**UNIVERSITY OF SÃO PAULO
SÃO CARLOS SCHOOL OF ENGINEERING**

Fernando Siqueira de Almeida

**Heuristics for multi-objective no-wait flow shops with
sequence-dependent setup times**

São Carlos

2021

Fernando Siqueira de Almeida

**Heuristics for multi-objective no-wait flow shops with
sequence-dependent setup times**

Thesis presented to the São Carlos School
of Engineering, University of São Paulo, to
obtain the degree of Master of Science - Grad-
uate Program in Production Engineering.

Concentration area: Processes and Operations
Management

Advisor: Prof. Dr. Marcelo Seido Nagano

VERSÃO CORRIGIDA

**São Carlos
2021**

I AUTHORIZE THE TOTAL OR PARTIAL REPRODUCTION OF THIS WORK, THROUGH ANY CONVENTIONAL OR ELECTRONIC MEANS, FOR STUDY AND RESEARCH PURPOSES, SINCE THE SOURCE IS CITED.

S856m Almeida, Fernando Siqueira de
Heuristics for multi-objective no-wait flow shops with sequence-dependent setup times / Fernando Siqueira de Almeida ; orientador Marcelo Seido Nagano. – São Carlos, 2021.
71 p. : il. (algumas color.) ; 30 cm.

Thesis (Master's degree) - Graduate Program in Production Engineering and Concentration area in Processes and Operations Management – Escola de Engenharia de São Carlos, University of São Paulo, 2021.

1. LaTeX. 2. abnTeX. 3. Classe USPSC. 4. Editoração de texto. 5. Normalização da documentação. 6. Tese. 7. Dissertação. 8. Documentos (elaboração). 9. Documentos eletrônicos. I. Nagano, Marcelo Seido, orient. II. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **FERNANDO SIQUEIRA DE ALMEIDA.**

Título da dissertação: "Heurísticas para no-wait flow shops multi-objetivo com tempos de preparação dependentes da sequência".

Data da defesa: 12/11/2021

Comissão Julgadora

Resultado

Prof. Dr. **Marcelo Seido Nagano**

(Orientador)

(Escola de Engenharia de São Carlos/EESC-USP)

Aprovado

Prof. Dr. **Mauricio Iwama Takano**

(Universidade Tecnológica Federal do Paraná/UTFPR)

Aprovado

Prof. Dr. **Fábio Molina da Silva**

(Universidade Federal de São Carlos/UFSCar)

Aprovado

Coordenadora do Programa de Pós-Graduação em Engenharia de Produção:

Profa. Dra. **Janaina Mascarenhas Hornos da Costa**

Presidente da Comissão de Pós-Graduação:

Prof. Titular **Murilo Araujo Romero**

To my family, who have consistently supported me during my life.

ACKNOWLEDGEMENTS

I could not have written this Thesis without the help of countless professionals, colleagues, friends and family members. Thank you all.

Special thanks to my advisor, Prof. Marcelo Seido Nagano, for his guidance and encouragement. He believed me even before I did.

Thank you, too, to my fellow members of the Laboratory of Applied Operational Research at São Carlos School of Engineering, for helping me through many stages of my study.

I am especially grateful to the University of São Paulo, for the opportunity to attend the Master's Program in Production Engineering and for providing its precious support during all these years.

Thanks also to the Coordination for the Improvement of Higher Education Personnel (CAPES), for sponsoring my master's scholarship No. 88882.379108/2019-01 and making this research feasible.

*“Hard work works. It may not make you the best,
but certainly will make you better than you are.”*

Jordan B. Peterson

ABSTRACT

Almeida, F. S. **Heuristics for multi-objective no-wait flow shops with sequence-dependent setup times**. 2021. 71p. Thesis (Master's degree) - São Carlos School of Engineering, University of São Paulo, São Carlos, 2021.

Productive systems often involve several objectives and constraints that need to be considered by the scheduler. Under these circumstances, solving scheduling problems with multiple criteria tends to be the most appropriate approach. In this context, the no-wait flow shop problem with sequence-dependent setup times is addressed. The performance measures makespan, total completion time and total tardiness are approached in pairs to form functions $\epsilon(M_1|M_2)$, in which the objective is to minimize M_1 subject to an upper bound on M_2 . Since this problem is known to be NP-hard, using exact methods for large instances can be impractical. As an alternative, heuristic methods have been developed to speed up the process of finding satisfactory solutions. In this Thesis, state-of-the-art methods for similar problems found in the literature are selected in order to explore opportunities for improvement. Focusing on simplicity of implementation and efficiency of execution, different heuristic methods are proposed. Extensive experiments are performed to evaluate performance. The results show that the proposed heuristics outperform the existing methods in solution quality and computational efficiency.

Keywords: Flow shop. No-wait. Sequence-dependent setup times. Makespan. Total completion time. Total tardiness.

RESUMO

Almeida, F. S. **Heurísticas para *no-wait flow shops* multi-objetivo com tempos de preparação dependentes da sequência.** 2021. 71p. Thesis (Master's degree) - São Carlos School of Engineering, University of São Paulo, São Carlos, 2021.

Sistemas produtivos geralmente envolvem vários objetivos e restrições que precisam ser considerados pelo programador. Nessas circunstâncias, resolver problemas de programação com múltiplos critérios tende a ser a abordagem mais adequada. Nesse contexto, o problema no-wait flow shop com tempos de preparação dependentes da sequência é abordado. As medidas de desempenho *makespan*, *total completion time* e *total tardiness* são abordadas em pares para formar funções $\epsilon(M_1|M_2)$, nas quais o objetivo é minimizar M_1 sujeito a um limite superior em M_2 . Como esse problema é conhecido por ser NP-hard, usar métodos exatos para instâncias grandes geralmente são impraticáveis. Como alternativa, métodos heurísticos têm sido desenvolvidos para acelerar o processo de busca de soluções satisfatórias. Nesta Dissertação, métodos considerados estado-da-arte para problemas semelhantes encontrados na literatura são selecionados para serem exploradas oportunidades de melhoria. Com foco na simplicidade de implementação e eficiência de execução, diferentes métodos heurísticos são propostos. Experimentos extensivos são realizados para avaliar o desempenho. Os resultados mostram que as heurísticas propostas superam os métodos existentes em qualidade de solução e eficiência computacional.

Palavras-chave: Flow shop. No-wait. Sequence-dependent setup times. Makespan. Total completion time. Total tardiness.

LIST OF FIGURES

Figure 1 – Workflow in a pure FSP	28
Figure 2 – Simple permutation FSP	29
Figure 3 – NWT-FSP	29
Figure 4 – Difference between attached and not attached setups	30
Figure 5 – NWT-FSP-SDST	33
Figure 6 – Examples of function $f(n)$	39
Figure 7 – ARPD trends of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j C_{max})$	42
Figure 8 – Multiple Comparisons of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j C_{max})$	43
Figure 9 – ARPD trends of IG_1, IG_2, IG_3 and IG_4	50
Figure 10 – ARPD trends of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j C_{max})$	51
Figure 11 – Multiple Comparisons of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j C_{max})$	52
Figure 12 – ARPD trends of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max} \sum C_j)$	61
Figure 13 – Multiple Comparisons of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max} \sum C_j)$	62

LIST OF TABLES

Table 1	– Research on NWT-FSP-SDST for C_{max} , $\sum C_j$ or $\sum T_j$	35
Table 2	– Research on NWT-FSP for $\epsilon(M_1 M_2)$	36
Table 3	– ARPD results of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j C_{max})$	41
Table 4	– Tukey' HSD test of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j C_{max})$	42
Table 5	– Destruction and construction mechanisms of IG_1 , IG_2 , IG_3 and IG_4 . .	46
Table 6	– Parameter calibration of IG_1 , IG_2 , IG_3 and IG_4	48
Table 7	– ARPD results of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j C_{max})$	50
Table 8	– Tukey' HSD test of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j C_{max})$	51
Table 9	– ARPD results of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max} \sum C_j)$	60
Table 10	– Tukey' HSD test of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max} \sum C_j)$	62

LIST OF ABBREVIATIONS AND ACRONYMS

ALNS	Adaptive Large Neighborhood Search
FIFO	First In First Out
FSP	Flow Shop Scheduling Problem
LNS	Large Neighborhood Search
MILP	The mixed integer linear programming
NWT	No-Wait
PM	Performance Measure
PFSP	Permutation Flow Shop Scheduling Problem
SDST	Sequence Dependent Setup Times

CONTENTS

1	INTRODUCTION	23
1.1	Objectives	25
2	PROBLEM STATEMENT	27
2.1	Flow shop	27
2.2	No-wait	28
2.3	Sequence-dependent setup times	30
2.4	Notation and classification system	31
2.4.1	Notation	31
2.4.2	Classification system	31
2.5	Problem formulation	32
3	LITERATURE REVIEW	35
4	THE NWT-FSP-SDST WITH TOTAL TARDINESS SUBJECT TO MAKESPAN	37
4.1	Heuristic algorithms	37
4.1.1	Literature algorithms	37
4.1.2	Iterated Greedy Algorithm (IG_A)	38
4.2	Computational experiments	40
4.3	Conclusion	42
5	THE NWT-FSP-SDST WITH TOTAL COMPLETION TIME SUB- JECT TO MAKESPAN	45
5.1	Heuristic algorithms	45
5.1.1	Literature algorithms	45
5.1.2	Iterated Greedy Algorithm ($IG_{\#}$)	46
5.2	Computational experiments	49
5.3	Conclusion	53
6	THE NWT-SDST WITH MAKESPAN SUBJECT TO TOTAL COM- PLETION TIME	55
6.1	Heuristic algorithms	55
6.1.1	Literature algorithms	55
6.1.2	Algorithm $ALNS_A$	56
6.2	Computational experiments	60
6.3	Conclusion	63

7	CONCLUSION	65
7.1	Future research directions	66
	REFERENCES	67

1 INTRODUCTION

Sequencing and scheduling are decision-making processes widely employed in both goods and services industries as well as computing designs, playing a crucial role in present times. It involves allocating time and resources to tasks in order to optimize an objective function (PINEDO, 2016). In production environments, scheduling problems are increasingly complex and propose practical solutions often involve analyzing several embedded constraints and objectives (T'KINDT; BILLAUT, 2006). In this context, the flow shop scheduling problem (FSP) is addressed with two features: no-wait (NWT) and sequence dependent setup times (SDST). The performance measures makespan, total completion time and total tardiness are approached in pairs to form multi-objective functions.

A flow shop is a processing system in which the operation sequence follows a chain precedence and all jobs visit the work stations in the same order. The most fundamental model, called pure flow shop, has only one machine at each work station that can handle one operation at a time. Most of the literature on flow shop scheduling is limited to the permutation flow shops, in which each machine processes the jobs in the same order. Thus, in a permutation FSP once the job sequence on the first machine is fixed it will be kept on all remaining machines. The result will be called permutation schedule. A permutation schedule can be completely characterized by a single sequence of jobs. The objective is to find the best sequence that optimize an objective function. (EMMONS; VAIRAKTARAKIS, 2013; PINEDO, 2016).

The no-wait flow shop is a case in which the waiting time between successive operations of a job is required to be zero. This implies that no job is permitted to utilize a buffer or to wait in an upstream machine. Therefore, whenever an operation has completed, the next machine has to be available. In those situations, it may be necessary to delay the starting time of a job at the first machine to fulfill this condition (PINEDO, 2016; BAKER; TRIETSCH, 2019). The no-wait constraint may occur for null waiting space, technological or process requirement, unavailability of resources and many other reasons (EMMONS; VAIRAKTARAKIS, 2013). In manufacturing industries, for example, the work material is often heated during deformation processes and delays are not allowed because they can result in cooling. In addition, modern environments such as flexible manufacturing systems, just-in-time and agile manufacturing can also be modelled as a no-wait flow shop scheduling problem (BERTOLISSI, 2000).

Some machines may require setups for incoming jobs. Setup is any preparation performed on a machine for it to be ready to process a particular job. This preparation may include cleaning, retooling, adjustments, inspection and rearrangement of work station.

In more complex systems, setup times depend on the similarities between the job just completed and the job about to be processed. These are called sequence-dependent setup times. Specifically, the length of setup times depends on the difficulty involved in switching from one processing configuration to another. SDST are commonly found where a single facility produces several kinds of items, or where a multipurpose machine carries out a variety of jobs. When setup times are sequence-dependent, instead of absorbing the setup times in the processing times, it is recommended to make explicit considerations to address the problem (BAKER; TRIETSCH, 2019; PINEDO, 2016; EMMONS; VAIRAKTARAKIS, 2013).

