Computers & Operations Research 136 (2021) 105488

Contents lists available at ScienceDirect

Computers and Operations Research

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

A learning-based two-stage optimization method for customer order scheduling

Zhongshun Shi [a,*], Hang Ma [a], Meiheng Ren [a], Tao Wu [b], Andrew J. Yu [a]

[a] *Department of Industrial and Systems Engineering, University of Tennessee, Knoxville, TN 37996, USA*
[b] *Advanced Analytics Department, Dow, Midland, MI 48642, USA*

ARTICLE INFO

Keywords:
Customer order scheduling
Artificial intelligence
Dispatching rules
Genetic programming
Heuristics

ABSTRACT

This paper addresses the customer order scheduling problem in parallel production environment commonly appearing in the pharmaceutical and paper industries. The problem aims to minimize the total completion time of the orders with their jobs processed on dedicated machines in parallel. To deal with the computational challenge of large-scale problems, we propose a learning-based two-stage optimization method consisting of a learned dispatching rule in the first stage and an adaptive local search in the second stage. The new dispatching rules are automatically generated by the proposed feature-enhanced genetic programming method in an off-line learning manner. Based on the high-quality initial solutions provided by the learned dispatching rule, we develop an adaptive local search to further improve the solution quality. Numerical results indicate the superiority of the learned dispatching rule and show the proposed two-stage optimization method significantly outperforms state-of-the-art methods in the literature.

1. Introduction

This paper studies the customer order scheduling (COS) problem in parallel machine environment to minimize the total completion time of all the customer orders. The manufacturing facility consists of multiple machines and each machine is dedicated to produce one particular type of product. Each customer order is composed of several individual products each with a demand that needs to be processed by a dedicated machine. Since the products of an order are independent with each other, they can be processed on different machines in parallel. An order is completed when all its included products are completed. The objective is to find a schedule to minimize the total completion time of the orders.

Problem COS has a large number of practical applications in manufacturing. Typical industrial applications include equipment maintenance and repair (Leung et al., 2005a), finishing paper processing (Leung et al., 2006) and pharmaceutical packaging (Venditti et al., 2010). Moreover, problem COS can also be regarded as a degenerate case of two-stage assembly scheduling problem if ignoring the assembly processing time in the second stage, such as part kitting (Yang and Posner, 2005) and semi-product manufacturing (Lu et al., 2019). With the increasingly diversified demands from the customers, modern manufacturing is developing rapidly towards mass customization under the make-to-order production manner. To avoid additional handling and shipment cost, most of the customers prefer to receive all kinds products included in the order in a single shipment. This shifts the focus of attention from the availability of individual products to that of the holistic order for both the producers and the customers.

In the literature, problem COS has attracted a lot of attentions since first studied in Julien and Magazine (1990). Many studies have considered this problem with configuration of parallel machines as that in this paper. Several studies focusing on the complexity analysis have shown that problem COS is strongly NP-hard even for two machines (Wagneur and Sriskandarajah, 1993; Leung et al., 2005b; Roemer, 2006). Blocher and Chhajed (1996) studied this problem to minimize the total completion time of orders with identical parallel machines. Several heuristics were proposed by considering both the order sequence and job placement, and two lower bounds were also developed. With the same setting, Yang and Posner (2005) proposed a heuristic with tight worst-case bound $2 - 1/m$ where m is the number of machines. Additionally, two heuristics that are examined with the tight worst-case bounds 6/5 and 9/7 were developed for two machines case. In Leung et al. (2005a), two heuristics, namely earliest completion time first (ECT) and shortest processing time first applied to the machine with the largest load (SPTL), were proposed where the former was proved to have the worst-case bound m. The authors also analyzed the other three dispatching rules, namely shortest total processing time first (STPT),

* Corresponding author.
E-mail addresses: tzshi@utk.edu (Z. Shi), hma19@vols.utk.edu (H. Ma), mren5@vols.utk.edu (M. Ren), danielwu9999@gmail.com (T. Wu), ajyu@utk.edu (A.J. Yu).

https://doi.org/10.1016/j.cor.2021.105488
Received 14 July 2020; Received in revised form 29 April 2021; Accepted 20 July 2021
Available online 31 July 2021

shortest maximum processing time first (SMPT) and smallest maximum completion time first (SMCT) proposed by Wang and Cheng (2003) for arbitrary m machines. Along this line with problem COS, many studies focus on the different objective functions. In Shi et al. (2017), a quadratic programming formulation was proposed to minimize the total weighted completion time. Some other objectives are minimizing the total weighted completion time (Wang and Cheng, 2007; Leung et al., 2007; Wang et al., 2013), minimizing the maximum lateness and number of late orders (Leung et al., 2006), minimizing the total tardiness (Framinan and Perez-Gonzalez, 2018), and multi-objective of minimizing the linear sum of the total flowtime and the maximum tardiness (Wu et al., 2018). In addition, many variants of problem COS have been studied. In Xu et al. (2016), a branch-and-bound algorithm was proposed to solve the order scheduling problem with learning effect to minimize the total tardiness. Some studies also investigate the variant with order release times (Wu et al., 2019; Lin et al., 2019). A new order scheduling on a product-service system with time windows is studied in Zhang et al. (2019), where an order is composed of a product processed by a manufacturing plant and a service from the service center. In Chen and Li (2020), the order scheduling problem with rejection was investigated to minimize a linear sum of the maximum completion time of the accepted orders and the total penalty of the rejected orders. De Athayde Prata et al. (2020) proposed two matheuristics to solve the order scheduling problem with sequence-dependent setup times. Other variants in terms of different machine environments have also been addressed including single machine (Gupta et al., 1997), two-stage assembly flowshop (Komaki and Kayvanfar, 2015), parallel batch processing machines (Shi et al., 2018) and permutation flowshop (Meng et al., 2019). In addition, metaheuristics are commonly used to solve problem COS and its variants, including genetic algorithm (GA) (Xiong et al., 2014; Cunha Campos and Claudio Arroyo, 2014; Jung et al., 2017; Daoud et al., 2018), particle swarm optimization (PSO) (Lin et al., 2017; Wu et al., 2018), simulated annealing (Xu et al., 2016; Kung et al., 2018; Zhang et al., 2019), artificial bee colony algorithm (Wang et al., 2017; Lin et al., 2019; Meng et al., 2019), tabu search (Hazır et al., 2008; Zhang et al., 2019) and differential evolution (Du et al., 2018). For more details about problem COS, the readers can refer to the survey paper (Framinan et al., 2019).

Recently, Framinan and Perez-Gonzalez (2017) proposed an effective dispatching rule for problem COS by utilizing a look-ahead indicator to assess the impact of both the scheduled and unscheduled orders. This dispatching rule denoted by NEW outperforms the existing dispatching rules in the literature. Based on this new dispatching rule, a greedy search algorithm was also developed to solve problem COS more efficiently. Riahi et al. (2019) proposed a new constructive heuristic by utilizing the processing times in multiple ways. This heuristic was then integrated into a newly developed perturbative search algorithm (PSA). PSA achieves a powerful computational performance on the total of 720 benchmark instances of problem COS and has updated the best solutions for almost all those instances.

Although existing methods in the literature have been proposed for solving problem COS efficiently, there is still a lot of room for improvement especially for the large-scale problems. This paper aims to develop a learning-based optimization approach to solving problem COS. The main contributions are summarized as follows. We develop an efficient feature-enhanced genetic programming (FEGP) method for the automatic design of high-quality dispatching rules for problem COS in an off-line learning manner. Two new dispatching rules learned by the proposed FEGP method outperform the existing well-known dispatching rules manually designed in the literature. Utilizing the high-quality initial solutions provided by the new dispatching rules, we further develop an adaptive local search algorithm to form a two-stage optimization method denoted by LDR-AS for problem COS. On solving a total of 1080 benchmark instances, LDR-AS achieves the best computational performance compared with the state-of-the-art methods and is capable of finding the best feasible solutions for 1016 instances.

The remainder of this paper is organized as follows. Section 2 presents the problem definition and a mixed-integer linear programming (MILP) model. In Section 3, we propose the FEGP method for the automatic design of dispatching rules for problem COS. Section 4 presents two new dispatching rules learned by the proposed EFGP method. Based on the high-quality initial solutions provided by LDR, we further design an adaptive local search algorithm as the two-stage method in Section 5. We conduct the computational experiments to test the performance of the proposed methods in Section 6. Section 7 concludes this paper.

2. Problem definition and formulation

Problem COS can be described as follows. There are a total of n available orders needed to be produced by a manufacturer for the customers. The manufacturing facility has m parallel machines with each machine dedicated to produce only one type of job. Each order from the customer is composed of at most m individual jobs, each of which should be manufactured exactly by a dedicated machine without preemption of interruption. Each machine can process at most one job at a time and the jobs in an order can be processed by multiple machines simultaneously. The completion time of each order is defined by the time at which all jobs in the order are completed. The objective of problem COS is to find a schedule to minimize the total completion time of the orders.

An important optimality property of problem COS with minimization of total weighted completion time is that there exists an optimal schedule in which the permutation of orders on each machine is the same (Blocher and Chhajed, 1996). Based on this optimality property, Shi et al. (2017) first formulated problem COS as an mixed-integer quadratic programming (MIQP) model, and then an MILP formulation was derived through applying the reformulation technique (by introducing a new decision variable α^t_{jk}) on the proposed MIQP formulation. This derived MILP formulation achieves the tightest LP-relaxation compared with the other models for this problem.

However, there is no physical meaning and illustration provided for the new introduced variable α^t_{jk} and the MILP formulation in Shi et al. (2017). In this paper, we introduce the same MILP formulation based on the following two purposes. First, we define the decision variable α^t_{jk} by giving its the physical meaning and show that we can formulate this problem directly as the same MILP model rather than the reformulation technique in Shi et al. (2017). Second, with the development of optimization technique, the state-of-the-art commercial optimization solver like CPLEX 12.8 integrates plenty of efficient algorithms and has a strong power to search the best feasible solutions for a given MILP model. Therefore, we want to compare the solution quality of the proposed method with the best feasible solutions of this MILP model solving by CPLEX 12.8. The notations and decision variables are given as follows.

Notations:

N: set of orders, $N = \{1, 2, \dots, n\}$;

M: set of machines, $M = \{1, 2, \dots, m\}$;

p^t_j: job processing time in order j on machine t, where $j \in N, t \in M$.

Decision Variables:

x_{jk}: $x_{jk} = 1$, if order j is at the position k when the jobs are processed on machines. Otherwise, $x_{jk} = 0$, where $j, k \in N$;

C_j: completion time of order j, where $j \in N$;

α^t_{jk}: $\alpha^t_{jk} = \sum_{k'=1}^{k} \sum_{j=1}^{n} x_{jk'} p^t_j - p^t_j$, i.e., the starting time of order j at position k on machine t if $x_{jk} = 1$, and otherwise, $\alpha^t_{jk} = 0$, where $j, k \in N, t \in M$.

Based on the above notations and variables, problem COS can be formulated as the following MILP model.

$$\text{(MILP)} \quad \min \sum_{j \in N} C_j \tag{1}$$

$$\text{s.t.} \quad \sum_{k \in N} x_{jk} = 1, \qquad \forall j \in N, \tag{2}$$

$$\sum_{j \in N} x_{jk} = 1, \qquad \forall k \in N, \tag{3}$$

$$0 \le \alpha^t_{jk} \le M^t_j x_{jk}, \qquad \forall j, k \in N, t \in M, \tag{4}$$

$$\sum_{j=1}^{n} \alpha^t_{jk} = \sum_{k'=1}^{k} \sum_{j=1}^{n} x_{jk'} p^t_j - \sum_{j=1}^{n} x_{jk} p^t_j, \qquad \forall k \in N, t \in M, \tag{5}$$

$$C_j \ge \sum_{k=1}^{n} \alpha^t_{jk} + p^t_j, \qquad \forall j \in N, t \in M, \tag{6}$$

$$x_{jk} \in \{0, 1\}, \qquad \forall j, k \in N. \tag{7}$$

The objective function (1) is to minimize the total completion time of the orders. Constraints (2) and (3) ensure each order can be assigned to only one position and each position can be occupied by only one order, which define a permutation of the orders. Constraints (4) ensure that $\alpha^t_{jk} = 0$ if $x_{jk} = 0$ and M^t_j is a large positive number where we set $M^t_j = \sum_{i=1}^{n} p^t_i - p^t_j$. Constraints (5) define the starting time equation for any order assigned at the position k of machine t. Constraints (6) indicate the completion time of order j on the machine t is greater than or equal to the summation of its starting time and processing time. Based on the constraints (2)–(4), it is simple to check that the left-hand side $\sum_{j=1}^{n} \alpha^t_{jk}$ of constraints (5) denotes the starting time of the order assigned at the position k of machine t and $\alpha^t_{jk} = \sum_{k'=1}^{k} \sum_{j=1}^{n} x_{jk'} p^t_j - p^t_j$ if $x_{jk} = 1$. According to this, $\sum_{k=1}^{n} \alpha^t_{jk}$ indicates the starting time of the order j on the machine t and thus constraints (6) hold. Constraints (7) are the binary variables restriction. Though the problem size that can be solved to optimality by the MILP model is small for most of scheduling problems, we will solve this model to compare the best feasible solutions obtained by CPLEX with our proposal method in Section 6.

