The International Journal of Advanced Manufacturing Technology (2022) 119:3729–3745
https://doi.org/10.1007/s00170-021-08163-3

ORIGINAL ARTICLE

Job shop rescheduling with rework and reconditioning in Industry 4.0: an event-driven approach

Gonzalo Mejía[1] · Carlos Montoya[2] · Stevenson Bolívar[2] · Daniel Alejandro Rossit[3,4]

Received: 27 July 2021 / Accepted: 29 September 2021 / Published online: 8 January 2022

Abstract
In this paper, we investigate the impact of rescheduling policies in the event of both rework and reconditioning in job shop manufacturing systems. Since these events occur in unplanned and disrupting manner, to address them properly, it is required to manage real-time information and to have flexible reaction capacity. These capabilities, of data acquisition and robotics, are provided by Industry 4.0 Technologies. However, to take full advantage of those capabilities, it is imperative to have efficient decision-making processes to deliver adequate corrective actions. In this sense, we propose an event-driven rescheduling approach. This approach consists of an architecture that integrates information acquisition, optimization process, and rescheduling planning. We study the performance of the system with several algorithms with two performance criteria, namely, (i) relative performance deviation (RPD) in terms of objective function and (ii) schedule stability. We also propose a hybrid policy that combines full rescheduling regeneration with stability-oriented strategies aimed to balance both criteria. We conducted extensive computational tests with instances from the literature under different scenarios. The results show that a sophisticated algorithm can obtain better quality schedules in terms of the objective function but at the expense of sacrificing stability. Finally, we analyze and discuss the results and provide insights for its use and implementation.

Keywords Event-driven rescheduling · Job shop · Manufacturing systems · Rework · Petri nets

1 Introduction

In recent years, important changes have taken place in the production paradigm associated with the Industry 4.0 concept [1]. This change has implied a greater penetration of digital technologies in the productive processes associated with a growing connectivity through the Internet of Things (IoT) [2]. The increasing responsiveness of Industry 4.0 is one of its main attributes [1]. Increasing responsiveness implies, roughly speaking, being able to fast-adjusting the manufacturing system itself in a non-rigid way to the scenarios and situations that it must face [3].This enables the production capacity to be boosted in the face of changing scenarios, either due to changes in demand [4] or due to the occurrence of unexpected events in the process [5].

These Industry 4.0 features directly impact the decision-making processes associated all levels of a manufacturing system, from resource and facility planning to shop floor control [6]. In particular, a quick and efficient response to disruptive events is of primary importance. There are two pre-requisites: on one hand, the ability to capture in real-time and process data related to the disruptions and on the other hand, to adapt the system to the new disrupted situation.

Rescheduling is a crucial aspect in the Industry 4.0 era [3]. Rescheduling relates the decisions made when an already existing production plan is in execution [7]. Once the schedule has been generated, unforeseen events will arise

✉ Gonzalo Mejía
gonzalo.mejia@unisabana.edu.co

Carlos Montoya
c_montoya@javeriana.edu.co

Stevenson Bolívar
s_bolivar@javeriana.edu.co

Daniel Alejandro Rossit
daniel.rossit@uns.edu.ar

1 Faculty of Engineering, Universidad de La Sabana, Campus Universitario Puente del Común, Chía, Colombia

2 Facultad de Ingeniería, Pontificia Universidad Javeriana, Bogotá, Colombia

3 Departamento de Ingeniería, Universidad Nacional del Sur, Bahía Blanca, Argentina

4 INMABB, Universidad Nacional del Sur and CONICET, Bahía Blanca, Argentina

and will modify the initial conditions, affecting the performance of the manufacturing system [8]. Rescheduling methods must therefore contemplate the modified scenario and propose actions that allow meeting the organization's objectives [7]. Two key factors for rescheduling are the speed of the calculations and the quality of the information required to devise a corrective action, as well as the ability to provide quality and stable schedules.

Industry 4.0 affects these two aspects directly [9]. On one hand, the availability of more and better information will improve the ability to analyze new scenarios and on the other hand, quality schedules can provide a competitive advantage [10]. Since the information arrives more quickly and efficiently to the shop supervisors, the control actions can also be quickly fed back to the shop floor. In this sense, Industry 4.0 enhances the potential for the development and implementation of sophisticated methods for recalculating. In their work, Zhang et al. [3] highlighted that the management of these dynamic environments in Industry 4.0 poses a challenge for the rescheduling mechanisms in the shop floor. Naturally, developing more sophisticated methods will allow a better performance than simpler methods such as dispatching rules or right shift schedules.

However, to take advantage of the Industry 4.0 Technologies, there are several issues unresolved: the first question concerns the appropriate tools to model and optimize production plans carried out by Cyber-physical Systems. Most tools nowadays provide either (i) powerful tailor-made algorithms for unrealistic and simplified representations of production systems or, (ii) accurate representations connected with limited simulation-based optimization methods based mostly on priority or probabilistic decision rules. Most of these approaches consider the uncertainty proper of rework events only through expected values or Monte-Carlo methods. However, modelling rework events in such manners can undermine severely system performance, since each rework event can differ significantly from others; then, an event-driven approach enables a more accurate and efficient management of disrupting events. The second question is the selection of appropriate optimization methods with rescheduling capabilities. It is well known that rescheduling not only involves decisions as to maintain the quality of the schedule but also related to stability. It is also clear that frequent re-calculation of schedules leads to system instability that affects the supply of materials and components, human resources and labor (e.g., shifts already committed), and the fulfilment of production orders [11].

In this paper, we study rework in job shop manufacturing systems with Industry 4.0 capabilities. We propose an integrated modeling and optimization approach able to handle rescheduling in real-time considering an event-driven logic. Rework in manufacturing is one of the many disruptions that affect their performance and it is one of the least studied. Rework occurs due to many causes that include worker errors, quality problems of raw materials, incorrect machine settings, and tool breakage. An issue that it is often overlooked is the fact that rework generally leads to additional reconditioning operations: for example, in a case of a machining defect, the workpiece may need additional machining operations (e.g., grinding); in a case of a paint quality problem, the paint must be scrapped off before the workpiece is repainted; assembly errors require disassembly before the rework is performed.

In this paper, we use an integrated timed Petri nets [12, 13] modeling and optimization approach to (i) represent the manufacturing system with provisions for rework, (ii) calculate the "baseline" or offline schedule, and (iii) trigger the schedule regeneration. In turn, the rescheduling system proposed in this work lays the foundations for the design of an automated system for scheduling and rescheduling functionalities, with capabilities to process information in real-time, as well as to model and solve the problems that arise during production runs.

We aim to study both scheduling quality (in terms of RPD) and stability with different scheduling/rescheduling methods and with different rescheduling policies in manufacturing systems with Industry 4.0 Technologies able to both detect the need for rework in real-time and estimate, based on previous occurrences, the duration of the reconditioning operation. We propose a hybrid policy that triggers either a full reschedule procedure or a more conservative modification of the current schedule. The choice of the rescheduling method is made based on a predefined threshold value that is related to the expected duration of the reconditioning operation. The specific goals of this papers are (1) to compare dedicated algorithms for offline scheduling vs. the more versatile dispatching rules with the above metrics and, (2) establish the right value of threshold that will be a trade-off between RPD and stability. The contributions of this paper can be summarized as follows:

- An integrated modeling and optimization approach for real-time event-driven rescheduling under rework/reconditioning events.
- An extensive experimentation framework to compare the tested algorithms under the two metrics mentioned above.
- A rescheduling method that provides a trade-off between RPD and schedule stability

The remainder of the paper is organized as follows: Sect. 2 examines the background on the topic; Sect. 3 describes the proposed approach, Sect. 4 presents the computer tests, the results and analysis, and finally, Sect. 5 concludes and suggests future work.

2 Background

2.1 Petri nets

The modeling approach used in this paper was Petri nets. Petri nets are an excellent modeling and optimization tool for the manufacturing system and for the rework and reconditioning tasks. In general, Petri nets are well-known for modeling and simulation. However, Petri net-based scheduling algorithms have been proved powerful on a variety of manufacturing systems such as job shops, flexible manufacturing systems, and project scheduling [14, 15]. An important advantage of using Petri nets over other modeling approaches is the integration of modeling with both simulation and optimization.

In this section, we give a brief introduction of the notation and definitions of Petri nets used in this paper[1]. Formally speaking an ordinary (timed place), Petri net is a 5-tuple $G = \{P, T, F, \tau, M_0\}$, where P is a set of places, T is a set of transitions, F is the flow relation between places and transitions (i.e., arcs), τ is the set of time delays associated with places, and M_0 is the initial marking. We model a job shop system following the concept of timed S^{3}PR [12]. In S^{3}PR nets, a number of concurrent serial processes (jobs) are modeled with an equal number of acyclic connected state machines.

Places There is a unique initial place p_{sj} and a unique end place p_{ej} on each job type j. These initial and the end places represent the conditions process instance ready to start and process instance finished. $M_0(p_{sj})$ represents the number of parts (instances) of job type j. Places belonging to the serial processes are denoted as "Operational Places" (P_O) and can be timed, if representing the execution of an operation, or untimed if representing a condition such as "part in buffer," "ready," "in queue," or "finished." As in Lee and DiCesare [16], a timed place p_{ij} is associated with operation o_{ij}. In this paper, we assume that buffers of infinite capacity are located before each stage. Places representing the condition "job j at buffer prior to execution of operation o_{ij}" are denoted as b_{ij}. Resource capacity constraints are incorporated to the model via "resource places." The set $P_R = \{p_r/r \in R\}$ contains all resource places. The subset

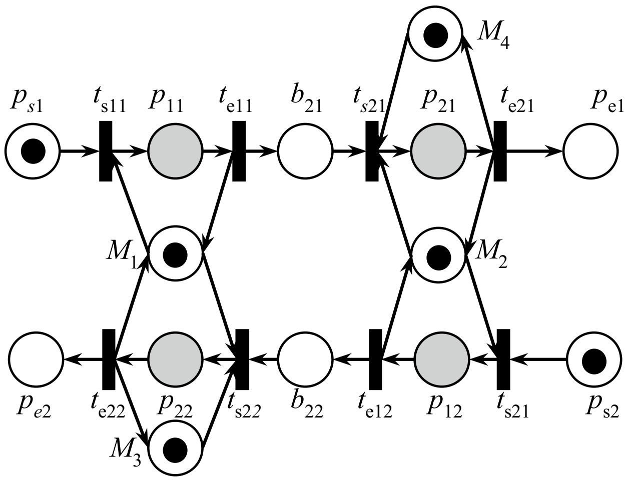

Fig. 1 Petri net Model of two jobs with two operations each

$P_{R(ij)} = \{p_r/r$ is used in operation $o_{ij}\}$ contains all resource places required in operation o_{ij}. Without loss of generality, the capacity of all resources $r\hat{I}R$ is set to 1, i.e., there is only one unit of resource of each resource type. All resource places are untimed.