The scheduling performance measures considered are makespan, total completion time and total tardiness. These are among the most studied for NWT-FSP (ALLAHVERDI, 2016). Makespan represents the maximal or latest completion time among all jobs in the system. Minimizing makespan is performed when the goal is the efficient utilization of resources; that is, decrease equipment idle time. A reduced makespan is important when a complete batch of jobs is required as soon as possible. Minimizing total completion time means in practice increasing the delivery rate, and is appropriate when it is intended to ship each job as soon as it is completed. This is important when the goal is to decrease the work-in-process inventory and to provide a rapid response to demands. Total tardiness is the performance objective of meeting jobs due dates. It increases according to the gaps between job due dates and their completion times. Minimizing total tardiness implies that time-dependent penalties are assessed, but there are no benefits from completing jobs early (BAKER; TRIETSCH, 2019; PINEDO, 2016). Using the notation presented by (T'KINDT; BILLAUT, 2006), the performance measures (PM) are studied in the form $\epsilon(PM_1|PM_2)$, in which the objective is to minimize PM_1 subject to an upper bound on PM_2 . This formulation is appropriate when there is no need to optimize PM_2 as long as it does not exceed the upper bound.

The NWT-FSP is classified as NP-*hard*. It means that using exact methods can be unfeasible for larger instances. As an alternative, heuristic methods have been developed. Heuristics are flexible tools capable of solving essentially any combinatorial optimization problem at acceptable computational times. Unfortunately, heuristics methods do not guarantee optimality. So, it is reasonable to explore the possibility of improving these methods to obtain better solutions. Furthermore, the cost efficiency of the algorithms is always an objective to be achieved. With these targets, some iterated greedy algorithms are proposed in Chapters 4 and 5. These methods are based on a search that iterates through a process that destroys and repairs an incumbent solution in an attempt to generate a better candidate solution. This strategy combined with other optimization mechanisms has been successfully implemented for different scenarios, e.g., Ruiz and Stützle (2007), Ruiz and Stützle (2008), Pan and Ruiz (2014), Ding *et al.* (2015) and Tasgetiren *et al.* (2017). Greedy algorithms have also been quite effective to solve multi-objective problems,

e.g., Minella, Ruiz and Ciavotta (2011), Dubois-Lacoste, Lopez-Ibanez and Stutzle (2011) and Ciavotta, Minella and Ruiz (2013).

Despite their flexibility, heuristic methods require great effort to properly design and calibrate the algorithms. Moreover, there is no unique optimal configuration suitable for all problems and instances. Adaptive meta-heuristics alleviate this problem by adjusting their behavior during the search to obtain better results. Focusing on this mechanism, in Chapter 6 an adaptive large neighborhood search (ALNS) algorithm is also proposed. The ALNS heuristic, originally proposed by Ropke and Pisinger (2006), extends the large neighborhood search (LNS) heuristic proposed by Shaw (1998). In LNS, a destroy and a repair method iteratively rebuild an initial solution in an attempt to improve it. In ALNS, multiple destroy and repair methods are used, and the performance of each method determines how often that particular method is executed during the search. ALNS have been mainly applied in vehicle routing problems (VRP), e.g., Hemmelmayr, Cordeau and Crainic (2012), Qu and Bard (2012), Demir, Bektaş and Laporte (2012) and Azi, Gendreau and Potvin (2014). However, the number of scheduling applications has also grown in recent years, e.g., Lin and Ying (2014), Rifai, Nguyen and Dawal (2016) and Beezão *et al.* (2017).

1.1 Objectives

The goal of this Thesis is to develop new methods to solve the NWT-FSP-SDST under the objective functions:

- (1) *Minimizing total tardiness subject to the constraint that makespan can not be greater than a given value.*
- (2) *Minimizing total completion time subject to the constraint that makespan can not be greater than a given value; and*
- (3) *Minimizing makespan subject to the constraint that total completion time can not be greater than a given value.*

To achieve this goal, the following objectives are defined:

- OBJ1** To provide a precise description of the NWT-FSP-SDST;
- OBJ2** To provide a literature review about the NWT-FSP with SDST and $\epsilon(PM_1|PM_2)$;
- OBJ3** To provide efficient methods to solve the NWT-FSP-SDST;
- OBJ4** To demonstrate the efficiency of the proposed methods through computational experiments.

The remaining content is structured as follows. Chapter 2 develops **OBJ1**, where the problem under consideration and important concepts are introduced. Chapter 3 provides a literature review which corresponds to **OBJ2**. Chapters 4, 5 and 6 are dedicated to

the NWT-FSP-SDST under the objective functions 1, 2 and 3, respectively. In each of these three chapters, **OBJ2**, **OBJ3** and **OBJ4** are developed specifically for its problem. Chapter 7 gives the final conclusions and some future research directions.

2 PROBLEM STATEMENT

In this work, the FSP is addressed with two manufacturing features: NWT and SDST. The performance measures makespan, total completion time, and total tardiness are approached in pairs to form multi-objective functions. The following sections are dedicated to describe the problem.

2.1 Flow shop

The items being processed in scheduling problems, referred to as jobs, usually can be decomposed into a number of operations, each of which requires a variety of resources and processing time. To be completed, the jobs generally go through several workstations composed by one or more processors, generally referred to as machines. A solution for a scheduling problem is called schedule, which is a specification of when and where each operation of a given job set is to be processed. There are many constraints that can restrict the timing of jobs, and those schedules that satisfy all constraints are known as feasible solutions. The problem consist of finding a feasible solution that minimizes a given objective; that is, an optimal solution. The objectives are almost always minimizing a function over all feasible solutions. Finding an optimal solution is often a hard task because the problem may be very large and complex. When optimality is unattainable, the objective can become to find the best sub-optimal solution possible (EMMONS; VAIRAKTARAKIS, 2013).

The FSP has widespread use and great practical importance. The flow shop is a processing system in which the operation sequence follows a chain precedence and all jobs visit the workstations in the same order. In a **pure** FSP, each job has m operations, each of which requires a different workstation. In other words, the shop contains m different workstations and all jobs require one operation on each of them. Most authors assume that the flow of work is unidirectional, and a job never revisits any workstation (BAKER; TRIETSCH, 2019). Figure 1 illustrates the flow of work in a pure FSP.

There are many variations on the general FSP. This thesis will be concerned with the most fundamental model called **simple** FSP, which has the following characteristics:

- There are n independent jobs that require processing;
- There are m workstations which are always available;
- All jobs are available simultaneously at the start (at time zero);
- Each workstation consists of a single machine (m -machine flow shop);

Figure 1 – Workflow in a pure FSP

Source: Prepared by the author.

- Each machine can handle only one operation at a time;
- Each job can be processed on one machine at a time;
- Each job requires m operations with known requirements and processing times greater than or equal to zero;
- No job visits a workstation more than once (no recirculation);
- The machine sequence is the same for all jobs;
- Once started, an operation must be processed to completion without interruption (no preemption).
- When a job completes processing on one machine, it is immediately available to the next;
- Intermediate storage between successive machines is unlimited.

2.2 No-wait

In a simple FSP, the jobs follow a fixed sequence of processors, but they are allowed to "overtake" each other. This makes the search process significantly harder because it is necessary to examine (at least implicitly) $(n!)^m$ different schedules in order to find an optimal sequence (i.e., $n!$ different possible job sequences for each one of the m machines available). There are situations, however, where only permutation schedules are acceptable, either because it is technologically impossible for one job to pass another, or just to keep things operationally simple. This implies that the queues in front of each machine operate according to the First In First Out (FIFO) discipline. The resulting schedule is called **permutation schedule**. A permutation schedule has the same job order on all machines

and can be fully determined by a vector $\pi = (\pi_1, \pi_2, \dots, \pi_n)$. As illustrated in the Figure 2, an operation can be started as soon as the job and machine are available and it does not need to be started immediately after the previous operation.

Figure 2 – Simple permutation FSP

Source: Prepared by the author.

The FSP with no-wait constraint may be a case in which only permutation schedule are feasible. The no-wait phenomenon implies that a job, once started, must flow through every workstation to completion without interruption. When all processing times are greater than zero, it is clear that the machines in NWT-FSP also operate according to the FIFO discipline and therefore only permutation schedule are possible. In Figure 3, it is shown how the no-wait constraint sometimes requires that the starting of the processing of a job on the first machine must be delayed to ensure that the job can go through the flow shop without having to wait for any machine.

Figure 3 – NWT-FSP

Source: Prepared by the author.

The no-wait condition can occur due to unavailability of space, technological or process requirements, unavailability of resources and many other particular reasons. Several cases are found in metal, chemical and food industries, where controlling some parameters of the work-in-process requires that each operation follows the previous one immediately (EMMONS; VAIRAKTARAKIS, 2013). Modern environments, such as flexible manufacturing systems, just-in-time and agile manufacturing, can also be modelled as no-wait scheduling problems (BERTOLISSI, 2000).

2.3 Sequence-dependent setup times

In many real-life situations, a machine must be prepared in some way before it can process a certain job. It may include cleaning, retooling, adjustments, inspection, rearrangement of work station, etc. The time needed for such preparation is called **setup time**. Machine setup time is a significant factor for production scheduling in all manufacturing environments, and minimizing the total time spent on setups is an important objective for the scheduler.

The setup time can be either attached or not attached from the process of a job (Figure 4). If it is attached, the setup can begin only if both machine and job are ready; that is, the setup cannot begin until the machine is available and the job is physically present. On the other hand, if it is not attached, the setup can begin even if the job is not ready for a machine as long as the machine is free. Thus, attached setup times can be treated as negative lags on the schedule.

Figure 4 – Difference between attached and not attached setups

Source: Adapted from Khalili and Naderi (2015).

The setup time can also be either sequence-independent or sequence-dependent. When the setup of a job is sequence-independent, it will be always constant for a determined job independently of the job that precedes it. Specially for the attached case, sequence-independent setup times can be simply incorporated to processing times instead of being considered explicitly. But when the setup is sequence-dependent, however, the length of the setup time is not constant because it depends on the difficulty involved in switching from one processing configuration to another. In those situations, it is not valid to absorb setup times in the processing times and therefore explicit considerations must be made.

Sequence-dependent setup times are commonly found where a single facility produces several kinds of items, or where a multipurpose machine carries out an assortment of jobs. In production systems such as chemical, printing, pharmaceutical, and automobile manufacturing, the setup operations are not only often required between jobs, but their times are also strongly dependent on the immediately preceding process on the same machine (KHALILI; NADERI, 2015).

2.4 Notation and classification system

To specify the problems concisely and precisely, the following notation and classification system will be used whenever possible.

2.4.1 Notation

INDEXES

i Machine
 j, k Job

PARAMETERS

n Number of jobs
 m Number of machines
 J_j Job j
 M_i Machine i
 O_{ij} Operation of J_j processed on M_i
 p_{ij} Processing time of O_{ij}
 d_j due date of J_j (i.e., the committed completion time)
 s_{jk}^i Setup time of O_{ik} after O_{ij}
 UB Upper bound

SETS

J Set of n jobs, $J = \{J_j \mid j = 1, 2, \dots, n\}$
 M Set of m machines, $M = \{M_i \mid i = 1, 2, \dots, m\}$

SCHEDULE-DEPENDENT VARIABLES

S_{ij} Start time of J_j on M_i
 C_j Completion time of J_j on M_m
 T_j Tardiness of J_j , $T_j = \max(C_j - d_j, 0)$

PERFORMANCE MEASURES (PM)

$\sum C_j$ Total completion time
 $\sum T_j$ Total tardiness
 C_{max} Makespan, $C_{max} = C_n$

2.4.2 Classification system

The problem classification system introduced by Graham et al. (1979) are used to specify the problems. The system consists of three fields separated by bars: $\alpha|\beta|\gamma$. The α field describes the machine environment or shop type and has just one entry. The β field provides details of special features and may contain multiple entries. The γ field describes the objective to be minimized (maximization criteria are rare and can easily be converted

to minimization). The objective functions addressed here have two performance measures. The notation utilized by T'kindt and Billaut (2006) for multicriteria objectives is used to describe the functions in the γ field.

MACHINE ENVIRONMENT - α field

F_m Simple flow shop

SPECIAL FEATURES - β field

nwt No-wait

s_{jk}^i Setup time of O_{ik} after O_{ij}

OBJECTIVE FUNCTION - γ field

$\epsilon(PM_1/PM_2)$ Minimize PM_1 subject to an upper-bound UB on PM_2

2.5 Problem formulation

Mathematical models are applied in many solution methods such as branch and bound, dynamic programming, and branch and price in order to solve scheduling problems. Yet, they are not among the most efficient solution algorithms due to limitations of computer capacity and lack of specified software. However, they still are the first natural starting point due to its capacity to explicitly describe all the characteristics of a scheduling problem. The mixed integer linear programming (MILP) model in this section is presented to serve this purpose.

