggested algorithm. Li et al. (2022) examines the mixed no-idle assumption for the distributed assembly PFSP (DAMNIPFSP) with a total tardiness objective as the most recent update in DPFSP. They developed a MIP model in addition to an improved IG algorithm called Referenced Iterated Greedy (RIG). The proposed algorithm utilizes several new destructions, repairing, and local search methods.

Regarding the previous studies, it can be concluded that the no-idle assumption between jobs with due windows has never been investigated. Additionally, the DFSP literature covers the TWET objective scarcely. Furthermore, this study contributes to the literature by introducing DNIPFSPDW for the first time by proposing three mathematical models and providing a comprehensive benchmark. A novel no-idle time adjustment approach is introduced with a metaheuristic solution approach to solve large-scale instances of up to 500 jobs.

Table 2.2 Summary of related literature

Article	**Objective Function**	**Solution Approach**
Naderi & Ruiz (2010)	C_{max}	VND
Gao & Chen (2011)	C_{max}	GA-LS
Gao et al. (2013)	C_{max}	TS-LS
Hatami et al. (2013)	C_{max}	VND
Lin et al. (2013)	C_{max}	MIG
Wang et al. (2013)	C_{max}	EDA
Naderi & Ruiz (2014)	C_{max}	SS
Xu et al. (2014)	C_{max}	HIA
Fernandez-Viagas & Framinan (2015)	C_{max}	BSIG
Lin & Ying (2016)	C_{max}	ICG
Bargaoui et al. (2017)	C_{max}	CRO
Ying & Lin (2017)	C_{max}	IRG
Deng & Wang (2017)	C_{max}, TT	CMA
Ling-Fang et al. (2018)	C_{max}	TSMA
Ruiz et al. (2019)	C_{max}	IG
Chen et al. (2019)	C_{max}	IG
Ribas et al. (2019)	C_{max}, TT	IG
Cheng et al. (2019)	C_{max}	COA
Shao et al. (2020)	C_{max}, TT	MNIG
Ren et al. (2021)	C_{max}	NASH Q-Learning
Shao et al. (2021)	C_{max}	IG
Zhao et al. (2022)	C_{max}	MDDE
Avci et al. (2022)	C_{max}	BC
Fernandez-Viagas et al. (2018)	TFT	EA
Pan et al. (2019)	TFT	DABC, SS, ILS, IG
Li et al. (2021)	TT	AIG
Jing et al. (2020)	$TWET$	IG_{ITE}
Pan et al. (2017)	$TWET$	ILS
Li et al. (2022)	$TWET$	RIG
This study	$TWET$	IG-TS

CHAPTER THREE
THE DISTRIBUTED NO-IDLE FLOWSHOP SCHEDULING PROBLEM WITH DUE WINDOWS

Table 3.1 Summary of notations

Sets	
F	Set of available factories, $F = \{1,2,3,\ldots,\|F\|\}$,
J	Set of Jobs, $J = \{1,2,3,\ldots,\|J\|\}$,
I	Set of machines, $I = \{1,2,3,\ldots,\|I\|\}$.
L	Set of available positions, $L = \{1,2,3,\ldots,\|J\|\}$,
Parameters	
p_{ji}	Process time of job $j \in J$ on machine $i \in I$,
d_j^-	Earliest possible completion time of job $j \in J$,
d_j^+	Latest possible completion time of job $j \in J$,
w_j^e	Unit earliness penalty of job $j \in J$,
w_j^t	Unit tardiness penalty of job $j \in J$,
M	Sufficiently big number.
Sequence-based model decision variables	
y_{jf}	Is 1 if job $j \in J$ is assigned to factory $f \in F$,
$x_{j'jp}$	Is 1 if job $j \in J$ is processed immediately after job $j' \in J'$ in factory $f \in F$,
c_{jif}	Completion time of job $j \in J$ on machine $i \in I$ in factory $f \in F$.
Minimal sequence-based model decision variables	
x_{kj}	Is 1 if job $k \in J$ is processed immediately before the job $j \in J$,
c_{ji}	Completion time of job $j \in J$ on machine $i \in I$.
Position-based Model decision variables	
x_{jlf}	Is 1 if job $j \in J$ is processed in position $l \in L$ if factory $f \in F$,
c_{ilf}	Completion time of the job in position $l \in L$ on machine $i \in I$ in factory $f \in F$.

In some industries such as casting and ceramic frit manufacturing, since the setup periods and running expenses are high, or technological limitations in some cases, shutting down the machines after the initial setup is infeasible. In such industries, jobs are scheduled in such a way that there is no-idle time on the machines. Moreover, it is critical to meet due date requirements of the jobs. If a job is finished too late, some penalties may be incurred to reimburse the loss of the customer. On the other hand, if a job is completed too early, finished good inventory costs will increase. In this regard, we address the DNIFSPDW which is an extension of the DPFSP proposed by Naderi & Ruiz (2010) with no-idle constraints and due windows. Following the three-field notation introduced by (Graham et al., 1979) for the scheduling problems, the DNIFSPDW can be represented as $DF_m|prmu, no-idle|TWET$. This study has

extended the proposed mathematical models by Naderi & Ruiz (2010) with considering due windows and no-idle constraints and to the best of our knowledge, this study is the first to address DNIFSPDW.

The DNIFSPDW involves a set of jobs indicated by J={1,2,3,...,$|J|$} to be processed in F={1,2,3,...,$|F|$} available factories. Each factory is a flowshop with I={1,2,3,...,$|I|$} machines. Each job $j \in J$ on machine $i \in I$ has a process time of p_{ji} and should be completed in a due window represented by $[d_j^-, d_j^+]$, where d_j^- and d_j^+ indicate the earliest and latest completion times, respectively. Additionally, Table 3.1 summarizes the used notations in this study. The general assumptions of the DNIFSPDW are as follows:

- Each machine is allowed to process one job at a time.
- A job is processed only on one machine at a time.
- All jobs must be assigned to a factory.
- Preemption is not allowed once processing of a job is started on a machine.
- Machines cannot be stopped once they begin to operate, i.e., once machine $i \in I$ starts operating on a given job sequence, it cannot be stopped or interrupted until all the jobs in the given sequence are completed.

Let c_j be the completion time of job $j \in J$ on the last machine. If c_j is realized to be less than d_j^-, the amount of earliness is e_j =max$(d_j^- - c_j, 0)$. When job $j \in J$ is completed later than its latest due date d_j^+, the amount of tardiness is t_j =max$(c_j - d_j^+, 0)$. The objective function of the DNIFSPDW is to minimize total weighted earliness and tardiness (TWET) which is calculated as follows:

$$TWET = \sum_{j \in J} (w_j^e e_j + w_j^t t_j) \tag{3.1}$$

where w_j^e and w_j^t represent unit penalties for earliness and tardiness, respectively.

To illustrate the characteristics of the DNIFSPDW, an example problem consisting

of four jobs, two factories, and three machines is presented. The associated data for the example problem is provided in the following.

$$\begin{pmatrix} w_j^e \\ w_j^t \end{pmatrix} = \begin{pmatrix} 8 & 3 & 7 & 4 \\ 4 & 7 & 6 & 9 \end{pmatrix}, \quad p_{ji} = \begin{pmatrix} 14 & 15 & 50 \\ 3 & 59 & 1 \\ 77 & 65 & 77 \\ 71 & 56 & 21 \end{pmatrix}, \quad \begin{pmatrix} d_j^- \\ d_j^+ \end{pmatrix} = \begin{pmatrix} 126 & 143 & 137 & 165 \\ 189 & 174 & 177 & 201 \end{pmatrix} \tag{3.2}$$

A solution for this problem is shown in Figure 3.1. As can be seen in Figure 3.1, the processing of Job 1 on Machine 2 of Factory 1 is delayed by 56 units due to the no-idle constraints. Additionally, the processing of Job 1 on Machine 3 of Factory 1 is delayed by 26 time units. In fact, 6 time units of delay is sufficient to meet no-idle constraints. However, additional 20 time units of delay is needed to fit Job 4 into its due window. If only 6 time units of delay were added, Job 4 would be as early as 4 time units. As a result of delaying the processing of Job 1 on Machine 3 by 4 time units, Job 4 is not early. As one can infer from this example, the calculation of the completion times in the DNIFSPDW is not straightforward. In this regard, this study proposes "no-idle adjustment" and "gap insertion" procedures to calculate the total amount of delay needed to meet no-idle and due window requirements. The no-idle adjustment and gap insertion procedures are presented in Chapter 4. As a result, the earliness and tardiness values of the jobs are calculated as follows.

$$\begin{pmatrix} e_j \\ t_j \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 46 & 42 & 0 \end{pmatrix} \tag{3.3}$$

The objected function value for this solution is $TWET = 46 \times 7 + 42 \times 6 = 574$

To the best of our knowledge, there is no mathematical formulation developed for the DNIFSPDW in the related literature. In this regard, this study proposes three mathematical models for the DNIFSPDW, namely, sequence-based model (M_{seq}), minimal sequence-based model (M'_{seq}), and position-based model (M_{pos}). The proposed models are described in the following.

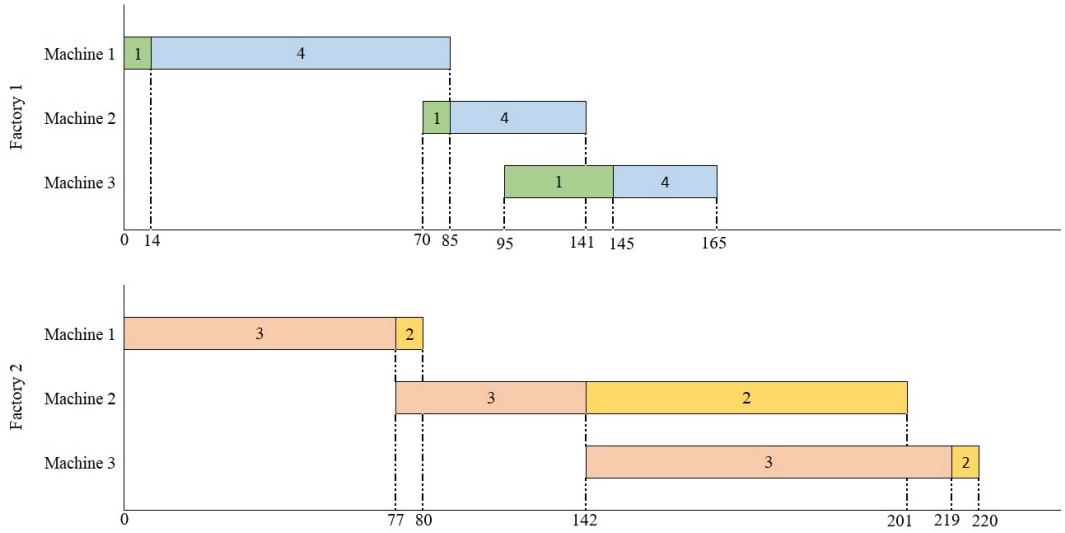

Figure 3.1 Gantt chart representing the solution of the example problem

3.1 Sequence-based Model (M_{seq})

M_{seq} employs a set of binary decision variables to represent the relative sequences of the jobs. Each sequence begins with a dummy job, $j = 0$ with zero processing time. Therefore, a dummy job is added to job set J, $J' = J \cup \{0\}$. Regarding M as a sufficiently big number, the decision variables employed in M_{seq} are as follows:

$$y_{jf} = \begin{cases} 1, & \text{if job } j \in J \text{ is assigned to factory } f \in F, \\ 0, & \text{otherwise.} \end{cases}$$

$$x_{j'jf} = \begin{cases} 1, & \text{if job } j \in J \text{ is processed immediately after } j' \in J',\ j' \neq j \\ & \text{in factory } f \in F, \\ 0, & \text{otherwise.} \end{cases}$$

c_{jif}: Completion time of job $j \in J$ on machine $i \in I$ in factory $f \in F$.

