Delft University of Technology

Delft Center for Systems and Control

Technical report 22-018

Predictive maintenance scheduling in twice re-entrant flow shops with relative due dates*

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

If you want to cite this report, please use the following reference instead:
E.-A. Eigbe, B. De Schutter, M. Nasri, and N. Yorke-Smith, "Predictive maintenance scheduling in twice re-entrant flow shops with relative due dates," *Proceedings of the 15th Scheduling and Planning Applications woRKshop (SPARK) — ICAPS 2022 Workshop*, (Virtual), 8 pp., June 2022. Paper 3.

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://pub.bartdeschutter.org/abs/22_018.html

Predictive Maintenance Scheduling in Twice Re-entrant Flow Shops with Relative Due Dates

Eghonghon-aye Eigbe[1], Bart De Schutter[1], Mitra Nasri[2], Neil Yorke-Smith[1]

[1]Delft University of Technology, [2]Eindhoven University of Technology
e.eigbe@tudelft.nl*, b.deschutter@tudelft.nl, m.nasri@tue.nl, n.yorke-smith@tudelft.nl

Abstract

Predictive maintenance is important for overall equipment effectiveness in a production process. This paper studies scheduling predictive maintenance in flow shops with re-entrancy and due dates. We propose a method to integrate maintenance operations on the re-entrant machine in any schedule produced by a list scheduling algorithm, provided we have knowledge of how the health status of the machine evolves. Additionally, we introduce a schedule repair strategy for instances where schedules become infeasible as a result of due date violations resulting from the insertion of maintenance operations. Our generic approach is evaluated on an industrial use case. The results demonstrate that integrated maintenance and production planning increases the productivity of the process while remaining applicable for online scheduling.

Introduction

The literature classifies three kinds of maintenance (Swanson 2001), (i) preventive where maintenance actions occur according to certain patterns, (ii) predictive where maintenance actions are carried out based on the future health status of machines (Phogat and Gupta 2017), and (iii) reactive where maintenance actions are only carried out upon machine failure. Of the three, predictive maintenance shows the most promise as it avoids the extreme case of waiting for machine breakdown and also avoids the other extreme case of maintaining too often and incurring excessive costs. However, predictive maintenance planning requires us to know the health status of machines and this is often dependent on patterns of machine use. Thus, predictive maintenance planning creates an integrated production and maintenance scheduling problem.

In many modern manufacturing systems, we have the added complexity that the same set of machines can cater to a variety of products, which can have different effects on the health status of these machines (El-Maraghy 2005). Additionally, since many of these systems receive manufacturing requests on the fly, scheduling decisions need to be made in an online fashion.

Existing research has proposed solutions for scheduling maintenance in manufacturing systems (Yang, Djurdjanovic, and Ni 2007; Herr, Nicod, and Varnier 2014). Artificial intelligence techniques have also been used both to determine health status of machines (Su et al. 2006; Susto et al. 2014) and to create schedules themselves (Yang, Djurdjanovic, and Ni 2007; Ladj, Tayeb, and Varnier 2018). However, predictive maintenance scheduling in flow shops with re-entrancy and due dates is, to the best of our knowledge, neglected.

This paper considers scheduling predictive maintenance in flow shops with re-entrancy and due dates. Motivated by the opportunity provided by list scheduling to quickly produce high quality solutions using simple heuristics (Carpov et al. 2012; Luo et al. 2013; van der Tempel et al. 2018), we propose an integrated production and maintenance planning method to integrate maintenance operations on the re-entrant machine in any schedule produced by a list scheduling algorithm, provided we can predict how the health status of the machine evolves. Additionally, we design a schedule repair method for instances where schedules are broken by the insertion of maintenance operations. The algorithmic approach proposed in this paper is generic and we demonstrate its performance on a concrete use case with evaluations based on the makespan and overall equipment effectiveness (OEE) (Hansen 2001).

The contributions of this paper to the literature are: an extension of list scheduling to handle predictive maintenance, a schedule repair algorithm for re-entrant flowshops and an evaluation of the above on an existing industrial use case.

We first discuss related work, then provide the background and problem definition. Following, we lay out our solution approach and analyse its complexity. We also examine an industrial use case and show experimental results from which we conclude that list scheduling heuristics can be extended to include maintenance operations on re-entrant flow shops with improvements in resulting makespans.