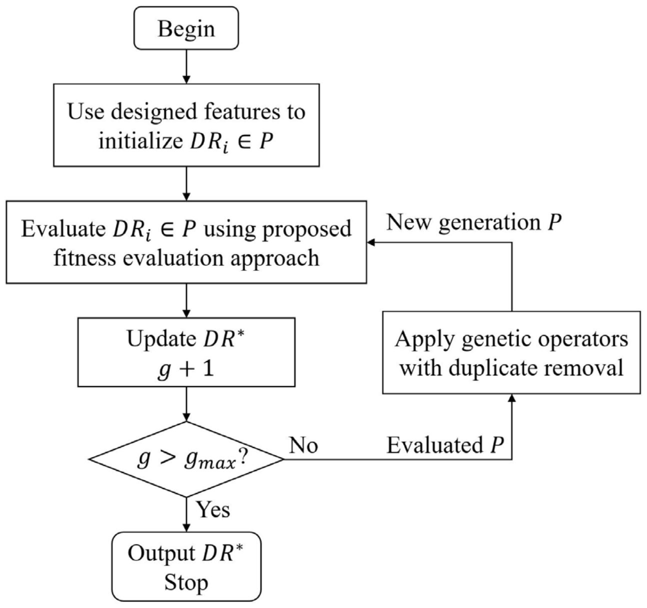
3. Proposed learning method: Feature-enhanced genetic programming

In this section, we propose the *feature-enhanced genetic programming*, denoted by FEGP, for the automatic design of dispatching rules for problem COS. Genetic programming (GP) is a powerful evolution-based artificial intelligence approach for automatic design of computer programs proposed by Koza (1992). GP can be regarded as a generalization of genetic algorithm (GA). The key focus of GP, instead of searching good solutions from the solution space conducted by GA, is to search high quality algorithms from the algorithm space. GP iteratively evolves a population of individuals using genetic operators such as mutation and crossover to find high-quality computer programs (see reviews Burke et al., 2013; Nguyen et al., 2017). During the last thirty years, extensive studies have been carried out, such as genetic operators (Koza, 1994; O'Reilly, 1995), representations (Ferreira, 2001), surrogate models (Hildebrandt and Branke, 2015; Nguyen et al., 2016) and evaluationefficiency (Shi et al., 2019), to enhance computational performance of GP.

In the proposed FEGP method, individuals in the population are dispatching rules for problem COS. By adequately utilizing the feature information of problem COS, the proposed FEGP is able to generate high-quality dispatching rules through an offline learning manner. Next, we first present the main idea of the proposed FEGP. Then we present the designs for the important components of the proposed FEGP in detail. At the end, we present the complete algorithm procedure of FEGP. For a better presentation, we summarize the notations used in FEGP in Table 1.

Table 1
Notations for the proposed FEGP.

$\mathcal{P}$	Population of the dispatching rules
$\mathcal{P}_1$	Population of the offspring dispatching rules
$N_{\mathcal{P}}$	Number of dispatching rules in population $\mathcal{P}$
g_{max}	Maximum generations of FEGP evolves
DR_i	ith dispatching rule in the population, $i \in \{1, \dots, N_{\mathcal{P}}\}$
DR^*	Best evolved dispatching rule
$f(DR_i)$	Fitness value of rule DR_i
p_e	Proportion of elite rules selected to form the next generation population
p_c	Probability of crossover operator is chosen to create offspring rules; $1 - p_c$ is the probability of mutation operator is chosen;
d_{max}	Maximum depth of a tree



Begin
Use designed features to initialize $DR_i \in P$
Evaluate $DR_i \in P$ using proposed fitness evaluation approach
New generation P
Update DR^* $g+1$
Apply genetic operators with duplicate removal
$g > g_{max}$?
No
Evaluated P
Yes
Output DR^* Stop

Fig. 1. Flow chart of the proposed FEGP.

3.1. Main idea of the proposed FEGP method

The main idea of the proposed FEGP method is to automatically generate effective dispatching rules for problem COS through an offline learning manner. The individuals in the population of the proposed FEGP are dispatching rules. FEGP iteratively evolves a population of dispatching rules, creates new dispatching rules through mutation and crossover, and selects the dispatching rules with superior performance to form new population. For ease of computer handling in FEGP, the dispatching rules of problem COS are transferred by tree-based representation. By adequately utilizing the feature information of problem COS to construct various tree-based dispatching rules, the proposed FEGP is able to generate a large number of dispatching rules and select high-quality dispatching rules based on effective fitness evaluations. Then the learned dispatching rules can be used to solve problem COS directly. According to this idea, Fig. 1 shows the detailed flow chart of our proposed FEGP method.

As shown in Fig. 1, the proposed FEGP generates a population $\mathcal{P}$ of dispatching rules and selects the current best dispatching rule iteratively. The initial population of dispatching rules is first constructed using the features of problem COS. Then all the dispatching rules are evaluated using a developed fitness evaluation function and the current best dispatching rule DR^* is updated. If the maximum generation is not reached, FEGP utilizes genetic operators such as mutation and crossover to generate the next population and the algorithm iterates. When the termination condition of maximum generation holds, the best dispatching rule DR^* is output and the FEGP method stops.

Table 2
Designed features of problem COS for FEGP to construct dispatching rules.

Feature	Description
MINPT	Minimum job processing time of order j, $MINPT_j = \min_t p_j^t$
MAXPT	Maximum job processing time of order j, $MAXPT_j = \max_t p_j^t$
OTPT	Total processing time of order j, $OTPT_j = \sum_{t=1}^{m} p_j^t$
OTWPT	Total weighted processing time of order j, $OTWPT_j = \sum_{t=1}^{m} w_t p_j^t$, herein $w_t = \sum_{j=1}^{n} p_j^t / (\sum_{t=1}^{m} \sum_{j=1}^{n} p_j^t)$
OTCWPT	Total completion time weighted processing time of order j, $OTCWPT_j = \sum_{t=1}^{m} w_t p_j^t$, $w_t = TCT^t / \sum_{i=1}^{m} TCT^i$. TCT^t is the total completion time (TCT) if following the order sequence on machine t by sorting the jobs in non-decreasing order of p_j^t
HMPT	Job processing time on the machine with the highest total processing time, $HMPT_j = p_j^{t'}$, $t' = \arg\max_t \sum_{j=1}^{n} p_j^t$
LMPT	Job processing time on the machine with the lowest total processing time, $LMPT_j = p_j^{t'}$, $t' = \arg\min_t \sum_{j=1}^{n} p_j^t$
LCPT	Job processing time on the machine with the lowest TCT, $LCPT_j = p_j^{t'}$, $t' = \arg\min_t TCT^t$. TCT^t is the same with that in OTCWPT
WLCPT	Weighted job processing time on the machine with the lowest TCT, $WLCPT_j = w_j p_j^{t'}$, t' is the same with that in LCPT, $w_j = \sum_{t=1}^{m} p_j^t / \sum_{j=1}^{n} \sum_{t=1}^{m} p_j^t$
MAXCT	Maximum completion time of jobs in order j assuming it is selected next, $MAXCT_j = \max_t C_j^t$
MINCT	Minimum completion time of jobs in order j assuming it is selected next, $MINCT_j = \min_t C_j^t$
ODCT	$ODCT_j = C_j - C_k$, difference of completion time of order j and that of order k immediately before j, assuming j is selected next
OMS	Makespan of order j assuming it is selected next, $OMS_j = C_j - C_k^{t'}$, where k is the order immediately before j and $C_k^{t'}$ denotes the earliest completion time of jobs in order k
EXPT	Extra processing time of order j assuming it is selected next, $EXPT_j = C_j - C_k^{t'} - \max_t p_j^t$ where k is the order immediately before j and $C_k^{t'}$ is the same with that in OMS
NUO	Number of unscheduled orders
EST	Earliest completion time of current partial schedule, $EST = \min_t C^t$
LST	Latest completion time of current partial schedule, $LST = \max_t C^t$
AST	Average completion time of current partial schedule, $AST = \sum_{t=1}^{m} C^t / m$
Constant	0, 1

There are several important components in the proposed FEGP significantly affecting the quality of the learned dispatching rules. First, the features of problem COS must be well designed such that useful problem information can be utilized to construct high-quality dispatching rules. Second, a good fitness evaluation function should be developed such that superior dispatching rules can be accurately selected. Thirdly, to maintain the diversity of the population, specific mutation and crossover operators with duplicate removal strategy should be developed such that more powerful dispatching rules are possible to be discovered. Next, we present the designs of these important components in detail.

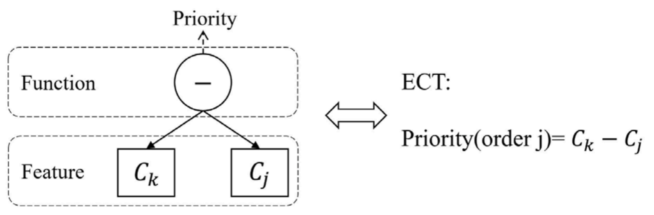


Fig. 2. An illustrative example on tree representation of dispatching rule ECT.

3.2. Feature and function design for FEGP to construct dispatching rules

In the proposed FEGP, dispatching rules are constructed using features and functions in tree-based form. A *feature* is a problem related attribute, variable of constant that contains useful information to help solve the studied problems. A *function* can be any arithmetic operator, logic operator and mathematical function. We use tree representation to express dispatching rules for problem COS in this paper. Fig. 2 shows an illustrative example how the well-known dispatching rule ECT proposed in Leung et al. (2005a) for problem COS can be expressed as a tree. In this example, C_k and C_j at the leaf nodes of the tree are features. Besides, operator "−" at the internal node is the minus, which is an important part to construct the tree of ECT. Feature C_k denotes the completion time of the last order k in the partial schedule and feature C_j is the completion time of order j assuming it is selected to schedule immediately after k. By this means, $C_k - C_j$ denotes the priority of order j, and the one with the earliest completion time thus has the largest priority to be scheduled next. Obviously, the features C_j and C_k are the key factors to determine the effectiveness of the dispatching rule ECT.

Motivated by the fast and robust computational performance of the well-known dispatching rules manually developed in the literature, we conduct the feature design for the proposed FEGP method by extracting the important features from those well-known dispatching rules. Some of the well-known dispatching rules like ECT are also approximation algorithms. In general, a dispatching rule for problem COS contains one or several features that are key factors to determine its effectiveness for solving the problem. By thoroughly exploring dozens of well-known dispatching rules, including STPT, SMPT, SMCT, ECT, SPTL (Wang and Cheng, 2003; Leung et al., 2005a) and eight dispatching rules proposed in Riahi et al. (2019), we design eighteen useful features of problem COS and summarize them in Table 2. These features cover most information of the unscheduled orders and the current partial schedule in solving problem COS. For example, the total job processing time of an unscheduled order denoted by OTPT, is the key feature of rule STPT. Furthermore, we include the constants 0 and 1 to increase varied structures of the dispatching rules. As discussed in Fig. 2, the ECT rule uses $C_k - C_j$ to calculate the priority of order j. According to Table 2, feature $ODCT = C_j - C_k$, thus ECT can also be expressed as $0 - ODCT_j$ to calculate the priority equivalently.

In addition, we design the functions used to organize features into dispatching rules and present them in Table 3. The function set consists of basic arithmetic operators, the maximum function and the ternary version of the if-then-else function (if $a \geq 0$ then b else c; a, b and c are the related sub-expressions evolved). Involving such logical operators would enrich the search power to find more combinations of features and evolve advanced scheduling logic.

Using the designed features and functions, population $\mathcal{P}$ can be initialized by integrating well-known dispatching rules as initial individuals. We first transfer these dispatching rules into trees, then insert them into the initial population $\mathcal{P}$ of FEGP. For example, rule STPT can be transferred into a tree $(0 - OTPT)$ using the developed features, where smaller OTPT will have larger priority. Then, the rest rules are randomly generated by the well known *Ramped half-and-half* method (Koza, 1992). *Ramped half-and-half* creates a half of population by *full* method and the other half by *grow* method. The *full* method cultivates a tree in a way that the features could only occur at nodes in the maximum depth d_{max}, and all the internal nodes are assigned with functions. The *grow* method, on the contrary, allows to select both the features and functions to assign on the nodes not at the d_{max} depth.

Table 3
Functions of the proposed FEGP for problem COS.

Function	Description
+	Addition, binary operator
−	Subtraction, binary operator
×	Multiplication, binary operator
÷ (or div)	Protected division, binary operator
if-then-else(a,b,c)	If $a \geq 0$ then b else c, ternary operator
$\max(a,b)$	Maximum of a and b, binary operator

With the designed features in Table 2 and functions in Table 3, dispatching rules of problem COS can be constructed in the tree-based form. Consider the LDR in which all the leaf nodes grow to the maximum depth d_{max}, it will give the worst-case time complexity for any LDR generated by FEGP. We present this complexity analysis in the following lemma.

