Flexible Services and Manufacturing Journal
https://doi.org/10.1007/s10696-023-09511-z

Integrated maintenance and production scheduling for unrelated parallel machines with setup times

Michael Geurtsen[1,2] · **Jelle Adan**[1,2] · **Alp Akçay**[1]

Accepted: 8 September 2023

Abstract
This paper considers jointly scheduling the production and resource-constrained maintenance activities in a manufacturing setting with unrelated parallel machines. In particular, a single maintenance activity needs to be scheduled on each machine in one of its available time windows, and the maintenance activities require a scarce resource, thereby limiting the number of maintenance activities that can be scheduled simultaneously on different machines. In addition, machine- and sequence-dependent setup times, machine eligibility constraints and job-specific release and due dates are considered. A mixed-integer linear program is formulated with objectives including the makespan and, motivated from practice, a weighted sum of total production completion times at machines and total job tardiness. Additionally, a hybrid genetic algorithm with a novel solution representation is proposed for solving industry-scale large instances. A case study is performed with real-world data from a semiconductor manufacturer, where production and maintenance are scheduled separately. The benefit of simultaneously scheduling production and maintenance is investigated. Tests with real-world data show that the proposed model results in schedules that substantially improve the current factory practice.

Keywords Production scheduling · Parallel machines · Integrated planning · Setup times · Mixed integer linear programming · Hybrid genetic algorithm

1 Introduction

The vast majority of research in production scheduling assumes that machines are continuously available during a planning horizon (Allahverdi , 2015). However, in real life, a machine can become unavailable due to maintenance within the planning

✉ Jelle Adan
jelle.adan@pm.me

[1] School of Industrial Engineering, Eindhoven University of Technology, De Zaale, 5600 MB Eindhoven, The Netherlands

[2] Nexperia, Jonkerbosplein 52, 6534 AB Nijmegen, The Netherlands

Published online: 18 October 2023

horizon. Since production and maintenance activities both utilize the available time of machines, it can be highly beneficial to coordinate these two tasks by scheduling them in an integrated manner (Yoo and Lee , 2016; Seif et al. , 2020; Chen et al. , 2021).

This study is motivated by the production of integrated circuits (ICs) at Nexperia, a global semiconductor manufacturer that produces more than 90 billion products annually. In particular, the back-end manufacturing (i.e., assembly or packaging processes after all of the features have been created on a wafer) at Nexperia can be considered as multiple machines in parallel. In industry, settings with multiple parallel machines are frequently encountered, where each job needs to be processed on exactly one machine. Due to the presence of machines of different generations and with different technologies, process times may depend on the machine to which a job is assigned. In the literature, this setting is referred to as *unrelated parallel machines* (Kaabi and Harrath , 2014). Nexperia's back-end production environment encompasses some critical constraints, in part due to the large amount of different types of products. First, not all jobs are eligible for all machines. Also, a machine needs to be prepared before processing a job, and the duration of this preparation can depend both on the current job, the job that was previously processed, and the machine itself; i.e. *sequence- and machine-dependent setup times*.

In general, maintenance can be categorized as corrective maintenance (CM) and preventive maintenance (PM). CM is a maintenance task performed after a machine unexpectedly fails. In our paper, we assume that an effective PM strategy is already in place to avoid any unexpected breakdowns, making the necessity of CM negligible. In particular, we consider that the machines which require a PM activity in a given scheduling horizon are already identified, but it is not yet known when a specific PM activity will be executed. At Nexperia, the maintenance itself is carried out by specialized personnel. Therefore, maintenance scheduling needs to take into account the availability of this specialized personnel, which can be considered as a limited resource for maintenance. Furthermore, this specialized personnel can be available in multiple days but only within a specific time window in a day. The current practice at Nexperia is to separately schedule maintenance and production activities: for a given scheduling horizon, the machines that require PM are selected. Then, taking into account the maintenance-resource constraint, a PM schedule is constructed. From a production scheduling point of view, this affects the machine availability. As the resulting schedules are often considered as suboptimal, it is of practical interest to integrate the scheduling of production and maintenance activities to improve operational performance.

The real-life setting described above leads to a new integrated production and maintenance scheduling problem in the context of unrelated parallel machines with sequence- and machine-dependent setup times and eligibility constraints. To be specific, we aim to simultaneously decide the production schedule of jobs (i.e., assignment of jobs to machines and the sequence of jobs on each machine) and the starting time of the PM on each machine (which requires a PM activity) given that (i) there are multiple time windows in which the PM activity for a machine can be scheduled and the time windows are larger than the duration of the PM activity, leading to flexibility in the planning of PM activities, and (ii) there is a limited maintenance

resource that imposes a constraint on the times at which a PM activity can be conducted. To the best of our knowledge, this setting has not been studied before. Only recently, Avalos-Rosales et al. (2018) included PM activities in an unrelated parallel machine environment with sequence- and machine-dependent setup times, however, it was assumed that each PM activity has a predefined starting time. In our problem, the starting times of PM activities are also optimized (within their predefined intervals). Similar to Avalos-Rosales et al. (2018), we consider minimizing the makespan (i.e., maximum job completion time) as it is an objective commonly used in the scheduling literature. In addition, we consider the weighted sum of total production completion times at machines and the total job tardiness as an objective to better capture the objective of Nexperia in our case study, given that job-specific release and due dates exist in real life and it is important to satisfy customer orders on time.

The main contributions of this article can be summarized as follows:

- A Mixed Integer Linear Programming (MILP) model is formulated to address the problem described above on simultaneously scheduling maintenance and production activities.
- A Hybrid Genetic Algorithm (HGA) is designed and implemented to efficiently solve for large instances that are common in industry. The performance of the proposed HGA is benchmarked against the MILP model.
- A case study from Nexperia is presented to quantify the benefit of simultaneously scheduling production and maintenance activities in practice. The real-world test instances are made publicly available.

The remainder of this paper is organized as follows: Sect. 2 provides an overview of the relevant literature. The problem is described in Sect. 3, and the MILP model is formulated in Sect. 4. The proposed HGA is explained in Sect. 5. Computational results are then presented in Sect. 6. Finally, concluding remarks and recommendations are provided in Sect. 7.

2 Literature review

We set the scope of our review to a production setting with parallel machines and no deterioration. The related work can be grouped into two main categories depending on the existence of maintenance activities. Section 2.1 provides a brief overview of the studies on unrelated parallel machine scheduling in the absence of maintenance activities. Section 2.2 focuses on the studies where the maintenance activities are explicitly considered in parallel machine scheduling.

2.1 Parallel machine scheduling without maintenance

If only the production activities are considered, the problem in our paper (without the maintenance activities) is known as unrelated parallel machine scheduling with sequence- and machine-dependent setups (UPMSP-SMDST); see, for example,

Vallada and Ruiz (2011), Afzalirad and Rezaeian (2016), Santos et al. (2019), and Arnaout (2020) with the makespan objective, and Fowler et al. (2003), Wang et al. (2013), and Diana and de Souza (2020) with objectives based on job-specific due dates (e.g., tardiness). Bitar et al. (2021) consider a UPMSP-SMDST problem with eligibility constraints and extend it with auxiliary resources where each job requires a specific resource of which at most one is available. For the UPMSP-SMDST problem with machine eligibility constraints, Eroglu and Ozmutlu (2017) propose a HGA to minimize makespan. Adan (2022) addresses the same problem, and a new HGA is proposed to solve large industry-size problem instances (with up to 60 machines and 800 jobs) by introducing a new local-search procedure. Our solution algorithm in this paper is built on the same algorithm by extending it with a new solution representation and a mutation mechanism. Different from the papers mentioned so far, our paper considers the maintenance activities in scheduling decisions.

2.2 Parallel machine scheduling with maintenance

We distinguish two groups of studies depending on whether the schedules (i.e., the start and completion times) of maintenance activities are fixed or flexible.

2.2.1 Fixed maintenance schedule

From a production scheduling point of view, the case where maintenance is fixed a priori simply affects the machine availability. Hence, this setting is also referred to as "scheduling with machine availability constraints" in the literature (Kaabi and Harrath , 2014). The problem with a general number of jobs and identical parallel machines is considered with a focus on exact solution methods (e.g., Mellouli et al. 2009), machine eligibility (e.g., Liao and Sheen 2008), resumable jobs (e.g., Hashemian et al. 2014), nonresumable jobs (e.g., Beaton et al. 2016), and periodic unavailability constraints (e.g., Li et al. 2017). Gedik et al. (2016) study unrelated parallel machines with sequence-dependent setup times, and Kaabi and Harrath (2019) study uniform parallel machines with periodic unavailability constraints. Hidri et al. (2021) consider identical parallel machines with machine-specific unavailability constraints and a single resource to perform the job setups on the machines. Different from the studies in this category, we aim to simultaneously schedule the production and maintenance activities, and focus on the most general situation of unrelated parallel machines with eligibility constraints and machine- and sequence-dependent setup times. Recently, Avalos-Rosales et al. (2018) also consider the presence of maintenance in this most general situation, but only focused on scheduling the jobs in machine-specific *fixed* time-blocks between maintenance activities.

2.2.2 Flexible maintenance schedule

There is a rich literature on jointly determining the schedules of maintenance and production activities by allowing the starting times of maintenance activities to be

flexible, which we also refer to as flexible maintenance. To the best of our knowledge, there are two different approaches through which scheduling flexibility for maintenance activities is considered.

The first approach is the following: maintenance can be scheduled at any time, but the time interval between subsequent maintenance activities is constrained by a given upper bound. For example, Graves and Lee (1999) address the scheduling of semiresumable jobs (i.e., if a job is not finished completely before maintenance, an additional setup time is incurred before it can resume) on a single machine together with at most two maintenance activities. Sun and Li (2010) apply a shortest-processing-time-first algorithm to minimize the total completion time in a setting with two identical parallel machines. Costa et al. (2016) consider uniform parallel machines and proposed a metaheuristic procedure based on genetic algorithms and local search to minimize total tardiness. Lee et al. (2018) consider identical parallel machines for the objective of minimizing total tardiness, and introduce a branch-and-bound algorithm with some problem-specific properties.

