Delft University of Technology

Delft Center for Systems and Control

Technical report 23-011

Sequence- and Time-Dependent Maintenance Scheduling in Twice Re-Entrant Flow Shops*

E.-A. Eigbe, B. De Schutter, M. Nasri, and N. Yorke-Smith

To cite this work, please refer to the published version:

E.-A. Eigbe, B. De Schutter, M. Nasri, and N. Yorke-Smith, "Sequence- and time-dependent maintenance scheduling in twice re-entrant flow shops," *IEEE Access*, vol. 11, pp. 103461–103475, 2023. doi:10.1109/ACCESS.2023.3317533

Delft Center for Systems and Control
Delft University of Technology
Mekelweg 2, 2628 CD Delft
The Netherlands
phone: +31-15-278.24.73 (secretary)
URL: https://www.dcsc.tudelft.nl

* This report can also be downloaded via https://dspub.eu/23-011

Sequence- and time-dependent maintenance scheduling in twice re-entrant flow shops

EGHONGHON-AYE EIGBE[1], Bart De Schutter[2],(Fellow, IEEE), Mitra Nasri [3],(Member, IEEE), and Neil Yorke-Smith [4]

[1]Delft University of Technology, The Netherlands (e-mail: e.eigbe@tudelft.nl)
[2]Delft Center for Systems and Control, Delft University of Technology, The Netherlands (e-mail: b.deschutter@tudelft.nl)
[3]Eindhoven University of Technology, The Netherlands (e-mail: m.nasri@tue.nl)
[4]Delft University of Technology, The Netherlands (e-mail: n.yorke-smith@tudelft.nl)

Corresponding author: Eghonghon-aye Eigbe (e-mail: e.eigbe@tudelft.nl).

"This work was executed under the SAM-FMS project (NWO project 17931) in the MasCot programme."

ABSTRACT Industrial and academic interest converge on scheduling flow shops with sequence- and time-dependent maintenance. We posit that anticipatory, integrated scheduling of operational and maintenance tasks leads to superior performance to purely 'wait-then-fix' handling of the maintenance tasks. Motivated by an industrial problem with (sequence dependent) setup times, maximum separation constraints, and a combination of sequence- and time- dependent maintenance tasks, this paper introduces an integer programming solution, a constraint programming solution and a heuristic solution based on list scheduling. The motivating use case provides a unique combination of concerns that is to the best of our knowledge, not yet studied in the literature. We build on existing work where we can by extending models for sequence-dependent maintenance scheduling to accommodate sequence- and time-dependent maintenance scheduling and also propose other new models. We show the relative performances of our methods through empirical evaluations and also show significant improvements – up to 25% reduction in makespan – when compared to a reactive scheduling approach that does not consider maintenance in its planning. Based on our evaluations on exact methods, constraint programming models scale better than mixed integer programming models for this problem.

INDEX TERMS Flexible manufacturing systems, maintenance scheduling, makespan minimisation, re-entrant flow shops.

I. INTRODUCTION

We consider a sequence- and time-dependent maintenance scheduling problem. Our problem is motivated by an industrial use case of a large-scale printer (LSP) and is modelled as a flow shop. The operations in this problem have ordering constraints that enforce precedence and also maximum separation constraints that limit the delay between some of the operations. We also face setup time considerations. There are maintenance tasks which depend on the schedule: different sequences of operations have different deterioration effects on the machines. Additionally, the contribution of an operation to the total deterioration effect in a sequence is dependent on the timing of operations. Thus, our maintenance planning problem is both sequence- and time-dependent. The key question is how to handle both operational and maintenance tasks.

This is a challenging question because deteriorated machines produce low-quality jobs; there are thresholds beyond which deterioration of machines must be fixed by carrying out a maintenance activity. The overall objective is to find a feasible schedule that minimises the makespan.

Integrated production and maintenance planning is a challenge in many industries such as in wind farms [1], [2], in the capital goods industry [3] and the pulp and paper industry [4]. In many cases, machine deterioration is dependent on use, i.e., the maintenance required depends on how production operations have been scheduled. Sometimes, this dependence can be ignored and solutions can focus on preventive or policy based maintenance [5]–[7]. Recent work in this direction has used reinforcement learning to come up with these policies [8]. In other cases, the maintenance and production planning problem can be so integrated that the effect of use patterns on maintenance cannot be ignored. Previous work along these lines has considered different ways in which maintenance planning is integrated with production planning such as time-dependent maintenance [9] and position-dependent maintenance [10], [11]. The literature also considers different models of maintenance activities with one of the most popular models being that maintenance activities affect the processing time of operations [12] . While similar problems have been tackled in the literature, our work deals with a unique combination of maximum separation constraints and a deterioration effect not on the processing times of jobs but on the quality of work produced.

We present three solutions to this problem namely, (i) a

mixed integer programming solution, (ii) a constraint programming solution, and (iii) a list-scheduling based heuristic solution that extends the capabilities of existing schedulers to handle the kind of maintenance activities presented in this problem.

Through empirical evaluations, we show that in comparison to the reactive approach of scheduling only production operations and then performing maintenance activities when deterioration thresholds are crossed during a production run, our proactive approach achieves significant improvements in the makespan.

Parts of this paper were presented in a non-archival workshop [13]. In the workshop paper [13], we introduced the list-scheduling based heuristic (Section VI). The current paper presents the work in archival form, introduces two other solution methods, and also expands the scope of the evaluation to include more list-schedulers from the literature.

This paper is organised as follows: Section II discusses related work, Section III provides the background and problem definition, Sections IV, V and VI present mixed integer programming, constraint programming and heuristic solutions respectively. We perform empirical evaluations in Section VII and Section VIII concludes the paper.

II. RELATED WORK

The literature has investigated the dynamic relationship between machine deterioration and production scheduling from multiple angles ranging from ways to accurately determine the deterioration of a machine [14], [15] to actually generating schedules. We group the research themes in this field based on two categories, namely; (i) the way deterioration is modelled and (ii) the way maintenance activities are modelled.

Based on the *deterioration model*, existing research can be split into three main categories or approaches [16]. The *time-dependent* approach relates deterioration to the time at which a job is scheduled, i.e., scheduling a job later in the schedule incurs some additional deterioration which typically leads to longer processing times compared to scheduling it earlier. Closely related to this is a *position-dependent* approach, where deterioration effect of an operation is dependent on the number of preceding completed operations. Finally, there is the *sequence-dependent* approach in which the deterioration depends on the ordering or sequence of the preceding operations on the machine. As a result of the industrial challenge addressed in this paper, we focus on the sequence-dependent case with an additional challenge that the deterioration effect of an operation on a machine is not known a priori and is itself time-dependent.

The survey of Gawiejnowicz et al. [9] into the state of time-dependent scheduling problems has shown that the problem has been studied for single machine, parallel machine and dedicated machine use cases with a wide range of solution methods. However, situations where time-dependence of maintenance activities is coupled with sequence-dependence are unaddressed.

Yang et al. [17] consider the position-dependent maintenance scheduling problem on a set of parallel machines assuming that machines can only be maintained once within the planning horizon and with a constant maintenance duration. [10], [18] and [11] all consider position-dependent maintenance on a single machine with varying considerations such as the impact of time-dependent improvements in machine conditions, constraining job processing times to lie within an interval, and a combination of time and position-dependent deterioration respectively. [12] and [19] also consider the position dependent case but both add due-window considerations for just-in-time scheduling considerations.

The *sequence-dependent* approach is a more recent addition to the literature and can be considered as a generalisation of the time- and position-dependent approaches. Notably, [20] and [21] study sequence-dependent deterioration on a set of parallel machines without and with maintenance activities respectively. [22] considers iterated greedy heuristics for a similar problem and [23] considers the case where the parallel machines are not identical and processing time is based on a combination of deterioration and the speed of the assigned machine. Recently, [16] explored multiple integer programming models for solving the sequence-dependent maintenance problem on parallel machines and provided a heuristic approach for larger instances. The combination of sequence-dependent maintenance with other approaches and its effect in more complex manufacturing systems has not yet been studied.

Based on the *model of maintenance activities*, there are also different approaches in the literature. Some works such as [20] do not consider the presence of maintenance activities at all and aim to schedule in a way that deterioration is minimised. Other works such as [16] consider maintenance activities that reset the status of a machine to full health or 0% deterioration while a third category [12] considers rate-modifying maintenance activities that restore machine health by modifying the rate such that machines are able to perform work faster after maintenance. The authors of [24] and [25], classify maintenance activities into those that completely reset the state of the machines and those that restore the machines to some better deterioration state only. Additionally, the maintenance activities can be of fixed duration or can also have varying types based on how deteriorated a machine is.

A core assumption in many scenarios is that deterioration makes machines slower, thus increasing processing times of operations. Our work differs fundamentally in this regard in that using deteriorated machines does not have an effect on processing times, but instead affects the quality of the jobs produced. Our problem defines deterioration thresholds beyond which maintenance activities must be carried out to meet the quality requirements of future jobs. We also consider the case of maintenance activities that reset the state of the machine but also consider that there exist different classes of maintenance activities each with their own deterioration thresholds and incurring different costs.

An additional complication in our problem is the presence of maximum separation constraints, which impose additional

feasibility requirements on the problem. Exact solutions are able to easily model these additional requirements but heuristics run the risk of generating infeasible solutions in some cases. We therefore consider it necessary to design a solution for schedules that become infeasible due to the incorporation of maintenance activities. This concept of re-organising or repairing a changed schedule has been studied with various heuristics such as left and right shift [26], [27]. [28] combines multiple of these heuristics and a genetic schedule repair algorithm to build a solution that caters to multiple classes of schedule disturbances in a prefabrication plant.

In the context of flow shops, an example of schedule repair algorithms can be found in [29] which considers re-scheduling in a two-machine flow shop where schedules are disrupted by machine breakdowns. Additionally, [30] considers re-scheduling due to inserting new jobs in already planned schedules and [31] considers re-scheduling due to a wider range of disruptions in flow shop schedules at runtime. These cases all consider unexpected interruptions and do not have the combination of precedence and maximum separation constraints which provide an additional challenge for our problem.