Lemma 1. *For a given maximum tree depth d_{max}, any learned dispatching rule by the proposed FEGP method is a polynomial-time algorithm with the worst-case time complexity of $O(mn^3/2 + n_f n^2/2 + m^2 n)$, where $n_f = (3^{d_{max}-1} - 1)/2$ is the maximum number of functions in the tree-based LDR, m is the number of machines and n is the number of orders.*

Proof. Let $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ be an order sequence, where $\pi(k), k = 1, \ldots, n$ is the order at position k. LDR constructs a sequence by selecting one order a time from the unscheduled orders and appending at the end of current partial sequence, denoted by $\pi = \{\pi(1), \pi(2), \ldots, \pi(k-1)\}$, $\pi = \emptyset$ when $k = 1$. This implies for each position $k = 1, \ldots, n$, LDR needs to calculate the priority of $n-k+1$ unscheduled orders, which gives $\sum_{k=1}^{n}(n-k+1) = n(n+1)/2$ times of priority calculation using LDR. Next, we analyze the worst-case time complexity for one-time priority calculation using LDR. Given a maximum tree depth d_{max}, the worst-case time complexity of LDR is obtained if all the leaf nodes of LDR grow to d_{max} and all the internal nodes from depth 1 to $d_{max} - 1$ are ternary functions, i.e. if-then-else(a,b,c). This indicates the total number of functions in LDR is $\sum_{d=1}^{d_{max}-1} 3^{d-1} = (3^{d_{max}-1} - 1)/2$. Let $n_f = (3^{d_{max}-1} - 1)/2$, it follows n_f time is needed to calculate the priority of one order if the features are ready. Next, we consider the feature with the maximum time complexity in the designed features in Table 2. Feature WLCPT first finds machine $t' = \arg\min_t TCT^t$ where TCT^t is the total completion time if following the order sequence on machine t by sorting the jobs in non-decreasing order of p_j^t. This takes $O(m^2 n + mn\log n)$ time. Then feature WLCPT calculates the order weight w_j by using $w_j = \sum_{t=1}^{m} p_j^t / \sum_{j=1}^{n}\sum_{t=1}^{m} p_j^t$, which takes time $O(mn)$. Overall, $WLCPT_j = w_j p_j^{t'}$ takes $O(m^2 n + mn\log n)$ time. Since feature WLCPT is static constant, meaning they can be used repeatedly without recalculating. If all the nodes at depth d_{max} are WLCPT, it takes $O\left(m^2 n + mn\log n + n_f n(n+1)/2\right) = O\left(m^2 n + mn\log n + n_f n^2/2\right)$ time. However, feature ODCT needs to be recalculated every time before priority calculation. As $ODCT_j = C_j - C_k$ is obtained by applying partial sequence $\pi = \{\pi(1), \ldots, \pi(k), j\}$, it takes $O(mn)$ time. Thus, if all the nodes at d_{max} are ODCT, it takes $O\left(n(n+1)(mn+n_f)/2\right) = O(mn^3/2 + n_f n^2/2)$ time. Therefore, a tree-based LDR will take worst-case time complexity if contains both WLCPT and ODCT. Overall, the worst-case time complexity for any LDR is $O\left(m^2 n + mn\log n + n(n+1)(mn+n_f)/2\right) = O(mn^3/2 + n_f n^2/2 + m^2 n)$ where $n_f = (3^{d_{max}-1} - 1)/2$. Obviously, it is polynomial for any given d_{max}. □

3.3. Proposed fitness evaluation function

During the learning process of FEGP, fitness evaluation is a critical step as it determines the performance of a dispatching rule and guides the FEGP search on the space of dispatching rules. An individual with better fitness indicates it has larger probability to be selected to reproduce the next generation. We use the following equation to calculate the average relative deviation of a dispatching rule DR_i

$$dev_{avg}(DR_i) := \frac{1}{T} \sum_{I_t^s \in \{I_1^s, I_2^s, \ldots, I_T^s\}} \frac{Obj(DR_i, I_t^s) - Obj(DR_{ref}, I_t^s)}{Obj(DR_{ref}, I_t^s)}, \tag{8}$$

where $\mathcal{I}^s = \{I_1^s, I_2^s, \ldots, I_T^s\}$ is a set of data instances having the same problem size $s \in \mathcal{S}$ and T denoted the total number of instances in this problem size. $\mathcal{S}$ is a set of the total problem sizes. $Obj(DR_i, I_t^s)$ and $Obj(DR_{ref}, I_t^s)$ are the objective values by applying DR_i and the reference rule DR_{ref} on instance I_t^s, respectively. Thus, $dev_{avg}(DR_i)$ measures the average deviation between the objective values of DR_i and DR_{ref}. Herein we select ECT as DR_{ref}.

Based on the above deviation equation, we propose the ranking-based fitness evaluation function of DR_i as follows

$$f(DR_i) := \sum_{s \in \mathcal{S}} rank(DR_i, \mathcal{I}^s), \tag{9}$$

where $rank(DR_i, \mathcal{I}^s)$ is the rank (ascending order) of $dev_{avg}(DR_i)$ among the individual population on solving a set of instances $\mathcal{I}^s$ that have the same problem size s. By summing up the ranks in all problem sizes $s \in \mathcal{S}$, $\sum_{s \in \mathcal{S}} rank(DR_i, \mathcal{I}^s)$ can represent the overall rank of DR_i on solving various sizes of problem instances. Thus, we define $\sum_{s \in \mathcal{S}} rank(DR_i, \mathcal{I}^s)$ as the fitness of the dispatching rule $f(DR_i)$. The pseudo-code of this proposed fitness evaluation function in Eq. (9), denoted by `FitnessEvaluate`, is given in Algorithm 1.

Algorithm 1 `FitnessEvaluate`$(\mathcal{P})$

```
1: Generate $T$ data instances for each $s \in \mathcal{S}$
2: for $s \in \mathcal{S}$ do
3:    for $DR_i \in \mathcal{P}$ do
4:       for $I_t^s \in \{I_1^s, I_2^s, \ldots, I_T^s\}$ do
5:          $Obj(DR_i, I_t^s) \leftarrow$ apply $DR_i$ on instance $I_t^s$
6:       end for
7:       Calculate $dev_{avg}(DR_i)$ by Eq. (8)
8:    end for
9:    $rank(DR_i, \mathcal{I}^s) \leftarrow$ sort $DR_i$ among $\mathcal{P}$ in ascending order of $dev_{avg}(DR_i)$
10: end for
11: for $DR_i \in \mathcal{P}$ do
12:    $f(DR_i) \leftarrow \sum_{s \in \mathcal{S}} rank(DR_i, \mathcal{I}^s)$
13: end for
14: return $f(DR_i)$ for all $DR_i \in \mathcal{P}$
```

Algorithm 1 evaluates a DR_i using various problem instance sizes. Before evaluation, T instances for each $s \in \mathcal{S}$ are generated and used to evaluate all the dispatching rules in the population. By Hildebrandt et al. (2010), we use the same data set for each generation to reduce the impacts of data uncertainty while avoids the premature convergency of the proposed FEGP method. After the ranking information $rank(DR_i, \mathcal{I}^s)$ for each problem size s is obtained, the ranking summation $\sum_{s \in \mathcal{S}} rank(DR_i, \mathcal{I}^s)$ is set to be $f(DR_i)$. The smaller $f(DR_i)$ indicates the better fitness DR_i can obtain. Such fitness evaluation function design is expected to reduce the impact of different value ranges of $dev_{avg}(DR_i)$ and focus on selecting individuals that can perform well and robustly for different problem sizes.

3.4. Genetic operators with duplicate removal

After the fitness evaluation for each generation, genetic operators are used to select candidates from the current generation and produce new individuals for the next generation, such as selection, mutation and crossover operators (Koza, 1994). In this study, we apply the tournament selection, subtree mutation and subtree crossover operators. In both the mutation and crossover, if the newly generated rules duplicate with existing rules on solving two instances with $n = 50$ and $m = 2$, i.e. having the same objective value, we abandon it and repeat applying the operators until different trees are obtained. In addition, depth of the new trees is stilled constrained by d_{max}; otherwise repeat applying the operators. Details of the three operators are presented as following.

`Tournament()`: Randomly select a predefined tournament size n_t dispatching rules from $\mathcal{P}$, return the rule with the best fitness value.

`Mutation(`DR_1^p`)`: One parent tree DR_1^p is modified by randomly choosing one part of it and replacing with a newly generated subtree. Check for constraints of d_{max} and duplicates. For example, in Fig. 3a, the left child node LST of the parent tree is picked and replaced with a newly generated subtree ($NUO \times HMPT$) to reproduce one offspring.

`Crossover(`DR_1^p, DR_2^p`)`: Two parent trees DR_1^p, DR_2^p are recombined by randomly picking a node in each tree and swapping over the subtree connected to the picked node. Check for constraints of d_{max} and duplicates. For example, in Fig. 3b, the subtree $OTPT$ of parent 1 and subtree ($EST \times OMS$) of parent 2 are chosen and swapped over to create two new offsprings.

3.5. Algorithm procedure of the proposed FEGP method

Based on the above developed components of the proposed FEGP method, we present the detailed algorithm procedure of FEGP in Algorithm 2.

Algorithm 2 FEGP

```
1:  Set $DR^* \leftarrow null$ and $f(DR^*) \leftarrow +\infty$
2:  Invoke the designed features and functions
3:  Initialize $\mathcal{P}$
4:  $g \leftarrow 0$
5:  while $g < g_{max}$ do
6:      $\mathcal{P}_1 \leftarrow \emptyset$
7:      Apply FitnessEvaluate($\mathcal{P}$) to obtain $f(DR_i)$ for each $DR_i$
8:      $f(DR^*) \leftarrow \min_{i=1}^{N_\mathcal{P}} f(DR_i)$
9:      $DR^* \leftarrow \arg\min_{i=1}^{N_\mathcal{P}} f(DR_i)$
10:     Sort $DR_i$ in $\mathcal{P}$ in ascending order of $f(DR_i)$; select the first $p_e \times N_\mathcal{P}$
        rules $DR_1, DR_2, \dots, DR_{p_e \times N_\mathcal{P}}$
11:     $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \bigcup \{DR_1, DR_2, \dots, DR_{p_e \times N_\mathcal{P}}\}$
12:     while $N_{\mathcal{P}_1} < N_\mathcal{P}$ do
13:         Generate a random number $rand \in [0, 1]$
14:         if $rand < p_c$ then
15:             $DR_1^p$ and $DR_2^p \leftarrow$ apply Tournament() twice to select two
                parents
16:             $DR_1^c$ and $DR_2^c \leftarrow$ apply Crossover($DR_1^p, DR_2^p$) to create
                two offspring rules
17:             $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \bigcup \{DR_1^c, DR_2^c\}$
18:         else
19:             $DR_1^p \leftarrow$ apply Tournament() to select one parent rule
20:             $DR_1^m \leftarrow$ apply Mutation($DR_1^p$) to create one offspring
21:             $\mathcal{P}_1 \leftarrow \mathcal{P}_1 \bigcup \{DR_1^m\}$
22:         end if
23:     end while
24:     $\mathcal{P} \leftarrow \mathcal{P}_1$
25:     $g \leftarrow g + 1$
26: end while
27: return $DR^*$
```

Algorithm 2 runs in the following procedure. Using the problem features and functions designed in Section 3.2, we first express several well-known dispatching rules into trees and insert these trees into the initial population. The rest individuals in population $\mathcal{P}$ are randomly initialized with the tree depth no exceeding the maximum tree depth d_{max}. Then, we use the proposed fitness evaluation function `FitnessEvaluate()` developed in Section 3.3 to evaluate the performance of all dispatching rules based on various problem instance sizes and the performance ranking information of DR_i among population $\mathcal{P}$. After the fitness evaluation, the enhanced genetic operators `crossover()` and `mutation()` designed in Section 3.4, are applied to generate different offsprings with duplicate removal strategy for the next generation. The best rule DR^* is returned when the maximum generation is reached.

Table 4 summarizes the training parameters setting of the proposed FEGP method. The systematical parameter tuning of GP has been thoroughly conducted in the literatures (Koza, 1992; Hildebrandt et al., 2010; Branke et al., 2015a), and those parameter settings are widely adopted by the GP-related studies. The population size is set to be 600 to ensure the diversity of the population. For the number of generations, we run the proposed FEGP with 300 generations for 3 times and the best dispatching rule at generation 50, 100, 150, 200, 250 and 300 are selected to test the performance, respectively. The testing results show the best dispatching rule at generation 200 obtains the best performance, while dispatching rules after generation 200 exhibit overfitting. The number of evaluations per individual is highly problem specific (Branke et al., 2015b). The choice of this parameter needs to be high enough to avoid over specialization within a single generation while it needs to be low enough to keep the experiments computationally feasible. Based on our proposed fitness evaluation function, we evaluate each dispatching rule using 120 randomly generated instances, which are further divided into 12 groups and each group is a combination from $n \in \{50, 100, 200\}$ orders and $m \in \{2, 5, 10, 20\}$ machines with 10 instances. Among the 10 instances of each combination, 5 instances are generated following the processing time generation rule of Test-1 and the rest 5 following that of Test-2. It should be noted that the generations of Test-1, Test-2 and the following Test-3 are summarized in Section 6.1. Similar to Hildebrandt et al. (2010), common random numbers are used for all dispatching rules in a certain generation while different random numbers are used for different generations to avoid overfitting and reduce the impact of data uncertainty simultaneously. As FEGP is trained using randomly generated data following generation rule of Test-1 and Test-2, the new dataset Test-3 randomly generated following different generation rule can be used to cross-validate the performance of LDRs. The crossover proportion, mutation proportion, selection method, initialization method, and the maximum tree depth 17 are following recommendations from Koza (1992) and are mostly the default settings of ECJ,[1] the evolutionary computation framework we use for our FEGP implementation. We also compared three sets of crossover and mutation proportions (85%,15%) (95%,5%) and (99%,1%) with the default (90%,10%). We found these settings do not lead to improved performance and thus retained using the default settings. While greater maximum tree depth with 17 can extend the search space of FEGP, we also choose this maximum tree depth to 5 to make the learned dispatching rules easier to analyze. Based on these settings and our computing environment, the proposed FEGP method takes the training time of 1.5 h to obtain the first new dispatching rule with the maximum tree depth of 5, of about 13 h to obtain the second new dispatching rule with the maximum tree depth of 17.

4. Two new dispatching rules learned by FEGP

In this section, we present two new dispatching rules for problem COS. These two new dispatching rules are automatically generated by the proposed FEGP method and all the related experimental details are presented in Section 6.

