Just-in-Time Scheduling in Multiple Parallel Machines

Daniela Amado Eloi

Instituto Superior Técnico
danielaamadoeloi@tecnico.ulisboa.pt

Producing the ordered products in the right quantity and at the right time is the main purpose of a Just-in-Time (JIT) production system. If, on the one hand, delays in the delivery of the orders can carry high losses and jeopardise the reputation of the company, on the other hand, producing too early will translate into unnecessary inventory costs. The Unrelated Parallel Machines with Job Sequence-Dependent Setup Times, Job Families and Precedence Constraints (UPM-JSDST-JF-PC) scheduling problem is the problem under study. The problem is addressed with an exact model, unable to provide optimal solutions in reasonable times for most of the generated instances. Then, two heuristic methods are proposed to address the problem under study: Multi-Start Iterated Local Search - Randomised Variable Neighbourhood Descent (MS ILS-RVND) heuristic and Adaptive Large Neighbourhood Descent (ALNS-RVND) heuristic. Despite not guaranteeing the optimality of the solutions, both heuristics provide higher quality solutions in shorter computation times than the exact model, for most of the generated instances. A performance comparison between the heuristic methods is also established.

***Index Terms*—Just-in-Time; Scheduling; Earliness and tardiness; Parallel machines; Optimisation; Heuristics.**

I. Introduction

Scheduling is a decision-making process that is used in the industries, dealing with the allocation of resources to jobs along time. Hence, analysing a scheduling problem and developing a procedure for dealing with it is, in the real world, a great challenge [26]. These problems started being studied in the 1950s, aiming to model complex real problems and optimise economic indicators. Over the years, scheduling problems with different processing characteristics, constraints and production environments were studied. Despite the great interest into investigating these problems, many real problems remain unstudied, which have different characteristics from the problems already addressed.

Aiming to optimise resources and fulfil customer orders, production scheduling is a crucial process on any company, developed at operational level. Production scheduling is the process that aims to minimise the production time and costs, by choosing when to produce, with which workforce, and on which machinery.

Scheduling problems have a proper notation to be defined proposed by [26]. The majority of authors in the existing literature adopt this notation. Let n and m denote, respectively, the number of jobs and the number of machines. The set of jobs is denoted by N, being $N = \{1, 2, ..., n\}$ and, the set of machines is denoted by M, being $M = \{1, 2, ..., m\}$. A job is denoted by j, where $j \in N$ and, a machine is denoted by i, where $i \in M$. A scheduling problem is described by a triplet $\alpha|\beta|\gamma$, where:

- α describes the machine environment and contains only a single entry;
- β describes processing characteristics and constraints, and can contain no entries, a single entry or multiple entries;
- γ describes the objective to be minimised and usually contains a single entry.

There are three types of parallel machine environment, which are distinguished depending on the job processing time: identical parallel machine environment (P_m), uniform parallel machine environment (Q_m) and unrelated parallel machine environment (R_m). The abbreviations between parenthesis are the entries of the α of the triplet $\alpha|\beta|\gamma$. Let p_{ij} denote the processing time of job j on machine i. In an identical parallel machine environment, the processing time of the job j does not depend on the machine i, that is, $p_{ij} = p_j$. In an uniform parallel machine environment, the processing time of the job j depends on the processing speed of the machine i, that is, $p_{ij} = p_j/v_i$ for all i and j, where v_i is the speed of machine i, and p_j is the processing time per unit speed. In an unrelated parallel machine environment, the processing time of the jobs is arbitrary.

Scheduling problems can have different job characteristics that include precedence constraints ($prec$), release dates (r_j), sequence-dependent setup times (s_{jk}), machine eligibility constraints (M_j), preemption ($prmp$) and batch processing ($batch$). The abbreviations between parenthesis are the entries of the β of the triplet $\alpha|\beta|\gamma$.

The objective function to be optimised is usually known as a performance indicator and it is always a function of the completion times of the jobs, which depends on the schedule. Some examples of minimisation objectives, which are the most approached ones in the literature, are: the makespan, the lateness, the tardiness, the earliness and the number of tardy jobs. The entry in the γ field of the triplet $\alpha|\beta|\gamma$ is the mathematical expression that defines the objective function.

Just-in-Time (JIT) philosophy emerged in the 1970s, and in the last decades has been adopted by many companies worldwide. Producing the orders in the proper quantity at the right time is the main purpose of a JIT production system. The growing interest in this philosophy has motivated the study of scheduling models. The JIT scheduling is concerned with the earliness and tardiness of job completion. On the one hand, delayed delivery negatively influences customer satisfaction

and company reputation and, on the other hand, early job completion can increase inventory costs. The JIT scheduling problems can be separated according to the approach of their due dates and the classification of the objective function [33], as illustrated in Figure 1.

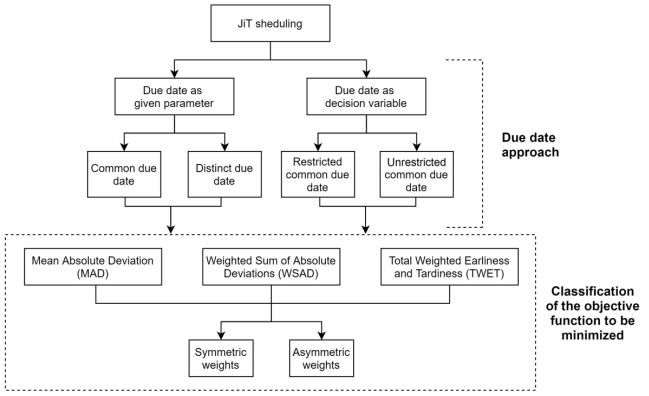

Fig. 1: Framework of JIT scheduling problems, adapted from [31]

There are two different due date approaches: due date as a given parameter or due date as a decision variable. Firstly, when the due dates result from a choice made by the decision-maker and constitute the input data, it is considered as a given parameter. The due dates can be common for all jobs, denoted by d, or distinct for all jobs, denoted by d_j. Lastly, an interesting aspect of JIT scheduling is the possibility of due date negotiation [14]. Sometimes the decision-maker and his customer negotiates about the due date of a job and, in this case, the decision-maker needs to set up an algorithm which returns a schedule and a preferable due date [33]. When this happens, the due date is considered as a decision variable and is common for all jobs. The jobs may are subject to a unrestricted common due date or to a restricted common due date, that is, the due date must be lower than the sum of the processing times of all jobs.

