Artificial Intelligence Review (2025) 58:160
https://doi.org/10.1007/s10462-024-11059-9

Learn to optimise for job shop scheduling: a survey with comparison between genetic programming and reinforcement learning

Meng Xu[1] · Yi Mei[1] · Fangfang Zhang[1] · Mengjie Zhang[1]

Accepted: 30 November 2024

Abstract
Job shop scheduling holds significant importance due to its relevance and impact on various industrial and manufacturing processes. It involves dynamically assigning and sequencing jobs to machines in a flexible production environment, where job characteristics, machine availability, and other factors might change over time. Genetic programming and reinforcement learning have emerged as powerful approaches to automatically learn high-quality scheduling heuristics or directly optimise sequences of specific job-machine pairs to generate efficient schedules in manufacturing. Existing surveys on job shop scheduling typically provide overviews from a singular perspective, focusing solely on genetic programming or reinforcement learning, but overlook the hybridisation and comparison of both approaches. This survey aims to bridge this gap by reviewing recent developments in genetic programming and reinforcement learning approaches for job shop scheduling problems, providing a comparison in terms of the learning principles and characteristics for solving different kinds of job shop scheduling problems. In addition, this survey identifies and discusses current issues and challenges in the field of learning to optimise for job shop scheduling. This comprehensive exploration of genetic programming and reinforcement learning in job shop scheduling provides valuable insights into the learning principles for optimising different job shop scheduling problems. It deepens our understanding of recent developments, suggesting potential research directions for future advancements.

Keywords Hyper-heuristic · Job shop scheduling · Genetic programming · Reinforcement learning

✉ Meng Xu
meng.xu@ecs.vuw.ac.nz

✉ Fangfang Zhang
fangfang.zhang@ecs.vuw.ac.nz

Yi Mei
yi.mei@ecs.vuw.ac.nz

Mengjie Zhang
mengjie.zhang@ecs.vuw.ac.nz

[1] Centre for Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Kelburn Parade, Wellington 6012, New Zealand

Published online: 15 March 2025

1 Introduction

Job shop scheduling (JSS) is a complex combinatorial optimisation problem that plays a critical role in both scholarly research and industrial applications (xiong et al. 2022). For production in manufacturing, JSS involves the allocation of operations of jobs to a set of machines, each with specific processing requirements, considering constraints such as machine availability and precedence relationships between operations (ouelhadj and Petrovic 2009). Efficient and effective JSS directly impacts operational efficiency, resource utilisation, and ultimately, the overall productivity of manufacturing processes. JSS has many applications in practical fields, such as automotive assembly (zhou and Wen 2023), textile manufacturing (wang and Zhang 2023), chemical material processing (xu et al. 2023), and semiconductor manufacturing (gao et al. 2019).

JSS can be categorised into various types based on its characteristics, including static and dynamic, flexible and non-flexible. There are several approaches for solving different kinds of JSS problems, including *exact* approaches such as branch-and-bound (Zhou et al. 1995) and dynamic programming (Chen et al. 1998), *iterative improvement heuristic* approaches such as genetic algorithm (Gonçalves et al. 2005; Mattfeld and Bierwirth 2004), and artificial bee colony optimisation (Chong et al. 2006; Zhang et al. 2013), and *scheduling heuristics* approaches like shortest processing time first (Shanker and Tzen 1985). *Scheduling heuristics* provide real-time scheduling capability, making them widely popular in real-world scheduling applications (Zhang et al. 2023). These approaches assign priorities to operations or machines at decision points based on the system state, with operations and machines being dispatched according to their priorities, selecting the highest-priority operation or machine.

However, manually designing effective scheduling heuristics demands a lot of domain knowledge and is a time-consuming process. *Hyper-heuristic* approaches (Burke et al. 2013), including both heuristic selection and heuristic generation, represent advanced approaches for complex and dynamic JSS. These approaches operate at a high level, automating the process of selecting or generating heuristics to optimise scheduling performance. *Heuristic selection* involves selecting from a predefined set of existing heuristics based on their performance in specific situations. *Heuristic generation* goes beyond using predefined high-level heuristics by creating new high-level heuristics by combining existing low-level ones to adapt to the given problem. Genetic programming (GP) (Koza and Koza 1992) is commonly employed as a heuristic generation method and has found extensive applications in automatically generating scheduling heuristics to solve different kinds of JSS problems (Nguyen et al. 2019; Zhang et al. 2020). Reinforcement learning (RL), especially deep RL, depending on the nature of the JSS problem, is utilised either as a heuristic selection method for selecting heuristics or a policy-driven decision framework for optimising sequences of job-machine pairs (Zhang et al. 2022; Chen et al. 2024; Luo 2020; Gui et al. 2023; Chang et al. 2022; Liu et al. 2022). GP uses the Darwinian natural selection (evolutionary) principles for learning scheduling heuristics through an iterative, population-based, and stochastic evolutionary process Chand et al. (2018). RL learns to optimise by interacting with an environment, receiving feedback in the form of rewards, and adjusting its actions to maximise cumulative rewards over time (Li 2017). The functionality of RL-learned scheduling heuristics/policies can vary. It can be designed to either select a specific job-machine pair at each decision point or choose from a set of predefined heuristics, with the selected heuristic then

making the scheduling decision. RL aims to learn to obtain an optimal schedule for different JSS problems by carefully designing states, actions, and rewards. However, achieving an optimal solution remains challenging, particularly for large-scale or complex JSS problems.

Recent reviews on learning to optimise for JSS can be found in the literature, including (Branke et al. 2015; Nguyen et al. 2017; Zhang et al. 2023; Wang et al. 2021, 2022; Cunha et al. 2020; Puiseau et al. 2022; Kayhan and Yildiz 2023; Wang et al. 2024). These reviews can be categorised into two main streams: those focusing on learning through GP (Branke et al. 2015; Nguyen et al. 2017; Zhang et al. 2023) and those concentrating on learning through RL (Wang et al. 2021, 2022; Cunha et al. 2020; Puiseau et al. 2022; Kayhan and Yildiz 2023; Wang et al. 2024). Figure 1 illustrates the publication trends of GP and RL for scheduling problems over the past couple of decades. The overall trend reveals a consistent increase in the number of publications. GP has been an emerging and dominant technique in this area for the past 20 years, resulting in a much larger number of publications on JSS compared with RL before 2022. RL, on the other hand, started later but has garnered significant attention recently, leading to a substantial volume of research papers on JSS since 2022. Moreover, both GP and RL have shown much less focus on multi-objective JSS compared with general JSS. GP publications have been gradually increasing since 2004, with steady growth peaking in 2022 at 17 publications, followed by slight fluctuations. In contrast, RL publications on multi-objective JSS are sparse in the early years, with minimal activity until 2017. However, reflecting the overall trend of RL, there has been significant growth in recent years, particularly in 2023, when RL publications on multi-objective JSS reached 18.

Concerning reviews on GP for JSS, the discussions in 2016 (Branke et al. 2015) focused on design choices, encompassing learning methods, representations of priority functions, and the evaluation of hyper-heuristic approaches. This study provides a high-level overview of automatic scheduling heuristics designed with basic GP, focusing on limited types of JSS. Moving forward, a unified GP framework was explored for JSS in 2017 (Nguyen et al. 2017), offering insights into how to leverage GP for learning scheduling heuristics, with a foundation in GP basics applied to general JSS scenarios. A more recent and comprehensive review in 2023 (Zhang et al. 2023) encompasses GP and machine learning techniques, pro-

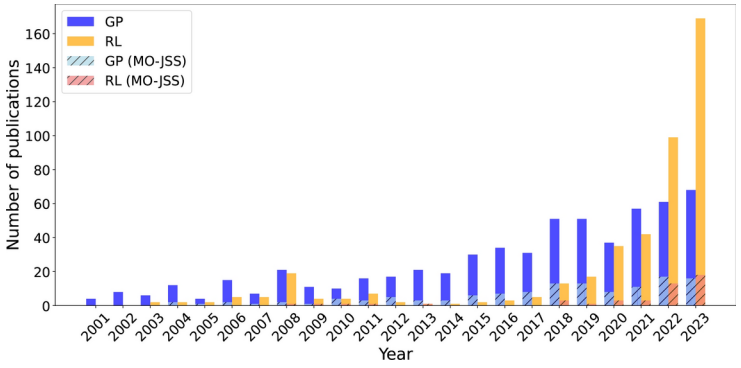


Fig. 1 The number of publications on GP and RL for JSS (retrieved from Scopus in October 2024 using the keywords "genetic programming"/"reinforcement learning" and "job shop scheduling" within the title, abstract, and keywords) is provided. Additionally, the number of publications specifically addressing multi-objective JSS (MO-JSS) for GP and RL (using the same keywords, along with "multi-objective"/"multiobjective") is also included

viding an extensive survey on automatic scheduling heuristic design with different machine learning techniques across various JSS problems.

Regarding reviews on RL for JSS, Cunha et al. (2020) presented a review in 2020, discussing two distinct types of approaches: iterative improvement heuristic and deep RL for JSS. A subsequent review in 2021 (Wang et al. 2021) summarised the designs of state and action of RL-based methods for different scheduling problems, also exploring the fusion modes of RL and meta-heuristics. In 2022, Wang et al. (2022) offered a brief review of RL on JSS, only with limited publications and a focus on general categories of RL in JSS. Another survey in 2022 (Puiseau et al. 2022) addressed questions regarding the reliability of schedules obtained through deep RL-based scheduling approaches. In 2023, Kayhan and Yildiz (2023) examined essential aspects of RL in JSS, identifying frequently investigated problem types, objectives, and constraints, and highlighting limitations and promising areas in the literature. In the same year, another review (Zhang et al. 2023) summarised and analysed the state?of?the?art research in applying deep RL for three optimisation problems in manufacturing, including scheduling, routing, and bin packing. A recent survey (Wang et al. 2024) focuses on the typical design patterns and combinations of common components in RL, including the agent, environment, state, action, and reward.

However, existing reviews on learning scheduling heuristics/policies for JSS tend to focus on either GP or RL, or cover only a limited number of studies, resulting in a lack of comprehensive coverage of the extensive body of work in this domain. A significant gap exists in reviews discussing both GP and RL for JSS, along with an exploration of their strengths and limitations. This survey aims to fill this gap by providing a comprehensive investigation with a comparison between GP and RL (mainly about deep RL) approaches in JSS.

Different from existing surveys, our survey involves discussing the similarities and differences between GP and RL approaches in terms of their learning principles and exploring recent methodologies in both GP and RL for tackling diverse types of JSS problems, thereby highlighting their respective characteristics. Furthermore, we discuss the strengths and limitations of GP and RL for automatic scheduling heuristics/policies learning in the JSS domain. We believe that this survey will capture the attention of scholars and industry professionals engaged in GP and RL, fostering a deeper understanding of challenges in JSS and other combinatorial optimisation problems. Moreover, this survey aims to promote further efforts in hybridising GP and RL, taking advantage of both approaches for effective JSS.

2 Problems and methods

2.1 Job shop scheduling

On the shop floor, there are a set of machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$ and numerous jobs await processing $\mathcal{J} = \{J_1, J_2, ..., J_n\}$. Each job J_i has an arrival time r_i, a due date d_i, and consists of multiple operations $[O_{i,1}, O_{i,2}, ..., O_{i,p_i}]$ that need to be processed in order. Additionally, each job may be assigned a weight w_i, indicating its relative importance. Each operation $O_{i,j}$ has a workload $\pi_{i,j}$ and each machine M_k has a unique processing rate γ_k . The processing time $t_{i,j,k}$ of operation $O_{i,j}$ on machine M_k is defined as $t_{i,j,k} = \pi_{i,j}/\gamma_k$

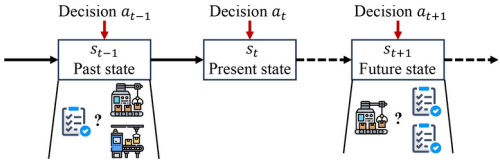


Fig. 2 The sequential decision-making process of a flexible JSS

Table 1 The characteristics of job information and decision requirements in different types of JSS

	Problem	Static JSS	Dynamic JSS	Flexible JSS	Dynamic flexible JSS
Job information	Known	√		√	
	Unknown		√		√
Decision	Sequencing	√	√	√	√
	Routing			√	√

. JSS is a sequential decision-making problem (Wen et al. 2023) wherein scheduling decisions must be made at multiple decision points throughout the scheduling process to achieve the ultimate goal (Huang et al. 2023). Figure 2 illustrates the sequential decision-making process of JSS. As depicted in Fig. 2, at each decision point t, the scheduling state is determined, with the present state s_t becoming active following the decision a_{t-1} made in the past state s_{t-1}. Then, based on the present state s_t, a decision a_t is necessary to advance the scheduling process to a future state s_{t+1} until the entire scheduling process is completed.

Recognising JSS as a sequential decision-making problem makes it straightforward to apply GP, RL, or other learning/optimisation techniques to find optimal or near-optimal scheduling decisions.

During the scheduling process, the following constraints need to be considered.

- An operation is only allowed to be processed after its preceding operations have been completed.
- The operation can only be processed by one of its optional machine(s).
- A machine can only process one operation at a time.
- Preemption is not allowed, meaning a machine cannot switch to process another operation until completing the current operation.JSS can be classified into various categories based on its characteristics. Table 1 outlines the distinctive features of job information and decision requirements across different categories of JSS. Subsequently, this survey offers a detailed description of each category to offer a comprehensive understanding of the challenges inherent in the problem.

2.1.1 Classical static JSS

In static JSS, the job information (e.g., operations, processing time of each operation, due date) is known in advance (Ingimundardottir and Runarsson 2018). Each operation can only be processed by a specified machine and its processing time depends on the machine that processes it. JSS aims to determine which operation will be chosen to process next by an idle machine (Brizuela and Sannomiya 1999).

2.1.2 Dynamic JSS

In reality, the job information is not always available in advance (Sharma and Jain 2015). The system state is dynamically changeable, e.g., jobs arrive dynamically (Zhou et al. 2018; Zhou and Yang 2019; Zhang et al. 2020, 2020), and machines break down during the assignment (Yin et al. 2003). The dynamically changed system state makes the problem more complex and makes it more relevant for practical applications, which is known as the dynamic JSS problem (Karunakaran et al. 2017).

2.1.3 Flexible JSS

In JSS, the operation can only be processed by the fixed single machine. Flexible JSS introduces flexibility in terms of machine assignments, allowing an operation to be processed on multiple machines (Pezzella et al. 2008; Xie et al. 2019). Consequently, in flexible JSS, decisions involve selecting the specific machine to process a ready operation, rather than just choosing an idle machine for processing a ready operation (Gomes et al. 2013). This distinction leads to the consideration of two types of decision points: routing decision point and sequencing decision point, which are shown as follows.

- *Routing decision point*: represents the situation when an operation becomes ready. At a routing decision point, a *routing rule* is required to select a machine for a ready operation.
- *Sequencing decision point*: represents the situation when a machine becomes idle. At the sequencing decision point, a *sequencing rule* is required to choose an operation from the waiting queue to be processed next.

2.1.4 Dynamic flexible JSS

Dynamic flexible JSS (Zhou et al. 2018) considers the flexibility of machines and incorporates the dynamic environments at the same time.

2.2 Methods for job shop scheduling

This section provides a comprehensive investigation of various approaches for solving JSS problems. The main approaches are categorised into three classes, which are exact approaches, (meta-)heuristic approaches, and hyper-heuristic approaches.

2.2.1 Exact approaches

Exact approaches for solving JSS problems involve algorithms and techniques that guarantee the identification of an optimal solution within a reasonable amount of time. These methods are particularly effective for smaller problem instances, where the computational complexity remains manageable. Some key exact methods used in JSS include branch-and-bound (Zhou et al. 1995) and dynamic programming (Chen et al. 1998). *Branch-and-bound* explores the solution space by dividing it into sub-problems and bounding the solution space based on certain criteria (Ashour and Hiremath 1973; Artigues et al. 2009). Branch-and-

bound with time-indexed formulation is a variation of branch-and-bound that formulates the problem in a time-indexed representation (Azem et al. 2007), allowing for a more efficient exploration of the solution space. Some other variations of branch-and-bound methods were proposed and successfully applied to many small-scale scheduling problems (Sarin et al. 1988; Potts and Van Wassenhove 1985). *Dynamic programming* (Chen et al. 1998) is often used for specific cases, such as in the context of parallel machine scheduling (Chen et al. 1998; Gromicho et al. 2012; Ozolins 2020). It breaks down the scheduling problem into smaller sub-problems and utilises optimal solutions to these sub-problems to construct an optimal solution for the overall problem. Although exact methods guarantee optimal solutions, they often become computationally intractable for larger problem instances due to their inherent complexity (Zhang et al. 2019). Additionally, exact methods struggle to adapt to dynamically changing environments.

2.2.2 Meta-heuristic approaches

Meta-Heuristic approaches, unlike exact approaches, do not guarantee to find an optimal solution. Instead, meta-heuristic approaches aim to find good enough solutions. Common *meta-heuristic* approaches for JSS include genetic algorithms (Gonçalves et al. 2005; Mattfeld and Bierwirth 2004), tabu search (Saidi-Mehrabad and Fattahi 2007; Vilcot and Billaut 2011; Fattahi et al. 2018), simulated annealing (Aydin and Fogarty 2004), artificial bee colony optimisation (Chong et al. 2006; Zhang et al. 2013), ant colony optimisation (Heinonen and Pettersson 2007; Huang and Liao 2008), and particle swarm optimisation (Lian et al. 2006; Lin et al. 2010). Genetic algorithms explore the solution space efficiently, leveraging the principles of survival of the fittest to converge towards good-quality solutions (Pezzella et al. 2008; Xie et al. 2019). Tabu search (Glover and Laguna 1998) starts from an initial feasible solution and explores the neighborhood of the current solution iteratively. In Saidi-Mehrabad and Fattahi (2007), tabu search was proposed for flexible JSS and it achieves better performance when compared to the branch-and-bound method. Simulated annealing is a probabilistic optimisation algorithm that mimics the annealing process in metallurgy (Aydin and Fogarty 2004). The algorithm accepts moves that lead to both better and worse solutions, allowing it to escape local optima (Bertsimas and Tsitsiklis 1993). The ant colony optimisation, bee colony optimisation, and particle swarm optimisation are nature-inspired optimisation algorithms that leverage the principles of collective intelligence observed in social insects or particles to guide the search for high-quality schedules, as evidenced by studies such as (Chong et al. 2006; Engin and Güçlü 2018; Fontes et al. 2023; Huang and Yu 2017; Wang et al. 2017). Overall, the iterative improvement heuristic approaches can find good solutions and can handle large-scale problems well. However, they are not suitable for dynamic JSS, since the rescheduling process inherent in iterative improvement heuristic approaches is time-consuming, making it inefficient for reacting to dynamic events.

2.2.3 Scheduling heuristics

Scheduling heuristics offer real-time schedule generation, making them suitable for dynamic JSS problems. These approaches assign priorities to operations or machines at decision points based on the system state. Then, operations and machines are dispatched based on their priorities, with the highest-priority operation or machine being selected first. Common

scheduling heuristics include first-in-first-out and shortest processing time first (Shanker and Tzen 1985). The choice of a scheduling heuristic depends on the specific practical problems or customer requirements. Designing scheduling heuristics capable of excelling across diverse scenarios and addressing multiple objectives simultaneously is a challenging task. It necessitates extensive domain knowledge and consumes significant time and effort.

2.2.4 Hyper-heuristic approaches

Hyper-heuristic (Burke et al. 2013), encompassing both heuristic selection and generation, stand as advanced approaches for tackling hard computational search problems. These approaches operate at a high level, automating the process of selecting or generating heuristics to enhance scheduling performance.

Heuristic selection involves selecting from a predefined set of heuristics based on their performance in specific situations (Bianchi et al. 2008). The selection mechanism relies on historical information or problem-specific features to identify the most suitable heuristic for the current situation. This adaptability allows the hyper-heuristic to dynamically choose the most effective heuristic for different instances. RL is widely used as a heuristic selection method, particularly beneficial when the number of candidate machines or operations varies at different decision points, making it challenging to define a predetermined action space (Luo 2020; Gui et al. 2023; Chang et al. 2022; Liu et al. 2022). In this case, RL is often applied to treat manual scheduling heuristics as actions (Gui et al. 2023), with the process involving the dynamic selection of appropriate actions when encountering decision points (Chang et al. 2022; Liu et al. 2022). RL also serves as a policy-driven decision framework for optimising sequences of job-machine pairs, often employed in an end-to-end manner, selecting appropriate machines or operations when encountering decision points.