[1] Version 23, available from https://cs.gmu.edu/~eclab/projects/ecj/.

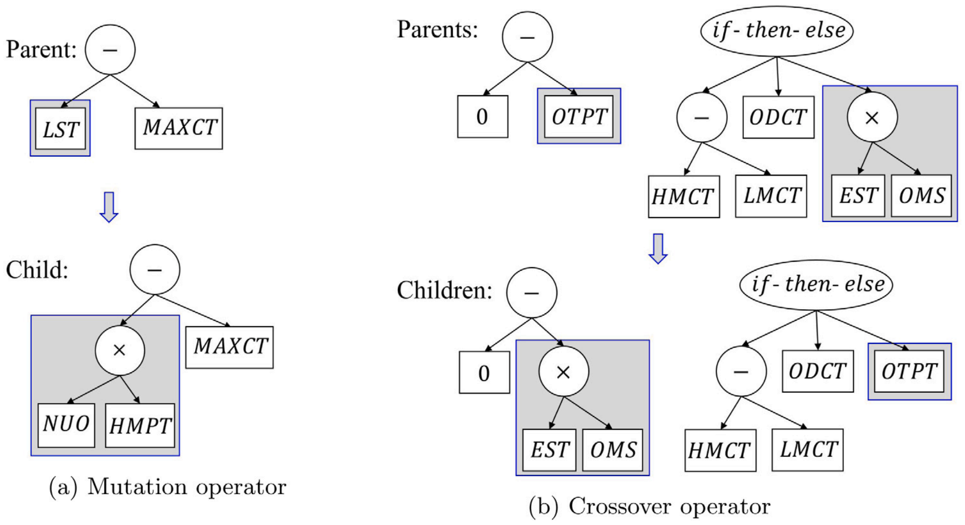


Fig. 3. Illustration of subtree mutation and crossover operators

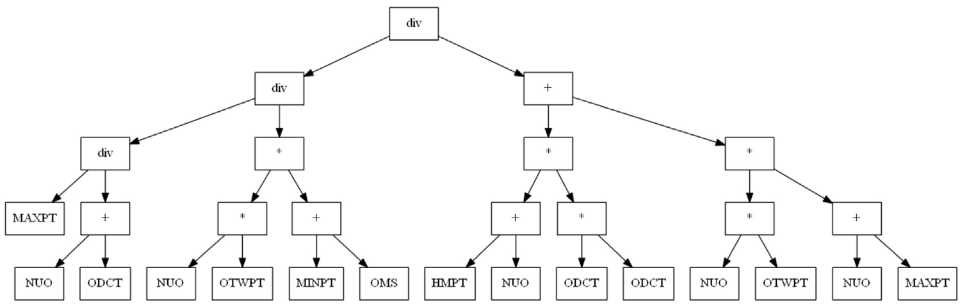


Fig. 4. Tree representation of LDR-I.

Table 4
Parameter settings of the proposed FEGP.

Population size	600
Generations	200
Number of evaluations per rule	120
Crossover proportion	90%
Mutation proportion	10%
Elites	5% of population size
Selection method	Tournament selection (Size: 7)
Initialization	Ramped half-and-half (Depth range: [2,6])
Maximum tree depth	5 or 17

4.1. New dispatching rule: LDR-I

The first new dispatching rule, denoted by LDR-I, is obtained by setting the maximum depth of the tree representation to be 5 during the training process of the proposed FEGP method. The best learned dispatching rule in the final generation is selected as LDR-I. Fig. 4 shows the tree structure of LDR-I. The corresponding closed-form of LDR-I to calculate the priority of an order j is:

$$\begin{aligned} Priority(j) = MAXPT_j / \Big(& (NUO + ODCT_j) \times NUO \times OTWPT_j \\ & \times (MINPT_j + OMS_j) \\ & \times \ \big((HMPT_j + NUO) \times ODCT_j^2 + NUO \times OTWPT_j \\ & \times (NUO + MAXPT_j) \big) \Big) \end{aligned} \tag{10}$$

In Eq. (10), all the features of problem COS such as the number of unscheduled orders NUO, are given in Table 2. By plugging in the value of each feature into Eq. (10), the order with the highest priority from the unscheduled orders is selected to schedule next. Let $\pi = \{\pi(1), \pi(2), \ldots, \pi(n)\}$ be a order sequence where $\pi(k)$ is the order at position k, C_j^t be the completion time of order j on machine t, and w_t be the weight of machine t. The detailed pseudo-code of LDR-I is presented in Algorithm 3.

Algorithm 3 LDR-I

```
1: $\pi \leftarrow \emptyset$
2: Find machine $t'$ subject to $t' = \arg\max_t \sum_{j=1}^{n} p_j^t$
3: Calculate the machine weight $w_t \leftarrow \sum_{j=1}^{n} p_j^t / (\sum_{t=1}^{m} \sum_{j=1}^{n} p_j^t)$, $t \in M$
4: for $j \in N$ do
5:     $HMPT_j \leftarrow p_j^{t'}$
6:     $MINPT_j \leftarrow \min_t p_j^t$
7:     $MAXPT_j \leftarrow \max_t p_j^t$
8:     $OTWPT_j \leftarrow \sum_{t=1}^{m} w_t p_j^t$
9: end for
10: for $k = 1$ to $n$ do
11:     $NUO \leftarrow n - (k-1)$
12:     for $j \in N$ do
13:         Obtain a temporary sequence $\pi_1 \leftarrow \{\pi(1), \ldots, \pi(k-1), j\}$
14:         $C_j, C_{k-1}$ and $C_{k-1}^t$, $t \in M \leftarrow$ apply sequence $\pi_1$
15:         $ODCT_j \leftarrow C_j - C_{k-1}$
16:         Find machine $t''$ subject to $t'' = \arg\min_t C_{k-1}^t$
17:         $OMS_j \leftarrow C_j - C_{k-1}^{t''}$
18:         $priority(j) \leftarrow$ apply Eq. (10) by plugging in $NUO$, $MINPT_j$,
            $MAXPT_j$, $HMPT_j$, $OTWPT_j$, $ODCT_j$ and $OMS_j$
19:     end for
20:     $j^* \leftarrow \arg\max priority(j)$, $j \in N$
```

```
21:     $N \leftarrow N \backslash \{j^*\}$
22:     Append $j^*$ at the end of $\pi$, i.e., $\pi = \{\pi(1), \dots, \pi(k-1), j^*\}$
23: end for
24: return $\pi$
```

Algorithm 3 constructs the order sequence by fixing one order a time at position k. Features related to the orders such as HMPT and MINPT are calculated before the scheduling process because they remain unchanged during the scheduling process. ODCT and OMS are calculated whenever they are required to be used during the scheduling process as they are dependent on the partial schedule. With all the needed features ready, priority of an order is calculated using Eq. (10) and the one with largest priority is chosen to append to π to schedule next. Next, we give the analysis on the time complexity of LDR-I.

Corollary 1. *The time complexity of LDR-I in Algorithm 3 is $O(mn^3/2 + n_f n^2/2)$, where $n_f = 14$ is the number of the functions used in LDR-I, m is the number of machines and n is the number of orders.*

Based on Lemma 1, we first identify the features with the maximum time complexity in LDR-I that are static constants, namely features $HMPT_j$, $MINPT_j$, $MAXPT_j$ and $OTWPT_j$ for order $j = 1, \dots, n$. Feature $HMPT_j = p_j^{t'}$ where $t' = \arg\max_t \sum_{j=1}^n p_j^t$ takes $O(mn)$ time. Features $MINPT_j = \min_t p_j^t$, $MAXPT_j = \max_t p_j^t$ and $OTWPT_j = \sum_{t=1}^m w_t p_j^t$ all take $O(mn)$ time for $j = 1, \dots, n$. Features $ODCT_j$ and OMS_j are obtained by applying partial sequence $\pi = \{\pi(1), \dots, \pi(k), j\}$, thus both of them take $O(mn)$ time. Then using these features, calculating the order priority by Eq. (10) contains $n_f = 14$ arithmetic operations. It follows $O\big(n(n+1)(mn + n_f)/2\big)$ time is taken for the two nested loops. Overall, the complexity of LDR-I in Algorithm 3 is $O\big(mn + n(n+1)(mn + n_f)/2\big) = O(mn^3/2 + n_f n^2/2)$. It should be noted that the term m^2n in Lemma 1 is not appearing in the time complexity of LDR-I. This is because the features like ODCT, OMS and EXPT with time complexity $O(m^2n + mn\log n)$ are not involved in LDR-I. With depth to be 5 and only 14 functions in the tree-based structure, LDR-I is very fast for solving problem COS. Among all the 1080 instances, LDR-I takes at most 0.07 s to solve each instance and achieves much better solutions than the existing dispatching rules (refer to Section 6.2 for more details).

To shed the light of the composition structure of LDR and its impacts, we can have some intuitive analysis to LDR-I and how it exhibits better performance compared with the existing dispatching rules. For example, the most commonly used features in LDR-I are: NUO (4 times), ODCT (3 times), OTWPT (2 times) and MAXPT (2 times). From the definition of ODCT and OTWPT in Table 2, $ODCT_j = C_j - C_k$ is exactly the feature of the ECT rule and $OTWPT_j = \sum_{t=1}^m w_t p_j^t$ is exactly the feature of TWPT rule. With several divisions, both ODCT and OTWPT will finally be located at the denominator of a complex fraction. This reflects the basic principles from the rules ECT and TWPT, namely, orders with smaller ODCT and OTWPT will have larger priority and thus be scheduled earlier. It is intuitive to connect the good performance of LDR-I with these well defined features in the well-known dispatching rules. Moreover, LDR-I enhances these features through various combinations. We give an illustrative example of the solutions using LDR-I and ECT on an instance of 10 orders and 2 machines as shown in Fig. 5. It can be seen LDR-I basically follows the dispatching of ECT with the exception of orders 1, 5, 6 and 9. LDR-I performs a pair swap for order pairs (6,5) and (9,1) on the basis of ECT sequence, resulting in the total completion time decreasing from 2899 to 2881.

4.2. New dispatching rule: LDR-II

In addition, by setting the maximum tree depth as 17 to further exert the power of the proposed FEGP method for learning dispatching rules, we obtained the second new dispatching rule, denoted by LDR-II. Compared with LDR-I, LDR-II is more complex in size and shape.

Table 5
Functions and features used in LDR-II.

Features (421)	0(73), ODCT(68), WLCPT(41), EST(38), 1(37), LMPT(36), EXTIME(35), HMPT(17), OTCWPT(13), AST(13), NUO(12), MAXPT(11), OTPT(11), OMS(9), OTWPT(3), MAXCT(2), LCPT(1), LST(1), MINPT(0), MINCT(0)
Functions (354)	+(111), ×(78), if-then-else(a, b, c)(66), ÷(46), max(40), −(13)

Number in the parenthesis is the frequency of the feature or function being used in LDR-II.

If we write the closed-form of LDR-II as that of LDR-I in Eq. (10), it will take more than one page. Due to the space limit and for a better representation, we do not present LDR-II in the closed-form. Alternatively, the visualization of the tree structure of LDR-II is given in Fig. 6.

Fig. 6 shows the overall tree structure of LDR-II without showing the explicit features composition. To further shed the light of LDR-II, Table 5 summarizes the statistics of the features and functions used in LDR-II, where the number in the parenthesis indicates the frequency of the feature or function being used. In LDR-II, 18 features among the total 20 are used except for feature MINPT and MINCT, while all the functions are used. The most commonly used features are 0(73 times), ODCT(68 times) and WLCPT(41 times), where feature ODCT is the exactly the key feature of rule ECT. It is thus intuitive to connect the performance of LDR-II with the frequently used features to some extent. In addition, LDR-II contains 354 functions, where function "+" is used for 111 times and the ternary function if-then-else(a, b, c) is used for 66 times. The limited total number of functions is important to the time efficiency of LDR-II. Next, we give the following corollary for the time complexity of LDR-II.

Corollary 2. *The time complexity of LDR-II is $O(mn^3/2 + n_f n^2/2 + m^2n)$ where $n_f = 354$ is the number of functions used in LDR-II, m is the number of machines and n is the number of orders.*

Based on Lemma 1, Corollary 2 is immediate as it contains both features ODCT and WLCPT. Even setting d_{max} to 17, LDR-II has only 354 functions, much smaller than the theoretical maximum number of functions $n_f = (3^{17-1} - 1)/2$. Thus, the time complexity of LDR-II is far smaller than the theoretical time complexity in Lemma 1. In fact, LDR-II takes less than 0.1 s for solving each of the 1080 instances and significantly outperforms all the well-known dispatching rules. LDR-II also obtains even much better performance than LDR-I(refer to Section 6.2 for more details). Comparing LDR-II with LDR-I, it can be seen the parameter maximum tree depth (d_{max}) plays an important role for the performance of LDR. While smaller d_{max} produces small size trees for ease of analysis and implementation, it also suffers a little loss in performance compared with larger d_{max}. Meanwhile, larger d_{max} results in the increasing of training time due to the large size of trees for fitness evaluation. From the experiment, it takes 1.5 h to train FEGP to obtain LDR-I when d_{max} is 5, and less than 13 h to obtain LDR-II when d_{max} is 17. We also provide the source code and executable program file of both LDR-I and LDR-II online.[2] We will compare LDR-I and LDR-II with existing well-known dispatching rules in Section 6. In the next section, we will develop a two-stage optimization method which uses LDR to provide initial solutions in the first stage.

