EINDHOVEN UNIVERSITY OF TECHNOLOGY

Unsupervised parallel machines scheduling with tool switches

Citation for published version (APA):
Dang, Q.-V., Herps, K., Martagan, T. G., Adan, I. J. B. F., & Heinrich, J. (2023). Unsupervised parallel machines scheduling with tool switches. *Computers and Operations Research*, *160*, Article 106361. https://doi.org/10.1016/j.cor.2023.106361

Document license:
CC BY

DOI:
10.1016/j.cor.2023.106361

Document status and date:
Published: 01/12/2023

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 08. Feb. 2024

Computers & Operations Research 160 (2023) 106361

Contents lists available at ScienceDirect

Computers and Operations Research

journal homepage: www.elsevier.com/locate/cor

Unsupervised parallel machines scheduling with tool switches

Quang-Vinh Dang [a,*], Koen Herps [a,b], Tugce Martagan [a], Ivo Adan [a], Jasper Heinrich [a]

[a] Department of Industrial Engineering and Innovation Sciences, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[b] KMWE Precision, BIC 1, P.O. Box 7930, 5605 SH Eindhoven, The Netherlands

ARTICLE INFO

Keywords:
Scheduling
Tool switches
Unsupervised production
Genetic algorithm
Mixed-integer linear programming

ABSTRACT

This paper addresses the problem of scheduling jobs on identical parallel machines with tool switches in a high-mix, low-volume manufacturing environment. Inspired by the initiatives on "lights-out factory" at our industry partner, our problem setting involves several complex features. For example, we consider unsupervised production hours (e.g., night shifts where operators are not available) in which tool switches cannot occur. Moreover, motivated by current practice, tool switches in our problem setting cause costs instead of delays. Also, a subset of jobs is prioritized to be completed within a scheduling horizon, and a job may consist of ordered operations due to reentry to machines. The objective is to maximize the profit generated by the manufacturing system, which is composed of revenue generated by the finished operations minus tool switching costs and penalty costs of unfinished priority jobs. The decisions involve assigning operations to machines, sequencing these operations, and determining a tool-switching plan. A mix-integer linear programming model is first formulated. We then propose a genetic algorithm to solve industry-size problem instances, in which tailored crossover and mutation mechanisms are introduced. We illustrate the performance of the proposed GA with industry case studies using real-world data. We also make the anonymized data set publicly available. Computational experiments reveal that approximately 26% profit improvement can be achieved by using the proposed GA instead of the current way of scheduling at our industry partner. Moreover, we find that the proposed GA brings higher benefits when the duration of the unsupervised shifts gets longer, and there is high pressure on prioritizing jobs in the schedule.

1. Introduction

Following the principles of smart industry and industry 4.0, many advanced manufacturing companies aim to have a "lights-out factory" that automatically controls activities, operations, and material flow on the shop floor, with minimal human interference. However, aside from the enhanced focus on an improved automation level to increase productivity, manufacturers face diversification in customer requirements. Customers' focus is moving towards customized products, resulting in a higher product variety, commonly referred to as the high-mix, low-volume (HMLV) industry. Although the use of automation equipment, e.g., computer numerical control (CNC) machines, can assist in efficiently and effectively processing production schedules, it is still crucial to consider the collaboration between machines and labor resources to optimize the production schedules (Qin et al., 2016; Lu, 2017).

CNC machines form the basis of flexible manufacturing systems (FMSs) by performing a wide range of machining operations to produce high-mix, low-volume job sets. Consequently, FMSs are capable of processing a variety of different product types and offer the required flexibility in production scheduling to react to changes in customer demand (Shivanand et al., 2006; Yadav and Jayswal, 2018; Florescu and Barabas, 2020). These CNC machines are equipped with tool magazines, marginalizing the internal tool switch time during processing. Appropriate planning of these machines can enable unsupervised running of jobs for extended periods, reducing the operator effort and associated overhead costs for support services. Therefore, manufacturing companies aim to improve system performance by increasing machine running hours while reducing the amount of direct supervision required for operations (Noël et al., 2007).

A CNC machine can operate unsupervised for a period if all tools required to satisfy the processing requirements for scheduled operations are loaded to the tool magazine. If the number of needed tools exceeds the tool magazine capacity, a tool switch will take place, in which one tool is removed to free up a slot to insert a needed tool. When multiple tool switches should occur at the same time to accommodate all needed tools, they belong to a so-called tool switching instance. According to Beezão et al. (2017), all activities associated with tool

* Corresponding author.
E-mail addresses: q.v.dang@tue.nl (Q.-V. Dang), k.herps@tue.nl (K. Herps), t.g.martagan@tue.nl (T. Martagan), i.adan@tue.nl (I. Adan), j.heinrich@student.tue.nl (J. Heinrich).

https://doi.org/10.1016/j.cor.2023.106361
Received 17 October 2022; Received in revised form 6 June 2023; Accepted 21 July 2023
Available online 31 July 2023
0305-0548/© 2023 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

switches (i.e., removing existing tools and loading required ones) and subsequent calibration represent about 25%–30% of the total fixed and variable costs in FMSs. It is even more in the HMLV industry due to its high product variety. This makes tool switches inevitable, and the incorporation of tooling constraints in the scheduling practices plays a vital role in enhancing the FMS productivity and utilization.

In this paper, we study a scheduling problem for a number of identical parallel CNC machines that process a set of jobs/operations while taking into account tool switches occurring in the machines. Our problem is similar to those of Beezão et al. (2017) and Dang et al. (2021), where the former considers minimizing the makespan, and the latter minimizes the total tardiness and tool setup time. Nevertheless, we extend their problems by the following features, which constitute the main novelties of our problem, to comply with industry needs.

(a) *Unsupervised production shifts*. CNC machines considered in our paper can operate unsupervised for a specified duration each day, but tool switches can only occur during supervised production hours. Hence, all the required tools must be present in the tool magazine before the beginning of unsupervised production hours.
(b) *Costs related tool switches*. Contrary to the existing literature, tool switches in our paper do not inevitably cause a delay but incur costs instead. The costs of tool switches are composed of a fixed cost per tool switching instance (e.g., preparation and measurement) and a variable cost dependent on the number of tools switched (e.g., labor cost). In this setting, the tool switches do not cause a delay due to several reasons. First, operators order the required tools in advance at the tooling department, ensuring timely delivery. In addition, the tool magazine of a CNC machine allows the loading and unloading of tools while running. In other words, operators can load the tools required for the next operation while the machine is still processing the current operation. An insignificant delay (i.e., less than 5 s) compared to the total operation processing time might only occur in an extreme case, where the last tool of the current operation makes room for the first tool of the next operation. However, the probability of encountering this case is practically negligible.
(c) *Profit maximization*. Inspired by our industry partner, this paper aims to maximize profit, i.e., the difference between the total revenue of the operations finished within a scheduling horizon and the total cost of executing these operations. The total cost here consists of the costs of tool switches and the penalty costs of the operations that are prioritized but not completed within the scheduling horizon. In our paper, the maximization of profit is relevant because it addresses an important industry characteristic, the distinction between priority and regular jobs, where each of the former incurs tardiness (resulting in a penalty cost) if not finished by the end of the horizon, while still allowing for maximum utilization of machines. Also, by considering the costs of tool switches, we indirectly minimize the number of tool switches and tool switching instances, which is well-known in the relevant literature (see, e.g., Tang and Denardo, 1988a,b; Keung et al., 2001b; Furrer and Torsten Mütze, 2017).

We will further elaborate on these distinguishing features in Section 3. In this setting, we consider the decisions on the allocation of operations to machines, the order in which operations are performed, and the tool switching plan under consideration of unsupervised production shifts and profit maximization. The main contributions of our work are summarized as follows:

(i) We develop a mixed-integer linear programming (MILP) model for our problem setting. As an important feature, the MILP model is equipped to capture the production and tooling constraints related to unsupervised hours (see Section 4).
(ii) We propose a genetic algorithm (GA), which can solve industry-sized problem instances, to improve the results of a current scheduling method being used in practice. In the proposed GA, we introduce using two crossover operators in distinct search phases to explore and exploit the solution space, where one of them is a problem-oriented crossover concentrating on increasing revenue (with more finished operations) while decreasing costs (with fewer tool switches and fewer unfinished priority operations). We also adapt a swap mutation operator to both job and operation levels, taking into account the limited tool magazine capacity (see Section 5).
(iii) We provide managerial insights based on a comprehensive numerical analysis generated from industry data. In particular, we analyze the impact of critical problem parameters on expected profit, such as unsupervised ratio (the percentage of production hours that operators are not available for tool switches), priority ratio (the percentage of jobs that are prioritized to be completed within a scheduling horizon), and tool ratio (the number of unique tools required for a set of operations). Our numerical results reveal that the proposed GA achieves high-quality solutions for realistic problem sizes, which yields around 26% improvement compared to the current industry practice.
(iv) The case study presented in this paper is provided by our industry partner, *Klein Mechanisch Werkplaats Eindhoven* (KMWE), located at the Brainport Industries Campus in the Netherlands. We make the anonymized data from KMWE publicly available for further use. The outcomes of our research are generalizable, and can be applied to settings where there is a central planner that controls a group of machines in a high-mix low-volume manufacturing environment.

The remainder of our paper is organized as follows. Related works are discussed in Section 2. Section 3 describes the problem, followed by a corresponding MILP formulation in Section 4. Section 5 proposes a GA to solve industry-size instances, while Section 6 introduces a practitioner heuristic used at the KMWE. Subsequently, we outline industry case studies, conduct parameter tuning, and present results of computational experiments in Section 7. Finally, conclusions and future works are drawn in Section 8.

2. Related work

Job scheduling with tool switches in FMS has received much attention in the literature. Ahmadi et al. (2018) present the job sequencing and tool switching problem (SSP) that focuses on sequencing a finite set of jobs and switching tools in a tool magazine with limited capacity to minimize the number of tool switches. The SSP is divided into the job sequencing problem (JSeP) and the tool replacement problem (TRP). The JSeP is a scheduling problem identifying the optimal sequencing of jobs on a machine. The TRP deals with planning to install required tools to enable processing a given sequence of jobs with the aim of minimizing the number of tool switches eventually. This decomposition of the SSP into the two sub-problems, JSeP and TRP, is also discussed by Tang and Denardo (1988a). Crama et al. (1994) prove that the SSP is NP-hard for all cases with any tool magazine capacity $C \geq 2$. The SSP is then extended to the identical parallel machines problem with tooling constraints (IPMTC), presented by Beezão et al. (2017). Consequently, the IPMTC is also NP-hard, since the SSP is seen as its special case (Dang et al., 2021). Various approaches have been proposed to deal with problems of these types, which the following paragraphs discuss in some detail.

First, several researchers have presented exact methods to solve the SSP considering a uniform tool switching time and uniform tool size to minimize the number of tool switches. They formulate different integer linear programs (ILP) to model the SSP as, e.g., a traveling salesmen problem (Laporte et al., 2004), nonlinear least cost Hamiltonian cycle problem (Ghiani et al., 2010), or multicommodity flow

model (da Silva et al., 2021). Branch-and-bound (BnB) and Branch-and-cut (BnC) algorithms, together with various bounding techniques for improving results, are also introduced to solve medium-sized problem instances (Laporte et al., 2004; Ghiani et al., 2010; Karakayalı and Azizoğlu, 2006; Catanzaro et al., 2015). Nevertheless, the results of these exact approaches are limited due to the NP-hard nature of the SSP, i.e., addressing only small and medium-sized instances with 25–40 jobs and 25–30 tools. Tang and Denardo (1988a) introduce the Keep Tool Needed Soonest (KTNS) policy to optimally solve the TRP in polynomial time, given any job sequence. This policy states that if a required tool for the next job must be inserted, the tools that should not be removed are those needed the soonest in the sequence. Variants of the SSP consider minimizing the number of switching instances (i.e., the number of machine stops) besides minimizing the number of tool switches. Tang and Denardo (1988b) propose a BnB procedure, including a maximal intersection minimal union heuristic and a sweeping heuristic, to minimize the number of tool switching instances while creating sets of jobs that can be processed together without incurring any tool switches. Furrer and Torsten Mütze (2017) also introduce a BnB algorithm to minimize both objectives for random and realistic problem instances. Another variant of the SSP is addressed by Schwerdfeger and Boysen (2017) in sequencing orders from a crane-supplied pick face, where orders and stock keeping units refer to jobs and tools, respectively. Their work minimizes the maximum number of switches between consecutive jobs by an adapted MILP from Tang and Denardo (1988a) and a BnB algorithm.

Furthermore, heuristics and meta-heuristics are the common approaches to cope with the SSP. Several researchers propose job grouping and construction heuristics (e.g., Tang and Denardo, 1988b; Crama et al., 1994; Hertz et al., 1998; Djellab et al., 2000; Salonen et al., 2006a; Burger et al., 2015; Schwerdfeger and Boysen, 2017), while others focus on developing meta-heuristics, for example, tabu search (TS) (Al-Fawzan and Al-Sultan, 2003), iterated local search (ILS) (Paiva and Carvalho, 2017), and adaptive large neighborhood search (ALNS) combined with simulated annealing (SA) (Rifai et al., 2022). Especially, population-based approaches have been implemented successfully for this problem type, such as memetic algorithm (Amaya et al., 2011, 2012) and hybrid GA (Amaya et al., 2008; Chaves et al., 2016; Ahmadi et al., 2016; Mecler et al., 2021). The SSP with multiple objectives solved by meta-heuristic methods are also studied in the works of, e.g., Keung et al. (2001b) using a GA to minimize tool switches and switching instances, Solimanpur and Rastgordani (2012) with an ant colony optimization and Baykasoğlu and Ozsoydan (2017, 2018) with SA frameworks to minimize tool switches and indexing time (i.e., the time to rotate between two tool slots). In addition, several other researchers propose heuristic methods to solve the SSP concerning unequal tool sizes, where a tool may occupy more than one slot in the tool magazine (Tzur and Altman, 2004; Raduly-Baka et al., 2005; Van Hop, 2005). Calmels (2019) classifies the SSP and its variants, concluding that further works should pay attention to more realistic problems with multiple machines and multiple objectives.

The IPMTC extending the SSP with parallel machines, in contrast, has not been much studied in the literature. Several works propose exact methods to solve problems of this type, aiming to minimize tool switches, switching instances, and/or makespan. Nonlinear programs are addressed by Sarmadi and Gholami (2011), Ghrayeb et al. (2003), and Van Hop and Nagarur (2004) for machines having different magazine capacities. Özpeynirci et al. (2016) and Gökgür et al. (2018) develop mathematical and constraint programming approaches for unrelated parallel machines with limited tool copies. Beezão et al. (2017) formulate extensions of ILP models from the works of Tang and Denardo (1988a) and Laporte et al. (2004) for the IPMTC. For multi-objectives, Keung et al. (2001a) propose an ILP model for machines with different magazine capacities and a limited number of tool copies. Calmels (2022) provides an MILP considering machine-dependent processing and tool switching times to minimize three objectives, i.e., tool switches, makespan, and flow time. Nevertheless, in general, their models can solve only small-sized instances with around 15–20 jobs, 9–10 tools, and up to 3 machines. Therefore, heuristic solutions have been favored for more industry-sized problems, e.g., TS (Özpeynirci et al., 2016), ILS (Calmels, 2022), ALNS (Beezão et al., 2017), and GA (Keung et al., 2001a; Van Hop and Nagarur, 2004). Also, Khan et al. (2000) consider the IMPTC where each job consists of several operations, each of which needs its own set of cutting tools. However, their proposed method can only be applied to two-machine instances with 22 operations and 37 tools. The main differences between our problem and the IPMTC are the following. First, we consider unsupervised production hours in which tool switches cannot occur, while the IPMTC does not. In order to deal with this, a tool-switching plan has to take into account the operations that are scheduled to execute during unsupervised hours. It means all tools required for those operations must be installed prior to the start of these unsupervised hours. Second, the IPMTC considers setup time per tool switch, which incurs idle time. However, in our problem, a tool switch does not cause a delay but incurs tool-switching costs. This links to the third difference, where our objective is to maximize the profit in which the costs of tool switches are involved. Here we maximize the profit by indirectly minimizing the number of tool switches and tool switching instances on all machines, whereas the IPMTC minimizes the makespan by minimizing the number of tool switches only on the critical machine, i.e., the most time-consuming machine. Additionally, we penalize prioritized operations if they are completed outside a scheduling horizon. Overall, the combination of these features rises the complexity of our problem over the IPMTC.

Studies on job scheduling with unsupervised production constraints are scarce. Agnetis et al. (2008) address an allocation of jobs to identical parallel machines for a fixed unsupervised period, where a job that fails during processing blocks all subsequent jobs scheduled on the allocated machine. Although rewards for completed jobs are considered, their work does not involve the tooling aspect. Noël et al. (2007) address the decision to select cutting speeds for a given number of tools that process some part types by an unsupervised metal cutting flexible machine, aiming to improve machine uptime in lights-out manufacturing. However, they do not consider tool switches, and the machine in their work is set up to operate unsupervised for one specific, finite duration only.

Our problem can be seen as an extension of the IPMTC, including three sub-problems such as machine allocation (PM), operation sequencing (JSeP), and tool replacement on machines (TRP). Our problem is also NP-hard, since a special case is the SSP, known as NP-hard (Crama et al., 1994). However, our problem has distinctive features from the literature, i.e., unsupervised production shifts, tool switches causing costs instead of time delay in processing operations, and profit maximization. To the best of our knowledge, the problem has not been addressed in the literature with this combination of features. We describe the problem in detail in the next section.

3. Problem description

This paper considers a parallel machine scheduling problem with tool switches, unsupervised shifts, and job prioritization. We consider a set of jobs J that is processed on a set of identical parallel machines M in a work center by using a set of tools T. All the jobs in J are released at the start of finite scheduling horizon H, thus scheduled all at once. Each job $j \in J$ may have multiple operations that are performed by revisiting the work center. We denote by $O_j = \{(j,k), k = 1,2,\dots,n_j\}$ the set of operations of job j, where n_j denotes the number of operations of job j, and (j,k) can be interpreted as the kth operation of job j. The order for processing the operations of job j must follow index k, i.e., $(j,k) \rightarrow (j,k+1)$ for $k = 1,2,\dots,n_j$ and $n_j > 1$. In other words, an operation of a job can only start when its preceding operation of that job is completed. We also denote by $O = \cup_{j\in J} O_j$ the set of all operations of all jobs. Each operation is non-preemptable

Table 1
Summary of parameters in Example 1.