Using previous notation, the three problem considered can be described as

$$F_m | nwt, s_{jk}^i | \epsilon \left(\sum T_j / C_{max} \right) \quad \text{problem 1}$$

$$F_m | nwt, s_{jk}^i | \epsilon \left(\sum C_j / C_{max} \right) \quad \text{problem 2}$$

$$F_m | nwt, s_{jk}^i | \epsilon \left(C_{max} / \sum C_j \right) \quad \text{problem 3}$$

The following model describes the three problems stated above:

$$\text{Minimize } Z = \begin{cases} C_{max} = C_n & \text{for problem 1} \\ \sum C_j = \sum_{j=1}^n C_j & \text{for problem 2} \\ \sum T_j = \sum_{j=1}^n \max \{C_j - dj, 0\} & \text{for problem 3} \end{cases} \quad (2.1)$$

Subject to:

$$\begin{aligned} \sum C_j &\leq UB && \text{for problem 1} \\ C_{max} &\leq UB && \text{for problems 2 and 3} \end{aligned} \quad (2.2)$$

$$\sum_{j=1}^n x_{jk} \leq 1, \quad k = 1, 2, \dots, n \quad (2.3)$$

$$\sum_{k=1}^n x_{jk} \leq 1, \quad j = 1, 2, \dots, n \quad (2.4)$$

$$x_{jk} + x_{kj} \leq 1, \quad j, k = 1, 2, \dots, n \quad (2.5)$$

$$\sum_{j=1}^n \sum_{k=1}^n x_{jk} = n - 1 \quad (2.6)$$

$$S_{1,k} = \max_{1 \leq j \leq n} \left\{ \max_{1 \leq l \leq m} \left\{ x_{jk} \left(S_{1,j} + \sum_{i=1}^l p_{i,j} + s_{jk}^l - \sum_{i=1}^{l-1} p_{ik} \right) \right\} \right\}, \quad k = 1, \dots, n. \quad (2.7)$$

$$C_j = S_{1,j} + \sum_{i=1}^m p_{i,j}, \quad j = 1, \dots, n, \quad j = 2, \dots, n \quad (2.8)$$

$$x_{jk} = \begin{cases} 1 & \text{if } J_j \text{ is processed immediately before } J_k, \\ 0 & \text{otherwise} \end{cases}, \quad j, k = 1, 2, \dots, n \quad (2.9)$$

Expression 2.1 defines the performance measure to be minimized, and expression 2.2 specify the upper bound of the objective function. Expressions 2.3–2.6 ensure that each job is assigned to only one position in the sequence of jobs. Expression 2.7 defines the start times of each job on the first machine. Expression 2.8 calculates the completion times of the jobs. Finally, expression 2.9 defines the binary decision variables.

Figure 5 is a gantt chart that illustrates the NWT-FSP-SDST with three jobs and three machines.

Figure 5 – NWT-FSP-SDST

Source: Prepared by the author.

3 LITERATURE REVIEW

Some researchers have proposed algorithms to minimize makespan, total completion time and total tardiness in NWT-FSP-SDST, which is summarized in Table 1. The most relevant studies include greedy algorithms (BIANCO; DELL'OLMO; GIORDANI, 1999; XU; ZHU; LI, 2012; LI *et al.*, 2018), simulated annealing (LEE; JUNG, 2005; ALDOWAISAN; ALLAHVERDI, 2015), hybrid genetic algorithm (FRANCA; JR; BURIOL, 2006), constructive heuristics (ARAÚJO; NAGANO, 2011; NAGANO; MIYATA; ARAÚJO, 2015), differential evolution (QIAN *et al.*, 2011; QIAN *et al.*, 2012), greedy randomized adaptive search procedure and evolutionary local search based (ZHU; LI; WANG, 2013), iterative algorithms (ZHU; LI; GUPTA, 2013), hybrid evolutionary cluster search (NAGANO; ARAÚJO, 2014), hybrid greedy algorithm (ZHUANG; XU; SUN, 2014), particle swarm optimization (SAMARGHANDI; ELMEKKAWY, 2014), genetic algorithms (SAMARGHANDI, 2015a; SAMARGHANDI, 2015b; ALDOWAISAN; ALLAHVERDI, 2015) and local search (MIYATA; NAGANO; GUPTA, 2019).

Table 1 – Research on NWT-FSP-SDST for C_{max} , $\sum C_j$ or $\sum T_j$

Reference	Approach	Objective
Bianco, Dell'Olmo and Giordani (1999)	Genetic Algorithm	C_{max}
Lee and Jung (2005)	Simulated Annealing	
Franca, Jr and Buriol (2006)	Hybrid Genetic Algorithm	
Araújo and Nagano (2011)	Constructive heuristic	
Xu, Zhu and Li (2012)	Genetic Algorithm	
Zhu, Li and Gupta (2013)	Iterative algorithm	
Zhu, Li and Wang (2013)	Randomized adaptive greedy	
Nagano and Araújo (2014)	Hybrid evolutionary cluster search	
Zhuang, Xu and Sun (2014)	Hybrid greedy algorithm	
Samarghandi and ElMekkawy (2014)	Particle swarm optimization	
Samarghandi (2015a)	Genetic Algorithm	
Samarghandi (2015b)	Genetic Algorithm	
Miyata, Nagano and Gupta (2019)	Local search	
Qian <i>et al.</i> (2011)	Differential evolution	$\sum C_j$
Qian <i>et al.</i> (2012)	Differential evolution	
Nagano, Miyata and Araújo (2015)	Constructive heuristic	$\sum T_j$
Aldowaisan and Allahverdi (2015)	Simulated Annealing, Genetic Algorithm	
Li <i>et al.</i> (2018)	Greedy algorithm	

Source: Adapted from Allahverdi (2016)

The literature on multi-objective optimization of NWT-FSP is relatively limited, especially when hierarchical objectives are addressed (Table 2). Allahverdi (2004) tried to minimize a linear combination of makespan and maximum tardiness under the condition of not allowing maximum tardiness to exceed a given value. Framinan and Leisten (2006) studied the FSP with the objective of minimizing makespan such that maximum tardiness is not greater than an acceptable limit. Aydılek and Allahverdi (2012) and Nagano, Almeida

and Miyata (2020) addressed the NWT-FSP with the objective of minimizing makespan under the constraint that mean completion time (or the equivalent total completion time) does not exceeding a maximum value. Allahverdi and Aydilek (2013) addressed the NWT-FSP and tried to minimize total completion time while keeping makespan less than or equal to an upper bound, and Allahverdi and Aydilek (2014) considered the same problem with separate setup times. Recently, Allahverdi, Aydilek and Aydilek (2018) proposed an algorithm to solve the NWT-FSP with the objective of minimizing total tardiness such that makespan does not exceed a given value, and Allahverdi, Aydilek and Aydilek (2020) studied the same problem with separate setup times.

Table 2 – Research on NWT-FSP for $\epsilon(M_1|M_2)$

Reference	Approach	Objective
Allahverdi (2004)	Iterative algorithm	$\epsilon(C_{max}, T_{max} T_{max})$
Framinan and Leisten (2006)	Dominant sequences	$\epsilon(C_{max} T_{max})$
Aydilek and Allahverdi (2012)	Simulated annealing, Local search	$\epsilon(C_{max} \sum C_j)$
Allahverdi and Aydilek (2013)	Local search	$\epsilon(\sum C_j C_{max})$
Allahverdi and Aydilek (2014)	Simulated annealing	$\epsilon(\sum C_j C_{max})$
Allahverdi, Aydilek and Aydilek (2018)	Simulated annealing	$\epsilon(\sum T_j C_{max})$
Allahverdi, Aydilek and Aydilek (2020)	Simulated annealing, Local search	$\epsilon(\sum T_j C_{max})$
Nagano, Almeida and Miyata (2020)	Greedy algorithm, Local search	$\epsilon(C_{max} \sum C_j)$

4 THE NWT-FSP-SDST WITH TOTAL TARDINESS SUBJECT TO MAKESPAN

In this chapter, the NWT-FSP-SDST with the objective of minimizing total tardiness subject to an upper bound on makespan is addressed. A greedy algorithm is proposed. The algorithm is capable of adjusting its destruction intensity according to the instance size and the number of iterations, keeping the mechanisms as simple and effective as possible. The proposed method is compared against the best algorithms for the similar problems found in the literature. The remaining content is structured as follows. The algorithms are described in Section 4.1. Section 4.2 presents the computational experiments. The final conclusions and some future directions are given in Section 4.3.

4.1 Heuristic algorithms

The problem of $Fm/nwt/\epsilon(\sum T_j \mid C_{max})$ has already been addressed in the literature by Allahverdi, Aydilek and Aydilek (2018) and Allahverdi, Aydilek and Aydilek (2020). In this work, the best algorithm of each study is implemented to solve the same problem with sequence-dependent setup times. The two methods are briefly explained in the next subsection, followed by a complete description of the proposed algorithm.

4.1.1 Literature algorithms

Allahverdi, Aydilek and Aydilek (2018) proposed the algorithm *AA* to solve the NWT-FSP with the objective of minimizing total tardiness subject to makespan. They adapted many existing algorithms for this problem and proves that *AA* performs better than all of them. In summary, the algorithm *AA* is composed by a simulating annealing (SA) algorithm and an insertion local search. First, the SA is performed to reduce $\sum T_j$ using a random pairwise exchange operator. Then, if the solution does not satisfy the constraint $C_{max} \leq M$, the insertion local search tries to improve the sequences to obtain a feasible solution. The algorithm repeat these steps until the stopping criterion is satisfied.

Allahverdi, Aydilek and Aydilek (2020) studied the same problem with separate setup times. These setups can be classified as simple because the preparation times do not depend on production in previous periods. They proposed the algorithm *PA*, which was shown to outperform different existing algorithms modified for this environment. The algorithm *PA* addresses the problem in two phases. In the first phase, a SA algorithm utilizes block insertion and block exchange operators to reduce $\sum T_j$. In the second phase, an insertion local search is performed as an attempt to find a feasible solution under the condition $C_{max} \leq M$. These steps are repeated until the stopping criterion is satisfied.

It should be noted that, in both methods, the process responsible for trying to

satisfy the restriction on makespan is done after the optimization of total tardiness and it does not accept worse solutions. Since optimizing total tardiness and makespan can be competing objectives, these algorithms tend to get stuck. Thus, the last process may not be able to satisfy the constraint, and this is in fact what the experiments pointed out. For each experiment in which *AA* or *PA* was not able to produce a feasible solution, we took the initial solution as final solution to avoid making fundamental changes to the algorithms.

4.1.2 Iterated Greedy Algorithm (*IG_A*)

The proposed IG (Algorithm 1 and 2) takes an initial solution and tries to improve it by cycling through a main loop composed by three phases: destruction, construction and acceptance criteria. First, the destruction process remove a percentage of jobs d uniformly at random from a complete solution. This simple stochastic method provides a fast destruction and at the same time reduces the chance of entrapment at local optima. The next step is the (re)construction of a complete candidate solution. This phase is based on the NEH mechanism (NAWAZ; ENSCORE; HAM, 1983), where the partial solution is iteratively extended by reinserting the last removed element back, one at a time, and testing it in all possible positions. The best solution is taken at the end of each iteration, which in this case is the sequence that minimizes total tardiness and satisfies the upper bound constraint on makespan. These steps are repeated until all jobs are reinserted. When the construction phase is not able to build a feasible solution, the initial solution from the destruction phase is returned. Finally, the acceptance criterion tries to decrease total tardiness by only accepting better candidate solutions to update the incumbent.

The percentage d is not fixed across different instance sizes or while the algorithm is solving a problem. Instead, some search-dependent properties are used by the algorithm to modify this value in order to find an appropriate balance between search diversification and intensification. Diversification is explored by removing a large number of jobs to drive the search to rather distant solutions in the construction phase. Intensification, on the other hand, is explored by removing few jobs to focus the search on more localized regions. Some initial experiments suggested that a diversification strategy is more effective at the beginning of the search and have the cost of higher computation times for large problems. Considering this observation, the algorithm was designed so that the initial value of d is inversely proportional to the number of jobs, and it decreases after each iteration. The adjustment is made at the end of each loop by multiplying d by the constant f defined as

$$f(n) = \frac{1}{\exp(\ln(2) \times (n/P)^Q)},$$

where n is the number of jobs. The values P and Q are parameters that have to be calibrated. The parameter P defines the number of jobs needed for f to reach 50%. Q is

a parameter used to attenuate the change rate of the function. Figure 6 illustrates the function with some values set for parameters P and Q.

Figure 6 – Examples of function $f(n)$

Source: Prepared by the author.

