Computers & Industrial Engineering 149 (2020) 106778

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A self-learning genetic algorithm based on reinforcement learning for flexible job-shop scheduling problem

Ronghua Chen [a], Bo Yang [a,*], Shi Li [b], Shilong Wang [a]

[a] State Key Laboratory of Mechanical Transmission, Chongqing University, Chongqing, 400044, China
[b] Beijing Aerospace XinFeng Mechanical Equipment Co. Ltd., Beijing 100854, China

ARTICLE INFO

Keywords:
Flexible job-shop scheduling problem (FJSP)
Self-learning genetic algorithm (SLGA)
Genetic algorithm (GA)
Reinforcement learning (RL)

ABSTRACT

As an important branch of production scheduling, flexible job-shop scheduling problem (FJSP) is difficult to solve and is proven to be NP-hard. Many intelligent algorithms have been proposed to solve FJSP, but their key parameters cannot be dynamically adjusted effectively during the calculation process, which causes the solution efficiency and quality not being able to meet the production requirements. Therefore, a self-learning genetic algorithm (SLGA) is proposed in this paper, in which genetic algorithm (GA) is adopted as the basic optimization method and its key parameters are intelligently adjusted based on reinforcement learning (RL). Firstly, the self-learning model is analyzed and constructed in SLGA, SARSA algorithm and Q-Learning algorithm are applied as the learning methods at initial and later stages of optimization, respectively, and the conversion condition is designed. Secondly, the state determination method and reward method are designed for RL in GA environment. Finally, the learning effect and performance of SLGA in solving FJSP are compared with other algorithms using two groups of benchmark data instances with different scales. Experiment results show that the proposed SLGA significantly outperforms its competitors in solving FJSP.

1. Introduction

Production scheduling plays a crucial role in many manufacturing systems, an effective production scheme is a vital factor to enables industry to improve production efficiency and improve utilization rate of resources (Pezzella et al., 2008). There are many kinds of scheduling problems and the job-shop scheduling problem (JSP) is one of the most difficult problems in practice manufacturing process. It is well known that JSP is NP-hard (non-deterministic polynomial time hard) problem (Garey, Johnson, & Sethi, 1976) and a combinatorial optimization problem that is a representative model of many real life optimization tasks (Acan and Ünveren, 2015).

The classical JSP can be described as a set of jobs to be processed on a group of machines, each job consists of a series of operations and has a specified processing order, and each operation is processed on a required machine. However, the application scenario of JSP could not meet the flexible production situation. In reality, a machine may have the capability to process more than one type of operations, in turn, each type of operations could be machined on several different machines, which is more flexible than JSP. This problem is generally referred as flexible job-shop scheduling problem (FJSP) (Brucker and Schlie, 1990; Brandimarte, 1993) and is used to describe the flexible manufacturing requirement of modern manufacturing systems.

FJSP is an extension of the classical JSP in which the constraints on machine selection are reduced, each operation of the FJSP is allowed to be processed on more than one machine from its alternative machine set. FJSP is a more complex combinatorial optimization problem than JSP as it introduces a new decision content to the sequencing and it consists of more problems, i.e., it is aimed to figure out two subproblems including operation sequencing and machine assignment (Garey et al., 1976; Li & Gao, 2016). FJSP has been proven to be more complicated and NP-hard (Gao, Yang, Zhou, Pan, & Suganthan, 2019; Kacem, Hammadi, Borne, & Man, 2002). The objective of FJSP is to obtain an allocation for each operation and to define the sequence of operations on each machine to minimize maximum processing completion time (makespan) (Nouiri et al., 2018).

Many methods have been applied for FJSP due to its nature of computational complexity. Some studies proposed exact methods, including mathematical programming and mixed integer goal programming (Choi and Choi, 2002). However, the exact method is time-

* Corresponding author at: State Key Laboratory of Mechanical Transmission, Chongqing University, 174 Shazheng Street, Shapingba District, Chongqing 400044, China.
E-mail addresses: 656764617@qq.com (R. Chen), yangbo61@cqu.edu.cn (B. Yang), 237840084@qq.com (S. Li), slwang@cqu.edu.cn (S. Wang).

https://doi.org/10.1016/j.cie.2020.106778
Received 10 January 2020; Received in revised form 30 June 2020; Accepted 18 August 2020
Available online 29 August 2020

consuming and difficult to achieve global optimum even for a small-size instances since the NP-hard characteristic of FJSP. Therefore, many researchers start develop more efficient solution methodologies to obtain nearly optimal solutions (Acan and Ünveren, 2015). In this trend, many optimization algorithms have been developed to solve combinatorial optimization problems, especially for FJSP. Zhang *et al.* (Zhang et al., 2009) presented a hybrid algorithm for FJSP, in which particle swarm optimization (PSO) algorithm and tabu search (TS) are combined. Nouiri *et al.* (Nouiri et al., 2018; Nouiri et al., 2017) proposed a multi-agent-based PSO and a two-stage PSO to solve FJSP. Xing *et al.* (Xing et al., 2010) proposed a knowledge-based ant colony optimization (ACO) for FJSP, in which knowledge model is used to guide the current heuristic searching. Liouane et al. (Liouane et al., 2007) also presented an approach based on the combination of ACO and local search (LS) methods. Jiang *et al.* (Jiang and Zhang, 2018) first applied discrete Grey Wolf Optimization (GWO) to solve JSP and FJSP where the adaptive mutation method is introduced with the objective of minimizing the makespan. A hybrid artificial bee colony (ABC) algorithm introducing a dynamic scheme to fine-tune the search scope adaptively was proposed by Meng (Meng et al., 2018). Li *et al.* (Li et al., 2017) applied hybrid ABC based on TS to solve FJSP with rescheduling strategy for new jobs inserted, old jobs cancelled, and machinery breakdowns.

Genetic algorithm (GA) is the most commonly used algorithm to solve FJSP due to its superior performance and strong universality (Shao et al., 2018). Kacem *et al.* (Kacem et al., 2002) proposed an assignment model generated by the approach of localization (AL) to control GA for total or partial FJSP. Pezzella *et al.* (Pezzella et al., 2008) presented a method to integrate different strategies for generating the initial population and selecting the individuals for reproducing new individual, which lead to better results. Zhang *et al.* (Zhang et al., 2010) presented a multi-objective GA based on immune and entropy principle for the multi-objective FJSP. Zhang *et al.* (Zhang et al., 2011) and Nasr *et al.* (Al-Hinai, 2011) presented similar improved GA with LS to generate high-quality initial population for FJSP. Li *et al.* (Li and Gao, 2016) proposed a hybrid algorithm, in which GA and TS are used to control global and local search, respectively. Zandieh *et al.* (Zandieh and Karimi, 2010) studied adaptive multi-population GA for FJSP, whose objectives are minimizing total weighted tardiness and maximum completion time simultaneously. Liu *et al.* (Liu et al., 2014) refined the encoding operator of GA that integrates probability concepts into a real-parameter encoding method to solve the distributed FJSP. Chang *et al.* (Chang et al., 2015) presented a hybrid taguchi GA to solve FJSP, which involves encoding feasible solutions to the initial chromosomes of a GA and embedded the Taguchi method behind mating to increase the effectiveness of the GA.

It can be observed that GA is very effective and popular in solving FJSP problems, whose performance is related to many aspects, one of the most important factors is the selection of key parameters, but there is little research on how to determine them (Shahrabi et al., 2017). For GA, crossover probability (*Pc*) and mutation probability (*Pm*) can be regarded as two key parameters. If *Pc* and *Pm* are too large, some good individuals could be easily lost; while it is hard to generate new individuals if *Pc* and *Pm* are too small (Bashir and Nadeem, 2017). From the above studies in the literature, we can find that almost all algorithms have the disadvantage in parameter setting. Since the key parameters of algorithms and their adjustment methods are hard to be reasonably determined, they are usually fixed or updated in some predetermined manners (Du et al., 2014; Wei et al., 2014), which seriously degrades the performance of the algorithms. At present, there are few studies on the intelligent adjustment of key parameters for algorithms, Emary *et al.* (Emary et al., 2018) studied the adaptive exploration rate of GWO based on RL and neural network. Chen *et al.* (Fei Chen et al., 2005) proposed an approach to control GA with SARSA(0), in which the parameters are adjusted by only SARSA, but the effective is very limited with only SARSA. Moreover, these methods have not been applied in solving practical engineering problems. Shahrabi *et al.* (Shahrabi et al., 2017) proposed a RL approach to parameter estimation of variable neighborhood search (VNS) for solving dynamic job shop scheduling (DJSS) problem, in which the optimization process of parameters are selected by continually improving policy and the parameters are adjusted when a new event generated, but the parameter variations during iteration process are not taken into account, which is not benefited for the performance of whole algorithm. Furthermore, there is only limited research on the intelligent adjusting of key parameters for algorithms in solving FJSP (Amjad et al., 2018).