In summary, there is a gap in the literature for sequence-dependent maintenance scheduling where deterioration effects of operations are not known a priori but are themselves time-dependent. The particular industrial challenge we consider has additional requirements of maximum separation that add to the complexity of the problem. Further, the schedule repair that is needed for heuristic schedulers that may produce infeasible schedules when we introduce maintenance activities, also requires new techniques.

III. PROBLEM DEFINITION

We consider a *maintenance-aware re-entrant flow shop with setup times and relative due dates* inspired by an industrial use case of a large-scale printer (LSP). The LSP prints different types of duplex sheets that need to be processed twice by the same print head at a speed of 100 or more pages per minute. In this setting, jobs to be scheduled refer to sheets to be printed.

In three-field or Graham notation [32], the base problem without maintenance is defined as $F|s_i, s_{ij}, limited-wait|C_{\max}$ indicating that it is a flow shop with both sequence-dependent and independent setup times, with maximum separation constraints between operations of the same job also known as limited-wait constraints, and with an objective to minimise makespan C_{max}. There is no preemption allowed and all jobs are released at time 0.

We represent the n-job m-machine maintenance-aware problem as the tuple $(M, J, O, P, S, D, \delta, X, O^M)$ where $M = \{\mu_1, \ldots, \mu_m\}$ is the set of machines and $J = \langle J_1, \ldots, J_n \rangle$ is the sequence of jobs. The set O represents the set of operations for every job $j_i \in J$ where each operation o_{ij} has a processing time P_{ij}. Each job has the same number of r operations as is in a standard flow shop. Moreover, $S : O \times O \to \mathbb{R}_{\geq 0}$ refers to setup times, which represent the required delay between the completion of an operation

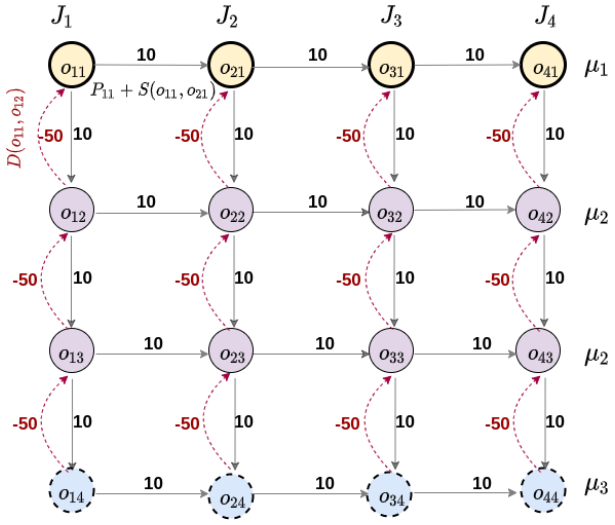

FIGURE 1: Sample re-entrant flow shop where the operations are represented by circles. Column-wise, we have operations of the same job and row-wise, we have operations on the same machine with one of these being the re-entrant machine that appears on rows 2 and 3. Operations with the same colour or boundary lines are mapped to the same machine. Setup times and maximum separation constraints are shown by solid and dashed edges respectively.

and the start of another operation. Setup times can exist between operations of the same job to model travelling time of a job for instance, or between operations on the same machine to model any machine preparation step that is needed between operations. Operations of the same job also have maximum separation constraints between them represented as $D : O \times O \to \mathbb{R}_{>0}$, i.e., the maximum delay between the start times of two consecutive operations of the same job. Such constraints model the fact that operations of a job can often not be delayed indefinitely due to physical constraints in the plant like the buffer size. In a situation where such constraints do not apply, separation constraints can simply be set to infinity and setup times to zero.

The solution to the problem is a schedule Ω, i.e., a sequence of both maintenance activities and production operations where each production operation is assigned a start time such that ω_{ij} represents the start time of operation o_{ij}.

In addition to being a flow shop with the above properties, we have a situation with re-entrancy such that the sequence of machines for each job is $\langle \mu_1, \ldots, \mu_k, \mu_k, \ldots, \mu_m \rangle$, i.e., there is one re-entrant machine that all jobs go through twice. Operations on the re-entrant machines are referred to as *first and second pass operations*. Re-entrancy occurs in many production processes, e.g. semi-conductor production, where wafers revisit machines at different stages of the production process and in painting processes where a job may revisit a machine for multiple coats of paint. Simple re-entrant setups have been shown to be NP-hard [33], [34]. Our motivating use case of production printing has a twice re-entrant setup arising from duplex printing.

We further have the constraints that (i) jobs are not allowed

to overtake each other, (ii) the required completion order of jobs is the same as the index of the jobs, and (iii) all setup times and due date constraints are hard constraints that must be obeyed. This situation means that the only scheduling freedom is in the sequence of operations on the re-entrant machine, i.e., first and second passes of the same jobs do not necessarily have to follow each other on this machine. This means we can also think of this as a single machine scheduling problem with precedence and maximum separation constraints.

In the same vein as only needing to schedule the re-entrant machine, we limit our maintenance planning to maintaining the re-entrant machine. While other machines also require maintenance, only the re-entrant machine requires maintenance in the same time scale as the operations being carried out, creating a very tightly coupled problem compared to maintenance of other machines. Additionally, the re-entrant machine is often a key machine of concern for cost reasons – re-entrant machines are too expensive to simply duplicate and remove the re-entrancy – or for quality reasons – some products need to be handled in a delicate state (chemical products for example) and moving the product from one machine to the other would change its state.

a: Deterioration model.

In our motivating industrial problem, there is a deterioration model $\delta : \Omega \rightarrow \mathbb{R}_{\geq 0}$, that maps a scheduled sequence of operations on a machine to a deterioration state, i.e., given a sequence of operations on a machine with their corresponding start times, i.e., a schedule, $\delta : \Omega\times \rightarrow \mathbb{R}_{\geq 0}$ informs us of the machine state at the end of the sequence. Here, δ is both sequence- and time-dependent in the sense that deterioration is measured by *idle time* of a machine part, i.e., the longer a machine part has been left idle, the more deteriorated it is. These idle times follow directly not only from the sequence themselves, but also from the assigned start times of operations in these sequences. We do not explicitly model machine parts and instead depend on the fact that different types of jobs use different machine parts and so it can be inferred which machine parts have been idle based on how long it has been since a certain job type has been scheduled. We assume that there is a set of job types $T = \{\tau_1, \ldots, \tau_n\}$ that can be presented to the machine and that there is a lexicographic ordering of job types such that every set of machine parts used by a job type τ_x is contained in the set of machine parts used by a job type $\tau_{y>x}$. It then follows that at the start of an operation of type τ_x, idle time is reset to 0 for all operations of type $\tau_{y\leq x}$. Note that while there could be other kinds of problems where different jobs use completely different machine parts and such a lexicographic ordering of types is not possible, it is still a realistic assumption for many scenarios, e.g., scenarios where jobs come in different sizes and bigger sizes simply use more machine parts for production or scenarios where jobs can be customised with different add-on properties processed by additional machine parts.

Finally, we also take as input a maintenance policy X. In our problem, the policy has a set of maintenance activity classes C. For every class $c \in C$, there is a corresponding maintenance duration P_c similar to processing times of production operations. The maintenance policy further maps intervals of deterioration values $[\theta_c, \Theta_c)$ to classes of maintenance activities such that whenever the deterioration falls in $[\theta_c, \Theta_c)$, a maintenance activity of at least class c is required before further production. Thus, $[\theta_c, \Theta_c)$ defines the interval of deterioration thresholds for a maintenance activity. We assume that these intervals are non-overlapping and that maintenance activities triggered by higher thresholds, i.e., harsher deterioration, are more intense and require longer durations. The deterioration thresholds serve to capture the limits at which the quality of a job would be too low if production carries on without a maintenance activity. In this context, a low-quality job refers to a poor print typically with colours bleeding into each other, blurry prints or unintended lines running across a page.

An example problem is shown in Figure 1. The problem is represented as a constraint graph where the due dates and setup times are treated as a system of difference constraints. Operations are represented as circles and each column of operations belong to one job, while each row of operations are mapped to the same machine. Solid arrows represent the minimum separation between operations and are made of the sum of processing and setup times while dashed arrows represent the maximum separation between operations and can be thought of as relative due dates. Minimum separation edges are represented with positive values and maximum separation edges are represented with negative values as they connote *at least* and *at most* constraints on the difference between the start times of the operations they connect.

IV. INTEGER PROGRAMMING APPROACH

Mixed Integer Programming (MIP) is one of the most popular exact solving paradigms and has been applied to other maintenance planning problems in the literature [16], [35] with some success. Due to the existence of a wide variety of commercial solvers, mixed integer formulations of a problem are valuable as solutions can be provided by these solvers. Furthermore, MIP models can give additional insight to the structure of a problem. Thus, we also consider such a solution for our problem.

In this section we present an exact MIP model for this problem. The model uses the concept of event based formulation as introduced by [16] and extends this concept to accommodate the kind of maintenance policies in this problem. The key idea here is the notion of *blocks* where a block is defined as a sequence of operations uninterrupted by a maintenance activity, i.e., a block is a sequence of operations separated from other operations in the sequence by at least one maintenance activity. For our model, we extend this idea to also include the effect of job types. A block is then defined as a sequence of operations uninterrupted by either a maintenance activity or an operation with a higher type than any other operation within the block.

We define binary variables to mark whether an operation

starts a block or not. These variables are further indexed by job type and maintenance activity class, i.e., an operation can be the start of a block delineated by a class of maintenance activities and/or the start of a block delineated by a job type. Blocks of different types are allowed to overlap with additional constraints added to ensure that maintenance is not triggered more than necessary.

The model retains all variables defined in the problem definition in Section III. Indices of variables corresponding to operations are either of the form x_{ij} when both the job and operation identifier are important or of the form x_a when it is only necessary to differentiate one operation from the other. For ease of modelling, we also define binary variables $\Gamma_{am}, \quad \forall o_a \in O, \mu_m \in M$ to represent machine assignment. Then, Γ_{am} is set to 1 if operation o_a is assigned to machine μ_m. Γ_{am} is not a decision variable and is part of the problem description.