Transitions $t_{(ij)s}$ and $t_{(ij)e}$ represent respectively the start and end of operation o_{ij}.

$T = \{t_{s(ij)}\} \cup \{t_{e(ij)}\}$ such that $o_{ij} \in O$. The following construction rules hold in this paper:

$\bullet t_{s(ij)} = \{p_{sj}\} \cup P_{R(ij)}$ if o_{ij} is the first operation (i.e., $i = 1$) of job type j and $\{b_{ij}\} \cup P_{R(ij)}$ otherwise; $t_{s(ij)}\bullet = \{p_{ij}\}$;

$\bullet t_{e(ij)} = \{p_{ij}\}$;

$t_{e(ij)}\bullet = \{b_{(i+1)j}, P_{R(ij)}\}$ if $i < |O_j|$ and $\{p_{ej}, P_{R(ij)}\}$ otherwise.

The S^{3}PR model of the production activities is defined here as the "*plant*" net as in Mejía et al. [17]. Figure 1 illustrates a Petri net model of the plant of two jobs with two stages each: stage 1 of job 1 (resp. job 2) has operation o_{11} (resp. o_{12}) represented by place p_{11} (resp. p_{12}). Stage 2 of job 1 (resp. job 2) has operation o_{21} (resp. o_{22}) represented by place p_{21} (resp. p_{22}). Place b_{21} and b_{22} are buffer places between stage 1 and stage 2. Places M_1 to M_4 represent the availability of machines 1 to 4.

Definition A reconditioning/rework trajectory is a sequence of actions required to finish an operation that, for instance, failed inspection and needed reprocessing. The reconditioning/rework consists of two actions: the repair (reconditioning) and the reprocessing (rework).

For the purpose of this work, reconditioning/rework trajectories are modelled with a subnet consisting of one single timed place $p_{rw(ij)}$, denoted as reconditioning place, which models the execution of the reconditioning action. Such a reconditioning place is linked to one input uncontrollable transition $t_{d(ij)}$ and to one output controllable transition

[1] Although this section can be skipped without loss of continuity, we leave it here as important aspects of the modeling approach regarding rework are presented.

$t_{c(ij)}$. These transitions $t_{d(ij)}$ and $t_{c(ij)}$ represent respectively the start and end of the reconditioning action required by operation o_{ij}. Let us assume that operation o_{ij} requires a set of resources $R_{ij} \subseteq R$ for its processing.

Rework subnets are created in real-time and attached to the "plant" subnet (i.e., the manufacturing system) whenever a rework is triggered. In real-life, a sensor will detect a defect in a workpiece and triggers an alert to the shop controller that makes a rescheduling decision depending on the duration of the corresponding reconditioning activity. The rework subnet is eliminated from the plant subnet, whenever the reconditioning action is finished.

Definitions A reconditioning subnet associated with the reconditioning of a single operation o_{ij} is defined as:

$$\mathrm{N}_{rw(i,j)} = (P_{rw(ij)}, T_{rw(ij)}, P_{rw(ij)} \times T_{rw(ij)}, T_{rw(ij)} \times P_{rw(ij)}, M_{0rw(i,j)}, \tau_{rw(i,j)})$$
in which:

$P_{rw(ij)} = \{p_{rw(ij)}\}$
$T_{rw(ij)} = \{t_{d(ij)}, t_{c(ij)}\}$
$M_{0rw(i,j)}$ is the initial marking of place $p_{rw(ij)}$
$\tau_{rw(i,j)}$ is the stochastic time delay of place $p_{rw(ij)}$.

The flow relations $P_{rw(ij)} \times T_{rw(ij)}, T_{rw(ij)} \times P_{rw(ij)}$ are the following:

$\bullet p_{rw(ij)} = \left\{t_{d(ij)}\right\}$.
$p_{rw(ij)}\bullet = \left\{t_{c(ij)}\right\}$.
$\bullet t_{d(ij)} = \left\{p_{ij}\right\}$.
$t_{d(ij)}\bullet = \left\{p_{rw(ij)} \cup p_r \in P_{R(ij)}\right\}$
$\bullet t_{c(ij)} = \left\{p_{rw(ij)}\right\}$
$t_{c(ij)}\bullet = \left\{b_{ij}\right\}$

See Fig. 2 for an example of a rework subnets. The plant, representing only production activities, is shown in Fig. 2a; the augmented net is shown in Fig. 2b. Places $p_1 - p_4$ are operational places and p_5 is a resource place representing the availability of resource r_1. Places p_1 and p_4 are respectively start and end places; p_2 represents a buffer place and place p_3 represents the execution of an operation. Transitions t_1 to $t_3 \in T$ are controllable plant transitions; transitions t_d and t_c represent respectively the start and end of the reconditioning action. All transitions t_1–t_3 and t_c are controllable whereas t_d is uncontrollable. Consider the case where place p_3 is marked: both transitions t_3 and t_d can fire. If t_d fires, the system enters into a disrupted state and can be eventually returned to a normal state when transition t_c fires. When t_c fires, the subnet is removed from the plant.

The augmented Petri net is $\mathrm{N}_A = \left(\mathrm{N}, \mathrm{N}_{rw(i,j)}\right)$. This augmented net is live and bounded [18].

2.2 Job shop scheduling problem (JSSP)

A job shop is manufacturing system in which a set M of machines, indexed in i, processes a set of jobs J, indexed in j. Each job consists of a set of operations which must be processed in a pre-defined order (route) and no pre-emption is allowed. In general, not all job routes are the same. The term o_{ij} denotes operation of job j on machine i. The deterministic processing time of operation o_{ij} is p_{ij}. The set of operations of job j is denoted as O_j.Without loss of generality, in this work, we set $\left|O_j\right| = |M| = m$. The set of all operations is denoted as $O = \bigcup O_j$.

A job shop scheduling problem (JSSP) consists of calculating the start (or end) times of all operations $o_{ij} \in O$ in such a way that an optimization criterion is minimized. A schedule S is an array of operations o_{ij} together with their corresponding start times s_{ij} sorted in the chronological order. The literature on offline JSSP and its variants is vast.

In this paper, we studied the minimization of the total flow time (TFT), which is related to the total time spent in queue. The TFT can be calculated following Eq. (1).

$$\text{Total flow time (TFT)} = \sum_{j=1}^{n} C_j \tag{1}$$

The term C_j is the completion time of job j in the original schedule and $n = |J|$ is the number of jobs. We define RPD as in Katragjini et al. [11]. The formula is:

$$RPD = \frac{TFT_e - TFT_{\text{best}}}{TFT_{\text{best}}} \tag{2}$$

where TFT_e (resp. TFT_{best}) is the value of TFT of the executed schedule (resp. the best offline schedule among all algorithms).

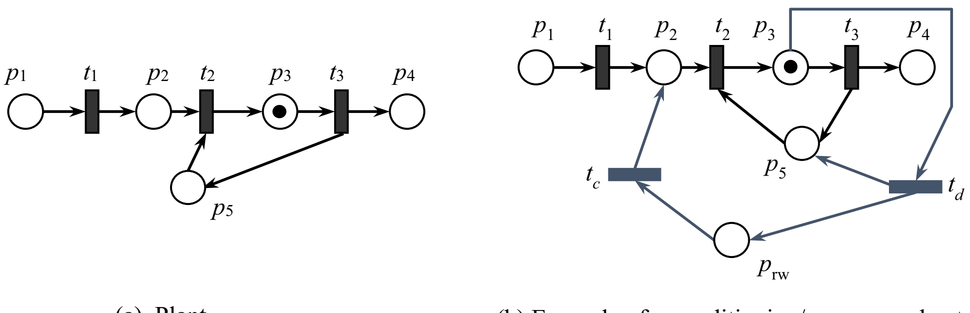

Fig. 2 Reconditioning/rework trajectories modeled with Petri nets. **a** Plant. **b** Example of reconditioning/recovery subnet

(a) Plant (b) Example of reconditioning/recovery subnet

In the rescheduling situation, we use the classical stability formula [19] as follows (Eq. 3):

$$\eta = \frac{\sum_{1}^{|O|} \sum_{o_{ij} \in O} \left| C_{ij} - C'_{ij} \right|}{\sum_{o_{ij}} p_{ij}} \tag{3}$$

where C_{ij} (resp. C'_{ij}) is the completion time of operation o_{ij} in the original schedule (resp. the executed schedule).

2.3 Rescheduling

Rescheduling is a topic that has been widely studied over the years. Efforts have been made to classify the environment, the rescheduling policies, the orksle regeneration, and the types of disruptions, and. We only will provide a brief review of the topic. Interested readers are referred to the paper of Vieira et al. [7].

Regarding the environment, these can be classified as static and dynamic: in static environments, all ork are known from the beginning whereas in dynamic environments, ork arrive continuously to the system. Schedule generation refers to whether the orksle is calculated with provisions for disruptions (i.e., robust schedules) or not (i.e., nominal schedules). In the first case, the orksle is expected to absorb most perturbations and rescheduling is orksle rarely recalculated; in a nominal orksle, rescheduling actions take place in one way or another when disruptions occur.

When disruptions occur, the three more common repair strategies to repair are full orksle regeneration, partial rescheduling, and right-shift scheduling [20]. In orksle regeneration, the entire orksle is recalculated, considering those operations that have not started yet their execution. This strategy produces the best results in terms of the objective function but requires more computer effort and produces instability. Partial rescheduling attempts to reschedule only a subset of operations. One of the better-known methods for partial rescheduling is AOR (affected operations rescheduling) in which only affected operations (those whose start times change) by the breakdown are rescheduled. The right-shift scheduling strategy consists of postponing the start of the affected operations by the amount of time of the breakdown. This is the easiest to implement and produces the highest stability but the orksle quality can be compromised [20].

In terms of policies, these can be periodic, event-driven, or hybrid. In periodic rescheduling, the orksle regeneration is performed at regular intervals whereas in event-driven approaches, the orksle is regenerated whenever disruptions occur. Hybrid rescheduling falls in between [21]. One of the questions is when to launch a full or a partial re-schedule. A common approach is to trigger a full rescheduling method orksle certain conditions on a case by case basis. For example, Pfeiffer et al. [22] used a simulation-based approach, taking into account machine breakdowns and stochastic processing times. Authors evaluated their solution approach in terms of stability and efficiency. They proposed a heuristic method for periodic rescheduling and a triggering mechanism that activates whenever the cumulative difference between the planned and simulated completion times are greater or a threshold. Later, Bidot et al. [23] investigated stochastic processing times and machine breakdowns. In that work, a orksle is generated offline and launched. A controller monitors the orksle execution and activates full, partial, or progressive rescheduling processes by checking case-specific performance conditions.