This paper presents a self-learning GA (SLGA) based on reinforcement learning (RL) to intelligently adjust the key parameters of GA for FJSP with the objective of minimizing the makespan. RL algorithm, also known as reward learning, evaluation learning, is a crucial machine learning algorithm (Sutton and Barto, 1998). RL uses a scalar reinforcement signal or a reward to interact with a complex environment (Wang et al., 2013), and it maps actions to environment for maximizing a reward by systematic learning, which has been applied to a wide range of fields. Qin *et al.* (Qin et al., 2019) investigated an optimal synchronization problem for a group of generic linear systems based on multi-agent RL system. Li *et al.* (Li et al., 2011) proposed fuzzy motion control based on RL for ait synthesis of biped robots. Aissani *et al.* (Aissani et al., 2011) studied dynamic scheduling based on reinforcement multi-agent learning. Zhang *et al.* (Zhicong Zhang et al., 2012) presented a RL approach to minimizing mean weighted tardiness in unrelated parallel machine scheduling. Nowadays, more and more researchers combine RL with other intelligent algorithms to improve the performance of whole algorithm. Wang *et al.* (Wang et al., 2013) proposed a backward Q-learning which combines Q-learning with SARSA to directly tunes the *Q*-values. Gyoung *et al.* (Gyoung Hwan Kim, 1998) proposed genetic reinforcement learning (GRL) which regards scheduling problem as a RL problems to solve it. and the policies of RL are encoded into the chromosomes of GA and a near-optimal policy is searched for by GA. RL and GA were integrated to make decisions when the robots cooperatively transport an object to a goal location while avoiding obstacles (Wang and de Silva, 2008). Hsieh *et al.* (Hsieh and Su, 2015) presented an improved PSO based on Q-learning for economic dispatch problem, which treat optimization problems as a kind of RL problems regarding an optimization procedure for searching an optimal solution as a RL procedure for finding the best policy. Alipour *et al.* (Alipour et al., 2017) used a hybrid GA with multi-agent RL heuristic to solve the traveling salesman problem (TSP) and got good results. None of these methods involve parameter tuning, therefore, this paper tries to use RL to update *Pc* and *Pm* of GA intelligently for solving FJSP which combines both SARSA and Q-learning with GA, so as to improve the efficiency and accuracy of solutions.

The remainder of this paper is organized as follows. Problem model and objective function are presented in Section 2, while the basic and general features of GA and RL algorithms are described in Section 3. The proposed SLGA is described completely and more details about SLGA are displayed in Section 4. Experimental verification is carried out in Section 5. Finally, conclusion and outlines future research are given in Section 6.

2. Problem formulation

2.1. Problem model

FJSP is an extension of the classical JSP and is strongly NP-hard due to: (1) sequence decisions of operations to a subset of machines and (2) assignment decisions of operations on each machine (Wang et al., 2013). A $n \times m$ FJSP can be formulated as there are n independent jobs $J = \{J_1; J_2, J_3, \ldots, J_n\}$ and m independent processable machines $M = \{M_1, M_2, M_3, \ldots, M_m\}$. Each job J_i contains a number of operations $O_{i,j}$. $O_{i,j}$ is the j-th operation of the i-th job ($j = 1, 2, 3, \ldots, h$), $t_{i,j,k}$ is the processing time of $O_{i,j}$ on machine M_k ($k = 1, 2, 3, \ldots, m$), which is fixed and known in advance. There are two types of FJSP, including Partial FJSP and Total FJSP. As for Partial FJSP, each operation $O_{i,j}$ can be processed on one or

more machines. As for Total FJSP, each operation $O_{i,j}$ could be processed on every machine of the set of all machines. Table 1 is an example of 4×3 Partial FJSP, where the numbers represent processing times $t_{i,j,k}$ and the symbol '-' indicates that operation cannot be processed on the corresponding machine. To complete a job J_i ($i = 1, 2, 3, \ldots, n$), all operations $O_{i,j}$ must be completed on a set of machines with machining capability. There are some limitations for the assignments of the operations. In this paper, the constraints are considered as follows:

(1) Each job J_i is independent and has a fixed processing time $t_{i,j,k}$.
(2) The order of precedence between operations must be performed, e.g., O_{i2} must be processed after O_{i1} is finished.
(3) All jobs J_i and machines M_k are available at time zero.
(4) Each operation $O_{i,j}$ has at least one machinable machine.
(5) Each machine can process only one operation at any time.
(6) The operations being processed cannot be interrupted until the machining is finished.
(7) The installation time of the machine and the transportation time between operations are negligible.
(8) Machine disruptions are ignored.

2.2. Objective

For FJSP, each job is composed of a sequence of precedence constrained operations, whose processing times are different between different assigned machines. The objective is to obtain a schedule that has the lowest makespan. The objective function can be represented by Eq. (1), and some limitations are given.

Objective:

$$Min.C_{max} = Min\left\{Max.\sum_{i=1}^{n}\sum_{j=1}^{h}(s_{i,j,k} + t_{i,j,k})\right\} \tag{1}$$

Subject to:

$$t_{i,j,k} > 0,\ i = 1,\ 2,\ \ldots,\ n;\ j = 1,\ 2,\ \ldots,\ h;\ k = 1,\ 2,\ \ldots,\ m. \tag{a}$$

$$s_{i,j,k} + t_{i,j,k} \leqslant s_{i,j+1,k},\ i = 1,\ 2,\ \ldots,\ n;\ j = 1,\ 2,\ \ldots,\ h;\ k = 1,\ 2,\ \ldots,\ m. \tag{b}$$

$$\sum_{k=1}^{m} X_{i,j,k} \geqslant 1,\ i = 1,\ 2,\ \ldots,\ n;\ j = 1,\ 2,\ \ldots,\ h. \tag{c}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{h} X_{i,j,k} = 1,\ k = 1,\ 2,\ \ldots,\ m. \tag{d}$$

$$X_{i,j,k} = \begin{cases} 1, & \text{if operation } O_{i,j} \text{ is assigned to machine } k \\ 0, & \text{otherwise} \end{cases} \tag{e}$$

where C_{max} is the maximum completion time of all job J_i, $s_{i,j,k}$ represents the start time of operation $O_{i,j}$ in machine M_k. Constraint (a) represents the processing time of all operations are greater than 0. Constraint (b) indicates the order of precedence between operations must be performed. Constraint (c) shows that each operation $O_{i,j}$ has at least one machinable machine. Constraint (d) ensures each machine can process only one operation at any time.

Table 1
An instance of 4 × 3 Partial FJSP.

Jobs	Operation	M_1	M_2	M_3
J_1	O_{11}	2	–	**5**
	O_{12}	–	**1**	4
J_2	O_{21}	**2**	3	–
	O_{22}	4	2	**1**
J_3	O_{31}	2	**3**	–
	O_{32}	**2**	–	5
J_4	O_{41}	**4**	–	2
	O_{42}	–	**3**	5

3. Basic algorithms

3.1. Genetic algorithm

Genetic algorithm (GA) can be regarded as the most typical population-based optimization algorithm to solve practical combinatorial optimization problems due to its superior performance and strong universality. It starts with a predefined size of population solutions that is composed of a certain number of individuals. Every individual solution has a chromosome representation that represents a solution to the optimization problem. GA finds the different solutions through crossover and mutation operations, and reserves the excellent individuals through selection operation (Zhang et al., 2011).