Algorithm 1 IG_A

```

1: Initialize:  $P, Q, \pi_0$ ;
2:  $\pi := \pi_0$ ;
3:  $F := \sum T_j(\pi)$ ;
4:  $f := 1 / \exp(\ln(2) \times (n/P)^Q)$ 
5:  $d := f$ ;
6: repeat
7:    $\pi' := \pi$ ;
8:    $\pi' := \text{DestroyRepair}(\pi', d)$ ;
9:    $F' := \sum T_j(\pi')$ ;
10:  if  $F' < F$  then
11:     $\pi := \pi'$ ;
12:     $F = F'$ ;
13:  end if
14:   $d = d \times f$ ;
15: until Time Limit
16: return  $\pi$ ;
```

Algorithm 2 DestroyRepair

```

1: Initialize:  $\pi, d$ ;
2:  $\pi_d := \pi$ ;
3:  $D := n \times d$ ; # integer
4:  $\pi_e := D$  random jobs removed from  $\pi_d$ 
5: while  $\pi_e$  is not empty do
6:    $\pi_d :=$  best feasible sequence gener-
     ated by inserting the last job of  $\pi_e$ 
     in all positions of  $\pi_d$ . Break the loop
     if no feasible solution is found.
7: end while
8: if  $\pi_d$  is feasible then
9:    $\pi := \pi_d$ 
10: end if
11: return  $\pi$ ;
```

The IRACE software package (LOPEZ-IBANEZ *et al.*, 2016) was used to determine the appropriate parameter settings. This package automatically find the most appropriate parameter values for optimization algorithms given a set of instances and parameter ranges. A set of 120 instances was created for this purpose, composed by all combinations of $n \in \{5, 10, 20, 40, 80, 160\}$ and $m \in \{4, 8, 12, 16\}$, with 5 different problems for each combination $n \times m$. It was considered for calibration $P \in \{150, 300, 450, 600, 750, 900\}$ and $Q \in \{4^{-1}, 3^{-1}, 2^{-1}, 1, 2, 3, 4\}$. The processing times have a uniform distribution in the range $[1, 99]$. The sequence dependent setup times are at most 10% of the maximum processing

times. In order to generate due dates, a uniform distribution between $L(1 - T - R/2)$ and $L(1 - T + R/2)$ was used. Parameter T represents a tardiness factor, R represents the due date range, and L denotes an approximate value for makespan. This is a consolidate approach in the literature to generate due dates, e.g. (ALDOWAISAN; ALLAHVERDI, 2012), (FERNANDEZ-VIAGAS; FRAMINAN, 2015), (FRAMINAN; PEREZ-GONZALEZ, 2018). The values 0.25 and 1.0 were defined for T and R , respectively. These values were chosen to uniformly distribute the due dates between 0 and L . The tuning was performed using a computation time limit of $n \times (m/2) \times 25$ *milliseconds* as stopping criterion (RUIZ; STÜTZLE, 2008). The best parameter values obtained are $P = 300$ and $Q = 2$.

4.2 Computational experiments

All algorithms were implemented in C++. The computer used was a PC with an AMD Quad-Core Processor A12-9720P 3.60 GHz and 8 GB RAM running under a Windows 10 operating system. This study considers the test problems proposed by Minella, Ruiz and Ciavotta (2008) and Ruiz and Stützle (2008), which are extensions of the benchmarks of Taillard (1993). The processing times have a uniform distribution in the range $[1, 99]$. We used five instance sets with ratios of setup times $s \in \{0, 10, 50, 100, 125\}$ of the maximum processing times. Each set consists of 10 problems for each combination $n \times m$ of $\{20, 50, 100\} \times \{5, 10, 20\}$ and $200 \times \{10, 20\}$. The first set called SDST0 is taken from Minella, Ruiz and Ciavotta (2008) and have no setup times. The remaining sets called SDST10, SDST50, SDST100, and SDST125 are taken from Ruiz and Stützle (2008) and have times uniformly distributed in the range $[1, 9]$, $[1, 49]$, $[1, 99]$ and $[1, 124]$, respectively. Thus, there are 550 instances in total. The initial solutions were generated by optimizing makespan on random solutions, with a local search based on pairwise exchanges in adjacent jobs with neighborhood space of size $(n - 1)$. The upper bounds were defined as the makespan values of the initial solutions. All methods were tested by using the same initial solutions and upper bounds.

The stopping criterion used was the CPU time limit defined as $n \times (m/2) \times t$ *milliseconds*. The experiments were performed for $t \in \{10, 20, 30, 40\}$ to analyse the algorithms in different available computational times. The solutions were evaluated by using the average relative percentage deviation (*ARPD*) defined as

$$ARPD = \frac{100}{N} \sum_{i=1}^N \frac{(\sum T_j)_i^h - (\sum T_j)_i^{best}}{(\sum T_j)_i^h},$$

where *ARPD* represents the performance of a heuristic h . In other words, this measure consists of the arithmetic mean of the deviations of a heuristic h from each best known solution. Therefore, the best heuristic is the one with the lowest *ARPD* value.

Table 3 – ARPD results of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j|C_{max})$

(a) jobs [n] × machines [m]					(b) setup [s] × time [t]				
Heuristic		IG_A	AA	PA	Heuristic		IG_A	AA	PA
n	m				s	t			
20	5	3.75	55.51	48.86	0	10	39.93	85.79	61.70
	10	6.76	82.07	37.66		20	4.87	75.48	57.11
	20	19.76	106.97	75.39		30	2.03	72.14	48.54
50	5	1.83	47.65	34.93		10	40	0.85	71.28
	10	1.88	58.04	41.17	10		4.65	58.63	41.67
	20	1.60	77.56	45.15	20		1.92	58.88	35.77
100	5	1.39	43.25	35.39	50		30	1.09	69.62
	10	1.38	47.37	40.31		40	1.30	61.37	52.78
	20	1.08	55.55	42.34		10	9.34	51.11	40.25
200	10	6.75	22.77	19.50		100	20	1.56	55.93
	20	1.24	25.71	21.24	30		0.86	55.99	32.50
	Average	4.23	56.28	39.98	40		0.75	57.02	33.41
125	10	3.32	39.99	31.84	Average		20	2.08	43.95
	20	1.89	44.62	29.36		30	1.72	48.18	34.05
	30	1.72	48.18	34.05		40	1.72	48.18	34.05
	Average	4.23	56.28	39.98					

The ARPD results over jobs and machines are presented in Table 3a. Table 3b gives the results over setup and time factors. It should be noted that the ARPD values of IG_A are disproportionately higher for time factor $t = 10$. The reason for this is that the time limits generated in this condition may not be sufficient for the algorithm to complete the first iteration, especially in cases with small instances. Others minor differences can be explained by the fact that the benchmarks used have due dates produced by different strategies. Despite that, the performance behaviors and their differences are very clear, as can be seen in Figure 7. As noted, the differences between the algorithms tend to decrease when the setup factor and especially the number of jobs increase. On the other hand, in general, the errors get bigger when the number of machines increases. Variations in the time factor do not appear to have much effect for values above 10. In all cases, IG_A has a significant advantage over the other methods. The overall ARPD values of IG_A , AA, and PA are 4.23, 39.98, and 56.28, respectively.

The Tukey' honestly significant difference (HSD) test was conducted to analyse the statistic difference between methods. The Tukey'HSD test is a multiple comparison procedure that compares all possible pairs of means. The null hypotheses that two algorithms have equal performances was tested at a significance level of 5%. The results in

Figure 7 – ARPD trends of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j|C_{max})$

Source: Prepared by the author.

Table 4 – Tukey' HSD test of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j|C_{max})$

Methods		Mean Difference (I-J)	Std. Error	Sig.	95% confidence interval		Reject
I	J				Lower Bound	Upper Bound	
AA	IG _A	52.06	1.28	.000	49.04	55.07	True
	PA	16.30	1.28	.000	13.29	19.31	True
IG _A	AA	-52.06	1.28	.000	-55.07	-49.04	True
	PA	-35.76	1.28	.000	-38.77	-32.74	True
PA	AA	-16.30	1.28	.000	-19.31	-13.29	True
	IG _A	35.75	1.28	.000	32.74	38.77	True

FWER = 0.05

Table 4 show that all algorithms are statistically different from each other. The differences at a confidence interval of 95% are illustrated in Figure 8. As can be noted, IG_A outperform the second best method by about 35% and the third best by about 51%.

4.3 Conclusion

In this chapter, the NWT-FSP-SDST is addressed. The objective is to minimize total tardiness subject to the constraint that makespan does not exceed a maximum

Figure 8 – Multiple Comparisons of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum T_j|C_{max})$

Source: Prepared by the author.

acceptable value. An IG algorithm was proposed to solve the problem. The proposed approach was tested against the existent algorithms *AA* and *PA*, which are designed to solve the most similar problems found in the literature. Experiments under the same computational conditions show that IG_A , *PA* and *AA* obtained overall *ARPD* values of 4.23, 39.98, and 56.28, respectively. Therefore, IG_A outperforms the other methods.

There are some issues that can be investigated in the future. First, the proposed algorithm was compared with the best literature methods for similar problems. However, adapting other methods for different applications in future experiments could be promising. Another option is to consider additional factors like number of machines to determine the destruction strength of the proposed algorithm. In this work, only the number of jobs was considered for this purpose, and therefore it may be possible to get better results with a broader approach. Additionally, the diversification/intensification behavior of the heuristic can be further explored. For example, it can be favorable to examine new strategies that combine greedy searching with random perturbations, both in destruction and construction phases. A study of best initial solutions can also be extension.

5 THE NWT-FSP-SDST WITH TOTAL COMPLETION TIME SUBJECT TO MAKESPAN

In this chapter, the NWT-FSP-SDST with sequence-dependent setup times and with the objective of minimizing total completion time subject to an upper bound on makespan is addressed. Four iterated greedy algorithms are proposed. The algorithms combine distinct destruction and construction mechanisms, where the search intensification-diversification as well as the greedy-random behavior are proposed at different levels. The best among them is compared against three methods for similar problems found in the literature. All results are statistically verified. The remainder of this chapter is organized as follows. The algorithms are presented in Section 5.1. Section 5.2 is dedicated to the experimental design and analysis of results. Some concluding remarks are given in Section 5.3.

5.1 Heuristic algorithms

As mentioned earlier, the problem of $Fm/nwt/\epsilon(\sum C_j|C_{max})$ has already been addressed in the literature by (ALLAHVERDI; AYDILEK, 2013) and (ALLAHVERDI; AYDILEK, 2014). In this work, the best algorithm of each study is implemented to solve the problem with SDST. An algorithm proposed by (YE; LI; NAULT, 2020) is also adapted to the problem. The three methods are briefly explained in the next subsection, followed by a detailed description of the proposed algorithms.

5.1.1 Literature algorithms

Allahverdi and Aydilek (2013) proposed the algorithm *PAL* to solve the NWT-FSP with the objective of minimizing total completion time subject to makespan. In summary, the algorithm *PAL* is composed by two local searches: Insertion and Interchange. The Insertion local search returns the best solution from $(n - 1)^2$ permutations by testing each job of a sequence in all possible positions. The Interchange local search tries to improve the incumbent solution by swapping pairs of adjacent jobs with a search space of $(n - 1)$ permutations. The authors proposed five versions of *PAL* (*PA1*, *PA5*, *PA10*, *PA15*, *PA20*) where the values L denotes the number of times that Insertion is performed. The *PA20* version was proved to generate better solutions, so we adapted it to our problem.

Allahverdi and Aydilek (2014) studied the same problem with separate setup times. These setups times are classified as simple because they are not influenced by the production sequence. The authors proposed the algorithm *ISA-2*, which was shown to outperform different algorithms proposed or modified for this environment. *ISA-2* is a simulated annealing algorithm that creates and tests two candidate solutions at each

iteration by using the exchange and insertion operators. The first operator exchanges the positions of two random jobs. The second operator inserts a random job into a random position. The algorithm repeats this process until its stop criteria defined by a final temperature is satisfied.

Ye, Li and Nault (2020) proposed the algorithm called *TOB* to the NWT-FSP. This method iterates through a reconstruction of the incumbent solution followed by a modified insertion local search. The reconstruction process is a combination of the NEH technique (NAWAZ; ENSCORE; HAM, 1983) with an exchange local search. The modified insertion local search inserts jobs only in positions ahead of the current one. The algorithm repeats these steps until a maximum number of iterations is achieved. Originally, this study consider a mathematical relationship between total completion time and makespan in the objective function. Here, we adapted the algorithm to our problem.

5.1.2 Iterated Greedy Algorithm ($IG_{\#}$)

The iterated greedy starts from an initial solution and then iterates through a main loop that consists of three steps: destruction, construction and acceptance criteria. In the first step, some jobs are removed from the incumbent solution. In the second step, a construction mechanism is used to reinsert the removed jobs back to the sequence, thus generating a complete candidate solution. Finally, the acceptance criteria decides whether the candidate should update the incumbent solution.