Lately, the research body has focused towards schedules that are modified whenever disruptions occur. For example, Dong and Jang [21] proposed new rescheduling methods orksle the AOR and compared their results with the classical AOR methodologies. He and Sun [24] combined robust schedules with rescheduling in flexible job shops and compared their results with other strategies in terms of stability and robustness. Katragjini et al. [11] combined stability measures with efficiency measures (i.e., makespan) in a single objective function in a flow shop manufacturing system and compared several metaheuristic algorithms. Ahmadi et al. [19] proposed a multi-objective model for flexible job shops and studied the performance of several classical algorithms (e.g., NSGA-II, NRGA). More recently, Larsen and Pranzo [25] developed a general rescheduling framework with different types of disruptions. A controller triggers a rescheduling algorithm whenever a deterministic or stochastic rescheduling condition is met. Authors evaluated their solution approach in a job shop orksle. Gao et al. [26] proposed a Jaya algorithm on a flexible job shop orksle with machine recovery. In the rescheduling phase, instability and makespan are minimized. Nie et al. [27] proposed an event-driven, rescheduling approach orksle a genetic algorithm for flexible job shop scheduling problems subject to machine breakdowns. In that work, authors focused on minimizing makespan in the scheduling phase, while in the rescheduling phase, the orksle's performance is measured by the weighted sum of RPD and stability.

In regard to Industry 4.0, Framinan et al. [28] investigated the impact of real-time information in the performance of rescheduling policies. Their results suggest that real-time information can indeed improve the performance of the manufacturing system but only under low variability conditions. Also, technological advances have enable to address scheduling problems using real-time information through Cloud architectures, as in Zhang et al. [29], where a multi-objective game-theoretic approach is developed. Similarly, Carlucci et al. [30] and Goodarzi et al. [31] implemented game theory approaches.

In other orks, Ghaleb et al. [32] proposed a real-time rescheduling system considering job shop problems and stochastic processing times. In this work, the arrival of new ork and machine breakdowns trigger the rescheduling process. The authors evaluated several rescheduling policies such as continuous and event-driven rescheduling.

In general, most orks aim to balance performance with stability measures. To do so, the different approaches adopt different strategies to launch either a full reschedule or a partial one. Our approach is different in the sense that it exploits real-time information to decide when to perform a full or a partial reschedule. On the other hand, game theoretical approaches involve agents with individual goals which is not the case in our paper. In our case, we have a single central controller with two distinct goals.

2.4 Rework

Rework in manufacturing is one of the many disruptions that affect its performance, and it is one of the least studied. The literature in rework includes aspects such as costs [33], lot sizing [34], and quality [35, 36]. Rework in manufacturing systems has been addressed from the perspectives of simulation-based dispatching rules [37, 38], and with classical heuristic and metaheuristic approaches: examples of the latter are cases of single machine scheduling [39], parallel machines [40–43], flow shops [44–46], and batch facilities [47].

Table 1 summarizes the findings on rework. In this table, papers are classified regarding scenario modelling (i.e., how rework events are considered): if reworks are modelled with estimated parameters, or if they are event-driven. The event-driven approach refers to those cases where the rework event data are known only when the production is started; thus, it is mandatory to make decisions on the fly with some rescheduling strategy. As seen in Table 1, few papers considered the event-driven approach, although these works studied single-machine problem and parallel machine settings [41, 48]. Then, to the best of our knowledge, in the literature, there are no event-driven approaches for the rescheduling problems with reworks for more complex manufacturing configurations as job shops, and this is a problem that occurs in real-life for instance in automotive manufacturing settings.

It can be seen from Table 1 that most works aim to optimize classical objective functions, although in terms of expected values. Only the paper of Liu and Zhou [41] addressed the topic of rework and stability although on single stage manufacturing systems.

To the best of our knowledge, no work has addressed the problem of the rescheduling with rework and reconditioning in job-shop systems. The literature on scheduling with rework estimate the schedule performance based on expected values calculated analytically or with simulation and consider the situation in which a job is processed on a machine, then inspected outside the machine, and immediately returned to the queue. These works do not consider reconditioning and/or stability. The works on rescheduling consider mostly machine breakdowns and other disruptions, but no rework.

3 Proposed approach

3.1 Assumptions

We consider the case in which a reconditioning action is needed prior to the rework itself. This is the same example of paint that needs to be scrapped off (i.e., the reconditioning) before the work piece is repainted (i.e., rework). The following assumptions hold in this paper:

– All job operations may eventually need rework. All operations that need rework require reconditioning actions.
– The corresponding reconditioning times are exponentially distributed. The mean of the distribution is denoted the mean time to reconditioning (MTTR).
– The rework probability of all operations is the same. This probability is denoted the percentage of reworked operations (PRO).
– The rework time of a job operation equals the original processing time.
– The time required for the reconditioning is known as soon as the rework/reconditioning request is triggered. In practice, we need technologies of industry.

3.2 Event-driven implementation

In this work, we study via simulation how the event-driven system reacts to disruptions that involve reconditioning and rework operations. This approach is different from previous works in rework in which the performance indicators are calculated based on expected values and not from the actual schedule execution.

The first step is to produce an "offline" schedule. In this paper, we generate schedules with two methods: The first method is dispatching rules, which are widely used and produce acceptable performance in terms of solution quality. The second method is a modified BAS (Beam A* Search) algorithm, originally presented in Mejía and Montoya [59] and later improved in Mejía and Lefebvre, [12] and in Rossit et al. [60]. BAS is a graph search algorithm for different types of scheduling problems. Such algorithm is amenable with our Petri Net model and performs well in comparison with other generic job shop scheduling algorithms. Like all

Table 1 Review of papers on reworks and scheduling

Authors	Scheduling problem	Solution approach and application	Scenario modelling	
			Estimation	Event-driven
Shin and Kang [49]	$P\left\|r_j, s_{kj}\right\|\sum_j T_j, L_{\max}, \sum_j C_j, C_{\max}$	Priority rules (PR)	✓	
Kang and Shin [40]	$R\|\|\sum_j T_j, L_{\max}, \sum_j C_j$	Priority rules (PR)	✓	
Rabiee et al. [50]	$F_2\|nwt, s_{ijk}\|\sum_j C_j$	Adapted imperialist competitive algorithm (AICA)	✓	
Moradinasab et al. [51]	$F_2\|nwt, s_{ijk} = 1\|\sum_j T_j$	Adapted imperialist competitive algorithm (AICA) and particle swarm optimization (PSO)	✓	
Liu and Zhou [41]	$P\|\|\sum_j w_j C_j, \eta$	Exact ad hoc method with stability		✓
Eskandari and Hosseinzadeh [45]	$FFSP\left\|s_{ikj}\right\|C_{\max}$	Variable neighborhood search (VNS)	✓	
Rambod and Rezaeian [42]	$R\|s_{ikj}, M_j\|C_{\max}$	GA, Bee Algorithms (BA)	✓	
Raghavan et al. [52]	$F\|\|\sum_j w_j T_j$	Heuristic (tailor-made). Semiconductor industry	✓	
Guo et al. [39]	$1\|prec\|waiting\ time$	Heuristics		✓
Bootaki and Paydar [53]	$F\|\|\sum_j E(C_{\max})$	Mixed integer programming + simulation	✓	
Foumani et al. [54]	$F\|\|Cycle\ Time$	Analytical method	✓	
Raghavan et al.[44]	$F\|\|\sum_j w_j T_j$	GA. Semiconductor industry	✓	
Zimmermann et al. [47]	$FMS\|batch\|C_{\max}$	Holonic multi-agent simulation	✓	
Zahedi et al. [55]	$F_2\left\|\overline{d_j}\right\|total\_cost$	Stepwise algorithm for mixed integer quadratic programming	✓	
Foumani et al. [56]	$F\|\|total\_cost$	Stochastic rework. E-constraint	✓	
Bian and Yang [57]	$FFSP\|\quad\|C_{\max}$	Bat metaheuristic algorithm	✓	
Wang et al. [43]	$R\|\|E(\sum_j T_j)$	PR, genetic algorithms (GA), simulated annealing (SA),	✓	
Guo et al. [58]	$1\|r_j\|\sum_j C_j$	Pseudo-polynomial DP, heuristic, GA	✓	
Gheisariha et al. [46]	$FFSP\|s_{ikj}, tt_{il}\|C_{\max}, \sum_j C_j$	Evolutionary multi-objective harmony search (EMOHS)	✓	
Our work	$J\|\|\sum_j C_j, \eta$	Modified beam search + Petri net modeling + stability		✓

graph search methods, BAS starts with a node (a marking in the Petri net) and progressively expands its children's nodes until a final goal is found (final marking). To avoid, "state explosion," the number of nodes at each state is limited at a certain predefined value (the "beam"). In BAS, the criterion for expansion is "best first," which implies an evaluation function for prioritizing expansion. We refer the reader to Mejía and Lefebvre [12] for more details of the BAS algorithm.

The pseudo code of the event-driven implementation is as follows:

Inputs An instance of a JSSP and its equivalent Petri net, the MTTR, and the PRO values.

Outputs The calculation of the executed total flow time and stability indicators.

1. Calculate an initial schedule array (S).
2. Execute the schedule until a reconditioning/rework request is issued for some operation o_{ij}^* or until all jobs are completed. A reconditioning/rework request occurs with a probability set as PRO.
3. If all jobs are completed, then terminate with success.
4. Else (some job operation o_{ij}^* needs rework)

   (a) Generate $\tau(o_{ij}^*)$, an exponentially distributed reconditioning time of operation o_{ij}^* with mean MTTR.

(b) Rebuild the schedule taking into account that job operation o^*_{ij} will be available for scheduling after $\tau(o^*_{ij})$ time units and go to 2. The new current schedule array (S) contains all job operations that have not started their processing.

This simulation of the event-driven process is illustrated in Fig. 3

3.3 Proposed rescheduling policies

In this paper, we study a hybrid policy that triggers either a full schedule regeneration or our adaptation of the right-shift policy depending on the duration of the reconditioning.

Definition Let U_t be the set of operations that have not started their execution at time t and that can start the earliest.

The "full regeneration" (FR) policy reschedules at time t all operations in U_t either with dispatching rules or with the BAS algorithm depending on how the initial schedule was computed. Clearly, the FR method must be consistent with the original generation method of the offline schedule. The complexity of FR will depend on the selected algorithm.

Our adaptation of the right-shift policy consists of rebuilding the schedule maintaining the original sequence as much as possible. We denote this policy as "keep sequence" (KS). The following pseudocode illustrates the KS approach. Let us suppose that a reconditioning/rework request is issued at time t (i.e., some job operation needs to be reworked).