3.1.1. Chromosome representation

The chromosome of FJSP is divided into operation sequence part and machine assignment part (Karimi et al., 2012), which makes encoding become easy and reduces the cost of decoding. Fig. 1 is an encoding for the instance of FJSP shown in Table 1, which expresses a solution scheme of optimization problem, and the machine assignment of operation is highlighted by using bold text. The number of operations is 8, while the length of chromosome is 16, so the length of chromosome is twice as long as the operations. By scanning the operation sequence from left to right, the first '1′ represents the first operation of Job J_1 that will be processed on machine M_3 corresponds to the '3′ in the machine assignment part; the second '1′ represents the second operation of Job J_1 that will be processed on machine M_2 corresponds to the '2′ in the machine assignment part. According to the above explanation, the operation sequences on machine M_1 is $O_{41} \rightarrow O_{21} \rightarrow O_{32}$, machine M_2 is $O_{31} \rightarrow O_{42} \rightarrow O_{12}$, machine M_3 is $O_{11} \rightarrow O_{22}$. Moreover, operation sequence and machine assignment of solution can be interpreted as: {(O_{11}, M_3), (O_{41}, M_1), (O_{31}, M_2), (O_{21}, M_1), (O_{42}, M_2), (O_{32}, M_1), (O_{12}, M_2), (O_{22}, M_3)}.

3.1.2. Population initialization

The initial population is important for GA which directly influences the convergence rate of fitness values and the quality of solutions (Chang et al., 2015). In this paper, every individual of initial population is randomly generated, in which two priority rules are employed, which are described as follows (Jiang and Zhang, 2018).

(1) CMO: Choose the job that has the greatest number of operations remaining
(2) HCMS: High probability to choose the machine with the shortest processing time for the corresponding operations.

CMO is applied to generate appropriate operation sequence, while HCMS is used for getting machine assignment with shorter time. Combining priority rules CMO and HCMS can generate initial populations with better quality.

3.1.3. Genetic operation

Crossover operation is the most significant step and is the backbone of GA, in which the two gene substrings of parents' chromosome will be crossed under the same probability Pc. During the last decades, several types of crossover operation have been applied to permutation

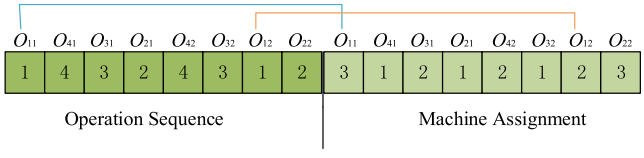

Fig. 1. A chromosome representation of Table 1.

representation. In this study, a precedence preserving order-based crossover (POX) is applied for operation sequence since its wonderful effect for crossover operation according to references (Jiang and Zhang, 2018; Zhang et al., 2011; Liu et al., 2014).

Mutation operation is also an important part of GA, which can enhance the diversity of the population to avoid the local optimum phenomena by introducing some additional variability into the population. As a result, the local search performance of the algorithm can be improved by mutation operation. Usually, mutation is applied with small probability since large probability may destroy the good chromosomes. In this paper, swap mutation is applied (Lei, 2012).

3.1.4. Selection operation

The selection operation is to generate a new population with higher fitness values from the current population. The common selection strategies such as the roulette wheel, which is a fitness-based approach, whose chromosome of optimal individuals could be destroyed after genetic operations. Therefore, the chromosome must be repaired, which could influence the optimal individuals (Zhang et al., 2011). While the elite retention strategy a procedure to preserve the optimal individuals during each iteration and leave bad individuals out of the population, which will impair the premature phenomena and accelerate the convergence speed (Wang et al., 2017). In this paper, elite retention strategy is adopted, which could enable individuals to naturally evolve in almost all generations and improve the quality of the entire population continuously.

3.2. Reinforcement learning

In recent years, researching on RL has received more and more attention. By mapping actions to environmental states that aims to achieve the most cumulative reward value of actions from the environment, RL evaluates actions according to environment signals instead of through test cases to train an agent to take the correct action (Wang et al., 2019). In RL, the agent continuously interacts with a complex environment to optimize action selection according to the feedback of the environment, finally achieves the best decision (Sutton and Barto, 1998). The model framework of RL is shown in Fig. 2, at time step t, the agent acquires current environment state s_t and takes action a_t. Then, at time step $t+1$, the environment transits to the state s_{t+1} after executing of action a_t, and the agent gets reward r_t from environment, which will make the agent update the action selection and take a suitable action a_{t+1}. The more suitable actions will make the agent finds an optimal strategy π_t to achieve maximum long-term rewards (Qin et al., 2019). The strategy formula can be expressed as Eq. (2).

$$\pi^* = \underset{\pi}{argmax} E_\pi \left\{ \sum_{k=0}^{\infty} \gamma^k r_{t+k} | s_t \right\} \forall s_t \in S, \forall t \geqslant 0 \tag{2}$$

where $\gamma \in$ (Pezzella et al., 2008) represents the discount rate, which specifies the future return value in the current state, t is current time step, k is the future time step, and s_t is the state of time step t, S is the state set which contains all states.

SARSA algorithm and Q-learning algorithm are two typical value-based RL algorithms, whose objective is to estimate the Q value of an optimal strategy (Hsieh and Su, 2015). Q value; which is used to characterize the rationality of action selection, is updated according to the feedback from environment and state-action. Q value is saved in the Q value table that used to record the learning experience of the agent. The initial Q value table is a zero-value matrix generated by the algorithm, whose number of rows is equal to the quantity of states and number of columns is equal to the quantity of actions. As shown in Table 2, the Q value table assumes that there are n states in the state set and m actions in the action set.

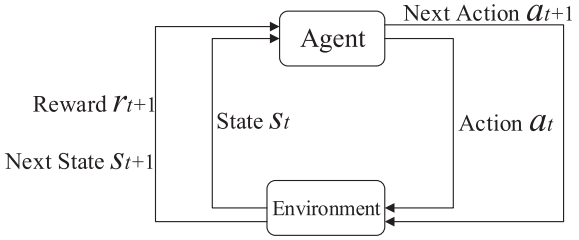

Fig. 2. The model framework of RL.

The calculation of the Q value is a comprehensive consideration of the experienced state, the selected action and the obtained reward by the agent, the Q value update function of one-step SARSA algorithm and one-step Q-learning algorithm are expressed as Eq. (3) and Eq. (4) (Wang et al., 2013), respectively.

$$Q(s_t, a_t) \leftarrow (1-\alpha) Q(s_t, a_t) + \alpha (r_{t+1} + \gamma Q(s_{t+1}, a_{t+1})) \tag{3}$$

$$Q(s_t, a_t) \leftarrow (1-\alpha) Q(s_t, a_t) + \alpha (r_{t+1} + \gamma \underset{a}{max} Q(s_{t+1}, a_{t+1})) \tag{4}$$

where $Q(s_t, a_t)$ is the Q value of taking action a_t in current state s_t; $\alpha \in$ (Pezzella et al., 2008) represents the learning rate; r_{t+1} is the reward value after the execution of action a_t at state s_t; γ is the discount rate. $max_a Q(s_{t+1}, a_{t+1})$ represents the highest expected Q value in Q value table at state s_{t+1} when an action a_{t+1} is executed, while $Q(s_{t+1}, a_{t+1})$ represents the expected Q value at state s_{t+1} when the action a_{t+1} which is chosen by a strategy (e.g. ε-*greedy* strategy) is executed.

The action of Q-learning with the highest expected Q value is selected in each state to update Q value, in which more accumulated knowledge is utilized, so the stronger final performance of Q-learning can be obtained. But it has lower learning effect and worse quality at the early stage. While SARSA has a higher learning effect and convergence, but it easily falls into local optimum and hard to obtain the optimal strategy.

4. Proposed SLGA

The crossover and mutation operations play important roles for the performance of GA, and the most significant parameters of crossover and mutation operations are *Pc* and *Pm*. The fitter individuals are easily destroyed in the population if *Pc* and especially *Pm* are too large, which is not conducive to the convergence of the solution and the generation of the optimal solution; while new individuals are hard generated if *Pc* and *Pm* are too small (Bashir and Nadeem, 2017).