Heuristic generation goes beyond selecting from predefined heuristics but creates new heuristics or combines existing ones to adapt to the given problem (Burke et al. 2007). It dynamically learns or evolves heuristics during the optimisation/learning process. GP is a popular heuristic generation method, which evolves a population of heuristics over multiple generations (Nguyen et al. 2019).

Both GP and RL have extensive applications in automatically learning scheduling heuristics/policies for solving JSS problems, capable of making effective scheduling decisions that outperform many manually designed ones by human experts in the literature. Overall, hyper-heuristic approaches, whether through heuristic selection or generation, represent a powerful paradigm to address JSS problems. These approaches leverage adaptability and computational intelligence to dynamically choose or create heuristics/policies, ultimately enhancing scheduling performance across diverse instances of the JSS problems.

3 Learning to optimise for JSS

In the field of automatic learning scheduling heuristics/policies for JSS, GP (Koza and Koza 1992) and RL (Luo et al. 2021) represent two typical hyper-heuristic approaches (Zhang et al. 2020; Song et al. 2022; Liu et al. 2022; Zhang et al. 2020; Zhou et al. 2020; Karunakaran 2019). Figure 3 illustrates the general framework depicting how these hyper-heuristic approaches, GP and RL, learn scheduling heuristics/policies for JSS problems. Notably, GP

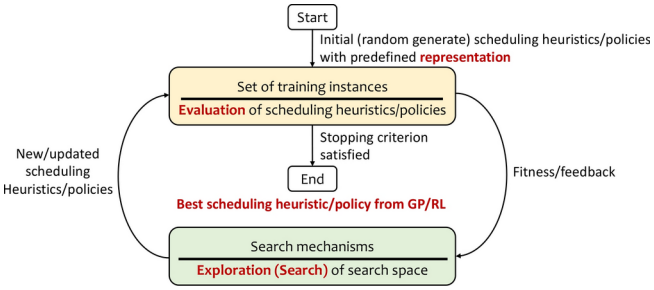
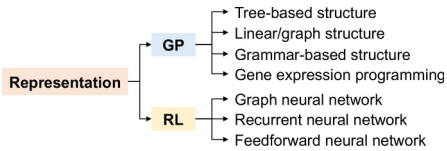


Fig. 3 The general framework of hyper-heuristic: GP and RL, for learning scheduling heuristics/policies for JSS problems

Fig. 4 The categories of mainly used representations in GP and RL for JSS



and RL share some similarities. Firstly, both GP and RL need to initialise (randomly generate) candidate scheduling heuristics/policies at the beginning. Secondly, both GP and RL need to evaluate the performance of these candidate scheduling heuristics/policies. Thirdly, both GP and RL approaches explore the search space and finally output a scheduling heuristic/policy by interacting with the dynamic environment throughout a learning process. However, they differ in the detailed processes of several key aspects: representation, evaluation, and search mechanisms. Next, we introduce and compare the principles of GP and RL for learning scheduling heuristics/policies in terms of these key aspects.

3.1 Representation

The representation is not only related to the algorithm itself but also closely tied to the nature of the specific scheduling problem being addressed. Figure 4 shows the categories of mainly used representations. All these representations necessitate problem-specific state features (terminals) as input. Table 2 presents examples of state features that can be used by both GP and RL about machines, jobs, and operations.

3.1.1 Representation of GP

Tree-based structure. In terms of GP for JSS problems, the commonly employed representation is the tree-based one. In (dynamic) JSS, where only sequencing decisions are considered, a single tree is used (Dimopoulos and Zalzala 2001). This tree is formed by combining different functions and states (Mei et al. 2017). In the case of (dynamic) flexible JSS, where two types of decision points need consideration (sequencing and routing), two typical representations can be used. One option involves a tree-based representation with two subpopulations, each dedicated to a specific decision point (Nguyen et al. 2013). Another option is a multi-tree-based representation, where one tree is designated for the sequencing rule, and the other is for the routing rule (Zhang et al. 2018; Xu et al. 2022). An illustrative example

Table 2 The example state features used for representations of GP and RL

Categories	States	Description
Machine	$WIQ_m(t)$	The remaining work (total processing time of all the operations) in the waiting queue of machine Ω_m at time t
	$NIQ_m(t)$	The number of operations in the waiting queue of machine Ω_m at time t
	$MRT_m(t)$	The ready time of machine Ω_m at time t, i.e., when machine becomes idle
	$MWT_m(t)$	The waiting time of machine Ω_m at time t, $MWT_m(t) = t - MRT_m(t)$
Job	$TIS_j(t)$	The time spent in the system by job J_j at time t
	$NOR_j(t)$	The number of the remaining operations of the job J_j at time t
	$WKR_j(t)$	The work remaining, representing the total processing time of the job J_j for the remaining operations at time t
	$TTD_j(t)$	The time until due, meaning the remaining time of the job J_j until the due date at time t
	$SLACK_j(t)$	The slack of the job J_j at time t, $SLACK_j(t) = t_j^{due} - t - WKR_j(t)$
Operation	$PT_{j,i,m}(t)$	The processing time of the operation $O_{j,i}$ on the machine Ω_m at time t
	$NPT_{j,i+1}(t)$	The median of the processing time for the next operation $O_{j,i+1}$ at time t

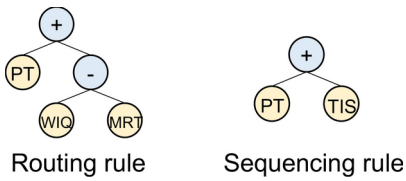


Fig. 5 An example of tree-based representation for a sequencing rule and a routing rule

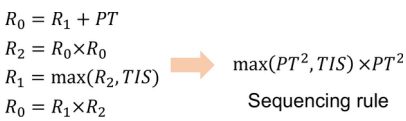


Fig. 6 An example of linear representation in GP

of the tree-based representation for a sequencing rule and a routing rule is depicted in Fig. 5. If a smaller value represents a higher priority, the routing rule prefers machines providing shorter processing time, smaller working in the queue, and larger ready time. The sequencing rule prefers operations with shorter processing time and smaller time in the system.

Linear/graph structure. Another commonly used type of representation in GP for JSS is the linear/graph representation. This representation involves a sequence of register-based instructions (Huang et al. 2021). Each register-based instruction comprises three components: the destination register, the operator (also known as the function indicator), and the source registers. During execution, the operation specified in the instruction takes the values from the source registers as inputs and assigns the calculation results to the destination register. These instructions are executed sequentially, and the values in the predefined output registers are considered the output of the entire program. The arrangement of these instructions and their sequential execution form the fundamental concept of "linear". Figure 6

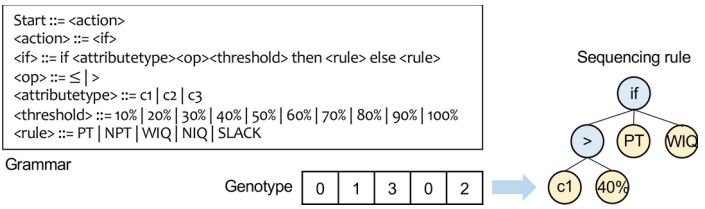


Fig. 7 An example of grammar-based representation in GP

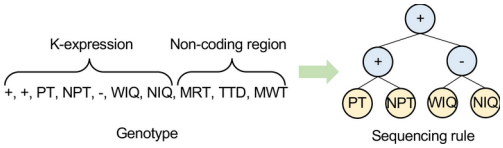


Fig. 8 An example of gene expression representation in GP

provides a simple example of a linear representation used to express a sequencing rule $\max(PT^2, TIS) \times PT^2$. In this example, all registers (i.e., R_i, $i \in [0, 1, 2]$) are initialised to 0 for simplicity. In the program, PT, PT^2, and $\max(PT^2, TIS)$ are stored in the registers accordingly, and these intermediate calculation results are ultimately aggregated into the predefined output register (i.e., R_0).

Grammar-based structure. Grammar-based representation in GP is an advanced technique that uses formal grammar to define the structure of individuals (Nguyen et al. 2012). Instead of representing individuals purely as trees (like in traditional GP), grammar-based GP ensures that the individuals generated during the evolutionary process follow a predefined syntax and structure. For example, Fig. 7 illustrates a grammar-based representation for a sequencing rule in the JSS problem. In this method, the genotype represents the selection of each component in the rule, and the genotype is decoded into a sequencing rule with a tree structure. Specifically, this grammar defines a fixed structure for a sequencing rule: "$if\ attribute\ operation\ threshold,\ use\ rule\ else\ use\ rule$". In this structure, five elements must be determined: the *attribute*, *operation*, *threshold*, and two *rule*s. For instance, possible candidates for the *attribute* are $c1$, $c2$, and $c3$. As shown in Fig. 7, the first value in the genotype is 0, then $c1$ is selected. The remaining four elements-operation, threshold, and the two rules-are selected in the same manner based on the genotype values. In this example, the final sequencing rule derived from the genotype is: $if\ c1\ >\ 40\%,\ use\ PT\ else\ use\ WIQ$.

Gene expression structure. Gene expression structure represents individuals as linear chromosomes, which are sequences of genes (Shady et al. 2023; Ozturk et al. 2019). Each gene encodes a part of the solution but does not directly represent a complete expression (Nie et al. 2013; Zhang et al. 2021). Instead, the genes are decoded into a tree structure. This involves translating the gene expression representation into a tree format. Figure 8 provides an example of a gene expression structure and its decoding process into a tree. Unlike the traditional tree-based structure, gene expression structures include K-expressions (coding regions) and non-coding regions. The K-expression is the part of the chromosome that gets decoded into a tree, while the non-coding region is ignored during the decoding process. However, this does not mean that the non-coding region is entirely useless. The non-coding region serves as a spacer or regulatory component. While it is not directly involved in the

decoding process into a tree structure, it plays an important role in maintaining the integrity and flexibility of the chromosome. The non-coding region may influence the structure of the solution indirectly by providing space for genetic operations such as mutation and crossover, which can help in exploring a broader solution space during evolutionary algorithms. The final expression is derived from the K-expression alone. In this example, the final sequencing rule derived from the genotype is: $PT + NPT + WIQ - NIQ$.

The tree-based representation in GP encompasses numerous studies for both static and dynamic, flexible and non-flexible scheduling scenarios. However, the utilisation of linear/graph representation in GP is predominantly observed in non-flexible scheduling problems. Although grammar-based and gene expression representations are also ultimately decoded into a tree structure, they are less frequently used compared to the traditional tree-based approach, as they involve more indirect encoding processes. Grammar-based representation is primarily employed to define a fixed structure for complex JSS problems or to narrow the search space, offering more control over the structure of evolved solutions (Huang et al. 2023). In Zhang et al. (2019), a novel tree-structure representation was proposed to account for the varying contributions of different terminals. In Nguyen et al. (2012), the effectiveness of different representations in GP for JSS is verified. Subsequently, in Nguyen et al. (2013), a new type of scheduling heuristic known as iterative scheduling heuristic is proposed to further enhance performance. This heuristic iteratively refines schedules by leveraging information recorded from previous or existing schedules (Nguyen et al. 2013). Similarly, Hunt et al. (2014) considers historical information to develop a scheduling heuristic that mitigates short-sightedness. Regardless of the chosen representation, GP has been shown to be suitable for addressing various scheduling problems, whether static or dynamic, flexible or non-flexible scenarios, indicating its strong generalisation ability.

3.1.2 Representation of RL

In the domain of RL, the choice of representation plays an important role across various scheduling problems. Representations such as graph neural networks, recurrent neural networks, and feedforward neural networks each excel in specific types of scenarios, including static and dynamic, as well as flexible and non-flexible scheduling scenarios.

Graph neural networks. Graph neural networks (Zhang et al. 2020; Park et al. 2021, 2021; Yuan et al. 2023; Zhang et al. 2024; Ho et al. 2023; Echeverria et al. 2024) are commonly utilised for tackling static and small-scale JSS problems due to their ability to capture relationships between machines and operations, as well as information throughout the scheduling process. In these studies, the scheduling state is typically denoted by a disjunctive graph, expressed as $G = (V, C \cup D)$ (Huang et al. 2023; Lee and Kim 2022; Hameed and Schwung 2020; Liu et al. 2024). Here, V comprises nodes representing job operations, C consists of directed arcs (conjunctions) illustrating the precedence constraints among operations, and D includes undirected arcs (disjunctions) indicating operations processed on the same machines. An illustrative example of a JSS problem represented by a disjunctive graph is provided in Fig. 9. Within this graph framework, solving a JSS instance involves determining the direction of each disjunction arc.

Although the graph representation allows for handling size-agnostic scheduling problems, which makes it applicable for dynamic and/or flexible JSS problems (Pu et al. 2024), it introduces a practical challenge when tackling large-scale JSS problems (Park and Park

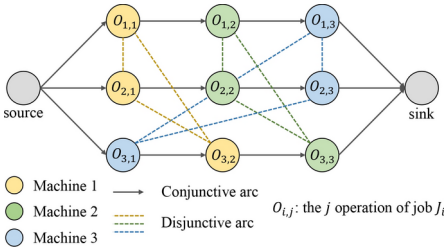


Fig. 9 An example of a disjunctive graph of a JSS problem

2021). As the number of jobs or machines increases, the graph becomes increasingly intricate and sizable, resulting in a rapid escalation of computational complexity (Liu and Huang 2023; Park and Park 2021). In such cases, learning in large graphs might not be effective. RL with the graph neural network as representation usually serves as a heuristic generation method (Huang et al. 2024; Park et al. 2021).

Recurrent neural networks. Another heuristic generation RL approach capable of addressing size-agnostic scheduling challenges involves employing recurrent neural networks (Monaci et al. 2021). Recurrent neural networks can address dynamically changing action spaces in scheduling problems. In Monaci et al. (2021), the recurrent neural network is utilised to map inputs of arbitrary length into fixed-length embeddings. The experiments conducted involve both static and dynamic JSS problems, showcasing commendable performance. However, a drawback of this method is the encode-decode mechanism, which compresses all information into a fixed-length vector, posing challenges with very long inputs (Bahdanau et al. 2014).

To address this limitation, certain studies incorporate the attention mechanism, focusing on encoding the most important features instead of all features into a single fixed-length vector (Yang 2022; Wu et al. 2024; Tassel et al. 2023; Wang et al. 2023). In (Yang 2022; Liu et al. 2024), the graph attention network is employed to derive a fixed-length vector from a disjunctive graph, with other aspects resembling graph neural network-based RL scheduling methods. Additionally, the Transformer (Vaswani et al. 2017), another widely used attention mechanism-based model, is applied to tackle JSS problems. In Zhao et al. (2022), a simplified Transformer is introduced to extract state features in vector form. Meanwhile, in Chen et al. (2022), a novel Transformer is proposed, where the graph-embedding method is initially employed to directly extract state features from the disjunctive graph. Subsequently, the attention mechanism aids agents in learning long-range dependencies more effectively.

However, the use of the attention mechanism is non-straightforward and can be computationally expensive (Adaloglou and Karagiannakos 2020), especially when dealing with large sequences or high-dimensional data. The process of computing attention scores for each element in the input sequence involves matrix multiplications, which can lead to increased computational overhead, particularly in large-scale and dynamic JSS scenarios. Meanwhile, attention mechanisms tend to focus more on local information due to the nature of attention weights (Adaloglou and Karagiannakos 2020). As a result, they might struggle to capture global dependencies within sequences, especially when long-range relationships are essential. Heuristic selection RL finds greater application in addressing flexible and/or dynamic JSS problems, effectively managing the changing action space in these scenarios. Importantly, the effectiveness of heuristic selection RL is not constrained by the scale of the problem, making it suitable for both small-scale and large-scale scenarios.

Feedforward neural networks. Typically, heuristic selection RL utilises feedforward neural networks as representations. Feedforward neural networks (Lin et al. 2019; Du et al. 2022; Park et al. 2019; Zhang et al. 2023) are highly flexible. Existing studies demonstrate their effectiveness as representations for both heuristic generation RL and heuristic selection RL. When the number of machines/operations remains constant across decision points, which ensures a fixed action space, feedforward neural networks usually serve as a representation for heuristic generation RL. Moreover, when the number of machines/operations varies, they usually serve as a representation for heuristic selection RL. In such cases, manually designed scheduling heuristics are often employed as actions instead of directly utilising machines/operations.

An example of the feedforward neural network-based representation for flexible JSS is depicted in Fig. 10, which includes the arrangement of nodes and connections (edges) between them. These layers can be broadly categorised into three types: the input layer, hidden layers, and the output layer. To be specific, the input layer represents the state features related to the dynamic flexible JSS scheduling system. Each node in the input layer corresponds to a specific state of the scheduling system. Between the input and output layers, there can be one or more hidden layers. Each node in a hidden layer represents a learned feature based on the input data. The output layer produces the final results of the neural network's computation. For heuristic generation RL, the actions directly represent the machines or operations. For heuristic selection RL, the actions correspond to the scheduling heuristics. Once the scheduling heuristic is determined, it subsequently makes the final decision regarding the machines or operations to be scheduled. The number of nodes in the output layer is determined by the desired output of the network. Each connection between nodes has an associated weight, representing the strength of the connection. Information flows from the input layer, through the hidden layers, and finally to the output layer through these weighted connections. To be noticed, the number of hidden layers and nodes in each input/hidden/output layer is a design choice and can vary based on the complexity of the problem and can vary for sequencing and routing rules.

Action design. Action design is closely tied to the representation of scheduling heuristics in RL (Serrano Ruiz et al. 2024). It defines what actions an RL agent can take at each decision point in the scheduling process. Action design must align with the problem at hand, considering the specifics of different types of JSS problems, such as standard versus flexible JSS or static versus dynamic JSS. The way actions are structured and represented fundamentally impacts the agent's ability to learn effective scheduling heuristics in complex environments. This includes specifying the types of decisions the agent can make.

In standard JSS, the primary focus is on sequencing jobs. The key challenge is determining which job should be processed at a given decision point. The action design here is

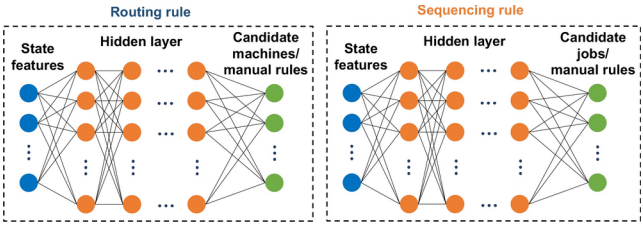


Fig. 10 An example of the feedforward neural network-based representation for flexible JSS

usually straightforward, as the RL agent selects a specific job to be processed next. This simplicity allows the agent to focus on optimising job sequencing without the added complexity of machine flexibility. Flexible JSS introduces additional complexity because multiple machines can process the same job. In this case, action design must allow the RL agent to choose both the job and the most appropriate machine from a set of alternatives. Some studies represent all possible job-machine pairs as distinct actions (Wang et al. 2023), which provides a comprehensive exploration of the solution space. However, this can lead to an overly complex action space in large-scale JSS, making it harder for RL to converge to an effective policy. To address this, hierarchical action structures are often employed (Huang et al. 2024). In this approach, the agent first selects a job and then chooses the best machine (Huang et al. 2024). This hierarchical design allows the agent to manage the larger action space more efficiently while still maintaining flexibility in decision-making. Furthermore, (Park and Park 2021) introduces a continuous action representation to keep the action space manageable, even as the number of jobs, machines, and operation types fluctuates. In dynamic JSS, uncertainty is introduced by unpredictable job arrivals or machine breakdowns. Therefore, action design must account for these real-time changes, enabling the RL agent to adapt dynamically. Actions should not only focus on job-machine pairings but also allow the agent to monitor the environment and adjust decisions as new jobs arrive or machines become available. In such cases, heuristic selection RL is a common method, where actions correspond to selecting from a predefined set of scheduling heuristics (Serrano Ruiz et al. 2024; Wu et al. 2024). The selected scheduling heuristic can easily adapt to dynamically changing scheduling environments. This simplifies the action space and allows the agent to learn which rules perform best under different dynamic conditions (Serrano Ruiz et al. 2024; Wu et al. 2024). This approach is also useful in large-scale JSS problems, where the number of possible job-machine pairings is large, and simplifying the action space can lead to faster learning and convergence. However, the limited set of actions can restrict the agent's ability to further improve performance. To address this limitation, some researchers propose composite action designs. In this approach, multiple scheduling heuristics are combined with continuous weight variables (Gui et al. 2023). This creates a more flexible and nuanced action space, allowing the RL agent to explore a wider range of potential solutions. The agent can dynamically adjust these weights to prioritise different heuristics based on the current state of the scheduling process (Gui et al. 2023).