Job j	Operation k	Processing time p_{jk}	Tool set T_{jk}	Job type
1	1	3	1	*Priority*
1	2	5	1	*Priority*
2	1	7	2	*Regular*
3	1	6	3	*Regular*
3	2	8	3	*Regular*
4	1	4	2	*Priority*
4	2	9	2	*Priority*
5	1	6	4	*Regular*
6	1	10	5	*Regular*
7	1	5	1	*Priority*

and has the sequence-independent setup time that is included in the operation's processing time, denoted as p_{jk}. In addition, a subset $J_P \subseteq J$ is categorized as priority jobs, and all the operations of these jobs, denoted by O_P, are categorized as priority operations. This job prioritization adds practical relevance, where the priorities may be caused by a variety of reasons, such as approaching deadlines, shifting requirements of customers, or requiring crucial components for the assembly of an entire product module. The other jobs, also the other operations, are considered regular.

In practice, demand or demand forecasts creating jobs and operations may be known for a period greater than scheduling horizon H (e.g., $\geq$ a week). Nevertheless, since the length of H is finite, it may happen that only part of all the operations in O can be completed within H. Let O_F, where $O_F \subseteq O$, denote a subset of operations that is finished within H. This subset can contain both priority and regular operations. Each operation in O_F, either priority or regular, generates a fixed revenue of r that is added up to the total revenue, denoted by $\mathcal{R}$. The remaining operations in O, which are not finished within H, may also consist of both priority and regular ones. Nevertheless, in contrast to those in O_F, they do not generate revenue in the scheduling horizon. Instead, the unfinished priority operations, denoted by O_U, all incur a total penalty cost of $C_{\mathcal{P}}$ with a cost rate of c_p per operation, whereas the unfinished regular ones do not. The remaining operations, together with newly arrived demand, will be considered for the next horizon.

Each operation of a job can be carried out on any machine in the work center. It means that different operations of the same job can be processed on different machines or on the same machine. Each machine $m \in M$ can process one operation (j,k) at the time, and each operation $(j,k) \in O$ can only be processed by one machine m at the time. Every machine has a tool magazine with the same limited capacity T_C for holding tools employed in processing. A machine m can only perform operation (j,k) when all tools $t \in T_{jk}$ are present in the magazine of machine m, where $T_{jk} \subseteq T$ is the predefined tool set used for operation (j,k). Each tool set T_{jk} is a unique collection of tools that has a size of $|T_{jk}|$ indicating the number of tools in the tool set. Different tool sets may contain some common tools that can be used for processing different operations. Note that all operations of the same job require the same tool set.

Example 1. Table 1 presents a problem containing 7 jobs. Three of them have reentrant operations, thus a total of 10 operations. Also, there are three priority jobs (five priority operations) among them. All the operations are processed on 2 identical parallel machines and require 5 unique tool sets. Each machine has a tool magazine capacity $T_C = 8$. The composition and size of each tool set are presented in Table 2. In this example, operations $(1,1)$, $(2,1)$, and $(3,1)$ require tool sets 1, 2, and 3, respectively. Here t_4 and t_5 are the common tools for $(1,1)$ and $(3,1)$, while t_{12} and t_{13} are the common tools for $(2,1)$ and $(3,1)$. We can also see that each of jobs 1, 3, 4 requires the same tool set to process all its operations.

The machines can keep track of the tools present in their magazines, thus indicating the missing tools from the required tool sets. Therefore,

Table 2
Tool set information in Example 1.

Tool set T_{jk}	Tools $t \in T_{jk}$	Size $\lvert T_{jk}\rvert$
1	$\{t_1, t_2, t_3, t_4, t_5\}$	5
2	$\{t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}\}$	7
3	$\{t_4, t_5, t_8, t_9, t_{10}, t_{11}, t_{12}, t_{13}\}$	8
4	$\{t_5, t_6, t_7\}$	3
5	$\{t_{15}, t_{16}, t_{17}, t_{18}, t_{19}, t_{20}\}$	6

only the missing tools from tool set T_{jk} need to be inserted for the start of operation (j,k). In most practical cases, the insertion of a missing tool may require the removal of a current tool in the magazine since the limited capacity T_C cannot contain all tools at once. We can see an example of this in Example 1. If operation $(3,1)$ is processed right after $(1,1)$, tools t_1, t_2, and t_3 need to be removed before inserting the missing tools of tool set 3 required for $(3,1)$ into the machine's magazine. It indicates that a tool switch only happens when a tool required for an operation is not present in the same machine processing the preceding operation. Here one tool switch corresponds to one inserted tool that is required for an operation (for example, to process $(3,1)$ after $(1,1)$, we have to insert six tools ($t_8, \ldots, t_{13}$), thus six tool switches counted).

Moreover, motivated by our industry partner, the machines in our paper, while working, allow operators to switch tools in the magazines. Consequently, tool switches do not incur any intermediate delay in the schedule. Instead, costs are incurred for executing tool switches. At any instance in which one or more tool switches occur simultaneously (referred to as tool switching instance), we incur a fixed cost that has a cost rate of c_f per instance. In addition, each tool switch causes a variable cost with a cost rate of c_v per tool switch. The total tool switching cost, denoted by $C_{\mathcal{T}}$, is the sum of all the fixed and variable costs. It can be seen that $C_{\mathcal{T}}$ is proportional to the number of tool switches and the number of tool switching instances. Therefore, the decision to execute tool switches needs to be taken carefully into account since removing a tool may cause additional fixed and variable costs if that tool is required for any subsequent operation on the same machine.

Furthermore, tool switches are influenced by unsupervised shifts in which operator availability constrains the manufacturing. Namely, tool switches are performed by operators and can only occur during supervised shifts with the presence of the operators. As a result, the operations requiring missing tools cannot start during unsupervised shifts. These operations are delayed and begin in the next supervised shift. Therefore, the decision for tool switching also affects how long the work center can continue manufacturing without supervision each day. The longer it can, or the more operations it can process during the night, the more revenue it can generate. Without loss of generality, all days are assumed to have a uniform unsupervised shift length of t_U hours during the night, thus $(24 - t_U)$ hours for each supervised shift.

In this paper, our goal is to obtain a schedule that determines (a) to which machine operations are assigned, (b) in which sequence operations are carried out on machines, and (c) a tool switching plan on machines such that production constraints are satisfied. The objective is to maximize the profit which is the difference between total revenue $\mathcal{R}$ and total cost C, i.e., $profit = \mathcal{R} - C$, where $C = C_{\mathcal{P}} + C_{\mathcal{T}}$. Here a trade-off is made between total revenue $\mathcal{R}$ resulting from the finished operations within the scheduling horizon and total cost C caused by producing these operations. We visualize this trade-off in Fig. 1.

A scheduling horizon starts with a set of jobs (operations) containing both priority and regular ones. Each finished priority or regular operation increases the profit by a revenue rate of r. On the other hand, not finishing any priority operation reduces the profit by a penalty rate of c_p. Hence, priority operations should be produced within the scheduling horizon if possible, while regular ones can be produced in the horizon to increase the revenue, but could also be produced in future periods without incurring penalty costs. However, finite time

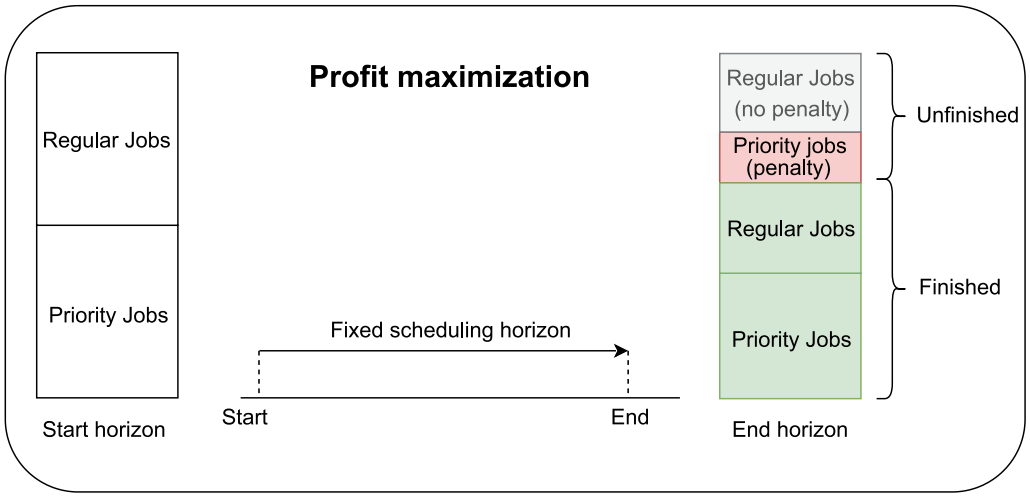

Fig. 1. Profit maximization with penalty costs.

and resources limit the number of finished operations. Therefore, determining which operations to be produced within the finite horizon is crucial for maximizing the profit. Also, the profit is affected by the costs associated with the tool switching plan. Finishing more operations tends to gain more revenue but may cause more tool switches, thus more tool-associated costs due to the diversity in processing requirements in the high-mix, low-volume, high-complexity manufacturing environment. Further, the tool switching plan is constrained by limited operator availability. One can think of allocating and sequencing operations in such a way that can prolong production in unsupervised shifts to generate more revenue. Nevertheless, it may lead to a negative influence on scheduling in supervised shifts, i.e., more tool switches. Hence, a smart, holistic decision-making method is needed to cope with this complex challenge to maximize profit.

In this paper, we take the following assumptions into account. First, the size of any tool set T_{jk} is at most equal to the magazine capacity, i.e., $\max\{|T_{jk}|, \forall T_{jk} \subseteq T\} \leq T_C$. Otherwise, the machines cannot be used, because the tool sets do not fit into their magazines. Second, each tool has a sufficient number of copies for manufacturing in practice, hence we do not consider its limitation. This assumption implies that a tool's copies can simultaneously be present in multiple machines' magazines. Third, no tool wear, tool fractures, or other types of machine malfunctions that might lead to quality rejections of processed items are considered. Hence, costs of electricity, depreciation of machines, and other factors are not considered.

Example 2. We illustrate a feasible solution for the problem in Example 1 with Fig. 2. Here, we consider an unsupervised shift of 12 h and a scheduling horizon of 48 h. As seen in this schedule, five operations are processed on machine 1, and four are processed on machine 2; each generates revenue at a rate of r. Operation $(7,1)$ remains unfinished after 48 h, incurring a penalty cost of c_p because it is a priority operation. Also, it can be seen that several operations are continued during unsupervised shifts, e.g., part of operation $(2,1)$ is processed in hours 12–15. After machine 1 finishes operation $(2,1)$, operation $(3,1)$ cannot start immediately and is delayed until the next supervised shift, leaving machine 1 idle in hours 15–24. This is because installing required tools for operation $(3,1)$ cannot happen during unsupervised hours. In addition, this schedule has no delay caused by tool switches, e.g., when machine 1 switches tool set 1 to 2, operation $(2,1)$ can start right after operation $(1,2)$. Instead, costs are incurred due to tool-switching activities, e.g., the cost to switch from tool set 1 to 2 is $c_f + 7c_v$ (fixed cost per instance + variable cost for inserting 7 individual tools of tool set 2).

4. Mathematical formulation

In this section, a mathematical model is formulated based on the problem description in Section 3. The model captures the objective of maximizing profit with the production constraints on unsupervised shifts and tool-switching requirements.

4.1. Decision variables

We introduce the following decision variables, in addition to the notation presented in Section 3.

s_{jk}	starting time of operation (j,k)
e_{jk}	ending time of operation (j,k)
$x^m_{jkj'k'}$	equal to 1 if operation (j,k) is directly followed by operation (j',k') on machine m, 0 otherwise
β^m_{jk}	equal to 1 if operation (j,k) is assigned to machine m, 0 otherwise
α_{jk}	equal to 1 if operation (j,k) is completed within scheduling horizon H, 0 otherwise
y^t_{jk}	equal to 1 if tool t is present at the start of processing operation (j,k), 0 otherwise
z^t_{jk}	equal to 1 if tool t is inserted at the start of operation (j,k), 0 otherwise
l_{jk}	equal to 1 if a tool switching instance occurs at the start of operation (j,k), 0 otherwise

4.2. Mixed-integer linear programming model

We formulate a mathematical model for the described problem as follows.

Objective (1) maximizes the profit $= \mathcal{R} - \mathcal{C}_{\mathcal{P}} - \mathcal{C}_{\mathcal{T}}$, where $\mathcal{R} = r\sum_{(j,k)\in O}\alpha_{jk}$ (the total revenue from finished operations), $\mathcal{C}_{\mathcal{P}} = c_p\sum_{(j,k)\in O_P}(1-\alpha_{jk})$ (the total penalty cost of unfinished priority operations), and $\mathcal{C}_{\mathcal{T}} = c_f\sum_{(j,k)\in O}\alpha_{jk}l_{jk} + c_v\sum_{(j,k)\in O}\sum_{t\in T}\alpha_{jk}z^t_{jk}$ (the total tool switching cost).

$$\max \quad \Big(r\sum_{(j,k)\in O}\alpha_{jk} - c_p\sum_{(j,k)\in O_P}(1-\alpha_{jk}) - \Big(c_f\sum_{(j,k)\in O}\alpha_{jk}l_{jk} + c_v\sum_{(j,k)\in O}\sum_{t\in T}\alpha_{jk}z^t_{jk}\Big)\Big) \tag{1}$$

A feasible solution must satisfy the following constraints:

$$\sum_{m\in M}\sum_{\substack{(j',k')\in O\\(j',k')\neq(j,k)}} x^m_{jkj'k'} \leq 1 \qquad \forall (j,k)\in O \tag{2}$$

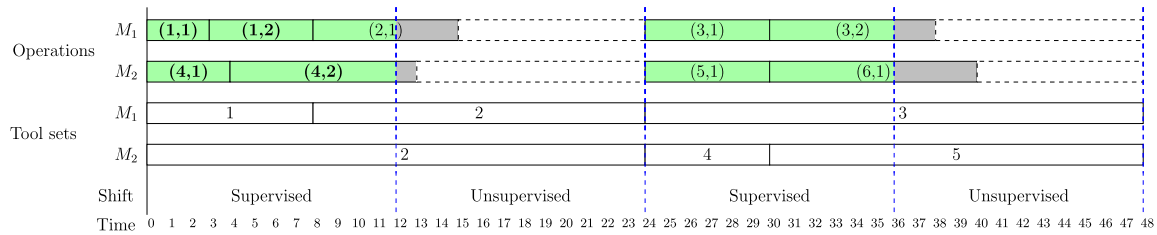

Fig. 2. Gantt chart of a feasible schedule for Example 1. The chart consists of two parts. The upper part "Operations" presents the operations sequenced on the machines. The lower part "Tool sets" indicates the tool sets present while processing the corresponding operations in the upper part. The vertical, blue-dashed lines separate supervised and unsupervised shifts. The green bars ▢ present finished operations, both priority and regular. The bold index (j,k) inside the green bars indicates those finished are priority operations. The grey parts ▢ indicate that part of operation processing is performed during unsupervised shifts. The white-dashed bars ▢ represent machine idle time, resulting from unavailable tools required for operations scheduled next in the sequence.

$$\sum_{m\in M}\sum_{\substack{(j,k)\in O\\(j,k)\neq(j',k')}} x^m_{jkj'k'} \le 1 \qquad \forall (j',k')\in O \tag{3}$$

$$\sum_{m\in M}\beta^m_{jk} = 1 \qquad \forall (j,k)\in O \tag{4}$$

$$2x^m_{jkj'k'} \le \beta^m_{jk} + \beta^m_{j'k'} \qquad \forall (j,k)\in O,\ \forall (j',k')\in O\backslash\{(j,k)\}, \forall m\in M \tag{5}$$

$$\sum_{(j,k)\in O}\sum_{\substack{(j',k')\in O\\(j',k')\neq(j,k)}} x^m_{jkj'k'} + 1 \ge \sum_{(j,k)\in O}\beta^m_{jk} \qquad \forall m\in M \tag{6}$$

Constraints (2) and (3) make sure that each operation can only be preceded and followed by at most one operation, respectively. Constraints (4) impose that each operation is assigned to one machine only. Constraints (5) and (6) enforce that two successive operations must be carried out on the same machine.

$$e_{jk} = s_{jk} + p_{jk} \qquad \forall (j,k)\in O \tag{7}$$

$$s_{jk} + \mathcal{L}(1 - x^m_{j'k'jk}) \ge e_{j'k'} \qquad \forall (j,k)\in O, \forall (j',k')\in O\backslash\{(j,k)\}, \forall m\in M \tag{8}$$

$$e_{jk} \le s_{i,j+1} \qquad \forall i\in I, j=(1,\dots,n_i-1), n_i>1 \tag{9}$$

Constraints (7) make sure that the ending time of an operation is constrained by its starting time and processing time. Constraints (8) guarantee that an operation can only start after its preceding operation on the same machine is completed, where $\mathcal{L} = \sum_{(j,k)\in O} p_{jk} + \left\lceil H\frac{t_U}{24}\right\rceil$ is a large number. Constraints (9) enforce the precedence relations between two operations of the same job.

$$\frac{H-e_{jk}}{\mathcal{L}} < \alpha_{jk} \le 1 - \frac{e_{jk}-H}{\mathcal{L}} \qquad \forall (j,k)\in O \tag{10}$$

Constraints (10) ensure that if an operation is completed within the scheduling horizon (i.e., $e_{jk} \le H$, thus $\alpha_{jk} = 1$), it accounts for a revenue of R in Objective (1). Otherwise (i.e., $e_{jk} > H$, thus $\alpha_{jk} = 0$), a penalty cost is incurred if that operation is a priority one.

$$\sum_{t\in T} y^t_{jk} \le T_C \qquad \forall (j,k)\in O \tag{11}$$

$$x^m_{j'k'jk} + y^t_{jk} - y^t_{j'k'} \le z^t_{jk} + 1 \qquad \forall (j,k)\in O, \forall (j',k')\in O\backslash\{(j,k)\}, \forall m\in M, \forall t\in T \tag{12}$$

$$y^t_{jk} = 1 \qquad \forall (j,k)\in O, \forall t\in T_{jk} \tag{13}$$

$$z^t_{jk} \le l_{jk} \qquad \forall (j,k)\in O, \forall t\in T \tag{14}$$

$$l_{jk} \le \sum_{t\in T} z^t_{jk} \qquad \forall (j,k)\in O \tag{15}$$

Constraints (11) impose that the sum of all tools present at the start of an operation does not exceed the tool capacity. Constraints (12) states that a tool switch occurs (i.e., $z^t_{jk} = 1$) when tool t required for operation (j,k) is not present in the tool magazine during the processing of preceding operation (j',k'). Constraints (13) ensure that all required tools for operation (j,k) are present at the start of the operation. Constraints (14) and (15) make sure that each tool switch results in a tool switching instance, and vice versa, i.e., that at each tool switching instance, one or more tools are switched.