Related Work

Scheduling manufacturing systems is an important field that has received attention for many years (Basnet and Mize 1994; Ruml, Do, and Fromherz 2005). Maintenance scheduling has been studied extensively in the

operations research literature in particular, with different goals such as reducing maintenance costs, minimising makespan, and even total tardiness (Phogat and Gupta 2017). Typically, these manufacturing systems are modelled as job or flow shops with many variants depending on the specifics of the area of application or type of problem considered (Buzacott and Shanthikumar 1980). The dynamic relationship between maintenance scheduling and production scheduling has been investigated from multiple angles. Some work has focused on accurately determining machine health status (Su et al. 2006; Susto et al. 2014), while others have focused on generating schedules (Cui et al. 2018). Scheduling solutions that use search optimisation have been proposed by Yang, Djurdjanovic, and Ni (2007) and Ladj, Tayeb, and Varnier (2018) while other work has considered mixed integer linear programming formulations of this problem (Varnier and Zerhouni 2012). Solutions like these typically consider an offline problem where there is sufficient time budget to carry out these evaluations; however, they become a bottleneck when applied to online scheduling problems.

Scheduling flow shops with re-entrancy is a common problem in semiconductor manufacturing (Kaihara, Kurose, and Fujii 2012) and production printing (Waqas 2017) and has been considered in many papers (e.g., (Choi and Kim 2008; Lin, Lee, and Ho 2013)). However, few works have considered maintenance scheduling for this scenario. One example is Kaihara et al. (2010) where Lagrangian decomposition coordination was used to tackle maintenance scheduling. This scenario is such that maintenance is known in advance and typically takes minutes to produce a schedule which makes it unsuitable for online scheduling.

Many common online scheduling algorithms use *list scheduling* heuristics such as priority scheduling, earliest deadline first scheduling, and Johnson's algorithm (Ruiz and Maroto 2005). Heuristic list schedulers have also been developed for the use case scenario on which our work is demonstrated (van Pinxten et al. 2017; van der Tempel et al. 2018). However, the presence of due dates in our problem, combined with the fact that typical list scheduling heuristics do not consider many candidate solutions make it necessary to address schedules that become infeasible. Indeed, re-organising or repairing a changed schedule has been studied with various heuristics like left and right shift (Kutanoglu and Sabuncuoglu 2001). Chan and Wee (2003) combine 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. These heuristics however, do not fully apply to a scenario with due dates and a required job completion order. In the context of flow shops, an example of schedule repair algorithms can be found in Allahverdi (1996) which considers re-scheduling in a two-machine flow shop where schedules are disrupted by machine breakdowns. Additionally, Caricato and Grieco (2008) consider re-scheduling due to inserting new jobs in already planned schedules. These cases all consider unexpected interruptions and do not have the combinations of re-entrancy and due dates.

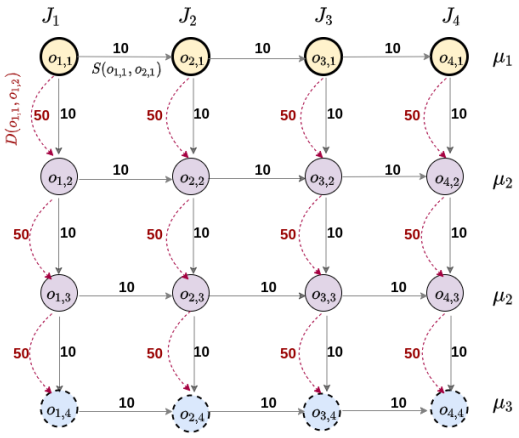

Figure 1: Sample re-entrant flow shop where the operations are represented by circles. Column wise, we have operations of the same job and row-wise, we have operations on the same machine with one of these being the re-entrant machine that appears on rows 2 and 3. Operations with the same colour or boundary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