The second approach for flexible maintenance assumes predefined time windows wherein maintenance activities must be scheduled, provided that the time window is somewhat larger compared to the duration of the maintenance activity. For example, Yang et al. (2002) consider a single machine scheduling problem where the machine should be stopped for a maintenance activity of constant duration within a predefined time window. As the size of the time window is larger compared to the duration of the maintenance activity, some scheduling flexibility is provided. It is demonstrated that the problem is NP-hard and a heuristic is proposed to solve the problem with the objective to minimize the makespan. For the same problem, Yang et al. (2011) propose a dynamic programming method, and Touat et al. (2017) also consider human resource availability and competence constraints. Lee et al. (2015) study an identical parallel-machine scheduling problem with one maintenance activity to be scheduled on each machine, and develop a branch-and-bound algorithm for small-size problems and a HGA for large problems. Rebai et al. (2013) consider a maintenance activity with optimistic and pessimistic deadlines. When the activity is performed between these deadlines, the maintenance cost is minimal. Otherwise, earliness and tardiness costs are imposed.

A special case of the second modeling approach is the case where the maintenance activity on each machine must be scheduled any time before a certain time limit. Lee and Chen (2000) consider parallel machines with nonresumable jobs and resource-constrained maintenance. Yoo and Lee (2016) propose a dynamic programming approach for the same problem as in Lee and Chen (2000). Several objective measures including makespan, (weighted) sum of completion times, maximum lateness and the sum of lateness are studied. Chen et al. (2021) consider two identical parallel machines with nonresumable jobs and the objective of minimizing the makespan. In our paper, there are (multiple) time windows during which maintenance can be conducted at a specific moment, and there is also a time limit such that the maintenance activity on each machine cannot be postponed beyond that limit. That is, our paper belongs to this second group within the flexible-maintenance literature described above. Furthermore, we assume the amount of maintenance resources is limited (e.g., headcount of the specialized maintenance personnel), which means that the number of machines

that can be maintained simultaneously is constrained. As far as we know, this combined setting has not been studied before. Furthermore, in many cases the production environment appears oversimplified, as important practical aspects such as sequence-dependent setup times and machine-dependent process times are neglected. Motivated by the complexity of real-life problems encountered in a semi-conductor manufacturer, our paper addresses this gap in the literature of integrated maintenance and production scheduling.

3 Problem description

We consider an unrelated parallel-machine scheduling problem with sequence- and machine-dependent setup times, eligibility constraints and release and due times for production jobs. As common in the unrelated parallel-machine scheduling literature, it is assumed that (i) jobs are non-preemptive, (ii) there are no precedence constraints among jobs, (iii) the setup time prior to the first job in the sequence is zero and (iv) all machines are available at time zero. Specialized maintenance workers are required to perform PM on the machines.

The maintenance workers only operate in the daytime shift of a 24-hour production day, and one worker executes one PM activity at a time. The machines which require a PM activity are known at the beginning of the scheduling horizon (e.g., typically a week), and only one PM activity is needed for each of these machines during the scheduling horizon. The duration of a PM activity is smaller than the duration of a shift. If the shift is seen as a window, the PM activity can be scheduled flexibly within that window, as long as the end time of the PM activity is before the end time of the window. Multiple maintenance workers operate on each day and there are multiple days possible for the PM activity during the scheduling horizon (i.e., for example a maintenance team can visit the factory on these days). Therefore, a multitude of windows is formed, equal to the *total* number of available workers during the scheduling horizon. The goal is to decide which maintenance activity to schedule in which window. Due to the flexibility within the window, maintenance can be scheduled along with the production jobs, thereby minimizing or even eliminating potential idle time between a maintenance activity and a job. The problem is challenging because this needs to be coordinated with scheduling of production (i.e., assigning production jobs to each machine and sequencing the jobs on machines), leading to an *integrated* planning problem for production and maintenance activities.

In Sect. 4, we formulate the integrated planning problem as an MILP model with the objective of minimizing the makespan. We further consider an alternative objective (i.e., a weighted average of total machine completion time and the total tardiness of production jobs) motivated by practice.

4 MILP formulation

The notation for this formulation is summarized in (Table 1). Each production activity (also referred to as a job) from the set J is to be scheduled on one of the unrelated parallel machines from the set M. The subset of machines that require PM is denoted by $R \subseteq M$. For each machine in R, one PM activity is scheduled. PM activities can be scheduled on each day of the set D, where D is a set comprising the eligible days (e.g. Tuesday, Wednesday, Thursday). However, PM requires a resource, and therefore, the maximum number of PM activities that can be scheduled on day $d \in D$ is constrained by the available maintenance resources (i.e., maintenance workers) on that particular day, which is denoted by w_d. Given the sets R and D, the Cartesian product $R \times D$ contains all possible occasions of PM activities. In the end, only a single PM activity is scheduled for a machine requiring a PM activity, while the other possible PM activities for that particular machine remain unscheduled. By using this formulation, at most one PM activity per machine can be scheduled. This matches a real-life scenario perfectly, as maintenance on one machine is typically performed just once during a scheduling horizon. The length of the scheduling horizon and therefore the interval of maintenance is usually defined by the user. By adopting this concise formulation, the constraints are kept at a minimum. Although we make the assumption for the

Table 1 Summary of notation

Sets	
J	Set of jobs.
M	Set of machines.
$R \subseteq M$	Set of machines that require maintenance.
D	Set of days available for maintenance.
$A = J \cup (R \times D)$	Set of all activities.
$A_0 = \{0\} \cup J \cup (R \times D)$	Set of all activities including dummy job 0.
$M_a \subseteq M$	Set of non-eligible machines for activity $a \in A$
Parameters	
p_{ia}	Processing time of activity a on machine i.
s_{iab}	Setup time for processing activity a just before activity b on machine i
r_a	Release time of activity a.
d_a	Due time of activity a.
w_d	Available number of resources for maintenance on day d.
V	A very large number.
Variables	
X_{iab}	Indicates if activity a is scheduled before activity b on machine i.
Y_{ia}	Indicates whether activity a is assigned to machine i.
C_a	Completion time of activity a.
$C_{\max}$	Maximum completion time.
S_{ab}	Setup time for processing activity b if it is preceded by maintenance activity a.

MILP model such that at most one maintenance activity can be performed per machine, the proposed HGA in Sect. 5 can handle more general situations without this assumption.

In our formulation, all jobs and all possible occasions for PM activities are regarded as activities. The set A comprises all the activities, i.e. $A = J \cup (R \times D)$. The release time and due time of activity $a \in A$ are denoted by r_a and d_a, respectively. For a job, the release time indicates the earliest time at which the production of that job can start (e.g. due to material availability) and the due time refers to the delivery time agreed with a customer. For the PM activities, the release and due times correspond to the availability of resources which can differ per day, e.g. a maintenance worker is available on Tuesday from 9:00 to 18:00, whereas on Wednesday this may be between 08:00 and 19:30. Consequently, all possible occasions of PM activities on a particular day need to share the same release and due times.

The setup time between the subsequent processing of activities a and b on machine i is s_{iab}. When two consecutive jobs are interrupted by a PM activity, the sequence-dependent setup time between the two jobs remains unaltered. To deal with this, the variable S_{ab} is introduced to denote the setup time for activity b if preceded by PM activity a, where $b \in J$ and $a \in R \times D$; i.e., S_{ab} takes the value of $s_{ib'b}$, where b' is the activity preceding PM activity a. Furthermore, each activity a is associated with a process time p_{ia} that depends on the machine i to which it is assigned. If $a \in J$, then p_{ia} equals the processing time of the job. If $a \in R \times D$, then p_{ia} is the duration of the corresponding PM activity. As not all activities can be processed on every machine, the set of non-eligible machines for each activity $a \in A$ is denoted by $M_a \subseteq M$. Furthermore, the variables C_a and T_a denote the completion time and the tardiness of each activity $a \in A$, respectively.

This formulation builds upon a foundation of existing formulations for the unrelated parallel machine scheduling problems with setup times, without the consideration of maintenance (Vallada and Ruiz , 2011; Tran and Beck , 2012; Avalos-Rosales and Angel-Bello , 2015). However, as mentioned in Sect. 1, the constraints associated with PM scheduling are new and different from standard production scheduling. In particular, there are two main novelties: first, for each PM activity there is a multitude of possible time windows in which it can be scheduled, while the maximum number of PM activities scheduled in the same time window is constrained by a scarce resource. The second novelty is a set of constraints that guarantees the sequence-dependency between two consecutive jobs in case they are interrupted by a PM activity.

Finding a solution to the problem means to determine which activities are allocated to which machines, and the sequence in which each machine will process the assigned activities. To establish this, two binary decision variables are introduced. First, the machine assignment is dictated by:

$$Y_{ia} = \begin{cases} 1 & \textit{if activity } a \textit{ is assigned to machine } i \\ 0 & \textit{otherwise.} \end{cases}$$

and secondly, the sequence of activities a and b at machine i is defined by:

$$X_{iab} = \begin{cases} 1 & \textit{if activity } a \textit{ is scheduled directly before activity } b \textit{ on machine } i \\ 0 & \textit{otherwise.} \end{cases}$$

In case activity a is a PM activity, it will be explicitly denoted by the index (i, d). For instance, $Y_{i(i,d)}$ indicates the assignment of the PM activity of machine $i \in R$ on day $d \in D$, since $R \times D$ is the matrix corresponding to all possible occasions of the PM activities.

As in Avalos-Rosales and Angel-Bello (2015), a dummy activity 0 is introduced. The set A plus this dummy job is denoted A_0. The variables X_{i0b} and X_{ia0} are used to specify which jobs a and b are processed first and at the end of each machine i, respectively. The process and setup times associated with this dummy job are all 0, i.e. $p_{i0} = 0$, $s_{i0b} = 0$ and $s_{ia0} = 0$.