In conventional GA, *Pc* and *Pm* are always determined by a large number of experiments or experience, which is burdensome and parsimonious. Some adaptive GAs appeared in literatures can solve this problem to a certain extent, but all of them formulate the values of *Pc* and *Pm* according to the current population fitness value in some predetermined manner (Du et al., 2014; Wei et al., 2014). The influence of previous generations or possible future populations is ignored, so their applicability and reliability cannot be guaranteed. In this section, a self-learning genetic algorithm (SLGA) is designed to intelligently adjust key parameters for GA based on RL, where the *Pc* and *Pm* can be chosen intelligently according to not only the current population state but also

Table 2
Initial Q value table.

$$Q(s_t, a_t) = \begin{array}{c} \\ s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_n \end{array} \begin{array}{c} \begin{array}{ccccc} a_1 & a_2 & a_3 & \cdots & a_m \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \end{array}$$

the past state and the appropriate prediction of the future state.

4.1. Combined model

The basic components of RL are strategy, reward, value function and environment, which must be fully considered when GA and RL are combined. Therefore, the following scheme is designed under the above premise. As is shown in Fig. 3, the combined model of GA and RL is divided into environment, learning module and reinforcement process. GA is regarded as environment, and each iteration will change its state from s_t to s_{t+1}. Learning module is composed of the RL agent and Q value table, in here, the agent is a virtual object of RL algorithms.

The execution of the composition is divided into four phases (Wang et al., 2013). Firstly, the state $s_t(GA)$ at time step t of GA iterative process is acquired by the agent (the state set is shown in part 4.3.1). Secondly, the corresponding action $a_t(GA)$ is executed according to the determined action selection strategy (the action set is shown in part 4.3.2). In SLGA, adjusting and updating of Pc and Pm are actions that taken by the agent, and GA will use the updated new Pc and Pm. Thirdly, the genetic operation is performed again, the state of GA is translated into $s_{t+1}(GA)$ and the feedback is returned to the agent. Finally, the agent will take action a_{t+1} for GA and the learning process is recorded by the agent according to the experienced state-action and the received feedback. Meanwhile, valuation calculation is performed with value function, and the Q value is updated under the corresponding state-action in the Q value table.

After k more iterations, the state of GA $s_{t+k}(GA) = s_t(GA)$ at a later iterative time step $t + k$, the reinforcement process is activated, and the selection of actions for Pc and Pm will be optimized based on the existing state of the past learning experience. The selection of action $a_t(GA)$ will be strengthened if the obtained reward is positive; the selection for the $a_t(GA)$ will be weakened accordingly if the obtained reward is negative (Emary et al., 2018). This process of continuously state acquired, actions performed, feedback obtained, and strategies adjusted is the reinforcement process, it can be well adapted to the dynamic of the GA's iterative process.

4.2. Construction of the learning module

SARSA algorithm and Q-learning algorithm are combined as main part of Learning Module, which merges the advantages of the two algorithms. Wang *et. al* (Wang et al., 2013) propose a Backward Q-learning which combines SARSA with Q-learning to enhance learning speed and improve final performance. In this paper, a different way to combine two RL algorithms is proposed to enhance the performance of GA, which applies SARSA and Q-learning in the different stages. In the initial stage of SLGA, SARSA algorithm is combined with GA to form SLGA. The Q value is updated by Eq. (3) and recorded in Q value table, whose Q-values will be used as the pretrain process of Q-learning algorithm.

SARSA algorithm will be converted to the Q-learning algorithm if the conversion condition is met. The conversion condition guarantees Q-learning algorithm have adequate learning materials or learning experiences, which is determined by the quantity of Non-zero Q value, and the quantity of Q value is relevant to the number of states and the number of actions. So, in SLGA, the conversion condition is considered as Eq. (5),

$$\mathrm{RL} = \begin{cases} \text{Sarsa} & N_{ti} < \dfrac{N_{ts} \times N_{ta}}{2} \\ \text{Q-learning} & N_{ti} \geqslant \dfrac{N_{ts} \times N_{ta}}{2} \end{cases} \tag{5}$$

where N_{ti} represents the number of current iterations. N_{ts} represents the total number of states. N_{ta} represents the total number of actions.

After initial stage, Q-learning algorithm replaces SARSA algorithm to combine with GA and the Q value is updated by Eq. (4). The execution steps of the Learning Module are illustrated as follows (Shahrabi et al., 2017; Hsieh and Su, 2015), and the schematic is shown in Fig. 4.

Step1: Initialize Q value table for each state-action pair.
Step2: Obtain the current state s_t, $s \leftarrow s_t$.
Step3: Select action a from action set in state s using the strategy derived from SARSA or Q-learning according to Eq. (5).
Step4: Execute action a.
Step5: Obtain the reinforcement reward r, and get the new state s_{t+1}.
Step6: Terminate the process if the terminal state of GA is reached, otherwise, go to **Step7**.
Step7: Update the Q value according to Eq. (3) or Eq. (4).
Step8: Update the state: $s \leftarrow s_{t+1}$.
Step9: Determine whether the conversion condition is met. If so, transform SARSA to Q-learning and go to **step2**. If not, go directly to **step2**.

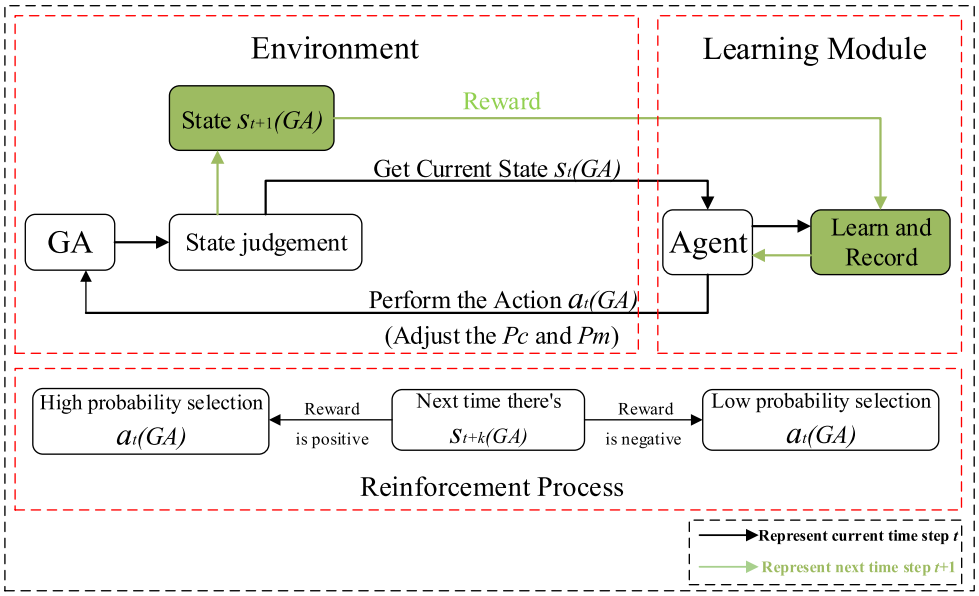

Fig. 3. The combined model of GA and RL.

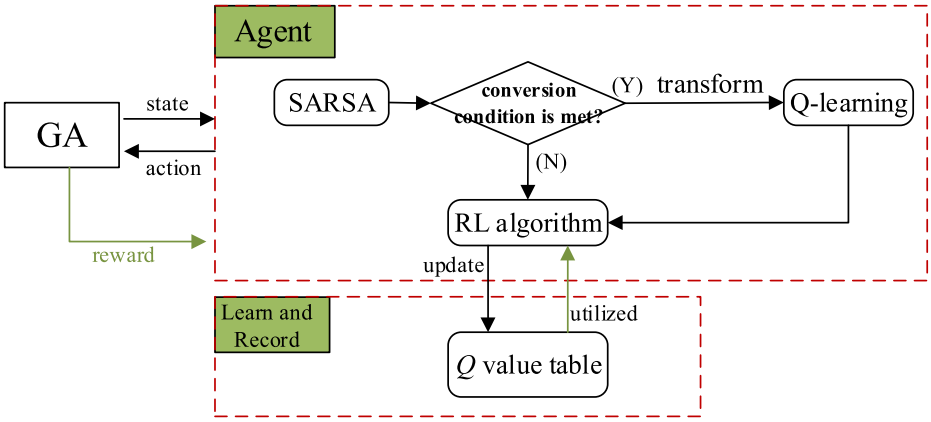

Fig. 4. The schematic of learning module.