Given that in heuristic selection RL, the scheduling heuristics as actions remain predefined and fixed during both training and test, leading to a lack of diversity in the RL agent's selection and making the performance heavily reliant on the quality of these heuristics. Compared to generating scheduling heuristics, which, in theory, can explore a vast heuristic space, learning to select scheduling heuristics involves limited exploration, covering only a subset of the space. Consequently, its performance is relatively less competitive. However, for heuristic generation RL, its interpretability of the learned scheduling heuristics is poor due to the challenge of understanding why a specific operation/machine is dispatched when the action is generated through a feedforward neural network (more like a black box), learning to select scheduling heuristics is demonstrated to be more understandable for human experts.

In conclusion, different representations of GP and RL exhibit distinct characteristics and excel in solving scenarios with different features (Zhang et al. 2023; Branke et al. 2015).

Table 3 illustrates the diverse representations of GP and RL and the scenarios in which they excel.

3.2 Evaluation

Evaluation is a critical process for assessing the effectiveness of scheduling heuristics/policies (Koza and Koza 1992). Both GP and RL utilise evaluation to measure the heuristic/policy quality and guide the learning process.

In GP, evaluation measures the effectiveness of a scheduling heuristic across the entire scheduling instance(s) using a fitness function (Xu et al. 2023). Conversely, in RL, evaluation determines the effectiveness of an action selected by a scheduling heuristic/policy through a reward function (Luo et al. 2021). Ideally, these functions should accurately represent the problem domain to which the learned scheduling heuristics/policies will be applied. In GP, the fitness function can straightforwardly reflect the objective function. However, for RL, the reward function is typically designed based on local information, and if not well-designed, it might not accurately reflect the objective function.

3.2.1 Problem instances

Static instances. For static scheduling problems, evaluations often utilise instances from widely used benchmark datasets such as TA (Taillard) (Huang et al. 2024; Nguyen et al. 2012; Mei and Zhang 2016; Park et al. 2021; Monaci et al. 2021; Pan et al. 2021; Zhang et al. 2024; Ho et al. 2023; Tassel et al. 2023; Huang et al. 2023), Brandimarte (Yska et al. 2018; Teymourifar et al. 2020; Yuan et al. 2024; Jing et al. 2024), Barnes (Yska et al. 2018; Yuan et al. 2024), Dauzere (Yska et al. 2018; Yuan et al. 2024), Hurink (Yska et al. 2018; Braune et al. 2022; Zhang et al. 2023; Yuan et al. 2024; Jing et al. 2024), FT(Zeiträg et al. 2024; Park et al. 2021; Yuan et al. 2023; Wu et al. 2024; Zhang et al. 2024), Synthetic (Yuan et al. 2024, 2023), DMU (Zhang et al. 2020; Nguyen et al. 2013, 2012; Lin et al. 2019), ORB (Lin et al. 2019; Nguyen et al. 2013, 2012; Park et al. 2021; Zhang et al. 2024), LA (Lawrence) (Lin et al. 2019; Nguyen et al. 2013, 2012; Wu et al. 2024; Zhang et al. 2024), PSPLIB (Chen et al. 2021; Đumić et al. 2021; Chand et al. 2018), ABZ (Zhang et al. 2024), SWV (Zhang et al. 2024), YN (Park et al. 2021; Yuan et al. 2023), or randomly generated instances based on specific assumptions (Jakobović and Budin 2006; Tay and Ho 2008). The majority of studies on these datasets focus on small-scale instances, typically involving no more than 200 jobs, with many considering only 20 or 50 jobs.

Table 3 Different representations of GP and RL and the scenarios in which they excel

Representations		Static	Dynamic	Small-scale	Large-scale
GP	Tree-based structure	√	√	√	√
	Linear/graph structure	√	√	√	√
	Grammar-based structure	√	√	√	√
	Gene expression programming	√	√	√	√
RL	Graph neural network	√		√	
	Recurrent neural network		√	√	
	Feedforward neural network	√	√	√	√

Dynamic instances. Dynamic scheduling problems are typically assessed using simulation models to determine the steady-state performance of scheduling heuristics/policies and usually consider large-scale scenarios with about 3000 or 6000 jobs (Zhang et al. 2018, 2021; Karunakaran et al. 2017; Holthaus and Rajendran 1997). Discrete event simulation (Liu et al. 2022; Xu et al. 2022) is the primary technique for estimating the performance of scheduling heuristics/policies (Hildebrandt et al. 2010; Liu et al. 2022). These simulation models often simulate job arrival events based on assumptions such as a Poisson distribution (Zhou and Yang 2019; Nguyen et al. 2019). While job arrival events (Zhang et al. 2019; Yska et al. 2018; Huang et al. 2023) are the most commonly simulated dynamic events, some studies also investigate machine breakdowns (Yin et al. 2003; Park et al. 2018, 2017) and other factors to simulate real-world dynamic scheduling environments.

Existing studies on GP typically focus on large-scale and dynamic scheduling problems involving approximately 3000 or 6000 jobs. Conversely, existing studies on RL typically address small-scale and static JSS problems with only hundreds of jobs. Table 4 gives a comparison of typical GP and RL across JSS problem scales and datasets. Note that some papers fall into multiple categories because they test their proposed methods on more than one dataset and sometimes include both static and dynamic scenarios. The training instances used for the evaluation process are obtained from the benchmark datasets or the simulators introduced above based on the given problem. In JSS, the final goal can be to optimise the flowtime, tardiness, or other scheduling performance metrics (Zhang et al. 2022).

3.2.2 Evaluation of GP

Fitness function. In GP, to evaluate an individual's fitness, in static JSS scenarios where problem parameters are fixed and known in advance, the scheduling heuristic is applied to a set of static training instance(s), and its performance is measured against established criteria. For dynamic JSS scenarios, the scheduling heuristic conducts evaluation using a simulator, which simulates the arrival of new jobs or changes in job characteristics during scheduling. The fitness function assesses the capacity of the scheduling heuristic to adapt to these changes and make effective scheduling decisions. After evaluation, a fitness value quantifying how effectively the scheduling heuristic addresses the given objectives is assigned. In scheduling with minimisation objectives, lower values are typically preferred (e.g., shorter flowtime, shorter tardiness). Fitness values play a crucial role in the selection process, influencing the probability of individuals being chosen as parents for breeding. Better fitness values increase the probability of an individual being selected, promoting the evolution of better solutions.

Surrogate. Different from RL, GP commonly employs the original goal as the fitness function for evaluation, indicating that scheduling heuristics are assessed based on global information. There exist some studies that propose alternative surrogate models (Zeiträg et al. 2022; Gil-Gala et al. 2023) to estimate the fitness of scheduling heuristics, such as the half-shop surrogate model (Nguyen et al. 2016) and phenotypic characteristic-based surrogate model (Zhang et al. 2022). These surrogate models demonstrate high consistency with the original fitness function, yielding scheduling performance comparable to using the original function while reducing training time in GP and providing better scheduling performance when using the same training time.

Table 4 The comparison of typical GP and RL across JSS problem scales and datasets

Dataset	Problem type	Scale (Jobs)	GP	RL
TA (Taillard)	JSS	Small-scale: 15-200 jobs	Nguyen et al. (2012, 2013)	Zhang et al. (2020); Park et al. (2021); Yuan et al. (2023); Zhang et al. (2024); Ho et al. (2023); Huang et al. (2023); Liu et al. (2024); Huang et al. (2024); Monaci et al. (2021); Yang (2022); Wu et al. (2024); Tassel et al. (2023); Chen et al. (2022); Lin et al. (2019); Pan et al. (2021); Wang and Pan (2021); Bonetta et al. (2023); Lee et al. (2024); Echeverria et al. (2024); Ho et al. (2024); Dong et al. (2024)
Brandimarte	Flexible JSS	Small-scale: 10-20 jobs	Yska et al. (2018); Teymourifar et al. (2020); Braune et al. (2022)	Pu et al. (2024); Wang et al. (2023); Yuan et al. (2024); Jing et al. (2024); Zhang et al. (2023); Echeverria et al. (2024); Ho et al. (2024); Chen et al. (2020); Yan et al. (2022); Wan et al. (2024); Xu et al. (2024); Wan et al. (2024); Zhang et al. (2024)
Barnes	JSS	Small-scale: 20-50 jobs	Yska et al. (2018)	Yuan et al. (2024)
Dauzere	Flexible JSS	Small-scale: 20-100 jobs	Yska et al. (2018)	Yuan et al. (2024)
Hurink	Flexible JSS	Small-scale: 10-30 jobs	Yska et al. (2018); Braune et al. (2022)	Echeverria et al. (2024); Pu et al. (2024); Yuan et al. (2024); Jing et al. (2024); Zhang et al. (2023); Echeverria et al. (2024); Wan et al. (2024); Xu et al. (2024); Lei et al. (2022, 2023)
FT	JSS	Small-scale: 6-20 jobs	Zeiträg et al. (2024)	Park et al. (2021); Yuan et al. (2023); Zhang et al. (2024); Wu et al. (2024); Lee et al. (2024); Ho et al. (2024); Liu et al. (2020)
DMU	Flexible JSS	Small-scale: 10-50 jobs	Nguyen et al. (2012, 2013)	Zhang et al. (2020); Yuan et al. (2023); Liu et al. (2024); Pu et al. (2024); Lin et al. (2019); Lee et al. (2024); Dong et al. (2024)
ORB	JSS	Small-scale: 10-20 jobs	Nguyen et al. (2012, 2013)	Park et al. (2021); Yuan et al. (2023); Zhang et al. (2024); Wu et al. (2024); Tassel et al. (2023); Ho et al. (2024); Lee et al. (2024); Liu et al. (2020)
LA (Lawrence)	JSS	Small-scale: 10-20 jobs	Nguyen et al. (2012, 2013); Braune et al. (2022)	Zhang et al. (2024); Tassel et al. (2023); Wang et al. (2023); Zhao et al. (2022); Lin et al. (2019); Liu et al. (2020)
ABZ	JSS	Small-scale: 10-20 jobs	–	Yuan et al. (2023); Zhang et al. (2024); Lee et al. (2024)

Table 4 (continued)

Dataset	Problem type	Scale (Jobs)	GP	RL
SWV	JSS	Small-scale: 10-50 jobs	–	Park et al. (2021); Yuan et al. (2023); Zhang et al. (2024); Tassel et al. (2023); Lee et al. (2024); Ho et al. (2024)
YN	JSS	Small-scale: 10-20 jobs	–	Park et al. (2021); Yuan et al. (2023); Zhang et al. (2024); Tassel et al. (2023); Lee et al. (2024); Ho et al. (2024)
Randomly generated	JSS	Small-scale: 5-100 jobs	Hart and Sim (2016)	Hameed and Schwung (2020); Liu and Huang (2023); Du et al. (2022); Serrano Ruiz et al. (2024); Liu et al. (2022)
	Flexible JSS	Small-scale: 10-40 jobs	–	Song et al. (2022); Park and Park (2021); Wang et al. (2023); Park et al. (2019); Zhang et al. (2023); Du and Li (2024); Du et al. (2022); Aissani et al. (2012); Ding et al. (2024)
Dynamic simulator	Dynamic JSS	Small-scale: 6-600 jobs	Yin et al. (2003); Dimopoulos and Zalzala (2001); Gil Gala et al. (2022); Gil-Gala et al. (2023); Planinić et al. (2021); Đurasević et al. (2023); Đurasević et al. (2016); Đurasević et al. (2023); Đurasević et al. (2020); Jaklinović et al. (2021); Đurasević et al. (2018); Gil Gala et al. (2019); Đurasević et al. (2018); Song and Lin (2021); Gil Gala et al. (2021)	Liu et al. (2024); Huang et al. (2024); Zeng et al. (2022); Lei et al. (2024)
		Large-scale: 3000 jobs	Nguyen et al. (2019); Karunakaran et al. (2017); Shady et al. (2023); Hunt et al. (2014); Holthaus and Rajendran (1997); Hildebrandt et al. (2010); Park et al. (2018, 2017); Sitahong et al. (2022); Shady et al. (2022); Karunakaran et al. (2017); Mei et al. (2016); Park et al. (2016, 2015); Karunakaran et al. (2018); Nguyen et al. (2015, 2013, 2014); Park et al. (2018); Nguyen et al. (2014); Hunt et al. (2016); Mei et al. (2017); Zeiträg et al. (2022); Fan et al. (2021); Park et al. (2018, 2016)	Wang (2020); Wang and Yan (2016)
		Large-scale: 6000 jobs	Nguyen et al. (2019); Mei et al. (2017); Nguyen et al. (2013); Mei et al. (2017); Nguyen et al. (2016) Nguyen et al. (2012); Xu et al. (2021); Zhang et al. (2022); Huang et al. (2023)	–

Table 4 (continued)

Dataset	Problem type	Scale (Jobs)	GP	RL
	Dynamic flexible JSS	Small-scale: 6-800 jobs	Ozturk et al. (2019); Nie et al. (2013); Zhang et al. (2021); Teymourifar et al. (2020); Tay and Ho (2008); Planinić et al. (2021); Gil Gala et al. (2019); Zakaria et al. (2021); Gao et al. (2015); Zhu et al. (2020); Ho and Tay (2005)	Luo (2020); Gui et al. (2023); Chang et al. (2022); Liu et al. (2022); Huang et al. (2023); Luo et al. (2021); Liu and Huang (2023); Wan et al. (2024); Luo et al. (2021); Li et al. (2022); Wu et al. (2023); Bouazza et al. (2017); Hu et al. (2020); Yan et al. (2022); Said et al. (2021); Zhang et al. (2022, 2024)
		Large-scale: 3000 jobs	Zhou et al. (2018); Zhou and Yang (2019); Zhou et al. (2020)	–
		Large-scale: 6000 jobs	(xu et al. (2023); Zhang et al. (2020, 2020, 2020, 2018); Xu et al. (2022); Zhang et al. (2019); Xu et al. (2023); Yska et al. (2018); Zhang et al. (2021); Xu et al. (2022); Zhang et al. (2019, 2022); Nguyen et al. (2012); Xu et al. (2021); Zhang et al. (2021, 2021, 2019); Xu et al. (2021); Yska et al. (2018); Zhang et al. (2018); Nguyen et al. (2018); Zhang et al. (2023, 2021); Xu et al. (2023, 2022); Zhang et al. (2023, 2023, 2022)	–

3.2.3 Evaluation of RL

In RL, the reward function is usually designed to assess the performance of each action by providing positive or negative feedback to direct the agent towards desirable outcomes and away from undesirable ones. RL relies heavily on a well-designed reward function, which is different from the original goal. Designing a reward function for RL in job shop scheduling involves careful consideration of the problem's objectives, state-action space, and the desired behaviour of the RL agent (Zhang et al. 2024). Specifically, the following issues must be addressed:

- The reward function should align with the scheduling objectives, such as minimising flowtime or minimising tardiness, to ensure that the RL agent learns scheduling heuristics that lead to desired outcomes.
- The reward function should be based on the current state of the system and the action taken by the RL agent. This ensures that the reward function accurately evaluates the impact of each action on achieving the scheduling objectives.
- The reward function should encourage exploration of the solution space while ensuring effective exploitation of learned knowledge. Balancing exploration and exploitation is important for finding optimal or near-optimal solutions. Care must be taken to avoid reward functions that lead to suboptimal scheduling heuristics.Overall, designing a re-

ward function for RL in job shop scheduling requires domain knowledge and iterative refinement to achieve effective learning and scheduling performance. The reward function serves as a signal guiding the agent's actions in various states, indicating the desirability of each action. By optimising its scheduling heuristic/policy based on received rewards, the agent learns to make decisions maximising cumulative rewards as Eq (1).

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{1}$$

where γ is the discount factor, and r_{t+k+1} is the reward at time $t + k + 1$.

The discount factor $\gamma \in [0, 1]$ is a parameter that determines the importance of future rewards relative to immediate rewards. It highlights that rewards expected to be received in the distant future are considered less significant than those obtained in the immediate future, indicating RL's emphasis on prioritising immediate/local information. A poorly designed reward function in RL could lead to suboptimal and even poor search directions and potential loss of effective scheduling heuristics/policies.

3.2.4 Number of objectives

In scenarios involving multiple conflicting objectives, known as multi-objective optimisation, the fitness or reward function(s) that considers multiple objectives simultaneously are typically utilised. In GP, the multi-objective fitness function either consolidates diverse objectives into a single metric or evaluates each objective independently and then employs multi-objective optimisation techniques such as Pareto dominance to merge multiple objectives (Zhang et al. 2023; xu et al. 2023; Đurasević et al. (2023). In RL, a straightforward way is to design multiple reward functions corresponding to distinct objectives and give feedback after an action considers these multiple reward functions collectively (Luo et al. 2021). Other studies develop more reward functions than the number of objectives considered, aiming to achieve a favorable compromise between the objectives over the long-term schedule (Luo et al. 2021; Wu et al. 2023). They accomplish this by adaptively selecting different reward functions at various decision points. To be specific, in Wu et al. (2023), six reward functions are developed for two objectives. They employ a hierarchical network architecture: a higher network determines a reward function corresponding to one optimisation objective (e.g., makespan or total tardiness), while a lower network selects an action based on the current scheduling situation and input from the selected reward function and other five reward functions. Through the collaboration between these two networks, different reward functions adaptive to the current environment alternate to guide the training process, maximising cumulative rewards. Some studies also propose using a single aggregated reward function by considering different objectives together (Du and Li 2024; Du et al. 2022).

Overall, regardless of GP or RL, evaluation plays a key role in guiding algorithms towards good scheduling performance. It steers the search process towards improved solutions across iterations. The effectiveness of an algorithm relies on the accurate representation of problem-specific objectives in the fitness/reward function. Building on the above introduction and discussion, Table 5 summarises the key characteristics of evaluation in GP

Table 5 The characteristics of evaluation in GP and RL

GP	RL
1. *Direct evaluation*: fitness functions are often easy to design and understand. 2. *Global view*: evaluation considers overall performance across all training instances using global information, ensuring the scheduling heuristic addresses final objectives. 3. *Population evaluation*: allows simultaneous evaluation of multiple scheduling heuristics, fostering exploration of diverse scheduling heuristics. 4. *Loss of vocal accuracy*: focus on global performance might overlook issues with specific decisions during scheduling	1. *Indirect evaluation*: reward functions might not perfectly align with the final objective, requiring careful design and potential domain expertise. 2. *Local focus*: allows for fine-grained learning by providing feedback for each action using local information. 3. *Single heuristic/policy evaluation*: typically focuses on training a single scheduling heuristic, potentially saving evaluation time. 4. *Sensitivity to reward design*: poorly designed reward functions can lead the agent towards suboptimal solutions or totally wrong direction

and RL. It is important to note that these characteristics often present a trade-off. While one approach might offer an advantage in a specific aspect, it might come at the cost of another.

3.3 Search mechanisms

The search mechanisms employed by GP and RL share similarities in their iterative nature and utilisation of feedback from the scheduling system to refine scheduling heuristics/policies (Xu et al. 2024). However, significant differences exist between the two approaches.