We want to note that, by Constraints (14), we do not restrict that a tool switch can only be performed when tool t is required to immediately process operation (j,k), as considered in the literature (i.e., $z^t_{jk} = 0$, $\forall (j,k)\in O$, $\forall t\in T\setminus T_{jk}$ (Beezão et al., 2017; Dang et al., 2021)). In order words, we can insert a tool into the magazine earlier than the time the tool is needed to process a subsequent operation in the schedule. By doing so, it can keep machines running in unsupervised shifts when an operator is not present for tool switches.

$$l_{jk} \le 2 - \frac{s_{jk} \bmod 24}{24-t_U} \qquad \forall (j,k)\in O \tag{16}$$

Constraints (16) impose that tool switching instances can only occur during supervised shifts, i.e., during the first $(24-t_U)$ supervised hours of each day, where $(s_{jk} \bmod 24)$ is a modulo that determines the time (hour) of a day at which operation (j,k) begins. For example, if $s_{jk} = 30$ and $t_U = 12$, then $l_{jk} \le 1.5$, which indicates that a tool switching instance may happen at hour 30. Constraints (17)–(21) ensure valid domains for all variables.

$$x^m_{jkj'k'} \in \{0,1\} \qquad \forall (j,k)\in O, \forall (j',k')\in O, \forall m\in M \tag{17}$$

$$\beta^m_{jk} \in \{0,1\} \qquad \forall (j,k)\in O, \forall m\in M \tag{18}$$

$$y^t_{jk}, z^t_{jk} \in \{0,1\} \qquad \forall (j,k)\in O, \forall t\in T \tag{19}$$

$$\alpha_{jk}, l_{jk} \in \{0,1\} \qquad \forall (j,k)\in O \tag{20}$$

$$s_{jk}, e_{jk} \ge 0 \qquad \forall (j,k)\in O \tag{21}$$

Since Objective (1) is nonlinear, it is linearized as follows. We define two additional binary variables δ_{jk} and λ^t_{jk}, where:

$$\delta_{jk} = \alpha_{jk} l_{jk} \qquad \forall (j,k)\in O \tag{22}$$

$$\lambda^t_{jk} = \alpha_{jk} z^t_{jk} \qquad \forall (j,k)\in O, \forall t\in T \tag{23}$$

Objective (1) is then rewritten as Objective (24).

$$\max \quad \Big(r\sum_{(j,k)\in O}\alpha_{jk} - c_p\sum_{(j,k)\in O_P}(1-\alpha_{jk}) - \Big(c_f\sum_{(j,k)\in O}\delta_{jk} + c_v\sum_{(j,k)\in O}\sum_{t\in T}\lambda^t_{jk}\Big)\Big) \tag{24}$$

Constraints (22) are replaced by Constraints (25)–(27) due to the nonlinerization. Constraints (23) are also replaced in a similar manner.

$$\delta_{jk} \le \alpha_{jk} \qquad \forall (j,k)\in O \tag{25}$$

$$\delta_{jk} \le l_{jk} \qquad \forall (j,k)\in O \tag{26}$$

$$\delta_{jk} \ge \alpha_{jk} + l_{jk} - 1 \qquad \forall (j,k)\in O \tag{27}$$

Moreover, since the parallel machines are identical, there exist many alternative solutions that are similar but mirror the allocation of operations over the machines. Hence, we exclude those alternatives by adding symmetry-breaking constraints (28) (Sherali and Cole Smith,

2001) to avoid searching for them, thus boosting the performance of the MILP.

$$\sum_{(j,k)\in O} \beta_{jk}^{m-1} \geq \sum_{(j,k)\in O} \beta_{jk}^{m} \qquad \forall m \in M\backslash\{1\} \tag{28}$$

We observe that the mathematical models proposed in several relevant works start struggling even for a small set of operations (e.g., 15 and 25 operations for the works of Beezão et al. (2017) and Dang et al. (2021), respectively). Our problem, an NP-hard problem as discussed, is more complex than those, which implies that our MILP can encounter a similar issue. It is demonstrated in our experiments (see Section 7), where the MILP finds it hard to solve the problem from 25 operations onward. In addition, it requires a computational effort that may increase exponentially in medium and large-sized problems with more operations and machines. It reinforces the necessity of developing another method. Therefore, a GA is presented in the next section.

5. Proposed genetic algorithm

In this section, we propose a GA, a population-based approach that has been successfully employed in the literature for studies of job scheduling with tool switches. Compared to alternative optimization methods, GA is widely recognized for its effectiveness in conducting global searches and reducing computational requirements. Also, GA offers the flexibility to be combined with other approaches, enabling the creation of more efficient implementations for various optimization problems (Gen et al., 2008a). Specifically, in this paper, our GA is enhanced by applying a hybrid method of the job grouping and the KTNS policy to reduce both tool switches and tool switching instances (Salonen et al., 2006b). Let P_κ, C_κ, and C'_κ denote the parents, offspring from crossover operations, and offspring from mutation operations of generation κ, respectively. Also, let f_κ and f_{best} denote the best (maximum) fitness value at generation κ and the best fitness value over generations until generation κ, respectively. The pseudocode of the proposed GA is presented in Algorithm 1.

Algorithm 1 Genetic algorithm

Require: set of operations O, set of machines M, set of tools T
1: $\kappa \leftarrow 1$, $best \leftarrow$ **false**
2: Initialize P_κ ▷ Section 5.2
3: Evaluate P_κ ▷ Section 5.8
4: $f_{best} \leftarrow \min\{f_v : \text{chromosome } v \in P_\kappa\}$
5: **repeat**
6: **if** $best =$ **true** $\vee\ q \leq Q$ **then**
7: Create C_κ from P_κ using problem-oriented crossover ▷ Section 5.3.2
8: **else**
9: Create C_κ from P_κ using combined crossover ▷ Section 5.3.1
10: **end if**
11: Create C'_κ from C_κ by swap mutation and uniform mutation ▷ Section 5.4
12: Merge classes S to maximal classes for C'_κ ▷ Section 5.5
13: Evaluate C'_κ ▷ Section 5.8
14: Generate $P_{\kappa+1}$ from P_κ and C'_κ by elitism and immigration ▷ Section 5.6
15: $f_\kappa \leftarrow \min\{f_v : \text{chromosome } v \in P_{\kappa+1}\}$
16: **if** $f_\kappa > f_{best}$ **then**
17: $f_{best} \leftarrow f_\kappa$, $best \leftarrow$ **true**, $q \leftarrow 1$
18: **else**
19: $best \leftarrow$ **false**, $q \leftarrow q + 1$
20: **end if**
21: $\kappa \leftarrow \kappa + 1$
22: **until** termination criteria met

The proposed algorithm is initialized with the first generation of parents P_κ that is created randomly and seeded with good initial solutions from a practitioner heuristic presented in Section 6 and a BnB grouping procedure described in Section 5.2.1 (line 2). This initial population is evaluated, and the best parent is recorded (lines 3–4). Next, the algorithm processes the parents of generation κ with crossover operations. If a new best solution is found, it benefits from using a problem-oriented crossover for the next Q generations to exploit the search area around this best solution, where q is the index to keep track of the number of times the problem-oriented crossover is used (lines 6–7 and 16–20). On the other hand, a combined crossover consisting of two-point and adapted partial-mapped crossovers is used to explore the search space (lines 8–10). Afterward, swap and uniform mutations are performed on offspring C_κ (line 11). The algorithm then merges classes (each of which consists of a number of operations) in each offspring C_κ to maximal classes (see Section 5.2.1 for more details) to prevent unnecessary tool switching instances (line 12). These resulting offspring C'_κ are evaluated, and the next generation $P_{\kappa+1}$ is created from parents P_κ and offspring C'_κ using an elitism and immigration mechanism to push the algorithm towards improved solutions (lines 13–14). The algorithm is terminated when it reaches the maximum computational time $maxTime$ or no best solution f_{best} is found for G_c consecutive generations. More details about the respective components of the algorithm are provided in the sections stated in comments in Algorithm 1.

5.1. Genetic representation

Encoding solution plays a crucial role in GAs. In our paper, due to the reentry of operations and the resulting precedence constraints, a standard permutation encoding scheme may cause infeasible solutions. Therefore, we adapt the encoding scheme presented by Gao et al. (2008) to deal with this problem. A chromosome using this encoding scheme consists of a vector of jobs in processing order (called job vector) and another vector representing the machine allocation (called machine vector). However, different from Gao et al. (2008), we use a job vector to represent operations that are processed subsequently without requiring a tool switching instance, referred to as maximal classes S (Tang and Denardo, 1988b). Maximal classes S are groups of operations requiring a set of tools that can fit together in the tool magazine of a machine. This includes any operation that would be added to this maximal class S and is not yet added to another maximal class, would require tool switches. Consequently, the total number of tools required for producing the operations of each maximal class is smaller than or equal to the tool magazine capacity, and tool switches occur between maximal classes (see Section 5.2.1 for more details of maximal class). The content of maximal classes within the job vector is not fixed, i.e., operations may switch between maximal classes throughout the GA process. One can note that constructing maximal classes does not limit the heuristic's ability to find optimal solutions but prevents unnecessary tool switching instances by combining operations that can be produced sequentially without requiring tool switches.

Accordingly, each chromosome v consists of n genes, with n being equal to the number of maximal classes S. Each gene v_g $(g = 1, \dots, n)$ consists of two elements, i.e., a maximal class S_g and a machine M_g processing this maximal class, so $v_g = \{S_g, M_g\}$. Hence, a chromosome can be represented as a sequence of genes, i.e., $(v_1, \dots, v_n) = (\{S_1, M_1\}, \dots, \{S_n, M_n\})$. In each chromosome, let $V^S = \{S_g : 1 \leq g \leq n\}$ and $V^{\mathcal{M}} = \{M_g : 1 \leq g \leq n\}$ denote the job vector and the machine vector, respectively. In the maximal classes S of the job vector, each job j occurs exactly n_j times, same as the number of operations of that job. Here, the order of occurrences of index j can be interpreted as the increasing order of the operations of job j. This handles the precedence constraints between each job's operations, thereby always resulting in a feasible schedule.

Example 3. The encoding scheme is illustrated by a solution for Example 1 in Fig. 4(a). In this figure, we add a layer of *Operations O* representing the operations in the maximal classes of job vector V^S.

Then, job vector V^S can be translated into the following operation sequence: $(1,1) \succ (1,2) \succ (2,1) \succ (4,1) \succ (4,2) \succ (3,1) \succ (3,2) \succ (5,1) \succ (6,1) \succ (7,1)$. Here the reentry of jobs is handled implicitly. For example, job 1 has two operations represented by S_1. These operations are subsequently interpreted as the first operation of job 1 $(1,1)$ and the second operation of job 1 $(1,2)$. Additionally, we provide another layer, i.e., *Tool sets* T, to indicate the tool set required for processing the operations of maximal class S_g on machine M_g.

5.2. Initialization

This section presents the initialization procedure generating the initial population. Often randomly generating solutions is the most common method to diversify the population and explore the search space for better solutions. On the other hand, seeding the population with potentially good solutions found by other heuristics may help GAs to improve their performance (Gen et al., 2008b). Therefore, we employ both methods to generate the initial population. First, we seed the initial population with one solution from a practitioner heuristic (see Section 6) and 5% number of solutions from a BnB grouping method (see Section 5.2.1). Next, we complement the initial population by adding randomly generated solutions until reaching the population size N_p. All initial solutions are generated with maximal classes to push the proposed GA towards reducing tool switching instances.

5.2.1. Branch-and-bound method

One of the ways to initialize chromosomes is using a non-LP-based BnB method proposed by Tang and Denardo (1988b). This BnB method is a heuristic that partitions the set of operations O in the minimum number of maximal classes S to minimize tool switching instances. It solves the job grouping problem by combining the Maximal Intersection Minimal Union (MIMU) and sweeping procedures, which are both described as follows.

First, a set of operations S is defined as a class if it needs no more than T_C tools altogether. For any given set of operations O, a class of operations S, where $S \subseteq O$, is called a maximal class of set O if the following properties are satisfied:

1. $|\cup_{(j,k)\in S} T_{jk}| \le T_C$
2. $|\cup_{(j,k)\in S\cup\{(j',k')\}} T_{jk}| > T_C$, for any operation $(j',k') \notin S$

where Properties 1 shows that S is a class complying with the tool magazine capacity, and Properties 2 indicates that S is a maximal class, i.e., no operation from the remaining operations can be added to this class without requiring tool switches.

Let S_i be a maximal class with respect to $\mathcal{O}_i$, where $\mathcal{O}_i$ is the set of operations obtained by removing the set $\{S_1 \cup \cdots \cup S_{i-1}\}$ from set O, for $i = 2,\dots,\mathcal{N}$. Here, a partition $\{S_1,\dots,S_{\mathcal{N}}\}$ of set O is called a sequential maximal partition if S_i is a maximal class of set $\mathcal{O}_i$ that contains operations $(j,k) \in \mathcal{O}_i$, where $\mathcal{O}_{i+1} = \mathcal{O}_i \setminus S_i$ for $i = 1,\dots,\mathcal{N}-1$. Then, the MIMU heuristic finds an upper bound for the number of maximal classes by constructing a sequential maximal partitioning of operations O. The MIMU expands a class S by selecting an operation (j',k') that maximizes the total number of common tools required by operation (j',k') and class S, i.e., $\max\{|T_{j'k'} \cap \{\cup_{(j,k)\in S} T_{jk}\}|\}, \forall (j',k') \notin S$. If there is a tie in this selection, an operation (j',k') is selected to minimize the number of tools required by operation (j,k) but not by the operations in class S, i.e., $\min\{|T_{j'k'} - \{\cup_{(j,k)\in S} T_{jk}\}|\}, \forall (j',k') \notin S$. A pseudocode of the MIMU can be seen in Algorithm E.1 (see Appendix E.1).

On the other hand, the sweeping procedure finds a lower bound for the number of maximal classes. This procedure sweeps away operation (j,k) compatible with the fewest number of operations and all other operations that are compatible with this operation. Nevertheless, there might be cases that every pair of operations is compatible. It leads to the lower bound equal to 1, which indicates a not-tight lower bound. Hence, this method is enhanced by checking a possibly better lower bound for these cases, i.e., $SW = \max\{\lceil |\cup_{(j,k)\in O} T_{jk}| / T_C \rceil, L\}$, where L is the lower bound computed by the sweeping procedure for a set O. A pseudocode of this procedure can be seen in Algorithm E.2 (see Appendix E.2).

Finally, the BnB method finds a partition of set O in the minimum number of maximal classes by repeatedly executing the MIMU and sweeping procedures. It starts with an initial upper bound obtained from the MIMU and a lower bound obtained from the sweeping procedure. Then, the gap between the upper and lower bound is reduced throughout the loop. This method terminates when there are no non-terminated branching nodes left in the search tree. The resulting partition of set O with the minimum number of maximal classes S prevents intra-class tool switches and reduces inter-class tool switches. A pseudocode of the BnB method can be seen in Algorithm E.3 (see Appendix E.3).

Example 4. The BnB method is demonstrated using the data from Example 1 (see Tables 1 and 2).[1] According to the MIMU, operations are sequentially grouped into maximal classes, resulting in the following partitioning of operations: $\{S_1, S_2, S_3, S_4\}$, where $S_1 = \{(3,1),(3,2)\}$, $S_2 = \{(2,1),(4,1),(4,2)\}$, $S_3 = \{(6,1)\}$, and $S_4 = \{(1,1),(1,2),(7,1),(5,1)\}$. Then, the sweeping procedure is done by sequentially sweeping away the minimum compatible operation together with the operations compatible with this operation. This results in the following order of sweeping operations: $S_1 = \{(6,1)\}$, $S_2 = \{(3,1),(3,2)\}$, $S_3 = \{(2,1),(4,1),(4,2)\}$, and $S_4 = \{(1,1),(1,2),(5,1),(7,1)\}$. Here, the current upper bound U^* found by the MIMU is equal to the current lower bound L^* from the sweeping procedure, i.e., $U^* = L^* = 4$. The BnB method is terminated, and the resulting partition with the minimum number of maximal classes is obtained.

5.3. Crossover

The GA performance is significantly influenced by crossover operators (Gen and Cheng, 1999). In this paper, crossover operates on two chromosomes chosen using tournament selection. Tournament selection contains random and deterministic selection features by randomly choosing a set of chromosomes from the set of parents in generation P_κ and picking the chromosomes with the best (highest) fitness values from this set for reproduction. The randomly chosen set of chromosomes is chosen with a tournament selection rate of γ_1, where $0 < \gamma_1 \le 1$, and $S_T = \gamma_1 \times N_p$ is the size of the chosen set of chromosomes. Tournament selection and crossover continue until N_p offspring are obtained. This section presents two crossover methods[2] used within our GA: combined crossover (CX) (Section 5.3.1) and problem-oriented crossover (POX) (Section 5.3.2). CX is used as a generic search method to diversify the search, while POX is applied for a Q number of generations after a new best solution is found before using CX again. The main aim is to take advantage of two crossover types in distinct phases to search more efficiently.

5.3.1. Combined crossover

CX consists of a two-point crossover (2X) and an adapted partial-mapped crossover (APMX). The CX simultaneously operates on job vector V^S and machine vector $V^{\mathcal{M}}$ of selected parent chromosomes to create offspring. Hence, maximal classes S_g from the job vector remain allocated to the same machine M_g of the machine vector. First, the 2X randomly selects two cutting points referring to the positions of genes v_g in the range $[0,\dots,n]$ of the parent chromosomes. The substring

[1] Step-by-step illustration for the BnB method's example can be seen at https://github.com/vinhise/pmstsup.

[2] Pretest of different crossover operators can be seen at https://github.com/vinhise/pmstsup/.

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Operations O	1 1	2	4 4	3 3	5	6	7
Machine vector $V^{\mathcal{M}}$	1	1	2	1	2	2	2
Tool sets T	1	2	2	3	4	5	1

Fig. 3. Encoding scheme of a solution for Example 1.

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Operations O	1 1	2	4 4	3 3	5	6	7
Machine vector $V^{\mathcal{M}}$	1	1	2	1	2	2	2
Tool sets T	1	2	2	3	4	5	1

(a) Chromosome P_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6
Operations O	4 4	2	3 3	1 1	5 7	6
Machine vector $V^{\mathcal{M}}$	1	2	2	1	2	1
Tool sets T_{ij}	2	2	3	1	4, 1	5

(b) Chromosome P_{κ_2}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Operations O	1 1	2	3 3	1 1	5	6	7
Machine vector $V^{\mathcal{M}}$	1	2	2	1	2	2	2
Tool sets T_{ij}	1	2	3	1	4	5	1

(c) Chromosome C_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6
Operations O	4 4	2	4 4	3 3	5 7	6
Machine vector $V^{\mathcal{M}}$	1	1	2	1	2	1
Tool sets T_{ij}	2	2	2	3	4, 1	5

(d) Chromosome C_{κ_2}

Fig. 4. Offspring chromosomes after 2X (Exchanged operations, Duplicated operations).