Problem Definition

We consider the *maintenance-aware re-entrant flow shop with setup times and relative due dates*. In this problem, the due dates and setup times form a system of difference constraints and can be represented as a constraint graph (Figure 1). The maintenance-aware flow shop model can be represented as the tuple $(M, J, O, P, S, D, \delta, X, O^M)$ where $M = \{\mu_1, ..., \mu_m\}$ is the set of machines and $J = \langle J_1, ..., J_n \rangle$ is the sequence of jobs. The set O represents the set of operations for every job $J_i \in J$ where each operation $o_{i,j}$ has a processing time $P(o_{i,j})$. Moreover, $S : O \times O \to \mathbb{R}_{\geq 0}$ refers to setup times, which represents the required delay between the completion of an operation 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 that is needed between operations. Operations of the same job also have due dates 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. Due dates model the fact that operations of a job can often not be delayed indefinitely due to physical constraints in the plant like buffer size. In a situation where such a constraint does not apply, we simply set the due dates to infinity. Also part of the model is $\delta : O^{\mathbb{N}} \times M \to \mathbb{R}_{\geq 0}$, a function that maps a scheduled sequence of operations on a machine to a deterioration state. Finally, we have a maintenance policy X that

maps deterioration states to maintenance actions where maintenance actions are modelled as operations from a set of possible maintenance actions O^M.

We assume a setup of re-entrancy such that the sequence of machines for each job is $\langle \mu_1, ..., \mu_k, \mu_k, ..., \mu_m \rangle$, i.e., there is one re-entrant machine that all jobs go through twice. We assume that jobs are not allowed to overtake each other, that the required completion order of jobs is the same as the index of the jobs, and all setup times and due date constraints are hard constraints that must be obeyed. This setup means that the only scheduling freedom is in the sequence of operations on the re-entrant machine, i.e., first and second operations (referred to as passes through the machine) of the same jobs do not necessarily have to follow each other on this machine. We limit our maintenance planning to maintaining the re-entrant machine in the sequence $\langle \mu_1, ..., \mu_k, \mu_k, ..., \mu_m \rangle$, i.e., μ_k. For simplicity of notations, a reference of deterioration $\delta(O^{\mathbb{N}})$ refers to the deterioration of re-entrant machine μ_k. A sample problem is shown in Figure 1. The solution to the problem is a schedule Ω , i.e., a sequence of both production and maintenance operations where each operation is assigned a start time, i.e., $\Omega(o_{i,j})$ represents the start time of operation $o_{i,j}$ on Ω.

Solution Approach

As explained in the introduction, we take a list scheduling approach and extend it to integrate maintenance operations in the schedule. This integration is motivated by the predictive approach we take to maintenance where maintenance actions are based on the use patterns of the machines and as such, we do not know what maintenance is needed until a sequence is selected.

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. 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 actions 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,

Algorithm 1 MALS(Maintenance Aware List Scheduling)

```
1:  function MALS(flow shop f, operation ordering
    order, ranking rank)                ▷ returns schedule Ω
2:      Ω ←<>                                ▷ empty schedule
3:      Ω' ← ∅                      ▷ empty set of schedules
4:      Ω'' ← ∅                     ▷ empty set of schedules
5:      o_p ← dummy
6:          ▷ operation initialised to dummy operation
7:      for o_c in order do
8:          Ω' ← generateOptions(o_c, f, Ω)
9:          for ω ∈ Ω' do
10:             ω ← triggerMaintenance(o_c, o_p, f, ω)
11:             Ω'' ← Ω'' ∪ {ω}
12:         Ω ← selectHighestRanked(Ω'', rank)
13:         o_p ← o_c
14:         Ω'' ← ∅
        return Ω
```

we trigger predicted maintenance in Line 10, which updates the schedules with predicted maintenance actions included. We keep track of the last 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*.

It is valuable to note that most of 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 action is triggered at any point in the schedule, the action is then inserted and the schedule re-evaluated in Lines 5–9. We approach this by creating an operation o^m to represent the maintenance operation[1] and adjusting the edges in the graph such that the constraints of the original problem remain intact after the insertion of the new operation. Since we have hard timing constraints between operations, inserting a maintenance action 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 we show in Theorem 1 what the necessary conditions are for this to be true.

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

[1]Maintenance operations are written with superscript m to differentiate them from production operations.