The JIT scheduling problems have proper objective functions to be minimised, according to [31]: Mean Absolute Deviation (MAD), Weighted Sum of Absolute Deviation (WSAD), and Total Weighted Earliness and Tardiness (TWET). Let be, respectively, α_j and β_j the earliness and the tardiness of the job j. The goal of a MAD problem is to find a schedule considering that the earliness and tardiness of the jobs are both equally penalised and is defined as $\frac{1}{n}\sum_{j\in N}(E_j+T_j)$. WSAD objective allows a different weight for earliness, α, and tardiness, β, which is modelled by $\sum_{j\in N}(\alpha E_j+\beta T_j)$. In a TWET problem, each job may have different earliness weight, α_j and tardiness weight, β_j and is defined by $\sum_{j\in N}(\alpha_j E_j+\beta_j T_j)$. The weights may be symmetric, $\alpha_j = \beta_j$ and $\alpha = \beta$, or asymmetric, $\alpha_j \neq \beta_j$ and $\alpha \neq \beta$, in WSAD and TWET problem, respectively.

Scheduling problems with earliness and tardiness penalties are commonly encountered in today's manufacturing environment due to the current emphasis on the JIT philosophy [29].

II. Literature Review

The reviewed publications about the JIT scheduling considering due date as a given parameter. Table I summarises papers reviewed that consider due date as given parameter. For each reviewed publication, it summarises the machine environment, the processing characteristics and constraints, the objective function, the due date approach and the solution procedures. Some authors considers the due date common for all jobs and others authors considers it distinct for all jobs.

A. Common due date for all jobs

[9] address a TWET problem in identical parallel machine environment with a common due date for all jobs. Genetic Algorithm (GA) is applied to solve this problem. [5] consider a TWET problem too, but in an unrelated parallel machine environment. A release date is given for each job, and a common due date is given for all jobs. They suggested different constructive algorithms for an heuristic solution of the problem, which are composed of two-stage procedures. The first phase assigns the jobs to machines, and then, for each machine and the corresponding set of jobs, a single machine problem is solved. The authors also proposed some iterative algorithms, which they use for performance comparison [31].

B. Distinct due date for all jobs

Considering an identical parallel machine environment, [20] address JIT scheduling problem: minimising the TWET with deadlines. As opposed to due dates, which may be violated at the weight of tardiness, deadlines must be met and cannot be violated. The authors present a search heuristic, combining elements of the solution methods known as Greedy Randomised Adaptive Search Procedure (GRASP) and tabu search.

[13] propose a heuristic procedure to deal with the WSAD scheduling on unrelated parallel machines, subject to job sequence-dependent setup times and machine eligibility restrictions. Their heuristic solution procedure involved two stages: at a first stage, a single machine sequencing heuristic is developed, and at the second stage, this single machine heuristic is built into a machine assignment heuristic.

[32] address the TWET problem. A set of independent jobs with sequence-dependent setups is given to be scheduled on a set of uniform parallel machines and each job has its own release date. The authors employ two GAs to deal with this problem: one with a crossover operator which is developed to solve multi-component combinatorial optimisation problems, and the other with no crossover operator. The results on 960 randomly generated instances indicate that GAs are an efficient algorithm. [4] consider the same problem as [32] and present a Mixed Integer Programming (MIP) model. [4] perform computational experiences using this model to solve small sized problems. To solve problems of larger size, the authors suggest a solution approach based on Bender's decomposition, that is a primal-dual-based procedure which brackets the optimum solution by iterating between a primal master problem and a dual subproblem. They realised that it is time consuming and using other solution approaches, such as heuristic approaches, to solve the "master" problem may accelerate the process allowing more rapid solution of larger problems.

Considering an identical parallel machine environment, [29] address the WSAD problem when $\alpha = \beta = 1$. Job sequence-dependent set-up times are taken into account. They provide the first step towards obtaining near-optimal solutions for this problem using LS heuristics in the framework of a meta-heuristic technique known as Simulated Annealing (SA).

[10] consider a set of jobs to be processed on a set of unrelated parallel machines subject to precedence constraints, where the objective is to minimise the WSAD. The authors develop an exact exponential-time algorithm. [28] extend the work performed by [10]. [28] model it as a MIP formulation, and solved using MAS^H, a deterministic heuristic based on Multi-Agent Systems (MAS).

[35] propose a MIP model for TWET problem in an unrelated parallel machine environment, considering job sequence-dependent setup times. The model can provide the optimal schedule to problems involving nine jobs and three or fewer machines in a reasonable computation time. The authors realised that the size of most industrial problems exceeds the capability of the proposed model and that could be beneficial if heuristic solution procedures were developed.

[2], [34], [24] and [15] consider the same problem as [35] but all of them propose different solution approaches. [2] explore the use of Artificial Neural Network (ANN). [34] propose a GA to address the problem. [24] propose three different heuristics: the first is a simple GRASP heuristic, the second heuristic includes an intensification procedure based on Path Relinking technique, and the third uses an Iterated Local Search (ILS) heuristic in the LS phase of the GRASP algorithm. [15] propose an exact solution approach: a MIP is proposed to formulate the problem and is solved optimally in small size instances. The authors employ Parallel Net Benefit Compression-Net Benefit Expansion (PNBC-NBE) heuristic, two meta-heuristics and a hybrid technique to solve medium-to-large sized instances.

Considering an identical parallel machine environment, [25] address the TWET problem with job families and product batches. A sequence-dependent setup time is incurred, when the production changes over from a job of one family to a job of another family. The authors develop a MIP formulation, which can provide optimal solutions for instances with up to 18 jobs and 4 families. They realised that an opportunity for further research is the use of heuristic methods for solving large real-life instances with hundreds of jobs.

[30] consider the TWET scheduling problem in an uniform parallel machine environment earliness–tardiness with job sequence-dependent set-up times. The authors propose a SA-fuzzy logic approach. The SA algorithm identifies the best sequences for the different weighted combinations of earliness and tardiness measures for each given set of jobs. The fuzzy logic is then used to select the optimal weighted combination, which satisfies the combined objective function to a larger extent.

[11] address the TWET scheduling problem on identical parallel machines with job sequence-dependent setup times. The authors apply a Squeaky Wheel Optimisation (SWO) framework to solve the problem.

In an identical parallel machine environment also, [17] address a TWET scheduling of a set of jobs with distinct release date for all jobs. Precedence constraints are taken into account on their problem. The authors propose new lower bounds, classified into two families: first, two assignment-based lower bounds for the single machine problem are generalised for the parallel machine case, and second, a time-indexed formulation of the problem is developed to derive an efficient lower bounds through column generation or Lagrangean relaxation. The authors also present a simple LS algorithm to generate an upper bound.

[22] develop a Parallel Machine Moving Block Heuristic (Pm-MBH) to address the WSAD scheduling with unit symmetrical weights, $\alpha = \beta = 1$, in identical parallel machine environment. Precedence constraints are considered. They present a MIP model in order to analyse the quality of the Pm-MBH solutions, and based on the computational results, they consider that Pm-MBH obtains good solutions in a short amount of computation time.