Two objective functions are considered in this work: (i) minimization of the maximum completion time and (ii) minimizing a weighted sum of the total machine completion time and total tardiness. The former is commonly used in the literature and is considered solely for benchmark purposes. However, minimization of the maximum completion time is often not the desired objective for practical applications with sequence and machine dependent setup times. A higher completion time is often accepted if it enables more efficient production overall, e.g. less time spent on setups, as long as due dates remain largely satisfied. Hence, to comply with practice, a multi-objective optimization problem (e.g., Hoseinpour et al. 2020, 2021) is proposed that attempts to balance between efficient production on one hand and customer satisfaction on the other. In the remainder of this section, the problem formulation for both objectives is presented. First, using the notation introduced previously, and with the objective to minimize the maximum completion time C_{max}, the problem is formulated as follows:

$$\textit{minimize } C_{max} \tag{1}$$

subject to

$$Y_{ia} = 0 \quad a \in J,\ i \in M_a \tag{2}$$

$$Y_{i0} = 1 \quad i \in M \tag{3}$$

$$\sum_{i \in M} Y_{ia} = 1 \quad a \in J \tag{4}$$

$$\sum_{d \in D} Y_{i(i,d)} = 1 \quad i \in R \tag{5}$$

$$\sum_{a \in A_0 \setminus \{b\}} X_{iab} = Y_{ib} \quad b \in A,\ i \in M \tag{6}$$

$$\sum_{b \in A_0 \setminus \{a\}} X_{iab} = Y_{ia} \quad a \in A,\ i \in M \tag{7}$$

$$\sum_{i \in R} Y_{i(i,d)} \leq w_d \quad d \in D \tag{8}$$

$$C_b + V(1 - X_{iab}) \geq C_a + s_{iab} + p_{ib} \quad i \in M,\ a \in J,\ b \in A \setminus \{a\} \tag{9}$$

$$C_b + V(1 - X_{iab}) \geq C_a + S_{ab} + p_{ib} \quad i \in R,\ a \in R \times D,\ b \in A \setminus \{a\} \tag{10}$$

$$S_{ab} + V(1 - X_{iab}) \geq \sum_{b' \in 0 \cup \{J\} \setminus b} X_{ib'a} s_{ib'b} \quad i \in R,\ a \in R \times D,\ b \in A \setminus \{a\} \tag{11}$$

$$C_{(i,d)} - p_{i(i,d)} Y_{i(i,d)} \geq r_{(i,d)} \quad i \in R,\ d \in D \tag{12}$$

$$C_{(i,d)} \leq d_{(i,d)} \quad i \in R,\ d \in D \tag{13}$$

$$C_a - p_{ia} Y_{ia} \geq r_a \quad i \in M,\ a \in J \tag{14}$$

$$X_{iab}, Y_{ia} \in \{0, 1\} \quad a \in A,\ b \in A,\ i \in M \tag{15}$$

$$S_{ab} \geq 0 \quad a \in R \times D,\ b \in J \tag{16}$$

$$0 \leq C_a \leq C_{max} \quad a \in A \tag{17}$$

Constraints (2) to (7) are generalized from the work proposed by Tran and Beck (2012) and constraints (9) and (10) are derived from the study by Vallada and Ruiz (2011). All other constraints presented in the formulation are novel additions. Constraints (2), (3) and (4) ensure that each activity is assigned to exactly one eligible machine. Constraints (5) indicate that a maintenance activity of a particular machine may be performed on only one of the available days for the maintenance. Constraints (6) establish that every activity has exactly one predecessor and both are assigned to the same machine. Similarly, Constraints (7) guarantee that every activity has exactly one successor and both are assigned to the same machine. Constraints (8) guarantee that the number of maintenance activities performed per day does not exceed the available resources at that day. Constraints (9) provide the right processing order, i.e the completion time of an activity in a sequence on a machine is greater than or equal to the completion time of the preceding activity, the sequence-dependent setup time and the processing time of the current activity. V denotes a large constant. Constraints (10) provide the correct processing order in case the activity prior to the current activity is a maintenance activity. The true sequence-dependent setup time must be selected by determining the job before the maintenance activity, and recall that the setup time remains unaltered in case of maintenance between two subsequent

jobs. Constraints (11) define the sequence-dependent setup time when a maintenance activity is scheduled between two jobs. Specifically, in case job b' precedes job b but maintenance activity a is processed between, the setup time between maintenance activity a and job b, S_{ab}, is set equal to the setup time between jobs b' and b on machine i, i.e., $s_{ib'b}$. Constraints (12) specify the release time restrictions for maintenance activities. Note that it is permitted to perform the setup time required for a PM activity before its release time. Similarly, Constraints (13) specify the due date restrictions for maintenance activities. Constraints (14) specify the release time restrictions for production activities. Note that these strict time restrictions imply that the resulting schedule may contain idle times between activities. Finally, Constraints (15) and (16) specify the domains of the variables X_{iab}, Y_{ia} and S_{ab}, and Constraints (17) define the makespan.

In addition, based on the makespan linearization of Avalos-Rosales and Angel-Bello (2015), Constraints (18) and (19) are added to accelerate a solver:

$$\sum_{d \in D} S_{(i,d)b} + p_{ib} Y_{ib} \leq C_{max} \quad i \in R,\, b \in A \tag{18}$$

$$\sum_{a \in J \setminus \{b\}} X_{iab} s_{iab} + p_{ib} Y_{ib} \leq C_{max} \quad i \in M \setminus R,\, b \in A \tag{19}$$

Although Eq. (1) attempts to minimize the maximum completion time, it does not focus on the due dates of individual production activities. As mentioned, this is often not desirable in practice. Hence, as an alternative, the following objective function is proposed:

$$minimize\ \alpha \sum_{i \in M} \sum_{a \in A} X_{ia0} C_a + \beta \sum_{a \in J} T_a \tag{20}$$

where α and β represent the weights of the Total Machine Completion Time (TMCT) and the Total Tardiness (TT) of the jobs, respectively. The machine completion time at a particular machine is the time at which the last activity assigned to that machine is completed, i.e., the activity for which X_{ia0} equals 1 for machine i (recall that X_{ia0} was introduced to specify the last job processed at machine i). The TMCT refers to the sum of the machine completion times over all machines. We refer to adopting the objective in (20) as the multi-objective optimization approach, as it includes both the total machine completion time and job tardiness as objectives. Note that the first term (i.e., TMCT) in (20) can be much greater than the second (i.e., TT), depending on factors such as the number of jobs and their due date. To determine the weights that best reflect the objective of the planner, a wide range of weight-combinations should be examined; see Sect. 6.1 for details. The following constraints are required to define the tardiness for production activities:

$$T_a \geq C_a - d_a \; a \in J \tag{21}$$

$$T_a \geq 0 \; a \in J \tag{22}$$

Besides Constraints (21)–(22), the formulation to optimize Eq. (20) requires Constraints (2)–(16), whereas Constraints (17)–(19) become redundant. Note that (20) is non-linear. However, it is rather straightforward to linearize this equation using the big-M method (Dantzig , 1948). For this purpose, the maximum completion time of a machine i is represented by F_i, which is defined as follows:

$$F_i \geq C_a - V(1 - X_{ia0})\, a \in A,\ i \in M \tag{23}$$

Then, F_i is incorporated in Eq. (20) to obtain the following linear objective function:

$$minimize\ \alpha \sum_{i \in M} F_i + \beta \sum_{a \in J} T_a \tag{24}$$

5 Hybrid genetic algorithm

It is well-known that an MILP may not be capable of solving large-sized problem instances in a reasonable amount of time within the context of unrelated parallel machines with setup times (e.g. $>$ 10 machines and $>$ 100 jobs) even when there are no maintenance activities (Avalos-Rosales and Angel-Bello , 2015). Therefore, metaheuristics are often used as an alternative to MILP formulations. A commonly used metaheuristic is genetic algorithms. A Hybrid Genetic Algorithm (HGA) is an extension of the traditional genetic algorithm (GA), in which the GA is equipped with a local search technique to reduce the likelihood of premature convergence. HGAs have been successfully applied to production scheduling in parallel machine environments with the additional complexities of sequence and machine dependent setup times, machine dependent process times and eligibility constraints (Vallada and Ruiz , 2011; Adan , 2022). Since our study focuses on a similar production environment, we also propose an HGA to solve the integrated planning problem with the additional decisions on scheduling PM activities. Compared to a standard genetic algorithm, our proposed HGA has several advantages. The HGA employs a minimal number of parameters, which allows easy calibration. For the production scheduling problem in the same context, an extensive comparison of HGAs is made not only with respect to standard genetic algorithms but also simulated annealing (Adan , 2022). It is important to note that an existing HGA developed only for scheduling the production jobs would not be suitable for our problem on hand due to its inefficiency in checking the availability of maintenance resources. Therefore, we develop a novel HGA that is capable of exploiting the characteristics of our problem. This exploitation ability is especially useful when the solution representation size is extended, which is the case with the presence of PM activities and resources.

5.1 Solution representation

The main novelty of the proposed HGA is its solution representation. As local search is recognized as a crucial component for the performance of a HGA, it is important to allow fast evaluation of changes made to the solution. The solution representation

in the proposed HGA aims to facilitate this. In classical parallel machine scheduling, solutions are typically represented by an array of jobs for each machine that reflects the processing sequence of the jobs assigned to that machine. When the processing sequence of a certain machine is translated into a schedule, process and setup times are added consecutively. In this work, PM activities are considered as well. The PM activities require resources (i.e., maintenance workers), the availability of which is limited. Thus, when the processing sequence of a machine is translated into a schedule, the availability of resources needs to be taken into account. As a resource can be claimed by any other machine, in principle, it demands evaluating the schedules of all the machines. Although additional computational effort is inevitable, this can be minimized through an extended solution representation.

More specifically, analogous to the MILP formulation, jobs and PM activities are both regarded as activities in our HGA. Each solution is composed of two components: (i) $|M|$ arrays of activities and (ii) a list that specifies which machines claim a resource at which times. A schematic example of this solution representation is shown in Fig. 1. When a change is made to the processing sequence of a machine, the objective function is evaluated by iterating through this sequence once, querying this additional list along the way. This eliminates the need to evaluate all other machines and thereby minimizes the computational effort. When a change in the processing sequence of machine assignment is implemented, this is not only reflected in a change in the arrays of activity sequences, but also the list of resource claims is updated accordingly.