Input The current schedule array S and $S' = \text{null}$;

Output A valid schedule S' that contains the start times of all operations that have not yet started.

1. Determine the set U_t at time t
2. Among all operations in U_t, find the operation that appears first on S, say operation o'_{ij}. Append o'_{ij} and its expected start time s'_{ij} to S' and remove it from S.
3. Repeat 1 and 2 until the set U_t is empty.

   Let n and m be the number of jobs and machines respectively. Computing U_t and finding o'_{ij}, both take mn steps in the worst case; therefore, the complexity of KS is $O(mn)$.

The selection of the rescheduling method (FR vs. KS) will depend of whether the reconditioning time exceeds a certain threshold. The rationale is that if the reconditioning time is short (i.e., below the threshold), there will be no need to reschedule all operations and the KS routine will be preferred; in the situation of above-threshold rework

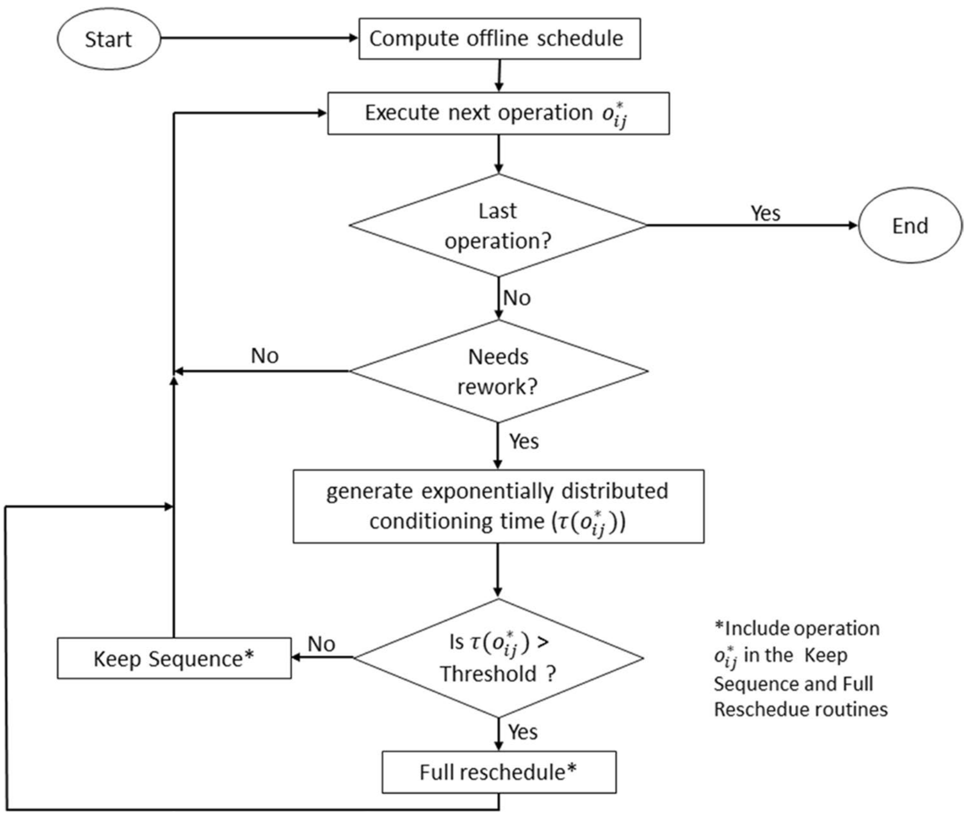
Start
Compute offline schedule
Execute next operation o^*_{ij}
Last operation?
Yes
End
No
Needs rework?
No
Yes
generate exponentially distributed conditioning time ($\tau(o^*_{ij})$)
Is $\tau(o^*_{ij})$ > Threshold ?
No
Keep Sequence*
Yes
Full reschedule*
*Include operation o^*_{ij} in the Keep Sequence and Full Reschedue routines

Fig. 3 Simulation process diagram

times, a full schedule regeneration will be selected. In the extreme cases, if the threshold is very small (~0), the schedule will always be fully regenerated (i.e., FG policy); if the threshold is very large (~∞), the KS routine will always be invoked. The main goal of this paper is to compare dedicated algorithms for offline scheduling vs. dispatching rules in a dynamic scenario. The section of computational results will present the behavior of the system under varying threshold values.

4 Experimental runs

In this section, we present the results of the computational experiments. In the experiments, we evaluated the performance of the BAS algorithm and of three dispatch rules: shortest processing time (SPT), longest processing time (LPT), and most work remaining (MWR). The three dispatching rules are selected for their capability to optimize each of the two criteria that we are considering in this study. We studied 100 instances, grouped in ten different sizes of instances, that is, ten different combinations of $J \times M$, namely, 10×5, 10×10, 15×5, 15×10, 15×15, 20×5, 20×10, 20×20, 30×05, and 30×10. The MTTR values were set as a percentage of the total work time of all job operations in particular each instance. The percentages were 0.5%, 1%, and 1.5%. The total work time is defined as the sum of processing times of all job operations. The PRO values were set as 2%, 5% and, 10% of the total number of operations. These values are consistent with the suggestions of Subramaniam et al. [20]. Threshold values were set between 0 and 500% of the MTTR in 50% steps [60].

The BAS algorithm and the three abovementioned dispatching rules were tested on 10 sets of different sizes, each set with 10 instances, with all combinations of MTTR (3 different values), PRO (3 different values), and threshold (11 different values). The number of replicates for each combination was set to 100. Thus, $100 \times 10 \times 10 \times 3 \times 3 \times 11 = 990,000$ different experiments were run.

The corresponding result analysis is presented as follows: first, we globally compare the performance of the rescheduling methods, then, the analysis is focused on comparing the dispatching rules and the BAS algorithm considering different threshold values in terms of stability and RPD. Finally, we analyze the effect of instances sizes, MTTRs and, PROs on the performance measures discussing the main insights that our study provide.

4.1 Evaluation of the rescheduling methods

In this section, we present a sample of the results obtained with the different scheduling methods and with different levels of the external factors, namely, MTTR and PRO. Due to the very large size of the experiments, only the illustrative results are presented; full results are presented in the supplementary material.

The charts of Fig. 4 show the performance of the tested algorithms in terms of RPD (averaged on all instances) vs. threshold. PRO values are identified with lines of different colors, while MTTR values are identified by markers. The

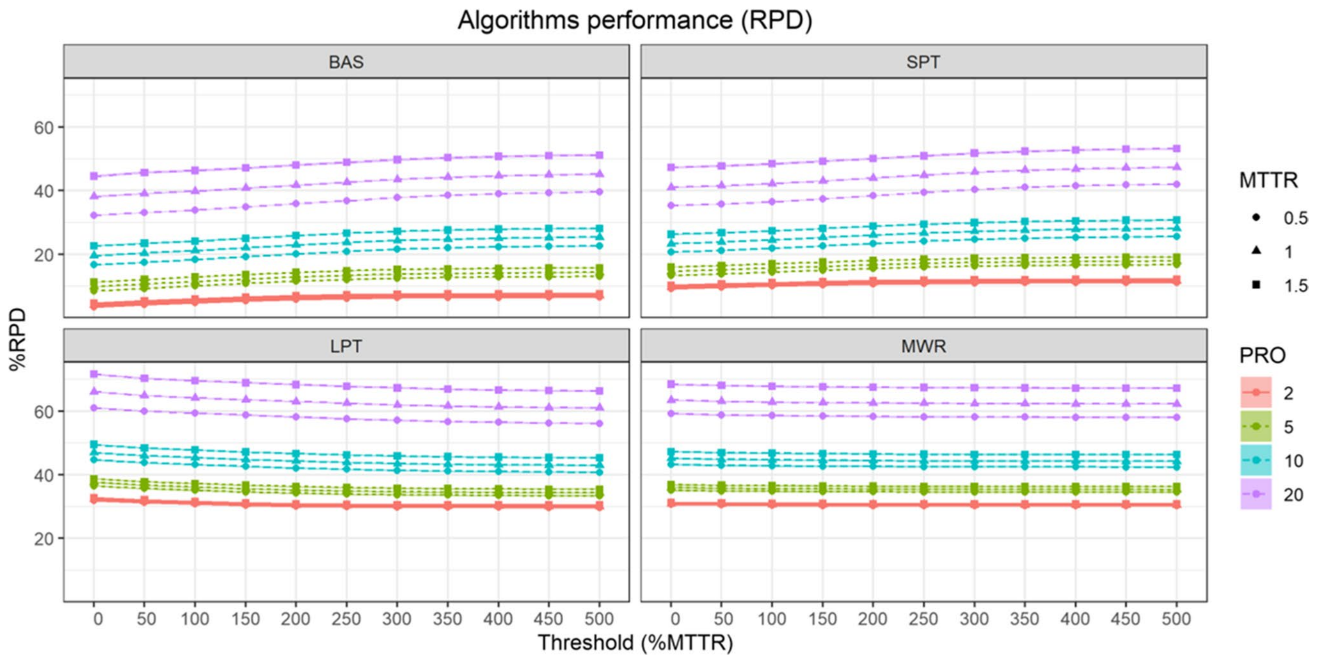

Fig. 4 Average RPD values vs. threshold

charts of Fig. 5 represent the stability values (averaged on all instances) vs. threshold, following a format analogous to that of Fig. 4.

We can observe the BAS algorithm is the best performer in terms of RPD, but it is also the worst performer in terms of stability. The opposite is true for the MWR rule. The performance of all algorithms is significantly affected by the threshold values. However, we could not observe a definite trend: for both BAS and SPT, as the threshold value increases, the RPD increases and the stability decreases (improves), also as expected. However, for the LPT and MWR rules, the pattern is the opposite; both RPD and stability improve with higher values of threshold. For the lowest value of PRO (i.e., PRO = 2), the RPDs of a particular algorithm are virtually the same for all values of MTTR and the same threshold. However, for greater values of PRO, the differences in RPD are more noticeable for the different values of MTTR. Also, for PRO = 2 and PRO = 5, the average RPD of the BAS algorithm is clearly better that those of the dispatching rules for all threshold and MTTR values. However, as the values of PRO increase, the differences between BAS and the second performer (SPT) are smaller. For example, for PRO = 2, MTTR = 0.5 and threshold = 0, the average RPD of BAS is around 3.5%, whereas for SPT is around %7.0; at the other extreme, for PRO = 20, MTTR = 1.5 and threshold = 500, the average RPD of BAS is around 49.0%, whereas for SPT is around %51.0.