4.3. State set

The objective for FJSP is to find a schedule that has the lowest makespan, which is regarded as the fitness of GA. In SLGA, the environment state should be constructed based on the population fitness, the following aspects are considered.

(1) : average fitness of the population;
(2) : population diversity;
(3) : the fitness of the best individual.

Eq. (6) gives the average fitness of the population that has been normalized by the average fitness of the first-generation population. Eq. (7) gives the population diversity that has been normalized by the first-generation population diversity. Eq. (8) gives the best fitness of the population after normalization by the best fitness of first generation. Eq. (9) is the population state value, which is obtained by weighting Eq. (6), Eq. (7) and Eq. (8).

$$f^* = \frac{\sum_{i=1}^{N} f(x_i^t)}{\sum_{i=1}^{N} f(x_i^1)} \tag{6}$$

$$d^* = \frac{\sum_{i=1}^{N} \left| f(x_i^t) - \frac{\sum_{i=1}^{N} f(x_i^t)}{N} \right|}{\sum_{j=1}^{N} \left| f(x_j^1) - \frac{\sum_{j=1}^{N} f(x_j^1)}{N} \right|} \tag{7}$$

$$m^* = \frac{maxf(x_i^t)}{maxf(x_i^1)} \tag{8}$$

$$S^* = w_1 f^* + w_2 d^* + w_3 m^* \tag{9}$$

where $f(x_i^t)$ represents the fitness of the i-th individual in the t-th generation, $f(x_i^1)$ represents the fitness of the i-th individual in the first generation, $f(x_j^1)$ represents the fitness of the j-th individual in the first generation, max $f(x_i^t)$ means the fitness value of best individual in the t-th generation. w_1, w_2, w_3 are the weight values and $w_1 + w_2 + w_3 = 1$, which represent the relative importance of three factors. In this paper, the population average fitness and the population diversity reflect the whole population state, which is good for improving quality of whole population and it could be easier obtain an excellent individual. But only an outstanding individual could be relatively different to get a better individual by cross operation with other poor individuals of population. So, the value of w_1 and w_2 are relatively larger than w_3. In SLGA, w_1, w_2, and w_3 are set as 0.35, 0.35 and 0.3, respectively.

The number of states is momentous, too many states could be more precisely for self-learning, but it needs more exploration, which will affect the convergence of GA, while too few states could lead poor results. According to the fitness of GA for solving FJSP and the value of S^*, the state set is divided into 20 states for $S = [s(1), s(2), \ldots, s(19), s(20)]$ (Shahrabi et al., 2017), where the interval value of S^* is set as 0.05, e.g., when $S^* \in [0, 0.05)$, $s = s(1)$; $S^* \in [0.05, 0.10]$, $s = s(2)$ and so on.

4.4. Action set

For each generation, the agent will adopt different action to get appropriate *Pc* and *Pm*, 10 actions are included in the action set (Shahrabi et al., 2017). As for *Pc*, whose range of commonly used values is from 0.4 to 0.9, and the interval value between each action is 0.05, e.g., when the action is set to a_1, $Pc \in [0.4, 0.45)$ and a random value within this range will be selected; when the action is set to a_2, $Pc \in [0.45, 0.5)$. The same method is suited for *Pm*, whose range of values is from 0.01 to 0.21, and the interval value is 0.02, e.g., when the action is set to a_1, $Pm \in [0.01, 0.03)$ and a random value within this range will be chosen.

4.5. Reward method

In SLGA, the reward is designed through the best individual fitness and the average fitness of population. Eq. (10) gives the reward of the adjusted *Pc*, and Eq. (11) is used for the reward of the adjusted *Pm* (Fei Chen et al., 2005). The agent is not told which actions to take but instead discovers which actions yield to higher reward by trying them, obviously, this form of reward could yield a positive reward. If the best individual of t-th generation is better than (t-1)-th generation, the agent is rewarded that current *Pc* is efficient. If the average fitness of t-th generation is better than (t-1)-th generation, the agent is rewarded that current *Pm* is effective.

$$r_c = \frac{maxf(x_i^t) - maxf(x_i^{t-1})}{maxf(x_i^{t-1})} \tag{10}$$

$$r_m = \frac{\sum_{i=1}^{N} f(x_i^t) - \sum_{i=1}^{N} f(x_i^{t-1})}{\sum_{i=1}^{N} f(x_i^{t-1})} \tag{11}$$

where $f(x_i^t)$ is the fitness of the i-th individual in the t-th generation.

4.6. Action selection strategy

The action selection strategy of RL is also called the search strategy, which offers a trade-off between exploration and exploitation. The unknown environment is explored and the acquired knowledge is utilized to guide the choice of action by the agent. At the beginning, all Q values are zero, which means that the agent does not have any learning experience to use, only the exploration can be performed and learned.

$\mathcal{E}$-greedy is an action selection strategy that considers both exploration and exploitation, which is expressed by Eq. (12) (Shahrabi et al., 2017; Hsieh and Su, 2015; Wang et al., 2019); where ε is called the *greedy* rate or the exploitation rate and $r_{0\text{-}1}$ is a random value from 0 to 1. When$\varepsilon \geqslant r_{0-1}$, the action a which maximizes the expected Q value is selected, which is also called *greedy* strategy. While$\varepsilon < r_{0-1}$, the exploration will be performed and a random action a is chosen.

$$\pi(s_t, a_t) = \begin{cases} max_a Q(s_t, a) & \varepsilon \geqslant r_{0-1} \\ a(Randomly) & \varepsilon < r_{0-1} \end{cases} \tag{12}$$

4.7. Procedure of the SLGA

Based on above work, the execution flow of the entire SLGA is shown in Fig. 5 and described in Algorithm 1.

Algorithm 1. SLGA

Initialize the GA: population size (N), maximum iterations (Max_t),
Initialize the RL: Q value table, state set (S), action set (A), policy (ε-*greedy*).
- Set current iteration number t=0. Calculate the fitness of all individuals.
- Choose a random action a_t, set $a \leftarrow a_t$. Calculate the State s_t of GA, set $s \leftarrow s_t$.
While $t \leq Max_t$ **do** /*major loop of SLGA*/
  Calculate the reward r_{t+1} according to Eq. (10) and Eq. (11).
  if conversion condition is not met /*SARSA algorithm*/
    Choose action a_{t+1} with ε-*greedy*.
    Update Q value according to Eq. (3)
  else
    Choose action a_{t+1} with *greedy*. /*Q-learning algorithm*/
    Update Q value according to Eq. (4).
    Update action a_{t+1} with ε-*greedy*.
  end
  - Calculate the state s_{t+1} of GA according to Eq. (6), Eq. (7), Eq. (8) and Eq. (9), update current $s \leftarrow s_{t+1}$.
  - Update current action $a \leftarrow a_{t+1}$.
  - Execute action a. /*action a will take new Pc and Pm*/
  - Execute genetic operation.
  - $t = t + 1$.
  - Calculate the fitness of all individuals.
End
Obtain solution

5. Experiment results and discussion

In this section, a series of experiments are carried out to test the effectiveness and performance of SLGA for solving FJSP. Two different groups of benchmark data instances are used for experiments, including three small-scale Kacem's (Kacem et al., 2002) total flexible FJSP