To illustrate this, we provide the example in Fig. 2: four activities $\{a_1, a_2, a_3, a_4\}$ are to be scheduled on two machines $\{m_1, m_2\}$. Activities a_1 and a_2 are PM activities, requiring a resource. There are two intervals of equal length (indicated by the vertical dotted lines) during which one resource is available. Suppose that the arrays of activities state that activity a_2 is processed after activity a_4 on machine m_1, and activity a_1 is processed after activity a_3 on machine m_2. When the schedule at machine m_1 is evaluated, activity a_4 is scheduled first. As this activity does not require a resource, the resource list is not queried. On the contrary, activity a_2 does require a resource. Thus, before activity a_2 follows activity a_4, the resource list is queried to check whether the required resource is available. If it is available, activity a_2 directly

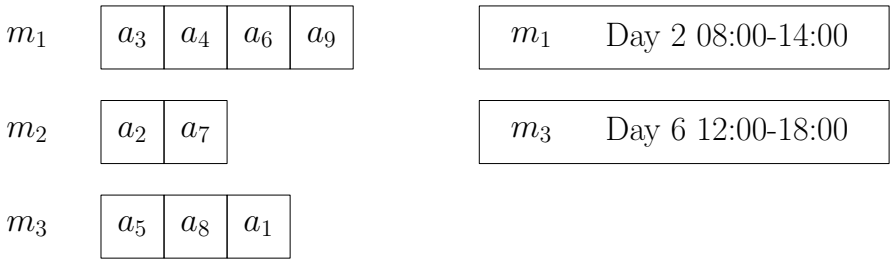

Fig. 1 A schematic example of the solution representation. Here, there are 3 machines and 9 activities. The array of arrays on the left dictates the machine allocation and the sequence of the activities on each machine. Two of the nine activities, a_1 and a_3, are maintenance activities and demand a resource. This resource claim is registered in the list that is shown on the right

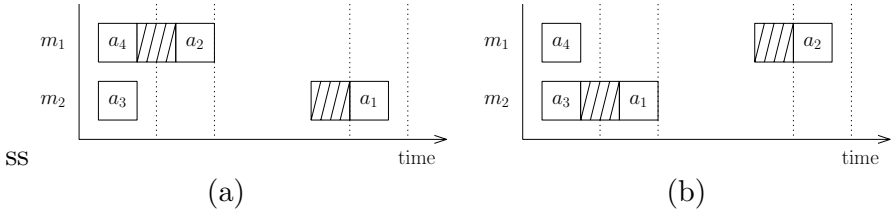

Fig. 2 An example of a resource allocation conflict. There are two time intervals during which one resource is available, as indicated by the vertical dotted lines. Activities a_1 and a_2 require an additional resource. Hence, either a_2 claims the resource in the earliest interval and a_1 is moved to the later interval **a** or vice versa **b**. Dashed blocks indicate setup times

follows activity a_4, as is shown in Fig. 2a. However, if no resource is available at that time (since it is claimed by activity a_1 on machine m_2), it is postponed to the earliest time that a resource becomes available, as is shown in Fig. 2b. If the solution representation had not kept track of the claimed resources, this would require the entire schedule to be evaluated and a heuristic to decide whether the resource is allocated to m_1 or m_2; all in all demanding significantly more computational effort compared to a single query to the aforementioned list. Moreover, by adopting this solution representation, the HGA is capable of scheduling multiple PM activities per machine, since release and due times are associated with each PM activity individually. Therefore, correct sequencing of PM activities is ensured due to the mechanism of the resource list to check the availability together with the multiple PM intervals defined by the release and due times per individual PM activity. Even in more complex scenarios with multiple PM activities per machine, correct sequencing is ensured by this mechanism.

5.2 Description of the proposed HGA

In an HGA, a population of candidate solutions within the search space, so-called individuals, evolves toward better solutions through an iterative evolutionary process. In Fig. 3, a schematic overview of the algorithm is shown. After initialization, an evolutionary cycle begins with selection of two individuals: the parents. A crossover mechanism is applied to the parents that generates two new solutions: the offspring. Then, to maintain a diverse population, each of the offspring is subjected to several mutations. At last, a local search operator is applied to the offspring. This operator is repeatedly applied until it fails to improve the offspring, i.e. until a local minimum is reached. The offspring is accepted into the population if (i) there are no identical individuals already in the population, i.e. they are unique, and (ii) they are fitter than the weakest individual in the population. If they are accepted, the offspring replaces the weakest individuals, otherwise they are discarded. Hereby, the population size, denoted with P, remains constant. These evolutionary cycles are repeated as long as a certain termination criterion remains unsatisfied. In the remainder of this section, a description of the components of our HGA is provided.

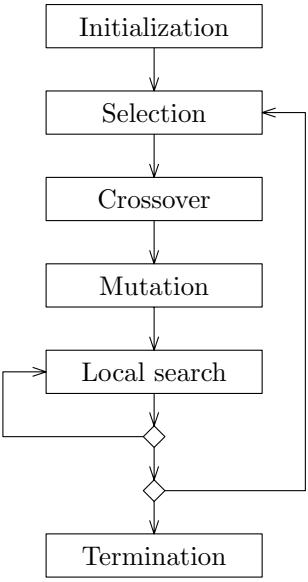

Fig. 3 Hybrid Genetic Algorithm (HGA) flow chart. The diamonds represent conditional decisions

Fitness Function Following the common terminology of GAs, the objective function of the optimization problem is referred to as the fitness function. For our problem, two fitness functions are considered: (i) the makespan, and (ii) a weighted sum of the Total Machine Completion Time (TMCT) and Total Tardiness (TT), see Eqs.s (1) and (20).

It is important to note how activities are penalized when the release or due dates cannot be satisfied. As mentioned, the solution representation specifies the processing sequence of the activities at each machine. To calculate the corresponding fitness value, each of these sequences need to be translated into a schedule. Hereby, process and setup times, but also release and due times are taken into account. For example, consider the sequence $\{a_3, a_2, a_4, a_1\}$. The corresponding schedule is schematically shown in Fig. 4. The size of the white blocks corresponds to the process time and the dashed blocks indicate setup time. The dotted vertical lines in Fig. 4 indicate the release and due time of activity a_1. Regardless of whether a_1 is a production activity or a maintenance activity, the release time is strict. Note that as a consequence of the strict release time constraint, there is an idle time between activities a_4 and a_1. If this activity is a production activity, the release time is strict, but the due date

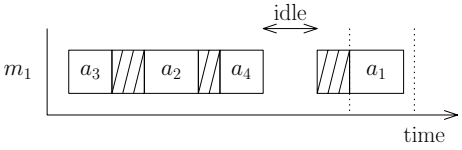

Fig. 4 The schedule corresponding to the sequence $\{a_3, a_2, a_4, a_1\}$. The vertical dotted lines indicate the earliest start and latest completion time for activity a_1. The dashed blocks indicate setup time

can be violated. With the objective in (1), due date violations are neglected, whereas the objective in (24) penalizes these violations through the tardiness. In case a_1 is a maintenance activity, both the release and due times are strict, as these times correspond to the availability of additional resources. Solutions where the due dates for the maintenance activities cannot be satisfied can be generated by the algorithm. If such a case occurs, a penalty is added to the fitness of the individual. In essence, the penalty pushes the maintenance activity far away to the end of the schedule, thereby creating a large gap between the maintenance activity and the last job activity on the machine. The value of the penalty is chosen such that a sufficiently large gap is created which results in a guaranteed 'correction' in the next iteration. This correction ensures that the maintenance activity will be scheduled within the strict release and due times.

Initialization A simple heuristic is applied to generate an initial population. First, a random permutation of all activities is generated by means of the Fisher-Yates shuffle mechanism (Fisher and Yates , 1963). Following this sequence, all activities are subsequently scheduled. This is done as follows: for each activity, all possible insertion positions are evaluated, thereby respecting machine eligibility constraints. To be specific, the insertion mechanism assigns the activity to a particular possible position in the schedule, and the effect on the fitness is evaluated. This procedure is repeated for each possible position in every eligible machine. Finally, the position that is most favorable in terms of the fitness function is selected and implemented. This yields an initial population that is both strong and diverse.

Selection Although various selection mechanisms are often applied in HGAs, e.g. random, roulette-wheel and tournament selection, several experiments with different operators showed that none proved significantly more effective over another. For this reason, random selection is applied in our HGA as this is the simplest and computationally most efficient method.

Crossover Once parents are selected, the crossover mechanism is applied. Many crossover operators are reported in literature, one of the most common being the one-point crossover technique. Here, a local search enhanced one-point crossover operator is applied, see Vallada and Ruiz (2011). In principle, the applied operator acts through the same mechanism, however, in our case, it is necessary to implement the checks to account for the resource constraints.

Mutation The purpose of a mutation operator is to maintain the diversity within the population, to prevent premature convergence to a local optimum, and to explore the solution space further. As for crossover operators, several mutation operators are proposed in the literature, see Soni and Kumar (2014). The mutation operator used within our HGA proceeds as follows: first, the worst machine of the individual in terms of the fitness function is selected. Then, all activities of this machine are removed, leaving the machine with an empty sequence. Each of these activities is then assigned to another randomly chosen machine. To be specific, these activities are allocated to the other machines in the following procedure: for each activity, a random eligible machine is chosen, and for that randomly chosen eligible machine, the activity is inserted in the sequence at the best possible position in terms of fitness. This procedure is repeated r times, where r is a design parameter of our HGA, representing the number of mutations.

Local Search To speed up the search towards a local optimum, we use a local search operator that evaluates three local search neighborhood structures consecutively: the insertion neighborhood, swap neighborhood, and nearest neighbor search. Each of these neighborhood structures is divided into smaller sub-neighborhoods. Specifically, a sub-neighborhood exists for each machine. The sub-neighborhoods are evaluated starting with the worst-fit machine, then the second worse, and so on.

The total number of sub-neighborhoods that are evaluated in each neighborhood structure is constrained by a set of parameters that specify a certain fraction of the machines: d_{ins} for the insertion neighborhood, d_{swap} for the swap neighborhood, and d_{nn} for the nearest neighbor search. These parameters can be tuned to specify the degree of exploitation in each neighborhood structure for the integrated planning problem on hand, as will be discussed in Sect. 6.3. After all the neighbors in a sub-neighborhood are evaluated, it needs to be decided which neighbor is implemented. With the objective to minimize C_{max}, the change in maximum completion time is checked first: if it increases, the neighbor is immediately discarded. From the remaining neighbors, if any, the neighbor that results in the largest decrease of the total completion time is selected and implemented. Concerning the weighted multi-objective fitness function, it is first checked whether a neighbor decreases the fitness. If there are any neighbors remaining, the one that results in the largest decrease of the fitness is selected and implemented.

Termination In parallel machine scheduling without PM activities, termination criteria that scale with both the number of jobs $|J|$ and the number of machines $|M|$ are commonly applied, usually in the form $C \cdot |J| \cdot |M|$, where C is some constant (Vallada and Ruiz , 2011; Avalos-Rosales and Angel-Bello , 2015; Santos et al. , 2019; Adan , 2022). As the presence of PM activities influences the difficulty of the scheduling problem, we incorporate the number of maintenance activities $|R|$ in the termination criterion, i.e., the termination criterion is extended to $C \cdot |J| \cdot |M| \cdot |R|$. The constant C can be set based on side experiments that show how much time is typically sufficient for the algorithm to converge.

Eligibility Each activity has a unique list of eligible machines. Before a move (crossover, mutation, local search) for an activity is performed, the list of eligible machines is examined. A move to another machine may only occur in case that machine is present in the eligibility list of the activity. This eligibility mechanism ensures that feasibility is always guaranteed.