M_{seq} for the DNIFSPDW is as follows:

$$\min \sum_{j \in J} (w_j^e e_j + w_j^t t_j) \tag{3.4}$$

Subject to:

$$\sum_{j' \in J' : j' \neq j} \sum_{f \in F} x_{j'jf} = 1 \qquad \forall j \in J \tag{3.5}$$

$$\sum_{j \in J} y_{jf} = 1 \qquad \forall f \in F \tag{3.6}$$

$$\sum_{j \in J} x_{0jf} \leq 1 \qquad \forall f \in F \tag{3.7}$$

$$\sum_{j' \in J' : j' \neq j} (x_{j'jf} + x_{jj'f}) = 2y_{jf} \qquad \forall j \in J, f \in F \tag{3.8}$$

$$\sum_{i \in I} c_{jif} \leq M y_{jf} \qquad \forall j \in J, f \in F \tag{3.9}$$

$$c_{jif} \geq c_{j(i-1)f} + p_{ji} - M(1 - y_{jf}) \qquad \forall\, i \in I, j \in J, f \in F \tag{3.10}$$

$$c_{jif} \geq c_{j'if} + p_{ji} - M(1 - x_{j'jf}) \qquad \forall\, i \in I, j \in J, j' \in J' : j \neq j', f \in F \tag{3.11}$$

$$c_{jif} \leq c_{j'if} + p_{ji} + M(1 - x_{j'jf}) \qquad \forall i \in I, j \in J, j' \in J' : j \neq j', f \in F \tag{3.12}$$

$$c_{jif} \geq c_{j'if} + p_{ji} - M(1 - x_{j'jf}) \qquad \forall j \in J, j' \in J' : j \neq j', i \in I, f \in F \tag{3.13}$$

$$e_j \geq \sum_{f \in F} c_{j|I|f} - d_j^+ \qquad \forall j \in J, i \in I \tag{3.14}$$

$$t_j \leq d_j^- - \sum_{f \in F} c_{j|I|f} \qquad \forall j \in J, i \in I \tag{3.15}$$

$$y_{jf} = \{0, 1\} \qquad \forall j \in J, f \in F \tag{3.16}$$

$$x_{j'jf} = \{0, 1\} \qquad \forall j \in J, j' \in J, f \in F \tag{3.17}$$

$$c_{jif} \geq 0 \qquad \forall j \in J, i \in I, f \in F \tag{3.18}$$

$$e_j \geq 0 \qquad \forall j \in J \tag{3.19}$$

$$t_j \geq 0 \qquad \forall j \in J \tag{3.20}$$

The objective function (3.4) minimizes the total weighted earliness and tardiness. Constraints (3.5) guarantee that each job $j \in J$ should be preceded by only one job and assigned to exactly one factory. Constraints (3.6) force each job $j \in J$ to be assigned

to only one factory. Constraints (3.7) and (3.8), guarantee that each job $j \in J$ assigned to factory $f \in F$ must have exactly one successor and one predecessor. Constraint set (3.9) states that completion times of a job $j \in J$ in factory $f \in F$ can be positive if and only if job $j \in J$ is assigned to factory $f \in F$. Constraints (3.10) determine the completion time of each job $j \in J$ on each machine $i \in I$. Constraints (3.11) and (3.12) prevent the machines from having idle time. Constraints (3.13) guarantee that if job $j \in J$ is scheduled immediately after job $j' \in J'$, the processing of job $j \in J$ on machine $i \in I$ cannot begin until job $j' \in J'$ on machine $i \in I$ is completed. Constraints (3.14) and (3.15) calculate earliness and tardiness values for each job $j \in J$. Constraints (3.16) - (3.20) control the boundaries of the decision variables.

3.2 Minimal sequence-based model (M'_{seq})

The minimal sequence-based model (M'_{seq}), solves the problem without actually indexing the factories. Similar to M_{seq}, it is required to define dummy jobs 0. The followings are variables utilized in this model:

$$x_{kj}: \begin{cases} 1, & \text{if job } k \in J \text{ is processed immediately before the job } j \in J, \\ 0, & \text{Otherwise.} \end{cases}$$

c_{ji}: Completion time of job $j \in J$ on machine $i \in I$.

This model uses the dummy job 0 to divide an entire sequence into F sections, each corresponding to a factory. This is accomplished by F repeats of dummy job 0 and other jobs in the sequence. Consequently, this model searches the space using a sequence that contains $J + F$ positions. One of these repeats occurs at the beginning of the sequence. All following jobs are planned in factory 1 with their existing relative order, up to the second iteration of dummy job 0. Jobs sent to factory 2 are those between the second and third iterations of dummy job 0. This is repeated for each successive occurrence of dummy job 0. Those jobs following the F^{th} repeat dummy job 0 until the last job in the sequence is allocated to factory F.

M'_{seq} for the DNIFSPDW is as follows:

$$\min \sum_{j \in J} (w_j^e e_j + w_j^t t_j) \tag{3.21}$$

Subject to:

$$\sum_{k \in J: k \neq j} x_{kj} = 1 \qquad \forall j \in J \tag{3.22}$$

$$\sum_{j \in J: k \neq j} x_{kj} \leq 1 \qquad \forall k \in J \tag{3.23}$$

$$\sum_{j \in J} x_{0j} = |F| \tag{3.24}$$

$$\sum_{j \in J} x_{k0} = |F| - 1 \tag{3.25}$$

$$x_{kj} + x_{jk} \leq 1 \qquad \forall j \in J : j \neq k, j > k \tag{3.26}$$

$$c_{ji} \geq c_{j(i-1)} + p_{ji} \qquad \forall i \in I \backslash \{1\}, j \in J \tag{3.27}$$

$$c_{ji} \geq c_{ki} + p_{ji} + M(x_{kj} - 1) \qquad \forall k, j \in J : k \neq j, i \in I \tag{3.28}$$

$$c_{ji} \leq c_{ki} + p_{ji} - M(x_{kj} - 1) \qquad \forall k, j \in J : k \neq j, i \in I \tag{3.29}$$

$$c_{j0} \geq p_{j0} x_{0j} \qquad \forall j \in J \tag{3.30}$$

$$t_j \geq c_{j|I|} - d_j^+ \qquad \forall j \in J' \tag{3.31}$$

$$e_j \geq d_j^- - c_{j|I|} \qquad \forall j \in J' \tag{3.32}$$

$$x_{kj} = \{0, 1\} \qquad \forall k \in J, j \in J' \tag{3.33}$$

$$c_{ji} \geq 0 \qquad \forall j \in J', i \in I \tag{3.34}$$

$$e_j \geq 0 \qquad \forall j \in J' \tag{3.35}$$

$$t_j \geq 0 \qquad \forall j \in J' \tag{3.36}$$

The objective function (3.21) minimizes the total weighted earliness and tardiness. The constraints (3.22) and (3.23) guarantee that each job can have only one predecessor and one successor. Constraints (3.24) and (3.25) guarantee to have the dummy job 0 in the sequence as the preceding job a total of F times, and it must appear as the succeeding job a total of $F - 1$ times, respectively. Constraints (3.26) prohibit job $j \in J$ from simultaneously being a successor and a predecessor of job

$k \in J$. Constraints (3.27) force an operation of job $j \in J$ to start only after the completion of its previous operation. Constraints (3.28) and (3.29) guarantee the machine no-idle constraint. Constraint set (3.30) controls the completion time of jobs on the first machine separately. Constraints (3.31) and (3.32) calculate earliness and tardiness values for each job $j \in J$. Lastly, Constraints (3.33) - (3.36) define the binary and continuous decision variables, respectively.

3.3 Position-based Model (M_{pos})

M_{pos} employs a set of binary decision variables representing positions that are occupied by the jobs in factories. Let $L = 1,..,|J|$ be the set of available positions in each factory. The following are the variables included in M_{pos}:

$$x_{jlf}: \begin{cases} 1, & \text{If job } j \in J \text{ is processed at position } l \in L \text{ in factory } f \in F, \\ 0, & \text{Otherwise.} \end{cases}$$

c_{ilf}: Completion time of the job in position $l \in L$ on machine $i \in I$ in factory $f \in F$.

M_{pos} for the DNIFSPDW is as follows:

$$\min \sum_{j \in J} (w^e_j e_j + w^t_j t_j) \tag{3.37}$$

Subject to:

$$\sum_{l \in L} \sum_{f \in F} x_{jlf} = 1 \qquad \forall j \in J \tag{3.38}$$

$$\sum_{j \in J} x_{jlf} \leq 1 \qquad \forall l \in L, f \in F \tag{3.39}$$

$$c_{ilf} = c_{i(l-1)f} + \sum_{j \in J} x_{jlf} p_{ji} \qquad \forall l \in L \backslash \{1\}, i \in I, f \in F \tag{3.40}$$

$$c_{1lf} \geq \sum_{j \in J} x_{jlf} p_{j1} \qquad \forall l \in L, f \in F \tag{3.41}$$

$$c_{ilf} \geq c_{(i-1)lf} + \sum_{j \in J} x_{jlf} p_{ji} \qquad \forall l \in L, f \in F, i \in I \backslash \{1\} \tag{3.42}$$

$$e_j \geq d_j^+ - c_{|I|lf} - M(1 - x_{jlf}) \qquad \forall l \in L, \quad f \in F, \quad j \in J \qquad (3.43)$$

$$t_j \geq c_{|I|lf} - d_j^- - M(1 - x_{jlf}) \qquad \forall l \in L, \quad f \in F, \quad j \in J \qquad (3.44)$$

$$x_{jlf} = \{0, 1\} \qquad \forall l \in L, \quad f \in F, \quad j \in J \qquad (3.45)$$

$$c_{ilf} \geq 0 \qquad \forall l \in L, \quad f \in F, \quad i \in I \qquad (3.46)$$

$$e_j \geq 0 \qquad \forall j \in J \qquad (3.47)$$

$$t_j \geq 0 \qquad \forall j \in J \qquad (3.48)$$

The objective function (3.37) minimizes the total weighted earliness and tardiness. Constraint set (3.38) enforces that each job $j \in J$ must occupy exactly one position in one factory. Constraints (3.39) stipulate that each position $l \in L$ in a factory $f \in F$ is assigned to at most one job. Constraints (3.40) prevent machine idle time between jobs. Constraints (3.41) and (3.42) determine each position's completion times on the machines. Constraints (3.43) and (3.44) specify the earliness and tardiness values of each job $j \in J$. Finally, constraints (3.45) - (3.48) indicate the boundaries of the decision variables.