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6	S_7
Operations O	4 4	2	3 3	1 1	5	6	7
Machine vector $V^{\mathcal{M}}$	1	2	2	1	2	2	2
Tool sets T_{ij}	2	2	3	1	4	5	1

(a) Chromosome C_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6
Job vector $V^{\mathcal{S}}$	S_1	S_2	S_3	S_4	S_5	S_6
Operations O	1 1	2	4 4	3 3	5 7	6
Machine vector $V^{\mathcal{M}}$	1	1	2	1	2	1
Tool sets T_{ij}	1	2	2	3	4, 1	5

(b) Chromosome C_{κ_2}

Fig. 5. Offspring chromosomes after APMX (Exchanged operations, Replaced operations).

within these two points is exchanged between the parents. Second, the APMX performs a mapping step on the resulting chromosomes to ensure that all operations occur precisely once. This mapping of operations is done by linking operations between the substrings and then replacing the duplicate operations outside the substrings (Dang et al., 2021).

Example 5. We illustrate the CX using Example 1 in Section 3 (see Tables 1 and 2). Specifically, chromosome P_{κ_1} in Fig. 4(a) (i.e., the example solution presented in Fig. 3) and an additional chromosome P_{κ_2} in Fig. 4(b) are selected for crossover. First, two cutting points are randomly selected. In this example, v_2 and v_4 are chosen. Hence, the substring with classes $S_2, S_3,$ and S_4 is exchanged from both chromosomes. This generates offspring chromosomes C_{κ_1} and C_{κ_2}, as shown in Figs. 4(c) and 4(d), respectively. In these offspring, the gray parts are the exchanged parts of both solutions, and the red operations become duplicate operations in these solutions. Then, the APMX is applied to these offspring to ensure that each operation appears exactly once in each solution. The resulting C_{κ_1} and C_{κ_2} after the APMX are presented in Figs. 5(a) and 5(b), respectively, where the green parts are the operations that are replaced using the APMX.

5.3.2. Problem-oriented crossover

The CX, composed of 2X and APMX, can be seen as generic crossovers that do not consider specific characteristics of the problem when generating offspring. While these generic crossovers aid in diversifying the search, they can also consume computational resources by exploring unpromising regions. Therefore, in order to improve the GA performance, we propose a problem-oriented crossover POX that is initiated for $\mathcal{Q}$ iterations when a new best solution is found. This POX supports exploring promising regions by further investigating the area of the best solution using problem-specific features, i.e., tool requirements and job prioritization. Thereby, it can help the GA converge to good solutions better. This POX consists of three sequential steps that aim to reduce the number of unfinished priority operations O_U, reduce tool switches, and increase the number of finished operations O_F.

The first step of the POX orders the maximal classes in the solution based on the number of priority operations in each maximal class. The maximal classes with the highest number of priority operations, i.e., $\max\{\{(j,k) \in S\} \cap \{(j,k) \in O_P\}\}$, are brought forward in the sequence. Then, the other maximal classes are arranged in decreasing order of the number of priority operations in their classes. This drives the GA towards solutions having more finished priority operations within the finite scheduling horizon. Consequently, it reduces the penalty cost $C_{\mathcal{P}}$ incurred for unfinished priority operations.

The second step of the POX aims to employ common tools in the required tool sets for operations, i.e., the overlap in tool sets, to reduce variable costs of tool switches and machine idle time caused by tooling constraints. All operations in a solution are checked for operations with similar tool requirements, starting with operations in v_1. Operations requiring similar tool sets are inserted at the position of the first gene that has those tool sets. By changing the operations' positions and placing them together with operations with similar tool requirements, the GA is pushed towards minimizing the tool switching cost $C_{\mathcal{T}}$.

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_4	S_7	S_2	S_3	S_5	S_6
Operations O	4 4	1 1	7	2	3 3	5	6
Machine vector $V^{\mathcal{M}}$	1	1	2	2	2	2	2
Tool sets T_{ij}	2	1	1	2	3	4	5

(a) Chromosome C_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6
Job vector $V^{\mathcal{S}}$	S_1	S_3	S_5	S_2	S_4	S_6
Operations O	1 1	4 4	5 7	2	3 3	6
Machine vector $V^{\mathcal{M}}$	1	2	2	1	1	1
Tool sets T_{ij}	1	2	4, 1	2	3	5

(b) Chromosome C_{κ_2}

Fig. 6. Offspring chromosomes after the first step (job prioritization) of POX (□ Classes with priority operations).

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector V^{S}	S_1	S_2	S_4	S_7	S_3	S_5	S_6
Operations O	4 4	2	1 1	7	3 3	5	6
Machine vector V^{M}	1	1	2	2	2	2	1
Tool sets T_{ij}	2	2	1	1	3	4	5

(a) Chromosome C_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6
Job vector V^{S}	S_1	S_5	S_3	S_2	S_4	S_6
Operations O	1 1	5 7	4 4	2	3 3	6
Machine vector V^{M}	1	1	2	2	1	2
Tool sets T_{ij}	1	4, 1	2	2	3	5

(b) Chromosome C_{κ_2}

Fig. 7. Offspring after the second step (tool similarities) and the third step (constructive heuristic) of POX (□ Tool similarities, □ Constructive heuristic).

The third step adopts a constructive heuristic proposed by Dang et al. (2021) (see Algorithm 2 in their work). This heuristic is used to create machine vector $V^{\mathcal{M}}$ of each solution based on the resulting job vector $V^{\mathcal{S}}$. It aims to allocate maximal classes to machines that already hold a subset of the required tools for processing these maximal classes. If no machine holds any of the required tools, the machine allocation is done based on the least sum of processing times of all operations already allocated to each machine. This workload balancing drives the heuristic towards solutions with more finished operations and fewer tool switches.

Example 6. The three steps of the POX are demonstrated by Figs. 6 and 7 in which the produced chromosomes result from those in Fig. 5 in Example 5. Fig. 6 illustrates the first step, where we, e.g., move maximal classes S_4 and S_7 forward in the C_{κ_1} sequence since they contain priority operations. Then, Fig. 7 illustrates the second and third steps. In Fig. 7(a), maximal class S_2 is inserted after class S_1 since they have similar tool requirements. Finally, the POX builds up the machine vector by, e.g., allocating maximal classes S_2 in C_{κ_1} to machine 1 since this machine already holds the required tools. Also, as no machine holds any required tools for S_4, machine 2 is allocated to this class based on the least sum of processing times (i.e., 0 of machine 2 as opposed to 20 of machine 1).

5.4. Mutation

In GAs, mutation is a genetic operator that makes spontaneous random changes to various chromosomes in offspring. It serves the crucial role of exploring the search space by diversifying the population and trying out useful genes at random positions (Gen et al., 2008b). Our GA adapts a swap mutation[3] for the job vector and makes use of a uniform mutation for the machine vector as follows.

The swap mutation is performed on the job vector of a chromosome by selecting each gene v_g with a probability P_s and swapping the maximal class S_g of that gene with the maximal class of another randomly selected gene. In this way, the GA can try out different orders of maximal classes within the chromosome. Also, the swap is performed on the operational level by selecting each operation, also with a probability P_s, and swapping this operation with another operation randomly selected in the chromosome. Before exchanging an operation to the maximal class of its new position, it is verified if the sum of all required tools for the maximal class, including the exchanged operation, exceeds the tool magazine capacity. If the tool magazine capacity is exceeded, the operation is added to the first following maximal class that is feasible, or a new maximal class is created for the operation and added at the end of the chromosome. In the latter case, an element M_g with a randomly generated machine is added to the machine vector to ensure that both job and machine vectors have the same length. This second way of swapping can result in different compositions of maximal classes, which enables a greater variety of possible solutions.

Furthermore, the uniform mutation is performed on the machine vector to attempt different machine allocations for the maximal classes. The uniform mutation selects each element M_g of the machine vector with a probability P_u and alters it by a randomly generated machine.

Example 7. The swap and uniform mutations are illustrated on the chromosomes in Fig. 7. First, the swap mutation is performed on the gene and operation levels of these solutions, resulting in Fig. 8. In C'_{κ_1}, maximal class S_2 is swapped with class S_5, and operation $(7, 1)$ is swapped with operation $(6, 1)$. In C'_{κ_2}, no maximal class is swapped, while operation $(5, 1)$ is swapped with operation $(2, 1)$. However, $(2, 1)$ cannot be exchanged to maximal class S_5 due to the tool magazine capacity, thus added to S_3. Second, the uniform mutation is performed on the machine vector, e.g., a machine is randomly generated for M_1 in C'_{κ_1}, as well as M_3 and M_6 in C'_{κ_2}.

5.5. Maximal classes

The crossover and mutation operations may generate solutions with some classes that are no longer maximal classes. It results in unnecessary additional tool switching instances. Hence, the classes allocated to the same machine are considered to be merged to a maximal class again in order to prevent the GA from generating redundant genes, pushing it towards better solutions. The merging can take place if the operations within these classes require tools that together fit in the tool magazine capacity. Here, subsequent classes in the job vector are merged at the position of the first relevant class. Fig. 9 shows an example of recreating maximal classes for chromosomes in Fig. 8. In this example, genes v_1 and v_2 in Fig. 8(b) are merged into gene v_1 (colored gray) in Fig. 9(b).

5.6. Elitism selection and immigration

Elitism selection is applied to retain the best solutions of each generation for the next generation, thereby increasing the GA's ability to learn from the history of the search (Gen et al., 2008b). First,

[3] Pretest of different mutation operators can be seen at https://github.com/vinhise/pmstsup/.

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_5	S_4	S_7	S_3	S_2	S_6
Operations O	4 4	5	1 1	6	3 3	2	7
Machine vector $V^{\mathcal{M}}$	2	1	2	2	2	2	1
Tool sets T_{ij}	2	4	1	5	3	2	1

(a) Chromosome C'_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6
Job vector $V^{\mathcal{S}}$	S_1	S_5	S_3	S_2	S_4	S_6
Operations O	1 1	7	4 4 2	5	3 3	6
Machine vector $V^{\mathcal{M}}$	1	1	1	2	1	2
Tool sets T_{ij}	1	1	2	4	3	5

(b) Chromosome C'_{κ_2}

Fig. 8. Offspring chromosomes after swap mutation and uniform mutation (□ Swapped classes, □ Swapped operations, □ Mutated machines).

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Job vector $V^{\mathcal{S}}$	S_1	S_5	S_4	S_7	S_3	S_2	S_6
Operations O	4 4	5	1 1	6	3 3	2	7
Machine vector $V^{\mathcal{M}}$	2	1	2	2	2	2	1
Tool sets T_{ij}	2	4	1	5	3	2	1

(a) Chromosome C'_{κ_1}

Chromosome v (gene v_g)	v_1	v_2	v_3	v_4	v_5
Job vector $V^{\mathcal{S}}$	S_1	S_3	S_2	S_4	S_6
Operations O	1 1 7	4 4 2	5	3 3	6
Machine vector $V^{\mathcal{M}}$	1	1	2	1	2
Tool sets T_{ij}	1	2	4	3	5

(b) Chromosome C'_{κ_2}

Fig. 9. Offspring chromosomes after recreating maximal classes (□ Positions of genes merged to maximal classes).

all parents are ordered based on their fitness values. Subsequently, an elitism rate γ_2 ($0 < \gamma_2 \leq 1$) determines the number of best parent chromosomes, i.e., $S_E = \gamma_2 \times N_p$, that is retained for the next generation. These S_E best parents replace S_E offspring chromosomes with the worst (lowest) fitness values and then form the population of the next generation together with the remaining offspring. Also, we remove duplicate chromosomes in the next generation's population and replace them with solutions randomly generated as in the initialization procedure in Section 5.2. This immigration and the elitism selection maintain a balance between exploiting good solutions and exploring the search space by population diversification (Gen et al., 2008b).

5.7. Tool switching method

As processing operations on machines, tool switches are inevitable due to the limited tool magazine capacity. According to Tang and Denardo (1988b), the tool switching problem with a known job sequence can be solved optimally in polynomial time by the KTNS policy. Therefore, in this section, we apply the KTNS method to determine the tool switching plan (i.e., which tools should be removed from a tool magazine to insert required tools) for an operation sequence generated using the genetic operators (i.e., initialization or crossover and mutation). This method minimizes the number of tool switches by counting subsequent operations for which present tools are used. Then, if missing tools must be inserted, the present tools that are needed the soonest are removed last. Naturally, the KTNS is employed if there is insufficient remaining capacity to insert required tools into the magazine. We also note that the KTNS is embedded in the fitness evaluation in Section 5.8 for calculating variable costs of tool switching.

In our paper, the KTNS is applied for tool switches between maximal classes on each machine. This method consists of three following steps:

Step 1. Let T_m denote the set of tools present on machine m. Also, let TS_m denote a subset of tools in T_m that will be used later for subsequent maximal classes on machine m, i.e., $TS_m \subseteq T_m$. Then, the method checks and determines TS_m for each machine m.

Step 2. Each tool $t \in TS_m$ is scored based on the sequence of subsequent maximal classes for which the tool is needed. Let Q_m denote this sequence on machine m. Note that each tool may be required by more than one class in Q_m. Therefore, we consider a subsequence $QS_m \subseteq Q_m$ that contains only the first class requiring each tool $t \in TS_m$ in Q_m. Afterward, the score is calculated as follows: $sc_t = |TS_m| - (u-1)$, with u being the position of the class in QS_m that requires tool t. Any tool not required for any subsequent maximal classes gets a score of 0, i.e., $sc_t = 0$, $\forall t \in T_m \backslash TS_m$.

Step 3. The method decides which tools are removed from T_m by first defining the sufficient number of tool slots Δ_m that should be freed up to insert missing tools for operations in maximal class S_g, where $\Delta_m = |TR_m| - (T_C - |T_m|)$, with $TR_m = \{t : t \in \cup_{(j,k)\in S_g} T_{jk} \wedge t \notin T_m\}$. Then, Δ_m number of tools are removed from the tool magazine in ascending order of their scores sc_t to free up capacity in machine m for inserting the missing tools. The method repeats until it is completed for the set of finished operations within scheduling horizon H.

Example 8. We consider chromosome C_{κ_1} from Fig. 9(b), where the sequence of the maximal classes on machine 1 is checked, i.e., $S_1 \succ S_3 \succ S_4$. In Step 1, machine 1 with $T_C = 8$ can hold tool set 1 that is required for processing operations $(1,1)$, $(1,2)$, and $(7,1)$ in maximal class S_1, thus $T_1 = \{t_1, t_2, t_3, t_4, t_5\}$. Among these tools, t_4 and t_5 are still needed for processing subsequent maximal class S_4 containing operations $(3,1)$ and $(3,2)$, so $TS_1 = \{t_4, t_5\}$. Next, in Step 2, scores are determined for tools t_4 and t_5 in TS_1 based on the order they are needed in the sequence of subsequent maximal classes $Q_1 = \{S_3, S_4\}$. Here, tools t_4 and t_5 are first used by maximal class S_4 in Q_1, thus $QS_1 = \{S_4\}$. Then, since S_4 is the first maximal class in QS_1 requiring the tools in TS_1, we have $sc_4 = sc_5 = 2-(1-1) = 2$. Tools t_1, t_2, and t_3 are not required for any subsequent maximal classes, so $sc_1 = sc_2 = sc_3 = 0$. Finally, in Step 3, we determine $TR_1 = \{t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}\}$, thus $|TR_1| = 7$. It results in $\Delta_1 = 7-(8-5) = 4$, which means four tools need to be removed. In this case, we remove tools t_1, t_2, t_3, and t_4 (or t_5) from T_1 according to the ascending order of the tools' scores. The tools in the magazine are now updated to $T_1 = \{t_5, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}, t_{17}, t_{18}\}$.

5.8. Fitness evaluation

Fitness evaluation is performed to compute the fitness value of each chromosome, i.e., the objective value of each solution. We employ chromosomes' fitness values for their ranks in the tournament selection (Section 5.3) and elitism selection (Section 5.6). With the presented genetic representation, each chromosome contains all operations in O. However, only the operations finished within the finite scheduling horizon H should be evaluated. Therefore, we let S_{F_m}, O_{F_m}, and O_{P_m} denote the sets of finished maximal classes, finished operations, and finished priority operations on machine m, respectively. Note that a maximal class is considered "finished" if any operation in this class is completed. Consequently, we let S'_g denote a maximal class containing

finished operations. The pseudocode of the fitness evaluation procedure is shown in Algorithm 2.

Algorithm 2 Fitness evaluation

Require: chromosome v, set of priority operations O_P, scheduling horizon H, unsupervised hours t_U

1: $S_{F_m} \leftarrow \emptyset$, $O_{F_m} \leftarrow \emptyset$, $O_{P_m} \leftarrow \emptyset$, $a_m \leftarrow 0$ $(\forall m \in M)$
2: **for** $g = 1$ to n **do**
3: $\quad m \leftarrow M_g$
4: $\quad \hat{S}_g \leftarrow \text{sortPriority}(S_g)$, $S'_g \leftarrow \emptyset$
5: $\quad$ **if** $a_m \le H$ **then**
6: $\quad\quad$ **if** $a_m \pmod{24} \ge 24 - t_U$ **then**
7: $\quad\quad\quad a_m \leftarrow a_m + (24 - a_m \pmod{24})$
8: $\quad\quad$ **end if**
9: $\quad\quad$ **for** $(j,k) \in \hat{S}_g$ **do**
10: $\quad\quad\quad \rho_m \leftarrow a_m + p_{jk}$
11: $\quad\quad\quad$ **if** $\rho_m \le H$ **then**
12: $\quad\quad\quad\quad a_m \leftarrow \rho_m$
13: $\quad\quad\quad\quad S'_g \leftarrow S'_g \cup (j,k)$
14: $\quad\quad\quad\quad O_{F_m} \leftarrow O_{F_m} \cup (j,k)$
15: $\quad\quad\quad\quad O_{P_m} \leftarrow O_{P_m} \cup \{(j,k) : (j,k) \in O_P\}$
16: $\quad\quad\quad$ **end if**
17: $\quad\quad$ **end for**
18: $\quad\quad S_{F_m} \leftarrow S_{F_m} \cup S'_g$
19: $\quad$ **end if**
20: **end for**
21: **for** $m \in M$ **do**
22: $\quad$ **for** $S'_g \in S_{F_m}$ **do**
23: $\quad\quad$ **if** $|TR_m| > T_C - |T_m|$ **then** $\quad \triangleright\ TR_m = \{t : t \in \cup_{(j,k)\in S'_g} T_{jk} \wedge t \notin T_m\}$
24: $\quad\quad\quad T_m \leftarrow T_m \setminus \{t : \text{KTNS}(t \in T_m)\}$ $\quad \triangleright$ See Section 5.7
25: $\quad\quad$ **end if**
26: $\quad\quad T_m \leftarrow T_m \cup TR_m$
27: $\quad\quad T_{\mathcal{V}} \leftarrow T_{\mathcal{V}} + |TR_m|$
28: $\quad$ **end for**
29: **end for**
30: $T_{\mathcal{F}} \leftarrow \sum_{m\in M} \max(|S_{F_m}| - 1, 0)$
31: $|O_U| \leftarrow |O_P| - \sum_{m\in M} |O_{P_m}|$
32: $f_v \leftarrow r \cdot \sum_{m\in M} |O_{F_m}| - c_p \cdot |O_U| - (c_f \cdot T_{\mathcal{F}} + c_v \cdot T_{\mathcal{V}})$

The procedure starts with initializing sets S_{F_m}, O_{F_m}, and O_{P_m}, as well as the available time of each machine a_m (line 1). Then, chromosome v is decoded from gene v_1 to v_n to update these sets (lines 2–20). The operations in each maximal class S_g are sorted based on their priorities (line 4). This way, when only part of a maximal class can be completed within scheduling horizon H, we may finish priority operations in this class first to reduce the total penalty cost $C_{\mathcal{P}}$. While iterating through maximal classes S, the starting time of operation (j,k) requiring tool switches during an unsupervised shift is postponed until the start of the next supervised shift (lines 6–8), where the starting time of the operation is the available time a_m of machine m to which it is allocated. Next, the procedure calculates the machine's ending time ρ_m involving operation (j,k) (line 10). If this ending time is still within H, it will update the machine's available time a_m, class S'_g, and sets O_{F_m} and O_{P_m} (lines 12–15). The set of finished maximal classes S_{F_m} is also updated with class S'_g (line 18).