Table 5 – Destruction and construction mechanisms of IG_1 , IG_2 , IG_3 and IG_4

Algorithm	Mechanism	IG_1	IG_2	IG_3	IG_4
Algorithm 3	GreedyRandomDestroy			x	x
Algorithm 4	RandomDestroy	x	x		
Algorithm 5	GreedyRepair	x			x
Algorithm 6	GreedyRandomRepair		x	x	

We use two destruction and two construction mechanisms where different levels of random-greedy behavior are explored. Algorithm 3 is a semi-greedy destruction mechanism that tries to remove the best set of jobs that optimize the objective, but skips some options at random. Algorithm 4 is a simple and fast destruction heuristic that removes a predefined number of jobs from a sequence at random. Algorithm 5 is a greedy construction mechanism that reinserts each removed job back to the partial sequence in the best position possible. Algorithm 6 is a construction mechanism similar to the last one, but it differs by skipping some insertion options at random. One mechanism of each type is chosen to be used in lines 5 and 6 of the Algorithm 7 to build four greedy algorithms as shown on Table 5. All four algorithms have two parameters in common: P and Q . These values are

used to adjust the destruction intensity during execution. Some mechanisms also have a random behavior that is adjusted with the parameter R .

Algorithm 3 GreedyRandomDestroy

```

1: Initialize:  $\pi, \pi_e, d, p$ ;
2:  $i := 0$ ;
3:  $\pi'_e := \text{empty}$ 
4: while  $i < d$  do
5:    $\pi' := \pi$ 
6:   move a random job from  $\pi$  to the
     start of  $\pi_e$ ;
7:    $j := 1$ ;
8:   while  $j \leq n - i$  do
9:     if  $p < \text{random} \in [0, 1]$  then
10:      move the  $j$ -th job from  $\pi'$  to the
        start of  $\pi'_e$ ;
11:      if  $\sum C_j(\pi') < \sum C_j(\pi)$  then
12:         $\pi := \pi'$ 
13:         $\pi_e := \pi'_e$ 
14:      end if
15:      move the first job of  $\pi'_e$  back to
        the  $j$ -th position on  $\pi'$ ;
16:    end if
17:     $j := j + 1$ ;
18:  end while
19:   $i := i + 1$ ;
20: end while
21: return  $\pi$ ;

```

Algorithm 4 RandomDestroy

```

1: Initialize:  $\pi, \pi_e, d$ ;
2:  $i := d$ ;
3: while  $i \geq 1$  do
4:   move a random job from  $\pi$  to the
     start of  $\pi_e$ ;
5:    $i := i - 1$ ;
6: end while
7: return  $\pi$ ;

```

Algorithm 5 GreedyRepair

```

1: Initialize:  $\pi, \pi_e$ ;
2: while  $\pi_e$  is not empty do
3:    $\pi_d :=$  best feasible sequence build (if
     any) by inserting the last job of  $\pi_e$ 
     in all positions of  $\pi_d$ 
4:   if no feasible sequence was build
     then
5:      $\pi := \text{empty}$ 
6:     break; # while ends
7:   end if
8: end while
9: return  $\pi$ ;

```

The parameters were calibrated in a set of 120 instances, with 5 different problems for each combination $n \in \{5, 10, 20, 40, 80, 160\} \times m \in \{4, 8, 12, 16\}$, by using the IRACE software package (LOPEZ-IBANEZ *et al.*, 2016). We created this set by generating processing times with a uniform distribution in the range $[1, 99]$. The SDST has a uniform distribution in the range $[1, 9]$. The due dates were generated with a uniform distribution in the range $[L(1 - T - U/2), L(1 - T + U/2)]$, where T denotes a tardiness factor, U denotes the due date range, and L denotes an approximate value for makespan. This is a consolidated approach in the scheduling literature to generate due dates, e.g. Aldowaisan and Allahverdi (2012), Fernandez-Viagas and Framinan (2015), Framinan and Perez-Gonzalez (2018). The values 0.25 and 1.0 were defined for T and U , respectively. The tuning was performed by using a computation time limit of $n \times (m/2) \times 25$ *milliseconds* as stopping criterion (RUIZ; STÜTZLE, 2008). The parameter values considered are given in Table 6.

Algorithm 6 GreedyRandomRepair

```

1: Initialize:  $\pi_d, \pi_e, p$ ;
2:  $\pi' := \text{empty}$ 
3: while  $\pi_e$  is not empty do
4:   move the last job of  $\pi'_e$  to the first
     position on  $\pi_d$ ;
5:    $i := 1$ ;
6:   while  $i \leq n$  do
7:     if  $C_{max}(\pi_d) \leq UB$  then
8:       if  $\pi'$  is empty then
9:          $\pi' := \pi_d$ 
10:      else
11:         $r := \text{random}$ ;
12:        if  $p < r$  and  $\sum C_j(\pi_d) < \sum C_j(\pi')$  then
13:           $\pi' := \pi_d$ 
14:        end if
15:      end if
16:    end if
17:    move the  $i$ -th job of  $\pi_d$  to position
       $i + 1$ ;
18:     $i := i + 1$ ;
19:  end while
20:   $\pi_d := \text{empty}$ 
21:  if  $\pi'$  is empty then
22:    break; # while ends
23:  else
24:     $\pi_d := \pi'$ 
25:     $\pi' := \text{empty}$ 
26:  end if
27: end while
28: return  $\pi_d$ ;

```

Algorithm 7 DestroyRepair

```

1: Initialize:  $\pi, d, [R]$ ;
2:  $\pi_d := \pi$ ;
3:  $\pi_e := \text{empty}$ ;
4:  $D := n \times d$ ; # integer
5:  $\pi_d := \text{DestroyMechanism}(\pi_d, \pi_e, d, [R \times d])$ 
6:  $\pi_d := \text{RepairMechanism}(\pi_d, \pi_e, [R \times d])$ 

7: if  $\pi_d$  is not empty then
8:    $\pi := \pi_d$ 
9: end if
10: return  $\pi$ ;

```

Algorithm 8 Iterated Greedy

```

1: Initialize:  $\pi_0, P, Q, [R]$ ;
2:  $\pi := \pi_0$ ;
3:  $F := \sum C_j(\pi)$ ;
4:  $f := 1 / \exp(\ln(2) \times (n/P)^Q)$ 
5:  $d := f$ ;
6: repeat
7:    $\pi' := \pi$ ;
8:    $\pi' := \text{DestroyRepair}(\pi', d, [R])$ ;
9:    $F' := \sum C_j(\pi')$ ;
10:  if  $F' < F$  then
11:     $\pi := \pi'$ ;
12:     $F = F'$ ;
13:  end if
14:   $d = d \times f$ ;
15: until Time Limit
16: return  $\pi$ ;

```

Table 6 – Parameter calibration of IG_1 , IG_2 , IG_3 and IG_4

Parameter	Considered values	Selected values			
		IG_1	IG_2	IG_3	IG_4
P	150, 300, 450, 600, 750, 900	750	750	450	450
Q	4^{-1} , 3^{-1} , 2^{-1} , 1, 2, 3, 4	2	2	3	3
R	0.0, 0.2, 0.4, 0.6, 0.8, 1.0	-	0.0	0.2	0.8

5.2 Computational experiments

All algorithms were implemented in C++. The PC used was an AMD Quad-Core Processor A12-9720P 3.60 GHz and 8 GB RAM. This study considers the problem instances of Ruiz and Stützle (2008), which are extensions of the Taillard's benchmark (TAILLARD, 1993). The extensions are divided in four sets, each consisting of 10 problems for each combination $n \times m$ for $\{20, 50, 100\} \times \{5, 10, 20\}$ and $200 \times \{10, 20\}$. The processing times have a uniform distribution in the range $[1, 99]$. Each set has setup times uniformly distributed in the ranges $[1, 9]$, $[1, 49]$, $[1, 99]$ and $[1, 124]$, respectively. This means that in total, there are 440 different problem instances. The stopping criterion is based on an interval CPU time given by $n \times (m/2) \times t$ *milliseconds*, with $t \in \{10, 20, 30, 40\}$ to be able to analyze consistency across different computational times. The initial solutions were generated by using a pairwise exchange local search with neighborhood space of size $(n - 1)$. The upper bounds K were defined as the makespan values of the initial solutions. The same initial solutions were used for all methods. The solutions were evaluated by using the Average Relative Percentage Deviation (*ARPD*) defined as

$$ARPD = \frac{100}{N} \sum_{i=1}^N \frac{(\sum C_j)_i^h - (\sum C_j)_i^{best}}{(\sum C_j)_i^h},$$

where *ARPD* represents the performance of a heuristic h . In summary, this measure represents the arithmetic mean of the deviations from the best solutions found. Therefore, the best heuristic is the one with the lowest *ARPD* value.

A preliminary analysis was performed with IG_1 , IG_2 , IG_3 and IG_4 in order to decide which algorithm should be chosen to be compared with the literature methods. This analysis is illustrated in Figure 9. As can be noted, IG_4 presents consistently better performance across different numbers of jobs and machines. For this reason, IG_4 was selected for the subsequent experiments.

Figure 9 – ARPD trends of IG_1 , IG_2 , IG_3 and IG_4

Source: Prepared by the author.

Table 7 – ARPD results of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j|C_{max})$

(a) jobs [n] × machines [m]						(b) setup [s] × time [t]					
Heuristic	IG_4	ISA-2	PAL	TOB		Heuristic	IG_4	ISA-2	PAL	TOB	
n	m					s	t				
20	5	0.10	1.01	3.99	1.87	10	10	0.51	1.68	16.31	11.11
	10	0.04	0.80	2.90	1.04		20	0.38	1.75	12.80	6.13
	20	0.04	0.51	2.07	1.07		30	0.34	1.68	10.81	1.18
	40	0.26	1.74	9.44	1.11		40	0.26	1.74	9.44	1.11
50	10	0.40	1.88	14.50	9.77	50	10	0.40	1.88	14.50	9.77
	20	0.34	1.92	11.43	5.45		20	0.34	1.92	11.43	5.45
	30	0.21	1.85	9.71	1.27		30	0.21	1.85	9.71	1.27
	40	0.34	1.71	8.56	1.23		40	0.34	1.71	8.56	1.23
100	10	0.47	2.36	13.21	8.80	100	10	0.47	2.36	13.21	8.80
	20	0.39	2.22	10.50	5.15		20	0.39	2.22	10.50	5.15
	30	0.29	2.17	9.04	1.57		30	0.29	2.17	9.04	1.57
	40	0.30	2.16	8.07	1.53		40	0.30	2.16	8.07	1.53
200	10	0.54	2.55	13.27	8.47	125	10	0.54	2.55	13.27	8.47
	20	0.36	2.44	10.59	5.11		20	0.36	2.44	10.59	5.11
	30	0.35	2.47	9.14	1.60		30	0.35	2.47	9.14	1.60
	40	0.34	2.48	8.21	1.54		40	0.34	2.48	8.21	1.54
Average						Average					
0.36						0.36					
2.07						2.07					
10.97						10.97					
4.44						4.44					

Figure 10 – ARPD trends of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j|C_{max})$

Source: Prepared by the author.

Table 8 – Tukey' HSD test of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j|C_{max})$

Methods		Mean Difference (I-J)	Std. Error	Sig.	95% confidence interval		Reject
I	J				Lower Bound	Upper Bound	
IG ₄	ISA-2	-1.70	0.27	.000	-2.40	-1.00	True
	PAL	-10.61	0.27	.000	-11.31	-9.91	True
	TOB	-4.07	0.27	.000	-4.77	-3.38	True
ISA-2	IG ₄	1.70	0.27	.000	1.00	2.40	True
	PAL	-8.91	0.27	.000	-9.60	-8.21	True
	TOB	-2.27	0.27	.000	-3.07	-1.68	True
PAL	IG _A	10.61	0.27	.000	9.91	11.31	True
	ISA-2	8.91	0.27	.000	8.21	9.60	True
	TOB	6.53	0.27	.000	5.84	7.23	True
TOB	IG _A	4.08	0.27	.000	3.38	4.77	True
	ISA-2	2.37	0.27	.000	1.68	3.07	True
	PAL	-6.53	0.27	.000	-7.23	-5.84	True

FWER = 0.05

Figure 11 – Multiple Comparisons of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(\sum C_j|C_{max})$

Source: Prepared by the author.

The further analysis focuses on the algorithms *PAL*, *TOB*, *ISA-2* and *IG₄*. The *ARPD* results over jobs and machines are presented in Table 7a. Table 7b gives the results over setup and time factors. It should be noted that the *ARPD* values of *PAL* and *TOB* tend to grow up when the number of jobs increases. As these algorithms are mainly based in local searches, the exponential behavior is expected. However, the two methods get better when more computational time is available. Variations in setup factor do not appear to have much effect. Again, *IG₄* presents consistently better performance when compared to other methods. These and other trends can be seen in Figure 10. The overall *ARPD* values of *PAL*, *TOB*, *ISA-2* and *IG₄* are 10.97%, 4.44%, 2.07% and 0.36%, respectively.