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 ∈ ⟨o_p, ..., o_c⟩ do
3:         Δ ← δ(⟨o_1, ..., o_i⟩)
4:                                  ▷ predict deterioration state
5:         if X(Δ) ↓ then
6:                                  ▷ deterioration triggers maint.
7:             o^m ← X(Δ)
8:                                  ▷ Insert maint. operation
9:             Ω ← insertMaintenanceOperation(o^m, Ω)
10:            Ω ← updateStartTimes(f, Ω)
11:            feasible ← checkFeasibility(f, Ω)
12:            if !feasible then
13:                Ω ← repairSchedule(f, Ω)
       return Ω
```

criterion which is the due date between operations. In a case where operations that are not completely part of the set of input operations 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 that minimises makespan after an infeasible event occurs. We therefore need to develop a strategy.

Schedule repair entails us reorganising a schedule to obtain a state where the schedule is valid again. 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 because any re-organisation of the schedule changes the sequence and as such could lead to a different set of maintenance actions which may or may not be feasible to include. Therefore, we define a recursive strategy where we take small steps forward and reevaluate the fix until the schedule is feasible again.

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 this process until the schedule becomes feasible, 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 actions.

Figure 2 shows an example of the schedule repair process. In step 1, the schedule is infeasible after the insertion of a maintenance action highlighted in green. The ultimate first pass is identified as $o_{4,2}$, the penultimate first pass as $o_{3,2}$ and the last second pass as $o_{1,3}$. From this point, the operations after the maintenance action are brought forward as can be seen in the new placement of $o_{2,3}$ in step 2. This continues in steps 3 and 4 until the schedule is evaluated to be feasible. This re-ordering works because due dates exist only between consecutive operations of the same job and moving second pass operations backwards does not violate any other due dates in the schedule.

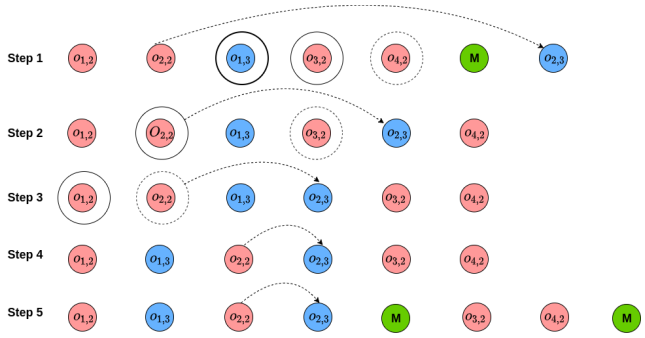

Figure 2: Schedule repair strategy showing progressive steps in the algorithm. Red and blue circles represent the first and second operations on the re-entrant machine μ_2 in Figure 1. In the first step, the schedule is infeasible because of the maintenance action (highlighted in green). From this point 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 actions. Operations circled in dotted lines are the ultimate first pass from the point of failure, circled in a thin line are the penultimate first pass and circled in a thick line are the last higher pass operation.

Safe Maintenance Policies A maintenance policy maps a deterioration state of the machine to an appropriate maintenance action. The policy in use determines when and where maintenance actions are necessary. As discussed above, inserting a maintenance operation in a schedule has the tendency to make the schedule become infeasible. We define 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 actions.

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 action unless the policy is such that that action cannot be triggered while a pending second pass exists. Concretely, this means that the processing time of any maintenance action o^m that can be triggered between passes of the same job $o_{i,k}$ and $o_{i,k+1}$ should fit in the available time between them, i.e.,

$$P(o^m) \leq D(o_{i,k}, o_{i,k+1}) - P(o_{i,k}) - S(o_{i,k}, o_{i,k+1}) \quad \forall\ o_{i,k}, o_{i,k+1}. \tag{1}$$

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 action o^m between operations $o_{i,k}$ 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., $\Omega(o_{i,k+1}) - \Omega(o_{i,k}) > D(o_{i,k}, o_{i,k+1})$. To avert this, the maintenance operation must be able to fit in the slack between both operations. Bearing in mind that other operations could be placed between $o_{i,k}$ and $o_{i,k+1}$, the slack $\Psi(o_{i,k}, o_{i,k+1})$ left between $o_{i,k}$ and $o_{i,k+1}$ is

$$\Psi(o_{i,k}, o_{i,k+1}) = D(o_{i,k}, o_{i,k+1}) - P(o_{i,k}) - \max((S(o_x, o_a) + P(o_a) + S(o_a, o_b) + P(o_b) - ...), S(o_{i,k}, o_{i,k+1})), \tag{2}$$

where o_a and o_b represent operations possibly placed between $o_{i,k}$ and $o_{i,k+1}$. 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 $o_{i,k}$. 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 (2) becomes

$$\Psi(o_{i,k}, o_{i,k+1}) = D(o_{i,k}, o_{i,k+1}) - P(o_{i,k}) - S(o_{i,k}, o_{i,k+1}). \tag{3}$$

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

Industrial Use Case

Our algorithmic approach described above is generic and in this section, we demonstrate it on a concrete use case – an in-use industrial printer. The setup is such that it has 3 machines and jobs refer to one sheet to be printed. The sequence of machines is $\langle \mu_1 \mu_2 \mu_2 \mu_3 \rangle$. The use case accepts different types of jobs each with different effects on the health status of the machines. As such, the maintenance actions triggered are dependent on the mix of job types presented to the flow shop. Additionally, the maintenance policy is such that the more deteriorated a machine is, the longer it may take to perform a maintenance action to restore it. There are three types of maintenance actions with up to one order of magnitude difference between the duration of each type as shown in Table 1d. The relationship between how long a maintenance action takes and the deterioration state is described as a step function with the same maintenance action being valid for a range of deterioration states. There is some flexibility in when we schedule maintenance operations as machines only have to be maintained right before they are used again to ensure the quality of the job. As such, a maintenance action is triggered only if the predicted deterioration state affects the quality of a job that is yet to be scheduled. However, not maintaining a machine at the right time drastically affects the quality of the print

Algorithm 3 Schedule Repair Strategy