In GP, the search process revolves around the operation and evolution of a population of scheduling heuristics (individuals) (Zhang et al. 2021). Initially, this population is generated randomly. Through iterations, individuals within this population undergo genetic operations such as selection, crossover, and mutation, mimicking the process of natural selection (Koza et al. 1994). Parent selection for crossover and mutation is driven by the fitness of each scheduling heuristic, as determined by a predefined fitness function. The population evolves over time, with better-performing individuals being more likely to survive and pass on their traits (building blocks) to the next generation. This population optimisation strategy allows GP to explore a diverse range of scheduling heuristics and potentially discover novel and good solutions. On the other hand, RL focuses on optimising a single scheduling heuristic/policy through repeated interactions with the scheduling environment (Kaelbling et al. 1996). The RL agent interacts with the environment by selecting actions (scheduling decisions) based on its current scheduling heuristic/policy and receiving feedback in the form of rewards or penalties. These rewards or penalties are given by a designed reward function. Then RL uses some learning algorithms and this feedback to update the scheduling heuristic/policy iteratively.

The key difference between GP and RL lies in their strategy of optimisation: GP explores a diverse population of scheduling heuristics collectively, while RL focuses on refining a single heuristic/policy through iterative interactions with the environment. This fundamental distinction shapes their respective search mechanisms and influences their applicability to different scheduling problems. The population-based strategy of GP offers several advantages. Firstly, it facilitates a diverse exploration of the search space, preventing premature

convergence to suboptimal and promoting the discovery of novel, high-quality scheduling heuristics. Moreover, it provides decision-makers with a range of options and insights into the problem space, potentially uncovering multiple viable solutions. Detailed descriptions of the search mechanisms and key components utilised by GP and RL are introduced as follows.

3.3.1 Search mechanism of GP

The search mechanism of GP involves selection and breeding among a population of scheduling heuristics (Gil Gala et al. 2021). Selection is essential in GP that determines the direction of evolution (Helmuth and Abdelhady 2020). Selection can be divided into elite selection and parent selection. *Elite selection* involves preserving a certain percentage of the best individuals from the current generation to be directly carried over to the next generation. This helps maintain high-performing individuals in the population. *Parent selection* determines which individuals from the current population will be chosen to serve as parents for creating the next generation (Fang and Li 2010). The choice of parents significantly influences the diversity and quality of the population. Individuals with better fitness values are more likely to be selected as parents. This approach emphasises the principle of "survival of the fittest" and aims to propagate genetic material from individuals that have performed well in solving the problem. Tournament selection and roulette wheel selection are two classical parent selection examples commonly used in GP.

Tournament selection involves randomly selecting a subset of individuals from the population and choosing the one with the highest fitness as a parent (Fang and Li 2010). *Roulette wheel selection* allocates a probability of selection to each individual in proportion to their fitness (Xie 2009). Individuals with better fitness have larger "slices" of the roulette wheel, increasing their chances of being chosen. Tournament selection and roulette wheel selection primarily select parents based on their fitness. However, more advanced parent selection methods in JSS further consider diversity and complementarity. *Cluster selection* identifies individuals exhibiting varied behaviours to serve as parents (Xu et al. 2022). *Diverse partner selection* prioritises individuals who have complementary strengths as parents (Xu et al. 2022). *Lexicase selection*, on the other hand, targets expert individuals who demonstrate effectiveness in various cases to be selected as parents (Xu et al. 2023). The choice of parent selection operator depends on the specific characteristics of the problem and the desired balance between exploration and exploitation.

The breeding process in GP involves the creation of new individuals by combining genetic material from selected parents (Koza and Koza 1992). The primary genetic operators used in GP include crossover, mutation, and reproduction.

Crossover involves exchanging genetic material between two parents to generate one or more offspring. Crossover facilitates the recombination of building blocks from different parents, allowing the offspring to inherit successful combinations of genetic material. Various crossover methods exist, with subtree crossover being a typical example. Subtree crossover involves selecting a subtree from one parent and replacing a randomly chosen subtree in the other parent (Zhang et al. 2018). *Mutation* introduces small random changes to an individual's genetic material (Koza and Koza 1992). Mutation introduces genetic diversity, helping the population explore new regions of the solution space that can lead to improved solutions. Mutation can take various forms, such as changing a function or terminal in a

subtree or adding/deleting nodes. Mutation serves as an exploration mechanism, allowing the algorithm to discover novel solutions and prevent premature convergence. *Reproduction* allows certain individuals selected by parent selection (not necessarily the best ones) to be directly copied into the next generation without modification (Koza et al. 1994). Reproduction allows successful building blocks (effective parts of a scheduling heuristic) discovered in previous generations to be carried forward without risking modification through crossover or mutation and continue to contribute to the next population. Reproduction helps maintain consistency and stability in the population.

These genetic operators often work simultaneously. Typically, a combination of crossover, mutation, and reproduction is applied to generate offspring in a new generation (Koza et al. 1994). Beyond these basic genetic operators, research on advanced genetic operators includes adaptive rate adjustment to achieve a balance between exploration and exploitation (Yang et al. 2024), depth-dependent operators that restrict the selection of subtrees at specific depths (Zhang et al. 2022), and semantic operators that bias offspring towards predefined semantics (Xu et al. 2023). More targeted studies control the exchange of specific subtrees by considering subtree importance. In (Zhang et al. 2020), a subtree importance measuring method is proposed by considering that subtrees containing more important features are more significant. This approach gives a higher probability of removing unimportant subtrees from an individual and a higher probability of acquiring important subtrees from the other parent. However, subtree importance based solely on the frequency of features might not be accurate, as GP individuals may contain redundant subtrees. To address this issue, (Zhang et al. 2021) measured subtree importance by comparing the behaviour of a subtree with that of the entire tree. Consequently, when performing crossover operations, if the offspring are generated by replacing an unimportant subtree with an important subtree from the other parent, the resulting offspring are more likely to exhibit good quality.

In addition to standard search mechanisms, some advanced machine learning techniques, such as transfer learning (Zhang et al. 2023, 2022) and ensemble learning (Đurasević et al. 2018; Hart and Sim 2016; Gil Gala et al. 2022; Đurasević et al. 2023) are also studied in some advanced GP approaches. Transfer learning is usually applied to address multi-task JSS problems (Zhang et al. 2023, 2022). By transferring knowledge between subpopulations or individuals for solving different tasks (often implemented through crossover), transfer learning allows GP to leverage experiences in solving related scheduling tasks (Zhang et al. 2023, 2022, 2021). Ensemble Learning involves learning a group of scheduling heuristics and then aggregating their decisions (Đurasević et al. 2018; Park et al. 2018). This can improve the stability and reliability of the final scheduling decisions (Xu et al. 2023).

3.3.2 Search mechanism of RL

The search mechanism of RL involves adjusting the parameters of the scheduling heuristics/policies to improve the accuracy of decisions based on some learning algorithms. Different learning algorithms in RL can vary in how they optimise the parameters of a neural network, which serves as the value function or function approximator for the heuristic/policy. Widely used learning algorithms in RL for scheduling problems can be categorised into three main types based on their strategy for learning: value-based, policy-based, and actor-critic methods, as shown in Fig. 11.

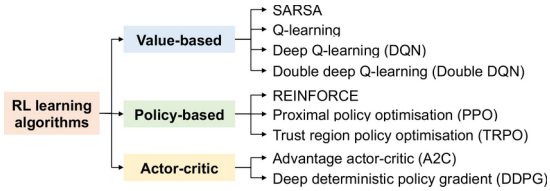


Fig. 11 The categories of RL learning algorithms

Value-based methods learn the value function, which estimates the expected return or cumulative reward associated with being in a particular state and taking a specific action. The construction and calculation of the value function are central to value-based RL. Representative algorithms of value-based learning algorithm in production scheduling optimisation include SARSA (Chen et al. 2020), Q-learning (Bouazza et al. 2017), deep Q-networks (DQN) (Hu et al. 2020), and double DQN (Van Hasselt et al. 2016). SARSA updates the Q-value (action-value) based on the action actually taken in the next state, as demonstrated in (Chen et al. 2020; Orhean et al. 2018; Aissani et al. 2012). Different from SARSA, Q-learning updates its Q-values based on the maximum Q-value of the next scheduling state regardless of the action taken (Bouazza et al. 2017; Wang 2020; Stricker et al. 2018; Wang and Yan 2016; Said et al. 2021; Yan et al. 2022). Both SARSA and Q-learning employ a table to record state-action values, but this approach becomes impractical when the state or action space is large. Therefore, the DQN is proposed by integrating Q-learning and the feedforward neural network to approximate the value function (Hu et al. 2020; Li et al. 2022; Liu et al. 2022; Yan et al. 2022; He et al. 2022). DQN uses experience replay and target network to overcome the instability of the algorithm. However, DQN's tendency to overestimate action values poses a limitation due to using the same network for action selection and evaluation. To overcome this limitation, double DQN is proposed by using two separate networks: one for action selection (the target network) and one for value evaluation (the online network) (Van Hasselt et al. 2016; Liu et al. 2022; Du et al. 2022; Wu et al. 2023; Zeng et al. 2022; Ho et al. 2023). The target network is periodically updated with the parameters of the online network, which helps stabilise training and reduce overestimation bias. By decoupling action selection from value evaluation, Double DQN provides more accurate Q-value estimates and leads to improved scheduling performance.

Policy-based methods directly parameterise the neural networks (scheduling heuristics/policies), which specifies the probability distribution over actions given states. REINFORCE (Wang and Pan 2021), proximal policy optimisation (PPO) (Zhang et al. 2022; Xu et al. 2024), and trust region policy optimisation (TRPO) (Kuhnle et al. 2019) are examples of policy-based methods used in scheduling. REINFORCE directly optimises the neural network by estimating the gradients of the expected return with respect to the neural network parameters (Wang and Pan 2021; Bonetta et al. 2023; Zhang et al. 2024). PPO aims to improve upon REINFORCE by addressing its high variance and instability issues. It constrains the neural network update step size using a clipping mechanism or a penalty term, ensuring that the neural network update does not deviate too far from the previous neural network (Zhang et al. 2020; Rummukainen and Nurminen 2019; Zhang et al. 2022). TRPO seeks to further improve sample efficiency and stability by constraining the neural network update step size based on a trust region constraint (Kuhnle et al. 2019). It aims to ensure that the updated neural network does not deviate too far from the previous neural

network, thereby preventing large neural network changes that could lead to performance degradation.

Actor-critic methods combine elements of both value-based and policy-based approaches. They involve two neural networks: an actor network that learns the scheduling heuristic/policy and a critic network that learns the value function. The parameters of both networks are optimised simultaneously using techniques such as stochastic gradient descent or variants like Adam. Deep deterministic policy gradient (DDPG) (Liu et al. 2020) and advantage actor-critic (A2C) (Hubbs et al. 2020) are examples of actor-critic methods. DDPG learns a deterministic scheduling heuristic/policy, directly mapping states to actions without any randomness (Liu et al. 2020). In DDPG, actions are selected by adding exploration noise to the output of the deterministic scheduling heuristic/policy. This noise encourages exploration and prevents the scheduling heuristic/policy from getting stuck in local optima. DDPG uses a separate target network to estimate the value of state-action pairs. This target network is periodically updated with the parameters of the main network to stabilise training. A2C learns a stochastic scheduling heuristic/policy, where the scheduling heuristic/policy outputs a probability distribution over actions for a given state (Hubbs et al. 2020). In A2C, actions are sampled from the probability distribution output by the stochastic scheduling heuristic/policy. This allows for more flexible exploration strategies and can lead to better exploration in high-dimensional or continuous action spaces. A2C uses a value function (critic) to estimate the state-value function, which represents the expected return starting from a given state (Wan et al. 2024). Additionally, A2C estimates the advantage function, which represents the advantage of taking a specific action compared to the average action value in a given state.

4 GP and RL for different job shop scheduling problems

In this section, we introduce different kinds of GP and RL methods designed for different kinds of JSS problems. In this survey, the JSS problems are classified according to environmental conditions. They are classified into static scheduling problems, including static JSS and static flexible JSS problems, and dynamic scheduling problems, including dynamic JSS and dynamic flexible JSS problems. Next, we give a literature review of GP and RL methods for each typic of JSS problem, static and dynamic, flexible and non-flexible.

4.1 Static scheduling problems

For static scheduling problems, all scheduling information is known in advance. Specifically, static JSS involves handling sequencing decision points, while static flexible JSS requires handling both sequencing and routing decision points. In the domain of learning scheduling heuristics/policies for static scheduling problems, GP is used as a heuristic generation method, while RL can serve as either a heuristic selection method or directly optimise sequences of job-machine pairs. However, RL is more usually used to directly optimise sequences of job-machine pairs for static scheduling problems. Following are some examples illustrating the application of GP and RL in JSS and flexible JSS.

4.1.1 Static JSS

For learning scheduling heuristics/policies for static JSS problems, both GP and RL usually only need to learn a sequencing rule, as the machine to process an operation is given in advance.

In the early stages, GP was primarily applied to address static JSS problems (Miyashita 2000). Experimental results revealed that GP could evolve effective scheduling heuristics for solving such problems. Some foundational studies (Nguyen et al. 2012, 2013) investigate representations in this domain. Beyond these, several studies have explored advanced strategies. For instance, Nguyen et al. proposed two selection strategies to aid surrogate-assisted GP in evolving scheduling heuristics for JSS (Nguyen et al. 2014). These strategies aim to enhance population diversity by selecting individuals with different behaviours (phenotypes) or genotypes. Additionally, Mei et al. introduced an efficient feature selection strategy for GP to solve JSS problems (Mei et al. 2017). Specifically, they developed a niching-based search framework to extract a diverse set of good scheduling heuristics, while employing a surrogate model to reduce the complexity of fitness evaluation.

In Karunakaran et al. (2016), the focus is on addressing the multi-objective JSS problems using a novel GP method assisted by the island model. The experimental results reveal that this innovative GP method, in conjunction with the island model, exhibits superior performance compared to classical multi-objective optimisation methods. The paper considers three objectives. In related work, the many-objective JSS problem is addressed, involving four or more objectives (Masood et al. 2016, 2018; Đurasević et al. 2018; Masood et al. 2019). To tackle this complex problem, the study proposes a new hybridised algorithm that combines GP and NSGAIII, showcasing a comprehensive approach to address the challenges posed by many objectives in the JSS domain.

For RL approaches that directly optimise sequences of job-machine pairs in static JSS, the method functions as an end-to-end approach, where the learned scheduling policy directly selects from available operations at each decision point. Particularly, some representative works regarding end-to-end RL include (Zhang et al. 2020; Park et al. 2021, 2021; Yuan et al. 2023). In these studies, the scheduling state in static JSS scenarios is typically denoted by a disjunctive graph (Dong et al. 2024). Within this graph framework, solving a JSS instance usually involves using a graph neural network to learn the embeddings of operations (nodes) within the disjunctive graph (Ho et al. 2024). Different learning algorithms for optimising the parameters of the graph neural network can be used. For example, in Zhang et al. (2020), the scheduling policy is trained using a policy gradient algorithm. In Park et al. (2021), Park et al. (2021), Yuan et al. (2023), Liu and Huang (2023), the PPO learning algorithm and its variants are applied for stable learning performance. Instead of only testing on static scheduling scenarios (Park et al. 2021, 2021; Yuan et al. 2023), the study by Liu and Huang (2023) extends its analysis to include dynamic scheduling scenarios for testing the trained scheduling policies on test scenarios. The authors claim that the scheduling policies can be statically trained and directly tested in either a static or dynamic environment without additional adjustments, showcasing the flexibility of their method. Nevertheless, training in a static environment and applying the model to a dynamic environment might result in the loss of important information during the training process. Consequently, the models trained on static instances might exhibit poor performance when faced with unseen dynamic instances. Beyond the graph neural network, in the study by Pan et al. (2021), a novel RL

method with an innovative network architecture is developed to address the static permutation flow shop scheduling problem. This novel network integrates the recurrent neural network and feedforward neural network, augmented with a convolutional layer to enhance its flexibility for solving different scales of scheduling problems. The parameters of this network are trained using the actor-critic algorithm.

Rather than employing RL as an end-to-end method for JSS, some studies leverage RL as a heuristic selection method, aiming to learn the selection of scheduling heuristics from a predefined set Ding et al. (2024); Lee et al. (2024). In these studies, feedforward neural networks are usually used to map the input scheduling state to the actions, which are usually manually designed scheduling heuristics (Lin et al. 2019). When a decision point is encountered, a scheduling heuristic is selected. Then, the chosen scheduling heuristic will determine which machine/operation to select. In addition, a replay memory is commonly used to store some experience which will be used as guidance for future decisions.

4.1.2 Static flexible JSS

For flexible JSS, we need to consider not only the operation sequencing but also the machine allocation (routing). The key aspect here is how to consider the sequencing and routing situation together in GP and RL.

In Braune et al. (2022), an investigation of both single-tree and multi-tree representations in GP for solving the flexible JSS problem is conducted. For the single-tree representation, the work involves learning sequencing and routing rules in two parallel GP methods, while the GP with multi-tree representation simultaneously learns scheduling heuristics with sequencing and routing rules. The effectiveness of both representations is validated on benchmark flexible JSS instances against manually designed scheduling heuristics. Furthermore, this study extends its evaluation to dynamic and real-world flexible JSS instances, verifying the effectiveness of the proposed methods. In a more specialised context, the study in Ho and Tay (2005) employs GP to evolve scheduling heuristics for solving a more complex flexible JSS problem with recirculation, which means a job can typically visit a machine more than once. In Zhu et al. (2020), GP is used to address a specific flexible JSS problem incorporating multi-process planning. The evolved scheduling heuristics demonstrate superiority over manually designed scheduling heuristics. These studies showcase the effectiveness of GP in solving specific challenges within the flexible JSS problem domain. In Tay and Ho (2008), the multi-objective flexible JSS problem is studied with three objectives considered. However, these objectives are linearly combined into a single objective, transforming the original multi-objective flexible JSS problem into a single-objective flexible JSS problem. In Zhou et al. (2019), a multi-agent-based GP is proposed for solving the flexible JSS problem, with a focus on a real-world application of GP in an Aero-Engine blade manufacturing plant.

Concerning RL for flexible JSS, in Song et al. (2022), an end-to-end RL is introduced for automatically learning scheduling policies for flexible JSS. This approach combines operation selection and machine assignment into a composite decision and incorporates a heterogeneous graph structure to capture intricate relationships among operations and machines Wan et al. (2024). The actions in this context correspond to feasible job-machine pairs. The obtained results indicate that the learned scheduling policies outperform traditional manually designed heuristics. Similarly, the study in Yuan et al. (2024) also employs a graph to

encode scheduling information and combines operation selection with machine assignment as a composite decision, and actions correspond to feasible job-machine pairs. In Zhang et al. (2023), Jing et al. (2024), the multi-agent graph is constructed to represent the relationships between operations among machines and jobs. Each agent collaborates with its neighboring agents to take coordinated actions, addressing both sequencing and routing decisions simultaneously. While end-to-end RL can address the changing action space of flexible JSS problems, its applicability is limited due to the encoding of scheduling information using a graph. This limitation renders it suitable primarily for small-scale JSS problems (Song et al. 2022). Heuristic selection RL is demonstrated to be more applicable and prevalent in the flexible JSS domain. For example, in Du et al. (2022), a heuristic selection RL is proposed to solve a multi-objective flexible JSS problem. To tackle the operation sequencing and machine routing within flexible JSS, this paper integrates manually designed composite scheduling heuristics, combining these sub-problems into a unified framework. Then an agent is learned to make decisions among these composite scheduling heuristics.

4.2 Dynamic scheduling problems

Differing from static JSS, dynamic JSS involves incomplete advanced knowledge of job or machine information (Đurasević et al. 2016; Đurasević et al. 2023). Specifically, dynamic JSS involves managing sequencing decision points in a dynamic environment, while dynamic flexible JSS requires handling both sequencing and routing decision points in such an environment. In the domain of learning scheduling heuristics/policies for dynamic scheduling problems, GP is used as a heuristic generation method, while RL can serve as either a heuristic selection method or directly optimise sequences of job-machine pairs. However, RL is more often used as a heuristic selection method for dynamic scheduling problems. Following are some examples illustrating the application of GP and RL in dynamic JSS and dynamic flexible JSS.