The Tukey' honestly significant difference (HSD) test was conducted to verify whether the means are significantly different from each other. Table 8 shows the results of the test at a significance level of 5%. As noted, the test rejected the hypothesis of equal means for all pairwise comparisons, confirming that all methods are statistically different. Figure 11 gives the means plot with confidence intervals of 95%. The illustration clearly shows that *IG₄* is the best performing algorithm, yielding the lowest *ARPD* value without overlapping intervals.

5.3 Conclusion

Scheduling problems involving SDST are very common in multi-purpose production environments. With a multicriteria approach, the constraint was addressed in a NWT-FSP with the objective of minimizing total completion time subject to makespan. We proposed four algorithms in which the search intensification-diversification as well as the greedy-random behavior are explored at different levels. The best method proposed, called *IG₄*, was compared to the algorithms *ISA-2*, *TOB* and *PAL*, which were designed to solve the most similar problems found in the literature. We performed computational experiments setting the same computational times for all methods. The results show that the proposed approach outperform the existent methods in different instance configurations. The general performance in *ARPD* of *IG₄*, *ISA-2*, *TOB* and *PAL* are 0.36%, 2.07%, 4.44%, and 10.97%, respectively. Therefore, *IG₄* is recommended for this problem.

Since this is a significant scheduling problem that has never been considered that way, this work fills an important gap in scheduling theory. However, there are still some issues that deserve further consideration. Other methods for different applications, for example, can be adapted and included in future experiments in order to get additional insights. Improve the proposed method is another option. One strategy to improve it could be to consider other instance parameters to determine the destruction strength of the algorithm (for example, the number of machines). In this work, only the number of jobs was considered for this purpose, and therefore it may be possible to get better results with a broader approach.

6 THE NWT–SDST WITH MAKESPAN SUBJECT TO TOTAL COMPLETION TIME

In this chapter, the NWT-FSP-SDST with sequence-dependent setup times and with the objective of minimizing makespan subject to an upper bound on total completion time is addressed. We present the algorithm called $ALNS_A$. The algorithm accesses a set of distinct mechanisms and dynamically select a pair of destroy and repair methods based on their performance history. The balance between greediness and randomness as well as between intensification and diversification in each method are proposed at different levels. Extensive experiments are made to compare $ALNS_A$ against three heuristics for similar problems found in the literature. All results are statistically verified. The remainder of this chapter is organized as follows. The algorithms are presented in Section 6.1. Section 6.2 is dedicated to the experimental design and analysis of results. Some concluding remarks are given in Section 6.3.

6.1 Heuristic algorithms

As mentioned earlier, the problem of $Fm/nwt/\epsilon(C_{max}/\sum C_j)$ has already been addressed in the literature by Aydilek and Allahverdi (2012) and Nagano, Almeida and Miyata (2020). In this work, the best algorithm of each study is implemented to solve the problem with sequence-dependent setup times. An algorithm proposed by Ye, Li and Nault (2020) is also adapted to the problem. The three methods are briefly explained in the next subsection, followed by a complete description of the proposed algorithm.

6.1.1 Literature algorithms

Aydilek and Allahverdi (2012) proposed the algorithm $HH1$, composed by the algorithms called mSA and HA . The algorithm mSA is a modified simulated annealing, which tries to improve the incumbent solution by iteratively moving a random job to a random position. The algorithm HA applies in a loop an insertion local search. Then, it tries to improve the solution generated by using an interchange local search (local search that swaps random pairs of jobs). Only solutions with C_j less than or equal to a given upper bound can be candidates to update the incumbent solution. The final solution generated by mSA is used as initial solution for the HA .

Nagano, Almeida and Miyata (2020) proposed the algorithm GL for the same problem. GL iterates through a reconstruction procedure followed by an insertion local search. The reconstruction step creates candidate solutions by removing random jobs from the incumbent solution and reinserting them back by using a constructive heuristic based on NEH (NAWAZ; ENSCORE; HAM, 1983). This loop repeats until a defined number of

iterations is achieved. Finally, a transposition local search (local search that swap pairs of adjacent jobs) tries to improve the incumbent solution. Throughout the execution, only sequences with C_j less than or equal to the upper bound are considered candidate solutions.

Ye, Li and Nault (2020) proposed the algorithm called *TOB*. First, this method performs a reconstruction process that combines the NEH heuristic with an interchange local search. Then, a modified insertion local search is applied (local search that inserts jobs only ahead of its original position). The algorithm repeats these steps until a maximum number of iterations is achieved. In this work, we implement *TOB* to minimize makespan subject to total completion time. Originally, the objective function was defined as a mathematical relationship between the two performance measures.

6.1.2 Algorithm $ALNS_A$

The Adaptive Large Neighborhood Search, originally proposed by Ropke and Pisinger (2006), allows multiple methods to be used within the search process. The algorithm has access to a set of destructive and constructive heuristics. At each iteration, a given solution is destroyed by a destructive procedure and rebuilt by a constructive procedure. A weight is assigned to each heuristic according to its previous performance, giving to those that perform well a higher probability to be selected. Usually, the weights are updated so that all heuristics have at least some chance to be executed. This dynamically selection adapts the algorithm to the instance and the state of the search at hand (PISINGER; ROPKE, 2019).

The proposed $ALNS_A$ (Algorithm 9) has access to the set Ω^- of destroy heuristics and the set Ω^+ of repair heuristics. The set Ω^- is composed by Algorithms 10, 11, and 12. Algorithm 10 removes a predefined number of jobs from a sequence at random. Algorithm 11 tries to remove from a sequence the best set of jobs that optimize the objective. Algorithm 12 is similar to Algorithm 11, but it differs by skipping some jobs at random. The set Ω^+ is composed by Algorithms 13, 14, and 15. Algorithm 13 reinserts the removed jobs back to the partial sequence at random positions. Algorithm 14 reinserts each removed job back to the partial sequence in the best possible position. Algorithm 15 is similar to Algorithm 14, but it differs by skipping some insertion options at random. The variables $\rho_i^- \in \mathbb{R}^{|\Omega^-|}$ and $\rho_i^+ \in \mathbb{R}^{|\Omega^+|}$ store the weights of destroy and repair methods, respectively.

The following expressions describe how Algorithm 9 calculates the probabilities ϕ_i^- and ϕ_i^+ to select the i th destroy and repair heuristics, respectively.

$$\phi_i^- = \frac{\rho_i^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-}, \quad \phi_i^+ = \frac{\rho_i^+}{\sum_{k=1}^{|\Omega^+|} \rho_k^+}.$$

After each iteration of Algorithm 9, a score Ψ for the last pair of methods used is computed as follows

$$\Psi = \max \begin{cases} 4, & \text{if generated solution is a new global best;} \\ 3, & \text{if generated solution is a new local best;} \\ 2, & \text{if generated solution is accepted;} \\ 1, & \text{if generated solution is rejected.} \end{cases}$$

Initially, we define $\rho^- = (1, 1, 1)$ and $\rho^+ = (1, 1, 1)$. Then, the methods have their weights updated by computing

$$\rho_i^- = \lambda \rho_i^- + (1 - \lambda) \Psi \alpha_i^-, \quad \rho_i^+ = \lambda \rho_i^+ + (1 - \lambda) \Psi \alpha_i^+,$$

where $\lambda \in [0, 1]$ is a parameter that controls the sensibility to change in performance. The variables α_i^- and α_i^+ represent the values used to normalize the scores regarding CPU time. This normalization is important because some heuristics are significantly faster than others, so the correction ensures a proper trade-off between CPU time and solution quality. Note that only the weights corresponding to the methods used in the current iteration are updated.

The parameter λ was calibrated in a set of 120 instances, with 5 different problems for each combination $n \in \{5, 10, 20, 40, 80, 160\} \times m \in \{4, 8, 12, 16\}$, by using the IRACE software package (LOPEZ-IBANEZ *et al.*, 2016). We create this set by generating processing times with a uniform distribution in the range $[1, 99]$. The SDST have a uniform distribution in the range $[1, 9]$. The tuning was performed by using a computation time limit of $n \times (m/2) \times 25$ *milliseconds* as stopping criterion (RUIZ; STÜTZLE, 2008). We consider $\lambda \in \{0.0, 0.1, \dots, 1.0\}$ for tuning in IRACE and the result obtained was $\lambda = 0.3$. The values for normalization are $\alpha^- = (0.98, 0.3, 0.72)$ and $\alpha^+ = (0.99, 0.44, 0.57)$. The parameters T_i (initial temperature), and c_f (cooling factor) control the convergence of the algorithm. Using as reference the test for the simulated annealing heuristic performed by (AYDILEK; ALLAHVERDI, 2012), we define $T_i = 0.1$ and $c_f = 0.98$.

Algorithm 9 $ALNS_A$

```

1: Initialize:  $\lambda, \pi_0, T_i, c_f$ ;
2:  $\rho^- := (1, 1, 1)$ ;  $\rho^+ := (1, 1, 1)$ ;
3:  $\pi_b := \pi_0$ ;
4:  $\pi' := \pi_b$ ;
5:  $T := T_i$ ;
6: repeat
7:    $\pi := \pi'$ ;
8:   select  $d \in \Omega^-$  and  $r \in \Omega^+$  using  $\rho^-$ 
     and  $\rho^+$ ;
9:    $\pi'' := r(d(\pi))$ ;
10:  if  $\pi'' \neq \text{empty}$  then
11:     $\pi := \pi''$ ;
12:  end if
13:  if  $C_{max}(\pi) < C_{max}(\pi')$  then
14:     $\pi' := \pi$ ;
15:    if  $C_{max}(\pi) < C_{max}(\pi_b)$  then
16:       $\pi_b := \pi$ ;
17:    end if
18:  else
19:     $\Delta := \frac{C_{max}(\pi) - C_{max}(\pi')}{C_{max}(\pi')}$ ;
20:     $p := \exp(-\Delta/T)$ ;
21:     $q := \text{random}$ ;
22:    if  $p > q$  then
23:       $\pi' := \pi$ ;
24:    end if
25:  end if
26:   $T := T \times c_f$ ;
27:  update  $\rho^-$  and  $\rho^+$ ;
28: until Time Limit
29: return  $\pi$ ;

```

Algorithm 10 RandomDestroy

```

1: Initialize:  $\pi$ ;
2:  $\pi_e := \text{empty}$ 
3:  $i := n/2$ ;
4: while  $i > 0$  do
5:   move a random job from  $\pi$  to the
     start of  $\pi_e$ ;
6:    $i := i - 1$ ;
7: end while
8: return  $\pi, \pi_e$ ;

```

Algorithm 11 GreedyDestroy

```

1: Initialize:  $\pi$ ;
2:  $\pi_e := \text{empty}$ 
3:  $i := n/2$ ;
4: while  $i > 0$  do
5:   move the best job that minimize
      $\sum C_{max}(\pi)$  from  $\pi$  to the start of
      $\pi_e$ ;
6:    $i := i - 1$ ;
7: end while
8: return  $\pi, \pi_e$ ;

```

Algorithm 12 GreedyRandomDestroy

```

1: Initialize:  $\pi$ ;
2:  $\pi_e := \text{empty}$ ;
3:  $i := 0$ ;
4: while  $i < (n/2)$  do
5:    $\pi' := \pi$ 
6:    $\pi'_e := \pi_e$ 
7:   move a random job from  $\pi$  to the
   start of  $\pi_e$ ;
8:    $j := 1$ ;
9:   while  $j \leq n - i$  do
10:    if  $\text{random} > 0.5$  then
11:      move  $j$ -th job from  $\pi'$  to the
      start of  $\pi'_e$ ;
12:      if  $\sum C_{\max}(\pi') < \sum C_{\max}(\pi)$ 
      then
13:         $\pi := \pi'$ 
14:         $\pi_e := \pi'_e$ 
15:      end if
16:      move the first job of  $\pi'_e$  back
      to the  $j$ -th position on  $\pi'$ ;
17:    end if
18:     $j := j + 1$ ;
19:  end while
20:   $i := i + 1$ ;
21: end while
22: return  $\pi, \pi_e$ ;

```

Algorithm 13 RandomRepair

```

1: Initialize:  $\pi, \pi_e$ ;
2: while  $\pi_e \neq \text{empty}$  do
3:   move the first job of  $\pi_e$  to a random
   position of  $\pi$ ;
4: end while
5: if  $\sum C_j(\pi) > UB$  then
6:    $\pi := \text{empty}$ 
7: end if
8: return  $\pi$ ;

```

Algorithm 14 GreedyRepair

```

1: Initialize:  $\pi, \pi_e$ ;
2: while  $\pi_e \neq \text{empty}$  do
3:    $\pi_d :=$  best feasible sequence (if any)
   build by testing the last job of  $\pi_e$ 
   in all positions of  $\pi_d$ ;
4:   if no feasible sequence was build
   then
5:      $\pi := \text{empty}$ 
6:     break; # While ends
7:   end if
8: end while
9: return  $\pi$ ;