[23] extends the work performed by [17] and propose an MIP formulation to find the exact solution for small instances. To address large instances, the problem is approximately solved via two constructive approaches, three meta-heuristics and several hybrid heuristics.

TABLE I: Summary of the reviewed papers

Paper	Problem	Due date	Solution approach
[20]	P_m | | TWET	distinct	Heuristic
[9]	P_m | | TWET	common	Heuristic
[13]	$R_m \mid s_{jk},\ M_j \mid$ WSAD	distinct	Heuristic
[32]	$Q_m \mid s_{jk},\ r_j \mid$ TWET	distinct	Heuristic
[4]	$Q_m \mid s_{jk},\ r_j \mid$ TWET	distinct	Exact
[29]	$P_m \mid s_{jk} \mid$ WSAD	distinct	Heuristic
[35]	$R_m \mid s_{jk} \mid$ TWET	distinct	Exact
[5]	$R_m \mid r_j \mid$ TWET	common	Heuristic
[10]	$R_m \mid prec \mid$ WSAD	distinct	Exact
[25]	$P_m \mid M_j, fmls, batch \mid$ TWET	distinct	Exact
[17]	$P_m \mid r_j,\ prec \mid$ TWET	distinct	Heuristic
[11]	$P_m \mid s_{jk} \mid$ TWET	distinct	Heuristic
[30]	$Q_m \mid s_{jk} \mid$ TWET	distinct	Heuristic
[2]	$R_m \mid s_{jk} \mid$ TWET	distinct	Heuristic
[22]	$P_m \mid prec \mid$ WSAD	distinct	Heuristic
[23]	$P_m \mid r_j,\ prec \mid$ TWET	distinct	Exact + Heuristic
[34]	$R_m \mid s_{jk} \mid$ TWET	distinct	Exact + Heuristic
[3]	P_m | | TWET	distinct	Heuristic
[15]	$R_m \mid s_{jk},\ M_j \mid$ TWET	distinct	Exact + Heuristic
[28]	$R_m \mid prec \mid$ TWET	distinct	Heuristic
[24]	$R_m \mid s_{jk} \mid$ TWET	distinct	Heuristic
[36]	$R_m \mid r_j,\ prmp \mid$ TWET	distinct	Heuristic
[1]	$P_m \mid prmp \mid$ TWET	distinct	Heuristic
[15]	P_m | | TWET	distinct	Heuristic
[8]	R_m | | WSAD	distinct	Exact + Heuristic

[3] and [16] address the TWET scheduling problem in identical parallel machines. [3] presents a hybrid algorithmic strategy involving GA with a smart LS approach. [16] propose a Mixed Integer Linear Programming (MILP) model for the considered problem. The authors apply a PNBC-NBE heuristic. An Intelligent Water Drop (IWD) algorithm, a new swarm-based nature-inspired optimisation, is also adopted to approach this problem.

[36] and [1] consider TWET scheduling, allowing jobs pre-emption. [1] propose two meta-heuristic algorithms, the Non-Dominated Sorting Genetic Algorithm II (NSGAII) and Non-Dominated Ranking Genetic Algorithm (NRGA) to approach

the problem on identical parallel machines. [36] address the problem on unrelated parallel machines with distinct release date for all jobs. The authors perform a Benders decomposition algorithm to solve the preemptive relaxation formulated as a MIP model, offering a near-optimal assignment of the jobs to the machines.

III. Mathematical Model

This section describes the proposed mathematical model. First, the problem under study is described. Then, the mathematical model is presented.

A. Problem Description

Despite the vast existing and revised literature on JIT scheduling on parallel machines, not all scenarios are studied. There are still many production plants that are not reflected in the existing literature. In order to explore a new scenario of production, the scheduling problem that will be addressed considers a combination of processing characteristics not yet studied in the literature. The Unrelated Parallel Machines with Job Sequence-Dependent Setup Times, Job Families and Precedence Constraints (UPM-JSDST-JF-PC) scheduling problem is the problem to be addressed, which is described by problem (1). This problem is a JIT scheduling to minimise TWET considering job sequence-dependent setup times, job families and precedence constraints. The due date is taken as a given parameter and distinct for all jobs. The UPM-JSDST-JF-PC scheduling problem is not yet studied in the reviewed literature.

$$R_m \mid s_{jk}, fmls, prec \mid \sum_{j \in N} (\alpha_j E_j + \beta_j T_j) \tag{1}$$

The characteristics presented in the UPM-JSDST-JF-PC scheduling problem are similar to the real problems of production firms. The UPM-JSDST-JF-PC scheduling problem can be illustrated by the following industrial scenario from the painting industry: a set of unrelated parallel machines that paint products with different types of ink (primary ink, colour ink and varnish) and different colours. Jobs that consists in painting with similar tones (e.g. dark tones, light tones, etc.) are considered from the same family since there is no need to clean the machine, so, there is no changeover time. The time needed to clean the machine between jobs is sequence-depend since it depends on the type of the ink and colour that need to be cleaned and the type of the ink and colour that need to be prepared. Therefore, this example considers job sequence-dependent setup times. Jobs also have precedences since the varnish only can be applied after the colour ink whereas the colour ink only can be applied after the primary ink.

B. Problem Formulation

The problem under study is to schedule n jobs with sequence-dependent setup times and different due dates on m unrelated parallel machines. The following assumptions are considered for formulating the problem. All jobs are available at time zero, that is, the ready time is zero. The presence of idle times is not allowed and no job preemption is allowed. All machines are available for processing and can process any job. The problem has been formulated for jobs that belong to different families and have precedence constraints between them with probability D.

1) Sets, indexes, parameters and decision variables

Table II presents the sets, the indexes, the parameters and the decision variables to be considered in the mathematical model, which is a MILP model.

TABLE II: Sets, indexes, parameters and decision variables

Sets

N	set of jobs
M	set of machines
F	set of families
F_g	set of jobs belonging to family $g \in F$
$P(j)$	set of predecessors of job j, where $P(j) \subset N$

Indexes

$j, k \in N$	indexes for job
$i \in M$	index for machine
$g, h \in F$	indexes for family

Parameters

p_{ji}	processing time for job j using machine i
s_{jk}	setup time for job k when it immediately follows job j
d_j	due date of job j
α_j	earliness cost per unit of time for job j
β_j	tardiness cost per unit of time for job j
G_{gh}	changeover time from job j belongs to family g to a job k belongs to a family h; G_{gh} is equal to 0 when job j and job k belong to the same family
M^+	a large positive number

Decision variables

	Non-negative variables
E_j	length of time job j is early
T_j	length of time job j is tardy
C_j	completion time of job j
	Binary variables
X_{ji}	1 if job j is the first processed on machine i; 0, otherwise
Y_{ji}	1 if job j is done on machine i, but not in the first place; 0, otherwise
Z_{jk}	1 if job j precedes job k; 0, otherwise

2) Objective functions

The objective function, which is the function that needs to be optimised, is presented. Expression (2) defines the objective function as the minimisation of the sum of total weighted earliness and tardiness.

$$min \sum_{j \in N} (\alpha_j E_j + \beta_j T_j) \tag{2}$$

3) Constraints

The constraints of the mathematical model are detailed, taking into account the specific characteristics of the problem under study, which is the UPM-JSDST-JF-PC scheduling problem.