Further, the resulting set S_{F_m} is used for computing tool switches between finished maximal classes on each machine m (lines 21–29). Suppose the number of missing tools TR_m is larger than the remaining magazine capacity. In that case, the procedure employs the KTNS, presented in Section 5.7, to determine which tools should be removed from the magazine to insert the missing tools (lines 23–25). Then, the set of present tools on each machine T_m and the number of tool switches $T_{\mathcal{V}}$ are updated (lines 26–27). The number of tool switching instances $T_{\mathcal{F}}$ is derived from the number of sets of finished maximal classes S_{F_m} which also contain finished operations O_{F_m} (line 30). Here minus one is owing to the assumption that the tools required by the first maximal class processed on each machine are ready at the start of H. In addition, the number of unfinished priority operations O_U is derived from the sets of finished priority operations O_{P_m} obtained earlier (line 31). Finally, the fitness value of chromosome v is calculated from the revenue of finished operations O_{F_m}, the penalty cost of unfinished priority operations O_U, and the total tool switching cost involving $T_{\mathcal{F}}$ and $T_{\mathcal{V}}$ (line 32).

6. Practitioner heuristic

This section describes the manufacturer's current way of scheduling, henceforth referred to as the practitioner heuristic (PH). The PH allocates and schedules operations to machines based on logical decision-making. The PH consists of two phases. The first phase allocates operations into groups based on production characteristics, of which similar methods can be found in grouping technology literature. Subsequently, these newly created groups are allocated to machines (Section 6.1). The second phase creates a production schedule based on this group and machine allocation (Section 6.2). The PH serves as a benchmark for performance comparison with the proposed GA (see Section 7).

6.1. Phase 1: Allocate operations to groups/machines

In the first phase, all operations are allocated to machines. This allocation procedure is described in Algorithm 3. In practice, parts (jobs) with similar tooling requirements can be considered part families. Minor setups within the same family can be included in the part processing times. Nevertheless, major setups between part families are explicit and should be minimized to obtain a good schedule. First, the PH groups operations (j,k) using identical tool set $i \in \Theta$, (i.e., $i = T_{jk}$ holds) into technological family F_i (lines 3–10), where Θ denotes the set of all tool sets. Next, we calculate the total processing time of operations that are grouped into family F_i, denoted by Φ_i, see line 7. Subsequently, we allocate these technological families to machines M using the shortest processing time (SPT) policy. Specifically, the PH selects a machine m' having the shortest total processing time and allocates a family F_i with the smallest Φ_i to this machine (lines 11–16), where A_m and Ψ_m denote the set of families and their total processing time allocated to machine m, respectively.

Further, we balance the workloads among the machines to prevent them from "starving" (lines 19–26). This is done by iterating over the machines and reallocating product families until (1) the differences in workload of all machines is less than benchmark B_1 and (2) the workload per machine is equal or greater than Benchmark B_2. In each iteration, the product family with the smallest Φ_i from the machine m'' with the highest workload w'', is transferred to machine m' with the lowest workload w'. Ultimately, this reduces the difference in the sum of processing times among machines. The benchmark levels are selected based on practitioners' experience and characteristics of the high-mix, low-volume, high-complexity manufacturing environment. Here, the maximum workload difference B_1 is set to 10% of all available hours in horizon H (e.g., a horizon H of 7 days results in a BenchMark B_1 of $24 \times 7 \times 10\% = 16.8$ h). On the other hand, the minimal workload B_2 on every machine is set to 80% of all available hours for manufacturing within the time horizon H.

Example 9. We illustrate the first phase of the PH using the data from Example 1 (see Tables 1 and 2) and Example 2 ($H = 48$ h) in Section 3. First, we group all operations into technological families with identical tool sets, i.e., $F_1 = \{(1,1),(1,2),(7,1)\}$, $F_2 = \{(2,1),(4,1),(4,2)\}$, $F_3 = \{(3,1),(3,2)\}$, $F_4 = \{(5,1)\}$, $F_5 = \{(6,1)\}$. Second, we allocate these families to the set of machines using the SPT policy. The new ordered set of families based on their sum of processing times is $\{F_4, F_5, F_1, F_3, F_2\}$. We obtain the following allocation after assigning these ordered sets to machines: $A_1 = \{F_4, F_1, F_2\}$ and $A_2 = \{F_5, F_3\}$. Third, we check whether one of the benchmark levels can be achieved for the workload balancing. It can be calculated that the difference in

Algorithm 3 Phase 1: Allocate operations to machines

Require: set of operations O, set of machines M, set of tool sets Θ
1: $F_i \leftarrow \emptyset$, $\Phi_i \leftarrow 0$ $(\forall i \in \Theta)$
2: $A_m \leftarrow \emptyset$, $\Psi_m \leftarrow 0$ $(\forall m \in M)$
3: **for** $i \in \Theta$ **do**
4: **for** $(j,k) \in O$ **do**
5: **if** $i = T_{jk}$ **then**
6: $F_i \leftarrow F_i \cup (j,k)$
7: $\Phi_i \leftarrow \Phi_i + p_{jk}$
8: **end if**
9: **end for**
10: **end for**
11: $\hat{\Theta} \leftarrow \{i \in \Theta : \text{sort}(\Phi_i)\}$ ▷ sort based on SPT rule
12: **for** $i \in \hat{\Theta}$ **do**
13: $m' \leftarrow \text{argmin}\{\Psi_m : \forall m \in M\}$
14: $A_{m'} \leftarrow A_{m'} \cup F_i$
15: $\Psi_{m'} \leftarrow \Psi_{m'} + \Phi_i$
16: **end for**
17: $m'' \leftarrow \text{argmax}\{\Psi_m : \forall m \in M\}$, $w'' \leftarrow \max\{\Psi_m : \forall m \in M\}$
18: $m' \leftarrow \text{argmin}\{\Psi_m : \forall m \in M\}$, $w' \leftarrow \min\{\Psi_m : \forall m \in M\}$
19: **while** $w'' - w' > B_1 \wedge w' < B_2$ **do**
20: $s \leftarrow \text{argmin}\{\Phi_i : \forall F_i \in A_{m''}\}$
21: $A_{m''} \leftarrow A_{m''} \setminus F_s$
22: $\Psi_{m''} \leftarrow \Psi_{m''} - \Phi_s$
23: $A_{m'} \leftarrow A_{m'} \cup F_s$
24: $\Psi_{m'} \leftarrow \Psi_{m'} + \Phi_s$
25: **do** lines 17–18
26: **end while**

workload between the two machines is higher than B_1 (i.e., $39 - 24 = 15 > 10\% \times 48$), and the minimal workload on machine 2 is lower than B_2 (i.e., $24 < 80\% \times 48$). Consequently, the workload balancing is performed by reallocating F_4 (with $\Phi_4 = 6$) from machine 1 to machine 2. It results in the following allocation: $A_1 = \{F_1, F_2\}$ and $A_2 = \{F_4, F_5, F_3\}$. This allocation satisfies benchmark B_1 (i.e., $33 - 30 = 3 < 10\% \times 48$) and thus is the result of Phase 1.

6.2. Phase 2: Create schedule

In the second phase, the PH creates a production schedule based on the machine allocation's results from phase 1. First, the PH sequences all the priority operations allocated to each machine. We insert a priority operation right after the position of the last operation in the sequence having similar tool requirements, if possible. Otherwise, this operation is added at the end of the sequence. Second, the PH considers the non-priority operations allocated to each machine. It inserts a non-priority operation into the sequence right after the position of the last operation with similar tool requirements, if possible. If there are no operations in the sequence with similar tool requirements, this non-priority operation is added at the end of the sequence. Consequently, all operations from a technological family are sequenced subsequently to maintain the benefit of similar tool requirements, which reduces tool switches and machine idle time. Finally, the resulting sequence is transformed into a schedule using a similar procedure as Algorithm 2, taking into account unsupervised shifts.

Example 10. From Example 9, we consider $A_1 = \{F_1, F_2\}$ and $A_2 = \{F_4, F_5, F_3\}$. First, we sequence the operations that are allocated to machine 1. We start by inserting priority operations at the position of operations with similar tool requirements, i.e., $\{(1,1),(1,2),(7,1),(4,1),(4,2)\}$. Afterward, we insert the regular operations in a similar manner, i.e.,
$\{(1,1),(1,2),(7,1),(4,1),(4,2),(2,1)\}$. We process the same steps for machine 2 (where A_2 has no priority operations), resulting in the following sequence: $\{(5,1),(6,1),(3,1),(3,2)\}$. Last, the operation sequences on machines 1 and 2 are transformed into the schedule as presented in Fig. 10. One may notice that this schedule is superior to the schedule from Fig. 2 since it finishes an additional priority operation $(7,1)$ instead of regular operation $(2,1)$. This reduces the penalty cost $C_{\mathcal{P}}$ and requires less tool switching cost $C_{\mathcal{T}}$.

7. Computational experiments

This section presents the experimentation process for evaluating the performance of the MILP, PH, and proposed GA. First, we introduce three industry case studies in Section 7.1, followed by tuning the proposed heuristic's parameters in Section 7.2. Finally, we study the performance of the three methods. This consists of the comparison of the three methods and sensitivity analysis to demonstrate the proposed heuristic's effectiveness and derive managerial insights in Section 7.3 and Section 7.4, respectively.

7.1. Base cases: industrial case studies

We introduce three industry base cases, shown in Table 3, to test the performance of the presented approaches. We denote the base cases $|M|$M$|O|$, where $|M|$ is the number of machines within the work center, and $|O|$ is the number of operations (e.g., 2M376 is the base case with two machines processing 376 operations). The number of reentrant operations, i.e., operations of which the start is constrained by a previous operation that belongs to the same job, is denoted by $|O_R|$. The reentrant ratio, denoted by ρ_R, varies per base case and can be calculated as follows: $\rho_R = |O_R|/|O|$. In addition, the number of reentries per job is limited to one, which indicates that a job consists of at most two operations. The priority ratio ρ_P of the base cases, where $\rho_P = |O_P|/|O|$, is around 0.5, implying that about half of all operations are prioritized. In addition, the tool ratio, denoted by ρ_T where $\rho_T = |T|/|O|$, represents the number of unique tools that are required for processing all operations. The tool ratio acts as an indicator of the tool heterogeneity where, generally speaking, a higher value for ρ_T implies more tool switches, reducing the profit and vice versa.

Moreover, the scheduling horizon H is set to 7 days based on the company's weekly planning policy. Each day within the scheduling horizon holds an unsupervised shift t_U of 12 h. The tool magazine capacity T_C of every machine is 80 tool slots. Also, the revenue and cost rates are set, in consultation with our industry partner, as follows: $r = \$30$ per operation, $c_p = \$30$ per operation, $c_f = \$10$ per instance, and $c_v = \$1$ per tool switch. Lastly, the operations' processing times (with $\mu = 91$ min, $\sigma = 51$ min) are derived from empirical data. The base cases originate from three different machine groups (i.e., with homogeneous machines per group) within the milling department of the manufacturer, and they represent the typical problem sizes that frequently happen in practice. Also, other case parameters (i.e., reentrant operations and priority operations) are generated from real-world scenarios. Due to confidentiality restrictions, all data is anonymized.[4]

All experiments in this section use the same stopping criteria for the proposed GA, i.e., $maxTime = 3600$ s and $G_c = 20$ (see Appendix A.3 for details of setting G_c). In practice, the manufacturer has approximately 6 h overnight to generate schedules. Nevertheless, this paper restricts the maximum computational time to one hour, because of limited computational resources and providing flexibility for rescheduling. In addition, all experiments are run on a computer with Intel® Core i5-7300U, @ 2.60 GHz CPU, and 8 GB RAM with Windows 10 operating system. The PH and GA are programmed in Python v3.6, while the MILP is written in the Gurobi Python package. Representative results of the experiments are discussed through case 6M1201 in Sections 7.3 and 7.4. In addition, to illustrate the generalizability and broader applicability of the proposed GA, Appendix D presents additional numerical experiments based on other problem instances obtained from the literature (Beezão et al., 2017).

[4] Anonymized data set is available on https://github.com/vinhise/pmstsup.

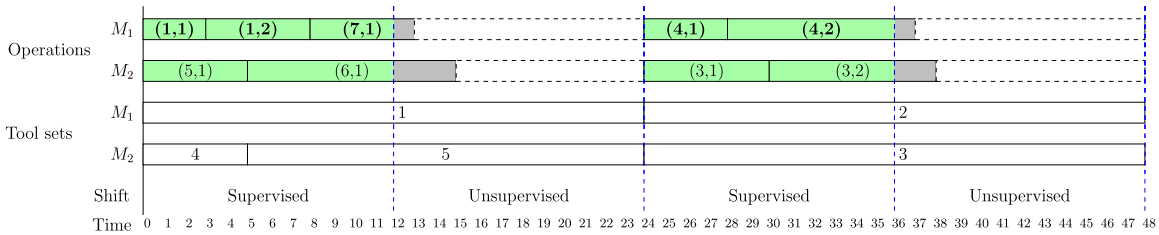

Fig. 10. Gantt chart of a schedule generated using the PH (Finished operations, Bold index (j, k): priority operations, Processing time during unsupervised shifts, Machine idle time).

Table 3
Information on cases studies.

Base cases	2-machine		6-machine			
	2M376		6M1201		6M1401	
	Ratio	No.	Ratio	No.	Ratio	No.
Jobs ($\|J\|$)	–	250	–	750	–	1000
Operations ($\|O\|$)	–	376	–	1201	–	1401
Reentrant operations (ρ_R, $\|O_R\|$)	0.50	188	0.60	720	0.40	560
Priority operations (ρ_P, $\|O_P\|$)	0.53	198	0.51	611	0.49	685
Tools (ρ_T, $\|T\|$)	1.73	650	1.27	1520	1.00	1398
Unsupervised hours/day (ρ_U, t_U)	0.50	12	0.50	12	0.50	12

Table 4
Factors and levels for parameter tuning.

Factor	Name	Low (–)	High (+)
Q	Number of iterations with POX	1	10
N_p	Population size	100	400
γ_1	Tournament selection rate	0.05	0.20
γ_2	Elitism selection rate	0.10	0.90
P_u	Probability of uniform mutation	0.01	0.20
P_s	Probability of swap mutation	0.01	0.20

7.2. Parameter tuning

The parameters of the GA have a significant impact on its effectiveness. Therefore, these parameters are tuned to improve the performance of the proposed GA. The parameter tuning is carried out using a factorial design of experiments (DOE) that shows the effect of parameters and assists in selecting appropriate parameters' values (Box et al., 2005). It also helps enhance the understanding of parameter interactions. In this section, the DoE considers six parameters at two levels, low (–) and high (+) values, as given in Table 4. In total, there are 64 (2^6) different combinations of parameter settings that are tested. Also, four (tuning) cases are used to tune the six parameters. Information on these cases is summarized in Appendix A.1 (see Table A.1). For each case and each combination of the parameter settings, the proposed GA is run 10 times. The detailed tuning steps are presented in Appendix A.2, and their results are summarized as follows: $Q = 1$, $N_P = 400$, $\gamma_1 = 0.2$, $\gamma_2 = 0.1$, $P_u = 0.01$, and $P_s = 0.01$. These parameter values will be used for the GA in the remaining of Section 7.

7.3. Comparison of MILP, PH, and GA

In this section, we investigate the performance of the MILP and GA. The results of the proposed GA are also evaluated with respect to the PH, the current way of scheduling at our industry partner KMWE. The three methods' performance is studied on case 6M1201, from which 13 problem instances are generated by subsequently increasing the number of operations from 5 to 1020. Their results are presented in Table 5 (see Tables B.1 and B.2 in Appendix B for the other base cases' results). To assure fair comparisons, the maximum computational time of 3600 s is set to all the three methods. Also, the mean and standard deviation of the objective value and computational time are calculated from those obtained in 10 runs. A percentage gap is derived from the mean objective values found by two respective methods, either the GA and MILP or the GA and PH.

Table 5 shows that the proposed GA obtains equal or better results than the MILP and PH for all instances in terms of the objective value within the time limit. All three methods can find the optimal solutions for the instances with up to 25 operations. Nevertheless, for larger instances, the MILP is not able to find any feasible solutions, whereas the PH and GA can find feasible solutions for all instance sizes. Also, there is an improvement, up to 27%, in the objective value gained by GA over PH when having more than 80 operations. The standard deviation of the objective value of the GA is significantly small compared to the mean. We can also observe a similar trend of results in the other base cases (see Appendix B). Generally, the proposed GA shows its outperformance relative to the other methods, especially the current way of scheduling employed at the company in a reasonable computational time.

7.4. Sensitivity analysis

The sensitivity analysis provides more insights and will help managers to understand the outcomes of the base case. In this section, we test the proposed methods in different scenarios which might occur due to future changing requirements. Specifically, the performance of the GA is compared to the PH to quantify the potential room for improvement in current practice. The MILP is not considered in this analysis as this method was not able to find solutions for industry-size instances. The following three scenarios for the sensitivity analysis are considered.