```

Algorithm 15 GreedyRandomRepair

```

1: Initialize:  $\pi, \pi_e$ ;
2:  $\pi' := \text{empty}$ 
3: while  $\pi_e \neq \text{empty}$  do
4:   move the first job of  $\pi'_e$  to the be-
   ginning of  $\pi$ ;
5:    $i := 1$ ;
6:   while  $i \leq n$  do
7:     if  $\sum C_j(\pi) \leq UB$  then
8:       if  $\pi' = \text{empty}$  then
9:          $\pi' := \pi_d$ 
10:      else if  $\text{random} > 0.5$  and
       $C_{\max}(\pi_d) < C_{\max}(\pi')$  then
11:         $\pi' := \pi_d$ 
12:      end if
13:    end if
14:    move the  $i$ -th job of  $\pi$  to position
     $i + 1$ ;
15:     $i := i + 1$ ;
16:  end while
17:   $\pi := \text{empty}$ 
18:  if  $\pi' = \text{empty}$  then
19:    break; # While ends
20:  else
21:     $\pi := \pi'$ 
22:     $\pi' := \text{empty}$ 
23:  end if
24: end while
25: return  $\pi$ ;

```

6.2 Computational experiments

All algorithms were implemented in C++. The PC used was an AMD Quad-Core Processor A12-9720P 3.60 GHz and 8 GB RAM. This study considers the problem instances of Ruiz and Stützle (2008), which are extensions of the Taillard's benchmark Taillard (1993). The extensions are divided in four sets, each consisting of 10 problems for each combination $n \times m$ for $\{20, 50, 100\} \times \{5, 10, 20\}$ and $200 \times \{10, 20\}$. The processing times have a uniform distribution in the range $[1, 99]$. Each set has setup times uniformly distributed in the ranges $[1, 9]$, $[1, 49]$, $[1, 99]$ and $[1, 124]$, respectively. This means that in total, there are 440 different problem instances. The stopping criterion is based on an interval CPU time given by $n \times (m/2) \times t$ *milliseconds*, with $t \in \{10, 20, 30, 40\}$ in order to analyze consistency across different computational times. The initial solutions were generated by using a pairwise exchange local search with neighborhood space of size $(n - 1)$. The upper bounds K were defined as the makespan values of the initial solutions. The same initial solutions were used for all methods. The solutions were evaluated by using the Average Relative Percentage Deviation (*ARPD*) defined as

$$ARPD = \frac{100}{N} \sum_{i=1}^N \frac{(C_{max})_i^h - (C_{max})_i^{best}}{(C_{max})_i^h},$$

Table 9 – *ARPD* results of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max} | \sum C_j)$

(a) jobs [n] × machines [m]						(b) setup [s] × time [t]					
Heuristic		<i>ALNS_A</i>	<i>GL</i>	<i>HH1</i>	<i>TOB</i>	Heuristic		<i>ALNS_A</i>	<i>GL</i>	<i>HH1</i>	<i>TOB</i>
n	m					s	t				
20	5	0.22	0.62	2.55	2.28	10	10	0.57	1.71	3.33	10.39
	10	0.13	0.19	2.03	1.88		20	0.37	1.51	3.04	5.67
50	20	0.05	0.04	1.60	1.31		30	0.24	1.42	2.80	1.69
	5	0.62	2.60	3.04	2.15		40	0.27	1.40	2.84	1.67
	10	0.41	1.81	2.38	1.88	50	10	0.61	1.66	3.27	8.95
	20	0.38	1.41	2.25	2.06		20	0.39	1.49	3.04	5.01
100	5	0.69	3.12	2.06	2.04		30	0.28	1.39	2.94	1.52
	10	0.46	2.35	2.27	1.64		40	0.21	1.34	2.86	1.50
	20	0.34	1.80	3.30	1.32	100	10	0.70	2.00	3.78	8.21
	5	0.53	2.45	4.69	20.19		20	0.37	1.77	3.34	4.94
20	0.37	2.08	5.14	11.60	30		0.30	1.71	3.14	1.81	
40	0.21	1.65	3.12	1.79	125		10	0.67	2.22	3.74	8.08
Average		0.38	1.68	3.21		4.40	20	0.44	2.01	3.49	5.04
							30	0.29	1.86	2.20	2.05
							40	0.18	1.73	3.42	2.02
Average						Average		0.38	1.68	3.21	4.40

where *ARPD* represents the performance of a heuristic h . In summary, this measure represents the arithmetic mean of the deviations from the best solutions found. Therefore,

the best heuristic is the one with the lowest *ARPD* value.

Figure 12 – *ARPD* trends of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max}|\sum C_j)$

Source: Prepared by the author.

The *ARPD* results over jobs and machines are presented in Table 9a. Table 9b gives the results over setup and time factors. Note that $ALNS_A$ performs consistently better when compared to other methods. The *ARPD* values of TOB are disproportionately high for 200 jobs because GPU times generated with $t \in \{10, 20\}$ are not enough for its constructive mechanisms to finish. However, it gets better when more computational time is available. These and other trends can be seen in Figure 12. The overall *ARPD* values of $ALNS_A$, GL , $HH1$, TOB are 0.028%, 1.322%, 2.322% and 4.019%, respectively.

The Tukey' honestly significant difference (HSD) test was conducted to analyze the statistic difference between the means. The null hypotheses that two algorithms have equal performances was tested at a significance level of 5%. The results in Table 10 show that all algorithms are statistically different from each other. The differences at a confidence interval of 95% are illustrated in Figure 13. As can be noted, $ALNS_A$ is the best performing algorithm, yielding the lowest *ARPD* value without overlapping intervals.

Table 10 – Tukey' HSD test of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max}|\sum C_j)$

Methods		Mean Difference (I-J)	Std. Error	Sig.	95% confidence interval		Reject
I	J				Lower Bound	Upper Bound	
<i>ALNS_A</i>	GL	-1.30	.166	.000	-1.73	-0.87	True
	HH1	-2.83	.166	.000	-3.25	-2.40	True
	TOB	-4.02	.166	.000	-4.44	-3.59	True
<i>GL</i>	<i>ALNS_A</i>	1.30	.166	.000	0.87	1.73	True
	HH1	-1.53	.166	.000	-1.95	-1.10	True
	TOB	-2.72	.166	.000	-3.14	-2.29	True
<i>HH1</i>	<i>ALNS_A</i>	2.83	.166	.000	2.40	3.25	True
	GL	1.53	.166	.000	1.10	1.96	True
	TOB	-1.19	.166	.000	-1.61	-0.76	True
<i>TOB</i>	<i>ALNS_A</i>	4.01	.166	.000	3.59	4.44	True
	GL	2.72	.166	.000	2.29	3.14	True
	HH1	1.19	.166	.000	0.76	1.61	True

FWER = 0.05

Figure 13 – Multiple Comparisons of algorithms for $F_m/nwt, s_{jk}^i/\epsilon(C_{max}|\sum C_j)$

Source: Prepared by the author.

6.3 Conclusion

In this chapter, the scheduling problem $Fm/nwt, s_{k,j}/\epsilon(C_{max}/\sum C_j)$ was addressed. We propose the algorithm called $ALNS_A$ to solve the problem. Its performance was compared with the existent algorithms GL , $HH1$ and TOB , which are designed to solve the most similar problems found in the literature. Experiments under the same computational conditions show that $ALNS_A$, GL , $HH1$ and TOB obtained overall $ARPD$ values of 0.38%, 1.68%, 3.21% and 4.40%, respectively. Therefore, $ALNS_A$ outperforms the other methods.

Despite the superior performance of $ALNS_A$, there is still room for improvement. For example, regarding the heuristic evaluation process, it can be appropriate to assign weights to pairs of methods instead of each method individually because their performances may not be the same across the different combinations. Another option is to calibrate more parameters. Here, the percentage of jobs removed by destruction heuristics and the random factors were fixed to 50%. The parameters to converge the search T_i and c_f also have not been specifically calibrated for this problem. Therefore, it could be possible to increase performance with additional tuning. Algorithms for different applications can also be adapted and included in future experiments in order to get additional insights.

7 CONCLUSION

In this Thesis, the no-wait flow shop scheduling problem with sequence-dependent setup times was addressed. The main objective was to provide a deep understanding about this important problem and provide efficient methods to solve it. In order to achieve this goal, specific objectives were defined, which have been addressed along all chapters as follows:

OBJ1 - To provide a precise description of the NWT-FSP-SDST

The problem was formally stated in Chapter 2. First, a notation and a classification system were presented in Section 2.4. Then, Sections 2.1 , 2.2 and 2.3 gave a theoretical introduction about flow shop, no-wait and sequence-dependent setup times, respectively. Finally, a mathematical model was presented in Section 2.5 to precisely describe the problem.

OBJ2 - To provide a literature review about the NWT-FSP with SDST and $\epsilon(PM_1|PM_2)$

As the problem $F_m/nwt, s_{jk}^i/\epsilon(C_{max}|\sum C_j)$ is new, the literature review, presented in Chapter 3, was done in parts. First, the NWT-FSP-SDST with mono-objective for makespan, total completion time and total tardiness was considered. Then, the NWT-FSP with objective of type $\epsilon(PM_1|PM_2)$ was covered.

OBJ3 To provide efficient methods to solve the NWT-FSP-SDST

New heuristic algorithms were presented in order to solve the NWT-FSP-SDST for the objective functions $\epsilon(\sum T_j|C_{max})$, $\epsilon(\sum C_j|C_{max})$ and $\epsilon(C_{max}|\sum C_j)$. All methods were developed specifically for their functions, but they can be easily adapted to other problems. The methods are summarized as follows:

- A greedy algorithm is proposed in Section 4.1.2 for function $\epsilon(\sum T_j|C_{max})$. The algorithm is capable of adjusting its destruction intensity according to the instance size and the number of iterations.
- Four greedy algorithms are proposed in Section 5.1.2 for function $\epsilon(\sum C_j|C_{max})$. The algorithms combine distinct destruction and construction mechanisms, where the search intensification-diversification as well as the greedy-random behavior are proposed at different levels.
- An adaptive large neighborhood search algorithm called $ALNS_A$ is presented in Section 6.1.2 for function $\epsilon(C_{max}|\sum C_j)$. The algorithm accesses a set of distinct

mechanisms and dynamically select a pair of destroy and repair methods based on their performance history.

OBJ4 To demonstrate the efficiency of the proposed methods through computational experiments

For each proposed method, computational experiments based on the well-known benchmarks and statistical performance comparisons with the state-of-the-art algorithms adapted from related scheduling problems were provided. The following results were presented:

- For function $\epsilon(\sum T_j | C_{max})$ (Section 4.2), the new method, called IG_A , was compared to the literature algorithms AA and PA . Experiments confirmed the superiority of the proposed approach with the $ARPD$ values of 1.45%, 36.29%, and 52.41% for IG_A , PA and AA , respectively.
- For function $\epsilon(\sum C_j | C_{max})$ (Section 5.2), the best method proposed called IG_4 was compared to the algorithms $ISA-2$, TOB and PAL . Experiments confirmed the superiority of the proposed approach with the $ARPD$ values of 0.04%, 1.73%, 4.10%, and 10.60% for IG_4 , $ISA-2$, TOB and PAL , respectively.
- For function $\epsilon(C_{max} | \sum C_j)$ (Section 6.2), the proposed algorithm called $ALNS_A$ was compared to the literature algorithms GL , $HH1$ and TOB . Experiments confirmed the superiority of the new approach with the $ARPD$ values of 0.028%, 1.322%, 2.322% and 4.019% for $ALNS_A$, GL , $HH1$, respectively.

7.1 Future research directions

Despite the superior performance of the new methods, there is still room for improvement. For example, the proposed algorithms were compared with the best literature methods for similar problems, but adapting methods from different applications could help to get additional insights. Another option is to improve the destruction mechanisms by considering additional parameters to determine the destruction strength of the algorithms. In this work, only the number of jobs was considered for this purpose, and therefore it may be possible to get better results with a broader approach.

The proposed algorithms can also be easily adapted to other manufacturing environments, specially when addressed as permutation scheduling problems. Flow shop layouts under the constraints no-idle, mixed no-idle or mixed no-wait are some examples. Another extension is to consider maintenance. In this work, this constraint was ignored or assumed to be included in the processing times. However, it may be more appropriate to treat it separately for some cases. Other objective functions with different performance measures can also be included with small modifications in the algorithm.