6 Case study

Section 6.1 presents how the problem instances are obtained as part of our case study at Nexperia. Section 6.2 describes how the weights needed for the multi-criteria objective function are obtained in practice. In Sect. 6.3, we describe how the parameters of the proposed HGA are calibrated to improve its performance. In Sect. 6.4, we focus on the instances where the MILP formulation can be solved (close) to optimality, and compare the performance of the calibrated HGA to the performance of the MILP solution. In Sect. 6.5, we apply the calibrated HGA to large-size instances, and quantify the benefit of integrating the scheduling of production and maintenance

activities. Finally, we perform additional experiments in Sect. 6.6 to gain managerial insights on how the constraints imposed on PM activities affect the objective value. The HGA is coded in C# 6.0 and all experiments are run on a computer with an Intel Core i5-540 M (2.53 GHz) processor and 4 GB of memory.

6.1 Real-world problem instances

As part of the case study, production schedules of various product types at Nexperia are gathered over a period of three months. The complete set comprises 25 production schedules, each of which contains various numbers of machines, jobs and maintenance activities; and it is publicly available at https://git.io/J8ny2. An overview of the schedules and the key characteristics is shown in Table 2, where "Maintenance" indicates the number of machines requiring maintenance, and "Horizon" denotes the maximum number of time windows available for maintenance. For each job, the machine dependent process time, the due date, and the sequence-dependent setup

Table 2 Overview of the real-world problem instances

Instance	Machines	Jobs	Maintenance	Horizon
1	55	134	4	6
2	33	124	4	7
3	33	101	4	5
4	33	87	4	6
5	56	171	3	8
6	32	102	3	6
7	33	101	3	8
8	33	102	3	6
9	33	92	3	6
10	32	88	3	8
11	56	171	2	7
12	33	165	2	7
13	33	109	2	4
14	32	100	2	7
15	33	98	2	7
16	33	98	2	6
17	32	92	2	6
18	33	94	2	4
19	33	83	2	6
20	33	80	2	5
21	33	97	1	5
22	33	90	1	7
23	10	65	1	6
24	12	62	1	4
25	10	57	1	6

The instances used for parameter calibration are in italic

times are provided. Although the duration of maintenance activities varies between 7 and 12 h, maintenance is always scheduled between 6:00 in the morning and 20:00 in the evening. Only one resource (i.e., maintenance worker) is available during these time intervals. That is, at most one maintenance activity can be scheduled at a time. The setup time required for a maintenance activity is constant, independent of the job sequence. It's worth mentioning that the instances we've gathered originate from real production data, ensuring that the dataset remains realistic. For all these instances, the number of maintenance tasks to be scheduled is always kept lower than the total available options for maintenance. These options, in turn, depend on how many maintenance resources are available each day and the total number of days in the scheduling horizon. When dealing with new instances, it's crucial to double-check feasibility to make sure not to try fit in more maintenance tasks than there have available spots for.

When the HGA is deployed in practice, it is important to decide how much computational time the algorithm is granted. Based on manual experimentation, the constant C of the termination criterion for our HGA is set to 125 milliseconds, as this proved to provide sufficient time for the algorithm to converge in all the instances.

In case the load of the schedule would be high enough such that it is impossible to schedule everything within the targeted planning horizon, some activities might overflow to the next planning horizon. Although this situation is not observed in our case study, we note that our HGA would still be capable of dealing with these scenarios by blocking the corresponding first parts in the schedule of machines.

6.2 Setting the multi-objective weights

In practice, planners make a trade-off between minimization of the total machine completion time and the total tardiness of jobs. To gain insight in this trade-off, the weights in Eq. (20) are set to values in $\{1, 3, 8, 10, 20\}$. For all possible weight combinations, the HGA is applied to a subset of the problem instances from Table 2 (i.e., 10 instances from the 25 available instances, as shown in italic). These instances are chosen such that this subset contains a diversity of problem sizes. For each weight setting, the average of both terms in the objective function over the selected instances is determined. The resulting Pareto chart is shown in Fig. 5.

The conflict between the two terms, total tardiness and total machine completion time, is clearly visible. The total machine completion time can be improved significantly at the cost of total tardiness. Clearly, it is not possible to achieve zero tardiness. The total tardiness cannot be brought below roughly 1600 h, regardless of the weight setting, which results in a relatively dense cluster in the lower right area of the Pareto chart. In consultation with the planners, it was chosen to opt for a setting where the total tardiness is close to its minimum, and the total machine completion time is as low as possible. Hence, the combination of weights (1, 1) is chosen as this combination shows a negligible increase in total tardiness, compared to a relatively high decrease in total machine completion time. This combination of weights is used in the remainder of the paper.

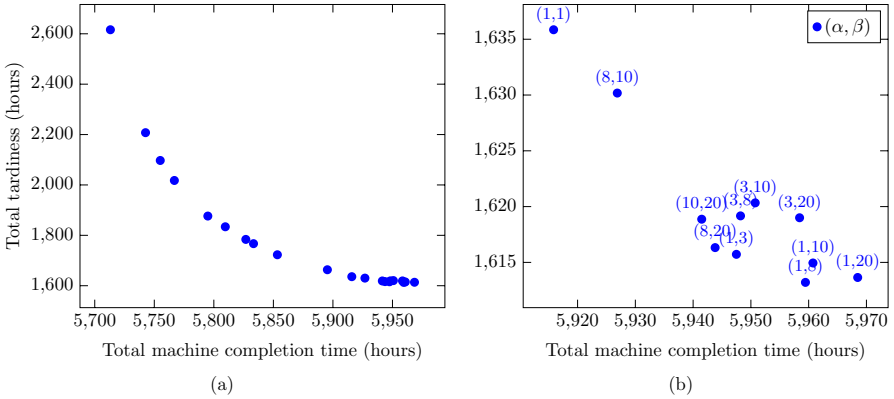

Fig. 5 The average of both terms in the objective function over the selected instances for various weight settings (**a**). A magnification of the lower right area in **a** is shown in (**b**)

6.3 Calibration of the proposed HGA

As for any metaheuristic algorithm, the performance of the HGA depends largely on the setting of its parameters. Calibration of these parameters is itself a very tough optimization problem (Yang , 2014). In this work, the parameters are calibrated by means of Design of Experiments (DOE), as in Chang and Chen (2011), Vallada and Ruiz (2011) and Adan (2022). For a given instance, the performance of the HGA is evaluated by means of the Relative Proportional Deviation (RPD), which is computed according to:

$$\text{RPD} = \frac{A - B}{B} \cdot 100\% \tag{25}$$

where A is the objective value found for a certain problem instance with a specific method and B is the best solution known for this problem instance. The population size, the mutation rate, the depth of the nearest neighbor search, and the depth of the insertion and swap neighborhoods are identified as tuneable parameters in the proposed algorithm. The calibration of these parameters is done as follows: first, two levels (low and high) are identified for each of the parameters, as is shown in Table 3. A full factorial design is generated for the same 10 instances used in Sect. 6.2 (i.e., the italic instances from Table 2). To be specific, 10 independent runs are performed for each one of these 10 instances at each possible parameter configuration.

For each parameter configuration, the average RPD over all instances is calculated. The results are analysed by means of analysis of variance (ANOVA). In Fig. 6, a Pareto chart is provided to show the standardized effects in decreasing order of statistical significance. For a more in-depth explanation on Pareto charts in ANOVA and the calculation of standardized effects, the reader is referred to Montgomery and Runger (2010). The length of each bar in Fig. 6 is proportional to the value of a t-statistic calculated for the corresponding effect. The higher the

Table 3 Tuneable parameters in the proposed algorithm and tested values for the calibration

Parameter	Low level	Manual	High level
Population size (P)	**10**	15	20
Mutation rate (r)	**1**	3	5
Depth nearest neighbor (d_{nn})	0.8	0.9	**1.0**
Depth insertion neighborhood (d_{ins})	0.8	0.9	**1.0**
Depth swap neighborhood (d_{swap})	0.5	0.75	**1.0**

The best parameter combination is shown in bold face

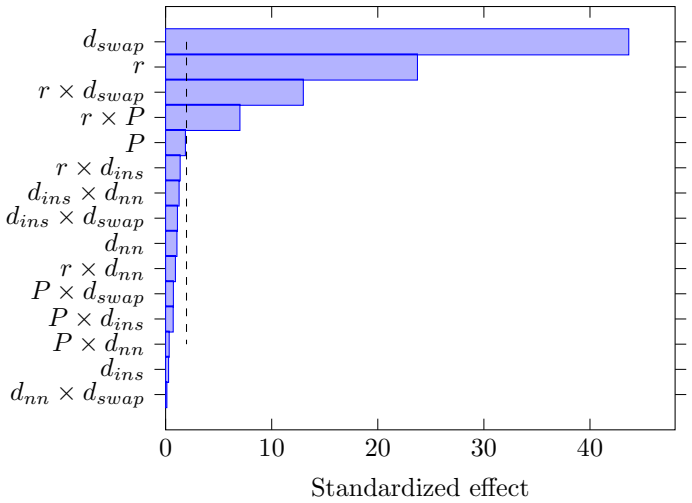

Fig. 6 Pareto chart that displays the standardized effects in decreasing order of significance. The vertical dashed line indicates the significance level γ, after standardization. Effects above γ are statistically significant

standardized effect, the larger the relative impact of a parameter change on the average RPD. Any bars beyond the vertical dashed line are statistically significant at the selected significance level γ, which is set at 5%. In this case, of the 5 main effects, only the mutation rate r and the depth of the swap neighbourhood d_{swap} are significant. Besides main effects, interaction effects are also estimated. The interaction between two effects is indicated by a '×' sign in Fig. 6. Here, the interaction effect between the mutation rate r and the depth of the swap neighborhood d_{swap} as well the interaction effect between the mutation rate r and the population size P appear to be significant.

Figure 7 shows how each of the 5 parameters affects the average RPD. The lines indicate the estimated change in average RPD as each parameter is moved from its low (left) to its high level (right), with all other parameters held constant at a value midway between their lows and their highs. Note that the parameters with a higher significance (Fig. 6) have a larger impact on the average RPD than the others.