Changes in RPD and stability are negligible with threshold values higher than 250% for all algorithms. Threshold values have a more noticeable impact on the BAS algorithm than on the dispatching rules, especially in terms of stability. It can be seen in Figs. 4 and 5 that for the BAS algorithm, the slopes of the lines are steeper at lower values of threshold in comparison with the dispatching rules. Generally speaking, the BAS algorithm is more sensitive to changes in threshold in comparison with the dispatching rules. This characteristic can be useful for the decision maker to establish the desired trade-off between schedule quality (measured by the objective function) and stability. On the other hand, dispatching rules are very robust to changes in threshold and therefore the KS policy can be adopted as it is the least demanding in terms of computational terms.

4.2 Comparison of the algorithms' performance

In this section, we compare the algorithms averaging for all values of MTTR and PRO and varying the thresholds. Figures 6 and 7 illustrate the average RPD and stability, respectively. In each figure, the four algorithms (BAS, SPT, LPT, and MWR) are shown with different colored lines. The values on the charts represent the global average of all the experiments for different values of threshold.

Figure 6 shows that BAS clearly outperforms the MWR, SPT, and LPT dispatching rules in terms of the average RPD for all threshold values. As the threshold values increases, the RPD increases for BAS and SPT, while MWR and LPT remain more or less constant. Notice that BAS and SPT present a similar pattern. This can be

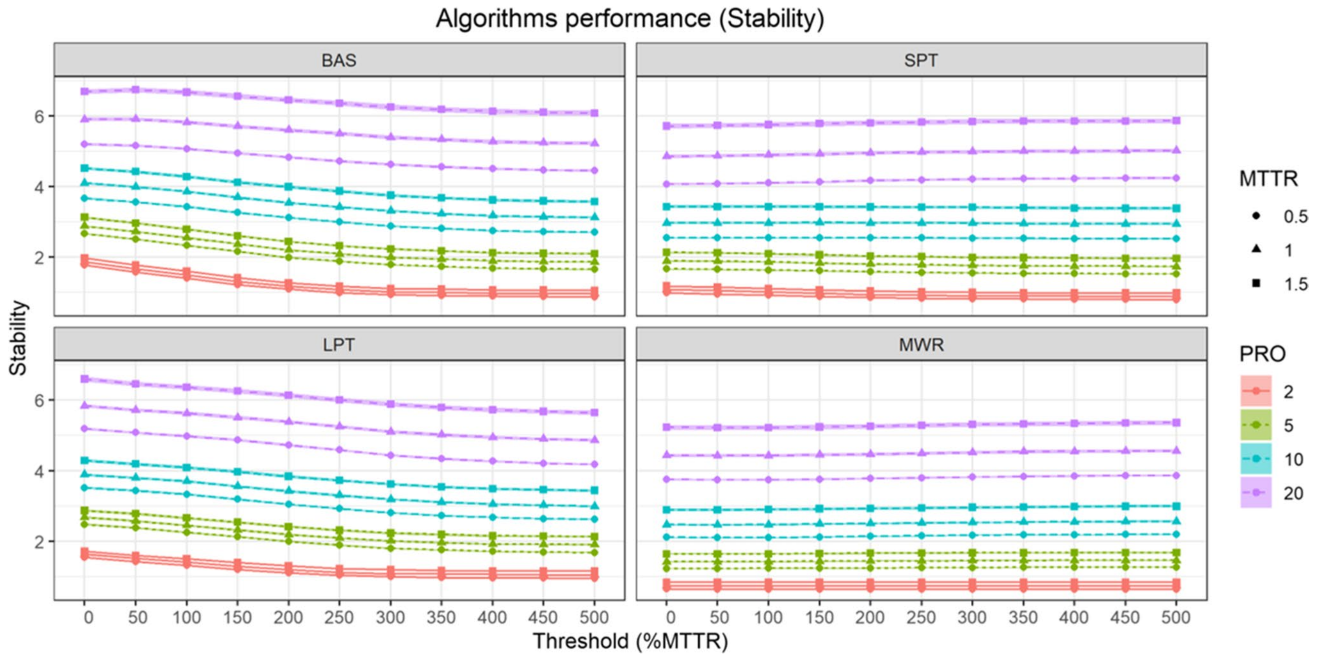

Fig. 5 Average stability values vs. threshold

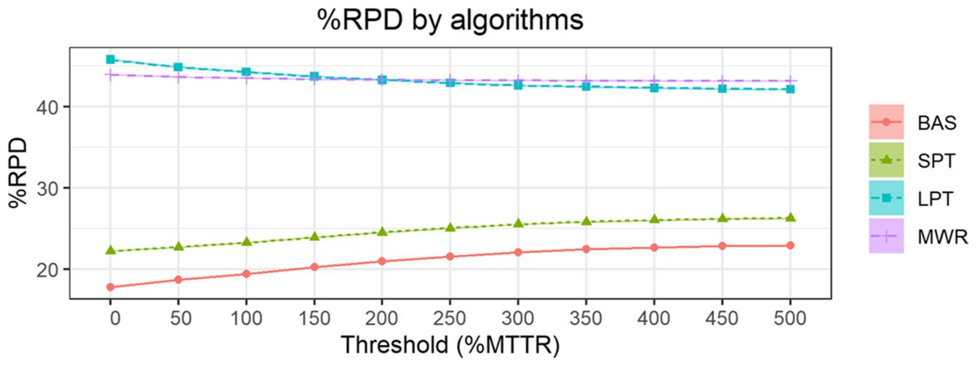

Fig. 6 Average RPD for all values of RPD and PRO

explained due to the fact that BAS uses SPT as its evaluation function.

Regarding the stability, Fig. 7 shows that MWR dispatching rule is the algorithm that, on average, leads to more stable schedules. On the other hand, the BAS algorithm is the one that on average leads to more unstable schedules. Nevertheless, at higher threshold values, the difference between BAS and the dispatching rules reduces. As the threshold values increases, the stability with the MWR and SPT does not seem to vary much, while it tends to decrease with the BAS and LPT algorithms.

Another feature of the problem that is relevant to analyze is the effect of the instance sizes in the performance of the compared algorithms in terms of the RPD and stability. The following findings show that the relation between the number of jobs and the number of machines of different instance sizes, i.e., ratio = $\frac{|J|}{|M|}$, has an important influence on the performance of all algorithms in terms of RPD and stability. The ratio can be seen as an indicator of congestion as more jobs are pushed into system. Figure 5 illustrates the average relative percentage differences (ARPD) of the algorithms with respect to the best performer (BAS) of RPD vs. threshold.

$$ARPD = \frac{\overline{RPD}_{algo} - \overline{RPD}_{best}}{\overline{RPD}_{best}} \times 100\%$$

$\overline{RPD}_{algo}$ and $\overline{RPD}_{best}$ are respectively the average relative percentage deviations of the algorithm and of the best performer.

Figure 8 shows a selection of the instance sizes, particularly, those with 10 machines (10×10, 15×10, 20×10, and 30×10) with ratios of 1, 1.5, 2, and 3 respectively. These results suggest that more congested systems (generally speaking, congested system are those with far more jobs that machines) would be benefited from better algorithms in the case of rescheduling with rework.

In Fig. 8, we can observe that all ARPD values are larger in the LPT and MWR charts (ranging from 10 to 30%) in comparison with the SPT chart (between 3 and 5%). For example, in the 30×10, 20×10, and 15×10 curves, with ratios of 3, 2, and 1.5 respectively, the ARPD values are larger in comparison with instances of ratio of 1 (10×10).

Figure 9 presents the average stability percentage differences (ASPDs) between each algorithm and the best performer (MWR dispatching rule in this case).

$$ASPD = \frac{\bar{\eta}_{algo} - \bar{\eta}_{best}}{\bar{\eta}_{best}} \times 100\%$$

$\bar{\eta}_{algo}$ and $\bar{\eta}_{best}$ are respectively the average stability values of the selected algorithm and of the best performer.

The results show that the ratio also impacts the performance of the algorithm. Notice that there is a similar pattern in comparison with the ARPD curves, although

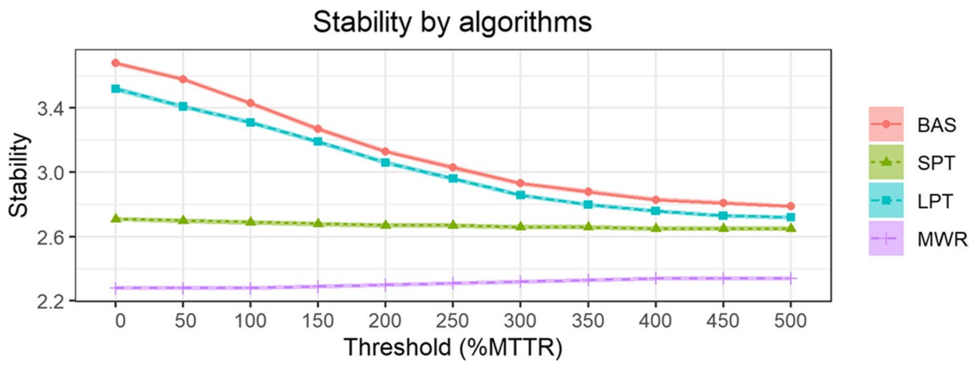

Fig. 7 Average stability %

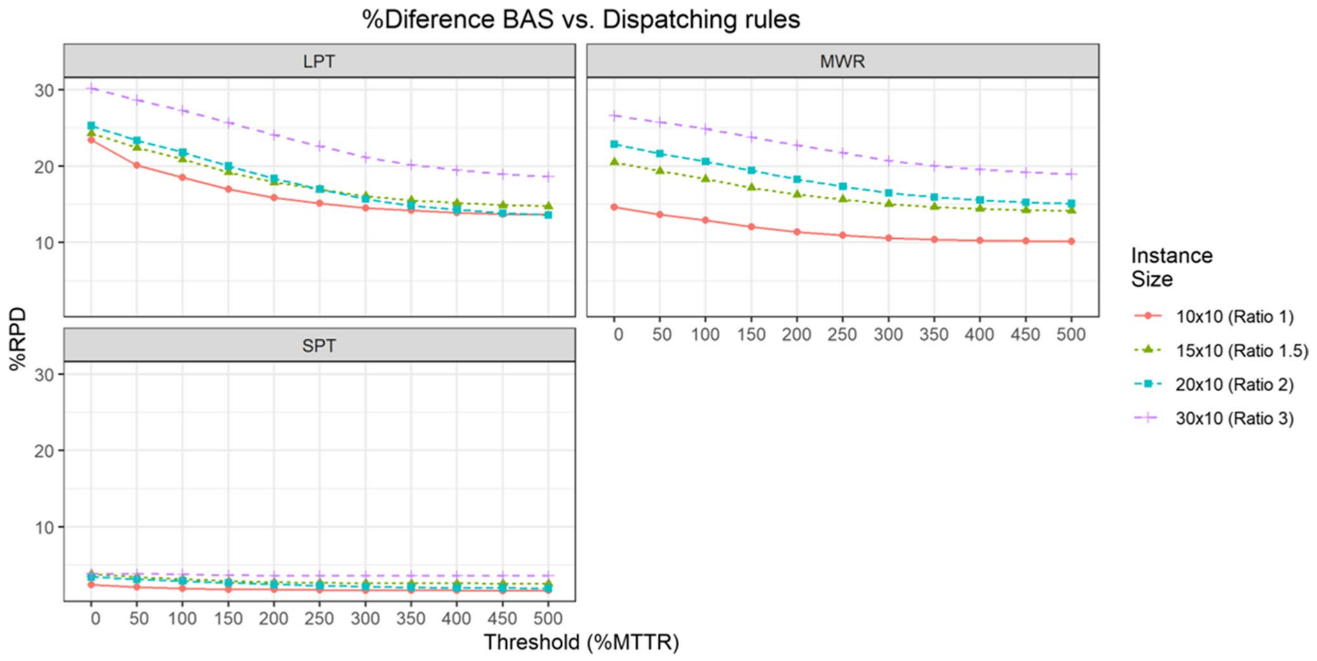

Fig. 8 % ARPD

the best performer is now the MWR rule. In addition, as the threshold values increases, the stability obtained with BAS and SPT improves. This improvement is much more accentuated for those cases with larger job/machine ratios. The sensitivity of the ratio vs. the threshold, in terms of stability, is clearly seen when comparing the performance of the algorithms on the 30×10 (ratio = 3) curves and on the 10×10 (ratio = 1) for BAS and SPT.