4.2.1 Dynamic JSS

In dynamic JSS, GP is applied to evolve effective scheduling heuristics considering various dynamic events. Example studies such as (Xu et al. 2021; Gil Gala et al. 2019; Fan et al. 2021; Karunakaran et al. 2017; Park et al. 2018) focus on GP for dynamic JSS, particularly with a focus on new job dynamic arrival events. These studies validate the effectiveness of GP against manually designed scheduling heuristics. A study in Park et al. (2018) extends their focus to consider both new job dynamic arrival events and machine breakdown events, with the introduction of new machine breakdown terminals. Enhanced GP methods, integrating various machine learning techniques such as feature selection (Shady et al. 2022, 2023; Mei et al. 2016; Sitahong et al. 2022), multi-task learning (Huang et al. 2023; Zhang et al. 2021), ensemble (Park et al. 2018), niching (Park et al. 2016; Zakaria et al. 2019), and surrogate (Zeiträg et al. 2022; Zhang et al. 2022), have been studied to further improve the effectiveness of GP in dynamic JSS. In the above studies, a single-tree representation is employed to learn scheduling heuristics with a single sequencing rule.

Nevertheless, in specific scenarios, employing two rules can yield superior results. Differing from the sequencing rule and routing rule typically utilised for flexible JSS, these two rules consist of a general sequencing rule and a specialised rule designed for a specific

goal. For example, in Park et al. (2018), a specialised rule for effectively handling various machine breakdown issues is developed. In Nguyen et al. (2012), a due-date assignment rule is evolved specifically for optimising the tardiness objective. In addition to the effectiveness, certain studies focus on the interpretability of evolved scheduling heuristics by GP for dynamic JSS (Mei et al. 2017; Hunt et al. 2016).

Moreover, several studies concentrate on the application of GP to address multi-objective dynamic JSS. For example, in Nguyen et al. (2013), novel multi-objective GP methods are formulated by integrating GP with classical multi-objective algorithms, including non-dominated sorting genetic algorithm II (NSGAII) and strength Pareto evolutionary algorithm 2 (SPEA2). In Karunakaran et al. (2018) a novel GP for evolving sampling heuristics for multi-objective dynamic JSS is developed. Specifically, during the evolutionary process, sampling heuristics are employed to discard poor instances in favor of good instances, thereby enhancing the Pareto front. More studies of GP for multi-objective dynamic JSS can refer to (Nguyen et al. 2013, 2015, 2012; Yin et al. 2003).

Both end-to-end RL and heuristic selection RL can be used for dynamic JSS. In Liu and Huang (2023), a novel end-to-end RL with graph neural networks is proposed for the dynamic JSS problems considering stochastic job arrivals and random machine breakdowns. Different from the studies of the RL with graph neural networks for static JSS, the graph neural networks are required to update frequently as new jobs continue to arrive and/or when machine breakdown happens. In the application of heuristic selection RL to dynamic JSS problems (Zeng et al. 2022; Wu et al. 2024; Lei et al. 2024), manually designed sequencing rules are commonly employed as the agents' actions, instead of directly utilising operations as actions. These RL methods address the challenges faced by end-to-end RL approaches, which encounter difficulties with dynamically changing action spaces (Zeng et al. 2022).

4.2.2 Dynamic flexible JSS

Dynamic flexible JSS is more complex compared to other types of JSS due to complicated interactions between sequencing and routing decisions in a dynamic environment (Nguyen et al. 2018). Dynamic flexible JSS has been attracting more and more attention from researchers due to its practical value.

For dynamic flexible JSS, cooperative coevolution GP (Yska et al. 2018) and multi-tree GP (Zhang et al. 2018) are two typical GP methods. In the cooperative coevolution GP approach, the sequencing rule and routing rule are evolved in two distinct sub-populations (Yska et al. 2018). On the other hand, the multi-tree GP method evolves both sequencing and routing rules simultaneously within a single population, with each individual comprising two trees (Zhang et al. 2018). Research indicates that both of these methods can achieve superior performance compared to manually designed scheduling heuristics. On top of that, certain improved cooperative coevolution GP methods and multi-tree GP methods are developed by combining them with one or multiple machine learning techniques, involving feature selection (Yska et al. 2018; Zhang et al. 2019; Zakaria et al. 2021; Zhang et al. 2020, 2020, 2020, 2021), surrogate (Zhang et al. 2018; Zhou et al. 2020; Zhang et al. 2021), multi-task learning (Zhang et al. 2023), ensemble (Gao et al. 2015; Park et al. 2015; Rodrigueset al. 2020; Xu et al. 2023; Park et al. 2016), and hybirdisation of these techniques (Zhang et al. 2021, 2021). A more detailed review of GP with these machine learning techniques for JSS problems can refer to a survey in Zhang et al. (2023).

Moreover, various GP methods find extensive application in addressing the dynamic flexible JSS problems with multiple objectives considering new job dynamic arrival events (Zhou et al. 2018; Zhang et al. 2019; Zhou and Yang 2019; Zhang et al. 2023, 2022) or machine breakdown events (Chen et al. 2018; Park et al. 2017). In Teymourifar et al. (2020), the multi-objective dynamic flexible JSS problems incorporating both new job arrivals and machine breakdowns are explored. These approaches typically make routing and sequencing decisions in a delay-free manner, which might have limitations in dynamic environments. To address this, Xu et al. (2021) proposes a delay method for the multi-objective dynamic flexible JSS problem, delaying routing decisions to ensure decisions are made under the latest and most accurate information. Specifically, ready jobs are sent to a pool instead of immediately assigning them to machines. When a machine becomes idle, the delayed routing policy is employed to create a candidate operation set for that idle machine. In Zhang et al. (2023), an interpretability-aware multi-objective GP is proposed, optimising not only for scheduling quality but also for smaller heuristic sizes to enhance interpretability. Some studies simultaneously handle multiple tasks and multiple objectives in dynamic flexible JSS (Zhang et al. 2022, 2023).

In terms of RL for dynamic flexible JSS, the study in Lei et al. (2023) proposed a end-to-end RL. The main principle is to divide the dynamic flexible JSS problem into multiple static scheduling problems. However, the inherent essence of such an approach is still to view the dynamic problem as a static problem to be solved. This does not allow for timely response to changes in the scheduling environment and real-time decision-making, which ultimately affects the quality of the solution. Moreover, such an approach is not efficient. Several other studies employ heuristic selection RL methods. Unlike the application of heuristic selection RL on static/dynamic JSS, where only a sequencing rule is required, dynamic flexible JSS necessitates both a routing rule and a sequencing rule. To address the simultaneous challenges of routing and sequencing, some studies manually design composite scheduling heuristics as actions for RL and train an agent to select among these composite scheduling heuristics (Zhang et al. 2023; Chang et al. 2022; Luo 2020; Wu et al. 2023). However, the strategy of combining the two sub-problems is typically not optimal. It fails to fully capture the interaction between routing and sequencing rules (Zhang et al. 2018), potentially leading to a reduction in the search space and limiting the exploration of superior solutions.

In addition, some approaches leverage RL to learn the weights associated with these manually designed composite scheduling heuristics, resulting in a weighted scheduling heuristic (Gui et al. 2023). This method can be effective in adapting to dynamic and changeable environments, as it does not necessitate consideration of the impact of variations in the number of candidate machines/operations at different decision points. However, a notable limitation of this method is the pre-determination of the aggregation function, which solely relies on a weighted sum function. This predetermined choice still narrows down the search space by limiting the range of functions available. Additionally, no sufficient evidence is provided to explain why this particular predetermined aggregation function can yield good results. Different from the aforementioned studies utilising composite scheduling heuristics as actions to jointly address sequencing and routing decisions, a distinct training strategy is employed in Liu et al. (2022). Specifically, the sequencing rule and the routing rule are trained separately.

4.3 Quantitative comparison of GP and RL

Comparing GP and RL in their performance on various JSS problems is essential for gaining insights into their respective capabilities, limitations, and potential synergies. This understanding can facilitate the development of more effective scheduling solutions, benefiting both academic research and practical applications. Despite its importance, there have been relatively few studies that directly compare GP and RL. For those interested in making this comparison, the following guidelines can be considered for effectively evaluating GP and RL methods in the context of JSS:

- Benchmark Instances: Selecting appropriate and well-established benchmark instances is crucial. We recommend using both static and dynamic JSS benchmarks to cover a range of problem complexities, where the widely used benchmarks can be found in Table 4.
- Experiment Setting: It is important to standardise the training and test datasets across methods to ensure a fair comparison. For both GP and RL, care should be taken to use consistent splits and random seeds where applicable, as different splits may significantly impact performance.
- Performance Metrics: The choice of performance metrics is crucial for evaluating methods. While makespan is commonly used in JSS, other objectives, such as flow time or tardiness, may be more relevant depending on the application. In multi-objective settings, comparing GP and RL should involve metrics like hypervolume or inverted generational distance. Additionally, incorporating metrics that analyse the structures of the heuristics learned by GP and RL can provide a more comprehensive and holistic comparison.
- Result Discussion: When discussing results, it is beneficial to analyse both the quantitative and qualitative aspects. For instance, GP's interpretability and generalisation ability versus RL's adaptability to complex environments can provide insights into which method may be more suited for a particular problem.
- Pitfalls to Avoid: A common pitfall is the use of inconsistent lower bounds or evaluation measures across studies. Recalculating results when necessary and ensuring the comparability of metrics are essential to avoid misleading conclusions. Another pitfall is neglecting the impact of training time. It is crucial to either use the same or similar training durations or ensure both GP and RL methods have fully converged before comparing the learned heuristics for a fair evaluation.This survey conducts a quantitative comparison of GP and RL on static and dynamic JSS datasets as Table 6 to give a more concrete understanding of the relative performance of GP and RL under the same conditions. It is important to highlight that, while many GP and RL studies utilise the same benchmark datasets (as indicated in Table 4), direct comparisons are often hindered by variations in experiment design, training and test set splits, and performance evaluation methods. In our analysis, we identified GP^1 (Yska et al. 2018), GP^2 (Braune et al. 2022), GP^3 (Xu et al. 2024), as well as RL^1 (Yuan et al. 2024), RL^2 (Song et al. 2022), RL^3 (Lei et al. 2022), and RL^4 (Xu et al. 2024), noting that these methods are comparable as they used the same training and test sets. Importantly, we do not implement their methods but directly collect results from their respective papers. To ensure a fair comparison, we recalculate the makespan results, as some studies reported only the makespan gap relative to

Table 6 Quantitative comparison of GP and RL on static and dynamic JSS datasets

Instance type	Instance	Method	Objective value gap (Makespan)	References
Flexible JSS	Barnes (21 instances)	GP[1]	**6.5%**	Yska et al. (2018)
		RL[1]	13.83%	Yuan et al. (2024)
		RL[2]	17.88%	Song et al. (2022)
		RL[3]	28.96%	Lei et al. (2022)
	Brandimarte (10 instances)	GP[1]	**6.2%**	Yska et al. (2018)
		GP[1]	10.64%	Braune et al. (2022)
		RL[1]	13.24%	Yuan et al. (2024)
		RL[2]	30.04%	Song et al. (2022)
		RL[3]	13.59%	Lei et al. (2022)
	Dauzere (18 instances)	GP[1]	**6.1%**	Yska et al. (2018)
		RL[1]	11.02%	Yuan et al. (2024)
		RL[2]	8.88%	Song et al. (2022)
		RL[3]	15.68%	Lei et al. (2022)
Instance type	Instance	Method	Objective value (Tardiness)	References
Dynamic flexible JSS	HH (124 jobs)	GP[3]	**962.94(178.56)**	Xu et al. (2024)
		RL[4]	1170.68(37.13)	Xu et al. (2024)
	HL (124 jobs)	GP[3]	**389.28(130.55)**	Xu et al. (2024)
		RL[4]	617.22(66.95)	Xu et al. (2024)
	LH (124 jobs)	GP[3]	**1856.63(216.37)**	Xu et al. (2024)
		RL[4]	2134.25(54.61)	Xu et al. (2024)
	LL (124 jobs)	GP[3]	**770.37(102.55)**	Xu et al. (2024)
		RL[4]	1089.42(62.75)	Xu et al. (2024)

The bold represents the better performance on the related instance

a lower bound without providing the actual makespan or using different lower bounds. By recalculating the makespan, we ensure a consistent comparison of the makespan gap for various GP and RL methods on static flexible JSS datasets, and directly compared makespan performance on dynamic flexible JSS datasets.

From the table, we observe that GP methods outperform RL methods on both static and dynamic datasets, particularly on dynamic datasets. Additionally, as most of these studies do not report computation time, we do not include it in the comparison. Ideally, a fair comparison would involve ensuring that both GP and RL methods have similar or equal training times when learning heuristics/policies. Unfortunately, the lack of reported training times in these papers makes this impossible. Nonetheless, since both GP and RL involve a training process to learn heuristics/policies, the application of the learned heuristics/policies can operate in real time. As a result, comparing response times directly is less critical, and both GP and RL methods are capable of delivering real-time responses in practical applications.

5 Pros and cons of GP and RL

Based on the literature review, both GP and RL demonstrate their respective strengths and limitations in solving JSS problems. The pros and cons of GP and RL are summarised as follows.

5.1 Scalability

Scalability refers to the algorithm's ability to handle more complex problems or much larger scale instances and maintain its effectiveness (Paliouras 1993). GP is more frequently used for large-scale JSS instances (as shown in Table 4) and is generally considered more scalable. GP representations are not directly affected by the problem scale, allowing them to handle variations in the problem scale. Conversely, the scalability of RL in JSS is influenced by its representations. When RL is used to select among existing scheduling heuristics, it can maintain scalability as the core decision-making logic is independent of the problem scale. When RL is used to directly choose a specific job-machine pair, scalability becomes a concern. If RL uses feedforward neural networks, it can maintain good scalability for problems with a constant number of candidate machines/operations at each decision point, as the network size remains manageable regardless of the problem scale. On the other hand, RL approaches leveraging graph neural networks for making scheduling decision may face scalability issues as the problem size and complexity increase. This is because graph neural networks can become significantly complex and large, potentially hindering performance in large-scale JSS instances. In a word, for large-scale JSS, GP is often the preferred choice due to its inherent scalability. For small-scale JSS, both GP and RL can be viable options. The specific RL approach (heuristic selection vs. end-to-end) and network architecture (representations) will influence scalability. Moreover, further research on RL for large-scale JSS is worth exploring.

5.2 Generalisation ability

Generalisation ability refers to the capacity of an algorithm to perform well on unseen or new data, beyond the specific data or tasks it is trained on. In the context of GP and RL for JSS, generalisation ability refers to how well the learned heuristics/policies can adapt to new scenarios or changes in the problem environment, such as different job configurations or dynamic conditions.

Before assessing whether the learned heuristics/policies from GP and RL can perform well on unseen scenarios, the first consideration is whether they can be directly applied to different unseen scenarios without retraining. GP evolves scheduling heuristics, typically represented as tree structures or other flexible representations. These representations are not constrained by the number of candidate jobs or machines, meaning that the learned scheduling heuristics can be applied to new scheduling scenarios with varying configurations or dynamic conditions without the need for retraining. This gives GP a degree of versatility in handling unseen situations. In contrast, the generalisation ability of RL is highly dependent on the design and representation of its action space. When RL is employed for heuristic selection-where actions involve selecting predefined heuristics-the learned scheduling heuristics can adapt to new problem instances without requiring retraining, which enhances

RL's generalisation ability in these cases. However, for end-to-end RL, the generalisation ability varies. If RL is trained with a fixed action space (e.g., a set number of machines or operations), its ability to generalise is limited, as the learned scheduling policies could not necessarily adapt to different scenarios involving a different number of machines or operations. On the other hand, if the RL network is designed to dynamically adjust to changes in the environment, it exhibits better generalisation capabilities.

Regarding how well the learned heuristics/policies can perform on unseen scenarios, a fair comparison between GP and RL is essential. A study in Xu et al. (2024) directly compared scheduling heuristics/policies learned by a typical GP and RL model for solving a dynamic flexible JSS problem. This study, using the same dataset, demonstrated that the chosen GP offered better performance (generalisation ability) than the selected RL on this specific problem type. This survey also includes a comparison between GP and RL based on existing studies, revealing that without sophisticated design and/or fine tuning, GP methods tend to outperform RL methods in static Barnes, Brandimarte, and Dauzere scenarios and randomly generated dynamic flexible JSS scenarios, as discussed in Sect. 4.3. However, existing research on directly comparing GP and RL for generalisation in JSS is limited. While the current evidence suggests GP might have an edge in generalisation for specific JSS problems (Xu et al. 2024), more research is necessary for a definitive conclusion. Since the study in Xu et al. (2024) focuses on a single type of JSS problem (dynamic flexible JSS). Generalisation ability might differ for other JSS variants. Moreover, both GP and RL encompass various algorithms and configurations. Different implementations of GP and RL might provide different conclusions. In this case, comparing GP and RL across diverse JSS problem types and exploring various configurations and algorithms within both GP and RL are required to compare their generalisation ability.

5.3 Considered scheduling information

Regardless of whether used in end-to-end or heuristic selection way, RL tends to focus on local information. Following each action, a reward is assigned to indicate the quality of that action based on a predefined reward function. This reward function often differs from the original optimisation goal, as it can only assess partial information before completing the entire scheduling task. Therefore, the design of the reward function is critical in guiding the learning process. A well-designed reward function facilitates the discovery of effective scheduling heuristics/policies, whereas a poorly designed one might yield inferior results. Unlike RL, GP directly evaluates the scheduling heuristics based on a fitness function, typically directly reflecting the original optimisation goal. However, this global optimisation approach prioritises overall performance but might result in suboptimal decisions at individual decision points throughout the scheduling process. GP's emphasis on global information and global performance might lead to a loss of local search capability, potentially overlooking expert scheduling heuristics good for specific scenarios. Besides, for heuristic selection RL, utilising manually designed scheduling heuristics as actions rather than directly employing machines/operations can constrain the search space, potentially limiting the model's ability to explore alternative solutions and use more information.

5.4 Search mechanisms

RL focuses on iteratively adjusting and improving a single scheduling heuristic/policy, while GP optimises a population of scheduling heuristics simultaneously. Theoretically, both GP and RL can achieve optimal schedules if the training data is sufficiently large, the model structure is flexible enough, and the search space includes the optimal mapping from scheduling states to selected candidates (e.g., job or machine). However, in practice, achieving optimal solutions is challenging, especially in large-scale or complex JSS problems. GP is designed to efficiently learn scheduling heuristics that offer good-quality solutions, rather than guaranteeing optimality. It fosters diversity within the population of heuristics, allowing for the coexistence of highly effective and moderately effective heuristics, thus avoiding stagnation in local optima. RL, on the other hand, is designed with a focus on finding the optimal solution. However, RL (whether in heuristic selection or end-to-end) is typically only able to find optimal solutions in small-scale or simple JSS scenarios. In large-scale or dynamic JSS problems, though, RL tends to focus on finding high-quality solutions rather than exact optimal solutions.

5.5 Training time

GP's training process can be time-consuming due to the management of a population of scheduling heuristics. In contrast, RL focuses on adjusting and improving a single scheduling heuristic/policy (neural network) iteratively, and thus generally requires less training time compared to GP.

5.6 Interpretability

GP typically represents scheduling heuristics using tree-based or linear/graph structures, offering flexibility in automatically utilising linear or non-linear functions to construct the heuristics. These structures also facilitate interpretability, making it easier to understand the decision-making process compared to neural networks (Mei et al. 2022). The interpretability of RL is rather limited. The complex neural networks inherent in RL, characterised by numerous nodes and weights, often resemble black boxes, potentially undermining user confidence in using the learned scheduling heuristics/policies.