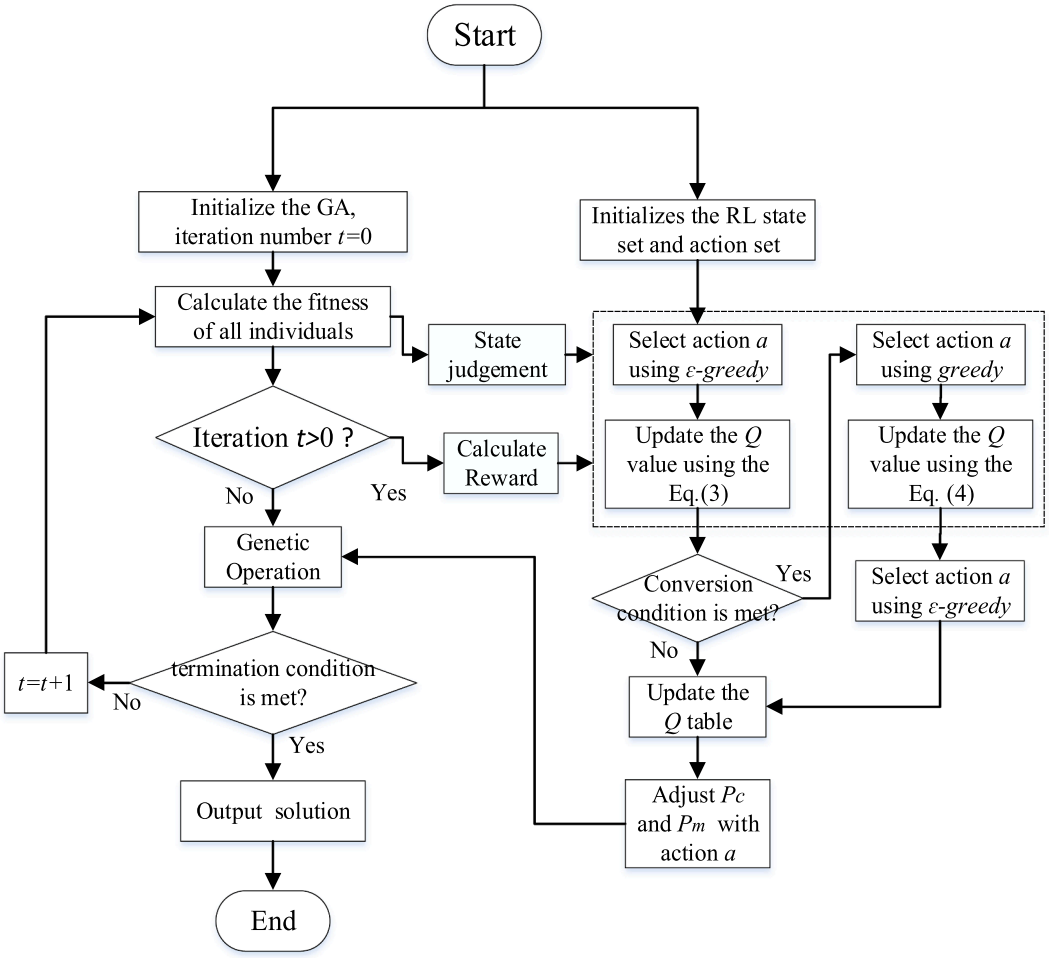

Fig. 5. Flow chart of SLGA algorithm.

instances and ten large-scale Brandimarte's (Brandimarte, 1993) partial flexible FJSP instances, respectively. Moreover, their lower bounds will be used as a benchmark for comparison and computational analysis. These experiments are implemented in Matlab 2018a with Matlab language. The computer configuration is Intel®Core™i5-4590 with 8 GB main memory under Win10.

5.1. Experimental comparison for hybrid learning strategy

In this part, to verify the efficiency of the hybrid learning strategy of SLGA, GA-SARSA (only SARSA is combined with GA), GA-Q (only Q-learning is combined with GA) and conventional GA (no RL algorithm is applied) are implemented and compared with SLGA. Ten Brandimarte's data instances are chosen for the experiment whose lower bounds will be used for comparison. In these instances, the number of jobs ranges from 10 to 20, the number of machines varies from 6 to 15, and the number of operations ranges from 58 to 232, each instance is run for 20 times. The parameters for four algorithms are defined as follows: the initial population size is $5 \times m \times n$, the number of iterations is $10 \times m \times n$, and the initial *Pc* and *Pm* are discretionarily selected in the given action set. The learning parameters tested by Alipour *et al.* (Alipour et al., 2017) are applied for GA-SARSA, GA-Q and SLGA, including learning rate $\alpha = 0.75$, discount rate $\gamma = 0.2$, initial reward $r = 1$, and greedy rate ε=0.85.

The dispersion degree of the solutions and the time consumptions of the four algorithms are compared, respectively. The dispersion degree is represented by relative percentage deviation (*RPD*) (Ziaee, 2014; Yuan et al., 2013). All experimental results are shown in Table 3 and Table 4, the first column of Table 4 is 10 instances of different scale, columns 2 to 5 are the best *RPD* values of each instance that solved by the four algorithms, which is calculated by Eq. (13). Columns 6 to 9 are the average *RPD* values of each instance that calculated by Eq. (14). Columns 10 to 13 are the mean time Consumption of four algorithms for every instance. Furthermore, the boldface indicates the best results for each instance.

$$RPD_{Best} = \frac{BSL - BKS}{BKS} \times 100 \tag{13}$$

$$RPD_{Avg} = \frac{ASL - BKS}{BKS} \times 100 \tag{14}$$

where *BSL* and *ASL* represents the best value and the average value obtained by compared algorithms, respectively, and *BKS* represents the lower bounds of Brandimarte's instances that known in the current literatures (Jiang and Zhang, 2018).

As can be seen from Table 4 that the results of SLGA are better than the results of GA-SARSA, GA-Q and conventional GA. For RPD_{Best} and RPD_{Avg}, SLGA has the smallest values for all instances from MK01 to MK10 and is overwhelming for its competitors, which indicates the solutions obtained by SLGA consistently exhibit superior solution quality.

Concerning the time consumption, SLGA has the fastest computation speed among the four algorithms since it performs better on 8 out of 10 test instances, which is due to the more appropriate parameters during solving process can be gained in SLGA, so more redundant crossover and mutation operations can be effectively reduced, meanwhile the quality of the solution can be guaranteed, which can be verified by statistical experiment.

Table 5 gives the average quantity of crossover and mutation operations of four algorithms for instance MK03, where row 2 and row 3 give the quantity of crossover and mutation operations, row 4 and row 5 give percent reduction in SLGA relative to competitors, each algorithm is run for 20 times. From Table 5 we can observe the quantity of crossover and mutation operations in SLGA is the lowest. Therefore, SLGA is regarded as the fastest algorithm and has the lowest time consumption in all four algorithms.

As is shown in Fig. 6, the boxplot of *RPD* for four algorithms in Table 4 is given, which could further validate numerical analysis. It can be observed that the *RPD* values obtained by SLGA have smaller median and range. Table 6 reports the descriptive statistical test of the results in Table 4, where Min represents minimum values and Max is maximum values. It can be obviously observed from the results of statistical test that SLGA have better central tendency than other three algorithms, which denotes SLGA is statistically better than other three algorithms in solving FJSP.

Fig. 7 gives the rate of convergence of MK08 towards the optimal solution that solved by four algorithms. From the figure we can see that SLGA has the fastest rate of convergence, followed by GA-SARSA. GA-Q and conventional GA have relatively poor convergence rates. Meanwhile, it can be observed that the solutions obtained by SLGA is more excellent.

Both GA-SARSA and GA-Q have the self-learning ability, but their performance and time consumption are worse than SLGA due to their respective shortcomings. The solution accuracy of SARSA is poorer, while the learn effect and convergence rate of Q-learning are insufficient. According to Table 3 and Table 4, the quality of the most solutions of GA-SARSA are better than the solutions of GA-Q, which is due to the learning features of SARSA being more suitable for the dynamic characteristic of GA, while Q-learning without pretrain process is unstable and worse solutions would be produced. SLGA combines the advantages of SARSA and Q-learning. At the inital stage of the algorithm, better learning effect and convergence rate of SARSA are inherited by SLGA; at the later stage, better optimization ability is achieved by Q-learning, which make SLGA significantly outperforms GA-SARSA and Q-learning and the significant learning effect of hybrid learning strategy has been verified.