Constraints (3) define the earliness and tardiness of job j. Each job can only be tardy or early, if it is not delivered timely, so it is obvious that T_j and E_j cannot take a positive value simultaneously.

$$C_j - T_j + E_j = d_j \qquad \forall j \in N \tag{3}$$

Constraints (4) ensure that each job is processed on one and only one machine, and also, that all jobs are processed.

$$\sum_{i \in M} (X_{ji} + Y_{ji}) = 1 \qquad \forall j \in N \tag{4}$$

Constraints (5) enforce that a job and its direct successor are both processed on the same machine.

$$\begin{aligned} X_{ji} + Y_{ji} + Z_{jk} + Z_{kj} \leq Y_{ki} + 1 \\ \forall j,k \in N: \ j \neq k \quad \forall i \in M \end{aligned} \tag{5}$$

Constraints (6) guarantee that machine i has, at most, one job in the first place.

$$\sum_{j \in N} X_{ji} \leq 1 \qquad \forall i \in M \tag{6}$$

Constraints (7) enforce that a job is either the first to be processed, or succeeds another.

$$\sum_{i \in M} X_{ji} + \sum_{k \in N} Z_{kj} = 1 \qquad \forall j \in N \tag{7}$$

Constraints (8) ensure that every job should be at most immediately succeeded by another job, unless it is the last job.

$$\sum_{k \in N} Z_{jk} \leq 1 \qquad \forall j \in N \tag{8}$$

Constraints (9) ensure that the processing start time for a job can never be lower than the completion time of its direct predecessor job. Constraints (10) ensures that the processing start time for a job can never be higher than the completion time of its direct predecessor job. Constraints (9) and (10) guarantee that idle times are not allowed.

$$C_j + G_{gh} + s_{jk} + \sum_{i \in M} p_{ki}(X_{ki} + Y_{ki}) \leq C_k + M^+(1 - Z_{jk}) \qquad \forall g,h \in F, \quad \forall j \in F_g, \quad \forall k \in F_h \tag{9}$$

$$C_k \leq C_j + G_{gh} + s_{jk} + \sum_{i \in M} p_{ki}(X_{ki} + Y_{ki}) + M^+(1 - Z_{jk}) \qquad \forall g,h \in F, \quad \forall j \in F_g, \quad \forall k \in F_h \tag{10}$$

The value of M^+ is calculated according to equation (11) to make the constraints redundant when job j does not precede job k.

$$M^+ = \sum_{j \in N} \left(\max_{i \in M} p_{ij} + \max_{k \in N} s_{jk} + \max_{k \in N} G_{jk} \right) \tag{11}$$

Constraints (12) state that completion time of a job must be higher than or equal to its processing time.

$$C_j \geq \sum_{i \in M} p_{ji}(X_{ji} + Y_{ji}) \qquad \forall j \in N \tag{12}$$

Constraints (13) state that completion time of a job must be lower than or equal to its processing time, when the job is the first job to be processed.

$$C_j \leq \ p_{ji} X_{ji} + M^+(1 - X_{ji}) \qquad \forall j \in N \qquad \forall i \in M \tag{13}$$

Constraints (14) define the precedence constraints.

$$C_k \geq C_j + \sum_{i \in M} p_{ki}(X_{ki} + Y_{ki}) \qquad \forall k \in N \quad \forall j \in P(k) \tag{14}$$

Constraints (15) define the domain of binary variables.

$$X_{ji}, Y_{ji}, Z_{jk} \in \{0,1\} \qquad \forall j,k \in N \qquad \forall i \in M \tag{15}$$

Constraints (16) define the domain of non-negative variables.

$$T_j, E_j, C_j \geq 0 \qquad \forall j \in N \tag{16}$$

The problem under study is modelled as a MILP model, which is an exact model. However, the exact model cannot address the large instances in an acceptable computational time. Therefore, heuristic methods are widely used to approach the large instances, due to the fact that these methods find good solutions within a time period smaller than the required by an exact method, and requiring less computational effort. Although they do not guarantee optimality, heuristic methods, in some cases, manage to reach the optimal solution. To approach the problem under study, in the next chapter, will be proposed heuristic methods.

C. Heuristic approaches

In this section the two proposed heuristic approaches are described. Firstly, the MS ILS-RVND heuristic is characterised and, lastly, the ALNS-RVND heuristic is described.

1) MS ILS-RVND heuristic

[18] propose the Multi Start Iterated Local Search - Randomised Variable Neighbourhood Descent (MS ILS-RVND), which is a unified heuristic algorithm for a large class of earliness and tardiness scheduling problems capable of producing high quality solutions when compared to the state-of-the-art methods available in the literature. Therefore, the UPM-JSDST-JF-PC scheduling problem will be addressed with the MS ILS-RVND algorithm. However, the algorithm needs some adaptions since [18] do not consider neither job families nor precedence constraints.

MS ILS-RVND is a meta-heuristic, which hybridises a multi-start ILS with a RVND meta-heuristic and is defined by the pseudo-code of the Algorithm 1 heuristic, which is characterised by restarting the ILS-RVND heuristic I_R times. The MS ILS-RVND heuristic starts by generating an initial solution. At each restart, a new initial solution is generated

and the ILS-RVND algorithm is applied. When I_R is achieved then the heuristic stops and returns the best solution found.

Algorithm 1 MS ILS-RVND

Parameters: I_R
$S \leftarrow ConstructInitialSolution()$
$S^* \leftarrow S$
$i \leftarrow 1$
while $i \leq I_R$ **do**
  $S \leftarrow ConstructInitialSolution()$
  $S^{'} \leftarrow ILS\text{-}RVND(S)$
  $i \leftarrow i + 1$
  if $f(S^{'}) \leq f(S^*)$ **then**
    $S^* \leftarrow S^{'}$

Algorithm 2 illustrates the pseudo-code used to construct the initial solutions. The Candidate List (CL) is built measuring the benefit of selecting each element. The earlier the due date is, the greater the benefit is of choosing the job first to be inserted in the initial solution. Precedence constraints are satisfied in the CL. Then, Restricted Candidate List (RCL) is generated, which is the list of the best candidates of CL. One of the jobs in the RCL is randomly chosen to be inserted in the solution. The feasibility is verified, that is, the precedence constraints have to be satisfied. Then, the job is added to the machine with the shorter completion time, that is, the freest available machine. A initial solution is constructed when the RCL is empty and all the jobs has assigned to a machine.