```
1: function REPAIRSCHEDULE(flow shop f, position n, schedule Ω)   ▷ returns schedule Ω
2:     feasible ← false
3:     end ← false
4:     while !feasible ∧ !end do
5:         (fp', o_fp,k) ← penultimateFirstPass(n, Ω)
6:         (ffp', o_ffp,k) ← ultimateFirstPass(n, Ω)
7:         (sp', o_sp,k+1) ← lastSecondPass(n, Ω)
8:             ▷ find operations and their positions in Ω
9:         if o_ffp = o_1,k then
10:                                  ▷ first operation on machine
11:            end ← true
12:        i ← sp' + 1
13:        while i ≤ ffp' do
14:            Ω ← removeSecondPassOp(o_i,k+1, Ω)
15:            Ω ← insertSecondPassOp(fp', o_i,k+1, Ω)
16:            fp' ← fp' + 1
17:            i ← i + 1
18:        n ← fp'
19:        Ω ← updateStartTimes(f, Ω)
20:        feasible ← checkFeasibility(f, Ω)
21:    Ω ← triggerMaintenance(o_sp, o_1,k, f, Ω)
22:    return Ω
```

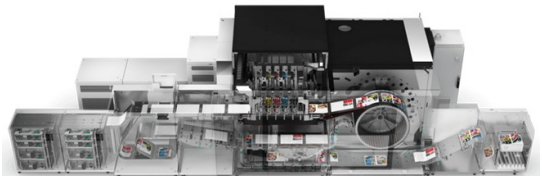

Figure 3: Industrial printer showing the use case.

Type	$P(o_{i,1})$	$P(o_{i,2})$	$P(o_{i,3})$	$P(o_{i,4})$	$D(o_{i,1}, o_{i,2})$	$D(o_{i,2}, o_{i,3})$	$D(o_{i,3}, o_{i,4})$
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

Action Type	Duration	Deterioration States
1	0.5s	10 – 15
2	10s	15 – 30
3	20s	30 – inf

(d) Maintenance Policy of the Use Case

Table 1: Use case job properties. Times in seconds; travelling times treated as setup times between operations of same job.

job with such prints likely having to be discarded and reprinted. We apply our maintenance scheduling idea to the use case based on the list scheduler developed by van Pinxten et al. (2017). We tune the prediction for this use case such that we maintain a machine if a threshold that affects the next operation is crossed or if 90% of the upper bound of a threshold that affects the quality of an operation further down the line is crossed.

Time Complexity Analysis

To insert a maintenance operation, we also have to update the start times of the operations in the schedule. Inserting an operation can be done in constant time but the longest path computation to update begin times takes $O(|V||E|)$ time where $|V|$ is the number of vertices and $|E|$ the number of edges. Our graph (without maintenance) has $|J|r$ edges and vertices – where $|J|$ is the number of jobs and r is the number of operations per job. Since we apply a bounding technique, there is a limited number of jobs $L \ll |J|$ for which we perform this re-computation. The cost of the re-computation is $O(L^2r^2)$. In the worst case, we have to insert a maintenance operation for every one of the L jobs in the window and therefore the cost of evaluating a schedule for maintenance is $O(L^3r^2)$. As L is at least 1 and at most $|J|$, the worst case time complexity is $O(|J|^3r^2)$.

The other algorithm that also needs analysing is the schedule repair algorithm. Repairing the schedule involves inserting elements, removing elements, and updating start times. In the worst case, we have to insert and remove an operation for each job. This can be done in $O(|J|)$ time. After the insertion and removal, we once again need the longest path algorithm to update start times, leading to $O(|J|^3r^2)$ time as there is no limited window for updating begin times at schedule repair. Finally, we cannot guarantee that one cycle of this is enough to bring the schedule back to a feasible state and as shown in Theorem 1, the highest number of recursions is $|J|$ which brings the time complexity to $O(|J|^4r^2)$. Although this is an expensive addition to the runtime of the scheduler, repair happens infrequently in practice.