5.7 Degree of automated design

The degree of automated design refers to the level of automation involved in designing an algorithm. It essentially reflects how much human intervention is required in the algorithm-building process. For RL, different scenarios might necessitate specific network architectures and hyperparameter tuning, demanding significant effort and domain knowledge in algorithm design. Moreover, when RL is used as a heuristic selection method, an important consideration is to ensure that the manually designed scheduling heuristics as actions are adept at handling different decision points. In this case, the manual scheduling heuristics need to be carefully designed, and the process of designing these scheduling heuristics manually can be time-consuming and requires a large amount of domain knowledge. In addition, RL's effectiveness is affected by the design of the reward function. Changing the JSS objec-

Table 7 The pros and cons of GP and RL

Features	GP	RL
Scalability	Maintains good scalability across different scenarios	Affected by RL's roles and representations. Heuristic selection RL and end-to-end RL with feed-forward neural networks scale well, while graph neural network-based end-to-end RL has poor scalability
Generalisation sbility	Good generalisation ability to unseen instances	Depends on the action design. Good generalisation ability to unseen instances with heuristic selection RL and end-to-end RL that is designed to handle dynamically changing action spaces. Poor generalisation ability with end-to-end RL with fixed action spaces
Information focus	Focuses on global information and might loss local accuracy	Focuses on local information through reward feedback, but sensitive to reward function design
Search mechanisms	Explores diverse scheduling heuristics simultaneously using Darwinian selection strategies. Theoretically capable of finding the optimal solution when it is designed to consider all possible job-machine pairs without sacrificing the solution space, but quite challenging. Focus on learning scheduling heuristics that offer reasonably good quality in complex scenarios rather than optimal solution	Maintains and modifies a single scheduling heuristic/policy through trial-and-error interaction with the environment. Theoretically capable of finding the optimal solution when it is designed to consider all possible job-machine pairs without sacrificing the solution space, but quite challenging. Focus on learning scheduling heuristics/policies to find optimal solutions on small-scale and simple JSS instances. Focus on learning scheduling heuristics/policies that offer reasonably good quality in complex scenarios rather than optimal solution
Training time	More training time due to maintaining a population of scheduling heuristics	Less training time as it maintains only one scheduling heuristic/policy
Interpretability	Good interpretability as the structure of learned scheduling heuristics can be easily inspected	Poor interpretability as the structure of learned scheduling heuristics/policies have complex neural networks with many nodes and weights
Degree of automated design	High degree of automated design, requiring less human effort in algorithm design	Low degree of automated design due to the need for domain knowledge in reward function and neural network design

tives necessitates adjustments to the reward function, which can be time-consuming and requires domain expertise. Different from RL, GP directly evaluates scheduling heuristics based on a fitness function, typically aligned with the optimisation goal. This eliminates the need for complex reward function design in RL. Additionally, GP representations are well-suited for various JSS scenarios and do not require the establishment of an initial structure like neural networks, thereby reducing the overall effort involved. Generally, RL demands more design effort and domain knowledge compared to GP.

Based on the above discussion, Table 7 summarises the pros and cons of GP and RL.

6 Issues and future directions

Based on the above literature review and taking into account the strengths and limitations of both GP and RL in JSS, some issues and future research directions are provided. These directions encompass advancements in both GP and RL individually, as well as the cooperative enhancement achieved by their integration.

6.1 Training time reduction

Both GP and RL involve iterative learning processes, leading to considerable computational demands during training. While surrogate techniques have been employed in GP to mitigate training time in large-scale JSS problems, their effectiveness remains limited. Meanwhile, in multi-objective JSS domains where evaluation processes and Pareto dominance calculations are particularly time-consuming, there is a significant demand for strategies to reduce training time. However, the exploration of surrogates in multi-objective JSS remains limited. In addition, existing RL methods primarily target small-scale JSS problems (as shown in Table 4), where training time concerns are less pronounced. However, there is a growing demand for RL to address large-scale JSS problems closer to real-world applications, necessitating attention to training time issues. While it is feasible to train scheduling heuristics/policies using small-scale scenarios and apply them to large-scale scenarios, the performance might not meet the desired standards compared to direct training on large-scale scenarios. Therefore, there is a pressing requirement for a more focused exploration of strategies aimed at reducing training time in both GP and RL. The following are some potential future directions:

1. Efficient fitness evaluation: Propose more accurate and lightweight surrogate models, approximation methods, or pre-selection strategies to identify promising candidates for more thorough evaluation.
2. Smart initialisation and warm starting: Develop intelligent initialisation strategies to provide a good starting point for GP and RL can aid in quicker convergence. Warm-starting approaches, where the initial scheduling heuristics/policies are initialised with information from previous runs, or integrating transfer learning techniques to leverage domain knowledge from similar tasks, offer promising avenues to accelerate the convergence process.
3. Algorithmic improvements: Investigate algorithmic enhancements, such as more efficient crossover and mutation operators for GP and more efficient neural network updating strategies for RL, can contribute to faster convergence.

6.2 Interpretability

In learning scheduling heuristics/policies for JSS, interpretability encompasses two key aspects. Firstly, it is the interpretability of scheduling heuristic/policy structures. Understanding the decision-making processes of evolved scheduling heuristics/policies by GP and RL is important for their acceptance and practical implementation (Mei et al. 2022). When comparing the structures commonly used in GP and RL, GP generally offers superior interpretability due to its flexible representation, particularly with its tree-based structure. In con-

trast, the neural network structures employed in RL tend to be complex, resembling black boxes with numerous layers, nodes, and weights. Consequently, understanding the decision-making process of scheduling heuristics/policies, especially in RL, becomes challenging. Although GP provides better interpretability than RL, explaining the rationale behind the combination of functions and terminals remains non-trivial. Various studies have attempted to quantify interpretability using metrics such as heuristic sizes (Zhang et al. 2023), feature importance analysis (Mei et al. 2017), and grouping terminals into distinct categories (Zhang et al. 2023), contributing to advancements in interpretability. However, achieving an intuitive interpretation of scheduling heuristics/policies and providing user trust in their utilisation remains a challenge.

Secondly, the interpretability of the learning process holds significance. Understanding how GP and RL learn enables researchers to design more effective and efficient algorithms. By dissecting the learning process, researchers can identify bottlenecks, inefficiencies, and areas for improvement, leading to the development of more effective algorithms. In addition, understanding the learning mechanisms within GP and RL can enable the exchange of insights and techniques between them and across various problem domains.

Overall, advancements in interpretability hold the potential to address challenges in understanding scheduling heuristic/policy decision-making and designing more effective algorithms. The following are some potential future directions:

1. Explanation mechanisms for evolved heuristics/policies: Develop techniques that provide human-readable explanations for the decisions made by learned heuristics/policies. This could involve identifying critical features or decision points within the evolved heuristics/policies and translating them into understandable rules or patterns.
2. Visualisations: Create visualisation tools that can represent the decision-making process of learned heuristics/policies or the learning process in an intuitive and accessible manner. This could involve graphical representations of the scheduling logic or interactive interfaces that allow users to explore the behaviour of evolved heuristics/policies.
3. Human-in-the-loop approaches: Investigate approaches that involve human feedback in the interpretability process. This could include methods where users provide input on the interpretability of evolved scheduling heuristics/policies, helping to refine the evolved scheduling heuristics/policies iteratively.

6.3 Effective utilisation of multiple scheduling heuristics/policies

In practical scenarios, a single "best" heuristic/policy might not always be optimal for diverse and dynamic situations. Leveraging the diversity of evolved heuristics/policies has the potential to yield more robust and adaptable scheduling solutions. Several studies have demonstrated the effectiveness of leveraging multiple scheduling heuristics/policies through ensemble techniques (Xu et al. 2023; Gao et al. 2015; Park et al. 2015). These techniques enable joint decision-making by integrating several scheduling heuristics/policies through methods such as voting or weighted-sum approaches (Rodrigueset al. 2020). However, existing ensemble methods often maintain the same decision-making strategy across all decision points, requiring a deeper exploration in this domain. Future research should focus on developing advanced strategies and methods that fully exploit the diversity

and adaptability offered by multiple scheduling heuristics/policies evolved through GP and RL. The following are some potential future directions:

1. Ensemble learning strategies and decision fusion techniques: Develop advanced ensemble learning strategies that can intelligently select appropriate scheduling heuristics/policies and combine the outputs of multiple heuristics/policies. This could involve dynamic weighting, where the influence of each heuristic/policy is adjusted based on the characteristics of the current scheduling instance or its historical performance.
2. Context-aware adaptive scheduling mechanisms: Design mechanisms that dynamically select and adapt scheduling heuristics/policies based on the characteristics of the jobs, machines, or the historical performance of heuristics in specific contexts of decision points.

6.4 Advanced multi/many-objective optimisation

Advanced multi/many-objective optimisation in JSS by GP and RL represents a key topic with a lot of real-world applications. Existing studies involve combining multi-objective optimisation methods such as NSGAII, SPEA2, or MOEA/D with GP (Zhang et al. 2019; xu et al. 2023). However, the use of indicator-based multi-objective optimisation methods has rarely been explored in GP for multi-objective JSS problems. Additionally, research on RL for solving multi-objective JSS problems is limited. Hence, there is considerable room for further advancements in GP and RL for multi-objective JSS. The following are some potential future directions:

1. Hybrid approaches: Investigate hybrid approaches, including integrating indicator-based multi-objective optimisation methods with GP, integrating multi-objective optimisation methods with RL, and combining GP/RL with other machine learning techniques, such as surrogate or transfer learning, to further enhance its performance.
2. Preference-based multi-objective optimisation: Integrate preference-based methods into GP/RL for multi-objective optimisation. This involves incorporating human preferences into the optimisation process to guide the algorithm towards solutions that align with the decision-maker's subjective preferences.

6.5 Hybridisation of GP and RL

It is indeed notable that the number of RL publications has recently surpassed that of GP. However, this trend only started around 2022 for general JSS and 2023 for multi-objective JSS, which is quite recent. Prior to this, GP consistently had a greater number of publications than RL. Thus, overall, GP still holds a dominant position. As highlighted in Table 4, which compares the datasets of GP and RL publications, RL has primarily focused on static and small-scale JSS problems, whereas GP has been applied more to dynamic and large-scale JSS problems. These data further underscore GP's advantages over RL in specific contexts. One possible reason for the recent increase in RL publications is that its success in other domains has likely encouraged researchers to explore its potential in JSS. In our view, this is a positive trend, as expanding the application of RL to JSS can lead to further

advancements in this important field, which has numerous real-world applications. Also, there is a possibility of combining GP and RL to further contribute to this domain.

We believe that this growing interest in RL does not diminish the value of GP, but rather reflects an evolving research focus. RL is gaining attention due to its recent successes, but the strengths of GP, such as its interpretability, remain crucial, especially in applications where understanding the decision-making process is essential. This also presents an exciting research opportunity to explore the integration of GP and RL, leveraging the strengths of both methods. This collaboration opens up new possibilities for research and holds promise for addressing complex challenges in JSS more effectively.

To be specific, considering the strengths and limitations of GP and RL, integrating them to address JSS problems offers a promising approach. In Xu et al. (2024), a comparison of a typical GP and a typical RL method on dynamic flexible JSS is conducted, highlighting their respective strengths and weaknesses. Building upon the findings of (Xu et al. 2024) and this survey, future research could concentrate on developing more sophisticated hybridisation techniques that integrate GP and RL. The following are some potential future directions:

1. Considering both global and local feedback: Learn diverse scheduling heuristics by leveraging GP's exploration capabilities with global information. Then, further refine and optimise these heuristics over time based on RL's exploitation capability with local information.
2. Automatically learning RL actions with GP: Existing heuristic selection RL approaches often rely on manually designed scheduling heuristics as actions. However, the manual design of diverse scheduling heuristics is laborious and demands significant domain expertise. Exploring the potential of using GP to autonomously learn a set of high-quality and diverse scheduling heuristics as actions for RL represents a promising future direction. Two recent studies have explored this research direction by employing the learned scheduling heuristics by GP as actions for RL (Xu et al. 2024), as well as learning the weights through RL of each learned scheduling heuristic by GP in an ensemble (Chen et al. 2024).
3. Automatically adapting GP operators with RL: Several studies have demonstrated the potential of using RL to guide the selection and adjustment of mutation and crossover operators in evolutionary algorithms (Durgut et al. 2021; Zojaji and Kazemi 2022; Miguel Gomez and Toosi 2021; Sakurai et al. 2010; Zhang et al. 2024). Building on these insights, an RL agent can learn rules to dynamically adapt genetic operators and adjust search parameters for GP.
4. Attention mechanisms: Certain RL methods utilise attention mechanisms to prioritise important information while disregarding noise. GP can integrate similar mechanisms to selectively focus on dominant information and eliminate the influence of redundant or noisy data that could potentially degrade performance.
5. Population mechanisms: GP operates by maintaining and optimising a population of scheduling heuristics concurrently. These heuristics explore the search space from various regions simultaneously, providing diversity. RL can leverage this concept for optimising multiple heuristics/policies simultaneously, allowing them to reinforce each other and collectively contribute to improved performance and can avoid being stuck in local optima.

7 Conclusions

This paper offers an extensive survey of recent studies on scheduling heuristics/policies learning, focusing particularly on the comparison between GP and RL in various JSS problems. The contributions of this survey are multifaceted.

Firstly, it offers in-depth discussions comparing key components of GP and RL for JSS, including representation, evaluation, and search mechanisms. This comprehensive analysis aids researchers, particularly beginners in the field, in gaining a solid understanding of these core components and offers guidance for designing GP/RL methods according to their specific requirements. Secondly, by encompassing recent publications on GP and RL across various JSS problems, this survey serves as a guide for researchers seeking to choose appropriate GP or RL techniques for addressing specific JSS challenges. Thirdly, this survey consolidates the strengths and weaknesses of GP and RL methods in JSS, providing valuable insights into heuristic/policy learning. Lastly, this survey highlights numerous promising avenues for future research, which are instrumental in advancing the fields of both GP and RL within JSS. These identified directions hold the potential to significantly contribute to ongoing development and innovation in the field. In addition, this survey suggests a potential future direction that combines GP and RL approaches to complement their pros and cons. This collaboration opens up new possibilities for research and holds promise for addressing complex challenges in JSS more effectively.

Author contributions Meng Xu drafted the main manuscript text. Yi Mei, Fangfang Zhang, and Mengjie Zhang critically reviewed and revised the manuscript. All authors have read and approved the final version of the manuscript.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions
No funding was received to assist with the preparation of this manuscript.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest All authors certify that they have no affiliations with or involvement in any organisation or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

Adaloglou N, Karagiannakos S (2020) How attention works in deep learning: understanding the attention mechanism in sequence models. https://theaisummer.com/

Aissani N, Bekrar A, Trentesaux D, Beldjilali B (2012) Dynamic scheduling for multi-site companies: a decisional approach based on reinforcement multi-agent learning. J Intell Manuf 23:2513–2529
Artigues C, Gendreau M, Rousseau L-M, Vergnaud A (2009) Solving an integrated employee timetabling and job-shop scheduling problem via hybrid branch-and-bound. Comput Operat Res 36(8):2330–2340
Ashour S, Hiremath S (1973) A branch-and-bound approach to the job-shop scheduling problem. Int J Prod Res 11(1):47–58
Aydin ME, Fogarty TC (2004) A simulated annealing algorithm for multi-agent systems: a job-shop scheduling application. J Intell Manuf 15:805–814
Azem S, Aggoune R, Dauzère-Pérès S (2007) Disjunctive and time-indexed formulations for non-preemptive job shop scheduling with resource availability constraints. In: Proceedings of the ieee international conference on industrial engineering and engineering management, pp 787–791
Bahdanau D, Cho K, Bengio Y (2014) Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473
Bertsimas D, Tsitsiklis J (1993) Simulated annealing. Stat Sci 8(1):10–15
Bianchi RA, Ribeiro CH, Costa AH (2008) Accelerating autonomous learning by using heuristic selection of actions. J Heuristics 14:135–168
Bonetta G, Zago D, Cancelliere R, Grosso A (2023) Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. In: Proceedings of the international conference on learning and intelligent optimization, pp 475–490
Bouazza W, Sallez Y, Beldjilali B (2017) A distributed approach solving partially flexible job-shop scheduling problem with a q-learning effect. IFAC-PapersOnLine 50(1):15890–15895
Branke J, Nguyen S, Pickardt CW, Zhang M (2015) Automated design of production scheduling heuristics: a review. IEEE Trans Evol Comput 20(1):110–124
Braune R, Benda F, Doerner KF, Hartl RF (2022) A genetic programming learning approach to generate dispatching rules for flexible shop scheduling problems. Int J Prod Econ 243:108342
Brizuela CA, Sannomiya N (1999) A diversity study in genetic algorithms for job shop scheduling problems. In: Proceedings of the genetic and evolutionary computation conference, pp 75–82
Burke EK, Hyde MR, Kendall G, Woodward J (2007) Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In: Proceedings of the genetic and evolutionary computation conference, pp 1559–1565
Burke EK, 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 64:1695–1724
Chand S, Huynh Q, Singh H, Ray T, Wagner M (2018) On the use of genetic programming to evolve priority rules for resource constrained project scheduling problems. Inf Sci 432:146–163
Chang J, Yu D, Hu Y, He W, Yu H (2022) Deep reinforcement learning for dynamic flexible job shop scheduling with random job arrival. Processes 10(4):760
Chen H, Chu C, Proth J-M (1998) An improvement of the lagrangean relaxation approach for job shop scheduling: a dynamic programming method. IEEE Trans Robot Autom 14(5):786–795
Chen R, Yang B, Li S, Wang S (2020) A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem. Comput Ind Eng 149:106778
Chen H, Ding G, Qin S, Zhang J (2021) A hyper-heuristic based ensemble genetic programming approach for stochastic resource constrained project scheduling problem. Expert Syst Appl 167:114174
Chen R, Li W, Yang H (2022) A deep reinforcement learning framework based on an attention mechanism and disjunctive graph embedding for the job-shop scheduling problem. IEEE Trans Ind Inf 19(2):1322–1331
Chen X, Bai R, Qu R, Dong J, Jin Y (2024) Deep reinforcement learning assisted genetic programming ensemble hyper-heuristics for dynamic scheduling of container port trucks. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2024.3381042
Chen C, Ji Z, Wang Y (2018) Nsga-ii applied to dynamic flexible job shop scheduling problems with machine breakdown. Modern Phys Lett B 32:1840111
Chong CS, Low MYH, Sivakumar AI, Gay KL (2006) A bee colony optimization algorithm to job shop scheduling. In: Proceedings of the winter simulation conference, pp 1954–1961
Cunha B, Madureira AM, Fonseca B, Coelho D (2020) Deep reinforcement learning as a job shop scheduling solver: a literature review. In: Proceedings of the international conference on hybrid intelligent systems, pp 350–359
Dimopoulos C, Zalzala AM (2001) Investigating the use of genetic programming for a classic one-machine scheduling problem. Adv Eng Softw 32(6):489–498
Ding L, Guan Z, Rauf M, Yue L (2024) Multi-policy deep reinforcement learning for multi-objective multiplicity flexible job shop scheduling. Swarm Evol Comput 87:101550
Dong Z, Ren T, Qi F, Weng J, Bai D, Yang J, Wu C-C (2024) A reinforcement learning-based approach for solving multi-agent job shop scheduling problem. Int J Product Res, 1–26