Algorithm 2 ConstructInitialSolution

$S \leftarrow BuildEmptySolution(m)$
$CL \leftarrow GenerateCL(n, D)$ ▷ Using EDD method and satisfying the precedence constraints
while CL is not empty **do**
  $RCL \leftarrow GenerateRCL(CL, S)$
  $j = SelectElementAtRandom(RCL)$
  if $Feasible(j)$ **then** ▷ Verify if the precedence constraints are satisfied
    $S \leftarrow InsertInFreestMachine(j, S)$ ▷ Insert into machine with shorter completion time
    $CL \leftarrow Remove(j, CL)$ ▷ Remove the inserted job from the CL

Algorithm 3 illustrates the pseudo-code of ILS-RVND heuristic, which is based on Local Search (LS) and perturbations that are applied until the termination condition is satisfied, which is the maximum number of consecutive perturbations without improvements (I_{ILS}). The LS part of ILS-RVND is managed by the RVND algorithm. The perturbation is used to escape from local optima by changing current solution in a random way. The used perturbation algorithm consists of moving a job j from a machine i to a machine i', while a job j' is moved from machine i' to machine i. The jobs, machines and positions to be inserted are chosen at random and such procedure is repeated three consecutive times, as proposed by [18].

Algorithm 4 illustrates the RVND meta-heuristic. The algorithm uses a set of neighbourhood structures $N = \{N_1, ..., N_v\}$ in a random way until the solution S^* becomes a local optimum for each of them. RVND starts by exploring a neighbourhood. If an improvement has been found within a neighbourhood, the neighbourhood for the new solution is applied again. If no improvement has been found, the next neighbourhood is explored. The algorithm terminates if all neighbourhoods for the current solution are explored without finding an improvement, that is, the current solution is a local optimum for all neighbourhoods. The searching of solutions in the neighbourhoods is performed until the best neighbour is found.

Algorithm 3 ILS-RVND

Parameters: I_{ILS}
$S^* \leftarrow ILS\text{-}RVND(S)$
$i \leftarrow 1$
while $i \leq I_{ILS}$ **do**
  $S^{'} \leftarrow perturb(S^*)$
  $S^{''} \leftarrow ILS\text{-}RVND(S^{'})$
  $i \leftarrow i + 1$
  if $f(S^{''}) \leq f(S^*)$ **then**
    $S^* \leftarrow S^{'}$
    $i \leftarrow 1$

Algorithm 4 RVND

$S^* \leftarrow S$
$\{i_j\}_{j=1}^{v} \leftarrow randomIndexes(1, v)$
$n \leftarrow 1$
while $n \leq v$ **do**
  $S^{'} \leftarrow N_{i_n}(S^*)$
  if $f(S^{'}) \leq f(S^*)$ **then**
    $S^* \leftarrow S^{'}$
    $n \leftarrow 1$
  else
    $n \leftarrow n + 1$

We consider five neighbourhoods. The neighbourhoods are described using blocks, which are sub-sequences of l consecutive jobs. The neighbourhoods used are based on insertion and swap moves of blocks. The neighbourhood $InsertionIntraMachine$ consists of moving the block from the current position p to a forward position $p+2$ or to a backward position $p-2$ in the same machine. The neighbourhood $SwapIntraMachine$ consists of interchanging a block in position p with the block in the position $p+2$ belonging to the same machine. The neighbourhood $InsertionInterMachine$ consists of removing a block in position p from a machine i and inserting it in machine i' at same position. The neighbourhood $SwapInterMachine$ consists of interchanging a block in a position p of a machine i with the block in position p of a machine i'. The neighbourhood $SwapMachines$ consists of interchanging all the jobs from machine i to machine i'. Neighbourhoods 1 to 4 are proposed by [18] and neighbourhood 5 is proposed within the scope of this dissertation because it causes improvements in the quality of the solution obtained by the meta-heuristic for the problem addressed.

2) ALNS-RVND heuristic

[27] suggest that the Adaptive Large Neighbourhood Search (ALNS) is an efficient heuristic to address scheduling problems and [19] propose the combination between the ALNS and a LS meta-heuristic with a certain probability. Therefore, in order to use a novel heuristic for the problem under study, we developed the Adaptive Large Neighbourhoods Search - Randomised Variable Neighbourhood Search (ALNS-RVND) heuristic to address the UPM-JSDST-JF-PC scheduling problem. It is equipped with destroy and repair methods applied according to weights that are the dynamically adjusted throughout the search process.

The pseudo-code of the ALNS-RVND heuristic is illustrated by Algorithm 5. The sets of neighbourhoods are divided into two: the set of destroy neighbourhoods Ω^- and the set of repair neighbourhoods Ω^+. The weights of each destroy and repair neighbourhood are denoted by ρ^- and ρ^+, respectively, and initially all the methods have the same weight.

Algorithm 5 ALNS-RVND

```
S ← ConstructInitialSolution()
S* ← S
i ← 1
ρ⁻ ← (1, ..., 1)
ρ⁺ ← (1, ..., 1)
while i ≤ I_ALNS do
    Select destroy and repair methods d ∈ Ω⁻ and r ∈ Ω⁺ using φ⁻ and φ⁺
    S' ← r ( d (S) )
    Apply RVND to S' with probability prob
    if f(S') ≤ f(S) then
        S ← S'
    if f(S') ≤ f(S*) then
        S* ← S'
    i ← i + 1
    Update φ⁻ and φ⁺
```

The stopping criteria is when the number of iterations performed, i, is equal to the maximum number of iterations, I_{ALNS}. In each iteration, a roulette wheel is used to choose a destroy and a repair methods, according to the probabilities schemes of ALNS (see in [27]).

Then, the chosen destroy and repair methods are applied to the current solution and a new temporary solution $S^{'}$ is built. Since ALNS-RVND algorithm applies a LS procedure with a certain probability, the RVND is applied to the new temporary solution $S^{'}$ with the probability $prob$. Afterwards, if the temporary solution $S^{'}$ is better than the current solution S, S become equal to $S^{'}$, if the temporary solution $S^{'}$ is better than the best solution found until now S^*, S^* become equal to $S^{'}$. Finally, the procedure updates the weights, ρ^- and ρ^+ (see in [27]).

Three destroy methods and two repair methods were developed for the UPM-JSDST-JF-PC scheduling problem. The destroy methods are the following:

1) $RandomRemoval$, as the name suggests, randomly selects Q jobs using a uniform distribution and removes them from the solution, in order to explore other parts of the solution space.
2) $WorstRemoval$ consists of removing the Q jobs with the highest cost in the solution, that is, the jobs with the highest total earliness and tardiness in the current solution.
3) $RelatedRemoval$ happens when a job is randomly selected and the remaining $Q-1$ are sorted in a decreasing order of relatedness. Relatedness is a weighted sum of two factors, which are enumerated below. All the factors have the same weight and the total relatedness is defined as $rel(j,k) = \frac{1}{2}rel_1(j,k) + \frac{1}{2}rel_2(j,k)$. The $rel_1(j,k)$ is equal to 1 if jobs j and k belong to the same family and 0, otherwise and the $rel_2(j,k)$ is equal to 1 if jobs have precedence between them and 0, otherwise. The $Q-1$ jobs with highest relatedness with job j and job j are removed from the current solution.