Additionally, since we only plan maintenance on the re-entrant machines μ_k, many constraints only apply to operations on this machine and are denoted as R such that $R = \{o_a \in O | \Gamma_{ak} > 0\}$. Furthermore, a dummy operation o_{dummy} of processing time 0 is defined and constrained to be the first operation on each machine. We also extend the use of a job type τ to serve as a function that returns the type of an operation when written as $\tau(o)$.

The following additional variables are developed for the MIP model: ω_{ij} refers to the start time of operation $o_{ij} \in O$, B_{ab} is a binary variable relating to the precedence constraints between operations o_a and o_b. Note that B_{ab} refers only to direct precedence and not the general notion of o_a being scheduled sometime before o_b. We discretise job types such that τ_a refers to the type of o_a and assume that there is a lexicographic ordering of job types such that processing a job type with a higher value is sufficient to reset the machine for lower job types according to the maintenance policy described in Section III.

Block starts are marked by binary variables Z_a^c and ζ_a^τ where Z_a^c determines if operation o_a starts a block of operations delineated by a maintenance activity of class c and ζ_a^τ determines if operation o_a starts a block of operations delineated by a job type τ. Idle time values are held by the variables K_a^c and L_a^τ, which correspond to the minimum time elapsed since a maintenance activity of class c preceding o_a and the minimum time elapsed since an operation of type τ preceding o_a respectively. Furthermore, deterioration values at the start of an operation o_a are held by the variable δ_a and are determined by the deterioration values K and L.

Some of the constraints are linearised using big-M variables namely, $\mathcal{M}^\tau$ and $\mathcal{M}^\omega$. We define some bounds for these variables in Section IV-B below.

A. THE INTEGER PROGRAMMING MODEL

In this section, we define the integer programming model made up of an objective function, decision variables and constraints.

Objective

$$\min(C_{max}) \tag{1}$$

Decision variables

B_{ab}	Operation o_a directly precedes o_b	$B_{ab} \in \{0,1\}$
ω_a	Start time of operation o_a	$\theta_a \in \mathbb{R}$
L_a^τ	Minimum time elapsed since operation of a type τ preceding o_a	$L_a \in \mathbb{R}$
K_a	Minimum time elapsed since any maintenance activity preceding o_a	$K_a \in \mathbb{R}$
δ_a	Deterioration of machine at start of o_a	$\delta_a \in \mathbb{R}$
Z_a^c	o_a starts a block delineated by a maintenance activity of class c	$Z_a^c \in \{0,1\}$
ζ_a^τ	o_a starts a block delineated by a job of type τ	$Z_a^c \in \{0,1\}$

Constraints

$$\omega_{(i+1)j} \geq \omega_{ij} \ \ \forall o_{ij} \in O \tag{2a}$$
$$\omega_{i(j+1)} \geq \omega_{ij} + P_{ij} + S(o_{ij}, o_{i(j+1)}) \ \ \forall o_{ij} \in O \tag{2b}$$
$$\omega_{i(j+1)} \leq \omega_{ij} + D(o_{ij}, o_{i(j+1)}) \ \ \forall o_{ij} \in O \tag{2c}$$
$$\sum_{o_a \in O} B_{ad} = 0 \tag{2d}$$
$$\sum_{o_a \in O \cup \{o_{\text{dummy}}\}} B_{ab} = 1 \ \ \forall o_b \in O \tag{2e}$$
$$\sum_{o_b \in O} B_{ab} \leq 1 \ \ \forall o_a \in O \tag{2f}$$
$$B_{ab} \leq \sum_{\mu_m \in \mu} \Gamma_{am}\Gamma_{bm} \ \ \forall o_a, o_b \in O \tag{2g}$$
$$\omega_b \geq B_{ab}(\omega_a + P_a + S(o_a, o_b)) \ \ \forall o_a, o_b \in O \tag{2h}$$
$$\omega_b \geq B_{ab}(\omega_a + P_a + Z_b^c(P_c)) \ \ \forall o_a, o_b \in O, \ c \in C \tag{2i}$$
$$L_b^\tau \geq B_{ab}(\omega_b - \omega_a - P_a) \ \ \forall o_a, o_b \in R, \ \tau \in T \tag{2j}$$
$$L_b^\tau \geq B_{ab}(L_a^\tau + \omega_b - \omega_a) - \mathcal{M}^\omega \zeta_b^\tau \ \ \forall o_a, o_b \in R, \ \tau \in T \tag{2k}$$
$$K_a \geq 0 \ \ \forall o_a \in R \tag{2l}$$
$$K_b \geq B_{ab}(K_a + \omega_b - \omega_a) - \mathcal{M}^\omega \sum_c Z_b^c \ \ \forall o_a, o_b \in R \tag{2m}$$
$$\delta_a \geq \min(K_a, L_a^{\tau(o_a)}) \ \ \forall o_a \in R \tag{2n}$$
$$\mathcal{M}^\omega(1 - Z_a^c) + (\delta_a - \theta_c) \geq 0 \ \ \forall o_a \in R, \ c \in C \tag{2o}$$
$$\mathcal{M}^\omega Z_a^c - (\delta_a - \theta_c) > 0 \ \ \forall o_a \in R, \ c \in C \tag{2p}$$
$$\mathcal{M}^\tau \zeta_b^\tau - B_{ab}(\tau(o_a) - \tau) \geq 0 \ \ \forall o_a, o_b \in R, \ \tau \in T \tag{2q}$$
$$\mathcal{M}^\tau(1 - \zeta_b^\tau) + B_{ab}(\tau(o_a) - \tau) \geq 0 \ \ \forall o_a, o_b \in R, \ \tau \in T \tag{2r}$$
$$C_{max} = \omega_{|J|r} + P_{|J|r} \tag{2s}$$

The objective of the model is to minimise makespan denoted by Equation (1). The constraints in Equations (2b) to (2h) apply to all operations while the constraints in Equations (2j) to (2r) only apply to operations scheduled on the re-entrant machine. All non-binary variables are constrained to be non-negative, i.e., start times, idle times and deterioration values all have a lower bound of 0.

Equation (2a) enforces the fixed-order relationship between operations at the same level of the flow shop. Equations (2b)

and (2c) enforce setup times and maximum separation constraints between operations of the same job respectively, while Equation (2d) enforces that the dummy operation has no predecessors. Equation (2e) ensures that every operation has exactly one predecessor and Equation (2f) enforces that every operation has at most one successor. Equation (2g) enforces that operations only follow each other if they are mapped to the same machine and Equation (2h) enforces that there is no overlap between operations leaving room for setup times. These make up the constraints that specify the problem without maintenance.

The maintenance constraints follow below. Equation (2i) enforces that there is no overlap between operations while leaving enough room for any maintenance activities that may have been triggered. Equations (2j) and (2k) specify constraints on the minimum time elapsed since an operation of a certain type has come through the machine. Similarly, Equations (2l) and (2m) specify the minimum time elapsed since a maintenance activity of a certain class has been scheduled. The constraints represented by Equations (2j) to (2m) are defined in a cumulative way based on predecessor operations. Equations (2k) and (2m) are activated depending on the presence of a job type or a maintenance activity respectively. This toggle is implemented by big-M values that are activated based on the binary variables Z and ζ.

The actual deterioration value is computed by Equation (2n) which is set to the minimum of both K and L. Equation (2n) computes deterioration based on the idle time so far and is set to the minimum of K and L so that maintenance is only triggered when necessary. Equations (2o) and (2p) specify that maintenance activities are triggered whenever deterioration thresholds are crossed thereby starting a new block while Equations (2q) and (2r) similarly start a new block based on the relationship between types of operations, i..e, a new block is triggered whenever an operations predecessor has a higher type. Note that this model allows multiple maintenance classes to be triggered simultaneously if the threshold violations cross multiple thresholds. However, Equation (2h) means that the gap left for maintenance corresponds to the largest processing time of all triggered maintenance activities, thus not paying unnecessary maintenance costs.

Finally, Equation (2s) calculates the makespan which in a fixed order problem, is the finishing time of the last operation of the last job.

B. BOUNDS FOR BIG-M VALUES

1) $\mathcal{M}^{\omega}$

Throughout the model, $\mathcal{M}^{\omega}$ is used as a big-M constraint in two instances. The first is in Equations (2k) and (2m) to sum up the minimum times since the last maintenance or the last occurrence of a type of job and in Equations (2o) and (2p) to toggle maintenance if deterioration thresholds are crossed. In each of these cases, the upper bound is the maximum possible deterioration value that can occur. Because our deterioration deals with idle times, we are then looking for a value that is larger than or equal to the maximum time the machine can be left idle.

An idea for this bound is to use an upper bound on the makespan as there always exists a solution with a better makespan than one in which the machine is left idle for the upper bound on the makespan.

This upper bound assumes the worst case which is that every operations incurs the maximum possible setup time and the maintenance activity with the longest duration occurs before every operation. Thus, the bound is

$$\mathcal{M}^{\omega} = \sum_{o_a \in O} \left(P_a + \max_{c \in C}(P_c) + \max_{o_b \in O}(S(o_a, o_b)) \right). \quad (3)$$

2) $\mathcal{M}^{\tau}$

The tightest bound for $\mathcal{M}^{\tau}$ is the largest job type available in the problem. This holds because:

- $\mathcal{M}^{\tau}$ is an upper bound on the types of jobs,
- we assume that job types are all given integer values corresponding to their quality requirements ,
- we assume there is a lexicographical ordering of these types that corresponds with the order of the integers representing each job type.

C. LINEARISING THE MODEL

Some equations are still quadratic namely Equations (2h), (2i), (2k) and (2m). They however all involve the product of a binary variable and a non-negative continuous variable and can also be linearised via the big-M method. The corresponding M is also $\mathcal{M}^{\omega}$. A detailed explanation of how this linearisation is achieved can be found in [36].