- Unsupervised ratio (ρ_U): this ratio indicates the percentage of production hours in which tool switches cannot occur. An increase in the unsupervised ratio indicates a move towards lights-out manufacturing. Varying the unsupervised ratio provides information on its impact on the performance of the GA and PH when a company aims for less labor-dependent production processes.
- Priority ratio (ρ_P): this ratio acts as a parameter that shows the number of jobs prioritized to be completed within the scheduling horizon. An increase in this ratio exerts more pressure on the production schedule. Therefore, this ratio is varied in the base cases to examine how both methods would react to different pressure levels, a common factor within the HMLV industry.
- Tool ratio (ρ_T): this ratio dictates the number of unique tools that are required for a set of operations. An increase in the tool ratio

Table 5
Performance comparison of MILP, PH, and GA for case 6M1201.

n	MILP				PH				GA				GAP in Obj. (%)	
	Obj.		C.T.		Obj.		C.T.		Obj.		C.T.		GA vs. MILP	GA vs. PH
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ		
5	[a]150.00	0.00	0.00	0.00	150.00	0.00	1.63	0.10	150.00	0.00	123.19	0.57	0.00	0.00
10	[a]300.00	0.00	0.94	0.00	300.00	0.00	1.39	0.07	300.00	0.00	16.87	0.12	0.00	0.00
15	[a]450.00	0.00	28.27	0.03	450.00	0.00	1.41	0.07	450.00	0.00	19.08	0.21	0.00	0.00
20	[a]600.00	0.00	210.32	0.24	600.00	0.00	1.47	0.02	600.00	0.00	22.23	0.15	0.00	0.00
25	[a]750.00	0.00	3600.00	0.00	750.00	0.00	1.40	0.05	750.00	0.00	25.84	0.17	0.00	0.00
30	–	–	–	–	900.00	0.00	1.42	0.06	900.00	0.00	33.26	0.38	–	0.00
50	–	–	–	–	1500.00	0.00	1.57	0.10	1500.00	0.00	57.10	0.43	–	0.00
80	–	–	–	–	2400.00	0.00	1.39	0.08	2400.00	0.00	120.12	3.89	–	0.00
130	–	–	–	–	3853.00	0.00	1.46	0.06	3900.00	0.00	276.32	4.74	–	1.22
210	–	–	–	–	6234.00	0.00	1.47	0.03	6300.00	0.00	412.16	4.12	–	1.06
340	–	–	–	–	9852.00	0.00	1.57	0.04	10127.40	3.10	809.96	12.40	–	2.80
630	–	–	–	–	15720.00	0.00	1.76	0.03	17772.00	76.06	1874.46	41.14	–	13.05
1201	–	–	–	–	14323.00	0.00	2.23	0.04	18099.90	221.74	3477.08	83.43	–	26.37

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Optimal value.

Table 6
Performance comparison of GA and PH when varying unsupervised ratio on 6M1201.

ρ_U	t_U	PH				GA				% Gap in Obj.
		Obj.		C.T.		Obj.		C.T.		
		μ	σ	μ	σ	μ	σ	μ	σ	
0.00	0	16410.00	0.00	2.29	0.12	17954.20	274.53	3600.07	0.08	9.41
0.25	6	15634.00	0.00	2.18	0.16	18202.90	170.50	3600.19	0.35	16.43
[a]0.50	12	14323.00	0.00	2.23	0.04	18099.90	221.74	3477.08	83.43	26.37
0.75	18	13555.00	0.00	2.28	0.06	17501.70	348.61	2752.79	64.38	29.12
1.00	24	12164.00	0.00	2.30	0.09	16403.10	321.12	2481.28	40.60	34.85

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Base case.

indicates a shift towards the HMLV manufacturing environment. It means a similar number of jobs requires more different tools, complicating scheduling further. Hence, the tool ratio is varied to analyze the potential effect it exercises on the performance of the GA relative to the PH.

The results of these scenarios are demonstrated for case 6M1401 in Sections 7.4.1–7.4.3. The results of the other base cases are presented in Appendix C.

7.4.1. Unsupervised ratio

The impact of varying unsupervised ratios is analyzed with five levels ranging from 0 to 1 (with steps of 0.25), i.e., from none to 24 unsupervised production hours per day. Consequently, Table 6 presents five different instances generated from case 6M1201. For each problem instance, each method is run 10 times to calculate the mean and standard deviation of the objective value and computational time. The results of case 6M1201 when varying ρ_U are shown in Table 6 and Fig. 11 (see Table C.1 in Appendix C for this scenario's results of the other base cases).

Fig. 11 shows a decreasing tendency in the objective value for both GA and PH when the unsupervised ratio increases. This is expected since an increase in this ratio imposes a constraint on the moments a tool switch instance can occur. When a tool switch instance is required during an unsupervised shift, it is delayed until the next supervised shift starts. This induces idle time, which ultimately harms the profit. Moreover, the higher objective values of the GA and an increase in the percentage gap illustrate that the GA can better cope with higher unsupervised ratios relative to the PH, with an average improvement of 23.2%. Note that even without unsupervised time ($t_U = 0$), the GA is still superior to the PH by about 10%. These arise from the capability of the genetic operators to change the order of maximal classes and swap operations between maximal classes. It helps to process more operations during unsupervised shifts. These results prove the potential of the proposed GA that assists practitioners in creating better production schedules with a higher profit than their current approach when moving towards lights-out manufacturing. In addition, practitioners can further interpret the magnitude of the decrease in profit to determine their roadmap towards longer unsupervised shifts.

7.4.2. Priority ratio

This scenario assesses the performance of the GA relative to the PH as changing priority ratio ρ_P. We vary the priority ratio from 0.25 to 0.75 with steps of 0.125, thus including the instance of $\rho_P = 0.5$ as in the base case. Each method also runs 10 times for each instance in this experiment. The results of five instances of case 6M1201 are presented in Table 7 and visualized in Fig. 12 (see Table C.2 in Appendix C for the other base cases' results in this scenario).

An increase in the priority ratio has a negative impact on the objective value since each unfinished priority operation incurs a penalty cost, so more priority operations increase the risk of a higher total penalty cost. The percentage gap between the two methods tends to rise, implying that the GA can better mitigate the risk of penalty costs. We observe that these gaps result mainly from our POX operator that sequences maximal classes in each solution in order of most priority to least priority operations. Consequently, it brings the maximal classes with many priority operations forward in the solution, thus increasing the probability of completing these operations within the scheduling horizon. In contrast, the PH first sequences priority operations and then inserts regular operations requiring similar tool sets. This makes the PH less capable of managing high priority ratios combined with a high workload since it finishes significantly fewer priority operations than the GA. In general, the results provide evidence of significant improvement gained by our GA as facing higher priority ratios. This ability of the GA to especially excel in (time) pressure increases its relevance for systems with tight deadlines or "hot" jobs, which are common in the high-tech manufacturing industry.

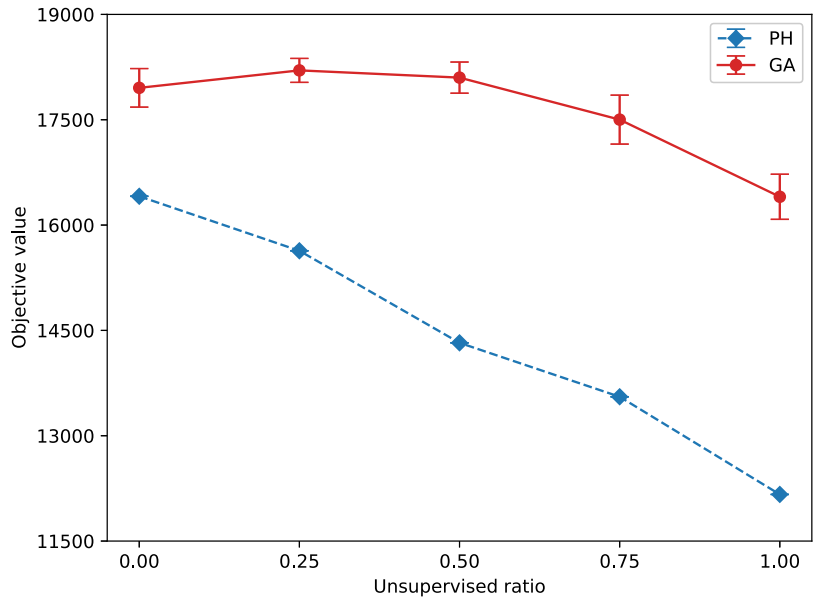

Fig. 11. Objective value when varying unsupervised ratio.

Table 7
Performance comparison of GA and PH when varying priority ratio on 6M1201.

ρ_P	$\|O_P\|$	PH				GA				% Gap in Obj.
		Obj.		C.T.		Obj.		C.T.		
		μ	σ	μ	σ	μ	σ	μ	σ	
0.25	300	17360.00	0.00	7.03	0.14	19949.90	483.35	3600.90	1.13	14.92
0.38	450	17106.00	0.00	5.62	0.06	19424.20	230.39	3601.04	1.32	13.55
[a]0.51	12	14323.00	0.00	2.23	0.04	18099.90	221.74	3477.08	83.43	26.37
0.63	751	9408.00	0.00	5.84	0.06	16005.10	302.72	3554.73	80.79	70.12
0.75	897	5992.00	0.00	5.57	0.06	12626.20	544.26	3601.47	1.65	110.72

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Base case.

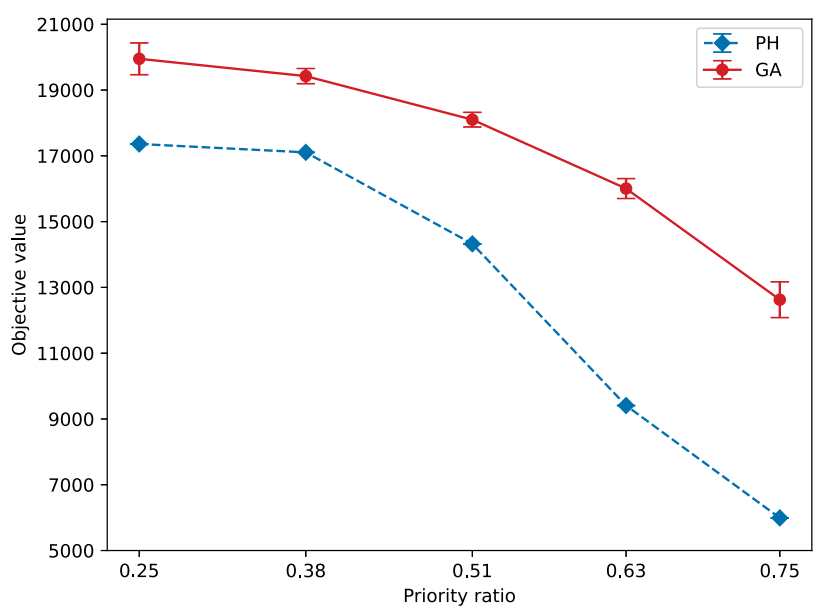

Fig. 12. Objective value when varying priority ratio.

7.4.3. Tool ratio

This scenario focuses on varying the tool ratio ρ_T, thus evaluating the effect of changing the tool diversity. For this analysis, the tool ratio ρ_T is varied from 1.0 to 2.0, with steps of 0.25, resulting in five instances. This range is selected because these values are, in consideration with practitioners at our manufacturer, reasonable possibilities forced by changing customer requirements in the HMLV manufacturing environment. Also, our manufacturer standardizes their tool

Table 8
Performance comparison of GA and PH when varying tool ratio on 6M1201.

ρ_T	$\|T\|$	PH				GA				% Gap in Obj.
		Obj.		C.T.		Obj.		C.T.		
		μ	σ	μ	σ	μ	σ	μ	σ	
1.00	1202	16528.00	0.00	7.46	0.13	19007.30	199.76	2806.36	45.48	15.00
[a]1.27	1520	14323.00	0.00	2.35	0.07	18099.90	221.74	3477.08	83.43	26.37
1.50	1801	14667.00	0.00	6.14	0.11	18027.00	135.46	3600.39	0.47	22.91
1.75	2101	14013.00	0.00	6.47	0.08	16924.70	265.96	3600.14	0.10	20.78
2.00	2401	13060.00	0.00	7.30	0.15	16105.50	255.55	3600.18	0.08	23.32

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Base case.

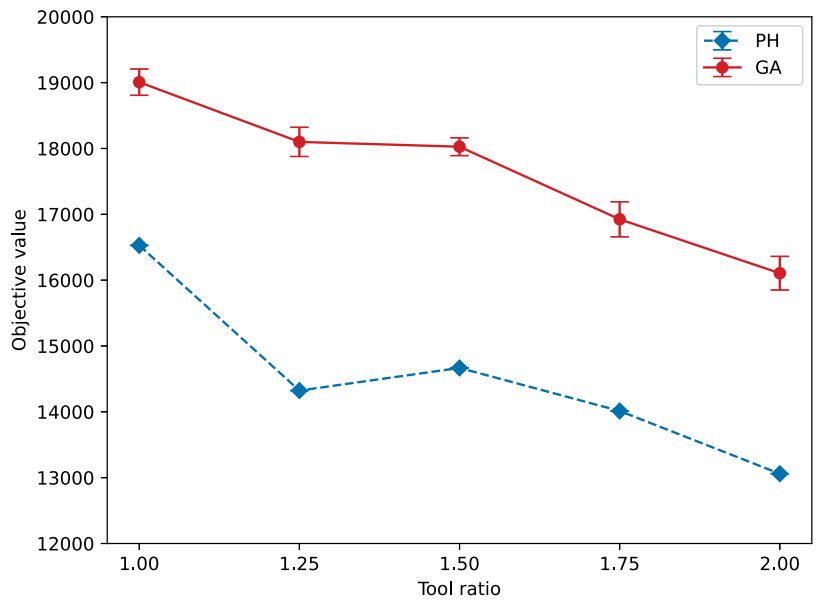

Fig. 13. Objective value when varying tool ratio.

requirement/usage per operation as much as possible to prevent a higher tool ratio. Hence, from a practical point of view, a higher tool ratio is not likely to be considered. The results of case 6M1201 are presented in Table 8 and visualized in Fig. 13 (see Table C.3 in Appendix C for the results of the other base cases in this scenario).

It can be seen that an increase in the tool ratio leads to lower objective values in both methods (due to the increase in tool switching costs) and vice versa. Here, the GA provides significantly better profit than the PH, with approximately 21%–22% on average. Also, the gap between them seems to fluctuate but tends upwards. The results show the outperformance of the GA regardless of changes in the tool ratio, which makes its use more appealing to practitioners as the company shifts toward HMLV manufacturing. In addition, these results recommend that practitioners should consider keeping their tool ratio as low as possible to reduce tool switching costs, hence increasing profit. This may be achieved by standardization of tools, thus creating general tools used for a variety of operations.

A similar trend of results can also be observed for each scenario of the other base cases (see Fig. C.1), as seen in the corresponding scenario of case 6M1201. In general, the proposed GA helps to gain, on average, 20%–60% more profit across all the base cases, compared to the PH within an acceptable computational time in practice. Although the PH takes much less time to generate solutions, it is still worth spending an hour of computational time for the proposed GA due to the significant profit improvement it can bring to the manufacturer. In particular, when there is more time pressure with "hot" jobs (due to high priority ratios), the manufacturer can even double the profit achieved using the proposed GA (see Tables 7 and C.2, the instances of $\rho_P = 0.75$). Conclusively, the use of the proposed GA is more profitable in case of longer unsupervised shifts, more pressure related to prioritizing jobs in the schedule, or more complex processing requirements concerning tool diversity.

8. Conclusions

In this paper, we study the problem of scheduling parallel identical machines with tool switches in the context of light-out manufacturing, inspired by a real-world manufacturer in the HMLV industry. In addition to the HMLV nature of the product portfolio, the complexity rises since our problem considers unsupervised production shifts within which no operators are available to switch tools in the machines. In addition, tool switches in our problem do not induce delay but costs. Also, we aim to maximize profit that comprises revenue minus the costs concerning tool switches and prioritized jobs unfinished within the scheduling horizon. To tackle this problem, we propose an MILP model and a GA. The computational experiments on the industry case studies show that while the MILP could only solve small-sized instances (i.e., up to 25 operations), the proposed GA could obtain good quality solutions for industry-sized instances within a realistic computational time. Additionally, a sensitivity analysis of various scenarios regarding different unsupervised, priority, and tool ratios indicates approximately 20%–60% improvement on average by using the proposed GA as opposed to the current practice. The results provide manufacturers in the HMLV industry with convincing evidence to investigate the applicability of the proposed GA, or other evolutionary algorithms, in their own practice.

Future research can consider several interesting directions. First, tools may be subjected to wearing and their maximum lifetime. Hence, a new approach, considering these aspects, should be developed to

allow practitioners to fully utilize manufacturing resources. Second, additional resources such as fixtures (i.e., work-holding or support devices) are worth considering because different jobs may require different fixture types, constraining job assignment and sequencing decisions. Finally, automating manufacturing for unsupervised shifts could be restricted by finite-capacity buffers storing unfinished products, which adds comprehensiveness and applicability to the HMLV production industry.

Nomenclature

Shop floor settings

M set of all machines
J set of all jobs
O set of all operations of all jobs
T set of all tools
H finite scheduling horizon
O_j set of operations of job j
n_j number of operations of job j
(j,k) kth operation of job j
J_p set of priority jobs ($J_p \subseteq J$)
O_p set of priority operations ($O_p \subseteq O$)
O_F set of (priority and regular) operations finished within H ($O_F \subseteq O$)
O_U set of priority operations not finished within H ($O_U \subseteq O_p$)
T_{jk} predefined tool set used for operation (j,k) ($T_{jk} \subseteq T$)
p_{jk} processing time of operation (j,k)
t_U duration of an unsupervised shift (hours)
T_C tool magazine capacity of every machine
ρ_R reentrant ratio
ρ_P priority ratio
ρ_T tool ratio

Revenue, costs, and profit

$\mathcal{R}$ total revenue
$C_{\mathcal{P}}$ total penalty cost
$C_{\mathcal{T}}$ total tool switching cost
C total cost ($C = C_{\mathcal{P}} + C_{\mathcal{T}}$)
c_p cost rate per unfinished priority operation
c_f fixed cost rate per tool switch instance
c_v variable cost rate per tool switch
r revenue rate per finished (priority or regular) operation

Genetic algorithm

κ index of generation
v index of chromosome
g index of gene in a chromosome
n number of genes in a chromosome
v_g gth gene of chromosome v
P_κ parent chromosomes of generation κ
C_κ offspring chromosomes, generated by crossover, of generation κ
C'_κ offspring chromosomes, generated by mutation, of generation κ
f_κ minimum fitness value of all chromosomes of generation κ
f_{best} current best fitness value
S maximal class
S_g maximal class of gene g
M_g machine processing maximal class of gene g
V^S job vector of a chromosome ($V^S = \{S_g : 1 \le g \le n\}$)
$V^{\mathcal{M}}$ machine vector of a chromosome ($V^{\mathcal{M}} = \{M_g : 1 \le g \le n\}$)
N_p population size
γ_1 tournament selection rate
S_T number of chromosomes chosen in tournament selection
γ_2 elitism selection rate
S_E number of chromosomes chosen in elitism selection
P_u probability of uniform mutation
P_s probability of swap mutation
G_c number of consecutive generations that no new best solution is found
S_{F_m} set of finished maximal classes on machine m
O_{F_m} set of finished operations on machine m
O_{P_m} set of finished priority operations on machine m
S'_g maximal class of gene g containing finished operations
a_m available time of machine m (to start an operation)
ρ_m ending time of machine m (after processing an operation)
$T_{\mathcal{V}}$ number of tool switches
$T_{\mathcal{F}}$ number of tool switching instances

Tool switching method

T_m set of tools present in machine m
TS_m subset of tools in T_m used later for subsequent maximal classes on machine m
Q_m sequence of subsequent maximal classes on machine m
QS_m subsequence of QS_m that contains only the first maximal class requiring each tool in TS_m
sc_t score of tool t required for a maximal class in QS_m
Δ_m sufficient number of tool slots that should be freed up to insert missing tools
TR_m set of tools required for insertion

Practitioner heuristic

F_i product family using tool set i
Φ_i total processing time of family F_i
Θ set of all tool sets
A_m set of families allocated to machine m
Φ_i total processing time of families allocated to machine m
B_1 maximum difference in workload of all machines
B_2 minimum workload of each machine

CRediT authorship contribution statement

Quang-Vinh Dang: Conceptualization, Methodology, Validation, Writing – original draft, Writing – review & editing. **Koen Herps:** Investigation, Software, Data curation, Writing – review & editing. **Tugce Martagan:** Conceptualization, Validation, Writing – review & editing. **Ivo Adan:** Conceptualization, Funding acquisition. **Jasper Heinrich:** Software, Visualization.