The repair methods are the following:

1) $SelfishRepair$ consists of greedly inserting the removed jobs based on the cheapest insertion. The job is inserted in all possible positions and its individual cost, that is, the earliness and tardiness of the job, is calculated. The chosen position is where the job has the lowest cost;
2) $AltruistRepair$ consists of inserting the removed jobs in the position that minimises the objective function, using the exact model.
3) $RandomRepair$ consists of inserting the job randomly in a position chosen using a uniform distribution.

All the repair methods guarantee that the precedence constraints are satisfied.

IV. Results

A. Instances generation

The scheme for the instance generation is similar to that presented by [21] and [24]. Job families are generated according the proposal from [7]. Jobs precedence are generated according [12] and [10]. According to the number of jobs n and the number of machines m, the instances are classified in two sets: set I and set II. Set I contains small instances with $n \in \{10, 20, 30\}$ and $m \in \{3, 5\}$. Set II contains large instances with $n \in \{50, 80, 120\}$ and $m \in \{10, 20\}$.

B. Exact model

Table III presents the average computational time in seconds ($\overline{time}$ [s]), the average gap in percentage ($\overline{gap}$ [%]) and the number of optimal solutions (# optimums), for each set of instances grouped by number of jobs (n). The exact method only obtains all the optimal solutions for instances with $n = 10$, which are the smallest instances. For instances with $n = 20$, the model solves 1 instance. Regarding $n \in \{30, 50, 80, 120\}$, the model does not find any optimal solution for the instances within the time limit. Therefore, the exact model is not suited to address the instances with the number of jobs higher than 10, because it is suited to address to the obtain optimal solutions.

The exact is not able to provide a feasible solution within the time limit for instance with $n = 50$, $m = 20$ and $D = 0$, instance with $n = 80$, $m = 20$ and $D = 0.005$ and instances with $n = 120$, $m = 20$ and $D \in \{0, 0.005, 0.01\}$.

TABLE III: Comparison between the instances by number of jobs

n	$\overline{time}$ [s]	$\overline{gap}$ [%]	# optimum
10	36	0	12 / 12
20	3305	66	1 / 12
30	3600	78	0 / 12
50	3600	93	0 / 12
80	3600	95	0 / 12
120	3600	95	0 / 12

Table IV has the same structure as Table III, but the set of instances are grouped by the probability of precedence (D). The instances with a higher value of precedence probability take less time than the instances with a low value of D. Moreover, the higher quality solutions are obtained for the instances with higher precedence probability. It happens because when D takes a higher value, the model becomes more constrained and with a smaller amount of feasible job orders and solution space.

TABLE IV: Comparison between the instances by job precedence probability

D	$\overline{time}$ [s]	$\overline{gap}$ [%]	# optimum
0	2411	80	2 / 12
0.005	3005	80	2 / 12
0.01	3011	77	2 / 12
0.02	3006	76	2 / 12
0.1	3001	65	2 / 12
0.3	2705	50	3 / 12

C. Heuristic results

Firstly, the results of MS ILS-RVND heuristic are presented. Secondly, the results of the ALNS-RVND heuristics are summarised. Lastly, a comparison between the performance of the heuristics is provided.

We will evaluate the quality of the solutions obtained by the heuristic algorithms using the gap in percentage. The gap is defined as in Equation (17). The reference objective is the feasible solution provided by the exact model within the time limit. The gap metric allows to identify if a given heuristic solution is better than the exact solution, since it will be negative if that is the case. This metric to evaluate heuristics was inspired in [6].

$$gap = \frac{objective_{heuristic} - objective_{exact}}{objective_{exact}} \cdot 100 \qquad (17)$$

When the exact model does not find any feasible solution within the time limit, the reference objective used is the Best Know Objective (BKO) of this instance. All the experiments are repeated for 5 different seeds, since the heuristic has some random components, and then the average values are calculated.

1) MS ILS-RVND heuristic

The parameters was tuned in a set of pre-experiments: $l = 1$, $I_{ILS} = n/2$ and $I_R = 5$. Table V summarises the results grouped by number of jobs provided by the MS ILS-RVND heuristic. This table contains the number of jobs (n), the average minimum gap in percentage ($\overline{gap_{min}}$ [%]) and the average time in seconds ($\overline{time}$ [s]). The MS ILS-RVND heuristic provides better quality solutions than the exact model, especially for the large instances. In all instances, excepted when $n = 10$, the heuristic provides a negative average gap, meaning that the solutions found is better than those provided by the exact model.

TABLE V: Summary of the results provided by MS ILS-RVND heuristic for the instances grouped by number of jobs

n	$\overline{gap_{min}}$ [%]	$\overline{time}$ [s]
10	1.743	0.205
20	-4.677	1.643
30	-18.908	6.294
50	-75.987	141.948
80	-80.275	675.981
120	-80.306	2601.468

When the number of jobs increases, the value of the average gap decreases. In other words, the MS ILS-RVND provides lower BKOs, which is great since the objective is of minimisation, when the instances are large.

2) ALNS-RVND heuristic

Table VI summarises the results grouped by number of jobs provided by the ALNS-RVND heuristic. This table has the same structure as Table V. The parameters was tuned in a set of pre-experiments: $prob$ = 0.3, $I_{ALNS} = 5$ and $\lambda = 0.55$.

For instances $n = 10$, the ALNS-RVND heuristic reaches all the optimal solutions, since the average minimum gap is equal to 0%. The ALNS-RVND obtains on average a better solution, for instances with $n \in \{20,\ 30,\ 50,\ 80,\ 120\}$ because the average minimum gap value is negative. The computational times are also lower, when compared to the times taken by the exact model. The higher the number of jobs, the lower is the gap, meaning that when the instances are larger, BKOs found by the heuristic is further from the objective found by the exact method.

The ALNS-RVND heuristic is unable to the UPM-JSDST-JF-PC scheduling problem in all instances with $n = 10$ since it finds the optimal solutions in less computational time. For the other instances, the solution found by the heuristic has, in average, more quality than the exact method solution and is obtained in less computational time.

TABLE VI: Summary of the results provided by ALNS-RVND heuristic for the instances grouped by number of jobs

n	$\overline{gap_{min}}$ [%]	$\overline{time}$ [s]
10	0	1.415
20	-7.046	4.839
30	-20.124	11.639
50	-76.191	107.823
80	-79.652	274.753
120	-80.422	683.655

Similarly to the MS ILS-RVND heuristic, the ALNS-RVND heuristic seems to provide a good compromise between the

quality of the solution and the computational time and the application of this heuristic to the JIT industries brings benefits, since it can reach good solutions in reasonable computation times.