Additionally, Equation (2n) requires us to compute the minimum which is also a non-linear equation. We define one more auxiliary binary variable γ_a that is set to 1 if K_a is less than $L_a^{\tau(o_a)}$ and linearise the minimum constraint by replacing it with the following set of equations:

$$\delta_a \leq K_a \ \ \forall o_a \in R, \quad (4a)$$
$$\delta_a \leq L_a^{\tau(o_a)} \ \ \forall o_a \in R, \quad (4b)$$
$$\delta_a \geq K_a - \mathcal{M}^{\omega}(1-\gamma_a) \ \ \forall o_a \in R, \quad (4c)$$
$$\delta_a \geq L_a^{\tau(o_a)} - \mathcal{M}^{\omega}(\gamma_a) \ \ \forall o_a \in R, \quad (4d)$$
$$K_a - L_a^{\tau(o_a)} \leq \mathcal{M}^{\omega}(1-\gamma_a) \ \ \forall o_a \in R, \quad (4e)$$
$$L_a^{\tau(o_a)} - K_a \leq \mathcal{M}^{\omega}(\gamma_a) \ \ \forall o_a \in R. \quad (4f)$$

Equations (4a) and (4b) set the deterioration value δ_a to be upper bounded by the minimum of K_a and $L_a^{\tau(o_a)}$. However, this is not enough as δ_a is still free to take any values less than this and can lead to violations of maintenance constraints. We further use Equations (4c) and (4d) which set δ_a to be lower bounded by the minimum of K_a and $L_a^{\tau(o_a)}$. This lower bound is also achieved via big-M constraints which activate either equation based on γ_a. The combination of the lower and upper bounds ensure that δ_a is exactly set to the minimum of the two values K_a and $L_a^{\tau(o_a)}$. Finally, Equations (4e) and (4f) set γ_a to 0 if K_a is less than $L_a^{\tau(o_a)}$ and 1 otherwise.

V. CONSTRAINT PROGRAMMING APPROACH

Constraint programming (CP) has recently been shown to perform well for scheduling problems [37]. This motivates us to also explore a constraint programming solution. In this section, we present a CP model.

Our CP model uses the idea of *interval variables* and *sequence variables*. These are known constraint programming concepts [37] with the following definitions. Interval variables refer to operations to be scheduled and are declared with a length equal to the processing time of the operation. The goal of the solver is to assign a start time to each of these variables. An additional characteristic of interval variables are that they have the option to either be compulsory, i.e., they must exist in any schedule produced by the solver, or be optional. Sequence variables on the other hand, represent orderings of interval variables. The solver receives these as a set of interval variables with its goal being to decide on a sequencing of these interval variables.

Apart from constraints and variables, constraint programming also provides some auxiliary functions such as **startOf** and **typeOf** which help us access variable properties – in this case, their assigned start times and types respectively.

We define two classes of interval variables, (i) operations which are always present and each retain the representation of o_a and (ii) maintenance activities which are optional and referred to as m_a^c where m_a^c is a maintenance activity of class c that precedes an operation o_a. The variables K, L and R retain their definitions from the MIP model in Section IV.

Given that we have $|A|$ maintenance classes and $|R|$ operations in total on the re-entrant machine, we define $|A||R|$ maintenance activities since the worst case is that there is one maintenance activity of a class before every regular operation. We add constraints such that the maintenance activities are included in the sequence only when deterioration thresholds are violated.

Sequence variables are defined per machine and referenced as $Sequence_m$ for corresponding machine μ_m where $Sequence_m$ contains all operations mapped to μ_m including the optional maintenance activities. For the re-entrant machine, we define an additional sequence variable $SequencePlain_m$ as a sequence of only production operations – excluding maintenance activities – and constrain this sequence to follow the same ordering as $Sequence_m$. The purpose of this duplicate sequence is to ensure that sequence-dependent setup times are respected. The details of how we achieve this follow in the constraint definitions below.

Objective

$$\min(C_{max}) \tag{5}$$

Decision variables

$Sequence_m$	Sequence of production and maintenance operations on machine μ_m	
$SequencePlain_m$	Sequence of production operations on machine μ_m	
L_a^τ	Minimum time elapsed since operation of a type τ preceding o_a	$L_a \in \mathbb{R}$
K_a	Minimum time elapsed since any maintenance activity preceding o_a	$K_a \in \mathbb{R}$

Constraints

$$C1: \textsf{before}(Sequence_1, o_{i1}, o_{(i+1)1}) \forall o_{i1} \in O \tag{C1}$$

$$C2: \textsf{sameSubsequence}(Sequence_1, Sequence_m) \quad \forall \mu_m \in \mu \tag{C2}$$

$$C3: \textsf{sameSubsequence}(Sequence_m, SequencePlain_m) \quad \text{for re-entrant machine } \mu_k \tag{C3}$$

$$C4: \textsf{startOf}(o_{i(j+1)}) \geq \textsf{startOf}(o_{ij}) + P_{ij} + S(o_{ij}, o_{i(j+1)}) \quad \forall o_{ij} \in O \tag{C4}$$

$$C5: \textsf{startOf}(o_{i(j+1)}) \leq \textsf{startOf}(o_{ij}) + D(o_{ij}, o_{i(j+1)}) \quad \forall o_{ij} \in O \tag{C5}$$

$$C6: \textsf{noOverlapDirect}(Sequence_m, P, S) \quad \forall \mu_m \in \mu \tag{C6}$$

$$C7: \textsf{noOverlapDirect}(SequencePlain_k, P, S) \quad \text{for re-entrant machine } \mu_k \tag{C7}$$

$$C8: \textsf{if}(\textsf{min}(K_a, L_a^{\tau(o_a)}) \geq \theta^c \wedge \textsf{min}(K_a, L_a^{\tau(o_a)}) < \Theta^c) \Rightarrow \textsf{presenceOf}(m_a^c) = 1 \quad \forall o_a \in R, c \in C \tag{C8}$$

$$C9: \textsf{if}(\textsf{presenceOf}(m_a^c) = 1) \Rightarrow K_a = 0 \quad \forall o_a \in R, c \in C \tag{C9}$$

$$C10: \textsf{if}(\textsf{presenceOf}(m_a^c) = 0) \Rightarrow K_a = K_{\textsf{indexOfPrev}(O_a)} + \textsf{startOf}(O_a) - \textsf{startOfPrev}(O_a) \quad \forall o_a \in R, c \in C \tag{C10}$$

$$C11: \textsf{if}(\textsf{typeOfPrev}(o_a) \geq \textsf{typeOf}(o_a)) \Rightarrow L_a^\tau = \textsf{startOf}(o_a) - \textsf{endOfPrev}(O_a) \quad \forall o_a \in R, \tau \in T \tag{C11}$$

$$C12: \textsf{if}(\textsf{typeOfPrev}(o_a) < \textsf{typeOf}(o_a)) \Rightarrow L_a^\tau = L^\tau_{\textsf{indexOfPrev}(o_a)} + \textsf{startOf}(o_a) - \textsf{startOfPrev}(o_a) \quad \forall o_a \in R, \tau \in T \tag{C12}$$

$$C13: C_{max} = \textsf{endOf}(o_{|j|r}) \tag{C13}$$

In this model, Constraint C1 enforces an ordering between the first operations of each job. The **before** constraint enforces precedence relationships between two operations in a sequence. We enforce the order of the first operations of each job which we know will be on the first machine (as we have a flow shop). Constraint C2 builds on C1 to then enforce that this ordering is respected across all other sequences using the **sameSubsequence** constraint. Note that we only enforce a subsequence because the re-entrant machine has operations on multiple levels of the flow shop. The **sameSubsequence** is set up such that only operations at the same level are constrained with the fixed ordering, which is in line with the requirements of our problem. Constraint C3 uses the **sameSubsequence** in a similar way to constrain the duplicate

sequences – with and without maintenance activities included – to have the same ordering.

Next, Constraints C4 and C5 enforce the sequence-independent setup times and maximum separation constraints respectively. Both of these apply to operations of the same job as is seen with the index of operations in the constraints.

Sequence-dependent setup time and no overlap constraints are handled by Constraints C6 and C7, which ensure that both the separations required by processing times and sequence-dependent setup times are obeyed. Since maintenance activities are also included in our sequences, we ensure correctness of Constraints C6 by extending the processing and setup times accordingly with setup times set to 0 for operations before or after maintenance activities. The noOverlapDirect constraint works such that the separation denoted by sequence-dependent setup times applies only between direct successors, i.e., say an operation o_a is followed by o_b with a maintenance activity m_b^c in-between, the setup time between o_a and o_b will not be enforced. Thus, setting the maintenance setup time to 0 can lead to constraint violations as the problem is now under constrained. It is worthy of note that there exists a noOverlapIndirect constraint, which applies sequence dependent setup time constraints to all successors; however, this over-constrains the problem[1]. We use the noOverlapDirect constraint and circumvent under-constraining the problem by using the supporting Constraint C7 on a duplicate sequence without maintenance.

Constraint C8 enforces the presence of a maintenance activity whenever the minimum deterioration is within the limits of threshold violations. We do not explicitly calculate a deterioration variable δ in this model but this is essentially the left hand side of Constraint C8. We depend on the fact that our problem defines non-overlapping maintenance threshold intervals to ensure that at most one maintenance activity is triggered before an operation.

Constraints C9 and C10 deal with the computation of the minimum time elapsed since a maintenance activity has occurred. Since we are guaranteed to trigger at most one maintenance activity per operation, we do not maintain different minimum elapsed times per maintenance class as was done in the MIP model. Similarly, Constraints C11 and C12 compute the minimum time elapsed since a job of a certain type has been through the machine.

Finally, C13 calculates the makespan, which we again know to be the finishing time of the last operation of the last job.

Worthy of note is that Constraints C10 and C12 are cumulative constraints that could be expressed using the cumulFunction constraint, which keeps track of each interval variables contribution to a function [37]. However, many implementations of this function within available solvers require that the contribution of each interval variable be known a priori whereas, in our case, the contribution of each interval variable is itself based on decision variables [37] due to maintenance also being time-dependent[2].

[1]Given a sequence of operations $o_a \rightarrow o_b \rightarrow o_c$, sequence-dependent setup times will be considered from $o_a \rightarrow o_b$, $o_b \rightarrow o_c$ and $o_a \rightarrow o_c$ whereas the only sequence-dependent setup times that should be considered are from $o_a \rightarrow o_b$ and $o_b \rightarrow o_c$.

VI. HEURISTIC SOLUTION APPROACH

While exact approaches such as those presented in Sections IV and V have lots of advantages, they often do not scale well. In this section, we present an alternate heuristic solution approach to handle larger problem instances. The work presented in this section has appeared earlier in a workshop paper [13].