The FSP with a total weighted tardiness objective is known to be NP-hard (Lawler, 1977) therefore since the DNIFSPDW is an extension of this problem it also is an NP-hard problem. Therefore, exact solution approaches are inefficient in solving large-size DNIFSPDW instances. For such complex problems, metaheuristic approaches are more suitable as can obtain solutions with high quality in reasonable computation times. In this regard, a hybrid iterated greedy-tabu search algorithm (IG-TS) is developed for the DNIFSPDW. The details of the proposed algorithm are explained in Chapter 4.

CHAPTER FOUR
THE PROPOSED HYBRID ITERATED GREEDY - TABU SEARCH ALGORITHM

This chapter describes the solution approach proposed for the DNIFSPDW. Section 4.1, describes the proposed algorithm in steps. Since there exist no benchmark results for the DNIFSPDW in the related literature, a basic IG with local search (IG-LS) is developed as a benchmark algorithm which is described in Section 4.2.

4.1 Hybrid Iterated Greedy - Tabu Search (IG-TS)

As indicated in Chapter 2, IG has been used widely for the DPFSP and its variants. IG is composed of three stages, namely, initialization, destruction, and reconstruction. Throughout the destruction and reconstruction stages, new solutions are generated iteratively using greedy heuristics. Figure 4.1 describes the steps of the proposed IG-TS algorithm. Firstly, an initial solution, S, is generated by a variant of the NEH heuristic described in Section 4.1.2. Then, the destruction stage removes some of the jobs from the original permutation described in Section 4.1.3. On the other hand, during the reconstruction step, the jobs that were deleted are reinserted to create a new, incumbent solution which is described further in Section 4.1.4. Furthermore, the TS is activated when the best-so-far solution does improve over a certain number of iterations, and Section 4.1.5 describes this stage in detail. To change the current solution to a new one, some acceptance criteria are considered to determine whether the move should take place. The proposed algorithm uses a temperature parameter T that controls the probability of accepting the solutions. If a move results in a solution S' with a better objective compared to the current solution S, then S' is accepted unconditionally. Otherwise, S' is accepted with the probability $e^{\frac{f(S^*)-f(S')}{T}}$, where f represents the objective function.

In this study, the initial value of parameter T is the initial solution's objective value. Throughout the iterations, the value of T is exponentially decreased to zero. In order to decide on the temperature change, the equilibrium condition is evaluated. An

equilibrium condition takes place whenever a certain number of iterations does not result in an improvement to the obtained best solution. In each iteration, if the equilibrium condition is established, the value of T is multiplied by a *CoolingRate* coefficient between [0,1], and the temperature decreases.

Algorithm 2 General framework of the solution approach

Input:
r: Number of elements to be removed;
$activeTabu$: The number of iterations with no improvements;
$maxIter$: maximum number of iterations;
$maxNoImprove$: maximum number of iterations with no improvements to activate termination criteria;
Output: S^*
$S \leftarrow$ **Initialization**();
$S^* \leftarrow S$; $iter \leftarrow 1$; $noImprove \leftarrow 0$; $continue \leftarrow$ True;
while ($continue$) **do**
  $S^D, R \leftarrow$ **Destruction** (S, r);
  $S' \leftarrow$ **Reconstruction** (S^D, R);
  if $(f(S') < f(S^*))$ **then**
    $S^* \leftarrow S'$;
    $S \leftarrow S'$;
    $noImprove \leftarrow 0$
  else
    $rand \leftarrow$ generate a random number in [0, 1];
    if $(rand \leq e^{\frac{f(S^*)-f(S')}{T}})$ **then**
      $S^* \leftarrow S'$;
    end if
  end if
  if $(noImprove \geq activeTabu)$ **then**
    $S \leftarrow$ **Tabu Search**(S^*);
    $S^* \leftarrow S$;
  end if
  if ($noImprove > maxNoImprove$ **or** $iter > maxIter$) **then**
    $continue \leftarrow$ False;
  end if
  $iter \leftarrow iter + 1$;
end while

Figure 4.1 General framework of the solution approach

4.1.1 Solution representation and evaluation

Jobs are assigned to the factories in a sequential manner, in other words, each factory $f \in F$ is assigned a sequence of jobs denoted as Q_f. The encoded solution is represented as a number sequence indicated by Q_f to reflect the order of jobs allocated to different factories. The order of Q_f is meant to display the sequential order of jobs entering the factory $f \in F$. An example of solution encoding for five jobs $J = \{1,2,3,4,5\}$ and two factories $F = \{1,2\}$ is presented as $S = \{(2,4,1),(3,5)\}$. Therefore, the order of jobs sequences for the first and second factories are as $Q_1 = \{(2,4,1)\}$ and $Q_2 = \{(3,5)\}$, respectively. The next subsections explain the components of the proposed solution approach in detail.

Due to the no idle constraints and due windows, decoding of the DNIFSPDW solutions is not straightforward. The conventional forward calculation method is not applicable for obtaining the job completion time in the DNIFSPDW. Therefore, this study employs no-idle adjustment and gap insertion procedures in calculating the completion times of jobs. To better illustrate the characteristics of no-idle adjustment and gap insertion methods, an example problem with two jobs and three machines is presented for a factory. The data for the example problem is as follows:

$$\begin{pmatrix} w_j^e \\ w_j^t \end{pmatrix} = \begin{pmatrix} 8 & 4 \\ 4 & 9 \end{pmatrix}, \quad p_{ji} = \begin{pmatrix} 14 & 15 & 50 \\ 71 & 56 & 21 \end{pmatrix}, \quad \begin{pmatrix} d_j^- \\ d_j^+ \end{pmatrix} = \begin{pmatrix} 126 & 172 \\ 159 & 192 \end{pmatrix} \tag{4.1}$$

Figure 4.2 (a) illustrates a normal forward completion calculation that cannot satisfy the no-idle constraint due to the different process times of each job in each machine. Therefore, there has remained 56 units of idle time between jobs 1 and 2 in machine two, and similarly, machine three has 62 units of idle time. To satisfy this constraint, a backward completion time is suggested, which, to prevent confusion, has been named the no-idle adjustment. As Figure 4.2 (b) portrays, the idle times between jobs 1 and 2 for both machines two and three are zero. On the other hand, job completion times do not fit their respective due windows, and to minimize this gap,

the suggested gap insertion method inserts 20 units of gap (4.2 (c)).

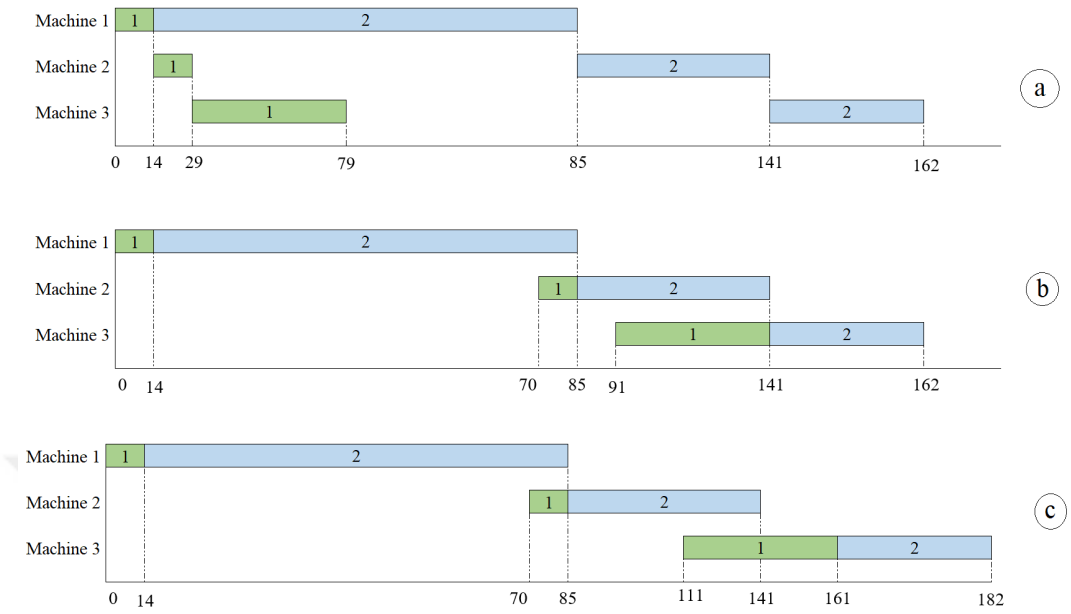

Figure 4.2 Gantt chart for illustrating no-idle adjustment and gap insertion

4.1.1.1 No-idle adjustment

One of the most common methods to obtain the completion time in FSP is the conventional forward approach. The forward calculation determines the jobs completion times on each machine in each factory by following the job sequence $Q_f = \{j^1, j^2, \ldots, j^q\}$ in factory $f \in F$, where j^q represents q^{th} job on the sequence. The finishing time of the first job (i.e. j^1) on the first machine equals to p_{11}, and the completion time of other jobs in Q_f on $i = 1$ is calculated as follows:

$$c_{j1} = c_{(j-1)i} + p_{ji} \qquad \forall j \in Q_f \backslash \{j^1\}, \quad f \in F \tag{4.2}$$

Then, for each machine $i \in I\backslash\{1\}$ the completion time of job $j \in Q_f \backslash \{j^1\}$ is calculated as follows:

$$c_{ji} = \max(c_{j(i-1)}, c_{(j-1)i}) + p_{ji} \qquad \forall j \in Q_f \backslash \{j^1\}, \quad i \in I\backslash\{1\} \tag{4.3}$$

When job $j \in J$ on machine $i-1$ takes longer to complete the job $j-1$ on machine i, machine i may remain idle between two consecutive jobs ($c_{ji} - c_{(j-1)i} > 0$). Therefore, to eliminate any possible idle time for machine $i \in I\backslash\{1\}$, the completion time of other jobs will be calculated backwardly starting from j^q.

$$c_{(j-1)i} = \max(c_{ji} - p_{ji}, c_{(j-1)i}) \qquad \forall j \in \{j^q, j^{q-1}, j^{q-2}, \ldots, j^2\}, \quad i \in I\backslash\{1\} \tag{4.4}$$

This method, however, was found to be inapplicable in this case since it would cause certain machines to go without work when a job on one machine took longer to complete than a job on the next. Hence, to calculate the completion time of each job on a machine, the conventional forward calculation method is unable to observe this fundamental assumption. Therefore, a novel forward-backward completion time calculation procedure is proposed to ensure the no-idle constraint.