3) Comparison between the MS ILS-RVND and ALNS-RVND heuristics

A comparison between MS ILS-RVND and ALNS-RVND heuristics is established in order to understand which heuristic is more adequate to address the UPM-JSDST-JF-PC scheduling problem.

Table VII summarises the results obtain by MS ILS-RVND and ALNS-RVND heuristics, considering the instances grouped by the number of jobs (n). The best average minimum gap is highlighted in bold. The last row of the tables are the global average of the metrics.

To solve instances with $n = 10$, the ALNS-RVND heuristic obtains better quality solutions, that is, reaches always the optimal solution. Therefore, to address these instances, we suggest to use ALNS-RVND heuristic. However, to address instances with $n \in \{20,\ 30,\ 50,\ 80,\ 120\}$, there is no consensus about which heuristic should be chosen, since there is no heuristic that is able to provide the best quality solution in every instance tested.

The MS ILS-RVND heuristic takes a shorter computational times for instances with $n \in \{10,\ 20,\ 30\}$ and, the ALNS-RVND heuristic takes a shorter computational times for instances with $n \in \{50,\ 80,\ 120\}$. The ALNS-RVND heuristic obtains lower values of the average minimum gap for instances with $n \in \{10,\ 20,\ 30,\ 50,\ 120\}$. For instances with $n = 80$, the MS ILS-RVND obtains a better average minimum gap.

TABLE VII: Comparison between the results provided by both heuristic methods grouped by number of jobs, n

Instance	**MS ILS-RVND**		**ALNS-RVND**	
n	$\overline{gap_{min}}$ **[%]**	$\overline{time}$ **[s]**	$\overline{gap_{min}}$ **[%]**	$\overline{time}$ **[s]**
10	1.743	0.205	**0**	1.415
20	-4.677	1.643	**-7.046**	4.839
30	-18.908	6.294	**-20.124**	11.639
50	-75.987	141.948	**-76.191**	107.823
80	**-80.275**	675.981	-79.652	274.753
120	-80.306	2601.468	**-80.422**	683.655
average	-43.068	571.257	-43.906	180.687

Table VIII summarises the results obtained by MS ILS-RVND and ALNS-RVND heuristics, considering the instances grouped by the probability of precedence D. This table was constructed in the same manner as Table VII. The ALNS-RVND heuristic obtains better solutions for all the instances than the MS ILS-RVND heuristic. The ALNS-RVND heuristic takes less average computational times.

TABLE VIII: Comparison between the results provided by both heuristic methods grouped by probability of precedence, D

Instance	**MS ILS-RVND**		**ALNS-RVND**	
D	$\overline{gap_{min}}$ **[%]**	$\overline{time}$ **[s]**	$\overline{gap_{min}}$ **[%]**	$\overline{time}$ **[s]**
0	-44.667	981.310	**-45.015**	331.143
0.005	-39.696	1255.281	**-41.122**	272.395
0.01	-47.670	624.960	**-49.522**	175.380
0.02	-49.984	266.979	**-50.360**	91.992
0.1	-39.479	116.123	**-40.715**	96.938
0.3	-21.850	152.887	**-22.093**	116.275

Although it is difficult to choose between the two heuristics, as shown in Table VII, the global average minimum gap and the global average computational time of the MS ILS-RVND is -43.068% and 571.257 seconds, respectively, whereas the global average minimum gap and the global average computational time of the ALNS-RVND heuristic are -43.906% and 180.687 seconds, respectively.

The MS ILS-RVND and the ALNS-RVND heuristics are both suited to address the UPM-JSDST-JF-PC scheduling problem since, in general, both cause major improvements in overall objective function when compared to the exact method. Since the exact model did not obtain acceptable solutions in a reasonable time, it is necessary to use heuristic algorithms to obtain good quality solutions efficiently. However, it is not possible to prefer one heuristic over the other, as there is no one that finds the lowest cost solution for all instances.

It is difficult to suggest a single heuristic because each production context is unique and needs to be studied according with their specific characteristics. To make the best decisions, production managers need to have the best solutions with them in the shortest possible time, as both the quality of the solution and the time to obtain it affect the efficiency of their production. We suggest to the production manager of an JIT factory to choose an heuristic approach, since the proposed heuristics present better solutions when compared to the exact model, especially when applied to large instances.

V. Conclusions and Future Research

The literature regarding JIT scheduling on parallel machines is quite vast. However, not all scenarios are studied. In order to study a new scenario of JIT scheduling on parallel machines, a mathematical model is proposed. First, the UPM-JSDST-JF-PC problem is modelled as MILP. The exact method can only reach the optimal solution in reasonable computational times for the smallest instances, which are instances with $n = 10$.

Then we adapt the MS ILS-RVND heuristic found in the literature for similar problems and include the characteristics of the UPM-JSDST-JF-PC problem. To develop a novel solution approach to the problem under study, the ANLS-RVND is used to address it.

To assess the performance of the heuristic solutions, the objectives found by the exact method was taken as reference values. Both proposed heuristics can address the problem under study, obtaining higher quality solutions than the exact method in shorter computational times.

It is not possible to prefer one heuristic over the other, since none of them can provide the best quality solution in every instance tested. It is difficult to suggest a single heuristic because each production context is unique and needs to be studied according their specific characteristics.

As a future research line, we would like to study the UPM-JSDST-JF-PC scheduling problem with different number of jobs families, since in this dissertation the number of families is considered to be equal to 20% of the number of jobs. In an industrial context, there are many production lines with job families, thus, evaluating the performance of the proposed heuristics, considering different number of families, would

be interesting to bring the model closer to some industries, such as the chemical, metallurgical and textile industries. Other line for the future research is to address the UPM-JSDST-JF-PC scheduling but with the due date is a decision variable. The due date would be considered common for all jobs (d), so, it must be the same for all the jobs. The proposal of research problem can be modelled as the problem $R_m \mid d, s_{jk}, fmls, prec \mid \sum_{j \in N}(\alpha_j E_j + \beta_j T_j)$. This problem is not yet studied in the reviewed literature. The objective function would be the same as the problem under study. The main difference is that the due date is considered as a decision value and takes a unique value for all jobs. The possibility of due date negotiation is one interesting aspect of JIT scheduling, thus, in some cases, the production manager and his customer may negotiate about the due date of a job and, in this case, the decision-maker needs to set up an algorithm which returns a schedule and a preferable due date.

REFERENCES

[1] A. Aalaei, V. Kayvanfar, and H. Davoudpour. A multi-objective optimization for preemptive identical parallel machines scheduling problem. *Computational and Applied Mathematics*, 36(3):1367–1387, 2017.