5. Development of two-stage method

To further enhance the quality of the solutions generated by LDR (LDR-I or LDR-II), we develop a two-stage optimization method denoted by *LDR-based adaptive search* (LDR-AS) in this section. Fig. 7 shows the outline of the proposed LDR-AS method, i.e., LDR generates the initial solution in the first stage and then feeds the initial solution to an adaptive local search algorithm in the second stage.

[2] https://github.com/hma19/customer-order-scheduling-LDR.

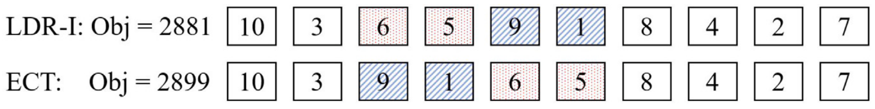


Fig. 5. Illustrative example of solutions by LDR-I and ECT.

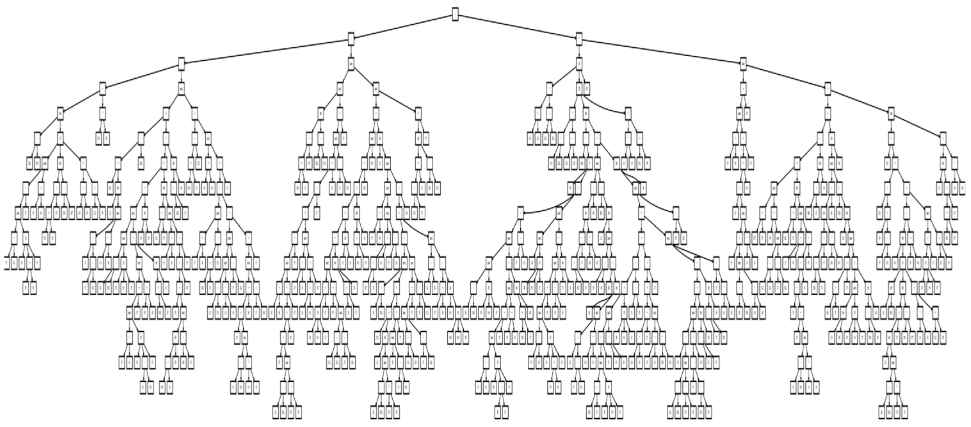


Fig. 6. Tree representation of LDR-II.

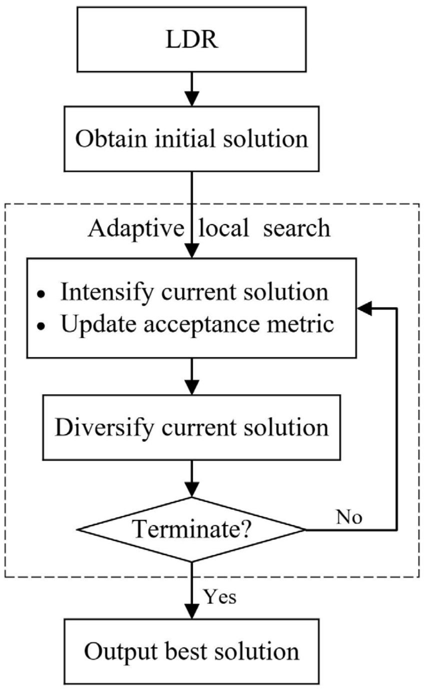


Fig. 7. Flow chart of the proposed two-stage optimization method LDR-AS.

The detailed procedure of LDR-AS is given in Algorithm 4. The LDR-AS method consists of LDR initialization stage and adaptive search stage. In the LDR initialization stage, initial solution is obtained by applying LDR to problem COS. Then the adaptive search conducts an iterative search scheme to explore the solution space based on this initial solution. In every iteration, the newly discovered best solution is appended into a collection of schedules $List$, which ensures every solution in the historical iteration does contribute to the final best solution.

Inspired by the PSA proposed in Riahi et al. (2019), the developed adaptive search algorithm also consists of two main operators: `Intensification` and `Diversification`. In the `Intensification` operator, the current best solution undergoes a bunch of local search operators to find if any potentially better solutions exist and can be accepted under certain acceptance criteria. Next, the `Diversification` operator introduces random disturbance to the solutions that have been trapped in the local optima. The iteration of adaptive search continues until the termination conditions are satisfied. $Obj(\pi)$ is the objective value of using sequence π to solve problem COS. We use i to count the number of iterations. Parameter s is used to count the times of reproducible results appear consecutively. For instance, if the result after intensification of the 10th (current) iteration is the same with that of the 9th and 8th but differs from that of the 7th iteration, then $s = 3$ and otherwise $s = 1$. Three stopping criteria are included: (1) maximum running time ($maxTime$), (2) maximum number of total iterations (Num), and (3) the maximum times of the reproducible results appear consecutively ($sNum$).

Algorithm 4 LDR-AS

```
1: $i \leftarrow 0$; $s \leftarrow 0$; start counting running time
2: $\pi_0 \leftarrow$ Use LDR to obtain the initial solution
3: **while** iteration number $i < Num$ and running time $< maxTime$ and
   $s < sNum$ **do**
4:     $\pi_1 \leftarrow$ Intensification($\pi_0$)
5:     Add $\pi_1$ to $List$; update $s$
6:     $\pi_0 \leftarrow$ Diversification($\pi_1$)
7:     $i \leftarrow i + 1$
8: **end while**
9: **return** $\pi^* = \arg\min_{\pi' \in List} Obj(\pi')$
```

We first discuss the `Intensification` as described in Algorithm 5. The `Intensification` starts with an initial solution π and then it proceeds with applying uniformly and randomly chosen local search operators from set `insert`, `swap`, `insert-pair`, `swap-pair` to π until the termination criteria satisfy. The newly generated schedule π' is further examined in correspondence of its performance $Obj(\pi')$. Performance of a new schedule after applying local search operators is classified into three categories: good, moderate and poor. We define the good category to be the solution whose improvement is larger than a predefined threshold δ, the moderate category to be the schedule whose improvement is smaller than δ but larger than 0, and the others with negative improvement as the poor category. Only good category schedules are accepted in the algorithm. Parameters used in the algorithm are given as follows: δ is the threshold to decide whether a new schedule can be accepted, $mNum$ is the maximum times of moderate improvement, $pNum$ is the maximum times of poor improvement and r is the range within which the local operators work. Let π^* be the best schedule and π_0 record the best schedule that is not accepted so far in current iteration. p and m are the counter of the moderate and poor improvement, respectively.

Algorithm 5 `Intensification`(π)

```
1: Initialize $\pi^* \leftarrow \pi$; $\pi_0 \leftarrow \pi$; $p = 0$; $m = 0$
2: while $p < pNum$ do
3:   operator← select an operator uniformly and randomly from operator set {insert, swap, insert-pair, swap-pair}
4:   $\pi' \leftarrow$ apply operator($\pi^*, r$) to $\pi^*$
5:   if $Obj(\pi^*) - Obj(\pi') \geq \delta$ then
6:     $\pi^* \leftarrow \pi'$
7:     $m \leftarrow 0$; $p \leftarrow 0$
8:   else if $0 \leq Obj(\pi^*) - Obj(\pi') < \delta$ then
9:     $m \leftarrow m + 1$; $p \leftarrow 0$
10:    if $Obj(\pi') < Obj(\pi_0)$ then
11:      $\pi_0 \leftarrow \pi'$
12:    end if
13:  else
14:    $p \leftarrow p + 1$
15:  end if
16:  if $m >= mNum$ then
17:    $\delta = \max\{1, Obj(\pi^*) - Obj(\pi_0)\}$
18:    $\pi^* \leftarrow \pi_0$; $m \leftarrow 0$
19:  end if
20: end while
21: return $\pi^*$
```

The key factor of the `Intensification` to categorize the schedule performance lies on the determination of the acceptance threshold δ. In the LDR-AS method, we propose an adaptive update mechanism to refresh δ to appropriate value timely. We use π_0 to store the best moderate (not accepted) schedule when the cumulative times of moderate improvements m reaches $mNum$. Then δ is set to be the difference between the current best accepted schedule π^* and the best moderate schedule π_0, i.e., $\delta = \max\{1, Obj(\pi^*) - Obj(\pi_0)\}$. This ensures even if the search continues to find moderate improvement for accumulative $mNum$ times, the current best moderate schedule π_0 will be used as π^*. Note that the minimum value of δ is 1. Intuitively, if the search algorithm continues achieving unacceptable moderate improvement repeatedly, this implicates the threshold δ is too large and needs to be decreased down. In contrast, when the number of poor improvement p exceeds its maximum $pNum$, it basically means there is no further potential improvement. In this case, it is not necessary to update the value of δ and the algorithm should be terminated. We tailor the local search operators by limiting its operators to work within range $[\max\{k-r,0\}, \min\{k+r,n\}]$ where k is the position in the sequence. The new local search operators in LDR-AS are given as follows.

`insert`(π, r): randomly and uniformly select a customer order at position k_1 in π, and an insertion position $k_2 \in [\max\{k_1 - r, 0\}, \min\{k_1+r, n\}]$ where $k_2 \neq k_1$. If $k_1 > k_2$, move the orders at positions $k_2 \leq k' < k_1$ to positions $k'+1$. Otherwise, move the orders at positions $k_1 < k' \leq k_2$ to positions $k'-1$. Insert order at position k_1 to position k_2 to obtain π'.

`swap`(π, r): randomly and uniformly select a customer order at position k_1 in π, and a swap position $k_2 \in [\max\{k_1 - r, 0\}, \min\{k_1 + r, n\}]$ where $k_2 \neq k_1$. Swap the orders at position k_1 and k_2 to obtain π'.

`insert-pair`(π, r): randomly and uniformly select a customer order at position k_1 in π, and an insertion position $k_2 \in [\max\{0, k_1 - r\}, \min\{k_1 + r, n\}]$ where $|k_2 - k_1| \geq 2$. If $k_1 > k_2$, move the orders at positions $k_2 \leq k' < k_1$ to positions $k' + 2$. Otherwise, move the orders at positions $k_1 + 1 < k' \leq k_2 + 1$ to positions $k' - 2$. Move orders at $k_1, k_1 + 1$ to $k_2, k_2 + 1$ to obtain π'.

`swap-pair`(π, r): randomly and uniformly select a customer order at position k_1 in π, and an insertion position $k_2 \in [\max\{0, k_1 - r\}, \min\{k_2 + r, n\}]$ where $|k_2 - k_1| \geq 2$. Swap orders at $k_1, k_1 + 1$ with those at positions $k_2, k_2 + 1$ to obtain π'.

Algorithm 6 `Diversification`(π)

```
1: for i=0 to $dNum$ do
2:   operator← select an operator uniformly and randomly from set {insert, swap, insert-pair, swap-pair}
3:   $\pi' \leftarrow$ apply operator($\pi, r$) to schedule $\pi$
4:   $\pi \leftarrow \pi'$
5: end for
6: return $\pi$
```

Schedule obtained from the `Intensification` is highly possible to be trapped within local optima, we thus force to involve random disturbance to get rid of the local optima basin through the `Diversification` as presented in Algorithm 6. Local search operators are randomly and uniformly selected to diversify the schedule for $dNum$ times, and the newly generated schedule π' is always accepted regardless of its performance.

6. Computational experiments

We conduct computational experiments to test the performance of the proposed algorithms and state-of-the-art methods for problem COS in the literature. A total of seven well-known dispatching rules, the MILP model, the efficient PSA method (Riahi et al., 2019), a PSO (Wu et al., 2018) and a GA (Xiong et al., 2014) are used for the comparison with the proposed LDR and LDR-AS. Table 6 briefly summarizes the selected seven well-known dispatching rules.

All runs are made on a 64-bit Windows 10 platform with Intel Core i7-8750H 2.20 GHz CPU and 16 GB RAM. The proposed FEGP is implemented based on the ECJ library,[3] a Java-based evolutionary computation system. The PSO, LDR and LDR-AS algorithms are coded in Java as well. The executable program file of PSA algorithm is obtained from the authors of Riahi et al. (2019). The MILP model is solved using CPLEX 12.8 with default configurations except that the time limit is set to be 3600 s and the “nodefile” configuration is turned on to avoid running out of memory. The source code and executable programs of LDR-I, LDR-II, the dataset and the best found solutions by our proposed methods are available online.[4]

6.1. Testing instances

We use three classes of testing instances to validate the effectiveness of the proposed methods in this paper. Among them, Test-1 and Test-2 are the same with that in Riahi et al. (2019) and originally generated by Framinan and Perez-Gonzalez (2017). Test-3 is a new dataset randomly generated following the processing generation rule in Chen and Li (2020). In each class, a total of 360 instances are generated and the generation rules are given in details below.

[3] Version 23, available from https://cs.gmu.edu/~eclab/projects/ecj/.
[4] https://github.com/hma19/customer-order-scheduling-LDR.

Table 6
Well-known dispatching rules for customer order scheduling

STPT	Arrange orders by non-decreasing sequence of the total job processing time, i.e., $\sum_{t=1}^{m} p_j^t$.
SMPT	Arrange the orders by the non-decreasing sequence of its maximum job processing time, i.e., $\max_t p_j^t$.
SMCT	First arrange jobs on each machine by non-decreasing order of p_j^t and compute the completion time C_j^t for each order j. Then schedule the orders in non-decreasing sequence of C_j^t.
SPTL	First find the machine t' with the smallest workload in the current partial schedule, then select the order with smallest $p_j^{t'}$ to schedule.
ECT	Select the order with the earliest completion time to schedule, i.e., $j^* = \arg\min_j (C_j - C_k)$ wherein k is the order immediately before j.
TWPT	Arrange orders by non-decreasing sequence of total weighted job processing time of the orders $\sum_{t=1}^{m} w_t p_j^t$, herein $w_t = \sum_{j=1}^{n} p_j^t / (\sum_{t=1}^{m} \sum_{j=1}^{n} p_j^t)$.
NEW	Design a look-ahead indicator $\psi_l = C(S_l) + C^*(S_l, R_l)/(n-k+1)$ including the contribution of both the scheduled orders S_l and unscheduled orders R_l. The order with the smallest ψ_l is selected to schedule.