Du Y, Li J (2024) A deep reinforcement learning based algorithm for a distributed precast concrete production scheduling. Int J Prod Econ 268:109102
Du Y, Li J, Chen X, Duan P, Pan Q (2022) Knowledge-based reinforcement learning and estimation of distribution algorithm for flexible job shop scheduling problem. IEEE Trans Emerg Top Comput Intell 7(4):1036–1050
Du Y, Li J, Li C, Duan P (2022) A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. IEEE Trans Neural Netw Learn Syst
Đumić M, Jakobović D (2021) Ensembles of priority rules for resource constrained project scheduling problem. Appl Soft Comput 110:107606
Đurasević M, Jakobović D (2018) Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. Genet Program Evolvable Mach 19(1):9–51
Đurasević M, Jakobović D (2018) Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. Genet Program Evolvable Mach 19:53–92
Đurasević M, Jakobović D (2020) Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment. Appl Soft Comput 96:106637
Đurasević M, Jakobović D, Knežević K (2016) Adaptive scheduling on unrelated machines with genetic programming. Appl Soft Comput 48:419–430
Đurasević M, Gil-Gala FJ, Planinić L, Jakobović D (2023) Collaboration methods for ensembles of dispatching rules for the dynamic unrelated machines environment. Eng Appl Artif Intell 122:106096
Đurasević M, Gala FJG, Jakobović D, Coello CAC (2023) Combining single objective dispatching rules into multi-objective ensembles for the dynamic unrelated machines environment. Swarm Evol Comput 80:101318
Durgut R, Aydin ME, Atli I (2021) Adaptive operator selection with reinforcement learning. Inf Sci 581:773–790
Echeverria I, Murua M, Santana R (2024) Offline reinforcement learning for job-shop scheduling problems. arXiv preprint arXiv:2410.15714
Echeverria I, Murua M, Santana R (2024) Solving the flexible job-shop scheduling problem through an enhanced deep reinforcement learning approach
Engin O, Güçlü A (2018) A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. Appl Soft Comput 72:166–176
Fan H, Xiong H, Goh M (2021) Genetic programming-based hyper-heuristic approach for solving dynamic job shop scheduling problem with extended technical precedence constraints. Comput Oper Res 134:105401
Fang Y, Li J (2010) A review of tournament selection in genetic programming. In: Proceedings of the international symposium on intelligence computation and applications, pp 181–192
Fattahi P, Messi Bidgoli M, Samouei P (2018) An improved tabu search algorithm for job shop scheduling problem trough hybrid solution representations. J Qual Eng Prod Optim 3(1):13–26
Fontes DB, Homayouni SM, Gonçalves JF (2023) A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. Eur J Oper Res 306(3):1140–1157
Gao KZ, Suganthan PN, Tasgetiren MF, Pan QK, Sun QQ (2015) Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion. Comput Ind Eng 90:107–117
Gao K, Cao Z, Zhang L, Chen Z, Han Y, Pan Q (2019) A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. IEEE/CAA J Automat Sin 6(4):904–916
Gil Gala FJ, Mencía C, Sierra MR, Varela R (2019) Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time. Appl Soft Comput 85:105782
Gil Gala FJ, Sierra MR, Mencía C, Varela R (2021) Genetic programming with local search to evolve priority rules for scheduling jobs on a machine with time-varying capacity. Swarm Evol Comput 66:100944
Gil Gala FJ, Sierra MR, Mencía C, Varela R (2022) Combining hyper-heuristics to evolve ensembles of priority rules for on-line scheduling. Nat Comput 21(4):553–563
Gil-Gala FJ, Sierra MR, Mencía C, Varela R (2023) Surrogate model for memetic genetic programming with application to the one machine scheduling problem with time-varying capacity. Expert Syst Appl 233:120916
Glover F, Laguna M (1998) Tabu search. Springer, Boston, pp 2093–2229
Gomes MC, Barbosa-Póvoa AP, Novais AQ (2013) Reactive scheduling in a make-to-order flexible job shop with re-entrant process and assembly: a mathematical programming approach. Int J Prod Res 51(17):5120–5141
Gonçalves JF, Magalhães Mendes JJ, Resende MG (2005) A hybrid genetic algorithm for the job shop scheduling problem. Eur J Oper Res 167(1):77–95
Gromicho JA, Van Hoorn JJ, Saldanha-da-Gama F, Timmer GT (2012) Solving the job-shop scheduling problem optimally by dynamic programming. Comput Operat Res 39(12):2968–2977

Gui Y, Tang D, Zhu H, Zhang Y, Zhang Z (2023) Dynamic scheduling for flexible job shop using a deep reinforcement learning approach. Comput Ind Eng 180:109255. https://doi.org/10.1016/j.cie.2023.109255
Hameed MSA, Schwung A (2020) Reinforcement learning on job shop scheduling problems using graph networks. arXiv preprint arXiv:2009.03836
Hart E, Sim K (2016) A hyper-heuristic ensemble method for static job-shop scheduling. Evol Comput 24(4):609–635
He Z, Tran KP, Thomassey S, Zeng X, Xu J, Yi C (2022) Multi-objective optimization of the textile manufacturing process using deep-q-network based multi-agent reinforcement learning. J Manuf Syst 62:939–949
Heinonen J, Pettersson F (2007) Hybrid ant colony optimization and visibility studies applied to a job-shop scheduling problem. Appl Math Comput 187(2):989–998
Helmuth T, Abdelhady A (2020) Benchmarking parent selection for program synthesis by genetic programming. In: Proceedings of the genetic and evolutionary computation conference companion, pp 237–238
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 genetic and evolutionary computation conference, pp 257–264
Ho KH, Cheng JY, Wu JH, Chiang F, Chen YC, Wu YY, Wu IC (2024) Residual scheduling: a new reinforcement learning approach to solving job shop scheduling problem. IEEE Access
Ho NB, Tay JC (2005) Evolving dispatching rules for solving the flexible job-shop problem. Proc IEEE Congress Evol Comput 3:2848–2855
Holthaus O, Rajendran C (1997) Efficient dispatching rules for scheduling in a job shop. Int J Prod Econ 48(1):87–105
Ho K, Wu J, Chiang F, Wu Y, Chen S, Kuo T, Wang F, Wu I (2023) Deep reinforcement learning based on graph neural networks for job-shop scheduling. In: Proceedings of the international conference on consumer electronics-Taiwan, pp 805–806
Huang K, Liao C (2008) Ant colony optimization combined with taboo search for the job shop scheduling problem. Comput Operat Res 35(4):1030–1046
Huang RH, Yu TH (2017) An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting. Appl Soft Comput 57:642–656
Huang J, Gao L, Li X, Zhang C (2023) A novel priority dispatch rule generation method based on graph neural network and reinforcement learning for distributed job-shop scheduling. J Manuf Syst 69:119–134
Huang Z, Mei Y, Zhang F, Zhang M (2023) Multitask linear genetic programming with shared individuals and its application to dynamic job shop scheduling. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2023.3263871
Huang JP, Gao L, Li XY, Zhang C-J (2023) A cooperative hierarchical deep reinforcement learning based multi-agent method for distributed job shop scheduling problem with random job arrivals. Comput Ind Eng 185:109650
Huang J, Gao L, Li X (2024) An end-to-end deep reinforcement learning method based on graph neural network for distributed job-shop scheduling problem. Expert Syst Appl 238:121756
Huang JP, Gao L, Li XY (2024) A hierarchical multi-action deep reinforcement learning method for dynamic distributed job-shop scheduling problem with job arrivals. IEEE Trans Autom Sci Eng. https://doi.org/10.1109/TASE.2024.3380644
Huang Z, Mei Y, Zhang M (2021) Investigation of linear genetic programming for dynamic job shop scheduling. In: Proceedings of the IEEE symposium series on computational intelligence, pp 1–8
Huang Z, Mei Y, Zhang F, Zhang M (2023) Grammar-guided linear genetic programming for dynamic job shop scheduling. In: Proceedings of the genetic and evolutionary computation conference, pp 1137–1145
Hubbs CD, Li C, Sahinidis NV, Grossmann IE, Wassick JM (2020) A deep reinforcement learning approach for chemical production scheduling. Comput Chem Eng 141:106982
Hu H, Jia X, He Q, Fu S, Liu K (2020) Deep reinforcement learning based agvs real-time scheduling with mixed rule for flexible shop floor in industry 4.0. Comput Ind Eng 149:106749
Hunt RJ, Johnston MR, Zhang M (2016) Evolving dispatching rules with greater understandability for dynamic job shop scheduling. Citeseer
Hunt R, Johnston M, Zhang M (2014) Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In: Proceedings of the genetic and evolutionary computation conference, pp 927–934
Ingimundardottir H, Runarsson TP (2018) Discovering dispatching rules from data using imitation learning: a case study for the job-shop problem. J Sched 21:413–428
Jaklinović K, Đurasević M, Jakobović D (2021) Designing dispatching rules with genetic programming for the unrelated machines environment with constraints. Expert Syst Appl 172:114548
Jakobović D, Budin L (2006) Dynamic scheduling with genetic programming. In: Proceedings of the European conference on genetic programming, pp 73–84

Jing X, Yao X, Liu M, Zhou J (2024) Multi-agent reinforcement learning based on graph convolutional network for flexible job shop scheduling. J Intell Manuf 35(1):75–93
Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. J Artif Intell Res 4:237–285
Karunakaran D (2019) Active learning methods for dynamic job shop scheduling using genetic programming under uncertain environment. PhD thesis, Open Access Te Herenga Waka-Victoria University of Wellington
Karunakaran D, Chen G, Zhang M (2016) Parallel multi-objective job shop scheduling using genetic programming. In: Proceedings of the Australasian Conference on Artificial Life and Computational Intelligence, pp 234–245
Karunakaran D, Mei Y, Chen G, Zhang M (2017) Evolving dispatching rules for dynamic job shop scheduling with uncertain processing times. In: Proceedings of the IEEE congress on evolutionary computation, pp 364–371
Karunakaran D, Mei Y, Chen G, Zhang M (2017) Toward evolving dispatching rules for dynamic job shop scheduling under uncertainty. In: Proceedings of the genetic and evolutionary computation conference, pp 282–289
Karunakaran D, Mei Y, Chen G, Zhang M (2018) Sampling heuristics for multi-objective dynamic job shop scheduling using island based parallel genetic programming. In: Proceedings of the international conference on parallel problem solving from nature, pp 347–359
Kayhan BM, Yildiz G (2023) Reinforcement learning applications to machine scheduling problems: a comprehensive literature review. J Intell Manuf 34(3):905–929
Koza JR, Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection vol. 1,
Koza JR et al (1994) Genetic programming II. MIT press, Cambridge
Kuhnle A, Röhrig N, Lanza G (2019) Autonomous order dispatching in the semiconductor industry using reinforcement learning. Procedia CIRP 79:391–396
Lee JH, Kim HJ (2022) Imitation learning for real-time job shop scheduling using graph-based representation. In: Proceedings of the winter simulation conference, pp 3285–3296
Lee J, Kee S, Janakiram M, Runger G (2024) Attention-based reinforcement learning for combinatorial optimization: application to job shop scheduling problem. arXiv preprint arXiv:2401.16580
Lei K, Guo P, Zhao W, Wang Y, Qian L, Meng X, Tang L (2022) A multi-action deep reinforcement learning framework for flexible job-shop scheduling problem. Expert Syst Appl 205:117796. https://doi.org/10.1016/j.eswa.2022.117796
Lei K, Guo P, Wang Y, Zhang J, Meng X, Qian L (2023) Large-scale dynamic scheduling for flexible job-shop with random arrivals of new jobs by hierarchical reinforcement learning. IEEE Trans Ind Inf. https://doi.org/10.1109/TII.2023.3272661
Lei Y, Deng Q, Liao M, Gao S (2024) Deep reinforcement learning for dynamic distributed job shop scheduling problem with transfers. Expert Syst Appl 251:123970
Li Y (2017) Deep reinforcement learning: an overview. ArXiv Preprint ArXiv:1701.07274
Li Y, Gu W, Yuan M, Tang Y (2022) Real-time data-driven dynamic scheduling for flexible job shop with insufficient transportation resources using hybrid deep q network. Robot Comput-Integrat Manuf 74:102283
Lian Z, Jiao B, Gu X (2006) A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. Appl Math Comput 183(2):1008–1017
Lin TL, Horng SJ, Kao TW, Chen YH, Run RS, Chen RJ, Lai JL, Kuo IH (2010) An efficient job-shop scheduling algorithm based on particle swarm optimization. Expert Syst Appl 37(3):2629–2636
Lin CC, Deng DJ, Chih YL, Chiu HT (2019) Smart manufacturing scheduling with edge computing using multiclass deep q network. IEEE Trans Ind Inf 15(7):4276–4284
Liu CL, Huang TH (2023) Dynamic job-shop scheduling problems using graph neural network and deep reinforcement learning. IEEE transactions on systems, man, and cybernetics: systems
Liu CL, Chang CC, Tseng CJ (2020) Actor-critic deep reinforcement learning for solving job shop scheduling problems. IEEE Access 8:71752–71762
Liu Z, Wang Y, Liang X, Ma Y, Feng Y, Cheng G, Liu Z (2022) A graph neural networks-based deep q-learning approach for job shop scheduling problems in traffic management. Inf Sci 607:1211–1223
Liu R, Piplani R, Toro C (2022) Deep reinforcement learning for dynamic scheduling of a flexible job shop. Int J Prod Res 60(13):4049–4069
Liu Z, Mao H, Sa G, Liu H, Tan J (2024) Dynamic job-shop scheduling using graph reinforcement learning with auxiliary strategy. J Manuf Syst 73:1–18
Luo S (2020) Dynamic scheduling for flexible job shop with new job insertions by deep reinforcement learning. Appl Soft Comput 91:106208
Luo S, Zhang L, Fan Y (2021) Dynamic multi-objective scheduling for flexible job shop by deep reinforcement learning. Comput Ind Eng 159:107489

Luo S, Zhang L, Fan Y (2021) Real-time scheduling for dynamic partial-no-wait multiobjective flexible job shop by deep reinforcement learning. In: IEEE transactions on automation science and engineering
Masood A, Chen G, Mei Y, Al-Sahaf H, Zhang M (2019) Genetic programming with pareto local search for many-objective job shop scheduling. In: Proceedings of the Australasian joint conference on artificial intelligence, pp 536–548
Masood A, Chen G, Mei Y, Zhang M (2018) Reference point adaption method for genetic programming hyper-heuristic in many-objective job shop scheduling. In: Proceedings of the European conference on evolutionary computation in combinatorial optimization, pp 116–131
Masood A, Mei Y, Chen G, Zhang M (2016) Many-objective genetic programming for job-shop scheduling. In: Proceedings of the IEEE congress on evolutionary computation, pp 209–216
Mattfeld DC, Bierwirth C (2004) An efficient genetic algorithm for job shop scheduling with tardiness objectives. Eur J Oper Res 155(3):616–630
Mei Y, Nguyen S, Xue B, Zhang M (2017) An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. IEEE Trans Emerg Top Comput Intell 1(5):339–353
Mei Y, Chen Q, Lensen A, Xue B, Zhang M (2022) Explainable artificial intelligence by genetic programming: a survey. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2022.3225509
Mei Y, Nguyen S, Zhang M (2017) Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling. In: Proceedings of the Asia-Pacific conference on simulated evolution and learning, pp 435–447 . https://doi.org/10.1007/978-3-319-68759-9_36
Mei Y, Nguyen S, Zhang M (2017) Evolving time-invariant dispatching rules in job shop scheduling with genetic programming. In: Proceedings of the European conference on genetic programming, pp 147–163
Mei Y, Zhang M (2016) A comprehensive analysis on reusability of gp-evolved job shop dispatching rules. In: Proceedings of the IEEE congress on evolutionary computation, pp 3590–3597
Mei Y, Zhang M, Nyugen S (2016) Feature selection in evolving job shop dispatching rules with genetic programming. In: Proceedings of the genetic and evolutionary computation conference, pp 365–372
Miguel Gomez A, Toosi FG (2021) Continuous parameter control in genetic algorithms using policy gradient reinforcement learning. In: Proceedings of the international joint conference on computational intelligence, pp 115–122
Miyashita K (2000) Job-shop scheduling with genetic programming. In: Proceedings of the genetic and evolutionary computation conference, pp 505–512
Monaci M, Agasucci V, Grani G (2021) An actor-critic algorithm with deep double recurrent agents to solve the job shop scheduling problem. arXiv preprint arXiv:2110.09076
Nguyen S, Zhang M, Johnston M, Tan KC (2012) A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. IEEE Trans Evol Comput 17(5):621–639
Nguyen S, Zhang M, Johnston M, Tan KC (2013) Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. IEEE Trans Evol Comput 18(2):193–208
Nguyen S, Zhang M, Johnston M, Tan KC (2013) Learning iterative dispatching rules for job shop scheduling with genetic programming. Int J Adv Manuf Technol 67(1–4):85–100
Nguyen S, Zhang M, Johnston M, Tan KC (2014) Automatic programming via iterated local search for dynamic job shop scheduling. IEEE Trans Cybern 45(1):1–14
Nguyen S, Zhang M, Tan KC (2016) Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules. IEEE Trans Cybern 47(9):2951–2965
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, Mei Y, Xue B, Zhang M (2019) A hybrid genetic programming algorithm for automated design of dispatching rules. Evol Comput 27(3):467–496
Nguyen S, Zhang M, Johnston M, Tan KC (2012) A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In: Proceedings of the IEEE congress on evolutionary computation, pp 1–8
Nguyen S, Zhang M, Johnston M, Tan KC (2013) Dynamic multi-objective job shop scheduling: a genetic programming approach. Automated scheduling and planning: from theory to practice, 251–282
Nguyen S, Zhang M, Johnston M, Tan KC (2014) Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In: Proceedings of the Asia-Pacific conference on simulated evolution and Learning, pp 656–667
Nguyen S, Zhang M, Johnston M, Tan KC (2019) Genetic programming for job shop scheduling. Evol Swarm Intell Algor, 143–167
Nguyen S, Zhang M, Tan KC (2015) Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems. In: Proceedings of the IEEE congress on evolutionary computation, pp 2781–2788

Nguyen S, Zhang M, Tan KC (2018) Adaptive charting genetic programming for dynamic flexible job shop scheduling. In: Proceedings of the genetic and evolutionary computation conference, pp 1159–1166
Nie L, Gao L, Li P, Li X (2013) A gep-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. J Intell Manuf 24(4):763–774
Orhean AI, Pop F, Raicu I (2018) New scheduling approach using reinforcement learning for heterogeneous distributed systems. J Parallel Distrib Comput 117:292–302
Ouelhadj D, Petrovic S (2009) A survey of dynamic scheduling in manufacturing systems. J Sched 12(4):417–431
Ozolins A (2020) Bounded dynamic programming algorithm for the job shop problem with sequence dependent setup times. Oper Res Int J 20(3):1701–1728
Ozturk G, Bahadir O, Teymourifar A (2019) Extracting priority rules for dynamic multi-objective flexible job shop scheduling problems using gene expression programming. Int J Prod Res 57(10):3121–3137
Paliouras G (1993) Scalability of machine learning algorithms. PhD thesis, University of Manchester
Pan Z, Wang L, Wang J, Lu J (2021) Deep reinforcement learning based optimization algorithm for permutation flow-shop scheduling. In: IEEE transactions on emerging topics in computational intelligence
Park IB, Park J (2021) Scalable scheduling of semiconductor packaging facilities using deep reinforcement learning. IEEE Trans Cybern 53(6):3518–3531
Park J, Mei Y, Nguyen S, Chen G, Zhang M (2018) An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling. Appl Soft Comput 63:72–86
Park IB, Huh J, Kim J, Park J (2019) A reinforcement learning approach to robust scheduling of semiconductor manufacturing facilities. IEEE Trans Autom Sci Eng 17(3):1420–1431
Park J, Chun J, Kim SH, Kim Y, Park J (2021) Learning to schedule job-shop problems: representation and policy learning using graph neural network and reinforcement learning. Int J Prod Res 59(11):3360–3377
Park J, Bakhtiyar S, Park J (2021) Schedulenet: Learn to solve multi-agent scheduling problems with reinforcement learning. arXiv preprint arXiv:2106.03051
Park J, Mei Y, Chen G, Zhang M (2016) Niching genetic programming based hyper-heuristic approach to dynamic job shop scheduling: an investigation into distance metrics. In: Proceedings of the genetic and evolutionary computation conference companion, pp 109–110
Park J, Mei Y, Nguyen S, Chen G, Johnston M, Zhang M (2016) Genetic programming based hyper-heuristics for dynamic job shop scheduling: cooperative coevolutionary approaches. In: Proceedings of the European conference on genetic programming, pp 115–132
Park J, Mei Y, Nguyen S, Chen G, Zhang M (2017) Investigating the generality of genetic programming based hyper-heuristic approach to dynamic job shop scheduling with machine breakdown. In: Proceedings of the Australasian conference on artificial life and computational intelligence, pp 301–313
Park J, Mei Y, Nguyen S, Chen G, Zhang M (2018) Evolutionary multitask optimisation for dynamic job shop scheduling using niched genetic programming. In: Proceedings of the Australasian joint conference on artificial intelligence, pp 739–751
Park J, Mei Y, Nguyen S, Chen G, Zhang M (2018) Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling. In: Proceedings of the European conference on genetic programming, pp 253–270
Park J, Nguyen S, Zhang M, Johnston M (2015) A single population genetic programming based ensemble learning approach to job shop scheduling. In: Proceedings of the genetic and evolutionary computation conference, pp 1451–1452
Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. Comput Oper Res 35(10):3202–3212
Planinić L, Durasević M, Jakobović D (2021) On the application of ϵ-lexicase selection in the generation of dispatching rules. In: Proceedings of the IEEE congress on evolutionary computation, pp 2125–2132
Potts CN, Van Wassenhove LN (1985) A branch and bound algorithm for the total weighted tardiness problem. Oper Res 33(2):363–377
Pu Y, Li F, Rahimifard S (2024) Multi-agent reinforcement learning for job shop scheduling in dynamic environments. Sustainability 16(8):3234
Puiseau C, Meyes R, Meisen T (2022) On reliability of reinforcement learning based production scheduling systems: a comparative survey. J Intell Manuf 33(4):911–927
Rodrigues NM, Batista JE, Silva S (2020) Ensemble genetic programming. In: Proceedings of the European conference on genetic programming, pp 151–166
Rummukainen H, Nurminen JK (2019) Practical reinforcement learning-experiences in lot scheduling application. IFAC-PapersOnLine 52(13):1415–1420