4.1.1.2 Gap insertion

Sometimes it is not possible for all jobs to be completed in their exact time window due to real-world's operational limits. Thus, a proper production plan must finish the jobs as close to their due date as possible to avoid causing extra penalties. Therefore, regarding the no-idle time assumption between jobs, some gaps should be considered for each machine prior to starting the production process. A question that naturally arises in this part is to what extent it is worth adding a gap to obtain the best objective function value. There are a number of gap insertion methods available in the related literature (Tseng & Liao (2008) Pan et al. (2017) Rossi & Nagano (2020) Zhu et al. (2022)), but due to similarities in the nature of problems, this study uses an adapted version of gap insertion approach based on Jing et al. (2020) in order to establish the ideal time for jobs to be completed. It should be noted that in the related literature, these methods are known as "idle time insertion," but to avoid any confusion with machine idle time, it is decided to call the method "gap insertion."

To compute gaps, it suffices to execute the gap insertion procedure just to the last

machine in each factory. Let S_M denote the sequence of jobs in the last machine, which can be divided into the following three subsets:

- S_E: The jobs that are completed earlier than their earliest due date ($S_E = \{j \in S_M | E_j > 0\}$)
- S_T: The jobs that are completed later than their latest due date ($S_T = \{j \in S_M | T_j > 0\}$)
- S_D: The jobs that are completed on time ($S_D = \{j \in S_M | c_j \geq d_j^- \ , \ c_j \leq d_j^+\}$)

regarding the one unite minimum gap insertion, S_T indicates job $j \in J$ completed on its latest possible due date (i.e., $c_j = d_j^+$) is considered. By adding one unit of gap, the total earliness penalty cuts down by $\sum_{j \in S_E} w_j^e$ and increases the total tardiness penalty by $\sum_{j \in S_T} w_j^t$. Therefore, if $\sum_{j \in S_E} w_j^e > \sum_{j \in S_T} w_j^t$, inserting gap will lead to a better solution in terms of objective value. Gap insertion will continue until adding one unit of gap results in a greater tardiness penalty than the earliness. Let θ denote the maximum insertable gap based on a given jobs sequence Q_f. The calculation of θ can vary depending on the S_E and S_D circumstances. Figure 4.3 explains different steps of calculating θ further in detail.

4.1.2 Initialization

This study utilizes a variant of the NEH heuristic as an initial solution. NEH heuristic needs a seed sequence to generate a solution constructively. To generate the required seed sequence, the earliest due date - weighted earliness tardiness (EDDWET) is utilized since it is suggested as the best heuristic by Jing et al. (2020). Which can be seen in Figure 4.4.

EDDWET considers both due dates and unit earliness/tardiness weight. Firstly, G_w^e and G_w^t, are created for the jobs considering their unit earliness and tardiness weights. If $w_j^t > w_j^e$, set G_w^t will be iclluded with job j, otherwise is added to the set, G_w^e. The

Algorithm 1 Gap insertion

Input: Q_f; $\theta \leftarrow 0$; C_{ji};
Output: E **and** T;
for (f **to** $|F|$ **do**
  $s \leftarrow |Q_f|$;
  while $s \geq 0$ **do**
    $\theta \leftarrow 0$;
    $S_E \leftarrow \{\ \}$;
    $S_T \leftarrow \{\ \}$;
    $S_D \leftarrow \{\ \}$;
    for all j **in** Q_f **do**
      if $(E_j > 0)$ **then**
        $S_E \leftarrow S_E \cup \{j\}$;
      end if
      if $(T_j > 0)$ **then**
        $S_T \leftarrow S_T \cup \{j\}$;
      end if
      if $(T_j = 0,\ E_j = 0)$ **then**
        $S_D \leftarrow S_D \cup \{j\}$;
      end if
      if $(\sum_{j \in S_E} W_j^E > \sum_{j \in S_T} W_j^T)$ **then**
        if $(S_E \neq \emptyset$ and $S_D \neq \emptyset)$ **then**
          if $(\sum_{j \in S_E} W_j^E > \sum_{j \in S_D} W_j^T)$ **then**
            $\theta = \min_{j \in S_E} E_j$;
          else
            $\theta = \min(\min_{j \in S_E}(E_j), \min_{j \in S_D}(d_j^+ - C_{jI}))$;
          end if
        end if
      else if $(S_E \neq \emptyset)$ **then**
        $\theta = \min_{j \in S_D}(E_j)$;
      else if $(S_D \neq \emptyset)$ **then**
        $\theta = \min_{j \in S_D}(d_j^+ - C_{jI})$;
      end if
      if $(\theta > 0)$ **then**
        for (**all** $j \in S_M$ **and** $i \in I$) **do**
          Set $C_{ji} \leftarrow C_{ji} + \theta$;
        end for
        for (**all** $j \in S_E$ **and** $i \in I$) **do**
          Set $E_j \leftarrow E_j - \theta$;
        end for
        for (**all** $j \in S_T$ **and** $i \in I$) **do**
          Set $T_j \leftarrow T_j + \theta$;
        end for
      else
        $s \leftarrow s - 1$;
      end if
    end for
  end while
end for

Figure 4.3 The pseudocode for gap insertion

jobs in set G_w^e are then represented by a non-decreasing order of their W_j^e to create an incomplete sequence τ_w^e. On the other hand, the jobs in set G_w^t are sorted based on W_j^t, in a non-increasing order to produce an incomplete sequence τ_w^t. After that, the first jobs of each partial sequence are picked and compared in terms of their d^+. The job with the smallest d^+ is removed from the associated list and added to the end of the seed sequence, τ. The procedure continues until one of the partial sequences (τ_w^t or τ_w^e) is depleted; in this case, the remaining jobs in other partial sequence are added to the end of the τ. Then, the first $|F|$ jobs in τ are assigned to be the first jobs of each factory. By following the seed sequence, τ, the remaining jobs are assigned to the factories one by one. Each job is assigned to the factory with the smallest incremental penalty in the objective function respective to adding the new job.

Algorithm 3 EDDWET

Input: $G_w^E \leftarrow \{\}$; $G_w^T \leftarrow \{\}$; $\tau_W^E \leftarrow \{\}$; $\tau_W^T \leftarrow \{\}$; $\tau \leftarrow \{\}$;
Output: τ
for (j=1 **to** |J|) **do**
  if $W_j^T > W_j^E$ **then**
    $G_w^T \leftarrow$ job j;
  else
    $G_w^E \leftarrow$ job j;
  end if
end for
$\tau_W^T \leftarrow$ Sort G_w^T according to w_j^T in non-increasing order;
$\tau_W^E \leftarrow$ Sort G_w^E according to w_j^E in non-increasing order;
for (i = 1 **to** min($sizeof(\tau_W^T), sizeof(\tau_W^E)$)) **do**
  Compare d_j^+ of the first job from τ_W^T and the first job from τ_W^E;
  Take the job with the smaller value from its sequence and append to τ
end for
if ($sizeof(\tau_W^T > 0)$) **then**
  $\tau \leftarrow \tau + \tau_w^T$
end if
if ($sizeof(\tau_W^E > 0)$) **then**
  $\tau \leftarrow \tau + \tau_w^E$
end if

Figure 4.4 The pseudocode for EDDWET

4.1.3 Destruction

The destruction stage removes randomly r selected jobs from the solution at hand to create a partial solution and a list of removed jobs (R). To this aim, firstly, a non-zero integer value denoting the number of deleted jobs (i.e., r) should be determined, and then, the destruction strategy by which the jobs are deleted is specified.

As an important component of the IG algorithm, destruction size remains constant during the iterations. Furthermore, this study utilizes four different destruction strategies, which are activated randomly at each destruction attempt. These operators are tailored upon the problem concept to effectively probe the solution space, trying to find a better solution.

- **Random selection:** This method is the simplest approach, which randomly selects and removes r jobs from an existing incumbent solution.
- **Factory-based selection:** The factory-based operator tries to distribute jobs evenly by trimming the job load on the factories with high penalty costs. This operator deletes one random job from up to r factories with the highest TWET value. If $r > |F|$, the remaining $r - |F|$ jobs are randomly eliminated.
- **Greedy selection:** One promising way to decrease the objective value is to replace the jobs with the highest penalty. Greedy selection intends to remove $\frac{r}{2}$ jobs with the greatest tardiness and the other $\frac{r}{2}$ with the highest earliness penalties.
- **Block selection:** In order to effectively search the neighboring solution space, this operator employs a block-based selection technique, in which a consecutive series of jobs from a random factory's sequence (Q^f) is picked at a random position. The size of the block takes an integer number between [1,min(r,$|Q^f|$]. This operator will apply multiple times on different factory sequences until the number of deleted jobs meets r.

4.1.4 Reconstruction

The reconstruction operators create an incumbent solution using the partial solution (S^D) and removed jobs list (R) resulting from the destruction stage. The four distinct operators are employed randomly at each reconstruction attempt.

- **Order selection:** The most typical selection sequence for the removed jobs is their removed order. Because the destruction phase frequently eliminates certain jobs randomly based on some heuristics, the order in which they are deleted could be illustrative. Therefore, the jobs are selected to be reinserted to reconstruct the partial solution. To construct a complete solution, the algorithm examines all possible positions of each removed job and saves a certain number of them having the lowest increase in objective value. Then, a position is picked using a Roulette Wheel approach, to insert a given job on each attempt until all the removed jobs are treated.

- **Random selection:** To further investigate the unexplored parts of solution space, the eliminated tasks are chosen one by one in random order. The reinsertion process is performed in a similar manner to the order selection operator.