Experimental Results

In this section, we evaluate the performance of our scheduler on the industrial use case. All experiments are performed on an 8-core 4.6GHz Intel i7 machine with 16GB memory running Ubuntu 20.04.

We generate benchmarks according to the types of jobs typically presented in our industrial use case as described in Table 1 with the assumption that all jobs are duplex, i.e., require re-entrancy. We generate two classes of benchmarks, first with random arrivals of job types, and secondly with patterned arrivals of jobs types. In the patterned arrival, jobs of a type appear in repeated blocks, for instance, 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 compare a base list scheduler – Bounded HCS (BHCS) (van Pinxten et al. 2017) – for this use case, with the maintenance-incorporated version as described in this paper which is indicated as MIBHCS. With BHCS, maintenance is reactive and interrupts the scheduler. We simulate the behaviour of BHCS by evaluating the schedule it produces for maintenance.

Performance Evaluation

In Figure 4, we compare the makespan of the schedules produced by MIBHCS to those produced by BHCS. The average improvement in makespan over all the data sets is 15% while we have minimum improvement of -15% and maximum improvement of 66%. The points where predictive maintenance worsens the makespan are a result of scenarios where the maintenance prediction is too conservative and performs maintenance even though the job set could be completed without it. Figures 6 and 7 show the distribution of the number of maintenance actions and the time spent on maintenance. With MIBHCS, we perform more maintenance actions but on average, spend 50% less time on maintenance. This is because predicting deterioration allows us to perform maintenance before machines deteriorate to a state where we have to pay larger maintenance costs. As another means of evaluation, we look at OEE which is a measure of manufacturing productivity

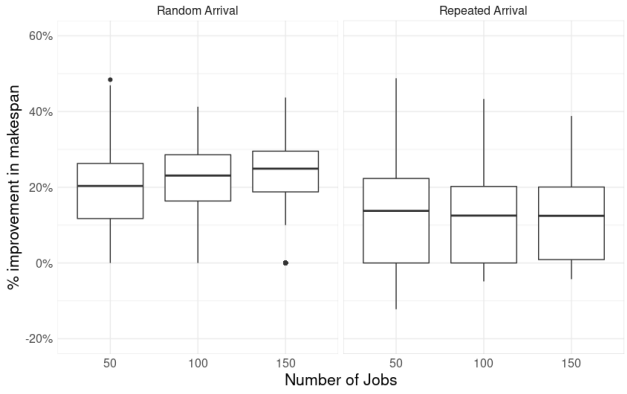

Figure 4: Makespan improvement of MIBHCS over BHCS

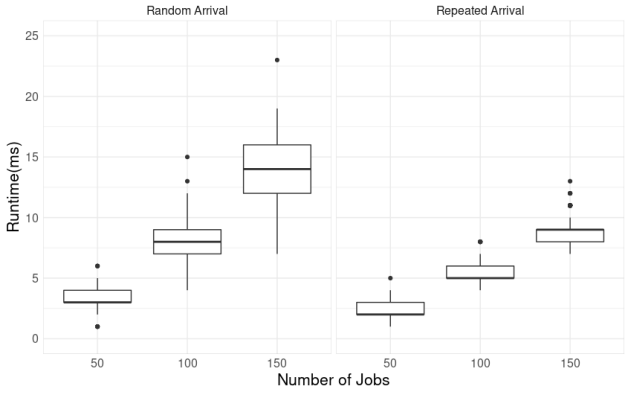

Figure 5: Runtime evaluation of MIBHCS

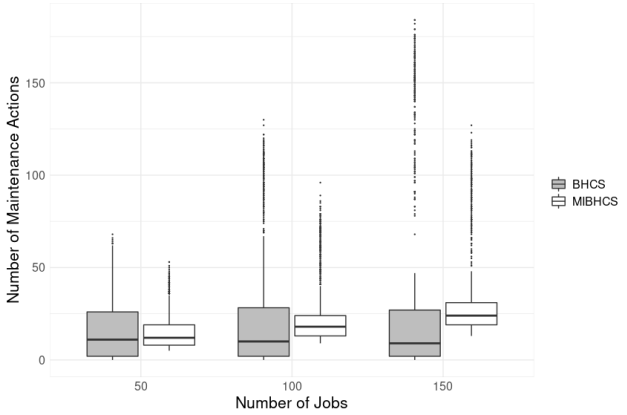

Figure 6: Number of Maintenance Actions

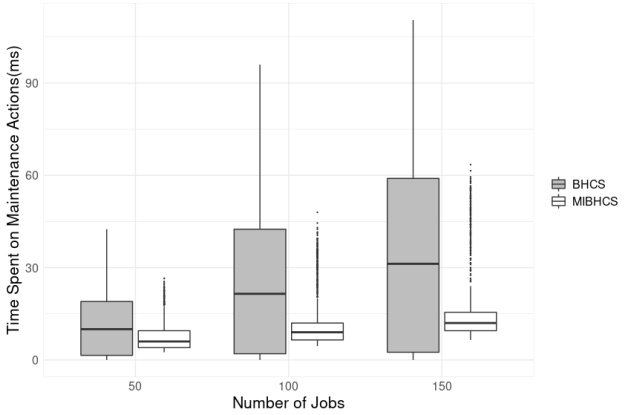

Figure 7: Duration of Maintenance Actions

Number of Jobs	Improvement in OEE
50	7%
100	8%
150	10%

Table 2: Improvement in OEE of MIBHCS over BHCS

(Hansen 2001). OEE is made of three metrics, namely, quality, performance and availability. We compute this under the assumption that maintenance actions are the only source of disruptions. We measure quality in terms of how many re-prints are required in the schedule, and measure availability in terms of the deviation of a schedule's makespan from the planned makespan. We see in Table 2 that we can have up to 10% improvement in the OEE with predictive maintenance planning.