Said NE-DA, Samaha Y, Azab E, Shihata LA, Mashaly M (2021) An online reinforcement learning approach for solving the dynamic flexible job-shop scheduling problem for multiple products and constraints. In: Proceedings of the international conference on computational science and computational intelligence, pp 134–139
Saidi-Mehrabad M, Fattahi P (2007) Flexible job shop scheduling with tabu search algorithms. Int J Adv Manuf Technol 32:563–570
Sakurai Y, Takada K, Kawabe T, Tsuruta S (2010) A method to control parameters of evolutionary algorithms by using reinforcement learning. In: Proceedings of the international conference on signal image technology and internet based systems, pp 74–79
Sarin SC, Ahn S, Bishop AB (1988) An improved branching scheme for the branch and bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime. Int J Prod Res 26(7):1183–1191
Serrano Ruiz JC, Mula J, Poler R (2024) Job shop smart manufacturing scheduling by deep reinforcement learning. J Ind Inf Integr 38:100582
Shady S, Kaihara T, Fujii N, Kokuryo D (2022) A novel feature selection for evolving compact dispatching rules using genetic programming for dynamic job shop scheduling. Int J Prod Res 60(13):4025–4048
Shady S, Kaihara T, Fujii N, Kokuryo D (2023) Feature selection approach for evolving reactive scheduling policies for dynamic job shop scheduling problem using gene expression programming. Int J Prod Res 61(15):5029–5052
Shanker K, Tzen YJJ (1985) A loading and dispatching problem in a random flexible manufacturing system. Int J Prod Res 23(3):579–595
Sharma P, Jain A (2015) Stochastic dynamic job shop scheduling with sequence-dependent setup times: simulation experimentation. J Eng Technol 5(1):19
Sitahong A, Yuan Y, Li M, Ma J, Ba Z, Lu Y (2022) Designing dispatching rules via novel genetic programming with feature selection in dynamic job-shop scheduling. Processes 11(1):65
Song HB, Lin J (2021) A genetic programming hyper-heuristic for the distributed assembly permutation flow-shop scheduling problem with sequence dependent setup times. Swarm Evol Comput 60:100807
Song W, Chen X, Li Q, Cao Z (2022) Flexible job shop scheduling via graph neural network and deep reinforcement learning. IEEE Trans Ind Inform
Stricker N, Kuhnle A, Sturm R, Friess S (2018) Reinforcement learning for adaptive order dispatching in the semiconductor industry. CIRP Ann 67(1):511–514
Tassel P, Gebser M, Schekotihin K (2023) An end-to-end reinforcement learning approach for job-shop scheduling problems based on constraint programming. arXiv preprint arXiv:2306.05747
Tay JC, Ho NB (2008) Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput Ind Eng 54(3):453–473
Teymourifar A, Ozturk G, Ozturk ZK, Bahadir O (2020) Extracting new dispatching rules for multi-objective dynamic flexible job shop scheduling with limited buffer spaces. Cogn Comput 12(1):195–205
Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence, vol. 30
Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. Adv Neural Inform Process Syst, 30
Vilcot G, Billaut J-C (2011) A tabu search algorithm for solving a multicriteria flexible job shop scheduling problem. Int J Prod Res 49(23):6963–6980
Wan L, Fu L, Li C, Li K (2024) Flexible job shop scheduling via deep reinforcement learning with meta-path-based heterogeneous graph neural network. Knowl Based Syst 296:111940
Wan L, Cui X, Zhao H, Fu L, Li C (2024) A novel method for solving dynamic flexible job-shop scheduling problem via differmer and deep reinforcement learning. Comput Ind Eng 198:110688
Wang YF (2020) Adaptive job shop scheduling strategy based on weighted q-learning algorithm. J Intell Manuf 31(2):417–432
Wang L, Pan Z (2021) Scheduling optimization for flow-shop based on deep reinforcement learning and iterative greedy method. Control Decis 36(11):2609–2617
Wang H, Yan H (2016) An interoperable adaptive scheduling strategy for knowledgeable manufacturing based on smgwq-learning. J Intell Manuf 27:1085–1095
Wang S, Zhang H (2023) A matheuristic for flowshop scheduling with batch processing machines in textile manufacturing. Appl Soft Comput 145:110594
Wang L, Pan Z, Wang J (2021) A review of reinforcement learning based intelligent optimization for manufacturing scheduling. Complex Syst Model Simul 1(4):257–270
Wang X, Lin Z, Lei R, Xie K, Wang K, Fei Y, Zhen C (2022) Brief review on applying reinforcement learning to job shop scheduling problems. J Syst Simul 33(12):2782–2791

Wang R, Wang G, Sun J, Deng F, Chen J (2023) Flexible job shop scheduling via dual attention network-based reinforcement learning. IEEE Trans Neural Netw Learn Syst. https://doi.org/10.1109/TNNLS.2023.3306421
Wang L, Cai J, Li M, Liu Z et al (2017) Flexible job shop scheduling problem using an improved ant colony optimization. Sci Programm 2017:9016303
Wang S, Li J, Jiao Q, Ma F (2024) Design patterns of deep reinforcement learning models for job shop scheduling problems. J Intell Manuf, 1–19
Wen M, Lin R, Wang H, Yang Y, Wen Y, Mai L, Wang J, Zhang H, Zhang W (2023) Large sequence models for sequential decision-making: a survey. Front Comp Sci 17(6):176349
Wu X, Yan X, Guan D, Wei M (2024) A deep reinforcement learning model for dynamic job-shop scheduling problem with uncertain processing time. Eng Appl Artif Intell 131:107790
Wu Z, Fan H, Sun Y, Peng M (2023) Efficient multi-objective optimization on dynamic flexible job shop scheduling using deep reinforcement learning approach. Processes 11(7):2018
Xie H (2009) An analysis of selection in genetic programming. PhD thesis, Open Access Te Herenga Waka-Victoria University of Wellington
Xie J, Gao L, Peng K, Li X, Li H (2019) Review on flexible job shop scheduling. IET Collab Intell Manuf 1(3):67–77
Xiong H, Shi S, Ren D, Hu J (2022) A survey of job shop scheduling problem: The types and models. Comput Oper Res 142:105731
Xu, M., Mei, Y., Zhang, F., Zhang, M.: Multi-objective genetic programming based on decomposition on evolving scheduling heuristics for dynamic scheduling. In: Proceedings of the companion conference on genetic and evolutionary computation, pp 427–430 (2023)
Xu B, Mei Y, Wang Y, Ji Z, Zhang M (2021) Genetic programming with delayed routing for multiobjective dynamic flexible job shop scheduling. Evol Comput 29(1):75–105
Xu M, Mei Y, Zhang F, Zhang M (2023) Genetic programming for dynamic flexible job shop scheduling: evolution with single individuals and ensembles. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2023.3334626
Xu M, Mei Y, Zhang F, Zhang M (2023) Genetic programming with lexicase selection for large-scale dynamic flexible job shop scheduling. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2023.3244607
Xu S, Li Y, Li Q (2024) A deep reinforcement learning method based on a transformer model for the flexible job shop scheduling problem. Electronics 13(18):3696
Xu M, Mei Y, Zhang F, Zhang M (2024) Genetic programming and reinforcement learning on learning heuristics for dynamic scheduling: a preliminary comparison. IEEE Comput Intell Mag 19(2):18–33
Xu M, Mei Y, Zhang F, Zhang M (2024) Niching genetic programming to learn actions for deep reinforcement learning in dynamic flexible scheduling. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2024.3395699
Xu M, Mei Y, Zhang F, Zhang M (2022) Genetic programming with cluster selection for dynamic flexible job shop scheduling. In: Proceedings of the IEEE congress on evolutionary computation, pp 1–8. https://doi.org/10.1109/CEC55065.2022.9870431
Xu M, Mei Y, Zhang F, Zhang M (2022) Genetic programming with diverse partner selection for dynamic flexible job shop scheduling. In: Proceedings of the genetic and evolutionary computation conference companion, pp 615–618 . https://doi.org/10.1145/3520304.3528920
Xu M, Mei Y, Zhang F, Zhang M (2023) A semantic genetic programming approach to evolving heuristics for multi-objective dynamic scheduling. In: Australasian joint conference on artificial intelligence. Springer, pp 403–415
Xu B, Tao L, Deng X, Li W (2021) An evolved dispatching rule based scheduling approach for solving djss problem. In: Proceedings of the Chinese control conference, pp 6524–6531
Xu M, Zhang F, Mei Y, Zhang M (2022) Genetic programming with multi-case fitness for dynamic flexible job shop scheduling. In: Proceedings of the IEEE congress on evolutionary computation, pp 1–8. https://doi.org/10.1109/CEC55065.2022.9870340
Yan Q, Wang H, Wu F (2022) Digital twin-enabled dynamic scheduling with preventive maintenance using a double-layer q-learning algorithm. Comput Oper Res 144:105823
Yan Q, Wu W, Wang H (2022) Deep reinforcement learning for distributed flow shop scheduling with flexible maintenance. Machines 10(3):210
Yang S (2022) Using attention mechanism to solve job shop scheduling problem. In: Proceedings of the international conference on consumer electronics and computer engineering, pp 59–62
Yang Y, Chen G, Ma H, Hartmann S, Zhang M (2024) Dual-tree genetic programming with adaptive mutation for dynamic workflow scheduling in cloud computing. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2024.3392968
Yin WJ, Liu M, Wu C (2003) Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. Proc IEEE Congress Evol Comput 2:1050–1055

Yska D, Mei Y, Zhang M (2018) Feature construction in genetic programming hyper-heuristic for dynamic flexible job shop scheduling. In: Proceedings of the genetic and evolutionary computation conference companion, pp 149–150
Yska D, Mei Y, Zhang M (2018) Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling. In: Proceedings of the European conference on genetic programming, pp 306–321
Yuan E, Cheng S, Wang L, Song S, Wu F (2023) Solving job shop scheduling problems via deep reinforcement learning. Appl Soft Comput 143:110436
Yuan E, Wang L, Cheng S, Song S, Fan W, Li Y (2024) Solving flexible job shop scheduling problems via deep reinforcement learning. Expert Syst Appl 245:123019
Zakaria Y, Zakaria Y, BahaaElDin A, Hadhoud M (2019) Niching-based feature selection with multi-tree genetic programming for dynamic flexible job shop scheduling. In: Proceedings of the international joint conference on computational intelligence, pp 3–27
Zakaria Y, Zakaria Y, BahaaElDin A, Hadhoud M (2021) Niching-based feature selection with multi-tree genetic programming for dynamic flexible job shop scheduling. In: Proceedings of the international joint conference on computational intelligence, pp 3–27
Zeiträg Y, Figueira JR, Horta N, Neves R (2022) Surrogate-assisted automatic evolving of dispatching rules for multi-objective dynamic job shop scheduling using genetic programming. Expert Syst Appl 209:118194
Zeiträg Y, Rui Figueira J, Figueira G (2024) A cooperative coevolutionary hyper-heuristic approach to solve lot-sizing and job shop scheduling problems using genetic programming. Int J Prod Res, 1–28
Zeng Y, Liao Z, Dai Y, Wang R, Li X, Yuan B (2022) Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism. arXiv preprint arXiv:2201.00548
Zhang R, Song S, Wu C (2013) A hybrid artificial bee colony algorithm for the job shop scheduling problem. Int J Prod Econ 141(1):167–178
Zhang F, Mei Y, Nguyen S, Zhang M (2020) Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling. IEEE Trans Cybern 51(4):1797–1811
Zhang C, Song W, Cao Z, Zhang J, Tan PS, Chi X (2020) Learning to dispatch for job shop scheduling via deep reinforcement learning. Adv Neural Inf Process Syst 33:1621–1632
Zhang F, Mei Y, Nguyen S, Zhang M, Tan KC (2021) Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling. IEEE Trans Evol Comput 25(4):651–665
Zhang F, Mei Y, Nguyen S, Zhang M (2021) Correlation coefficient based recombinative guidance for genetic programming hyper-heuristics in dynamic flexible job shop scheduling. IEEE Trans Evol Comput 25(3):552–566. https://doi.org/10.1109/TEVC.2021.3056143
Zhang F, Mei Y, Nguyen S, Zhang M, Tan KC (2021) Surrogate-assisted evolutionary multitasking genetic programming for dynamic flexible job shop scheduling. IEEE Trans Evol Comput 25(4):651–665. https://doi.org/10.1109/TEVC.2021.3065707
Zhang F, Mei Y, Nguyen S, Tan KC, Zhang M (2021) Multitask genetic programming-based generative hyperheuristics: a case study in dynamic scheduling. IEEE Trans Cybern 52(10):10515–10528
Zhang C, Zhou Y, Peng K, Li X, Lian K, Zhang S (2021) Dynamic flexible job shop scheduling method based on improved gene expression programming. Meas Control 54(7–8):1136–1146
Zhang F, Mei Y, Nguyen S, Zhang M (2022) Multitask multiobjective genetic programming for automated scheduling heuristic learning in dynamic flexible job-shop scheduling. IEEE Trans Cybern 53(7):4473–4486
Zhang Y, Bai R, Qu R, Tu C, Jin J (2022) A deep reinforcement learning based hyper-heuristic for combinatorial optimisation with uncertainties. Eur J Oper Res 300(2):418–427
Zhang F, Mei Y, Nguyen S, Tan KC, Zhang M (2022) Instance rotation based surrogate in genetic programming with brood recombination for dynamic job shop scheduling. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2022.3180693
Zhang M, Lu Y, Hu Y, Amaitik N, Xu Y (2022) Dynamic scheduling method for job-shop manufacturing systems by deep reinforcement learning with proximal policy optimization. Sustainability 14(9):5177
Zhang J, He Z, Chan WH, Chow C (2023) Deepmag: deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. Knowl-Based Syst 259:110083
Zhang L, Feng Y, Xiao Q, Xu Y, Li D, Yang D, Yang Z (2023) Deep reinforcement learning for dynamic flexible job shop scheduling problem considering variable processing times. J Manuf Syst 71:257–273
Zhang F, Mei Y, Nguyen S, Tan KC, Zhang M (2023) Task relatedness based multitask genetic programming for dynamic flexible job shop scheduling. IEEE Trans Evol Comput 27(6):1705–1719. https://doi.org/10.1109/TEVC.2022.3199783
Zhang C, Wu Y, Ma Y, Song W, Le Z, Cao Z, Zhang J (2023) A review on learning to solve combinatorial optimisation problems in manufacturing. IET Collab Intell Manuf 5(1):12072

Zhang G, Yan S, Song X, Zhang D, Guo S (2024) Evolutionary algorithm incorporating reinforcement learning for energy-conscious flexible job-shop scheduling problem with transportation and setup times. Eng Appl Artif Intell 133:107974
Zhang L, Yan Y, Yang C, Hu Y (2024) Dynamic flexible job-shop scheduling by multi-agent reinforcement learning with reward-shaping. Adv Eng Inform 62:102872
Zhang C, Cao Z, Song W, Wu Y, Zhang J (2024) Deep reinforcement learning guided improvement heuristic for job shop scheduling. In: Proceedings of the international conference on learning representations. https://openreview.net/forum?id=jsWCmrsHHs
Zhang J, Ding G, Zou Y, Qin S, Fu J (2019) Review of job shop scheduling research and its new perspectives under industry 4.0. J Intell Manuf 30(4):1809–1830
Zhang F, Mei Y, Nguyen S, Zhang M (2020) Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling. In: Proceedings of the European conference on evolutionary computation in combinatorial optimization, pp 214–230
Zhang F, Mei Y, Nguyen S, Zhang M (2020) Guided subtree selection for genetic operators in genetic programming for dynamic flexible job shop scheduling. In: Proceedings of the European conference on genetic programming, pp 262–278
Zhang F, Mei Y, Nguyen S, Zhang M (2021) Collaborative multifidelity-based surrogate models for genetic programming in dynamic flexible job shop scheduling. IEEE Trans Cybern
Zhang F, Mei Y, Nguyen S, Zhang M (2022) Phenotype based surrogate-assisted multi-objective genetic programming with brood recombination for dynamic flexible job shop scheduling. In: Proceedings of the IEEE symposium series on computational intelligence, pp 1218–1225
Zhang F, Mei Y, Nguyen S, Zhang M (2023) Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. IEEE Trans Evol Comput. https://doi.org/10.1109/TEVC.2023.3255246
Zhang F, Mei Y, Zhang M (2018) Genetic programming with multi-tree representation for dynamic flexible job shop scheduling. In: Proceedings of the Australasian joint conference on artificial intelligence, pp 472–484
Zhang F, Mei Y, Zhang M (2018) Surrogate-assisted genetic programming for dynamic flexible job shop scheduling. In: Proceedings of the Australasian joint conference on artificial intelligence, pp 766–772
Zhang F, Mei Y, Zhang M (2019) A new representation in genetic programming for evolving dispatching rules for dynamic flexible job shop scheduling. In: Proceedings of the European conference on evolutionary computation in combinatorial optimization, pp 33–49
Zhang F, Mei Y, Zhang M (2019) A two-stage genetic programming hyper-heuristic approach with feature selection for dynamic flexible job shop scheduling. In: Proceedings of the genetic and evolutionary computation conference, pp 347–355
Zhang F, Mei Y, Zhang M (2019) Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics. In: Proceedings of the IEEE congress on evolutionary computation, pp 1366–1373
Zhang F, Mei Y, Zhang M (2023) An investigation of terminal settings on multitask multi-objective dynamic flexible job shop scheduling with genetic programming. In: Proceedings of the companion conference on genetic and evolutionary computation, pp 259–262
Zhang F, Shi G, Mei Y (2023) Interpretability-aware multi-objective genetic programming for scheduling heuristics learning in dynamic flexible job shop scheduling. In: Proceedings of the IEEE congress on evolutionary computation
Zhao L, Shen W, Zhang C, Peng K (2022) An end-to-end deep reinforcement learning approach for job shop scheduling. In: Proceedings of the international conference on computer supported cooperative work in design, pp 841–846
Zhou B, Wen M (2023) A dynamic material distribution scheduling of automotive assembly line considering material-handling errors. Eng Comput 40(5):1101–1127
Zhou Y, Yang J (2019) Automatic design of scheduling policies for dynamic flexible job shop scheduling by multi-objective genetic programming based hyper-heuristic. Procedia CIRP 79:439–444
Zhou M, Chiu H-S, Xiong HH (1995) Petri net scheduling of fms using branch and bound method. Proc IEEE Conf Ind Electron 1:211–216
Zhou Y, Yang J, Zheng L (2018) Hyper-heuristic coevolution of machine assignment and job sequencing rules for multi-objective dynamic flexible job shop scheduling. IEEE Access 7:68–88
Zhou Y, Yang JJ, Zheng LY (2019) Multi-agent based hyper-heuristics for multi-objective flexible job shop scheduling: a case study in an aero-engine blade manufacturing plant. IEEE Access 7:21147–21176
Zhou Y, Jj Yang, Huang Z (2020) Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming. Int J Prod Res 58(9):2561–2580

Zhu X, Wang W, Guo X, Shi L (2020) A genetic programming-based evolutionary approach for flexible job shop scheduling with multiple process plans. In: Proceedings of the IEEE international conference on automation science and engineering, pp 49–54

Zojaji Z, Kazemi A (2022) Adaptive reinforcement-based genetic algorithm for combinatorial optimization. J Comput Secur 9(1):71–84