[2] D. E. Akyol and G. M. Bayhan. Multi-machine earliness and tardiness scheduling problem: An interconnected neural network approach. *International Journal of Advanced Manufacturing Technology*, 37:576–588, may 2008.

[3] R. Amorim, B. Dias, R. De Freitas, and E. Uchoa. A hybrid genetic algorithm with local search approach for E/T scheduling problems on identical parallel machines. In *Proceedings of the 15th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '13 Companion)*, pages 63–64, 2013.

[4] N. Balakrishnan, J. J. Kanet, and S. V. Sridharan. Early/tardy scheduling with sequence dependent setups on uniform parallel machines. *Computers and Operations Research*, 26(2):127–141, feb 1999.

[5] J. Bank and F. Werner. Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling*, 33(4-5):363–383, feb 2001.

[6] R. Bernardino and A. Paias. The family traveling salesman problem with incompatibility constraints. *Networks*, mar 2021.

[7] Z. L. Chen and W. B. Powell. Exact algorithms for scheduling multiple families of jobs on parallel machines. *Naval Research Logistics*, 50(7):823–840, 2003.

[8] C. Y. Cheng and L. W. Huang. Minimizing total earliness and tardiness through unrelated parallel machine scheduling using distributed release time control. *Journal of Manufacturing Systems*, 42:1–10, jan 2017.

[9] R. Cheng, M. Gen, and T. Tozawa. Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. *Computers and Industrial Engineering*, 29(1-4):513–517, 1995.

[10] F. Della Croce and M. Trubian. Optimal idle time insertion in early-tardy parallel machines scheduling with precedence constraints. *Production Planning & Control*, 13(2):142, 2002.

[11] G. Feng and H. C. Lau. Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. *Annals of Operations Research*, 159(1):83–95, mar 2008.

[12] N. G. Hall and M. E. Posner. Generating Experimental Data for Computational Testing with Machine Scheduling Applications. *Operations Research*, 49(6):854–865, December 2001.

[13] R. B. Heady and Z. Zhu. Minimizing the sum of job earliness and tardiness in a multimachine system. *International Journal of Production Research*, 36(6):1619–1632, 1998.

[14] J. Józefowska. *Just-in-time scheduling: models and algorithms for computer and manufacturing systems*. Springer Science & Business Media, 2007.

[15] V. Kayvanfar, G. M. Komaki, A. Aalaei, and M. Zandieh. Minimizing total tardiness and earliness on unrelated parallel machines with controllable processing times. *Computers and Operations Research*, 41(1):31–43, 2014.

[16] V. Kayvanfar, M. Zandieh, and E. Teymourian. An intelligent water drop algorithm to identical parallel machine scheduling with controllable processing times: A just-in-time approach. *Computational and Applied Mathematics*, 36(1):159–184, 2017.

[17] S. Kedad-Sidhoum, Y. R. Solis, and F. Sourd. Lower bounds for the earliness-tardiness scheduling problem on parallel machines with distinct due dates. *European Journal of Operational Research*, 189(3):1305–1316, sep 2008.

[18] A. Kramer and A. Subramanian. A unified heuristic and an annotated bibliography for a large class of earliness-tardiness scheduling problems. *Journal of Scheduling*, 22(1):21–57, 2019.

[19] H. Kuhn, D. Schubert, and A. Holzapfel. Integrated order batching and vehicle routing operations in grocery retail – A General Adaptive Large Neighborhood Search algorithm. *European Journal of Operational Research*, 294(3):1003–1021, 2021.

[20] M. Laguna and J. L. G. Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2(4):253–260, 1991.

[21] Y. H. Lee and M. Pinedo. Scheduling jobs on parallel machines with sequence-dependent setup times. *European Journal of Operational Research*, 100(3):464–474, 1997.

[22] S. J. Mason, S. Jin, and J. Jampani. A moving block heuristic to minimise earliness and tardiness costs on parallel machines. *International Journal of Production Research*, 47(19):5377–5390, oct 2009.

[23] R. M'hallah and T. Al-Khamis. Minimising total weighted earliness and tardiness on parallel machines using a hybrid heuristic. *International Journal of Production Research*, 50(10):2639–2664, may 2012.

[24] J. Nogueira, J. Arroyo, H. Villadiego, and L. Gonçalves. Hybrid GRASP heuristics to solve an unrelated parallel machine scheduling problem with earliness and tardiness penalties. *Electronic Notes in Theoretical Computer Science*, 302:53–72, feb 2014.

[25] M. K. Omar and S. C. Teo. Minimizing the sum of earliness/tardiness in identical parallel machines schedule with incompatible job families: An improved MIP approach. *Applied Mathematics and Computation*, 181(2):1008–1017, oct 2006.

[26] M. L. Pinedo. *Scheduling: Theory, algorithms, and systems*. Springer Science & Business Media, 2008.

[27] D. Pisinger and S. Ropke. Large neighborhood search. *International Series in Operations Research and Management Science*, 272(June 2014):99–127, 2019.

[28] S. Polyakovskiy and R. M'Hallah. A multi-agent system for the weighted earliness tardiness parallel machine problem. *Computers and Operations Research*, 44:115–136, apr 2014.

[29] S. Radhakrishnan and J. A. Ventura. Simulated annealing for parallel machine scheduling with earliness-tardiness penalties and sequence-dependent set-up times. *International Journal of Production Research*, 38(10):2233–2252, 2000.

[30] K. Raja, V. Selladurai, R. Saravanan, and C. Arumugam. Earliness-tardiness scheduling on uniform parallel machines using simulated annealing and fuzzy logic approach. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 222(2):333–346, 2008.

[31] G. A. Rolim and M. S. Nagano. Structural properties and algorithms for earliness and tardiness scheduling against common due dates and windows: A review. *Computers and Industrial Engineering*, 149:1–30, 2020.

[32] F. Sivrikaya-Şrifoğlu and G. Ulusoy. Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, 26(8):773–787, 1999.

[33] V. T'kindt and J. C. Billaut. *Multicriteria scheduling: Theory, models and algorithms*. Springer Science & Business Media, 2006.

[34] E. Vallada and R. Ruiz. Scheduling unrelated parallel machines with sequence dependent setup times and weighted earliness-tardiness minimization. In *Just-in-time Systems*, pages 67–90. Springer Science & Business Media, 2012.

[35] Z. Zhu and R. B. Heady. Minimizing the sum of earliness/tardiness in multi-machine scheduling: A mixed integer programming approach. *Computers and Industrial Engineering*, 38(2):297–305, jul 2000.

[36] H. Şen and K. Bülbül. A strong preemptive relaxation for weighted tardiness and earliness/tardiness problems on unrelated parallel machines. *INFORMS Journal on Computing*, 27(1):135–150, 2015.