4.3 Problem features assessment: PRO and MTTR values

Finally, in this section, we will present a detailed analysis of the PRO and MTTR factors on the performance of the BAS algorithm. First, the impact of the PRO for the BAS algorithm is analyzed with fixed threshold and MTTR values. Figures 10 and 11 show the RPD and the stability values

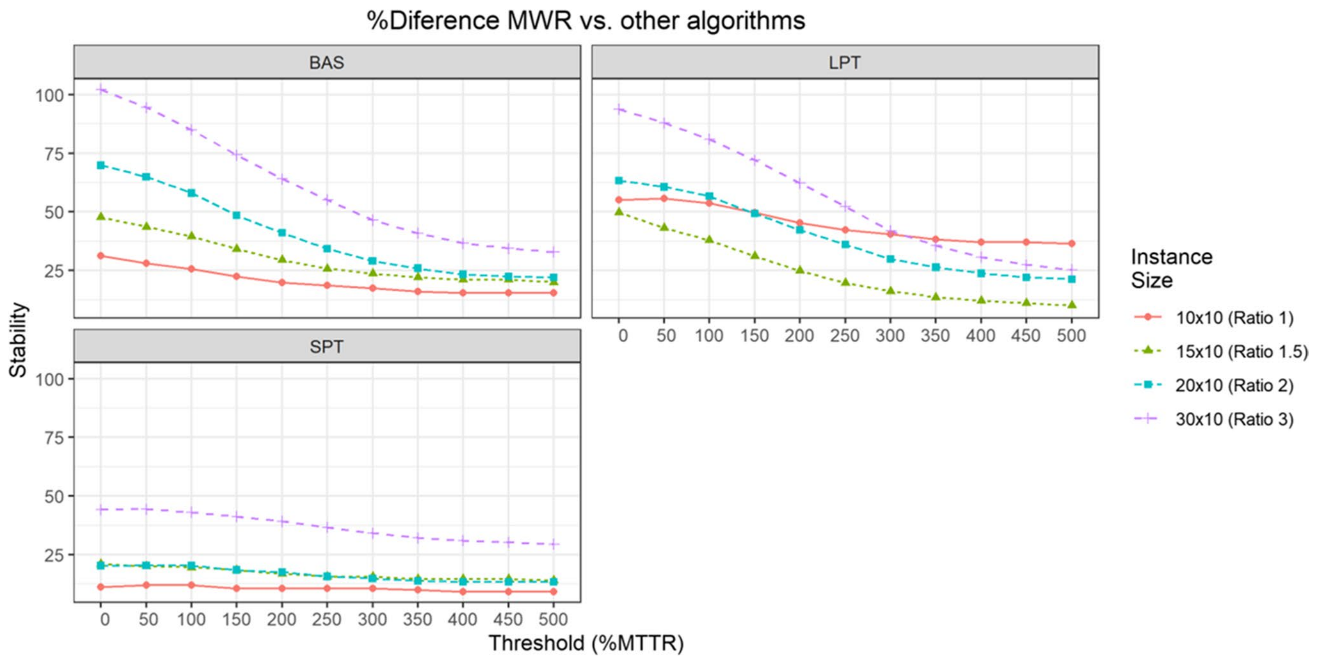

Fig. 9 % stability difference SPT; LPT; BAS vs. MWR

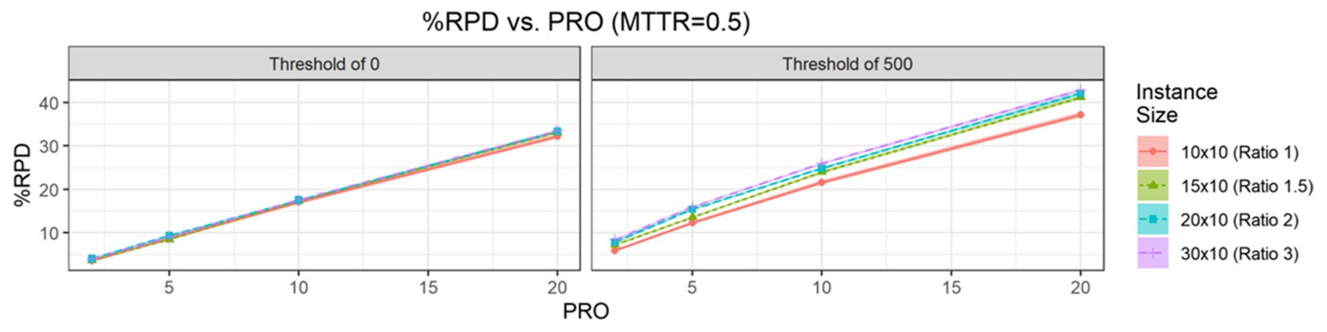

Fig. 10 BAS algorithm RPD impact depending on the number of machines

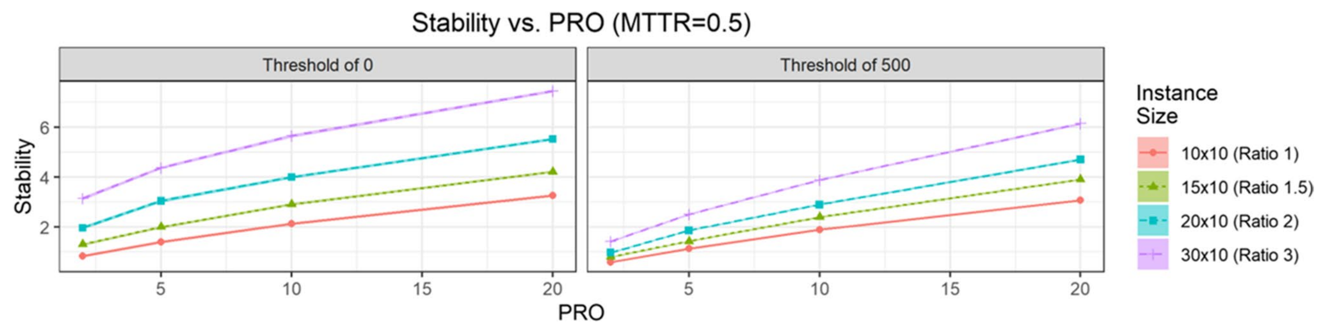

Fig. 11 BAS algorithm stability impact depending on the number of machines

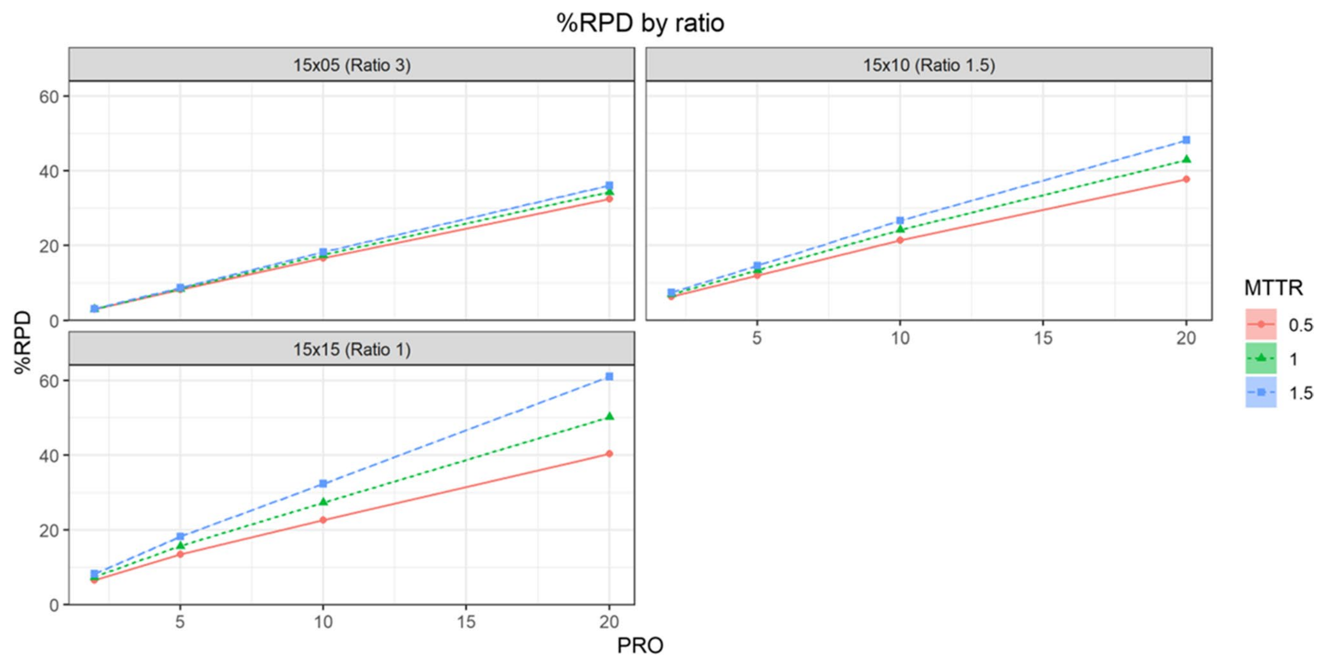

Fig. 12 BAS algorithm RPD depending on the PROs and instance sizes

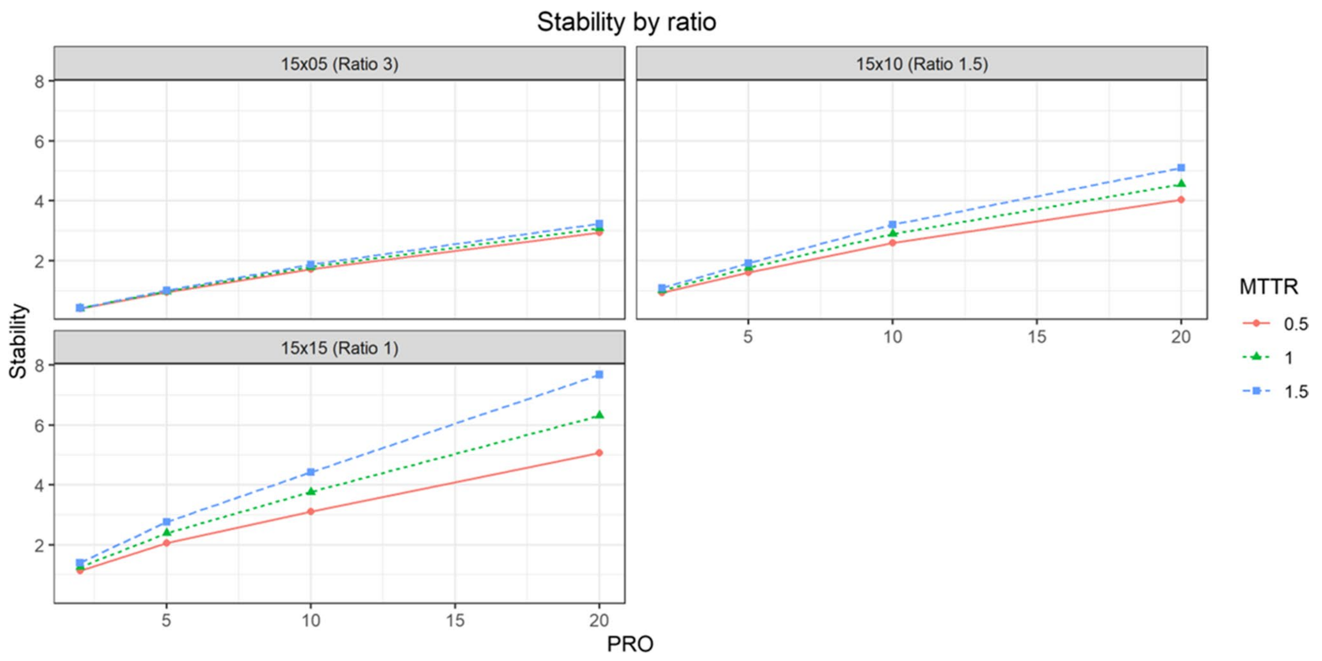

Fig. 13 BAS algorithm stability impact depending on the number of machines

of the BAS algorithm for different values of PRO. In these figures, the number of machines is fixed at 10, with different job/machine ratios. The threshold is fixed at 0% on the left panels of Figs. 10 and 11, and on the right panels, the threshold is fixed at 500%. The MTTR considered for the cases depicted at Figs. 8 and 9 is of 0.5.

In general terms, it can be noticed that both the RPD and the stability increase in a linear fashion as the PRO values increase. Another observation illustrated by Figs. 10 and 11 is that the RPD and the stability tends to deteriorate as the number of jobs increases (with the same number of machines). Both figures represent instances of 10 machines, but different number of jobs, and when the charts are compared to each other, it is observed that the larger the number the jobs (the larger the ratio), the larger increase in the values of the RPD and the stability. Nevertheless, it can be mentioned that this behavior is more notorious for stability than for the RPD. It is important to mention that these findings will also take place when considering other MTTR values, such as 1.0 and 1.5 that for sake of brevity are not introduced in this manuscript.

Finally, we studied the effect of the MTTR on the stability and the RPD, considering the different PRO, depending on the features of the instance sizes, as is shown in Figs. 12 and 13. This analysis is carried out for the BAS algorithm with a fixed threshold value, but it can be extended for the other algorithms and threshold values. From Figs. 12 and 13, it is possible to notice that, in general terms, for each MTTR, both RPD and stability tend to deteriorate while the PRO increases. In order to identify other behavior patterns, we analyze each instance size according to the number of jobs and the ratio. As an example, Figs. 12 and 11 show the RPD and stability behavior respectively for instances with 15 jobs (15×5, 15×10, and 15×5). Particularly, we can observe that, for instances with larger ratio values, when considering lower PROs values, the RPD and the stability do not seem to present a relevant variation between the ones obtained with the different MTTR values. Nevertheless, as the ratio decreases, the RPD and stability tend to deteriorate as the MTTR values increases.

5 Conclusions and future work

In this work, we studied and presented the results of job shop rescheduling under rework and reconditioning. We used an event-driven scheduling method embedded in a Petri nets formalism for all the algorithms. The results were analyzed in terms of RPD and stability, showing that the BAS algorithm in combination with high threshold values is the algorithm that achieves the best results. On the other hand, simple dispatching rules are more stable than the more sophisticated BAS. These results also indicate that the threshold has a significant impact on the algorithm performance, with higher thresholds resulting in better stability but worse RPD. The overall conclusion is that, for the studied methods, no algorithm fully outperforms the others in both criteria. However, if the PRO and MTTR values can be reduced through better manufacturing methods and quality control, the use a better scheduler such as BAS is justified. Even more, the experimentation justifies the dispatching rule selection, since MWR is the best in the stability criterion alone; meanwhile, SPT is the best for total flow time alone. However, BAS algorithm has the best performance regarding the two criteria together, yielding reschedules that can represent the compromise solution from both criteria.

We also analyzed the influence of the instance size on the performance of the system. When analyzing the instance, an important factor that describes the behavior of the system was the ratio between the number of machines and the number of jobs. For higher ratio values (i.e., increasing the number of jobs for a fixed number of machines), the stability was worse.

In terms of potential implementations, Industry 4.0 Technologies and machine learning algorithms are a must. The system needs to quickly detect the need of rework and to compute the time to reconditioning so to establish the "best" reactive strategy. The computation of the reconditioning time requires automatic learning to provide better estimates. As future research lines, we aim to extend the analysis to study the impact of real-time information and the impact of other events such as machine breakdowns.