5.2. Experimental comparison with other algorithms

In this part, the computational experiments are given in order to test the performance of SLGA for solving FJSP, whose best results are compared with other algorithms used in the existing literature, including EDPSO (Nouiri et al., 2018), GWO (Jiang and Zhang, 2018); Kacem's GA (Kacem et al., 2002), HA (Sutton and Barto, 1998), KBACO (Xing et al., 2010), GENACE (Ho et al., 2007); PSO + SA (Xing et al.,

Table 3
Comparison to lower bounds of makespan.

Dataset Instance	*BKS*	*BSL*				*ASL*			
		GA-SARSA	GA-Q	GA	SLGA	GA-SARSA	GA-Q	GA	SLGA
MK01	36	41	42	44	**40**	43.2	44.1	46.5	**42**
MK02	24	30	31	36	**27**	31.8	32.3	37.1	**29.7**
MK03	204	205	211	222	**204**	211.7	214.9	228.9	**210.2**
MK04	48	67	75	83	**60**	73.1	78.05	85.4	**66.7**
MK05	168	176	177	191	**172**	185.8	187.4	193.1	**183.5**
MK06	33	72	73	81	**69**	80.3	82.3	84.7	**76.9**
MK07	133	151	155	178	**144**	156.7	159.9	181.6	**151.3**
MK08	523	533	526	542	**523**	539.4	529.6	551.2	**526.6**
MK09	299	338	342	348	**320**	354.6	359.5	359.3	**340.4**
MK10	165	278	281	310	**254**	281.2	287.2	323.5	**269.9**

Table 4
Comparison between GA-SARSA, GA-Q, GA and SLGA.

Dataset Instance	RPD_{Best}				RPD_{Avg}				Time Consumption (second)			
	GA-SARSA	GA-Q	GA	SLGA	GA-SARSA	GA-Q	GA	SLGA	GA-SARSA	GA-Q	GA	SLGA
MK01	13.89	16.67	22.2	**11.11**	20	22.5	29	**16.7**	31.42	32.44	32.86	**27.63**
MK02	25	29.17	50	**12.5**	32.5	34.6	54.6	**23.9**	31.5	35.36	36.36	**29.11**
MK03	0.49	3.43	8.8	**0**	3.8	5.3	12.2	**3.1**	118.2	119.11	122.7	**112.6**
MK04	39.58	56.25	72.9	**25**	52.2	62.6	77.9	**39**	72.8	73.85	76.5	**63.21**
MK05	4.76	5.35	13.7	**2.38**	10.6	11.5	14.9	**9.22**	**59.44**	61.55	63.92	60.35
MK06	118.18	121.2	145	**109.09**	143	149	156.8	**133**	84.81	85.53	89.52	**72.80**
MK07	13.53	16.54	33.8	**8.27**	17.8	20.2	36.5	**13.7**	**56.31**	58.96	62.22	57.77
MK08	1.9	0.57	3.6	**0**	3.1	1.3	5.4	**0.69**	551.4	561.2	578	**521.69**
MK09	13.04	14.4	16.4	**7.02**	18.6	20.2	20.2	**13.8**	582.3	596.73	610.1	**552.5**
MK10	68.48	70.3	87.9	**53.9**	70.4	74	96	**63.6**	1411	1432	1446	**1335.18**

Table 5
Average quantity of crossover and mutation operations.

Operation	GA-SARSA	GA-Q	GA	SLGA
Crossover Operation	173,404	174,934	177,158	**169,330**
Mutation Operation	85,592	87,683	91,789	**83,644**
Decrease for Cross Operation (%)	2.41	3.31	4.62	
Decrease for Mutation Operation (%)	1.15	2.39	4.81	
Total	3.56	5.70	9.43	

2009), TS (Brandimarte, 1993), MATSPSO (Henchiri and Ennigrou, 2013), MACROG (Marzouki et al., 2017). The summary of computational results for the makespan is given in Tables 7 and 8, where the symbol '-' indicates that the solution is not available in the literature, the *BKS* represents the best solutions in the current literatures and the boldface denotes the best results for each instance.

From Table 7 we can see that the best solutions for all the three instances can be obtained by SLGA, GWO and KBACO, only the best solution of instance 4 × 5 can be found by EDPSO, HA, GENACE, and PSO + SA, while Kacem' GA and GA cannot get any optimal solutions. The comparison results prove that SLGA has a good performance in small scale problems, and has better performance than conventional GA.

To further verified the performance of SLGA on medium and large scale FJSP problems, Brandimarte's ten instances from MK01 to MK10 (Brandimarte, 1993) are used for comparison with published algorithms, and the calculation results of the makespan are shown in Table 8. It can be observed that SLGA can get 8 best results out of ten results. MATSPSO, HA and GWO can find 3 optimal results, respectively. TS and MACROG can obtain only one best value. While GA cannot offer any.

Table 9 gives the *RPD* values of all makespan in Table 8, it can be observed that SLGA can find the best solutions for MK02, MK03, MK04, MK05, MK06, MK07, MK08 and MK09, and the solutions closed to the optimal results can be produced by SLGA for the other three instances. Moreover, the mean *RPD* value is 22.9%, which is the lowest value in all 7 algorithms, which denotes the better performance of SLGA for medium and large scale FJSP problems. Fig. 8 depicts the over distribution of all results for *RPD* values, it can be seen from figure that the *RPD* values of SLGA is lower and more concentrated, which also indicates SLGA

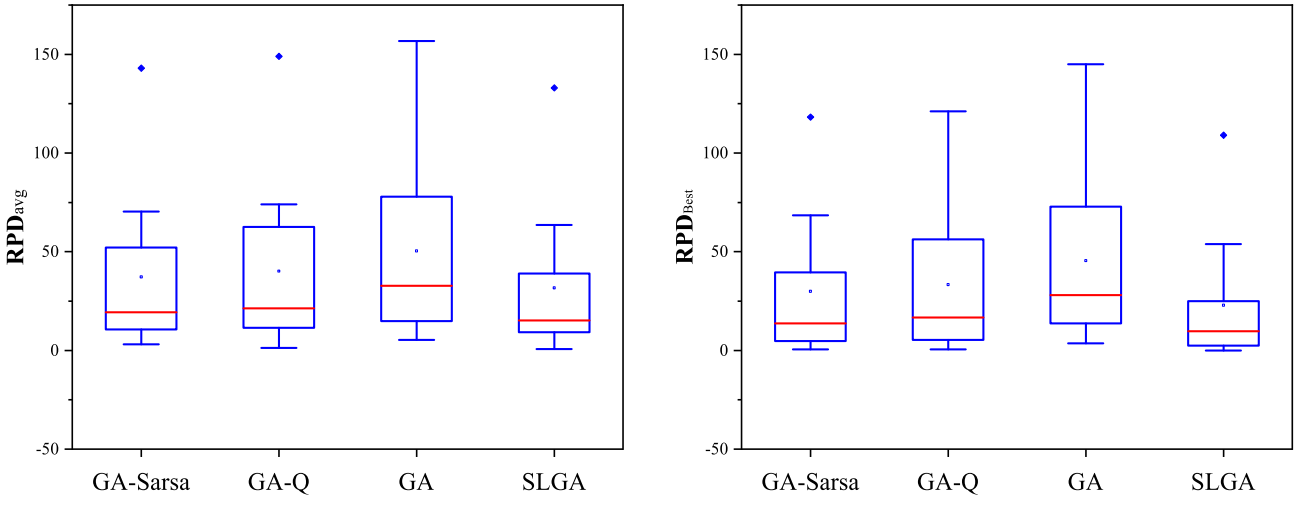

Fig. 6. Boxplot of RPD for four algorithms in Table 4.

Table 6
The results of descriptive statistical test.