REFERENCES

- ALDOWAISAN, T.; ALLAHVERDI, A. Minimizing total tardiness in no-wait flowshops. **Foundations of Computing and Decision Sciences**, Sciendo, v. 37, n. 3, p. 149–162, 2012.
- ALDOWAISAN, T.; ALLAHVERDI, A. Total tardiness performance in m-machine no-wait flowshops with separate setup times. **Intelligent Control and Automation**, v. 6, p. 38–44, 2015.
- ALLAHVERDI, A. A new heuristic for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. **Computers & Operations Research**, Elsevier, v. 31, n. 2, p. 157–180, 2004.
- ALLAHVERDI, A. A survey of scheduling problems with no-wait in process. **European Journal of Operational Research**, Elsevier, v. 255, n. 3, p. 665–686, 2016.
- ALLAHVERDI, A.; AYDILEK, H. Algorithms for no-wait flowshops with total completion time subject to makespan. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 68, n. 9-12, p. 2237–2251, 2013.
- ALLAHVERDI, A.; AYDILEK, H. Total completion time with makespan constraint in no-wait flowshops with setup times. **European Journal of Operational Research**, Elsevier, v. 238, n. 3, p. 724–734, 2014.
- ALLAHVERDI, A.; AYDILEK, H.; AYDILEK, A. No-wait flowshop scheduling problem with two criteria; total tardiness and makespan. **European Journal of Operational Research**, Elsevier, v. 269, n. 2, p. 590–601, 2018.
- ALLAHVERDI, A.; AYDILEK, H.; AYDILEK, A. No-wait flowshop scheduling problem with separate setup times to minimize total tardiness subject to makespan. **Applied Mathematics and Computation**, Elsevier, v. 365, p. 124688, 2020.
- ARAÚJO, D. C.; NAGANO, M. S. A new effective heuristic method for the no-wait flowshop with sequence-dependent setup times problem. **International Journal of Industrial Engineering Computations**, v. 2, n. 1, p. 155–166, 2011.
- AYDILEK, H.; ALLAHVERDI, A. Heuristics for no-wait flowshops with makespan subject to mean completion time. **Applied Mathematics and Computation**, Elsevier, v. 219, n. 1, p. 351–359, 2012.
- AZI, N.; GENDREAU, M.; POTVIN, J.-Y. An adaptive large neighborhood search for a vehicle routing problem with multiple routes. **Computers & Operations Research**, Elsevier, v. 41, p. 167–173, 2014.
- BAKER, K. R.; TRIETSCH, D. **Principles of sequencing and scheduling**. [*S.l.*: *s.n.*]: John Wiley & Sons, 2019.
- BEEZÃO, A. C. *et al.* Scheduling identical parallel machines with tooling constraints. **European journal of operational research**, Elsevier, v. 257, n. 3, p. 834–844, 2017.

BERTOLISSI, E. Heuristic algorithm for scheduling in the no-wait flow-shop. **Journal of Materials Processing Technology**, Elsevier, v. 107, n. 1-3, p. 459–465, 2000.

BIANCO, L.; DELL'OLMO, P.; GIORDANI, S. Flow shop no-wait scheduling with sequence dependent setup times and release dates. **INFOR: Information Systems and Operational Research**, Taylor & Francis, v. 37, n. 1, p. 3–19, 1999.

CIAVOTTA, M.; MINELLA, G.; RUIZ, R. Multi-objective sequence dependent setup times permutation flowshop: A new algorithm and a comprehensive study. **European Journal of Operational Research**, v. 227, n. 2, p. 301–313, 2013. ISSN 0377-2217.

DEMIR, E.; BEKTAŞ, T.; LAPORTE, G. An adaptive large neighborhood search heuristic for the pollution-routing problem. **European Journal of Operational Research**, Elsevier, v. 223, n. 2, p. 346–359, 2012.

DING, J. Y. *et al.* An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. **Applied Soft Computing**, v. 30, p. 604–613, 2015. ISSN 1568-4946.

DUBOIS-LACOSTE, J.; LOPEZ-IBANEZ, M.; STUTZLE, T. A hybrid tp plus pls algorithm for bi-objective flow-shop scheduling problems. **Computers & Operations Research**, v. 38, n. 8, p. 1219–1236, 2011. ISSN 0305-0548.

EMMONS, H.; VAIRAKTARAKIS, G. **Flow Shop Scheduling. Theoretical Results, Algorithms, and Applications**. [*S.l.: s.n.*]: Springer Science & Business Media, 2013.

FERNANDEZ-VIAGAS, V.; FRAMINAN, J. M. Neh-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. **Computers & Operations Research**, Elsevier, v. 60, p. 27–36, 2015.

FRAMINAN, J. M.; LEISTEN, R. A heuristic for scheduling a permutation flowshop with makespan objective subject to maximum tardiness. **International Journal of Production Economics**, Elsevier, v. 99, n. 1-2, p. 28–40, 2006.

FRAMINAN, J. M.; PEREZ-GONZALEZ, P. Order scheduling with tardiness objective: Improved approximate solutions. **European Journal of Operational Research**, Elsevier, v. 266, n. 3, p. 840–850, 2018.

FRANCA, P. M.; JR, G. T.; BURIOL, L. Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions. **International Journal of Production Research**, Taylor & Francis, v. 44, n. 5, p. 939–957, 2006.

HEMMELMAYR, V. C.; CORDEAU, J.-F.; CRAINIC, T. G. An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. **Computers & operations research**, Elsevier, v. 39, n. 12, p. 3215–3228, 2012.

KHALILI, M.; NADERI, B. A bi-objective imperialist competitive algorithm for no-wait flexible flow lines with sequence dependent setup times. **The International Journal of Advanced Manufacturing Technology**, Springer, v. 76, n. 1-4, p. 461–469, 2015.

- LEE, Y. H.; JUNG, J. W. New heuristics for no-wait flowshop scheduling with precedence constraints and sequence dependent setup time. *In: SPRINGER. International Conference on Computational Science and Its Applications*. [S.l.: s.n.], 2005. p. 467–476.
- LI, X. *et al.* An iterated greedy heuristic for no-wait flow shops with sequence dependent setup times, learning and forgetting effects. **Information Sciences**, Elsevier, v. 453, p. 408–425, 2018.
- LIN, S.-W.; YING, K.-C. Minimizing shifts for personnel task scheduling problems: A three-phase algorithm. **European Journal of Operational Research**, Elsevier, v. 237, n. 1, p. 323–334, 2014.
- LOPEZ-IBANEZ, M. *et al.* The irace package: Iterated racing for automatic algorithm configuration. **Operations Research Perspectives**, v. 3, p. 43–58, 2016. ISSN 2214-7160.
- MINELLA, G.; RUIZ, R.; CIAVOTTA, M. A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. **INFORMS Journal on Computing**, INFORMS, v. 20, n. 3, p. 451–471, 2008.
- MINELLA, G.; RUIZ, R.; CIAVOTTA, M. Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. **Computers & Operations Research**, v. 38, n. 11, p. 1521–1533, 2011. ISSN 0305-0548.
- MIYATA, H. H.; NAGANO, M. S.; GUPTA, J. N. Integrating preventive maintenance activities to the no-wait flow shop scheduling problem with dependent-sequence setup times and makespan minimization. **Computers & Industrial Engineering**, Elsevier, v. 135, p. 79–104, 2019.
- NAGANO, M. S.; ALMEIDA, F. S. d.; MIYATA, H. H. An iterated greedy algorithm for the no-wait flowshop scheduling problem to minimize makespan subject to total completion time. **Engineering Optimization**, Taylor & Francis, p. 1–19, 2020.
- NAGANO, M. S.; ARAÚJO, D. C. New heuristics for the no-wait flowshop with sequence-dependent setup times problem. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, Springer, v. 36, n. 1, p. 139–151, 2014.
- NAGANO, M. S.; MIYATA, H. H.; ARAÚJO, D. C. A constructive heuristic for total flowtime minimization in a no-wait flowshop with sequence-dependent setup times. **Journal of Manufacturing Systems**, Elsevier, v. 36, p. 224–230, 2015.
- NAWAZ, M.; ENSCORE, E. E. J.; HAM, I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. **Omega**, Elsevier, v. 11, n. 1, p. 91–95, 1983.
- PAN, Q. K.; RUIZ, R. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. **Omega-International Journal of Management Science**, v. 44, p. 41–50, 2014. ISSN 0305-0483.
- PINEDO, M. **Scheduling. Theory, Algorithms, and Systems**. 5. ed. [S.l.: s.n.]: Springer Science & Business Media, 2016.
- PISINGER, D.; ROPKE, S. Large neighborhood search. *In: Handbook of metaheuristics*. [S.l.: s.n.]: Springer, 2019. p. 99–127.

QIAN, B. *et al.* A differential evolution algorithm with two speed-up methods for nfssp with sdsts and rds. *In: IEEE. Proceedings of the 10th World Congress on Intelligent Control and Automation.* [S.l.: s.n.], 2012. p. 490–495.

QIAN, B. *et al.* Hybrid differential evolution optimization for no-wait flow-shop scheduling with sequence-dependent setup times and release dates. *In: SPRINGER. International Conference on Intelligent Computing.* [S.l.: s.n.], 2011. p. 600–611.

QU, Y.; BARD, J. F. A grasp with adaptive large neighborhood search for pickup and delivery problems with transshipment. **Computers & Operations Research**, Elsevier, v. 39, n. 10, p. 2439–2456, 2012.

RIFAI, A. P.; NGUYEN, H.-T.; DAWAL, S. Z. M. Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. **Applied Soft Computing**, Elsevier, v. 40, p. 42–57, 2016.

ROPKE, S.; PISINGER, D. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. **Transportation science**, Informs, v. 40, n. 4, p. 455–472, 2006.

RUIZ, R.; STÜTZLE, T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. **European Journal of Operational Research**, Elsevier, v. 177, n. 3, p. 2033–2049, 2007.

RUIZ, R.; STÜTZLE, T. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. **European Journal of Operational Research**, Elsevier, v. 187, n. 3, p. 1143–1159, 2008.

SAMARGHANDI, H. A no-wait flow shop system with sequence dependent setup times and server constraints. **IFAC-PapersOnLine**, Elsevier, v. 48, n. 3, p. 1604–1609, 2015.

SAMARGHANDI, H. Studying the effect of server side-constraints on the makespan of the no-wait flow-shop problem with sequence-dependent set-up times. **International Journal of Production Research**, Taylor & Francis, v. 53, n. 9, p. 2652–2673, 2015.

SAMARGHANDI, H.; ELMEKKAWY, T. Y. Solving the no-wait flow-shop problem with sequence-dependent set-up times. **International Journal of Computer Integrated Manufacturing**, Taylor & Francis, v. 27, n. 3, p. 213–228, 2014.

SHAW, P. Using constraint programming and local search methods to solve vehicle routing problems. *In: SPRINGER. International conference on principles and practice of constraint programming.* [S.l.: s.n.], 1998. p. 417–431.

TAILLARD, E. Benchmarks for basic scheduling problems. **European Journal of Operational Research**, v. 64, n. 2, p. 278–285, 1993. ISSN 0377-2217.

TASGETIREN, M. F. *et al.* Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. **Computers & Operations Research**, v. 77, p. 111–126, 2017. ISSN 0305-0548.

T'KINDT, V.; BILLAUT, J.-C. **Multicriteria scheduling: theory, models and algorithms.** [S.l.: s.n.]: Springer Science & Business Media, 2006.

T'KINDT, V.; BILLAUT, J.-C. An approach to multicriteria scheduling problems. **Multicriteria Scheduling: Theory, Models and Algorithms**, Springer, p. 113–134, 2006.

XU, T.; ZHU, X.; LI, X. Efficient iterated greedy algorithm to minimize makespan for the no-wait flowshop with sequence dependent setup times. *In: IEEE. **Proceedings of the 2012 IEEE 16th International Conference on Computer Supported Cooperative Work in Design (CSCWD)**. [S.l.: s.n.], 2012. p. 780–785.*

YE, H.; LI, W.; NAULT, B. R. Trade-off balancing between maximum and total completion times for no-wait flow shop production. **International Journal of Production Research**, Taylor & Francis, v. 58, n. 11, p. 3235–3251, 2020.

ZHU, X.; LI, X.; GUPTA, J. N. Iterative algorithms for no-wait flowshop problems with sequence-dependent setup times. *In: IEEE. **2013 25th Chinese Control and Decision Conference (CCDC)**. [S.l.: s.n.], 2013. p. 1252–1257.*

ZHU, X.; LI, X.; WANG, Q. An adaptive intelligent method for manufacturing process optimization in steelworks. *In: IEEE. **Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)**. [S.l.: s.n.], 2013. p. 363–368.*

ZHUANG, W. J.; XU, T.; SUN, M. Y. A hybrid iterated greedy algorithm for no-wait flowshop with sequence dependent setup times to minimize makespan. *In: TRANS TECH PUBL. **Advanced Materials Research**. [S.l.: s.n.], 2014. v. 933, p. 459–466.*