Game theoretical approaches can be also included in further research. For example, an interesting approach will be handling customer orders. The Petri net approach is also well suited to handle communication protocols and can be combined with the manufacturing system model.

Acknowledgements Gonzalo Mejía would like to express his gratitude to Universidad de La Sabana for the time and resource usage for this project.

Funding Daniel A. Rossit and Gonzalo Mejía have been funded by CYTED under the project Red Iberoamericana Industria 4.0, grant number 319RT0574.

Availability of data and material The instances are available in the supplementary material.

Code availability The code is custom code.

Declarations

Competing interests The authors declare no competing interests.

References

1. Zhong RY, Xu X, Klotz E, Newman ST (2017) Intelligent manufacturing in the context of Industry 4.0: a review. Engineering 3(5):616–630
2. Xu LD, Xu EL, Li L (2018) Industry 4.0: State of the art and future trends. Int J Prod Res 56(8):2941–2962
3. Zhang J, Ding G, Zou Y, Qin S, Fu J, Zhong RY, Newman ST (2019) Review of job shop scheduling research and its new perspectives under Industry 4.0. J Intell Manuf 30(4):1809–1830. https://doi.org/10.1016/J.ENG.2017.05.015
4. Dolgui A, Ivanov D, Sethi SP, Sokolov B (2019) Scheduling in production, supply chain and Industry 4.0 systems by optimal control: fundamentals, state-of-the-art and applications. Int J Prod Res 57(2):411–432. https://doi.org/10.1080/00207543.2018.1442948
5. Rossit DA, Tohmé F, Frutos M (2019) Industry 4.0: Smart scheduling. Int J Prod Res 57(12):3802–3813. https://doi.org/10.1080/00207543.2018.1504248
6. Li Y, Carabelli S, Fadda E, Manerba D, Tadei R, Terzo O (2020) Machine learning and optimization for production rescheduling in Industry 4.0. Int J Adv Manuf Technol 110(9–10):2445–2463. https://doi.org/10.1007/s00170-020-05850-5
7. Vieira G, Herrman J, Lin E (2003) Rescheduling manufacturing systems: a framework of strategies, policies and methods. J Sched 6(1):39–62
8. Ouelhadj D, Petrovic S (2009) A survey of dynamic scheduling in manufacturing systems. J Sched 12:417–431
9. Nasir V, Sassani F (2021) A review on deep learning in machining and tool monitoring: methods, opportunities, and challenges. Int J Adv Manuf Technol 2683–2709. https://doi.org/10.1007/s00170-021-07325-7
10. Leusin ME, Frazzon EM, Uriona Maldonado M, Kück M, Freitag M (2018) Solving the job-shop scheduling problem in the Industry 4.0 era. Technologies 6(4):107. https://doi.org/10.3390/technologies6040107
11. Katragjini K, Vallada E, Ruiz R (2013) Flow shop rescheduling under different types of disruption. Int J Prod Res 51(3):780–797. https://doi.org/10.1080/00207543.2012.666856
12. Mejía G, Lefebvre D (2020) Robust scheduling of flexible manufacturing systems with unreliable operations and resources. Int J Prod Res 58(21):6474–6492. https://doi.org/10.1080/00207543.2019.1682706
13. Mejía G, Pereira J (2020) Multiobjective scheduling algorithm for flexible manufacturing systems with Petri nets. J Manuf Syst 54(December 2019):272–284. https://doi.org/10.1016/j.jmsy.2020.01.003
14. Caballero-Villalobos J, Mejía-Delgadillo GE, García-Cáceres RG (2013) Scheduling of complex manufacturing systems with Petri nets and genetic algorithms: a case on plastic injection moulds. Int J Adv Manuf Technol 69(9–12):2773–2786. https://doi.org/10.1007/s00170-013-5175-7
15. Mejía G, Niño K, Montoya C, Sánchez MA, Palacios J, Amodeo L (2016) A Petri Net-based framework for realistic project management and scheduling: an application in animation and videogames. Comput Oper Res 66:190–198. https://doi.org/10.1016/j.cor.2015.08.011
16. Lee DY, DiCesare F (1994) Scheduling flexible manufacturing systems using Petri nets and heuristic search. IEEE Trans Robot Autom 10(2):123–132. https://doi.org/10.1109/70.282537
17. Mejia G, Caballero-Villalobos JP, Montoya C (2018) Petri nets and deadlock-free scheduling of open shop manufacturing systems. IEEE Trans Syst Man Cybern: Syst 48(6):1017–1028. https://doi.org/10.1109/TSMC.2017.2707494
18. Zhou MC, DiCesare F (1989) Adaptive design of Petri net controllers for error recovery in automated manufacturing systems. IEEE Trans Syst Man Cybern. https://doi.org/10.1109/21.44011
19. Ahmadi E, Zandieh M, Farrokh M, Emami SM (2016) A multi objective optimization approach for flexible job shop scheduling problem under random machine breakdown by evolutionary algorithms. Comput Oper Res 73:56–66. https://doi.org/10.1016/j.cor.2016.03.009
20. Subramaniam V, Raheja AS, Rama Bhupal Reddy K (2005) Reactive repair tool for job shop schedules. Int J Prod Res 43(1):1–23. https://doi.org/10.1080/0020754042000270412
21. Dong YH, Jang J (2012) Production rescheduling for machine breakdown at a job shop. Int J Prod Res 50(10):2681–2691. https://doi.org/10.1080/00207543.2011.579637