Our heuristic approach is based on extending list schedulers to integrate maintenance activities in the schedule. Heuristic list schedulers have been developed for the industrial problem we consider [38]–[40] and are also suitable for online scheduling. Thus, we look into extending them to handle integrated production and maintenance scheduling. The typical flow of a list scheduler is to *order* operations according to some metric and insert them in a schedule one after the other until all operations are scheduled [40].

A. MAINTENANCE-AWARE LIST SCHEDULING

To make a list scheduling approach maintenance-aware, we propose to evaluate the effect of any operation placement on maintenance triggering before making a decision. This leads to a schedule with the necessary maintenance activities triggered by the operation sequence already included. This is shown in Algorithm 1. In Line 1, the scheduler takes as input the flow shop to be scheduled, the chosen ordering of the operations *order*, and the ranking of decisions *rank*. Lines 2–6 initialise the variables used in the algorithm, i.e., an empty schedule Ω that is filled with operations by the algorithm, empty sets of schedules Ω' and Ω'' used to keep track of scheduling options, and an operation o_p to track the last operation that was inserted in the schedule. Specifically, o_p is initialised to a dummy operation for the first run where no insertions have occurred yet. In Line 7, the scheduler loops through each operation o_c in the chosen order and Line 8 finds positions to place the operation in the schedule being built with each possible option resulting in a different schedule stored in the set Ω'. For every one of these schedules, we trigger predicted maintenance in Line 10, which updates the schedules with predicted maintenance activities included. We keep track of the last regular operation placed in the schedule o_p to reduce the amount of work it takes to trigger maintenance as the schedule is already evaluated up to that operation o_p. Eventually, we pick the best option in Line 12 where the 'best' is as determined by the supplied ranking *rank*.

The steps shown in Algorithm 1 are generic and can be customised to any list scheduler of choice. However, evaluating maintenance is performed according to the steps described in Algorithm 2. For a given schedule, we first go through the operations in the schedule from the last inserted operation o_p to the current operation being inserted o_c in Line 2. For each operation, we evaluate the deterioration state in Line 3. If a maintenance activity is triggered at any point in the schedule,

[2]Start times of operations are decision variables.

Algorithm 1 Maintenance Aware List Scheduling (MALS)

1: **function** MALS(flow shop f, operation ordering *order*, ranking *rank*) ▷ **returns** schedule Ω
2: $\Omega \leftarrow <>$ ▷ empty schedule
3: $\Omega' \leftarrow \emptyset$ ▷ empty *set* of schedules
4: $\Omega'' \leftarrow \emptyset$ ▷ empty *set* of schedules
5: $o_p \leftarrow dummy$ ▷ operation initialised to dummy operation
6: **for** o_c in *order* **do**
7: $\Omega' \leftarrow generateOptions(o_c, f, \Omega)$
8: **for** $\omega \in \Omega'$ **do**
9: $\omega \leftarrow triggerMaintenance(o_c, o_p, f, \omega)$
10: $\Omega'' \leftarrow \Omega'' \cup \{\omega\}$
11: $\Omega \leftarrow selectHighestRanked(\Omega'', rank)$
12: $o_p \leftarrow o_c$
13: $\Omega'' \leftarrow \emptyset$
return Ω

Algorithm 2 Trigger Maintenance

1: **function** triggerMaintenance(current operation o_c, previous operation o_p, flow shop f, schedule Ω) ▷ **returns** schedule Ω
2: **for** $o_i \in \langle o_p, ..., o_c \rangle$ **do**
3: $\Delta \leftarrow \delta(\langle o_1, ..., o_i \rangle, \mu_k)$ ▷ predict deterioration state
4: **if** $X(\Delta)$is defined **then** ▷ deterioration triggers maint.
5: $a^c \leftarrow X(\Delta)$ ▷ insert maint. activity
6: $\Omega \leftarrow insertMaintenanceOperation(a^c, \Omega)$
7: $\Omega \leftarrow updateStartTimes(f, \Omega)$
8: $feasible \leftarrow checkFeasibility(f, \Omega)$
9: **if** $\neg$ *feasible* **then**
10: $\Omega \leftarrow repairSchedule(f, \Omega)$
return Ω

the action is then inserted and the schedule is re-evaluated in Lines 5–9. We approach this by creating an operation a^c to represent the maintenance activity and adjusting the edges in the graph such that the constraints of the original problem remain intact after the insertion of the new operation. This is illustrated in Figure 2 where we show the edges added after inserting a maintenance activity. Since we have hard timing constraints between operations, inserting a maintenance activity can lead to a previously feasible schedule becoming infeasible. In such a case, a schedule repair action is triggered to return the schedule to a feasible state in Line 11. Algorithm 2 assumes that a schedule is always repairable and below in Section VI-B2, we show what the necessary conditions are for this to be true.

B. SCHEDULE REPAIR

Flow shop schedules generally need to obey a certain ordering of operations to be valid. However, re-entrant flow shops with due dates have an additional validity criterion, which is the due date between operations. In a case where operations that

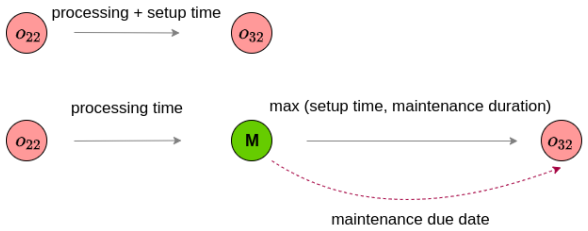

FIGURE 2: Edge update after inserting a maintenance activity. The original constraints between operations o_{22} and o_{32} are now between operation o_{22} and the maintenance activity with new edges added to connect the maintenance activity to operation o_{32} This ensures the original constraints of the problem are present and the maintenance activity is scheduled before operation o_{32}.

are not completely part of the set of input operations – such as maintenance activities – have to be scheduled, due date violations become even more likely. Since these operations are only known when schedules are evaluated, we always have the possibility that a schedule becomes infeasible as a result of these insertions. Furthermore, it is still combinatorial to decide on the repaired version of the schedule that minimises the makespan after an event that causes infeasibility occurs. We therefore need to develop a schedule repair strategy for this problem.

1) Our Strategy

Schedule repair entails reorganising a schedule to obtain a state where the schedule is valid again [41]. Since we start from a valid schedule that is rendered infeasible by inserting new operations, the infeasibility is due to a due date violation, i.e., an operation has been delayed too long after its preceding operation. Therefore, the fix is to systematically bring operations closer to their predecessors. However, it is not immediately obvious which operations need to be brought forward and how far this needs to go. As such we define a recursive strategy where we take small steps forward and reevaluate the fix until the schedule is feasible again. Additionally, moving operations around can violate the maintenance policy so after re-organisation, it is necessary to re-evaluate the schedule. This solution falls under the class of proactive-reactive dynamic scheduling [42].

As shown in Algorithm 3, every time we reorganise the operations in the schedule, we first identify three key operations, namely, the *penultimate first pass operation* from the point where the schedule was broken, the *last second pass operation* from the point where the schedule was broken, and finally the *last second pass operation* that has been included in the schedule. This is shown in Lines 4-6 where we identify these key operations and their positions in the schedule. We then move all scheduled second pass operations belonging to jobs ranging from the *last second pass* to the *ultimate first pass* in the schedule – this occurs in the remove and insert calls on Lines 13–17. This way, the schedule has been reorganised such that second pass operations from the point of failure are at least a step closer to their first pass operations. We repeat

Algorithm 3 Schedule Repair Strategy

1: **function** repairSchedule(flow shop f, position n, schedule Ω) ▷ **returns** schedule Ω
2: $feasible \leftarrow false$
3: $end \leftarrow false$
4: **while** $!feasible \wedge !end$ **do**
5: $(fp^{'}, o_{fp,k}) \leftarrow penultimateFirstPass(n, \Omega)$
6: $(ffp^{'}, o_{ffp,k}) \leftarrow ultimateFirstPass(n, \Omega)$
7: $(sp^{'}, o_{sp,k+1}) \leftarrow lastSecondPass(n, \Omega)$
8: **if** $o_{ffp} = o_{1,k}$ **then** ▷ first operation on machine
9: $end \leftarrow true$
10: $i \leftarrow sp^{'} + 1$
11: **while** $i \leq ffp^{'}$ **do**
12: $\Omega \leftarrow removeSecondPassOp(o_{i,k+1}, \Omega)$
13: $\Omega \leftarrow insertSecondPassOp(fp^{'}, o_{i,k+1}, \Omega)$
14: $fp^{'} \leftarrow fp^{'} + 1$
15: $i \leftarrow i + 1$
16: $n \leftarrow fp^{'}$
17: $\Omega \leftarrow updateStartTimes(f, \Omega)$
18: $feasible \leftarrow checkFeasibility(f, \Omega)$
19: $\Omega \leftarrow triggerMaintenance(o_{sp}, o_{1,k}, f, \Omega)$
20: **return** Ω

this process until the schedule becomes feasible[3], moving the point of failure a step backward each iteration – this is as seen on Line 18 where the point of failure is updated ahead of the next iteration. After the schedule is deemed feasible, a last step is taken to trigger maintenance again in Line 20 as re-ordering operations could have invalidated or triggered maintenance activities. This re-ordering works because due dates exist only between consecutive operations of the same job.

Figure 3 shows an example of the schedule repair process. In Step 1, the schedule is infeasible after the insertion of a maintenance activity highlighted in green. The ultimate first pass is identified as o_{42}, the penultimate first pass as o_{32} and the last second pass as o_{13}. The operations after the maintenance activity are then brought forward as can be seen in the new placement of o_{23} in Step 2. This continues in Steps 3 and 4 until the schedule is evaluated to be feasible.

It is valuable to point out that the overall algorithm proposed is flexible enough to adopt other repair strategies depending on the use case. An alternate example could be the strategy of reducing the rate of production to prevent or delay maintenance activities. A host of possible rescheduling and repair strategies are surveyed in [41].

2) Safe Maintenance Policies

A maintenance policy X maps a deterioration state of the machine to an appropriate maintenance activity. The policy in use determines when and where maintenance activities are necessary. As discussed above, inserting a maintenance activity in a schedule may make the schedule infeasible. We define

[3]It is always possible to find a feasible solution as long as the maintenance policy in use is safe. This is as shown in Theorem 1 below.