STPT, SMPT and SMCT from Wang and Cheng (2003), ECT and SPTL from Leung et al. (2005a), TWPT from Riahi et al. (2019) and NEW from Framinan and Perez-Gonzalez (2017).

- Test-1: a total of 360 instances are divided into 12 groups with 30 instances in each. Each group is a combination of $n \in \{50, 100, 200\}$ and $m \in \{2, 5, 10, 20\}$. For the job in order j on machine t, its processing time p_j^t is randomly generated from a uniform distribution $[1, 100)$.
- Test-2: a total of 360 instances remain the same configurations with Test-1 except for the generation of job processing time. For each customer, the number of jobs included in this order, l, is first randomly selected from a uniform distribution [1, m]. l jobs are then randomly selected from the job set and then the processing time p_j^t is generated from uniform distribution $[1, 100)$.
- Test-3 (newly generated): a total of 360 instances remain the same configurations with Test-1 except for the generation of job processing time. For a specific *n-m*, 30 instances are generated with job processing time p_j^t randomly generated from uniform distribution $[1, l_{max}]$, where $l_{max} \in \{20, 50, 100\}$. For each $l_{max} \in \{20, 50, 100\}$, 10 instances are generated.

We use the metric, *average relative percentage deviation* (ARPD), to compare the computational performance of different methods on the given instances in each group. The ARPD is defined as follows.

$$ARPD(Alg) := \frac{1}{T} \sum_{I_t \in \{I_1, I_2, \ldots, I_T\}} \frac{Obj(Alg, I_t) - Obj_{ref}(I_t)}{Obj_{ref}(I_t)} \times 100,$$

where $Obj(Alg, I_t)$ denotes the objective value of algorithm Alg for solving instance I_t, and $Obj_{ref}(I_t)$ denotes the best objective value among all the algorithms for solving instance I_t. $T = 30$ is the total number of instances in each group. It is simple to check that smaller ARPD indicates better performance for a given algorithm. Particularly, if the ARPD of an algorithm is equal to zero, that means this algorithm outperforms all the others for each instance in the given group.

6.2. Comparison results of LDRs and well-known dispatching rules

Tables 7–9 present the ARPD comparison of LDR-I, LDR-II and seven well-known dispatching rules on Test-1, Test-2 and Test-3, respectively. In these tables, the entries of each column shows the ARPD value of the corresponding method over 30 instances for the combination *n-m*.

In Table 7, LDR-II outperforms all the other rules in all the combinations of *n-m*. In particular, LDR-II obtains an ARPD of zero in rows 100-2, 200-2, 200-5 and 200-10, and generates the best solution for the 30 instances in combinations of 200-2 and 200-10. At row 50-20, LDR-II obtains the largest ARPD 0.23, but is still significantly smaller than that of the others. This indicates LDR-II is able to better handle more complex and large-scale problem instances than the other rules. In terms of LDR-I, it performs slightly worse than LDR-II but still significantly better than all the well-known dispatching rules for all *n-m* combinations. In Table 8, LDR-II obtains the best ARPD on 9 rows while LDR-I obtains the best ARPD on 5 rows. LDR-I generates the best solution for all the 30 instances in combination 200-5, having an ARPD of zero. This shows LDR-I can perform very well even with much simpler algorithm structure comparing with LDR-II. The superiority of both LDR-I and LDR-II is more obvious when comparing with well-known dispatching rules. For instance, SPTL obtains the smallest ARPD 12.71 at row 50-2 but the largest ARPD of 30.50 at row 200-20 while LDR-II obtains 0.06 and 0.03 on corresponding rows. Notably, NEW performs the best among the seven selected well-known dispatching rules, and followed by ECT. In terms of the solution quality, LDRs outperform the existing dispatching rules for all the combinations in Test-1 and Test-2. It should be noted that the maximum solution times for each instance by LDR-I and LDR-II are 0.07 and 0.1 s respectively, much faster than 0.6 s by NEW. All the other testing dispatching rules take less than 0.01 s to solve each instance.

In Table 9, LDR-II achieves the best ARPD on 10 rows while LDR-I achieves the best ARPD on 2 rows for Test-3. As LDR-I and LDR-II are learned by FEGP using randomly generated data following generation rule of Test-1 and Test-2, this indicates LDRs have robust performance regardless of the data structure information. Fig. 8 graphically shows the ARPD comparison of LDR-I, LDR-II and NEW according to the results in Tables 7–9. Obviously, LDR achieves much better solutions than the dispatching rule NEW in all combinations of problem instance sizes.

Table 10 summarizes the number of best found solutions by LDR and the existing well-known dispatching rules. As shown in this table, LDR-I and LDR-II obtain the best solution for 275 and 767 instances, respectively. In total, LDR-I and LDR-II altogether obtain the best solution for 1037 instances compared with 41 by NEW among the total 1080 instances. There are only two instances that ECT achieves the best solution. All the other dispatching rules are not capable of finding the best solution for any of the instances.

6.3. Comparison results of LDR-AS and state-of-the-art methods

We further compare the proposed two-stage method LDR-AS with PSA, MILP (solving the MILP formulation using CPLEX) and two representative metaheuristics (PSO and GA) on Test-1, Test-2 and Test-3. For PSO and GA, LDR-I, LDR-II and NEW are used to create initial solutions, respectively.

We perform the executable program of PSA 5 times on each instance. The maximum iteration number `IterMax` in the PSA method is set to be 2000. The smallest objective value of these 5 runs together with the published records in Riahi et al. (2019) is selected as the best performance of PSA method. Accordingly, we run LDR-AS method 5 times on each instance and the smallest among the 5 runs is chosen as the best performance of the LDR-AS method. Based on a number of preliminary testing with consideration of convergence speed and solution quality, we set the parameters of the LDR-AS method as follows: $Num = 400$, $sNum = 5$, initial $\delta = 20$, $mNum = 1500$, $pNum = 5000$, $dNum = 2$, $r = 0.1 \times n$ where n is the number of the orders in the problem instance. In addition, we adopt the well-tuned PSO method developed in Wu et al. (2018) wherein pairwise swap operator is used to improve the solution in every PSO iteration. The same parameters in that paper are also adopted as following: $B_1 = 0.5$, $B_2 = 0.5$ and $w = 0.5$. We set the number of particles $N = 2 \times n$, number of PSO iterations $ITRN = 300$ and number of local search improvement $NN = 20$ after a number of testing experiments considering both the problem size and algorithm convergency. Regarding to the GA, we adopt the hybrid GA with a variable neighborhood local search in Xiong et al. (2014). We use the

Table 7
ARPD comparison of LDR with well-known dispatching rules on Test-1.

n-m	STPT	SMPT	SMCT	SPTL	ECT	TWPT	NEW	LDR-I	LDR-II
50-2	3.60	6.03	5.08	4.10	1.48	2.83	0.28	0.09	**0.01**
50-5	6.50	10.97	8.93	5.39	1.53	5.52	0.70	0.21	**0.04**
50-10	8.04	12.71	9.11	5.22	1.26	6.99	0.85	0.21	**0.12**
50-20	7.82	12.17	9.18	4.97	1.03	7.25	0.58	0.28	**0.23**
100-2	3.27	6.40	5.59	4.35	2.42	2.71	0.29	0.05	**0.00**
100-5	6.31	11.21	8.94	5.64	1.53	5.60	0.80	0.14	**0.01**
100-10	7.03	11.33	9.36	5.62	1.36	6.28	0.87	0.17	**0.01**
100-20	7.42	11.21	8.78	4.98	0.95	6.93	0.62	0.10	**0.05**
200-2	2.70	6.26	5.32	4.18	2.90	2.26	0.28	0.05	**0.00**
200-5	4.92	10.64	9.08	6.07	1.91	4.44	1.04	0.15	**0.00**
200-10	5.35	10.96	9.20	6.06	1.51	4.95	0.93	0.20	**0.00**
200-20	5.77	10.98	8.77	5.21	1.19	5.43	0.75	0.12	**0.02**

Table 8
ARPD comparison of LDR with well-known dispatching rules on Test-2.

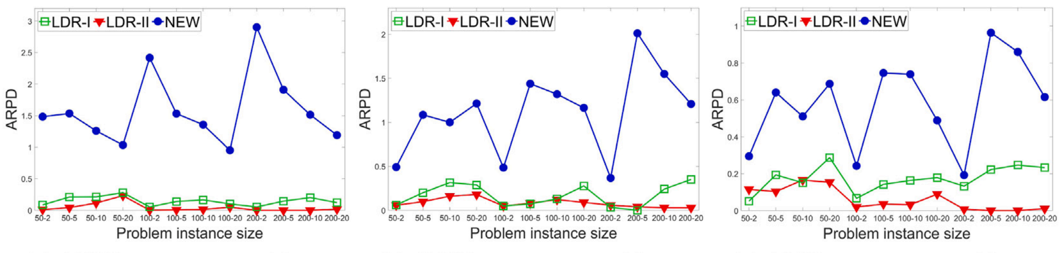
n-m	STPT	SMPT	SMCT	SPTL	ECT	TWPT	NEW	LDR-I	LDR-II
50-2	7.02	9.90	6.47	12.71	2.20	5.04	0.54	**0.06**	**0.06**
50-5	12.10	20.96	14.88	19.84	2.78	9.69	1.16	0.20	**0.10**
50-10	15.15	25.89	19.03	23.19	2.09	13.42	1.13	0.31	**0.16**
50-20	14.70	28.25	19.81	22.67	1.72	13.22	1.36	0.29	**0.18**
100-2	5.96	9.71	6.81	13.29	2.80	4.76	0.53	**0.05**	**0.05**
100-5	10.55	20.99	15.60	23.16	3.43	8.58	1.52	**0.07**	0.08
100-10	13.23	26.29	20.21	23.54	2.54	11.94	1.44	0.13	**0.12**
100-20	13.54	28.61	21.83	27.22	1.73	12.67	1.26	0.28	**0.09**
200-2	3.79	8.61	6.76	13.67	3.73	2.74	0.40	**0.03**	0.06
200-5	9.91	20.50	16.18	23.32	4.19	8.19	2.05	**0.00**	0.04
200-10	11.20	25.82	22.04	29.09	2.77	10.24	1.58	0.24	**0.03**
200-20	11.48	29.85	24.22	30.50	2.02	10.74	1.24	0.35	**0.03**

Table 9
ARPD comparison of LDR with well-known dispatching rules on Test-3.

n-m	STPT	SMPT	SMCT	SPTL	ECT	TWPT	NEW	LDR-I	LDR-II
50-2	3.39	6.30	5.35	4.02	1.75	2.91	0.29	**0.05**	0.12
50-5	6.40	11.39	8.87	5.84	1.50	5.55	0.64	0.19	**0.10**
50-10	7.29	11.09	8.77	5.37	1.22	6.70	0.51	**0.15**	0.17
50-20	7.46	10.15	8.08	5.10	1.02	6.99	0.69	0.29	**0.15**
100-2	2.64	5.96	5.04	4.15	2.39	2.37	0.24	0.07	**0.02**
100-5	5.59	10.86	8.47	5.52	1.50	4.93	0.75	0.14	**0.04**
100-10	6.88	11.32	9.00	5.41	1.23	6.28	0.74	0.16	**0.03**
100-20	6.56	11.23	8.68	5.02	0.98	6.14	0.49	0.18	**0.09**
200-2	2.89	5.79	–5.56	4.08	2.69	2.38	0.19	0.13	**0.01**
200-5	5.11	10.31	8.67	5.90	1.83	4.62	0.96	0.22	**0.00**
200-10	5.40	10.72	8.99	5.81	1.36	4.99	0.86	0.25	**0.00**
200-20	5.74	10.27	8.23	4.97	1.07	5.38	0.62	0.23	**0.01**

Table 10
Number of best found solutions by LDR and the selected dispatching rules.

n-m	STPT	SMPT	SMCT	SPTL	ECT	TWPT	NEW	LDR-I	LDR-II	LDR
Test-1	0	0	0	0	0	0	9	66	**287**	**351**
Test-2	0	0	0	0	0	0	7	132	**223**	**353**
Test-3	0	0	0	0	2	0	25	77	**257**	**333**
Total	0	0	0	0	2	0	41	275	**767**	**1037**



(a) ARPD comparison on Test-1 (b) ARPD comparison on Test-2 (c) ARPD comparison on Test-3

Fig. 8. Comparison of LDRs and NEW.