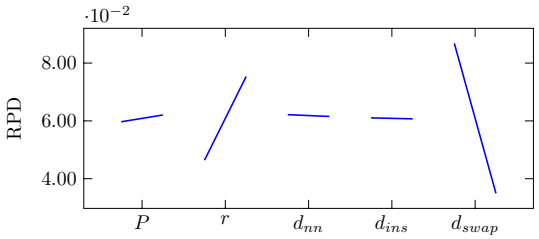

Fig. 7 The estimated effect of each of the 5 tuneable parameters on the average RPD

In Fig. 8, the estimated interaction effect of the mutation rate r and the depth of the swap neighborhood d_{swap} is shown. The impact of increasing the mutation rate is much more pronounced when the depth of the swap neighborhood is low. Clearly, a low mutation rate and a high d_{swap} is expected to result in the lowest RPD value. This configuration is in line with the observations in Fig. 7. For each parameter, the level that yields the lowest RPD is selected. In Table 3, these values are indicated in bold face.

Finally, to validate the calibration, the algorithm is applied to the remaining 15 problem instances that were not used for the calibration experiments, using the initial as well as the optimized values. Notice that the use of the same 10 instances to test the calibrated algorithm would result in a biased estimate of the performance. This is why we consider the remaining instances in validating the calibration. Again, 10 independent runs are performed for each parameter configuration and instance. The results are shown in Table 4. A comparison between the RPD before and after calibration shows an improvement for all instances except for instances 14 and 18. Nevertheless, on average the RPD is decreased by a factor of two. Now that the performance with the calibrated parameter configuration is validated, this configuration will be used throughout the remainder of this study.

6.4 Benchmarking the proposed HGA against the MILP formulation

The objective of this section is to evaluate the performance of the proposed HGA by making a comparison with the MILP formulation. We focus on instances that can be considered as small size (relative to industry-scale applications), because this allows us to efficiently obtain (close to) optimal solutions of the MILP formulation to benchmark against the proposed HGA. The following combinations

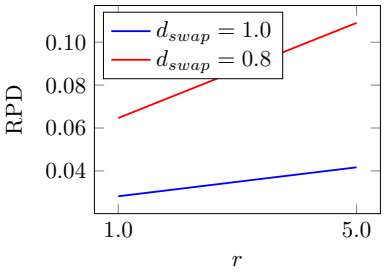

Fig. 8 The estimated interaction between the mutation rate r and the depth of the swap neighborhood d_{swap}

Table 4 Average relative proportional deviation (RPD) for the HGA with manual (HGA_{man}) and calibrated parameter settings (HGA_{cal})

Instance	HGA_{man}	HGA_{cal}
2	0.288	0.093
5	0.009	0.005
7	0.100	0.026
9	0.008	0.0007
11	0.064	0.036
14	0.032	0.032
15	0.020	0.010
17	0.147	0.075
18	0.016	0.024
19	0.038	0.028
21	0.014	0.003
22	0.058	0.043
23	0.106	0.064
24	0.014	0.0003
25	0.482	0.223
Average	0.061	0.032

of number of jobs j, number of machines m, and number of available resources ar are considered: $j \in \{8, 12, 16, 20\}$, $m \in \{1, 2, 3\}$ and $ar \in \{1, 2, 3\}$. Production activities can be scheduled 24 h per day. Maintenance resources are available on the first 7 days between 6:00 and 20:00. Every machine is subject to one maintenance activity and requires one resource during maintenance. One instance is generated for every possible combination of jobs, machines, and maintenance resources. This results in a total of 24 problem instances (i.e., it is redundant to have more maintenance resources than the number of machines, so these cases are not considered).

In the literature, synthetic instances are frequently used, where the process times of jobs and setup times between two jobs are randomly generated according to a chosen probability distribution. In our paper, we use real-world problem instances from Nexperia (see Sect. 6.1 for a detailed description) to generate a set of small-size (i.e., up to 3 machines and 20 jobs) instances by first pooling all jobs from the real-life instances together and then randomly sampling a certain number of jobs from this pool. As the process and setup times correspond to the actual values, the realistic relation between the jobs is sustained. The maintenance activities are generated similarly. The resulting set of instances are publicly available at https://git.io/J8ny2. An advantage of using real-life instances is to retain realistic relationships between the setup times of various job types. Consider an example with three jobs on the same machine. In practice, there is often a clear explainable relation between $s_{1,1,2}$, $s_{1,2,3}$ and $s_{1,1,3}$, and in particular the setup times will obey the triangular inequality $s_{1,1,3} \leq s_{1,1,2} + s_{1,2,3}$ (Kim et al. , 2002). This is also what we observe in the real-life instances from Nexperia. By using real-life instances in our computational experiments, we avoid setup times that are unlikely to occur in practice, as illustrated in Fig. 9.

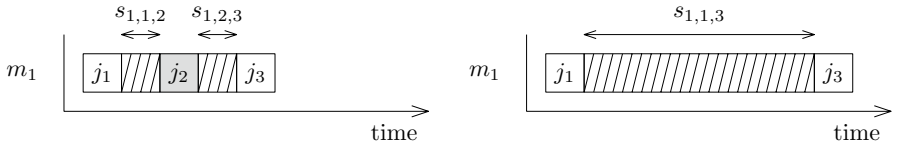

Fig. 9 Illustration of the setup-time realizations that violate the triangular inequality

The MILP formulation is implemented in Gurobi Optimizer (version 9.0.2). All parameters are set to default values, as recommended in Gurobi Optimization (2020), except for the integer feasibility tolerance, which is set to 1e-8. The solver is terminated when either a proven optimal solution is found, or when 1 h of computational time has elapsed.

In this subsection, for the objective function, we focus on both the minimization of the makespan and the minimization of the weighted average of total completion time and tardiness. In the following section (Sect. 6.5), where we focus on solving large industry-scale instances with the HGA, we will solely focus on the minimization of the weighted sum of TMCT and TT.

In Table 5, the instances are grouped by the number of available resources, number of machines, and number of jobs. For each instance, the objective value of the best found solution (the incumbent) and a lower bound is reported. The percentage difference between the incumbent and the lower bound, the so-called optimality gap, is also shown. In cases where the optimality gap is 0%, optimality is proven. In the subsequent columns, the time at which the incumbent solution is found by Gurobi, and the time at which Gurobi is terminated are reported. The latter is smaller than 3600 s when optimality is proven before the time limit is reached, and equal to 3600 s otherwise. The HGA is also applied to the same instances and terminated after 20 s of computational time. The best found solution after 20 s, the incumbent, as well as the time at which this solution is discovered are reported in Table 5.

Additionally, the percentage difference $\Delta\%$ in the objective function between the incumbent found by Gurobi Optimizer and the HGA is shown in the rightmost column.

For the makespan objective, Gurobi Optimizer is able to prove optimality well within the computational time limit for all instances that contain up to 12 jobs. For instances with 16 jobs, optimality is only proven for the cases with 1 and 3 machines, whereas optimality is not proven for any of the instances with 20 jobs. It can be seen that the time required to prove optimality increases steeply with the number of jobs. On the contrary, the number of machines and resources appears to be of minor influence. For all instances, the HGA is able to find a solution with either the same or a better objective value. In cases where a solution with the same objective value is found, the HGA outperforms Gurobi Optimizer in terms of computational time. Interestingly, for the weighted average of total completion time and tardiness objective, Gurobi Optimizer performs worse compared to the makespan objective, as it requires substantial computation time to reduce the gap. Already for the smallest instances, it is unable to find optimal solutions. The HGA takes a couple

Table 5 Performance of Gurobi Optimizer and the HGA for small-size problem instances; *ar* is the number of maintenance resources available per day, *m* equals the number of machines, and *j* equals the number of jobs

ar	*m*	*j*	Gurobi					HGA		
			Incumbent	Lower bound	Gap (%)	Time Inc. (s)	Time (s)	Incumbent	Time (s)	Gurobi Δ (%)
Makespan										
1	1	8	137.479	137.479	0	0.091	2.605	137.479	0.009	0
		12	263.235	263.235	0	2.180	416.281	263.235	0.013	0
		16	372.380	372.380	0	5.301	2748.641	372.380	0.027	0
		20	434.011	431.211	0.645	2449.814	3600	434.011	0.067	0
	2	8	83.540	83.540	0	1.062	2.150	83.540	0.007	0
		12	136.718	136.718	0	3.279	182.033	136.718	0.012	0
		16	192.122	190.573	0.806	2251.114	3600	192.105	4.927	+ 0.009
		20	241.897	237.627	1.765	2561.727	3600	241.724	0.042	+ 0.071
	3	8	67.149	67.149	0	1.273	3.079	67.149	0.007	0
		12	90.724	90.724	0	261.798	354.650	90.724	0.115	0
		16	121.721	121.721	0	257.265	1494.423	121.721	0.241	0
		20	188.193	184.772	1.817	3094.947	3600	187.980	0.853	+ 0.113
2	2	8	83.540	83.540	0	1.236	1.965	83.540	0.009	0
		12	136.718	136.718	0	7.417	316.968	136.718	0.014	0
		16	192.105	189.327	1.446	2775.702	3600	192.105	1.229	0
		20	241.787	237.289	1.895	3312.489	3600	241.724	0.297	+ 0.026
	3	8	67.149	67.149	0	1.382	3.495	67.149	0.021	0
		12	90.724	90.724	0	7.544	398.798	90.724	0.863	0
		16	121.721	121.721	0	2143.957	2228.154	121.721	1.882	0
		20	187.980	184.620	1.787	2947.100	3600	187.980	8.444	0
3	3	8	67.149	67.149	0	1.390	3.662	67.149	0.017	0

Table 5 (continued)

ar	*m*	*j*	Gurobi					HGA		
			Incumbent	Lower bound	Gap (%)	Time Inc. (s)	Time (s)	Incumbent	Time (s)	Gurobi Δ (%)
		12	90.724	90.724	0	34.128	334.710	90.724	0.475	0
		16	121.721	121.721	0	789.157	1791.145	121.721	1.169	0
		20	188.193	184.694	1.859	3384.159	3600	187.980	11.134	+ 0.113
Multi-objective										
1	1	8	137.479	137.479	0	2.541	23.605	137.479	0.010	0
		12	442.889	63.767	85.698	340.876	3600	440.106	0.112	+ 0.628
	2	8	164.782	94.795	42.514	2300.876	3600	162.949	0.029	+ 1.112
		12	279.738	44.292	84.281	80.980	3600	270.682	0.128	+ 3.346
	3	8	187.314	119.467	36.221	2870.781	3600	184.648	0.037	+ 1.423
		12	276.768	229.416	17.1	920.816	3600	267.558	0.060	+ 3.328

of milliseconds to find solutions that are better than the incumbents found by Gurobi after 1 h of computation. As Gurobi is unable to find any solutions that are close to the solutions found by the HGA and since the gap between the Incubent and Lower bound is very high for the instances with 12 jobs, only the results for the instances with up to 12 jobs are shown in Table 5.