Data availability

I have shared the link to my data in the manuscript.

Acknowledgments

This research work has been funded by a grant provided by Provincie Noord-Brabant, the Netherlands for the project "Advanced Manufacturing Logistics".

Appendix A. Parameter tuning

A.1. Information of cases for parameter tuning

Table A.1
Information of parameter tuning cases.

Tuning cases	6M1249	6M1121	6M1333	6M751
Number of operations ($\|O\|$)	1249	1121	1333	751
Number of jobs ($\|J\|$)	960	700	740	500
Number of reentrant operations ($\|O_R\|$)	289	421	593	251
Number of priority operations ($\|O_P\|$)	620	540	688	367
% Reentrant operations (ρ_R)	0.23	0.38	0.44	0.33
% Priority operations (ρ_P)	0.50	0.48	0.52	0.49
Number of tools ($\|T\|$)	1820	1547	1275	1875
Tool ratio (ρ_T)	1.46	1.38	0.96	2.50

A.2. Tuning steps

Table A.2 presents each factor's effect on the performance of the proposed GA in the base case scenario 6M12. The effect is determined by $E_b^a = \bar{y}_{(+)} - \bar{y}_{(-)}$, where $\bar{y}_{(+)}$ and $\bar{y}_{(-)}$ are the average responses of objective value b for the set of parameters with factor a at the low (–) or high (+) level, respectively. Due to the maximization nature of the problem, a higher value of b is preferable to a lower one, indicating that it is more beneficial to set a factor to the high level (+) with a positive effect and vice versa.

In Table A.2, the number of iterations applying POX Q and elitism selection rate γ_2 produce distinctive effects in comparison to the other factors. For both Q and γ_2, setting their values to the low level increases the objective value. Thus, we set Q and γ_2 to 1 and 0.1, respectively. Afterward, the interactions between the remaining factors and the chosen levels of Q and γ_2 are investigated, as shown in Table A.3. Here, when Q and γ_2 are at the low level, swap mutation probability P_s has the most considerable effect, and setting this factor to the low level is preferable. Hence, the parameter settings at this stage are: Q, $\gamma_2 = 0.1$, and $P_s = 0.01$. Similarly, the interactions between the tuned parameters and the remaining ones are subsequently presented in Tables A.4 and A.5. In summary, the parameters' values are set as follows: $Q = 1$, $N_p = 400$, $\gamma_1 = 0.2$, $\gamma_2 = 0.1$, $P_u = 0.01$, and $P_s = 0.01$.

Table A.2
Factorial design analysis of parameters.

Factor	Q		N_p		γ_1		γ_2		P_u		P_s	
	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.
Low (–)	13617.34	1903.54	12665.02	1012.79	12651.27	2048.38	13415.63	1655.99	12892.12	2082.64	13154.40	1737.17
High (+)	12055.20	2151.20	13007.52	3041.95	13021.27	2006.36	12256.91	2398.75	12780.42	1972.10	12518.14	2317.57
Effect	−1562.13		342.49		369.99		−1158.72		−111.70		−636.26	

Obj.: objective value, C.T.: computational time (s).

Table A.3
Analysis of parameters with low-level β, γ_2.

Factor	N_p		γ_1		P_u		P_s	
	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.
Low (–)	13988.52	653.65	14180.61	1587.51	14676.88	1549.50	15060.06	1044.63
High (+)	14890.92	2464.22	14698.83	1530.36	14202.56	1568.38	13819.38	2073.24
Effect	902.41		518.22		−474.31		−1240.69	

Obj.: objective value, C.T.: computational time (s).

Table A.4
Analysis of parameters with low-level β, γ_2, P_s.

Factor	N_p		γ_1		P_u	
	Obj.	C.T.	Obj.	C.T.	Obj.	C.T.
Low (–)	14527.50	406.84	14807.56	1100.95	15400.94	1025.72
High (+)	15592.63	1479.94	15312.56	988.32	14719.19	1063.54
Effect	1065.13		505.00		−681.75	

Obj.: objective value, C.T.: computational time (s).

Table A.5
Analysis of parameters with low-level β, γ_2, P_s and high-level N_p.

Factor	γ_1		P_u	
	Obj.	C.T.	Obj.	C.T.
Low (–)	15459.13	1774.50	16030.56	1645.65
High (+)	15726.13	1583.37	15154.69	1712.23
Effect	267.00		−875.88	

Obj.: objective value, C.T.: computational time (s).

A.3. Tuning G_c (the number of non-improvement consecutive generations)

The stopping criterion G_c is set empirically to prevent unnecessarily long computational time in, e.g., small problem instances. As an example, we show the effect of G_c on tuning case 6M1121 with 630 operations in Table A.6. We vary G_c from 5 to 25 with a step of 5, which results in the five levels of G_c. The numbers in the table show the percentage of improvement (in the mean objective value) when increasing from one level to another. We observe that there is only a small improvement of about 0.05% (compared to the others) when G_c is increased from 20 to 25 with over 400 s of additional computational time. Therefore, we set G_c to 20 generations.

Table A.6
Varying G_c for case 6M1121, $|O| = 630$.

G_c	5	10	15	20	25	C.T.
5	–					655.01
10	1.23	–				1125.02
15	1.48	0.24	–			1596.70
20	1.72	0.48	0.23	–		2055.68
25	1.77	0.53	0.28	0.05	–	2423.42

C.T.: computational time (s).

Appendix B. Performance of the approaches for other cases

Table B.1
Performance comparison of MILP, PH, and GA for 2-machine case: 2M376.

n	MILP				PH				GA				GAP in Obj. (%)	
	Obj.		C.T.		Obj.		C.T.		Obj.		C.T.		GA vs. MILP	GA vs. PH
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ		
5	[a]150.00	0.00	0.01	0.00	150.00	0.00	1.84	0.09	150.00	0.00	9.18	1.87	0.00	0.00
10	[a]300.00	0.00	0.53	0.00	300.00	0.00	1.34	0.06	300.00	0.00	15.28	0.13	0.00	0.00
15	[a]450.00	0.00	113.17	0.75	450.00	0.00	1.34	0.09	450.00	0.00	16.82	0.23	0.00	0.00
20	587.00	0.00	3600.00	0.24	600.00	0.00	1.34	0.11	600.00	0.00	19.96	0.26	2.21	0.00
25	–	–	–	–	750.00	0.00	1.58	0.08	750.00	0.00	24.01	0.25	–	0.00
30	–	–	–	–	900.00	0.00	1.35	0.07	900.00	0.00	28.53	0.32	–	0.00
50	–	–	–	–	1500.00	0.00	1.37	0.08	1500.00	0.00	61.80	1.10	–	0.00
80	–	–	–	–	2309.00	0.00	1.36	0.06	2400.00	0.00	169.41	1.13	–	3.94
130	–	–	–	–	3727.00	0.00	1.47	0.04	3783.40	0.84	472.34	6.35	–	1.51
210	–	–	–	–	5862.00	0.00	1.45	0.02	6040.00	2.58	852.56	31.61	–	3.04
376	–	–	–	–	5276.00	0.00	1.50	0.04	7200.90	395.50	1063.48	72.35	–	36.48

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Optimal value.

Table B.2
Performance comparison of MILP, PH, and GA for 6-machine case: 6M1401.

n	MILP				PH				GA				GAP in Obj. (%)	
	Obj.		C.T.		Obj.		C.T.		Obj.		C.T.		GA vs. MILP	GA vs. PH
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ		
5	[a]150.00	0.00	0.00	0.00	150.00	0.00	1.79	0.08	150.00	0.00	13.80	0.75	0.00	0.00
10	[a]300.00	0.00	2.65	0.00	300.00	0.00	1.47	0.05	300.00	0.00	17.02	0.14	0.00	0.00
15	[a]450.00	0.00	64.8	0.14	450.00	0.00	1.41	0.02	450.00	0.00	20.15	0.18	0.00	0.00
20	[a]600.00	0.00	455.95	0.65	600.00	0.00	1.46	0.05	600.00	0.00	24.19	0.14	0.00	0.00
25	728.00	0.00	3600.00	0.00	750.00	0.00	1.43	0.07	750.00	0.00	31.95	0.31	3.02	0.00
30	–	–	–	–	900.00	0.00	1.43	0.08	900.00	0.00	40.57	0.50	–	0.00
50	–	–	–	–	1500.00	0.00	1.43	0.09	1500.00	0.00	73.58	1.10	–	0.00
80	–	–	–	–	2378.00	0.00	1.46	0.02	2400.00	0.00	140.25	2.13	–	0.93
130	–	–	–	–	3872.00	0.00	1.50	0.02	3900.00	0.00	220.45	4.32	–	0.72
210	–	–	–	–	6037.00	0.00	1.49	0.02	6300.00	0.00	422.77	5.63	–	4.36
340	–	–	–	–	9571.00	0.00	1.56	0.03	10097.90	7.81	962.57	43.80	–	5.51
630	–	–	–	–	15296.00	0.00	1.81	0.02	18297.70	26.25	2243.25	93.33	–	19.62
1401	–	–	–	–	12577.00	0.00	2.75	0.12	19167.60	438.61	3602.10	1.75	–	52.40

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Optimal value.

Appendix C. Results of three scenarios of other cases

Table C.1
Performance comparison of GA and PH as varying unsupervised ratio on 2M376 and 6M1401.

Case	ρ_U	t_U	PH				GA				% Gap in Obj.
			Obj.		C.T.		Obj.		C.T.		
			μ	σ	μ	σ	μ	σ	μ	σ	
2M376	0.00	0	6436.00	0.00	1.59	0.11	7490.70	278.33	1534.09	76.37	16.39
	0.25	6	6216.00	0.00	1.58	0.11	7322.00	370.21	1369.19	81.16	17.79
	[a]0.50	12	5276.00	0.00	1.50	0.04	7200.90	395.50	1063.48	72.35	36.48
	0.75	18	4926.00	0.00	1.62	0.02	7369.00	250.76	771.39	47.35	49.59
	1.00	24	4456.00	0.00	1.63	0.11	6777.70	326.93	666.57	63.38	52.10
6M1401	0.00	0	14771.00	0.00	2.43	0.12	17661.70	403.23	3600.13	0.08	19.57
	0.25	6	12989.00	0.00	2.43	0.07	18821.40	311.42	3600.51	1.33	44.90
	[a]0.50	12	12577.00	0.00	2.75	0.12	19167.60	438.61	3602.10	1.75	52.40
	0.75	18	10815.00	0.00	2.43	0.11	17909.70	585.55	3601.22	2.35	65.60
	1.00	24	9014.00	0.00	2.46	0.14	16293.30	630.43	3601.14	1.43	80.76

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Base case.

Table C.2
Performance comparison of GA and PH when varying priority ratio on 2M376 and 6M1401.

Case	ρ_P	$\lvert O_P \rvert$	PH				GA				% Gap in Obj.
			Obj.		C.T.		Obj.		C.T.		
			μ	σ	μ	σ	μ	σ	μ	σ	
2M376	0.25	100	6297.00	0.00	1.84	0.11	7895.70	154.15	1125.57	31.88	25.39
	0.38	146	6340.00	0.00	1.56	0.06	7797.70	105.73	1107.48	46.42	22.99
	[a]0.53	183	5276.00	0.00	1.62	0.09	7200.90	395.50	1063.48	72.35	36.48
	0.63	232	5249.00	0.00	1.72	0.08	6660.30	87.07	1017.68	38.41	26.89
	0.75	277	2740.00	0.00	1.79	0.08	6365.30	102.96	1036.11	41.91	132.31
6M1401	0.25	351	16968.00	0.00	2.45	0.11	20608.30	530.59	3601.40	1.71	21.45
	0.38	527	15525.00	0.00	2.44	0.10	19205.10	584.37	3600.65	0.96	23.70
	[a]0.49	698	12577.00	0.00	2.40	0.08	19167.60	438.61	3602.10	1.75	52.40
	0.63	874	8526.00	0.00	2.39	0.08	15738.30	795.33	3600.87	1.10	84.59
	0.75	1050	6441.00	0.00	2.39	0.09	14034.70	716.41	3600.69	1.18	117.90

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Base case

Table C.3
Performance comparison of GA and PH when varying tool ratio on 2M376 and 6M1401.

Case	ρ_T	$\lvert T \rvert$	PH				GA				% Gap in Obj.
			Obj.		C.T.		Obj.		C.T.		
			μ	σ	μ	σ	μ	σ	μ	σ	
2M376	1.00	376	6487.00	0.00	1.74	0.09	7197.50	222.67	548.55	26.69	10.95
	1.25	470	6210.00	0.00	1.59	0.05	6964.20	222.29	698.27	32.39	12.14
	1.50	564	6143.00	0.00	1.54	0.05	6948.30	277.89	877.03	71.94	13.11
	[a]1.66	650	5276.00	0.00	1.57	0.07	7200.90	395.50	1063.48	72.35	36.48
	2.00	752	5525.00	0.00	1.56	0.05	7301.60	168.69	1427.81	66.34	32.16
6M1401	[a]1.00	1398	12577.00	0.00	2.51	0.09	19167.00	582.88	3601.48	1.60	52.40
	1.25	1750	9539.00	0.00	2.53	0.10	14880.20	662.47	3600.81	1.07	55.99
	1.50	2101	8620.00	0.00	2.78	0.09	14183.80	518.88	3600.72	1.48	64.55
	1.75	2451	8884.00	0.00	2.90	0.08	12766.40	414.50	3605.89	3.37	43.70
	2.00	2801	6555.00	0.00	3.20	0.07	11172.80	639.85	3600.42	0.35	70.45

Obj.: objective value, C.T.: computational time (seconds), μ: mean, σ: standard deviation.
[a]Base case.

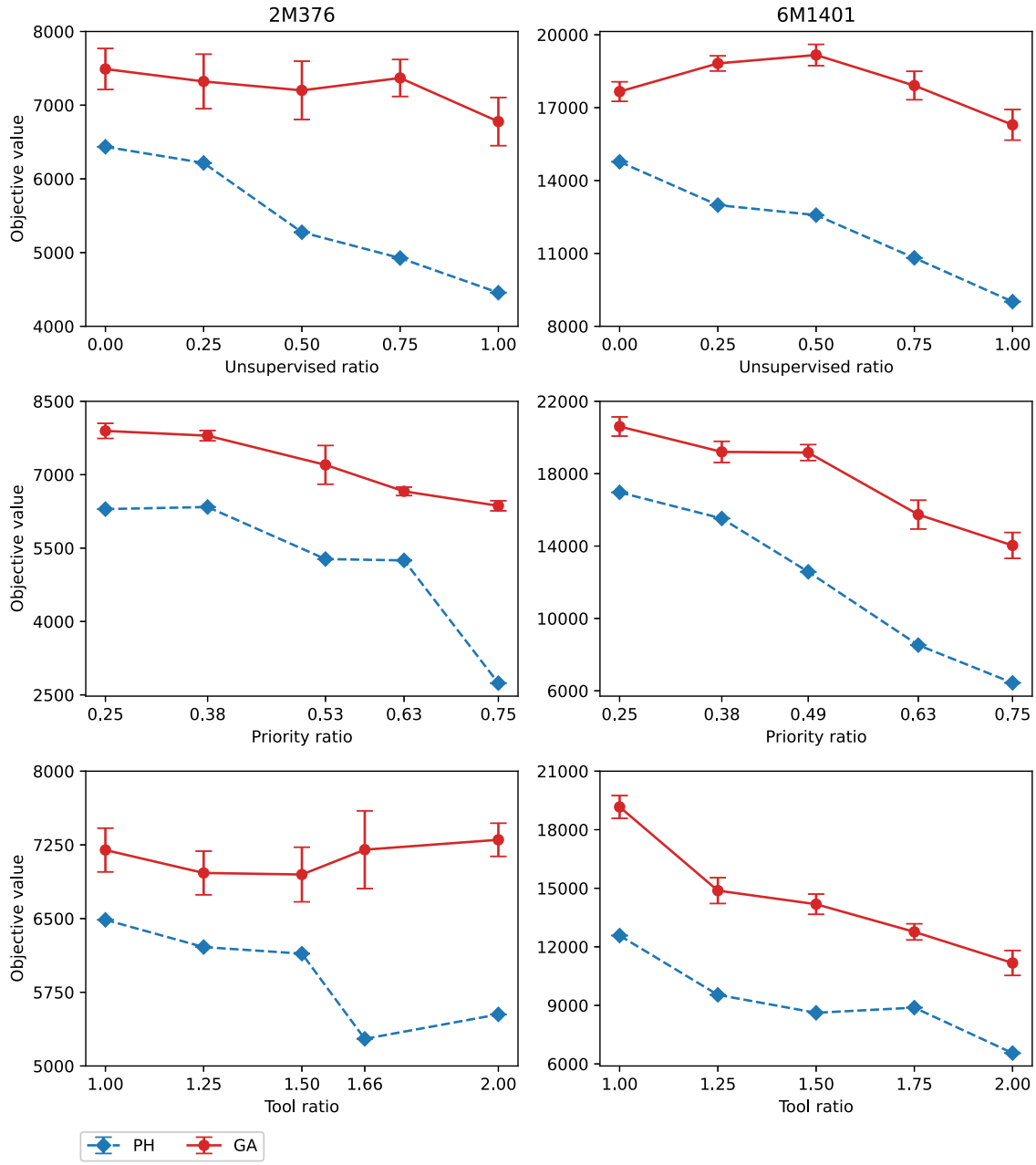

Fig. C.1. Objective value when varying unsupervised, priority, and tool ratios for base cases 2M376 and 6M1401.