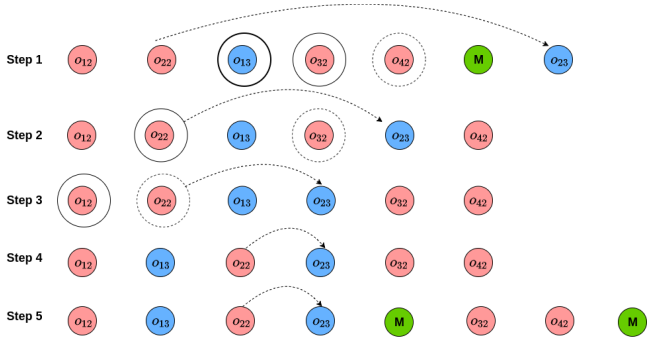

FIGURE 3: **Schedule repair strategy showing progressive steps in the algorithm. In the first step, the schedule is infeasible because of the maintenance activity (highlighted in green). From this point on, the future steps re-organise the schedule until we achieve a feasible schedule in Step 4. In Step 5, a last step is taken to trigger maintenance again as re-ordering operations could have invalidated existing or triggered new maintenance activities. Operations encircled in dotted lines are the ultimate first pass from the point of failure, the ones circled in a thin line are the penultimate first pass, and the ones circled in a thick line are the last higher pass operation.**

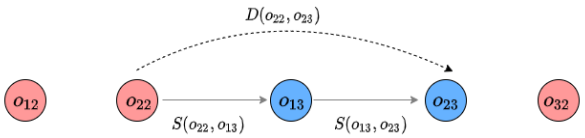

FIGURE 4: **Slack between operations o_{22} and o_{23}**

a *safe* maintenance policy as a policy that ensures that there exists at least one maintenance-aware solution to the flow shop provided there is a feasible schedule for the flow shop alone without considering maintenance activities. Since a schedule becoming infeasible after a maintenance insertion is a result of a violated due date, there should be enough room between consecutive first and second passes of the same job to fit a particular maintenance activity unless the policy is such that the maintenance activity cannot be triggered between first and second passes of the same job. Concretely, this means that the processing time of any maintenance activity a^c that can be triggered between passes of the same job o_{ik} and $o_{i(k+1)}$ should fit in the available time between them, i.e.,

$$P_c \leq D(o_{ik}, o_{i(k+1)}) - P_{ik} - S(o_{ik}, o_{i(k+1)}) \;\; \forall \, o_{ik}, o_{i(k+1)}. \tag{7}$$

Theorem 1. *Given an infeasible schedule, the schedule repair strategy defined in Algorithm 3 is always able to return it to a state of feasibility in at most $|J|$ iterations, where $|J|$ is the number of jobs in the schedule, provided that a solution exists for the problem and the maintenance policy in use is safe.*

Proof: For an insertion of a maintenance activity a^c between operations o_{ik} and $o_{i(k+1)}$ to become infeasible due to a due date violation, it means that $o_{i(k+1)}$ has been delayed too long, i.e., $\omega_{i(k+1)} - \omega_{ik} > D(o_{ik}, o_{i(k+1)})$. To avert this, the maintenance activity must be able to fit in the slack between

both operations. Bearing in mind that other operations could be placed between o_{ik} and $o_{i(k+1)}$, the slack $\Psi(o_{ik}, o_{i(k+1)})$ left between o_{ik} and $o_{i(k+1)}$ is

$$\begin{aligned}\Psi(o_{ik}, o_{i(k+1)}) = D(o_{ik}, o_{i(k+1)}) - P_{ik} - \\ \max((S(o_{ik}, o_a) + P_a + ... + S(o_a, o_b) + P_b), \\ S(o_{ik}, o_{i(k+1)})),\end{aligned} \tag{8}$$

where operations $o_a, ..., o_b$ represent operations possibly placed between o_{ik} and $o_{i(k+1)}$. Figure 4 shows an example where the slack between operations o_{22} and o_{23} is

$$\begin{aligned}\Psi(o_{22}, o_{23}) = D(o_{22}, o_{23}) - P_{22} - \\ \max(S(o_{22}, o_{13}) - P_{13} - S(o_{13}, o_{23}), S(o_{22}, o_{23})).\end{aligned} \tag{9}$$

The repair algorithm progressively brings operations closer to their direct predecessors by at least one step per iteration. In the last possible iteration of the schedule repair, each operation $o_{i(k+1)}$ follows its direct predecessor o_{ik}. It follows that this occurs in at most $|J|$ iterations of the schedule repair as the re-entrant machine can only have $|J|$ higher pass operations to be re-ordered. At this point, Equation (8) becomes

$$\Psi(o_{ik}, o_{i(k+1)}) = D(o_{ik}, o_{i(k+1)}) - P_{ik} - S(o_{ik}, o_{i(k+1)}). \tag{10}$$

For this to be infeasible, it means that a^c cannot fit in $\Psi(o_{ik}, o_{i(k+1)})$, i.e., $P_c > D(o_{ik}, o_{i(k+1)}) - P_{ik} - S(o_{ik}, o_{i(k+1)})$, which violates the rules of a safe maintenance policy shown in Equation (7). ■ ■

VII. EXPERIMENTAL RESULTS

This section evaluates the empirical performance of the three solution approaches we propose. We apply the heuristic approach as an add-on to three existing list schedulers in the literature to evaluate the applicability of this approach to list-scheduling. We compare our heuristics against the two exact approaches (integer and constraint programming) to evaluate their accuracy and scalability.

A. EXPERIMENTAL SETUP

All experiments are performed on a 16-core 1.9GHz AMD machine running Ubuntu 20.04 with 32GB RAM. Algorithms are implemented in C++ and the MIP and CP models are solved by CPLEX version 22.1 and CP Optimizer version 22.1, respectively. The exact approaches are all given a 30 minute timeout.

We generate benchmarks according to the types of jobs typically presented in our industrial use case as described in Table 1. We generate benchmarks with patterned arrivals of job types such that jobs of a type appear in repeated blocks, e.g., a set of 50 jobs can be made of 20 type 1 jobs followed by 10 type 2 jobs and then 20 type 3 jobs. We randomise the length of the blocks and number of times these blocks repeat to mimic arrival patterns of jobs in practice. We generate 50 instances for each job size in {5,10,50,100,150,200,300,500,1000}.

Our heuristic approach is implemented as an extension to three schedulers from the literature – Bounded Heuristic Constraint Scheduler (BHCS) [39], As Soon As Possible (ASAP) Scheduler, and Modified Nawaz-Enscore-Ham (MNEH) Heuristic [43]. BHCS is a list scheduler developed specifically for our use case, while the ASAP scheduler is also a list scheduler that uses the same ordering requirements as BHCS but places operations as soon as possible (ASAP). MNEH is a modification of the popular NEH heuristic [44], [45] that is suitable for re-entrancy. Maintenance-incorporated versions of these schedulers are referred to as MIBHCS, MIASAP, and MINEH respectively where the MI prefix refers to ''maintenance incorporated''. In each of these experiments, we tune the heuristic approach to include a maintenance activity if a deterioration threshold is crossed or if 90%[4] of the upper bound of a threshold that affects the quality of an operation further down the line is crossed. Since we insert maintenance between two operations, we always have complete information about the next operation. We can also reliably infer what operations are further down the line for the entire planning window based on which operations have already been scheduled.

In the basic schedulers – BHCS, ASAP, and MNEH – maintenance is reactive and interrupts the schedule during production runs. We simulate the behaviour of reactive maintenance in these schedulers by evaluating the completed schedules they produce for maintenance and compare these with versions of the scheduler that incorporate our proactive maintenance heuristic.

B. PERFORMANCE EVALUATION

Figure 5 compares the makespan of the schedules produced by MIBHCS, MIASAP, and MINEH to the makespan of schedules produced by BHCS, ASAP, and MNEH respectively. We also compare the exact solutions CP and MIP with the best solutions provided by MIBHCS, MIASAP, and MINEH. MNEH has the least performance improvement due to it not being a pure list scheduler. With MNEH, only relative positions of operations are decided in each iteration and there is no partial sequence that is guaranteed to remain the same from one iteration to the next; as such the evaluation of the deterioration of a machine loses some meaning from one iteration to the next since sequences change at each iteration. The exact approaches – CP and MIP – should ideally always be better than all of the heuristic approaches but they are sometimes worse because they do not always solve till optimality within the time out.

Figure 6 shows the distribution of the time spent on maintenance. We see that with maintenance-included versions, we spend up to 70% less time on maintenance. This is because considering deterioration allows us to perform maintenance before machines deteriorate to a state where we have to pay larger maintenance costs. The difference is also this significant because there is up to one order of magnitude difference between the durations of different maintenance activities for this use case (see Table 1d). This difference translates to shorter makespans for the schedulers.

In both Figures 5 and 6, there are instances where the

[4]This value can be tuned. We chose 90% after performing a parameter sweep that showed this value performed best.

Type	$P(o_{i1})$	$P(o_{i2})$	$P(o_{i3})$	$P(o_{i4})$	$D(o_{i1}, o_{i2})$	$D(o_{i2}, o_{i3})$	$D(o_{i3}, o_{i4})$
0	0.25	0.30	0.30	0.21	0.85	12.30	1.00
1	0.35	0.42	0.42	0.30	0.95	12.42	1.12
2	0.50	0.59	0.59	0.42	1.10	12.59	1.29
3	0.70	0.84	0.84	0.60	1.30	12.84	1.54
4	0.99	1.19	1.19	0.85	1.59	13.19	1.89

(a) Job processing times and due dates

Machine	Setup Time
μ_1	0.20
μ_2	0.05
μ_3	1.00

(b) Machine setup times

Path	Travelling Time
μ_1 to μ_1	0.60
μ_2 to μ_2	10.00
μ_2 to μ_3	0.70

(c) Job travelling times

Activity Class	Duration	Deterioration States
1	0.5s	10 – 15
2	10s	15 – 30
3	20s	30 – ∞

(d) Maintenance policy

TABLE 1: Properties of jobs in use case. All timings are in seconds and job travelling times are treated as setup times between operations of the same job.