Table 11
Comparison of MILP, PSO, GA, PSA and LDR-AS on Test-1.

n-m	MILP		NEW-PSO		LDR-I-PSO		LDR-II-PSO		NEW-GA		LDR-I-GA		LDR-II-GA		PSA		LDR-I-AS		LDR-II-AS	
	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)
50-2	0.045	3600	0.001	2.06	0.001	2.18	**0.000**	2.30	0.001	6.99	0.001	4.73	0.001	4.63	**0.000**	3.18	**0.000**	1.07	**0.000**	**1.00**
50-5	1.169	3600	0.054	5.97	0.031	7.34	0.029	6.45	0.033	13.67	0.027	9.98	0.025	9.78	0.002	6.47	0.001	**5.72**	**0.000**	5.87
50-10	3.675	3600	0.157	10.54	0.081	12.33	0.127	13.02	0.071	22.29	0.067	18.24	0.070	18.05	0.007	11.61	0.002	**9.38**	**0.001**	9.80
50-20	3.529	3600	0.263	20.66	0.152	26.14	0.156	26.87	0.111	46.84	0.101	33.32	0.129	32.79	0.024	21.26	0.008	17.31	**0.004**	**17.20**
100-2	0.294	3600	0.005	17.89	0.005	15.46	0.006	16.05	0.006	30.97	0.006	24.46	0.005	23.94	**0.000**	20.92	0.001	**15.30**	**0.000**	15.46
100-5	3.145	3600	0.135	50.32	0.098	36.25	0.094	32.60	0.153	55.58	0.112	46.38	0.100	45.35	0.006	44.02	0.007	25.62	**0.005**	**25.49**
100-10	7.446	3600	0.419	98.53	0.226	74.41	0.222	67.87	0.342	105.73	0.224	89.93	0.215	89.01	0.012	78.78	0.008	**45.96**	**0.005**	46.95
100-20	8.391	3600	0.775	154.38	0.351	149.00	0.384	151.85	0.553	204.44	0.320	171.31	0.335	169.47	0.057	145.91	0.011	**90.77**	**0.005**	95.65
200-2	1.353	3600	0.010	173.89	0.008	107.47	0.005	108.94	0.014	122.05	0.010	113.65	0.007	111.53	0.001	149.87	0.002	47.76	**0.000**	**46.73**
200-5	11.172	3600	0.212	283.25	0.112	298.67	0.086	276.13	0.370	253.17	0.135	234.80	0.100	231.92	0.016	322.95	**0.002**	93.51	0.004	**92.77**
200-10	17.355	3600	0.752	574.41	0.226	544.06	0.195	523.64	0.849	476.93	0.241	415.03	0.211	410.24	0.023	576.25	0.005	**162.17**	**0.004**	168.36
200-20	14.564	3600	1.110	603.20	0.367	605.54	0.415	605.27	1.139	605.10	0.403	606.37	0.429	603.76	0.029	600.00	0.009	**324.03**	**0.003**	332.94

Table 12
Comparison of MILP, PSO, GA, PSA and LDR-AS on Test-2.

n-m	MILP		NEW-PSO		LDR-I-PSO		LDR-II-PSO		NEW-GA		LDR-I-GA		LDR-II-GA		PSA		LDR-I-AS		LDR-II-AS	
	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)
50-2	0.039	3600	0.004	1.83	**0.000**	2.00	**0.000**	2.29	**0.000**	7.40	**0.000**	5.58	**0.000**	5.44	**0.000**	3.17	**0.000**	0.64	**0.000**	**0.52**
50-5	2.062	3600	0.046	5.72	0.029	5.58	0.033	6.27	0.026	14.03	0.023	10.39	0.018	9.95	0.001	6.44	**0.000**	**5.43**	**0.000**	**5.43**
50-10	4.333	3600	0.119	8.74	0.057	10.54	0.072	10.92	0.052	23.30	0.043	17.67	0.045	17.08	0.003	11.42	0.001	**8.17**	**0.000**	8.31
50-20	5.970	3600	0.173	15.52	0.054	20.74	0.070	21.35	0.052	47.99	0.033	34.69	0.040	33.47	0.009	21.23	0.001	16.13	**0.000**	**14.85**
100-2	0.495	3600	0.004	13.80	0.004	18.54	0.003	19.42	0.004	32.21	0.004	24.47	0.004	23.97	**0.000**	20.60	**0.000**	14.33	**0.000**	**14.18**
100-5	3.848	3600	0.177	40.78	0.132	31.04	0.175	35.46	0.196	57.49	0.135	45.77	0.161	44.22	0.005	43.55	0.012	26.98	**0.004**	**25.91**
100-10	8.879	3600	0.441	71.09	0.256	59.95	0.311	65.62	0.356	109.08	0.230	87.75	0.277	86.52	0.023	77.61	0.016	51.60	**0.001**	**47.31**
100-20	29.278	3600	0.658	139.23	0.373	120.71	0.395	130.28	0.454	209.34	0.296	215.83	0.346	211.97	0.041	144.23	0.017	**93.57**	**0.002**	94.62
200-2	1.921	3600	0.007	128.91	0.006	89.63	0.006	109.19	0.008	127.97	0.007	128.76	0.007	123.70	0.001	150.33	**0.000**	**43.87**	**0.000**	46.15
200-5	40.524	3600	0.282	348.49	0.161	313.87	0.191	342.70	0.418	267.03	0.191	272.12	0.227	264.55	0.016	323.39	0.009	**93.60**	**0.002**	94.66
200-10	61.882	3600	0.830	578.14	0.286	547.18	0.433	515.68	0.940	488.49	0.339	478.73	0.494	469.05	0.031	561.37	0.011	**160.69**	**0.002**	166.66
200-20	58.994	3600	1.290	605.55	0.516	605.70	0.622	604.66	1.415	608.27	0.530	606.39	0.672	604.47	0.053	600.00	0.017	**318.79**	**0.004**	329.86

Table 13
Comparison of PSO, GA, PSA and LDR-AS on Test-3.

n-m	NEW-PSO		LDR-I-PSO		LDR-II-PSO		NEW-GA		LDR-I-GA		LDR-II-GA		PSA		LDR-I-AS		LDR-II-AS	
	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)	ARPD	T (s)
50-2	0.006	2.02	0.002	2.12	0.001	2.20	**0.000**	6.41	**0.000**	4.73	**0.000**	4.64	**0.000**	2.57	0.001	**0.82**	**0.000**	0.90
50-5	0.054	5.67	0.026	6.58	0.029	6.90	0.037	11.79	0.030	9.89	0.033	9.74	0.025	**5.21**	0.002	5.90	**0.001**	5.77
50-10	0.207	10.13	0.104	12.32	0.097	12.19	0.073	20.49	0.059	21.23	0.066	20.82	0.055	9.64	0.008	9.12	**0.004**	**8.75**
50-20	0.319	21.95	0.153	26.40	0.150	26.48	0.126	38.94	0.109	40.65	0.094	40.10	0.065	17.10	0.010	**15.87**	**0.005**	15.98
100-2	0.007	14.97	0.007	15.36	0.006	15.39	0.008	28.20	0.006	26.10	0.005	25.68	0.003	16.58	0.001	**14.07**	**0.000**	14.35
100-5	0.140	45.96	0.081	33.31	0.084	30.75	0.147	51.90	0.089	50.93	0.084	50.19	0.045	34.66	0.007	**26.20**	**0.001**	26.67
100-10	0.427	87.65	0.192	66.76	0.186	67.93	0.350	96.03	0.187	99.77	0.181	99.25	0.059	62.25	0.012	**47.92**	**0.003**	48.55
100-20	0.734	149.80	0.345	138.85	0.340	144.64	0.570	174.92	0.297	189.15	0.298	187.92	0.087	115.27	0.015	**94.80**	**0.004**	96.73
200-2	0.011	138.30	0.009	103.88	0.009	99.78	0.015	120.31	0.011	121.37	0.011	119.31	0.006	118.72	0.002	41.50	**0.001**	**41.10**
200-5	0.244	257.25	0.093	277.64	0.074	273.31	0.379	240.68	0.113	252.82	0.089	250.22	0.052	256.07	0.003	**92.91**	**0.002**	92.93
200-10	0.764	569.67	0.205	523.36	0.178	532.88	0.860	418.42	0.224	456.47	0.208	450.03	0.065	453.91	0.006	**169.82**	**0.003**	170.53
200-20	1.053	604.34	0.335	605.68	0.364	605.16	1.063	607.70	0.359	607.00	0.388	604.62	0.055	600.00	0.007	339.38	**0.004**	**339.07**

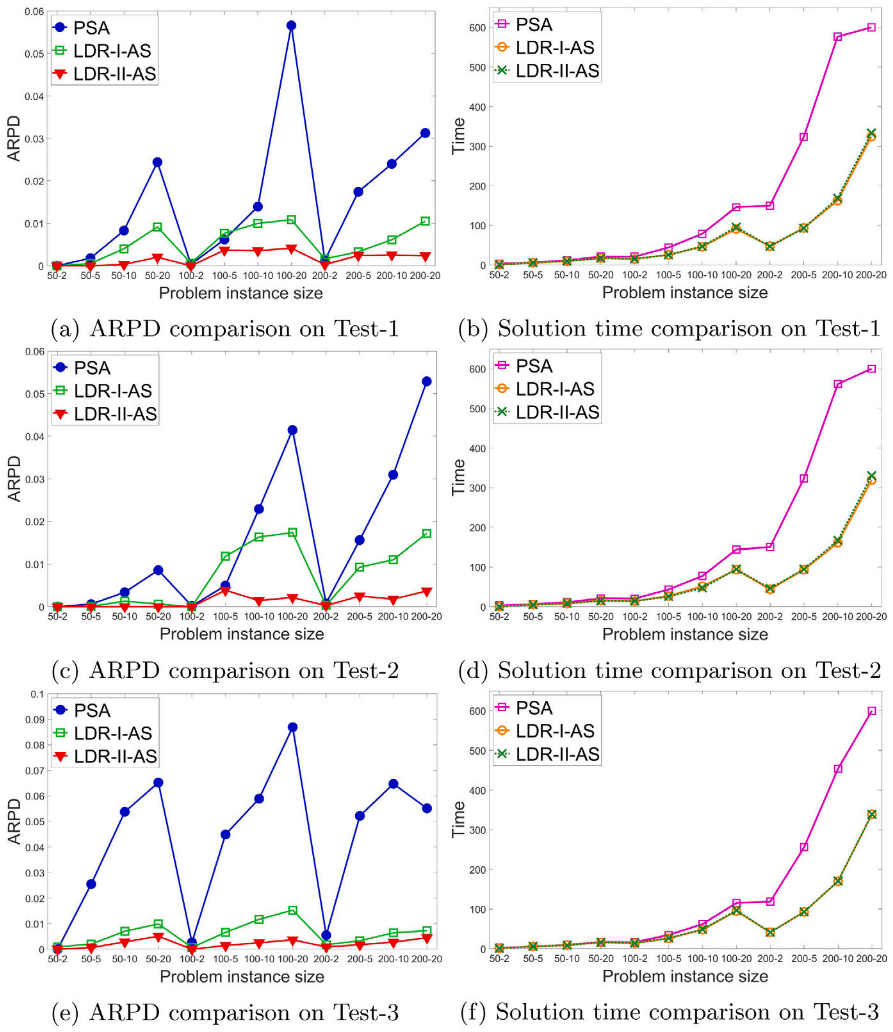


Fig. 9. Comparison of PSA and LDR-AS on Test-1, Test-2 and Test-3.

well tuned parameters in the original paper as following: local search probability $PM = 0.1$, mutation probability $P_m = 0.1$ and crossover probability $P_c = 0.8$. Considering both the problem size and comparable time with other algorithms, we set the population size $PS = 2 \times n$ and the iteration for local search $Inloop_{max} = 500$. PSA, LDR-AS, PSO and GA are executed with a time limit $maxTime = 600$ s. It should be noted that the PSA executable file does not support setting the running time limit with the input parameters, we realize this through controlling the running time of the executable file using Bash script.

Tables 11–13 present the comparison results of MILP, PSA, PSO, GA and LDR-AS on Test-1, Test-2 and Test-3, respectively. In addition, we use LDR-I, LDR-II and NEW to create initial solutions for PSO and GA. Accordingly, we have methods NEW-PSO, LDR-I-PSO, LDR-II-PSO, NEW-GA, LDR-I-GA, LDR-II-GA, LDR-I-AS and LDR-II-AS. For the corresponding method in those tables, column “ARPD” indicates the ARPD value and column “T (s)” is the average solution time (in seconds) over the whole 30 instances in each combination.

As shown in Table 11, the proposed LDR-AS method obtains the best solutions among all the methods. Specifically, LDR-II-AS obtains the best ARPD on 11 rows, both LDR-I-AS and PSA obtain the best on 2 rows whereas LDR-II-PSO obtains the best ARPD on only 1 row. LDR-II-AS outperforms PSA in both solution quality and solution time. Comparing with LDR-II-AS, LDR-I-AS performs slightly worse. Only at combination 200-5, LDR-I-AS outperforms LDR-II-AS. This indicates with the same adaptive search method in the second stage, the good quality of initial solution is critical to improve the overall performance. Nevertheless, LDR-I-AS still outperforms PSA, achieving the best ARPD on 9 rows comparing with PSA. The performance improvement brought by LDR can also be seen by comparing NEW-PSO with LDR-I-PSO and LDR-II-PSO, and also comparing NEW-GA with LDR-I-GA and LDR-II-GA. Specifically, LDR-I-PSO and LDR-II-PSO obtain the best ARPD on 12 and 11 rows respectively comparing with NEW-PSO. Both the LDR-I-GA and LDR-II-GA outperform the NEW-GA on 10 rows. The GA performs quite similarly with PSO on ARPD. On rows 50-2, LDR-I-AS, LDR-II-AS, PSA and LDR-II-PSO obtain the same ARPD value of zero. This is reasonable for less machines result in less computational challenge, and thus all the four methods are capable of finding similar near-optimal solutions. While for the instances with more orders and machines, the ARPD values of PSA are much larger than that of LDR-AS. For instance, the ARPD of PSA at 100-20 is 0.057 while that of LDR-I-AS and LDR-II-AS are 0.011 and 0.005, respectively. As almost all the best solutions are generated by LDR-AS, this indicates LDR-AS exhibits better computational performance on solving large-scale instances.