22. Pfeiffer A, Kádár B, Monostori L (2007) Stability-oriented evaluation of rescheduling strategies, by using simulation. Comput Ind 58(7):630–643. https://doi.org/10.1016/j.compind.2007.05.009
23. Bidot J, Vidal T, Laborie P, Beck JC (2009) A theoretic and practical framework for scheduling in a stochastic environment. J Sched 12(3):315
24. He W, Sun DH (2013) Scheduling flexible job shop problem subject to machine breakdown with route changing and right-shift strategies. Int J Adv Manuf Technol 66(1–4):501–514. https://doi.org/10.1007/s00170-012-4344-4
25. Larsen R, Pranzo M (2019) A framework for dynamic rescheduling problems. Int J Prod Res 57(1):16–33. https://doi.org/10.1080/00207543.2018.1456700
26. Gao K, Yang F, Li J, Sang H, Luo J (2020) Improved jaya algorithm for flexible job shop rescheduling problem. IEEE Access 8:86915–86922
27. Nie L, Wang X, Liu K, Bai Y (2020) A rescheduling approach based on genetic algorithm for flexible scheduling problem subject to machine breakdown. In Journal of Physics: Conference Series (Vol. 1453, p 12018)
28. Framinan JM, Fernandez-Viagas V, Perez-Gonzalez P (2019) Using real-time information to reschedule jobs in a flowshop with variable processing times. Comput Ind Eng 129(January):113–125. https://doi.org/10.1016/j.cie.2019.01.036
29. Zhang W, Xiao J, Zhang S, Lin J, Feng R (2021) A utility-aware multi-task scheduling method in cloud manufacturing using extended NSGA-II embedded with game theory. Int J Comput Integr Manuf 34(2):175–194. https://doi.org/10.1080/0951192X.2020.1858502
30. Carlucci D, Renna P, Materi S, Schiuma G (2020) Intelligent decision-making model based on minority game for resource allocation in cloud manufacturing. Manag Decis 58(11):2305–2325. https://doi.org/10.1108/MD-09-2019-1303
31. Goodarzi EV, Houshmand M, Valilai OF, Ghezavati V, Bamdad S (2020) Manufacturing cloud service composition based on the non-cooperative and cooperative game theory. In 2020 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM) (pp 1122–1125). https://doi.org/10.1109/IEEM45057.2020.9309921
32. Ghaleb M, Zolfagharinia H, Taghipour S (2020) Real-time production scheduling in the Industry-4.0 context : Addressing abstract. Comput Oper Res 105031. https://doi.org/10.1016/j.cor.2020.105031
33. Inderfurth K, Kovalyov MY, Ng CT, Werner F (2007) Cost minimizing scheduling of work and rework processes on a single facility under deterioration of reworkables. Int J Prod Econ 105(2):345–356. https://doi.org/10.1016/j.ijpe.2004.02.010
34. Wee H-M, Wang W-T, Cárdenas-Barrón LE (2013) An alternative analysis and solution procedure for the EPQ model with rework process at a single-stage manufacturing system with planned backorders. Comput Ind Eng 64(2):748–755. https://doi.org/10.1016/j.cie.2012.11.005
35. Ko H-HH, Kim J, Kim S-SS, Baek J-GG (2010) Dispatching rule for non-identical parallel machines with sequence-dependent setups and quality restrictions. Comput Ind Eng 59(3):448–457. https://doi.org/10.1016/j.cie.2010.05.017
36. Moshtagh MS, Taleizadeh AA (2017) Stochastic integrated manufacturing and remanufacturing model with shortage, rework and quality based return rate in a closed loop supply chain. J Clean Prod 141:1548–1573. https://doi.org/10.1016/J.JCLEPRO.2016.09.173
37. Shin HJ, Kang YH, Fitts EP (2010) A rework-based dispatching algorithm for module process in TFT-LCD manufacture. Int J Prod Res 48(3):915–931. https://doi.org/10.1080/00207540802471264
38. Kang YH, Kim SS, Shin HJ (2010) A dispatching algorithm for parallel machines with rework processes. J Oper Res Soc 61(1):144–155. https://doi.org/10.1057/jors.2008.148
39. Guo Y, Huang M, Wang Q, Leon VJ (2016) Single-machine rework rescheduling to minimize maximum waiting-times with fixed sequence of jobs and ready times. Comput Ind Eng 91:262–273. https://doi.org/10.1016/j.cie.2015.11.021
40. Kang YH, Shin HJ (2010) An adaptive scheduling algorithm for a parallel machine problem with rework processes. Int J Prod Res 48(1):95–115. https://doi.org/10.1080/00207540802484903
41. Liu L, Zhou H (2013) On the identical parallel-machine rescheduling with job rework disruption. Comput Ind Eng 66(1):186–198. https://doi.org/10.1016/j.cie.2013.02.018
42. Rambod M, Rezaeian J (2014) Robust meta-heuristics implementation for unrelated parallel machines scheduling problem with rework processes and machine eligibility restrictions. Comput Ind Eng 77:15–28. https://doi.org/10.1016/j.cie.2014.09.006
43. Wang X, Li Z, Chen Q, Mao N (2020) Meta-heuristics for unrelated parallel machines scheduling with random rework to minimize expected total weighted tardiness. Comput Ind Eng 145(100):106505. https://doi.org/10.1016/j.cie.2020.106505
44. Raghavan VA, Yoon SW, Srihari K (2018) A modified Genetic Algorithm approach to minimize total weighted tardiness with stochastic rework and reprocessing times. Comput Ind Eng 123:42–53. https://doi.org/10.1016/j.cie.2018.06.002
45. Eskandari H, Hosseinzadeh A (2014) A variable neighbourhood search for hybrid flow-shop scheduling problem with rework and set-up times. J Oper Res Soc 65(8):1221–1231. https://doi.org/10.1057/jors.2013.70
46. Gheisariha E, Tavana M, Jolai F, Rabiee M (2021) A simulation–optimization model for solving flexible flow shop scheduling problems with rework and transportation. Math Comput Simul 180:152–177. https://doi.org/10.1016/j.matcom.2020.08.019
47. Zimmermann E, El Haouzi HB, Thomas P, Pannequin R, Noyel M, Thomas A (2018) A case study of intelligent manufacturing control based on multi-agents system to deal with batching and sequencing on rework context BT - service orientation in holonic and multi-agent manufacturing: proceedings of SOHOMA 2017. In Borangiu T, Trentesaux D, Thomas A, Cardin O (Eds.), (pp 63–75). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-73751-5_6
48. Chang C-K, Hsiang C-L (2011) Using generalized stochastic Petri nets for preventive maintenance optimization in automated manufacturing systems. J Qual 18(2):117–135. Retrieved from: http://www.scopus.com/inward/record.url?eid=2-s2.0-79955735548&partnerID=40&md5=f619e198e82f1fcb594e6b6156a4adbc
49. Shin HJ, Kang YH (2010) A rework-based dispatching algorithm for module process in TFT-LCD manufacture. Int J Prod Res 48(3):915–931. https://doi.org/10.1080/00207540802471264
50. Rabiee M, Zandieh M, Jafarian A (2012) Scheduling of a no-wait two-machine flow shop with sequence-dependent setup times and probable rework using robust meta-heuristics. Int J Prod Res 50(24):7428–7446. https://doi.org/10.1080/00207543.2011.652747
51. Moradinasab N, Shafaei R, Rabiee M, Mazinani M (2012) Minimization of maximum tardiness in a no-wait two stage flexible flow shop. Int J Artif Intell 8(12 S):166–181. Retrieved from: https://www.scopus.com/inward/record.uri?eid=2-s2.0-84863570724&partnerID=40&md5=51f5461dc743effa7f38a77639763491
52. Raghavan VA, Yoon SW, Srihari K (2015) Heuristic algorithms to minimize total weighted tardiness with stochastic rework and reprocessing times. J Manuf Syst 37(Part 1):233–242. https://doi.org/10.1016/j.jmsy.2014.09.004
53. Bootaki B, Paydar MM (2016) A probabilistic model toward a permutation flowshop scheduling problem with imperfect jobs. Int J Manage Sci Eng Manage 11(3):186–193. https://doi.org/10.1080/17509653.2015.1045048
54. Foumani M, Smith-Miles K, Gunawan I (2017) Scheduling of two-machine robotic rework cells: In-process, post-process and in-line inspection scenarios. Robot Auton Syst 91:210–225

55. Zahedi Z, Salim A, Yusriski R, Haris H (2019) Optimization of an integrated batch production and maintenance scheduling on flow shop with two machines. Int J Ind Eng Comput 10(2):225–238. https://doi.org/10.5267/j.ijiec.2018.7.001
56. Foumani M, Razeghi A, Smith-Miles K (2020) Stochastic optimization of two-machine flow shop robotic cells with controllable inspection times: from theory toward practice. Robot Comput-Integr Manuf 61(April 2019):101822. https://doi.org/10.1016/j.rcim.2019.101822
57. Bian J, Yang L (2020) A study of flexible flow shop scheduling problem with variable processing times based on improved bat algorithm. Int J Simul Process Model 15(3):245–254. https://doi.org/10.1504/IJSPM.2020.107329
58. Guo Y, Huang M, Wang Q, Leon VJ (2021) Single-machine rework rescheduling to minimize total waiting time with fixed sequence of jobs and release times. IEEE Access 9:1205–1218. https://doi.org/10.1109/ACCESS.2019.2957132
59. Mejía G, Montoya C (2008) A Petri Net based algorithm for minimizing total tardiness in flexible manufacturing systems. Ann Oper Res 164(1):63–78
60. Rossit DA, Tohmé F, Mejía G (2020) The tolerance scheduling problem in a single machine case BT - scheduling in Industry 4.0 and cloud manufacturing. In Sokolov B, Ivanov D, Dolgui A (Eds.), Scheduling in and Cloud Manufacturing (pp. 255–273). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-43177-8_13