- **Greedy selection:** Each removed job is tentatively inserted into all the positions of a partial solution. The insertion with the minimum objective value increment will apply iteratively until all the removed jobs are reinserted.

- ***k*-regret:** This study employs the idea of regret function with some modifications to fit into the current problem's concept. The present k-regret function attempts to compute the regret value, which represents the difference between the value of inserting a job into its first best position and its k^{th} best position ($k > 1$). For a given solution and a value of k, the regret is obtained as follows:

$$\Delta\lambda_j = \lambda_j^1 + \sum_{n=2}^{k} (\lambda_j^n - \lambda_j^1) \qquad \forall\ j \in J \tag{4.5}$$

  Where λ_j^n denotes the objective value for the nth-best insertion position of job j.

Then, the job with the highest $\Delta\lambda_j$ is picked for insertion in its best place until all removed jobs are resolved.

4.1.5 Tabu Search (TS)

In most cases, the IG algorithm and its variants consist of a search technique known as a Local Search to carefully explore the solution space. As an optional phase, a local search function can be applied to the solution of the reconstruction process, hoping to make the solution better. This study employs an effective search technique to empower the search scheme extensively, namely Tabu Search.

In the context of a search strategy, TS refers to the process of moving from one feasible solution to another with each iteration. In PFSP, the TS algorithm is often structured, to begin with, a sequence and evolves consecutively through neighboring solutions to find a better one. A "Tabu List (TL)" allows TS to avoid going back to the same solutions from which it recently emerged. By doing so, the solution space is comprehensively explored, and trapping in locally optimal solutions is easily avoided. For doing so, recently evaluated solutions are declared tabu for a certain number of iterations. A move made in iteration t is called tabu until iteration $(t+\delta)$ where δ is the prespecified tabu tenure (TT) value.

Tabu list can be considered as a short-term memory consisting of the history of the recent moves to prevent returning back to visited sequences. This study employs a customized version of the TL generation technique proposed by Ying & Lin (2017) considering α jobs included. From each factory, one job with the lowest earliness, absent in TL, is inserted to set $\mathscr{A}_1$. Similarly, $\mathscr{A}_2$ represents the set of jobs with the lowest tardiness from each factory, not included in TL. Initially the jobs included in TL will be composed of those in $\mathscr{A}_1 \cup \mathscr{A}_2$. The rest of $\alpha - |\mathscr{A}_1 \cup \mathscr{A}_2|$ distinct jobs are selected randomly from the input solution.

TS algorithm proposed in this study begins with the best-found solution based on the IG-based procedure described already and sets it as the *bestGlobal* solution. Similar to

the destruction phase, r random jobs are removed from the *bestGlobal* solution. Then the new solution is reconstructed using the k-regret operator.

After generating a bunch of neighboring solutions according to the *bestGlobal*, each new solution is identified to be whether forbidden or free. If a move is recognized as forbidden according to the TL, it will not be approved unless it meets the aspiration criterion. When a move results in a better solution than the best found, the aspiration criteria is a measure used to override the move's tabu status.

Next, the best of forbidden and free solutions will be determined to specify the best solution for the current iteration (*bestIter*). If there exists a best-free solution (*bestFree*), *bestIter* is updated to it even if the objective of the best-forbidden solution (*bestForbiden*) is better. The aspiration criterion, however, allows the *bestForbiden* if it improves the *bestGlobal*. Subsequently, the TT for all jobs in the TL will be decreased by one. At the same time, the *bestIter* will be compared with *bestGlobal*. If it is better than *bestGlobal*, the *bestGlobal* will be updated accordingly. The TL will be updated based on the new *bestGlobal*. This procedure will go on until either the algorithm meets the number of unimproved iterations exceeds *maxNoImprove*.

Each iteration of TS generates a predetermined number of neighbors. In evolutionary algorithms, it is conventional to generate neighbors sequentially. Even though the operating system may allow parallel execution, sequential processing never benefits from multi-core processors. A parallel approach has been employed to accelerate process efficiency in terms of execution time. Parallel programming unlocks a program's ability to execute multiple processes simultaneously. Parallel computing addresses a given issue by breaking it into subproblems, simultaneously solving those subproblems in parallel (with each subproblem executing in a distinct thread), and then integrating the results of the subproblems. In this study, the primary challenge is to generate a bunch of neighbor solutions, and the subproblem is to create each neighbor individually. To take advantage of the parallel feature, the effort of creating a maximum number of neighbors is uniformly split among all available threads. This method of implementation considerably reduces the algorithm's

Algorithm 4 Tabu search

Input: *maxNeighbor*; *TSmaxNoImprove*; *TSmaxIter*;
Output: S^*TT; α; S^*, $S^{'}$, $S \leftarrow$ the input solution;
initialize:
$iter \leftarrow 0$; $Continue \leftarrow$ true; $noImprove \leftarrow 0$; generate the initial TL based on S^* and α;
while ($Continue$) **do**
  $bestForbidden \leftarrow \emptyset$, $bestFree \leftarrow \emptyset$;
  $\wp^D \leftarrow$ generate *maxNeighbor* partial solutions using **Random Selection**(S', r);
  $k \leftarrow$ generate a random integer between [2, 3];
  $\wp^R \leftarrow k$**-regret(**$\wp^D$**,** k**)**
  for (each $S \in \wp^R$) **do**
    if (S is forbidden and its better than $bestForbidden$ **or** S is forbidden **and** $bestForbidden = \emptyset$) **then**
      $bestForbidden \leftarrow S$;
    end if
    if (S is free and its better than $bestFree$ **or** S is free **and** $bestFree = \emptyset$) **then**
      $bestFree \leftarrow S$;
    end if
  end for
  if ($betsFree \neq \emptyset$) **then**
    $S^{'} \leftarrow bestFree$;
  end if
  if ($bestForbidden \neq \emptyset$ and $bestForbidden$ is better than S^*) **then**
    $S^{'} \leftarrow bestFree$;
  end if
  update TL by decreasing each non-zero element by 1.
  if (the objective of $S^{'}$ < the objective of S^*) **then**
    $S^* \leftarrow S^{'}$;
    regenerate TL based on S^* and α;
    $noImprove \leftarrow 0$;
  else
    $noImprove \leftarrow noImprove + 1$;
  end if
  if ($noImprove > maxNoImprove$ or $iter > TSmaxIter$) **then**
    $Continue \leftarrow$ false;
  end if
  $iter \leftarrow Iter + 1$;
end while

Figure 4.5 The pseudocode for the proposed Tabu search algorithm

execution time.

4.2 Iterated Greedy with Local search (IG-LS)

Since no other benchmarks exist for this problem, this study employs a classic IG algorithm with local search (IG-LS) (i.e. Figure 4.6) as a benchmark algorithm in the analysis of IG-TS performance. Therefore, to outline a fair comparison of results, IG-LS utilizes the same initial solution, destruction, and reconstruction methods as in the proposed IG-TS. Furthermore, IG-LS employs an LS technique that sequentially uses three search methods. Local search begins with the first neighborhood and moves on to the next operator when no better solution can be identified. This continues until the last operator can't find any better solution and reaches its local optimum. The following summarizes the local search operators chosen to be applied in the algorithm.

- **Single Insertion:** As one of the most used operators in FSPs, it improves a given solution by relocating a job and reinserting it into another position. Initially, a job will be chosen randomly from factories with more than one job, and then it will be reinserted into another random factory and position. The new candidate position is picked randomly upon a probability mass function, which considers a higher chance for the best possible factory and position in terms of objective value.
- **Block Insertion:** In this operator all the removed jobs are reinserted as a block into a randomly selected factory with a randomly assigned location with no change in the sequence of jobs in the block.
- **Exchange:** this operator randomly chooses and exchanges two distinct jobs of an incumbent solution from two randomly selected factories.

Algorithm 5 The pseudocode for the IG-LS

Input:
r: Number of elements to be removed;
$maxIter$: maximum number of iterations;
$maxNoImprove$: maximum number of iterations with no improvements to activate termination criteria;
Output: S^*
$S \leftarrow$ **Initialization**();
$S^* \leftarrow S$; $iter \leftarrow 1$; $noImprove \leftarrow 0$; $continue \leftarrow$ True;
while ($continue$) **do**
  $S^D, R \leftarrow$ **Destruction** (S, r);
  $S^{'} \leftarrow$ **Reconstruction** (S^D, R);
  if $(f(S') < f(S^*))$ **then**
    $S^* \leftarrow S^{'}$;
    $S \leftarrow S^{'}$;
    $noImprove \leftarrow 0$;
  else
    $rand \leftarrow$ generate a random number in [0, 1];
    if $(rand \leq e^{\frac{f(S^*)-f(S^{'})}{T}})$ **then**
      $S^* \leftarrow S^{'}$;
    end if
  end if
  $S^{'} \leftarrow$ **SingleInsertion**(S^*);
  while $(f(S^{'}) < f(S^*))$ **do**
    $S^* \leftarrow S^{'}$;
    $S^{'} \leftarrow$ **SingleInsertion**(S^*);
  end while
  $S^{'} \leftarrow$ **BlockInsertion**(S^*);
  while $(f(S^{'}) < f(S^*))$ **do**
    $S^* \leftarrow S^{'}$;
    $S^{'} \leftarrow$ **BlockInsertion**(S^*);
  end while
  $S^{'} \leftarrow$ **Exchange**(S^*);
  while $(f(S^{'}) < f(S^*))$ **do**
    $S^* \leftarrow S^{'}$;
    $S^{'} \leftarrow$ **Exchange**(S^*);
  end while
  if $(noImprove > maxNoImprove$ **or** $iter > maxIter)$ **then**
    $continue \leftarrow$ False;
  end if
  $iter \leftarrow iter + 1$;
end while

Figure 4.6 The pseudocode for the IG-LS

CHAPTER FIVE
COMPUTATIONAL STUDY

This section summarizes the findings of computational experiments conducted using the created benchmark in a series of tests. Additionally, the created benchmark will be presented in detail. Section 5.1 explains the generated benchmarks and the experimental setting. Then the proposed mathematical models are analyzed in section 5.2. Furthermore, Sections 5.3 and 5.4 explain the proposed algorithm's parameters calibration and analyze the algorithm's components, respectively. Finally, section 5.5 evaluates the performance of the IG-TS algorithm.