In terms of the solution time, either LDR-I-AS or LDR-II-AS is the fastest comparing with other methods in Table 11. Also, LDR-I-AS and LDR-II-AS take almost the same time for solving each *n*-*m* combination. The time superiority of LDR-AS is more obvious at large instance cases. For example, PSA takes 576.25 s on row 200-10 while LDR-II-AS takes 168.36 s. Also, for row 200-20, the solution time for PSA on all instances exceeds time limit 600 s. There is no obvious relationship of the times of GA and PSO. In general their computational times are comparable. It should be noted that CPLEX cannot find good enough

Table 14
Number of best found solutions by PSA and LDR-AS.

Instances	PSA	LDR-I-AS	LDR-II-AS	LDR-AS
Test-1	122	191	**245**	**327**
Test-2	169	194	**297**	**333**
Test-3	50	187	**285**	**356**
Total	341	572	**827**	**1016**

solutions for all the instances by solving the MILP model within one hour.

The comparison results on Test-2 and Test-3 are presented in Tables 12 and 13. Since solving MILP models is not able to obtain good results in Test-1 and Test-2, we do not present the MILP results in Table 13. The conclusions of comparing LDR-AS with state-of-the-art methods in both Tables 12 and 13 are consistent with that in Table 11. We thus conclude that the proposed two-stage optimization method achieves the better computational performance using less solution time on all the instances.

Fig. 9 shows the comparison of PSA and LDR-AS based on the ARPD value and solution time in Tables 11–13, respectively. As shown in the figure, the LDR-AS method obtains the smallest ARPD over all combinations of the problem instances, which shows that LDR-AS outperforms PSA in terms of the solution quality. The superiority of LDR-AS is more significant when the number of machines increases. This indicates that the proposed LDR-AS is more efficient for solving large-scale instances. Also, LDR-AS uses less solution time compared with PSA.

Table 14 summarizes the number of best solutions obtained by PSA, LDR-I-AS and LDR-II-AS on Test-1, Test-2 and Test-3. As shown in this table, LDR-I-AS has found the best solutions for 572 instances, LDR-II-AS obtains the best solutions for 827 instances compared with 341 by PSA. In total, LDR-I-AS and LDR-II-AS altogether has found the best solutions for 1016 instances among the total of 1080 instances.

7. Conclusions

In this paper, we address the customer order scheduling problem in parallel production environment with multiple products in each order. Each product needs to be produced on a dedicated machine in parallel and the objective is to minimize the total completion time of the customer orders. By exploring the problem structure and extracting the important features from the well-know dispatching rules manually designed in the literature, we propose an artificial intelligence method, denoted by the feature-enhanced genetic programming (FEGP), to achieve the automatic design of dispatching rules for this problem. Through an off-line learning manner, two new dispatching rules are automatically generated by the proposed FEGP method. Utilizing the high-quality initial solutions provided by the new dispatching rules as the first stage, we develop an adaptive local search algorithm for the further solution quality improvement to form the two-stage optimization method for this problem. Based on the large-scale testing on a total of 1080 instances, two learned dispatching rules outperforms all the efficient well-known dispatching rules manually designed for this problem previously. The proposed two-stage optimization method also achieves the best computational performance using less solution time compared with the state-of-the-art methods in the literature and is capable of finding the best solutions for almost all the benchmark instances. Future work should be focused on improving the search efficiency of the proposed FEGP method to obtain better dispatching rules and extending the proposed method to other complex production systems with uncertainty. In addition, we will investigate to apply the proposed FEGP method into practical case studies to further validate its effectiveness on handling practical production problems.

CRediT authorship contribution statement

Zhongshun Shi: Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Validation, Supervision. **Hang Ma:** Methodology, Writing – original draft, Writing – review & editing, Data curation, Programming, Validation. **Meiheng Ren:** Methodology, Writing – original draft, Writing – review & editing, Programming. **Tao Wu:** Methodology, Writing – review & editing, Validation. **Andrew J. Yu:** Writing - review & editing, Validation.

Acknowledgments

The authors sincerely appreciate Dr. Vahid Riahi for sharing the PSA program file and Professor Jose M. Framinan for sharing the benchmark instances.

References

Blocher, J.D., Chhajed, D., 1996. The customer order lead-time problem on parallel machines. Nav. Res. Logist. 43 (5), 629–654.
Branke, J., Hildebrandt, T., Scholz-Reiter, B., 2015a. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. Evol. Comput. 23 (2), 249–277.
Branke, J., Nguyen, S., Pickardt, C.W., Zhang, M., 2015b. Automated design of production scheduling heuristics: A review. IEEE Trans. Evol. Comput. 20 (1), 110–124.
Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics: A survey of the state of the art. J. Oper. Res. Soc. 64 (12), 1695–1724.
Chen, R.-X., Li, S.-S., 2020. Minimizing maximum delivery completion time for order scheduling with rejection. J. Comb. Optim. 1–21.
Cunha Campos, S., Claudio Arroyo, J.E., 2014. NSGA-II with iterated greedy for a bi-objective three-stage assembly flowshop scheduling problem. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. pp. 429–436.
Daoud, H., Li, D., Yoon, S.W., Srihari, K., 2018. Multi-objective optimization of the order scheduling problem in mail-order pharmacy automation systems. Int. J. Adv. Manuf. Technol. 99 (1), 73–83.
De Athayde Prata, B., Rodrigues, C.D., Framinan, J.M., 2020. Customer order scheduling problem to minimize makespan with sequence-dependent setup times. Comput. Ind. Eng. 106962.
Du, W., Zhong, W., Tang, Y., Du, W., Jin, Y., 2018. High-dimensional robust multi-objective optimization for order scheduling: A decision variable classification approach. IEEE Trans. Ind. Inf. 15 (1), 293–304.
Ferreira, C., 2001. Gene expression programming: a new adaptive algorithm for solving problems. arXiv preprint cs/0102027.
Framinan, J.M., Perez-Gonzalez, P., 2017. New approximate algorithms for the customer order scheduling problem with total completion time objective. Comput. Oper. Res. 78, 181–192.
Framinan, J.M., Perez-Gonzalez, P., 2018. Order scheduling with tardiness objective: Improved approximate solutions. European J. Oper. Res. 266 (3), 840–850.
Framinan, J.M., Perez-Gonzalez, P., Fernandez-Viagas, V., 2019. Deterministic assembly scheduling problems: A review and classification of concurrent-type scheduling models and solution procedures. European J. Oper. Res. 273 (2), 401–417.
Gupta, J.N., Ho, J.C., van der Veen, J.A., 1997. Single machine hierarchical scheduling with customer orders and multiple job classes. Ann. Oper. Res. 70, 127–143.
Hazır, O., Günalay, Y., Erel, E., 2008. Customer order scheduling problem: a comparative metaheuristics study. Int. J. Adv. Manuf. Technol. 37 (5–6), 589–598.
Hildebrandt, T., Branke, J., 2015. On using surrogates with genetic programming. Evol. Comput. 23 (3), 343–367.
Hildebrandt, T., Heger, J., Scholz-Reiter, B., 2010. Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. ACM, pp. 257–264.
Julien, F., Magazine, M., 1990. Scheduling customer orders: An alternative production scheduling approach. J. Manuf. Oper. Manage. 3 (3), 177–199.
Jung, S., Woo, Y.-B., Kim, B.S., 2017. Two-stage assembly scheduling problem for processing products with dynamic component-sizes and a setup time. Comput. Ind. Eng. 104, 98–113.
Komaki, G., Kayvanfar, V., 2015. Grey wolf optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. J. Comput. Sci. 8, 109–120.
Koza, 1992. Genetic Programming: On the Programming of Computers By Means of Natural Selection. MIT Press, Boston, MA.
Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. Stat. Comput. 4 (2), 87–112.

Kung, J.-Y., Duan, J., Xu, J., Chung, I., Cheng, S.-R., Wu, C.-C., Lin, W.-C., et al., 2018. Metaheuristics for order scheduling problem with unequal ready times. Discrete Dyn. Nat. Soc. 2018.

Leung, J.Y.-T., Li, H., Pinedo, M., 2005a. Order scheduling in an environment with dedicated resources in parallel. J. Sched. 8 (5), 355–386.

Leung, J.Y.-T., Li, H., Pinedo, M., 2006. Scheduling orders for multiple product types with due date related objectives. European J. Oper. Res. 168 (2), 370–389.

Leung, J.Y.-T., Li, H., Pinedo, M., 2007. Scheduling orders for multiple product types to minimize total weighted completion time. Discrete Appl. Math. 155 (8), 945–970.

Leung, J.Y., Li, H., Pinedo, M., Sriskandarajah, C., 2005b. Open shops with jobs overlap—-revisited. European J. Oper. Res. 163 (2), 569–571.

Lin, W.-C., Xu, J., Bai, D., Chung, I.-H., Liu, S.-C., Wu, C.-C., 2019. Artificial bee colony algorithms for the order scheduling with release dates. Soft Comput. 23 (18), 8677–8688.

Lin, W.-C., Yin, Y., Cheng, S.-R., Cheng, T.E., Wu, C.-H., Wu, C.-C., 2017. Particle swarm optimization and opposite-based particle swarm optimization for two-agent multi-facility customer order scheduling with ready times. Appl. Soft Comput. 52, 877–884.

Lu, S., Pei, J., Liu, X., Qian, X., Mladenovic, N., Pardalos, P.M., 2019. Less is more: variable neighborhood search for integrated production and assembly in smart manufacturing. J. Sched. 1–16.

Meng, T., Pan, Q.-K., Wang, L., 2019. A distributed permutation flowshop scheduling problem with the customer order constraint. Knowl.-Based Syst. 184, 104894.

Nguyen, S., Mei, Y., Zhang, M., 2017. Genetic programming for production scheduling: a survey with a unified framework. Complex Intell. Syst. 3 (1), 41–66.

Nguyen, S., Zhang, M., Tan, K.C., 2016. Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans. Cybern. 47 (9), 2951–2965.

O'Reilly, U.-M., 1995. An Analysis of Genetic Programming. (Ph.D. thesis). Carleton University.

Riahi, V., Newton, M.H., Polash, M., Sattar, A., 2019. Tailoring customer order scheduling search algorithms. Comput. Oper. Res. 108, 155–165.

Roemer, T.A., 2006. A note on the complexity of the concurrent open shop problem. J. Sched. 9 (4), 389–396.

Shi, Z., Gao, S., Du, J., Ma, H., Shi, L., 2019. Automatic design of dispatching rules for real-time optimization of complex production systems. In: 2019 IEEE/SICE International Symposium on System Integration (SII). IEEE, pp. 55–60.

Shi, Z., Huang, Z., Shi, L., 2018. Customer order scheduling on batch processing machines with incompatible job families. Int. J. Prod. Res. 56 (1–2), 795–808.

Shi, Z., Wang, L., Liu, P., Shi, L., 2017. Minimizing completion time for order scheduling: Formulation and heuristic algorithm. IEEE Trans. Autom. Sci. Eng. 14 (4), 1558–1569.

Venditti, L., Pacciarelli, D., Meloni, C., 2010. A tabu search algorithm for scheduling pharmaceutical packaging operations. European J. Oper. Res. 202 (2), 538–546.

Wagneur, E., Sriskandarajah, C., 1993. Openshops with jobs overlap. European J. Oper. Res. 71 (3), 366–378.

Wang, G., Cheng, T.E., 2003. Customer order scheduling to minimize total weighted completion time. In: Proceedings of the First Multidisciplinary Conference on Scheduling Theory and Applications. pp. 409–416.

Wang, G., Cheng, T.E., 2007. Customer order scheduling to minimize total weighted completion time. Omega 35 (5), 623–626.

Wang, B., Guan, Z., Ullah, S., Xu, X., He, Z., 2017. Simultaneous order scheduling and mixed-model sequencing in assemble-to-order production environment: a multi-objective hybrid artificial bee colony algorithm. J. Intell. Manuf. 28 (2), 419–436.

Wang, L., Shi, Z., Shi, L., 2013. A novel quadratic formulation for customer order scheduling problem. In: 2013 IEEE International Conference on Automation Science and Engineering (CASE). IEEE, pp. 576–580.

Wu, C.-C., Liu, S.-C., Lin, T.-Y., Yang, T.-H., Chung, I.-H., Lin, W.-C., 2018. Bicriterion total flowtime and maximum tardiness minimization for an order scheduling problem. Comput. Ind. Eng. 117, 152–163.

Wu, C.-C., Yang, T.-H., Zhang, X., Kang, C.-C., Chung, I.-H., Lin, W.-C., 2019. Using heuristic and iterative greedy algorithms for the total weighted completion time order scheduling with release times. Swarm Evol. Comput. 44, 913–926.

Xiong, F., Xing, K., Wang, F., Lei, H., Han, L., 2014. Minimizing the total completion time in a distributed two stage assembly system with setup times. Comput. Oper. Res. 47, 92–105.

Xu, J., Wu, C.-C., Yin, Y., Zhao, C., Chiou, Y.-T., Lin, W.-C., 2016. An order scheduling problem with position-based learning effect. Comput. Oper. Res. 74, 175–186.

Yang, J., Posner, M.E., 2005. Scheduling parallel machines for the customer order problem. J. Sched. 8 (1), 49–74.

Zhang, Y., Dan, Y., Dan, B., Gao, H., 2019. The order scheduling problem of product-service system with time windows. Comput. Ind. Eng. 133, 253–266.