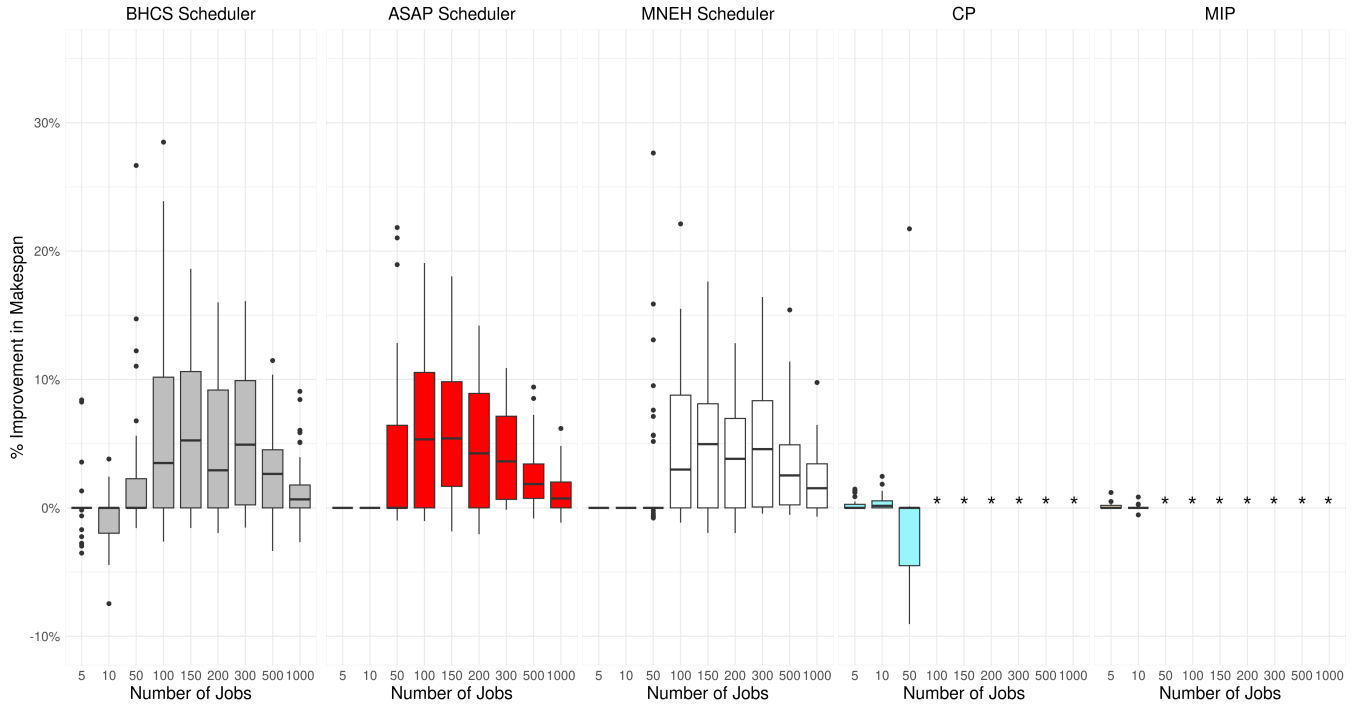

FIGURE 5: Makespan improvement of maintenance-included versions over base versions
Instances where the solver timed out without providing any solution are marked with *.

Jobs	Optimality gap (%)		Time to find first solution (s)		% of instances solved	
	CP	MIP	CP	MIP	CP	MIP
5	0.60	0.00	0.82	2.44	100	100
10	1.10	4.60	2.38	419.80	100	26
50	166.01	-	84.95	-	100	0

TABLE 2: Performance of CP and MIP solutions

heuristic approach worsens the results particularly for smaller job sets. The instances that are worsened by the heuristic are a result of (i) scenarios where the heuristic maintenance trigger is too conservative and performs maintenance even though the job set could be completed without it, and (ii) scenarios where the list scheduler picks a sequence that triggers shorter maintenance activities.

Neither of the exact solutions are able to scale to provide solutions for larger job sets within the 30 minute time out – this accounts for the missing columns in Figures 5 and 6. In Table 2 we show the performance of the CP and MIP solutions. We see that the CP model is able to solve more instances than

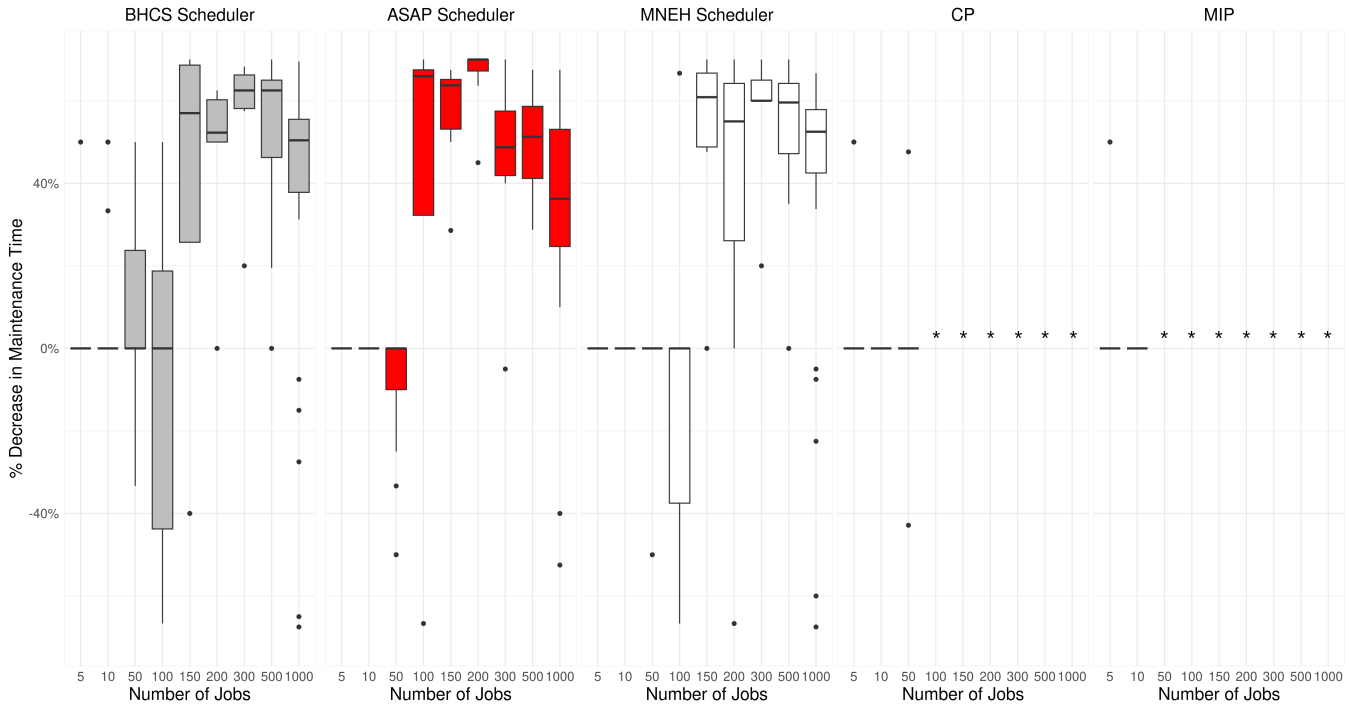

FIGURE 6: Duration of maintenance activities
Instances where the solver timed out without providing any solution are marked with *.

the MIP model but for the instances where the MIP model is able to provide solutions, the optimality gap is smaller.

The runtime increases with the number of jobs as expected and Table 3 shows the average runtime over the job size of the different schedulers compared in this evaluation. The exact approaches are given a 30 minute timeout and in bold are the solutions with the worst run times for a job size. Instances where no solution was provided by a method before time out are left unfilled and are the worst for that job size. The heuristic solutions are able to provide solutions in runtimes below 350*ms* for job sizes up to 500. Above that, the runtime grows to 1800*ms*. The biggest time sink for the heuristic solutions is how often the maintenance evaluation and consequently schedule repair is triggered[5]. This is based on the operation of the base scheduler itself. MNEH evaluates whole sequences while ASAP and BHCS evaluate partial sequences at every decision point thus triggering maintenance evaluations more often, leading to higher runtimes.

In summary, we find that the heuristic approach is scalable and can produce competitive results compared to exact solvers even for small instance sizes. In general, we also find that apart from improving the actual goal of reduced makespan, integrated production and maintenance planning can also reduce the total time spent on maintenance which can result in reduced costs in some cases.

Jobs	CP	MIASAP	MIBHCS	MINEH	MIP
5	**505.16**	0.00	0.00	0.00	5.81
10	1083.43	0.01	0.01	0.00	**1522.37**
50	**1611.23**	0.67	0.56	0.07	-
100	-	2.72	1.94	0.32	-
150	-	7.06	4.84	1.12	-
200	-	14.19	8.65	2.23	-
300	-	42.10	23.11	5.94	-
500	-	164.71	84.80	30.19	-
1000	-	1756.40	854.44	303.66	-

TABLE 3: Average runtime of solution methods (s)

VIII. CONCLUSION

Efficient maintenance scheduling is important for sustained productivity of industrial processes. This paper studied the problem of sequence- and time-dependent maintenance and presented three solution methods namely, mixed integer programming, constraint programming and a heuristic solution. As the problem is motivated by an industrial use case, we have evaluated all the methods on jobs in this case. We show that list scheduling heuristics can be extended to include proactive maintenance with significant performance gains over reactive approaches.

This paper considers maintenance activities that are on the same time scale as the jobs themselves. An interesting future direction is to include longer-term maintenance planning in the scope and to investigate the combined problem of production and maintenance planning over multiple time scales.

Additionally, we solve the problem from a predictive maintenance perspective, i.e., where maintenance actions are carried out based on the health status of machines. However, this requires knowledge of how machines deteriorate and this information is not always available. Many other papers consider

[5]Runtimes of ASAP, BHCS, and MNEH are similar.

a preventive maintenance perspective where the challenge is either scheduling around a set maintenance schedule or determining what the maintenance schedule itself should be. While we know that preventive maintenance runs the risk of either maintaining machine too little or too often compared to the needs-based approach of predictive maintenance, and both preventive and predictive maintenance have been shown to outperform reactive maintenance approaches, it is still interesting to compare both approaches and determine what problem properties make it necessary to use one or the other. This is because even when complete information on the health status of machines is available, the gains made by integrating them in the decision making process may not necessarily be worth the increased runtime.