5.1 Benchmark and experimental setting

Despite the existence of benchmarks for FSP issues, the unique characteristic of the DNIPFSPDW problem examined in this study (the due windows) necessitates specific attention to due dates. Therefore, Naderi & Ruiz (2010) DPSP data were extended with T', W, and R parameters to create two sets of instances: small and large benchmarks. Instance generation is governed by six variables: the number of machines (I), jobs (J), factories (F), tardiness factor (T'), due date range (R), and the width of the due window with respect to the due date (W). Data are developed using Naderi & Ruiz (2010) processing time data. The unit earliness and tardiness weights are generated using a uniform distribution in the range U [1,9]. The due dates are generated with a random uniform distribution based on the equation (5.1) proposed method where $\lfloor X \rceil$ indicates rounding X to the nearest integer. Equation 5.1 portraits due date generation formula:

$$d_j = \max(0, U[\lfloor P(1 - T' - R/2) \rceil, \lfloor P(1 - T' + R/2) \rceil]) \tag{5.1}$$

The calculated due dates vary from simple to difficult to fulfill depending on the parameters of T' and R. Thus P indicates the makespan upper bound calculated by the NEH2 heuristic for DPFSP developed by Naderi & Ruiz (2010). The due windows

are generated centered around d_j with $W\%$ width of d_j as follows: $d_j^- = \max(0, \lfloor d_j - d_j \times H/100 \rceil)$ and $d_j^+ = \max(0, \lfloor d_j + d_j \times H/100 \rceil)$ where $H = [1, W]$. Data are generated for $F = \{2,4\}$, $I = \{3,4,5\}$, and $J = \{4,10,16\}$ for small-sized instances and $F = \{4,7\}$, $I = \{5,10,20\}$, and $J = \{2050,100,200,500\}$ for large-instances employing the following parameter combinations: $T' = \{0.2,0.4\}$, $R = \{0.2,0.6\}$ and $W = \{20,50\}$ with each combination having five replicates with different seeds. Table 5.1 summarizes the considered values for each parameter. In general, there are 720 small and 960 large instances.

Table 5.1 Value of the considered parameters

Parameter	**Values**
T'	0.2, 0.4
R	0.2, 0.6
W	20, 50

The IG-TS algorithm was implemented in Java, and the mathematical models were solved in python using the Gurobi solver 9.0. All the computational tests have been carried out on a computer with Intel(R) Core (TM) i3-7130U CPU @ 2.70GHz and 8 GB of RAM, using the Microsoft Windows 10 operating system. The computational CPU time for Gurobi is limited to 3600s (1 h) in the experiments.

5.2 The performance of the proposed mathematical models

Tables 5.2 and 5.3 show the computational results of the proposed mathematical models for small-sized instances with 2 and 4 factories, respectively, where the number of jobs is 4, 10, and 16. It should be noted that the "UB" column depicts the upper bound found by the Gurobi solver, the "Gap" column illustrates the gap between Gurobi's best bound and lower bound, and "Opt Num" indicates the number of instances that the Gurobi solver was able to solve optimally in the given time out of 40 instances. All models deliver an equal performance for the instances with four jobs regarding the computational time and optimality gap.

Table 5.2 The average results for the small-sized instances with two factories.

Job (J)	Machine (I)	M_{seq}				M'_{seq}				M_{pos}			
		UB	Gap (%)	Opt Num	Time	UB	Gap (%)	Opt Num	Time	UB	Gap (%)	Opt Num	Time
	3	684.97	0.00	40	0.05	684.97	0.00	40	0.00	684.97	0.00	40	0.00
4	4	1020.27	0.00	40	0.00	1020.27	0.00	40	0.00	1020.27	0.00	40	0.00
	5	1016.85	0.00	40	0.00	1016.85	0.00	40	0.00	1016.85	0.00	40	0.00
	3	1345.20	0.00	35	43.43	1345.20	0.00	40	129.55	1345.20	0.00	40	35.83
10	4	2619.92	32.94	17	2544.18	2619.92	0.00	40	360.78	2619.92	0.00	40	64.70
	5	2934.07	33.35	17	2642.75	2924.63	0.00	40	390.58	2924.62	0.00	40	71.03
	3	1710.39	34.59	6	3162.20	1706.52	74.60	8	2986.43	1668.11	31.78	9	2847.33
16	4	4072.29	97.08	1	3511.00	3980.90	91.01	2	3468.58	3882.50	84.96	3	3461.43
	5	4599.44	97.18	0	3600.00	4482.02	89.55	1	3523.63	4254.60	84.73	2	3473.88

Table 5.3 The average results for the small-sized instances with four factories.

Job (J)	Machine (I)	M_{seq}				M'_{seq}				M_{pos}			
		UB	Gap (%)	Opt Num	Time	UB	Gap (%)	Opt Num	Time	UB	Gap (%)	Opt Num	Time
	3	747.67	0.00	40	0.00	747.67	0.00	40	0.00	747.67	0.00	40	0.00
4	4	615.47	0.00	40	0.00	615.47	0.00	40	0.00	615.47	0.00	40	0.00
	5	511.30	0.00	40	0.00	511.30	0.00	40	0.00	511.30	0.00	40	0.00
	3	949.00	11.44	28	1582.20	949.00	0.00	40	3.78	949.00	0.00	40	247.95
10	4	1605.52	19.81	19	3207.50	1604.12	0.00	40	3.88	1604.12	0.44	40	594.78
	5	2396.97	34.08	8	3254.50	2391.00	0.00	40	11.88	2391.00	0.00	40	713.07
	3	1278.12	67.38	8	3057.83	1270.30	40.24	11	2644.30	1258.30	66.41	8	2885.00
16	4	1997.05	38.61	5	3210.95	1947.32	43.41	11	2707.05	1942.44	61.74	9	2898.75
	5	3446.62	78.46	1	3577.28	3329.57	53.98	2	3372.68	3331.06	78.21	2	3423.48

In instances with 10 or more jobs, the models become more complex, which is reflected in the higher CPU time required to solve the instances. The performance comparison of the models reflected in Table 5.2 with ten jobs suggests that M'_{seq} and M_{pos} perform far better than the M_{seq} since they obtain the optimal solution for all the cases. Nevertheless, M_{pos} can attain this result within a relatively shorter time. Although a similar performance is observed for 16 jobs with 2 factories, the M_{pos}, as the best model in this case, finds the optimal solution of 14 out of 120 instances. In addition, the Gap column for 16 jobs suggests that M_{pos} performs better on finding the lower bounds.

According to Table 5.3 the M_{seq} finds the optimal solution only for 55 instances out of 120 with ten jobs, while the other models find the optimal solution for all the cases. In terms of CPU time, M'_{seq} presents a significantly better performance by solving the instances of ten jobs in an average time of 6.5s, while M_{pos} obtains the same results in 518.60s. The outperformance of the M'_{seq} does not limit to the ten jobs, and it delivers better results for the instances of 16 jobs by finding the optimal solution of 24 out of 120 instances in an average time of 2908.01s. Additionally, the lower bounds quality proposed by M'_{seq} is better than the others according to the Mean Gap column of Table 5.3.

Table 5.3 shows the solver demands more computational time for M_{pos} and M_{seq} models to prove the optimality in comprising with Table 5.2, as having four factories provides more potential positions for the jobs to accommodate. In contrast, M'_{seq} model requires far less CPU time for solving the instances with four factories rather than two.

Generally, the M_{seq} fails to provide a better lower bound, incumbent solution, and CPU time on average for all the cases under study. The performance of the other models highly depends on the number of factories in the given instance. Although M_{pos} performs a little worse for the instances of four factories compared to M'_{seq}, it well performs significantly for the cases with two factories. Therefore, to analyze the effect of other parameters, the performance of the M_{pos} will be investigated as it. To determine the effect of the number of factories on the M_{pos}, an ANOVA with a

significance level of 0.05 was conducted. The finding indicates the influence of the number of factories' influence with a P-value of 0.02, and since P-value > α so, it is statistically significant. Therefore as the number of factories increases the complexity also increases.

The results show larger values for T', W, and R, resulting in lower values for the objective value and less computational time due to generating wider due dates. Thus, it is easier for models to find the appropriate position for the jobs, which causes low penalties for earliness and tardiness. In the instances with two factories considering 20 for W, the average required time for model 2 is 1204.12s with a mean gap of 29.89%, while in the four-factory instances, average time and gap are 1195s and 27.98% for all jobs and instances. When the value of W is increased to 50, the average time and optimality gap are 1007.91s and 23.34% for instances with 2 factories, while these values are 971.48s and 18.38% for instances with 4 factories. Therefore, tighter values for W demand more time to converge to the optimal solution. The conducted ANOVA test with a significance level of 0.05 for W as a fixed variable in response to the gap for M_{pos} model also confirms the effect of W on decreasing objective value and elapsed time with a P-value of 0. For different values of R, the variation of the optimality gap is insignificant, while it has a considerable impact on the computational time. In two-factory and four-factory instances for R with the value of 0.2, the average CPU time for the M_{pos} is 1162.26s and 1219.61s, respectively, while for $R = 0.6$ the average elapsed time is 1049.72s and 1172s. Regarding the optimality gap, for $R = 0.2$, the average gap equals 28.47% for the instances with 2 factories and 25.56% for those with 4 factories. By performing a similar ANOVA test with the fixed variable R in response to the gap, a P-value of 0.001 demonstrates the significant effect of R in increasing objective value and CPU time. A totally different observation is offered while changing the value of T' from 0.2 to 0.4. For two factories with $T = 0.2$, the average gap and time are 25.23% and 1003.18s, and for $T = 0.4$, they are 28.20% and 1208.85s. On the other hand, for the instances with four factories, the mean gap and time for $T = 0.2$ are 20.06% and 906.49s, while for $T = 0.4$, they are 25.89% and 1485.25s. These results imply that larger values of T' narrow down the due date of

jobs, which in return, increases the complexity of solution space, requiring the model to spend more time investigating the solution space. Moreover, a slight change in the value of T' results in a greater change in average CPU time and gap compared to R. Similarly, in the ANOVA test with the fixed variable T' in response to gap, a P-value of 0.001 confirms that T' has a noticeable effect on increasing both CPU time and objective value. Although the number of machines in each factory does not have a direct impact on the model, its variation impacts the average computational gap and mean CPU time. By growing the number of machines the average gap and mean CPU time for instances with 10 and 16 jobs increase. This impact comes down to the fact that adding one machine to a factory will exert its processing time on the production sequence, having the jobs finish later. Thus, the jobs finished in their regular time are pushed to have a delay, so their tardiness penalty should be considered now. Therefore, finding the best combination of delayed jobs and their associated penalty cost becomes more challenging as the flexibility of the on-time jobs diminishes or cancels. Another ANOVA test was carried out to assess the significance of the number of machines on the performance of M_{pos}. With a P-value of 0.026, the result demonstrates that machine number has a significant effect on increasing both elapsed time and objective value. Table 5.4 summarizes the conducted ANOVA test on M_{pos} with a significance level of 0.05.