Appendix D. Results on benchmark instances

To illustrate the generalizability and broader applicability of our proposed GA, we conduct an additional numerical experiment based on the Beezão instances[5] from the literature (Beezão et al., 2017). Their instances contain the number of machines, jobs, and required tools. However, Beezão's instances does not contain information about unsupervised hours, priority jobs, and a fixed scheduling horizon. Therefore, in order to experiment Beezão's instances in our paper, we provided some modifications. We first set the unsupervised shift ratio $\rho_U = 0$ (no unsupervised hours), which means we only consider supervised hours in which tool switches can always occur. Second, we set the priority ratio $\rho_P = 1$, meaning that all jobs have the same priority. Also, since Beezão et al. (2017) does not provide revenue and cost rates, we set $r = 0$ and $c_f = 0$, while keeping $c_v = 1$ and $c_p = 30$ as in our paper. Finally, we set a fixed scheduling horizon H to the obtained makespan by Beezão et al. (2017), excluding the idle time caused by the tool switches (i.e., time of completing all jobs without tool switching time; let denote it by Δ^*, so $H = \Delta^*$).[6] This is because our problem setting does not consider delay caused by tool switches. Among the largest Beezão instances, we selected 12 instances, with 6 machines, 200 jobs, and 40 tools.

[5] Beezão instances are available from https://github.com/vinhise/pmstsup.

[6] The results of Beezão instances are available from https://ars.els-cdn.com/content/image/1-s2.0-S0377221720301995-mmc1.xls.

Table D.1
Results on Beezão's instances.

Instance name	ALNS (Beezão et al., 2017)		GA	
	Δ^*	$T_{\mathcal{V}}(m_\Delta)$	$T_{\mathcal{V}}$	$T_{\mathcal{V}}(m_\Delta)$
931	1068.80	105	745	124
932	1160.90	102	684	121
933	1101.10	109	718	122
946	998.70	101	725	127
947	967.70	102	718	124
948	1001.20	117	666	121
952	1113.80	104	736	129
953	1024.40	104	696	122
954	1085.50	117	715	126
958	982.80	104	713	130
959	960.40	103	713	119
960	983.40	117	699	120

Δ^*: time of completing all jobs without tool switching time, $T_{\mathcal{V}}$: the total number of tool switches, $T_{\mathcal{V}}(m_\Delta)$: the number of tool switches on the critical machine (the most time-consuming machine)

The results of this experiment are shown in Table D.1. We report Δ^* and the number of tool switches on the critical machine ($T_{\mathcal{V}}(m_\Delta)$) from the work of Beezão et al. (2017). For our GA, we present the total number of tool switches ($T_{\mathcal{V}}$) and the number of tool switches on the critical machine ($T_{\mathcal{V}}(m_\Delta)$). Also, we run our GA for 14 400 s for each instance, and 10 runs per instance, as done in Beezão et al. (2017).

Appendix E. BnB grouping method

E.1. Maximal Intersection Minimal Union (MIMU)

For each class S, let $R(S)$ be the set of tools required by the operations in class S, i.e.,

$$R(S) = \cup_{(j,k)\in S} T_{jk}$$

In addition, let $L(S)$ be the set containing each operation $(j,k) \notin S$ such that $S \cup \{(j.k)\}$ is a class, i.e.,

$$L(S) = \{(j,k) : (j,k) \in O \setminus S, |R(S) \cup T_{jk}| \leq C\}$$

The MIMU heuristic is presented in Algorithm E.1 as follows.

Algorithm E.1 Maximal Intersection Minimal Union (MIMU)

Require: set of operations O, set of tools T
Step 1. Set $i = 1$
Step 2. Set $S_i = \emptyset$
Step 3. Pick an operation $(j,k) \in O$ that maximizes $|T_{jk}|$ over the set O. Set $S_i = \{(j,k)\}$ and $O = O \setminus \{(j,k)\}$. Stop if $O = \emptyset$; else continue.
Step 4. If $L(S_i) = \emptyset$ then set $i = i + 1$ and go to *Step 2*.
Step 5. Select an operation $(j,k) \in O$ that maximizes $[|R(S) \cap T_{jk}|, -|R(S) \cup T_{jk}|]$ lexicographically over the set $L(S_i)$. Set $S_i = S_i \cup \{(j,k)\}$, $O = O \setminus \{(j,k)\}$. Stop if $O = \emptyset$; else go to *Step 4*.

E.2. Sweeping procedure

For a set of operations O, operation (j,k) is *compatible* with operation (j',k') if the set $\{(j,k),(j',k')\}$ is a class, i.e., $|T_{jk} \cup T_{j'k'}| \leq T_C$. Each operation is compatible with itself since no operation requires more than T_C tools. The set O has the $|O| \times |O|$ compatibility matrix $C(O)$, where $((j,k),(j',k'))$ entry is equal to 1 if operation (j,k) is compatible with operation (j',k'), and 0 otherwise. Let $B(j,k)$ denote the set of operations that are compatible with operation (j,k), thus $B(j,k) = \{(j',k') : C(O)_{(j,k),(j',k')} = 1\}$. The sweeping procedure is presented in Algorithm E.2 as follows.

Algorithm E.2 Sweeping procedure

Require: set of operations O, set of tools T
Step 1. Set $i = 1$
Step 2. Set $S_i = \emptyset$. Compute the compatible matrix O.
Step 3. Select an operation $(j,k) \in O$ that minimizes $[|B(j,k)|, |\cup_{(j',k')\in B(j,k)} T_{j'k'}|]$ lexicographically over the set O. Set $O = O \setminus B(j,k)$ and $S_i = B(j,k)$.
Step 4. Stop if $O = \emptyset$; else set $i = i + 1$ and go to *Step 2*.

E.3. Bnb procedure

The BnB method, proposed by Tang and Denardo (1988b), creates a search tree whose the root node is labeled $\overline{n}$. Each node i is associated with a maximal class S_i. For each node i, let $D(i)$ denote the depth of node i, i.e., the number of nodes in the path from node $\overline{n}$ to node i. In addition, $TS(i)$ is the total set of operations associated with the nodes from node $\overline{n}$ to node i, i.e., the operations that have been placed in classes are those

in $TS(i)$. For each node i, it defines $O(i) = O \setminus TS(i)$, so $O(i)$ is the set of operations that must yet be placed in classes. Each node i also adjusts the number of tool slots: $T_C = T_C - |\cup_{(j,k)\in TS(i)} T_{jk}|$.

Let $SW(i)$ be the lower bound corresponding to set $O(i)$, i.e., $SW(i) = \max\{\lceil|\cup_{(j,k)\in O(i)} T_{jk}|/C\rceil, L\}$, where L is the lower bound derived from the sweeping procedure for set $O(i)$. Also, let $lb(i)$ denote the lower bound associated with node i, where $lb(i) = D(i) + SW(i)$. Similarly, let $ub(i)$ denote the upper bound associated with node i, where $ub(i) = D(i) + Ul(i)$, and $Ul(i)$ is the number of subsets in the partition of $O(i)$ derived from the MIMU procedure.

The BnB method is presented in Algorithm E.3. Initially, at root node $\overline{n}$, $TS(\overline{n}) = \emptyset$, $L^* = SW(\overline{n})$, and $U^* = Ul(\overline{n})$, where L^* and U^* denote the current lower and upper bounds, respectively.

Algorithm E.3 BnB procedure

Require: set of operations O, set of tools T

Step 0. (Initialization) Set $U^* = Ul(\overline{n})$ and $L^* = SW(\overline{n})$. Set $i = \overline{n}$ and $S_i = \emptyset$.

Step 1. (Branching) Compute $O(i)$. Choose the operation $(j,k) \in O(i)$ compatible with the fewest number of operations in $O(i)$. Create a new node h for each maximal class of $O(i)$ containing operation (j,k) Create an arc from node i to the new node h and compute C for this new node. Set $D(h) = D(i) + 1$.

Step 2. (Bounding) For each of this new node h, compute $lb(h)$ and ub(h).
(a) If the partition found by the sweeping procedure at node h is a feasible partition, set $ub(h) = lb(h)$.
(b) If $ub(h) < U^*$, set $U^* = ub(h)$ and record the corresponding partition.

Step 3. (Pruning) Terminate a node h in the tree whenever one of the following conditions holds.
(a) $lb(h) = ub(h)$.
(b) $lb(h) \geq U^*$.

Step 4. (Stopping) If there is no non-terminated branching node in the tree, then STOP. U^* is the optimal value and the optimal partition is the one recorded currently.

Step 5. (Node selection) Select l as the branching node l that minimizes $[SW(j,k), Ul(j,k)]$ lexicographically over the set of non-terminated branching nodes. Set $i = l$ and go to *Step 1*.

References

Agnetis, A., Detti, P., Pranzo, M., Sodhi, M.S., 2008. Sequencing unreliable jobs on parallel machines. J. Sched. 12 (1), 45.

Ahmadi, E., Goldengorin, B., Süer, G.A., Mosadegh, H., 2018. A hybrid method of 2-TSP and novel learning-based GA for job sequencing and tool switching problem. Appl. Soft Comput. 65 (January), 214–229.

Ahmadi, E., Zandieh, M., Farrokh, M., Emami, S.M., 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.

Al-Fawzan, M., Al-Sultan, K., 2003. A tabu search based algorithm for minimizing the number of tool switches on a flexible machine. Comput. Ind. Eng. 44 (1), 35–47.

Amaya, J.E., Cotta, C., Fernández, A.J., 2008. A memetic algorithm for the tool switching problem. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (Eds.), Hybrid Metaheuristics. In: Lecture Notes in Computer Science, vol. 5296, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 190–202.

Amaya, J.E., Cotta, C., Fernández-Leiva, A.J., 2011. Memetic cooperative models for the tool switching problem. Memetic Comput. 3 (3), 199–216.

Amaya, J.E., Cotta, C., Fernandez-Leiva, A.J., 2012. Solving the tool switching problem with memetic algorithms. Artif. Intell. Eng. Des. Anal. Manuf. 26 (2), 221–235.

Baykasoğlu, A., Ozsoydan, F.B., 2017. Minimizing tool switching and indexing times with tool duplications in automatic machines. Int. J. Adv. Manuf. Technol. 89 (5), 1775–1789.

Baykasoğlu, A., Ozsoydan, F.B., 2018. Minimisation of non-machining times in operating automatic tool changers of machine tools under dynamic operating conditions. Int. J. Prod. Res. 56 (4), 1548–1564.

Beezão, A.C., Cordeau, J.-F., Laporte, G., Yanasse, H.H., 2017. Scheduling identical parallel machines with tooling constraints. European J. Oper. Res. 257 (3), 834–844.

Box, G.E.P., Hunter, W.G., Hunter, J.S., 2005. Factorial designs at two levels. In: Statistics for Experimenters: Design, Innovation, and Discovery, second ed. John Wiley & Sons, Inc., pp. 173–222.

Burger, A.P., Jacobs, C.G., van Vuuren, J.H., Visagie, S.E., 2015. Scheduling multi-colour print jobs with sequence-dependent setup times. J. Schedul. 18 (2), 131–145.

Calmels, D., 2019. The job sequencing and tool switching problem: State-of-the-art literature review, classification, and trends. Int. J. Prod. Res. 57 (15–16), 5005–5025.

Calmels, D., 2022. An iterated local search procedure for the job sequencing and tool switching problem with non-identical parallel machines. European J. Oper. Res. 297 (1), 66–85.

Catanzaro, D., Gouveia, L., Labbé, M., 2015. Improved integer linear programming formulations for the job Sequencing and tool Switching Problem. European J. Oper. Res. 244 (3), 766–777.

Chaves, A., Lorena, L., Senne, E., Resende, M., 2016. Hybrid method with CS and BRKGA applied to the minimization of tool switches problem. Comput. Oper. Res. 67, 174–183.

Crama, Y., Kolen, A.W., Oerlemans, A.G., Spieksma, F.C., 1994. Minimizing the number of tool switches on a flexible machine. Int. J. Flexible Manuf. Syst. 6 (1), 33–54.

da Silva, T.T., Chaves, A.A., Yanasse, H.H., 2021. A new multicommodity flow model for the job sequencing and tool switching problem. Int. J. Prod. Res. 59 (12), 3617–3632.

Dang, Q.V., van Diessen, T., Martagan, T., Adan, I., 2021. A matheuristic for parallel machine scheduling with tool replacements. European J. Oper. Res. 291 (2), 640–660.

Djellab, H., Djellab, K., Gourgand, M., 2000. A new heuristic based on a hypergraph representation for the tool switching problem. Int. J. Prod. Econ. 64 (1), 165–176.

Florescu, A., Barabas, S.A., 2020. Modeling and simulation of a flexible manufacturing system—A basic component of industry 4.0. Appl. Sci. 10 (22), 8300.

Furrer, M., Torsten Mütze, T., 2017. An algorithmic framework for tool switching problems with multiple objectives. Eur. J. Oper. Res. 259, 1003–1016.

Gao, J., Sun, L., Gen, M., 2008. A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. Comput. Oper. Res. 35 (9), 2892–2907.

Gen, M., Cheng, 1999. Genetic Algorithms and Engineering Optimization. John Wiley & Sons, Inc., p. 512.

Gen, M., Cheng, R., Lin, L., 2008a. Network Models and Optimization: Multiobjective Genetic Algorithm Approach. Springer, London.

Gen, M., Cheng, R., Lin, L., 2008b. Multiobjective Genetic Algorithms. In: Network Models and Optimization: Multiobjective Genetic Algorithm Approach. Springer, London, pp. 1–48.

Ghiani, G., Grieco, A., Guerriero, E., 2010. Solving the job sequencing and tool switching problem as a nonlinear least cost Hamiltonian cycle problem. Networks 55 (4), 379–385.

Ghrayeb, O.A., Phojanamongkolkij, N., Finch, P.R., 2003. A mathematical model and heuristic procedure to schedule printed circuit packs on sequencers. Int. J. Prod. Res. 41 (16), 3849–3860.

Gökgür, B., Hnich, B., Özpeynirci, S., 2018. Parallel machine scheduling with tool loading: A constraint programming approach. Int. J. Prod. Res. 56, 5541–5557.

Hertz, A., Laporte, G., Mittaz, M., Stecke, K.E., 1998. Heuristics for minimizing tool switches when scheduling part types on a flexible machine. IIE Trans. 30 (8), 689–694.

Karakayalı, İ., Azizoğlu, M., 2006. Minimizing total flow time on a single flexible machine. Int. J. Flexible Manuf. Syst. 18 (1), 55–73.

Keung, K.W., Ip, W.H., Lee, T.C., 2001a. A genetic algorithm approach to the multiple machine tool selection problem. J. Intell. Manuf. 12 (4), 331–342.

Keung, K.W., Ip, W.H., Lee, T.C., 2001b. The solution of a multi-objective tool selection model using the GA approach. Int. J. Adv. Manuf. Technol. 18 (11), 771–777.

Khan, B.K., Gupta, B.D., Gupta, D.K.S., Kumar, K.D., 2000. A generalized procedure for minimizing tool changeovers of two parallel and identical CNC machining centres. Prod. Plan. Control 11 (1), 62–72.

Laporte, G., Salazar-González, J.J., Semet, F., 2004. Exact algorithms for the job sequencing and tool switching problem. IIE Trans. 36 (1), 37–45.

Lu, Y., 2017. Industry 4.0: A survey on technologies, applications and open research issues. J. Ind. Inf. Integr. 6, 1–10.

Mecler, J., Subramanian, A., Vidal, T., 2021. A simple and effective hybrid genetic search for the job sequencing and tool switching problem. Comput. Oper. Res. 127, 105153.
Noël, M., Sodhi, M.S., Lamond, B.F., 2007. Tool planning for a lights-out machining system. J. Manuf. Syst. 26 (3–4), 161–166.
Özpeynirci, S., Gökgür, B., Hnich, B., 2016. Parallel machine scheduling with tool loading. Appl. Math. Model. 40, 5660–5671.
Paiva, G.S., Carvalho, M.A.M., 2017. Improved heuristic algorithms for the job sequencing and tool switching problem. Comput. Oper. Res. 88, 208–219.
Qin, J., Liu, Y., Grosvenor, R., 2016. A categorical framework of manufacturing for industry 4.0 and beyond. Procedia cirp 52, 173–178.
Raduly-Baka, C., Knuutila, T., Nevalainen, O.S., 2005. Minimising the Number of Tool Switches with Tools of Different Sizes. TUCS Technical Reports 690, Turku Centre for Computer Science.
Rifai, A.P., Windras Mara, S.T., Norcahyo, R., 2022. A two-stage heuristic for the sequence-dependent job sequencing and tool switching problem. Comput. Ind. Eng. 163, 107813.
Salonen, K., Raduly-Baka, C., Nevalainen, O.S., 2006a. A note on the tool switching problem of a flexible machine. Comput. Ind. Eng. 50 (4), 458–465.
Salonen, K., Smed, J., Johnsson, M., Nevalainen, O., 2006b. Grouping and sequencing PCB assembly jobs with minimum feeder setups. Robot. Comput.-Integr. Manuf. 22 (4), 297–305.
Sarmadi, H., Gholami, S., 2011. Modeling of tool switching problem in a flexible manufacturing cell: with two or more machines. In: Xie, Y. (Ed.), International Conference on Mechanical and Electrical Technology, 3rd, Vol. 1–3. ICMET-China 2011, ASME Press, pp. 2345–2349.
Schwerdfeger, S., Boysen, N., 2017. Order picking along a crane-supplied pick face: The SKU switching problem. European J. Oper. Res. 260 (2), 534–545.
Sherali, H.D., Cole Smith, J., 2001. Improving discrete model representations via symmetry considerations. Manage. Sci. 47 (10), 1396–1407.
Shivanand, H., Benal, M., Koti, V., 2006. FMS Introduction and Description. In: Flexible Manufacturing System. New Age International (P) Ltd., Publishers, New Delhi, pp. 1–17.
Solimanpur, M., Rastgordani, R., 2012. Minimising tool switching and indexing times by ant colony optimisation in automatic machining centres. Int. J. Oper. Res. 13 (4), 465–479.
Tang, C.S., Denardo, E.V., 1988a. Models arising from a flexible manufacturing machine, part I: minimization of the number of tool switches. Oper. Res. 36 (5), 767–777.
Tang, C.S., Denardo, E.V., 1988b. Models arising from a flexible manufacturing machine, part II: minimization of the number of switching instants. Oper. Res. 36 (5), 778–784.
Tzur, M., Altman, A., 2004. Minimization of tool switches for a flexible manufacturing machine with slot assignment of different tool sizes. IIE Trans. 36 (2), 95–110.
Van Hop, N., 2005. The tool-switching problem with magazine capacity and tool size constraints. IEEE Trans. Syst. Man Cybern. - A 35 (5), 617–628.
Van Hop, N., Nagarur, N.N., 2004. The scheduling problem of PCBs for multiple non-identical parallel machines. European J. Oper. Res. 158 (3), 577–594.
Yadav, A., Jayswal, S.C., 2018. Modelling of flexible manufacturing system: a review. Int. J. Prod. Res. 56 (7), 2464–2487.