6.5 Flexible versus fixed maintenance scheduling

In this section, an attempt is made to quantify the benefit of integrated production and maintenance scheduling. Currently, within Nexperia, the maintenance schedule is determined beforehand. This schedule is then communicated to the production planners who consider the maintenance activities as immutable unavailability periods and are required to schedule production along these intervals. For each of the historic problem instances from Table 2, it is known when the maintenance activity was scheduled. To mimic the current scenario, these activities are fixed and the proposed HGA is only used to schedule jobs along these intervals. This is referred to as fixed maintenance scheduling. Alternatively, all the constraints concerning maintenance scheduling outlined in Sect. 6.1 can be included in the HGA to schedule production and maintenance activities simultaneously, referred to as flexible maintenance scheduling. For both scenarios, the calibrated HGA is applied to solve all the available problem instances. The length of the daily time interval for maintenance activities is 14 h, and there is 1 resource available, i.e. only a single maintenance activity can be scheduled at a time (except for instances 2, 3, 4 and 6 where 2 resources are available). For each instance, 10 independent executions are performed with different random seeds (this appeared to be enough as the variability of the objective value over the runs proved to be sufficiently low). For each instance, the RPD with respect to the best known solution, the total machine completion time and the total tardiness are determined. The best known solutions are the results obtained after running the HGA for a duration of 72 h, which is the number at which full convergence was reached for most instances and improvements were observed to be extremely small. The results are averaged over the executions and reported in Table 6.

For all the instances, the solution with the lowest objective value is always found with flexible maintenance scheduling. This is clearly reflected in the RPD, as flexible maintenance scheduling always results in a lower RPD than its fixed maintenance counterpart. For the vast majority of the instances, flexible maintenance scheduling enables an improvement in terms of the total machine completion time as well as the total tardiness, when compared to fixed maintenance scheduling. In some cases, one of the constituents is improved at the cost of the other. Nevertheless, the sum of both terms decreases with respect to fixed maintenance scheduling.

The instances contain a variable number of maintenance activities. It can be hypothesized that the potential benefit of flexible maintenance scheduling increases with the number of maintenance activities. From Table 6 it is difficult to establish to what extent the potential benefit is related to the number of maintenance activities. Therefore, without performing any additional experiments, the instances are

Table 6 Average relative proportional deviation (RPD) of the objective value and the average total machine completion time (TMCT) and total tardiness (TT) constituents, for both fixed and flexible maintenance scheduling

Instance	Fixed			Flexible			Improvement (%)	
	RPD	TMCT	TT	RPD	TMCT	TT	Δ TMCT	Δ TT
1	0.186	6950.404	19.671	0.029	6941.370	17.759	0.130	9.720
2	0.205	8022.060	266.429	0.007	8013.614	258.503	0.105	2.974
3	0.732	5502.116	173.932	0.040	5492.669	144.383	0.172	16.989
4	1.377	4898.571	2419.180	0.0007	4902.117	2316.300	−0.072	4.253
5	0.232	2986.727	0.429	0.075	2979.631	2.857	0.237	−565.967
6	0.677	7679.730	2798.669	0.010	7654.057	2754.890	0.334	1.564
7	0.113	5194.831	141.586	0.040	5224.029	108.497	−0.562	23.370
8	0.278	5170.487	1648.219	0.025	5167.389	1634.149	0.060	0.854
9	0.483	4810.745	2391.869	0.001	4820.277	2347.796	−0.198	1.843
10	0.276	6199.026	277.204	0.041	6201.221	259.849	−0.035	6.261
11	0.046	6520.648	0.470	0.029	6519.978	0.014	0.010	97.021
12	0.324	7679.364	4155.068	0.035	7679.276	4121.096	0.001	0.818
13	0.578	6710.781	4264.405	0.031	6697.617	4217.836	0.196	1.092
14	0.281	7256.333	2626.895	0.044	7261.430	2598.462	−0.070	1.082
15	0.082	6135.500	743.208	0.023	6132.624	741.972	0.047	0.166
16	0.647	6024.432	884.720	0.008	6005.255	860.011	0.318	2.793
17	0.191	6978.413	1852.926	0.078	6976.237	1845.167	0.031	0.419
18	0.209	5637.955	4762.196	0.021	5632.485	4748.123	0.097	0.296
19	0.391	5882.236	707.174	0.037	5861.711	704.461	0.349	0.384
20	0.199	5311.549	112.502	0.058	5301.360	115.061	0.192	−2.275
21	0.006	6775.688	4740.624	0.002	6775.371	4740.510	0.005	0.002
22	0.094	6221.749	13.485	0.038	6221.069	10.674	0.011	20.845
23	0.090	1983.686	0.000	0.053	1982.966	0.000	0.036	0
24	0.374	2251.714	2799.280	0.0004	2250.606	2781.606	0.049	0.631
25	0.802	2026.039	383.552	0.247	2025.189	371.141	0.042	3.236
Average	0.355	5632.431	1527.348	0.039	5628.782	1508.045	0.065	1.264

grouped according to the number of maintenance activities, which varies between 1 and 4. For each group, the average RPD is calculated in the case of fixed and flexible maintenance scheduling. Then, for each group, the average difference between these values is determined, as is shown in Fig. 10.

Clearly, an increasing trend is observed, i.e. the more maintenance activities, the larger the potential improvement.

6.6 The effect of maintenance-related constraints on performance measures

The previous experiment demonstrated the potential benefit of integrated maintenance and production scheduling. This was done in accordance with

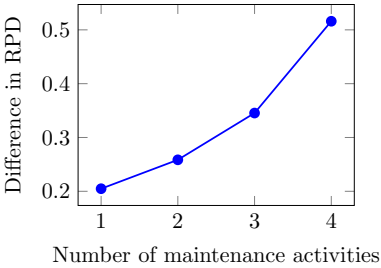

Fig. 10 The average difference in RPD between fixed and flexible maintenance scheduling versus the number of maintenance activities

maintenance-related constraints currently in place at Nexperia. From a managerial point of view, it is valuable to know whether the relaxation of these constraints enables further improvement in terms of the objectives. In our case study, this relaxation is possible in two ways: (i) extending the working hours of the maintenance personnel and (ii) increasing the head count of the maintenance personnel.

In our analysis, the intervals are increased from 14 to 24 h with steps of 2 h. Note that in the ultimate case of 24 h, maintenance personnel is always available, i.e. there are no intervals and maintenance can be scheduled at any time during the schedule horizon. The number of resources is varied from 1 to 4.

For each combination of constraints, as for the previous experiments, 10 independent runs of the HGA are performed at each instance. The average TT and TMCT over all instances and executions are shown in Fig. 11 for each combination of constraints. It is important to note that increasing the number of available resources beyond the number of maintenance activities in an instance will not have any effect. Hence, each point in Fig. 11 includes only those instances for which the number of maintenance activities is higher or equal to the number of available resources.

Clearly, relaxing either one of the two constraints results in lower objective values. The effect of increasing the number of available resources appears to be stronger compared to stretching the working hours. The results show that significantly more

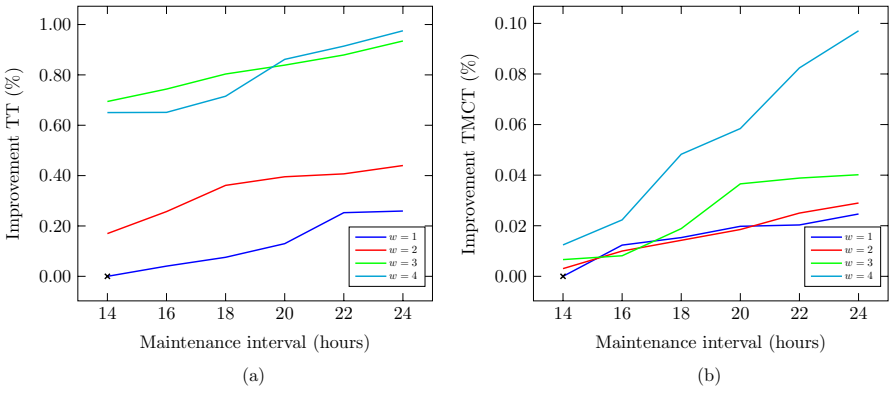

Fig. 11 TT and TMCT versus the daily working hours for various numbers of resources w. The black cross represents the improvement obtained from the case study in Sect. 6.5 shown in Table 6

gain can be achieved in terms of TT, compared to TMCT. Interestingly, increasing the resources appears to have a larger effect on TT than on TMCT. A significant difference is only noticeable for TMCT in the case of four resources.

It is also interesting to see that when the working hours are relatively low, the effect of adding an additional resource is small only for TMCT. These results provide valuable insights for managers and indicate directions for further improvements. Based on this analysis, practical recommendations would be to increase the workforce of maintenance if reducing total tardiness is the goal. Another suggestion is to extend the working hours of maintenance only if the workforce is already expanded, as only then extending the working hours will pay off. We find that a reduction of up to 0.165% and 2.264% in total machine completion time and total tardiness, respectively, can be realized. This is derived from adding the numbers in Table 6 to the maximum realizable improvement from Fig. 11. In our case study, these results can be translated to an approximate reduction of 50 h in total machine completion time and 207 h in total tardiness for the complete factory per week, showing the potential gain associated with relaxing the maintenance constraints in practice. To put these numbers into perspective, 50 h of total machine completion time corresponds to the output that can be produced by

one-third of an assembly line per produced schedule, where the scheduling horizon is typically one week. Therefore, 50 h of capacity can be made available per week. As the costs of such an assembly line are typically multiple millions of dollars throughout the entire lifetime, the savings by adopting the smart maintenance scheduling approach proposed in this work can save multiple millions of dollars in the long run. In addition, the company considered in this use case, Nexperia, produces hundreds of billions of products per year. Without expanding in new assembly lines, tens of millions of additional products can be produced per year. On top of that, they can be produced with 207 h less tardiness on a weekly basis, resulting in higher customer satisfaction and more accurate targets downstream in the supply chain process.