Table 5.4 Summary of conducted ANOVA test on M_{pos}

Parameter	**P-value**	**Relation**
F	0.020	As the F increases the complexity and CPU time increase
I	0.026	As the I increases the complexity and CPU time increase
J	0.000	As the J increases the complexity and CPU time increase
W	0.000	As the W increases the complexity and CPU time decrease
R	0.001	As the R increases the objective value increases
T'	0.001	As the T' decreases the complexity and CPU time increases

5.3 Calibration of algorithm parameters

For optimal algorithm performance, it is necessary to calibrate the algorithm's affecting parameters. Therefore a Taguchi experiment using the L27 orthogonal array was designed by randomly selecting eight instances and considering three levels for

each parameter and each instance was run 15 times. The proposed algorithm contains five parameters α, r, $TS\,MaxNoImprove$, $maxNeighbor$, and TT. Table 5.5 illustrates the considered values for each parameter which are mostly from existing literature.

Table 5.5 Considered values for each parameter

Parameter	Values
α	$\{0.25\times\|J\|, 0.50\times\|J\|, 0.75\times\|J\|\}$
r	{5, 7, 10}
TT	{3, 4, 5}
$maxNeighbor$	{80, 100, 120}
$TS\,MaxNoImprove$	{5, 10, 15}

The main effects of S/N ratios on IG-TS parameter levels are presented in Figure 5.1. As one can infer from the Figure 5.1, the best parameter levels of the algorithm are $tabuMaxNoImprove = 10$, $tabuMaxNeighbor = 80$, $TT = 3$, $\alpha = 0.25\times|J|$, and $r = 7$

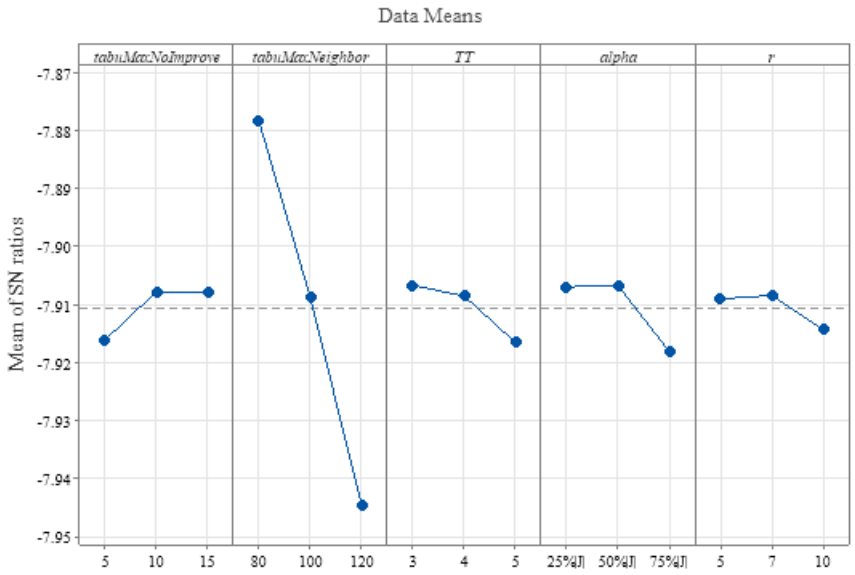

Figure 5.1 Main effects of S/N ratios for IG-TS parameters

5.4 Analysis of the algorithm components

The proposed hybrid algorithm consists of two main parts, namely, IG and TS, each of which can be employed as an independent solution approach. In this regard,

additional computational experiments were conducted to show the value of hybridization. In this analysis, a subset of the large-sized DNIFSPDW instances with four factories including 96 instances which are the first replications of each combination of T', R, W, J, and I. Table 5.6 illustrates the averages of objective function values and computational times obtained from ten runs of each part. The column entitled "Obj" displays the average objective function value for each algorithm over the course of the ten runs, while the column labeled "Time" presents the mean computation time required to obtain the results for each algorithm. It has been observed that, despite their shorter computational times, the TS and IG algorithms fall short in achieving the level of results obtained by the IG-TS algorithm. The difference in objective values between TS and IG-TS is approximately 8%, with this gap increasing to a substantial 108% when comparing IG and IG-TS.

Table 5.6 The average results of analysis of the algorithm components.

Jobs (J)	Machines (I)	IG-TS		TS		IG	
		Obj	Time	Obj	Time	Obj	Time
20	5	2799.00	11.67	2818.00	9.04	4179.25	0.32
	10	16008.13	13.24	16255.00	10.20	22578.25	0.32
	20	40886.13	15.47	43222.88	12.18	55359.00	0.35
50	5	7602.13	29.74	8257.88	21.29	15455.38	10.52
	10	28752.75	36.75	31825.25	26.54	53905.88	7.37
	20	140589.38	44.95	150096.75	32.68	246271.50	8.42
100	5	13185.50	194.29	14775.50	134.22	35375.25	102.79
	10	48453.50	198.47	55636.75	141.73	119506.25	59.60
	20	293605.13	153.58	314939.63	115.76	640868.75	61.39
200	10	89304.00	749.97	97236.50	529.54	239163.13	218.16
	20	585672.88	680.23	609173.63	516.10	1297382.75	197.77
500	20	1349265.38	1000.00	1528109.63	1000.00	3793522.25	1000.00

In order to confirm the validity of the results obtained, a Wilcoxon signed-rank test was employed as a non-parametric statistical method for comparing two correlated sets of data. Non-parametric tests are useful for evaluating algorithms that generate average outcomes for a given problem, even in the absence of any correlation among them (García et al., 2009). The Wilcoxon test yielded a P-value of 0, which supports the alternative hypothesis and indicates that the IG-TS and TS algorithms vary in a statistically significant way. Additionally, a similar test was conducted for the IG-TS and IG algorithms, which also yielded a P-value of 0, further establishing the existence of a statistically significant difference between the two algorithms.

5.5 Performance analysis of IG-TS

Tables 5.7 and 5.8 illustrate the computational results of the algorithms for small-sized instances with 2 and 4 factories, where the numbers of jobs are 4, 10, and 16. The column "RPD" indicates the average RPD value for each instance set where RPD is calculated as ($\frac{SomeSol-BestSol}{BestSol}$). Additionally, the "Time" column indicates the required average time for the algorithms to solve a problem configuration over 10 multiple runs. The proposed IG-TS algorithm is able to identify the optimal solution of instances with two factories and four jobs in less than one second. A similar performance is observed for the IG-LS algorithm, but with a relatively higher mean time compared to the IG-TS. For instances of 10 jobs with the two factories, the proposed IG-TS algorithm finds the optimal solution in a relatively small CPU time compared to the mathematical model. This demonstrates that the proposed IG-TS performs better even for small-sized instances. Similarly, the IG-LS algorithm solves the problems for the mentioned instances to optimality in the same CPU time as IG-TS. On the other hand, by increasing the number of jobs to 16 the IG-TS algorithm was able to either find or improve the solution obtained by the Gurobi solver in a considerably lower CPU time. Although the IG-LS algorithm gives a solution in relatively less time compared to IG-TS, it fails to find the best solution proposed by IG-TS in some cases regarding the RPD mean values.

Table 5.7 The average results for the small-sized instances with two factories.

Job (J)	Machine (I)	IG-TS		IG-LS	
		RPD	Time	RPD	Time
4	3	0.00	0.11	0.00	0.30
	4	0.00	0.13	0.00	0.23
	5	0.00	0.02	0.00	0.21
10	3	0.00	2.19	0.00	1.34
	4	0.00	2.16	0.00	1.35
	5	0.00	2.17	0.00	0.14
16	3	0.00	5.82	0.52	5.35
	4	0.00	5.06	2.14	5.06
	5	0.00	5.05	1.01	5.03

Table 5.8 illustrates that for all small-sized instances with four factories the IG-TS, and IG-LS deliver the same performance as they did for two factories, in terms of

Table 5.8 The average results for the small-sized instances with four factories.

Job (J)	Machine (I)	IG-TS		IG-LS	
		RPD	Time	RPD	Time
4	3	0.00	0.00	0.00	0.00
	4	0.00	0.00	0.00	0.00
	5	0.00	0.00	0.00	0.00
10	3	0.00	0.52	0.00	0.02
	4	0.00	2.55	0.00	0.11
	5	0.00	2.62	0.00	0.12
16	3	0.00	4.98	0.08	0.33
	4	0.00	4.72	0.31	0.32
	5	0.00	4.99	1.20	0.31

finding the best solution. However, RPD mean values confirm that both perform better in finding the best solution since there are more possible locations for jobs to accommodate more available factories. Additionally, the average time of IG-TS and IG-LS for instances of four factories is even less than that of two factories. Hence, for the large instances, only the results of the proposed heuristic algorithms are provided.

Tables 5.9 and 5.10 illustrate the obtained results by IG-TS and IG-LS algorithms for large-sized instances. In general, for all large-sized instances, the IG-TS algorithm is able to deliver solutions of higher quality in a shorter amount of CPU time. The impact of increasing the number of jobs on the mean time and RPD of both algorithms is noteworthy in this context. With growing the number of jobs the required time for finding the best solution for both algorithms increases as it demands more computational efforts, in each iteration. However, the time gap between both algorithms for a given instance is significant. This time gap basically roots back to the utilized operators in both algorithms especially the k-regret. Since the k-regret method examines and evaluates an extensive number of possible insertions of removed jobs at each attempt, it demands a larger time to come to a conclusive solution at each iteration. Therefore, each iteration of the IG-TS algorithm takes longer to process compared to IG-LS, as it employs k-regret in the TS body. However, the solution quality obtained from IG-TS surpasses that of IG-LS regarding the RPD column. The quantity of machines accessible at each factory is a crucial aspect in problem situations. Since increasing the number of machines in each factory demands more calculation owing to the no-idle time adjustment, growing the number of

machines increases the overall CPU time for a given problem setting.

Table 5.9 The average results for the large-sized instances with four factories.

Job (J)	Machine (I)	IG-TS		IG-LS	
		RPD (%)	Time	RPD (%)	Time
20	5	0.00	11.49	68.11	0.37
	10	0.00	12.75	12.56	0.33
	20	0.00	15.14	4.14	0.39
50	5	0.00	28.38	74.95	11.74
	10	0.00	36.20	34.45	8.53
	20	0.00	41.20	10.18	9.31
100	5	0.00	157.23	38.81	66.10
	10	0.38	171.66	24.02	68.59
	20	0.66	188.00	3.28	112.62
200	10	0.03	691.45	315.51	216.12
	20	0.02	839.46	10.88	240.96
500	20	1.06	1000.00	6.80	1000.00

Table 5.10 The average results for the large-sized instances with seven factories.