Runtime Evaluation

As all the algorithms discussed are to be suitable for online applications, we evaluate the runtime of our scheduler. The runtime increases with number of jobs as expected and with a planning window of up to 150 jobs, the maximum runtime encountered per job was 23 *ms* while the average runtime across scenarios was 6 *ms*.

Conclusion and Future Work

This paper shows that list scheduling heuristics can be extended to include maintenance operations on reentrant flow shops. We have presented a generic approach to integrated maintenance and production planning. We have demonstrated our approach on a concrete industrial use case, finding a mean 15% improvement in the resulting makespans. Notably, our approach remains applicable for online scheduling.

We have considered maintenance actions on the reentrant machine alone. An interesting direction for future work is to include planning maintenance on all the machines in the flow shop. There is a cyclic dependency of this integrated problem because inserting maintenance operations on one machine affect the start times of operations on other machines. We also look to experiment more extensively, considering other use cases and other schedulers.

Acknowledgements

Thanks to the anonymous reviewers for their suggestions. This work was supported by the MasCot programme of the Dutch Research Council (NWO) under the SAM-FMS project 17931.

References

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

Basnet, C.; and Mize, J. H. 1994. Scheduling and control of flexible manufacturing systems: a critical review. *International Journal of Computer Integrated Manufacturing*, 7(6): 340–355.

Buzacott, J. A.; and Shanthikumar, J. G. 1980. Models for understanding flexible manufacturing systems. *AIIE Transactions*, 12(4): 339–350.

Caricato, P.; and Grieco, A. 2008. An online approach to dynamic rescheduling for production planning applications. *International Journal of Production Research*, 46(16): 4597–4617.

Carpov, S.; Carlier, J.; Nace, D.; and Sirdey, R. 2012. Two-stage hybrid flow shop with precedence constraints and parallel machines at second stage. *Computers & Operations Research*, 39(3): 736–745.

Chan, W. T.; and Wee, T. H. 2003. A multi-heuristic GA for schedule repair in precast plant production. In *13th International Conference on Automated Planning and Scheduling (ICAPS)*, 236–245.

Choi, S.-W.; and Kim, Y.-D. 2008. Minimizing makespan on an m-machine re-entrant flowshop. *Computers & Operations Research*, 35(5): 1684–1696.

Cui, W.; Lu, Z.; Li, C.; and Han, X. 2018. A proactive approach to solve integrated production scheduling and maintenance planning problem in flow shops. *Computers & Industrial Engineering*, 115: 342–353.

ElMaraghy, H. A. 2005. Flexible and reconfigurable manufacturing systems paradigms. *International Journal of Flexible Manufacturing Systems*, 17(4): 261–276.

Hansen, R. C. 2001. *Overall Equipment Effectiveness: a Powerful Production/Maintenance Tool for Increased Profits*. Industrial Press Inc.