7 Conclusions

Motivated from practice, this paper considers the problem of simultaneously scheduling resource-constrained maintenance activities and production jobs for unrelated parallel machines. The novel elements of the problem come from the constraints related to maintenance activities. In particular, each maintenance activity that is to be scheduled on a machine holds a multitude of possible windows in which it can be scheduled, and the maximum number of maintenance activities that can be scheduled in each of these windows is limited by a scarce shared resource. In addition, sequence- and machine-dependent setup times, due dates and eligibility constraints are addressed. The makespan and a weighted sum of total machine completion time and total tardiness are considered as objectives. A Mixed Integer Linear Programming (MILP) formulation is presented, and a Hybrid Generic Algorithm (HGA) is developed to be able to solve industry-scale large instances. A real-world case study is performed where the factory practice of a semiconductor manufacturer is

compared to the integrated maintenance and production scheduling approach proposed in this work. Experiments with real-world factory data show that the fixed-maintenance scheduling approach of the factory can be substantially improved by adopting flexible-maintenance scheduling with the proposed approach, and the benefit grows as the number of maintenance activities increases. The benefit from flexible-scheduling is also quantified as the maintenance window is stretched or when more resources are added.

In our case study, data sets only included instances in which at most one maintenance activity per machine needs to be scheduled. While our HGA is capable of scheduling multiple maintenance activities per machine, the MILP formulation has not been designed with this property. It would be interesting to extend the MILP and gather real-life instances such that multiple maintenance activities per machine are scheduled. Another suggestion for future research would be to apply different solution methods on this novel integrated production and maintenance scheduling problem, and use the data set provided in this study for benchmarking purposes to seek the algorithm characteristics that best suit this problem type. Our final suggestion for future research concerns the setups. Although a setup does not require specialized personnel, it does require human effort and is usually performed by the machine operator. However, time spent on a setup may be shortened significantly when the operator is assisted by another operator. This is a practically relevant extension and a challenging problem to solve.

Declarations

Conflict of interest The authors declare no conflict of interest (financial or non-financial).

Human and animal rights The research does not involve any data collected from human participants or experiments with animals.

References

Adan J (2022) A hybrid genetic algorithm for parallel machine scheduling with setup times: a comparative study of metaheuristics on large problem instances. J Intell Manuf 33(7):2059–2073

Afzalirad M, Rezaeian J (2016) Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. Comput Ind Eng 98:40–52

Allahverdi A (2015) The third comprehensive survey on scheduling problems with setup times/costs. Eur J Oper Res 246(2):345–378
Arnaout JP (2020) A worm optimization algorithm to minimize the Makespan on unrelated parallel machines with sequence-dependent setup times. Ann Oper Res 285(1):273–293
Avalos-Rosales O, Angel-Bello F (2015) Efficient metaheuristic algorithm and re-formulations for the unrelated parallel machine scheduling problem with sequence and machine-dependent setup times. Int J Adv Manuf Technol 76(9):1705–1718
Avalos-Rosales O, Angel-Bello F, Álvarez A, Cardona-Valdés Y (2018) Including preventive maintenance activities in an unrelated parallel machine environment with dependent setup times. Comput Ind Eng 123:364–377
Beaton C, Diallo C, Gunn E (2016) Makespan minimization for parallel machine scheduling of semi-resumable and non-resumable jobs with multiple availability constraints. INFOR Inform Syst Oper Res 54(4):305–316
Bitar A, Dauzère-Pérès S, Yugma C (2021) Unrelated parallel machine scheduling with new criteria: complexity and models. Comput Oper Res 132:105291
Chang PC, Chen SH (2011) Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. Appl Soft Comput 11(1):1263–1274
Chen YY, Huang PY, Huang CJ, Huang SQ, Chou FD (2021) Makespan minimization for scheduling on two identical parallel machiens with flexible maintenance and nonresumable jobs. J Ind Prod Eng 38(4):271–284
Costa A, Cappadonna FA, Fichera S (2016) Total tardiness minimization in a parallel machine system with flexible periodic maintenance. J Ind Prod Eng 33(7):485–494
Dantzig G (1948) Programming in a linear structure. Comptroller, United States Air Force, Washington DC
Diana ROM, de Souza SR (2020) Analysis of variable neighborhood descent as a local search operator for total weighted tardiness problem on unrelated parallel machines. Comput Oper Res 117:104886
Eroglu DY, Ozmutlu HC (2017) Solution method for a large-scale loom scheduling problem with machine eligibility and splitting property. J Textile Inst 108(12):2154–2165
Fisher RAS, Yates F (1963) Statistical tables for biological, agricultural, and medical research (6th ed., rev. and enlarged ed.). Edinburgh: Oliver and Boyd
Fowler J, Horng SM, Cochran JK (2003) A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence dependent setups. Int J Ind EngTheory Appl Pract 10(3):232–243
Gedik R, Rainwater C, Nachtmann H, Pohl EA (2016) Analysis of a parallel machine scheduling problem with sequence dependent setup times and job availability intervals. Eur J Oper Res 251(2):640–650
Graves G, Lee C (1999) Scheduling maintenance and semiresumable jobs on a single machine. Nav Res Logist 46(7):845–863
Gurobi Optimization 2020. Gurobi optimizer reference manual- version 9.0. http://www.gurobi.com, accessed 12th March 2020
Hashemian N, Diallo C, Vizvári B (2014) Makespan minimization for parallel machines scheduling with multiple availability constraints. Ann Oper Res 213(1):173–186
Hidri L, Alqahtani K, Gazdar A, Badwelan A (2021) Integrated scheduling of tasks and preventive maintenance periods in a parallel machine environment with single robot server. IEEE Access 9:74454–74470
Hoseinpour Z, Kheirkhah AS, Fattahi P, Taghipour M (2020) The problem solving of bi-objective hybrid production with the possibility of production outsourcing through meta-heuristic algorithms. Management 4(2):1–17
Hoseinpour Z, Taghipour M, Beigi JH, Mahboobi M (2021) The problem solving of bi-objective hybrid production with the possibility of production outsourcing through imperialist algorithm, NSGA-II, GAPSO hybrid algorithms. Turkish J Compu Math Educ 12(13):8090–8111
Kaabi J, Harrath Y (2014) A survey of parallel machine scheduling under availability constraints. Int J Comput Inform Technol 3(2):238–245
Kaabi J, Harrath Y (2019) Scheduling on uniform parallel machines with periodic unavailability constraints. Int J Prod Res 57(1):216–227
Kim DW, Kim KH, Jang W, Chen FF (2002) Unrelated parallel machine scheduling with setup times using simulated annealing. Robot Comput Integr Manuf 18(3):223–231
Lee C, Chen Z (2000) Scheduling jobs and maintenance activities on parallel machines. Nav Res Logist 47:145–165

Lee JY, Kim YD, Lee TE (2018) Minimizing total tardiness on parallel machines subject to flexible maintenance. Int J Ind Eng 25(4):472–489
Lee WC, Wang JY, Lee LY (2015) A hybrid genetic algorithm for an identical parallel-machine problem with maintenance activity. J Oper Res Soc 66(11):1906–1918
Li G, Liu M, Sethi SP, Xu D (2017) Parallel-machine scheduling with machine-dependent maintenance periodic recycles. Int J Prod Econ 186:1–7
Liao LW, Sheen GJ (2008) Parallel machine scheduling with machine availability and eligibility constraints. Eur J Oper Res 184(2):458–467
Mellouli R, Sadfi C, Chu C, Kacem I (2009) Identical parallel-machine scheduling under availability constraints to minimize the sum of completion times. Eur J Oper Res 197(3):1150–1165
Montgomery DC, Runger GC (2010) Applied statistics and probability for engineers. Wiley, Hoboken
Rebai M, Kacem I, Adjallah KH (2013) Scheduling jobs and maintenance activities on parallel machines. Oper Res Int J 13(3):363–383
Santos HG, Toffolo TA, Silva CL, Van den Berghe G (2019) Analysis of stochastic local search methods for the unrelated parallel machine scheduling problem. Int Trans Oper Res 26(2):707–724
Seif J, Dehghanimohammadabadi M, Yu AJ (2020) Integrated preventive maintenance and flow shop scheduling under uncertainty. Flex Serv Manuf J 32(4):852–887
Soni N, Kumar T (2014) Study of various mutation operators in genetic algorithms. Int J Comput Sci Inform Technol 5(3):4519–4521
Sun K, Li H (2010) March. Scheduling problems with multiple maintenance activities and non-preemptive jobs on two identical parallel machines. Int J Prod Econ 124:151–158
Touat M, Bouzidi-Hassini S, Benbouzid-Sitayeb F, Benhamou B (2017) A hybridization of genetic algorithms and fuzzy logic for the single-machine scheduling with flexible maintenance problem under human resource constraints. Appl Softw Comput 59:556–573
Tran T, Beck J (2012) Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. 20th European Conference on Artificial Intelligence: 774–780
Vallada E, Ruiz R (2011) A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. Eur J Oper Res 211(3):612–622
Wang IL, Wang YC, Chen CW (2013) Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics. Flex Serv Manuf J 25(3):343–366
Yang D, Hung C, Hsu C (2002) Minimizing the Makespan in a single machine scheduling problem with a flexible maintenance. J Chin Inst Ind Eng 19(1):63–66
Yang S, Ma Y, Xu D, Yang J (2011) Minimizing total completion time on a single machine with a flexible maintenance activity. Comput Oper Res 38(4):775
Yang XS (2014) Nature-Inspired Optimization Algorithms. Elsevier, Oxford
Yoo J, Lee I (2016) Parallel machine scheduling with maintenance activities. Comput Ind Eng 101:361–371

Michael Geurtsen is a PhD student in the Department of Operations, Planning, Accounting and Control at Eindhoven University of Technology. He received his BSc and MSc Cum Laude in Mechanical Engineering at the Eindhoven University of Technology. Currently, he conducts research on predictive maintenance and the integration of maintenance, production and resource planning in a semiconductor shop-floor. His current research interests are in the area of production, maintenance and resource scheduling/planning, where he aims to apply discrete event simulation, digital twins, machine learning and deep reinforcement learning tools.

Jelle Adan is a PhD graduate from the School of Industrial Engineering at Eindhoven University of Technology. He has extensive experience with factory automation in the semiconductor industry across Europe and Southeast Asia. He is a co-founder of Atlas4, a forward-thinking company dedicated to the development and implementation of data-driven optimization solutions using AI to enhance the efficiency of manufacturing processes.

Alp Akcay is an Associate Professor in the Department of Industrial Engineering and Innovation Sciences at Eindhoven University of Technology. He received his Ph.D. in Operations Management and Manufacturing from Carnegie Mellon University. His research interests include data-driven decision making under uncertainty with applications in semiconductor manufacturing systems and supply chains. He is an Associate Editor for the Journal of Simulation and serves as a track chair for the "Manufacturing & Industry 4.0 applications" track of the Winter Simulation Conference.