Job (J)	Machine (I)	IG-TS		IG-LS	
		RPD (%)	Time	RPD (%)	Time
20	5	0.00	2.24	38.05	0.73
	10	0.00	2.43	3.95	0.77
	20	0.00	2.75	2.72	0.87
50	5	0.00	15.35	809.35	7.64
	10	0.00	15.09	19.92	8.14
	20	0.01	16.32	6.87	8.47
100	5	0.00	90.68	1131.62	34.79
	10	0.29	95.60	9.87	35.95
	20	0.59	96.51	2.21	48.07
200	10	0.26	226.80	194.55	120.23
	20	0.49	313.72	6.56	172.68
500	20	0.47	1000.00	4.60	1000.00

CHAPTER SIX
CONCLUSION

This thesis is the first to study the distributed no-idle flowshop scheduling problem with due windows. This problem deals with two interrelated decisions: first job assignment to factories, and second the sequence of the assigned jobs. Three variants of mathematical models were proposed to address this concept in the flowshop scheduling problem aiming to minimize the total weighted earliness and tardiness penalties. The proposed position-based mathematical model was found to be the most efficient among the three formulations evaluated. Furthermore, an ANOVA test was performed to investigate the statistical impact of input parameters on the model's performance. The test's findings demonstrated that adding more jobs, machines, and factories to the model makes it more complicated, which in turn results in either a greater amount of CPU time required to solve the model or a larger gap in the obtained solution. This finding highlights the importance of considering the complexity of the model when selecting input parameters and emphasizes the need for efficient algorithms to handle large-scale problems. Additionally, it may also provide insight into the trade-offs between solution quality and computational time when solving such models. In this regard, an efficient hybrid IG-TS algorithm is presented as a solution approach for this problem. Furthermore, a classic IG algorithm with local search was utilized to create a performance comparison measure. The proposed IG-TS algorithm was able to identify the optimal solution or improve upon the results obtained by the Gurobi solver for small-sized instances. Furthermore, for large-sized instances, the results obtained by the benchmark IG-LS were improved upon by the proposed IG-TS algorithm. This demonstrates the effectiveness of the hybrid approach in finding optimal or near-optimal solutions for both small and large-sized instances of the problem and also highlights the potential of combining different optimization techniques to enhance the performance of a given algorithm. The proposed algorithm was calibrated through the application of a Taguchi experimental design approach. This method was used to determine the optimal set of parameters that would enhance the performance of the algorithm. Furthermore, some

experiments are carried out to assess the performance of the proposed models and algorithm for different problem settings. Furthermore, the computational research was conducted on benchmarks developed based on Naderi & Ruiz (2010) data.

The results demonstrate that the numbers of jobs and machines have a significant impact on the performance of mathematical models and the algorithm, as the solution space dramatically changes with these numbers. In contrast, for high numbers of factories, the algorithm outperforms the mathematical models regarding the solution quality and computation time. This study solves the problem with up to 500 jobs for the first time. Several research avenues remain unexplored in terms of solution approaches and the problem itself. Other combinatorial heuristics seem to be applicable to solve this problem for comparison purposes, like the Genetic algorithm. Additionally, other mathematical models with different sequence representative variables can be developed to solve the problem for larger sizes. Furthermore, the cutting planes can be devised to enhance the mathematical models' performance by accelerating the lower-bound growth.

REFERENCES

Alidaee, B., Li, H., Wang, H., & Womer, K. (2021). Integer programming formulations in sequencing with total earliness and tardiness penalties, arbitrary due dates, and no idle time: A concise review and extension. *Omega*, *103*, 102446.

Avci, M., Avci, M. G., & Hamzadayı, A. (2022). A branch-and-cut approach for the distributed no-wait flowshop scheduling problem. *Computers & Operations Research*, *148*, 106009.

Bargaoui, H., Driss, O. B., & Ghédira, K. (2017). A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion. *Computers & Industrial Engineering*, *111*, 239–250.

Chen, J.-f., Wang, L., & Peng, Z.-p. (2019). A collaborative optimization algorithm for energy-efficient multi-objective distributed no-idle flow-shop scheduling. *Swarm and Evolutionary Computation*, *50*, 100557.

Cheng, C.-Y., Ying, K.-C., Chen, H.-H., & Lu, H.-S. (2019). Minimising makespan in distributed mixed no-idle flowshops. *International Journal of Production Research*, *57*(1), 48–60.

De Giovanni, L., & Pezzella, F. (2010). An improved genetic algorithm for the distributed and flexible job-shop scheduling problem. *European journal of operational research*, *200*(2), 395–408.

Deng, J., & Wang, L. (2017). A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem. *Swarm and evolutionary computation*, *32*, 121–131.

Fernandez-Viagas, V., & Framinan, J. M. (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, *53*(4), 1111–1123.

Fernandez-Viagas, V., Perez-Gonzalez, P., & Framinan, J. M. (2018). The distributed

permutation flow shop to minimise the total flowtime. *Computers & Industrial Engineering*, *118*, 464–477.

Gao, J., & Chen, R. (2011). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, *4*(4), 497–508.

Gao, J., Chen, R., & Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, *51*(3), 641–651.

García, S., Molina, D., Lozano, M., & Herrera, F. (2009). A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. *Journal of Heuristics*, *15*(6), 617–644.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, vol. 5, 287–326.

Hatami, S., Ruiz, R., & Andres-Romano, C. (2013). The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*, *51*(17), 5292–5308.

Huang, J.-P., Pan, Q.-K., & Gao, L. (2020). An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation*, *59*, 100742.

Jing, X.-L., Pan, Q.-K., Gao, L., & Wang, Y.-L. (2020). An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows. *Applied soft computing*, *96*, 106629.

Kalczynski, P. J., & Kamburowski, J. (2005). A heuristic for minimizing the makespan in no-idle permutation flow shops. *Computers & Industrial Engineering*, *49*(1), 146–154.

Lawler, E. L. (1977). A “pseudopolynomial” algorithm for sequencing jobs to minimize total tardiness. In *Annals of discrete Mathematics*, vol. 1, 331–342.

Li, Y.-Z., Pan, Q.-K., Li, J.-Q., Gao, L., & Tasgetiren, M. F. (2021). An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems. *Swarm and Evolutionary Computation*, *63*, 100874.

Li, Y.-Z., Pan, Q.-K., Ruiz, R., & Sang, H.-Y. (2022). A referenced iterated greedy algorithm for the distributed assembly mixed no-idle permutation flowshop scheduling problem with the total tardiness criterion. *Knowledge-Based Systems*, *239*, 108036.

Lin, S.-W., & Ying, K.-C. (2016). Minimizing makespan for solving the distributed no-wait flowshop scheduling problem. *Computers & Industrial Engineering*, *99*, 202–209.

Lin, S.-W., Ying, K.-C., & Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, *51*(16), 5029–5038.

Ling-Fang, C., Ling, W., & Jing-jing, W. (2018). A two-stage memetic algorithm for distributed no-idle permutation flowshop scheduling problem. In *2018 37th Chinese Control Conference (CCC)*, *IEEE*, 2278–2283.

Mao, J.-y., Pan, Q.-k., Miao, Z.-h., & Gao, L. (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*, *169*, 114495.

Mao, J.-Y., Pan, Q.-K., Miao, Z.-H., Gao, L., & Chen, S. (2022). A hash map-based memetic algorithm for the distributed permutation flowshop scheduling problem with preventive maintenance to minimize total flowtime. *Knowledge-Based Systems*, *242*, 108413.

Naderi, B., & Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & operations research*, *37*(4), 754–768.

Naderi, B., & Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research*, *239*(2), 323–334.

Pan, Q.-K., Gao, L., Wang, L., Liang, J., & Li, X.-Y. (2019). Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem. *Expert Systems with Applications*, *124*, 309–324.

Pan, Q.-K., & Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. *Omega*, *44*, 41–50.

Pan, Q.-K., Ruiz, R., & Alfaro-Fernández, P. (2017). Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, *80*, 50–60.

Ren, J., Ye, C., & Li, Y. (2021). A new solution to distributed permutation flow shop scheduling problem based on nash q-learning. *Advances in Production Engineering & Management*, *16*(3), 269–284.

Ribas, I., Companys, R., & Tort-Martorell, X. (2019). An iterated greedy algorithm for solving the total tardiness parallel blocking flow shop scheduling problem. *Expert Systems with Applications*, *121*, 347–361.

Rossi, F. L., & Nagano, M. S. (2020). Heuristics and metaheuristics for the mixed no-idle flowshop with sequence-dependent setup times and total tardiness minimisation. *Swarm and Evolutionary Computation*, *55*, 100689.

Ruiz, R., Pan, Q.-K., & Naderi, B. (2019). Iterated greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, *83*, 213–222.

Saadani, N. E. H., Guinet, A., & Moalla, M. (2003). Three stage no-idle flow-shops. *Computers & industrial engineering*, *44*(3), 425–434.

Shao, W., Shao, Z., & Pi, D. (2020). Modeling and multi-neighborhood iterated greedy algorithm for distributed hybrid flow shop scheduling problem. *Knowledge-Based Systems*, *194*, 105527.

Shao, Z., Shao, W., & Pi, D. (2021). Effective constructive heuristic and iterated greedy algorithm for distributed mixed blocking permutation flow-shop scheduling problem. *Knowledge-Based Systems*, *221*, 106959.

Tseng, C.-T., & Liao, C.-J. (2008). A discrete particle swarm optimization for lot-streaming flowshop scheduling problem. *European Journal of Operational Research*, *191*(2), 360–373.

Wang, S.-y., Wang, L., Liu, M., & Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, *145*(1), 387–396.

Xu, Y., Wang, L., Wang, S., & Liu, M. (2014). An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem. *Engineering Optimization*, *46*(9), 1269–1283.

Ying, K.-C., & Lin, S.-W. (2017). Minimizing makespan in distributed blocking flowshops using hybrid iterated greedy algorithms. *IEEE Access*, *5*, 15694–15705.

Ying, K.-C., Lin, S.-W., Cheng, C.-Y., & He, C.-D. (2017). Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems. *Computers & Industrial Engineering*, *110*, 413–423.

Zhao, F., Hu, X., Wang, L., & Li, Z. (2022). A memetic discrete differential evolution algorithm for the distributed permutation flow shop scheduling problem. *Complex & Intelligent Systems*, *8*(1), 141–161.

Zhu, N., Zhao, F., Wang, L., Ding, R., Xu, T., et al. (2022). A discrete learning fruit fly algorithm based on knowledge for the distributed no-wait flow shop scheduling with due windows. *Expert Systems with Applications*, *198*, 116921.