Herr, N.; Nicod, J.-M.; and Varnier, C. 2014. Prognostics-based scheduling in a distributed platform: Model, complexity and resolution. In *10th IEEE International Conference on Automation Science and Engineering (CASE)*, 1054–1059.

Kaihara, T.; Fujii, N.; Tsujibe, A.; and Nonaka, Y. 2010. Proactive maintenance scheduling in a re-entrant flow shop using Lagrangian decomposition coordination method. *CIRP Annals*, 59(1): 453–456.

Kaihara, T.; Kurose, S.; and Fujii, N. 2012. A proposal on optimized scheduling methodology and its application to an actual-scale semiconductor manufacturing problem. *CIRP Annals*, 61(1): 467–470.

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

Ladj, A.; Tayeb, F. B.-S.; and Varnier, C. 2018. Tailored genetic algorithm for scheduling jobs and predictive maintenance in a permutation flowshop. In *23rd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, 524–531.

Lin, D.; Lee, C. K.; and Ho, W. 2013. Multi-level genetic algorithm for the resource-constrained re-entrant scheduling problem in the flow shop. *Engineering Applications of Artificial Intelligence*, 26(4): 1282–1290.

Luo, H.; Du, B.; Huang, G. Q.; Chen, H.; and Li, X. 2013. Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics*, 146(2): 423–439.

Phogat, S.; and Gupta, A. K. 2017. Identification of problems in maintenance operations and comparison with manufacturing operations: a review. *Journal of Quality in Maintenance Engineering*, 23(2): 226–238.

Ruiz, R.; and Maroto, C. 2005. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2): 479–494.

Ruml, W.; Do, M. B.; and Fromherz, M. P. 2005. Online planning and scheduling for high-speed manufacturing. In *15th International Conference on Automated Planning and Scheduling (ICAPS)*, 30–39.

Su, Y.-C.; Cheng, F.-T.; Hung, M.-H.; and Huang, H.-C. 2006. Intelligent prognostics system design and implementation. *IEEE Transactions on Semiconductor Manufacturing*, 19(2): 195–207.

Susto, G. A.; Schirru, A.; Pampuri, S.; McLoone, S.; and Beghi, A. 2014. Machine learning for predictive maintenance: A multiple classifier approach. *IEEE Transactions on Industrial Informatics*, 11(3): 812–820.

Swanson, L. 2001. Linking maintenance strategies to performance. *International Journal of Production Economics*, 70(3): 237–244.

van der Tempel, R.; van Pinxten, J.; Geilen, M.; and Waqas, U. 2018. A heuristic for variable re-entrant scheduling problems. In *21st IEEE Euromicro Conference on Digital System Design (DSD)*, 336–341.

van Pinxten, J.; Waqas, U.; Geilen, M.; Basten, T.; and Somers, L. 2017. Online scheduling of 2-re-entrant flexible manufacturing systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 16(5s): 1–20.

Varnier, C.; and Zerhouni, N. 2012. Scheduling predictive maintenance in flow-shop. In *Proceedings of the 3rd IEEE Prognostics and System Health Management Conference*, 1–6.

Waqas, U. 2017. *Scheduling and Variation-aware Design of Self-re-entrant Flowshops*. Ph.D. thesis, Technische Universiteit Eindhoven.

Yang, Z.; Djurdjanovic, D.; and Ni, J. 2007. Maintenance scheduling for a manufacturing system of machines with adjustable throughput. *IIE Transactions*, 39(12): 1111–1125.