ACKNOWLEDGEMENTS

The authors thank Hadi Ara, Joost van Pinxten and Joan Marcè i Igual for their support.

REFERENCES

[1] Jichuan Kang and C. Guedes Soares. An opportunistic maintenance policy for offshore wind farms. *Ocean Engineering*, 216:108075, 2020.

[2] Zhengru Ren, Amrit Shankar Verma, Ye Li, Julie JE Teuwen, and Zhiyu Jiang. Offshore wind turbine operations and maintenance: A state-of-the-art review. *Renewable and Sustainable Energy Reviews*, 144:110886, 2021.

[3] Sirikarn Chansombat, Pupong Pongcharoen, and Christian Hicks. A mixed-integer linear programming model for integrated production and preventive maintenance scheduling in the capital goods industry. *International Journal of Production Research*, 57(1):61–82, 2019.

[4] Francisco N Avilés, Renato Maynard Etchepare, Maichel M Aguayo, and Mario Valenzuela. A mixed-integer programming model for an integrated production planning problem with preventive maintenance in the pulp and paper industry. *Engineering Optimization*, 55(8):1352–1369, 2023.

[5] Hassan Gharoun, Mahdi Hamid, and S Ali Torabi. An integrated approach to joint production planning and reliability-based multi-level preventive maintenance scheduling optimisation for a deteriorating system considering due-date satisfaction. *International Journal of Systems Science: Operations & Logistics*, 9(4):489–511, 2022.

[6] Rolando Jacyr Kurscheidt Netto, Eduardo de Freitas Rocha Loures, Eduardo Alves Portela Santos, and Cleiton Ferreira dos Santos. Joint industrial preventive maintenance and production scheduling: A systematic literature review. In *International Conference on Flexible Automation and Intelligent Manufacturing*, pages 614–621. Springer, 2023.

[7] Nan Zhang, Kaiquan Cai, Yingjun Deng, and Jun Zhang. Determining the optimal production–maintenance policy of a parallel production system with stochastically interacted yield and deterioration. *Reliability Engineering & System Safety*, 237:109342, 2023.

[8] Alexander Valet, Thomas Altenmüller, Bernd Waschneck, Marvin Carl May, Andreas Kuhnle, and Gisela Lanza. Opportunistic maintenance scheduling with deep reinforcement learning. *Journal of Manufacturing Systems*, 64:518–534, 2022.

[9] Stanisław Gawiejnowicz. A review of four decades of time-dependent scheduling: Main results, new topics, and open problems. *Journal of Scheduling*, 23(1):3–47, 2020.

[10] Suh-Jenq Yang. Single-machine scheduling problems with both start-time dependent learning and position dependent aging effects under deteriorating maintenance consideration. *Applied Mathematics and Computation*, 217(7):3321–3329, 2010.

[11] Weiguo Liu, Xuyin Wang, Lu Li, and Peizhen Zhao. A maintenance activity scheduling with time-and-position dependent deteriorating effects. *Mathematical Biosciences and Engineering*, 19(11):11756–11767, 2022.

[12] Baruch Mor and Gur Mosheiov. Scheduling a maintenance activity and due-window assignment based on common flow allowance. *International Journal of Production Economics*, 135(1):222–230, 2012.

[13] Eghonghon-aye Eigbe, Bart De Schutter, Mitra Nasri, and Neil Yorke-Smith. Predictive maintenance scheduling in twice re-entrant flow shops with relative due dates. In *15th Workshop on Scheduling and Planning Applications (SPARK) at the 32nd International Conference on Automated Planning and Scheduling (ICAPS)*, 2022.

[14] Yu-Chuan Su, Fan-Tien Cheng, Min-Hsiung Hung, and Hsien-Cheng Huang. Intelligent prognostics system design and implementation. *IEEE Transactions on Semiconductor Manufacturing*, 19(2):195–207, 2006.

[15] Gian Antonio Susto, Andrea Schirru, Simone Pampuri, Seán McLoone, and Alessandro Beghi. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3):812–820, 2014.

[16] Maxence Delorme, Manuel Iori, and Nilson FM Mendes. Solution methods for scheduling problems with sequence-dependent deterioration and maintenance events. *European Journal of Operational Research*, 295(3):823–837, 2021.

[17] Suh-Jenq Yang. Parallel machines scheduling with simultaneous considerations of position-dependent deterioration effects and maintenance activities. *Journal of the Chinese Institute of Industrial Engineers*, 28(4):270–280, 2011.

[18] Lei Jin, Xianyu Yu, and Zhuojia Dong. Single-machine scheduling with piece-rate maintenance, interval constrained processing times and rejection penalties. In *2018 3rd Joint International Information Technology, Mechanical and Electronic Engineering Conference (JIMEC 2018)*, pages 32–37. Atlantis Press, 2018.

[19] Hui Zhu, Min Li, Zhangjin Zhou, and Yun You. Due-window assignment and scheduling with general position-dependent processing times involving a deteriorating and compressible maintenance activity. *International Journal of Production Research*, 54(12):3475–3490, 2016.

[20] Alex J Ruiz-Torres, Giuseppe Paletta, and Eduardo Pérez. Parallel machine scheduling to minimize the makespan with sequence dependent deteriorating effects. *Computers & Operations Research*, 40(8):2051–2061, 2013.

[21] Alex J Ruiz-Torres, Giuseppe Paletta, and Rym M'Hallah. Makespan minimisation with sequence-dependent machine deterioration and maintenance events. *International Journal of Production Research*, 55(2):462–479, 2017.

[22] Vívian L Aguiar Santos and José Elias C Arroyo. Iterated greedy with random variable neighborhood descent for scheduling jobs on parallel machines with deterioration effect. *Electronic Notes in Discrete Mathematics*, 58:55–62, 2017.

[23] Junwen Ding, Liji Shen, Zhipeng Lü, and Bo Peng. Parallel machine scheduling with completion-time-based criteria and sequence-dependent deterioration. *Computers & Operations Research*, 103:35–45, 2019.

[24] AS Xanthopoulos, Athanasios Kiatipis, Dimitris E Koulouriotis, and Sepp Stieger. Reinforcement learning-based and parametric production-maintenance control policies for a deteriorating manufacturing system. *IEEE Access*, 6:576–588, 2017.

[25] Yusheng Wang, Ehsan Elahi, and Lei Xu. Selective maintenance optimization modelling for multi-state deterioration systems considering imperfect maintenance. *IEEE Access*, 7:62759–62768, 2019.

[26] Raida Jarrar Abumaizar and Joseph A Svestka. Rescheduling job shops under random disruptions. *International Journal of Production Research*, 35(7):2065–2082, 1997.

[27] Erhan Kutanoglu and İhsan Sabuncuoglu. Routing-based reactive scheduling policies for machine failures in dynamic job shops. *International Journal of Production Research*, 39(14):3141–3158, 2001.

[28] Weng Tat Chan and Tan Heng Wee. A multi-heuristic GA for schedule repair in precast plant production. In *13th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 236–245, 2003.

[29] Ali Allahverdi. Two-machine proportionate flowshop scheduling with breakdowns to minimize maximum lateness. *Computers & Operations Research*, 23(10):909–916, 1996.

[30] Pierpaolo Caricato and Antonio Grieco. An online approach to dynamic rescheduling for production planning applications. *International Journal of Production Research*, 46(16):4597–4617, 2008.

[31] Ketrina Katragjini, Eva Vallada, and Rubén Ruiz. Rescheduling flowshops under simultaneous disruptions. In *6th IEEE International Conference on Industrial Engineering and Systems Management (IESM)*, pages 84–91, 2015.

[32] Ronald Lewis Graham, Eugene Leighton Lawler, Jan Karel Lenstra, and Alexander Hendrik George Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[33] Michael Yingmeng Wang, Suresh P Sethi, and Steef L van de Velde. Minimizing makespan in a class of reentrant shops. *Operations Research*, 45(5):702–712, 1997.

[34] Hamilton Emmons, George Vairaktarakis, Hamilton Emmons, and George

Vairaktarakis. Reentrant flow shops. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*, pages 269–289, 2013.

[35] Faicel Hnaien, Farouk Yalaoui, A Mhadhbi, and M Nourelfath. A mixed-integer programming model for integrated production and maintenance. *IFAC-PapersOnLine*, 49(12):556–561, 2016.

[36] H Paul Williams. *Model Building in Mathematical Programming*. John Wiley & Sons, 2013.

[37] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG. *Constraints*, 23:210–250, 2018.

[38] Umar Waqas, Marc Geilen, Jack Kandelaars, Lou Somers, Twan Basten, Sander Stuijk, Patrick Vestjens, and Henk Corporaal. A re-entrant flowshop heuristic for online scheduling of the paper path in a large scale printer. In *18th IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 573–578, 2015.

[39] Joost van Pinxten, Umar Waqas, Marc Geilen, Twan Basten, and Lou Somers. Online scheduling of 2-re-entrant flexible manufacturing systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s):1–20, 2017.

[40] Roel van der Tempel, Joost van Pinxten, Marc Geilen, and Umar Waqas. A heuristic for variable re-entrant scheduling problems. In *21st IEEE Euromicro Conference on Digital System Design (DSD)*, pages 336–341, 2018.

[41] Guilherme E Vieira, Jeffrey W Herrmann, and Edward Lin. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62, 2003.

[42] Djamila Ouelhadj and Sanja Petrovic. A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417–431, 2009.

[43] BongJoo Jeong and Yeong-Dae Kim. Minimizing total tardiness in a two-machine re-entrant flowshop with sequence-dependent setup times. *Computers & Operations Research*, 47:72–80, 2014.

[44] Muhammad Nawaz, E Emory Enscore Jr, and Inyong Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[45] Weibo Liu, Yan Jin, and Mark Price. A new improved NEH heuristic for permutation flowshop scheduling problems. *International Journal of Production Economics*, 193:21–30, 2017.