Algorithms	RPD_{Best}					RPD_{Avg}				
	Min	Max	Median	Mean	Standard deviation	Min	Max	Median	Mean	Standard deviation
GA-SARSA	0.49	118.18	13.71	29.89	37.25	3.1	143	19.3	37.2	42.88
GA-Q	0.57	121.2	16.61	33.39	38.44	1.3	149	21.35	40.12	44.95
GA	3.6	145	28	45.43	44.86	5.4	156.8	32.75	50.35	47.7
SLGA	**0**	**109.09**	**9.69**	**22.93**	**34.28**	**0.69**	**133**	**15.25**	**31.67**	**40.2**

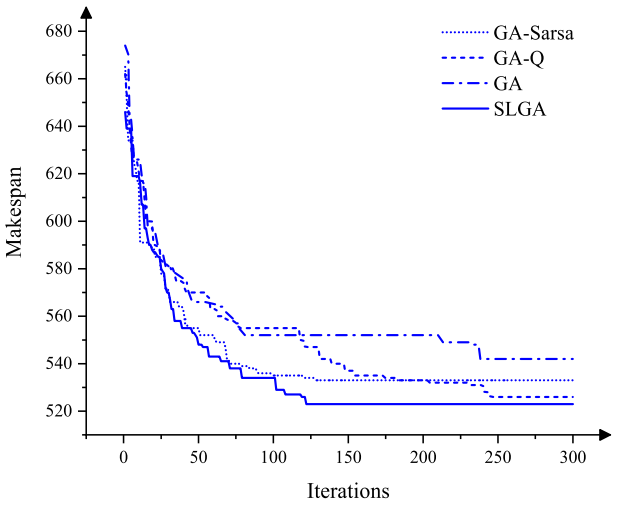

Fig. 7. The rate of convergence for MK08.

outperforms other algorithms in addressing FJSP.

In order to get a statistically significant comparison and find significant differences among the results obtained by proposed SLGA and other 6 algorithms, the *RPD* values of all algorithms are carried out with the significance test. The Friedman test (Cheraghalipour et al., 2018) is performed in this experiment, which is a non-parametric tests that used when the results present either non-normal distribution or non-homogeneity of variance. Before it, the test of normality with Shapiro-Wilk and homogeneity of variance with one way ANOVE is performed and the results are presented in Table 10. The results show that the *RPD* values are not normal distribution except GA, and they are homogeneity of variance.

The Friedman tested results are offered in Table 11, it can be observed that proposed SLGA has the best mean rank and highest priority over other 6 algorithms for *RPD* values. Moreover, the *p-value* is 0.000025 and closes to zero, which significantly less than the level of significant $\alpha = 0.05$. The *p-value* proves that there are significant differences in the optimization performance of the 7 algorithms on each instance. Fig. 9 gives the results of Mean Difference with Least Significance Difference (LSD), where the *RPD* values of SLGA are compared with other algorithms in pairs, it can be observed that the mean of SLGA is lower than the others. The Friedman test and LSD test agreement with the results of Table 9 and further verifies the outstanding performance of SLGA versus its competitors in solving FJSP.

6. Conclusion

There are many algorithms for addressing FJSP but the limitation of parameter adjustment has not been solved effectively, the goal of this paper is to establish a method of parameter intelligent adjustment for GA in solving FJSP. Therefore, a self-learning genetic algorithm (SLGA) to

Table 9
The *RPD* values (%) for the experiment results of Table 8.

Instance	TS	MATSPSO	HA	GWO	MACROG	GA	SLGA
Mk01	16.7	**8.3**	16.7	11.1	11.1	22.2	11.1
Mk02	33.3	**12.5**	16.7	20.8	33.3	50	**12.5**
Mk03	3.4	1.5	**0**	**0**	**0**	8.8	**0**
Mk04	69	35.4	56.3	33.3	33.3	72.9	**25**
Mk05	10.7	3.6	6.6	4.2	6.5	13.7	**2.4**
Mk06	161	118	**109**	**109**	157	145.5	**109**
Mk07	18.1	15.8	12	10.5	29.3	33.8	**8.3**
Mk08	**0**	**0**	6.1	**0**	5.5	3.6	**0**
Mk09	23.4	13.7	14.4	7.7	40.8	16.4	**7**
Mk10	79.4	81.2	**46.7**	50.9	117	87.9	53.9
Mean	41.4	29	28.4	24.8	43.5	45.5	**22.9**

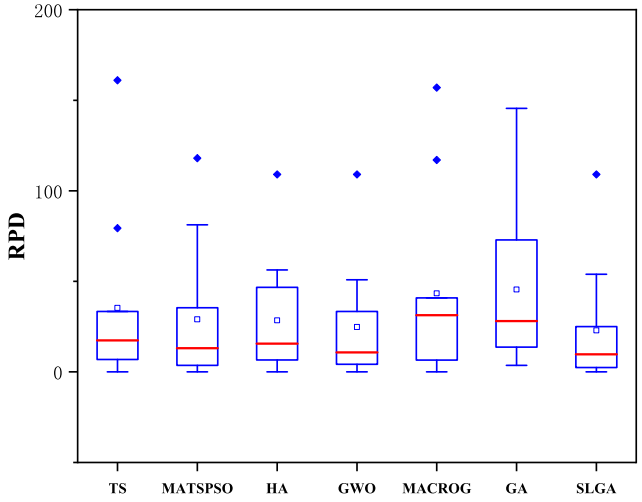

Fig. 8. Boxplot of *RPD* for seven algorithms in Table 8.

Table 7
The makespan for three instances of Kacem.

Dataset Instance	*BKS*	EDPSO	GWO	Kacem's GA	HA	KBACO	GENACE	PSO + SA	GA	SLGA
4 × 5	11	**11**	**11**	16	**11**	**11**	**11**	**11**	16	**11**
8 × 8	14	17	**14**	–	15	**14**	–	15	16	**14**
10 × 7	11	–	**11**	15	13	**11**	12	–	15	**11**

Table 8
The makespan for ten instances of Bandimarte.

Instance	*BKS*	TS	MATSPSO	HA	GWO	MACROG	GA	SLGA
Mk01	36	42	**39**	42	40	40	44	40
Mk02	24	32	**27**	28	29	32	36	**27**
Mk03	204	211	207	**204**	**204**	**204**	222	**204**
Mk04	48	81	65	75	64	64	83	**60**
Mk05	168	186	174	179	175	179	191	**172**
Mk06	33	86	72	**69**	**69**	85	81	**69**
Mk07	133	157	154	149	147	172	178	**144**
Mk08	523	**523**	**523**	555	**523**	552	542	**523**
Mk09	299	369	340	342	322	421	348	**320**
Mk10	165	296	299	**242**	249	358	310	254

Table 10
The test of normality and homogeneity of variance (the level of significant $\alpha = 0.05$).

		TS	MATSPSO	HA	GWO	MACROG	GA	SLGA
Normality	Statistic	0.7	0.737	0.77	0.745	0.767	0.854	0.699
	df	10	10	10	10	10	10	10
	Significant	0.001	0.002	0.006	0.003	0.006	0.065	0.001
Homogeneity of variance	Statistic		df_1		df_2		Significant	
	0.445		6		63		0.846	

Table 11
The results of Friedman tests for *RPD* values (the level of significant $\alpha = 0.05$).

Algorithm	Value of the mean rank	Final priority
TS	4.85	5
MATSPSO	3.30	3
HA	3.90	4
GWO	2.65	2
MACROG	5.05	6
GA	6.30	7
SLGA	1.95	1
Test statistics	Friedman	
N	10	
Chi-Square	31.00	
df	6	
p-value	0.000025	

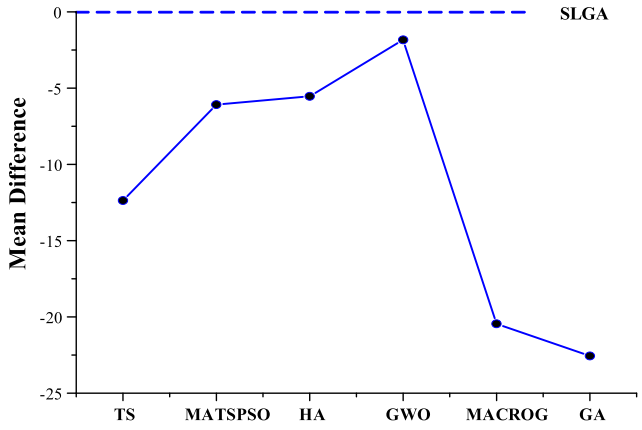

Fig. 9. The Mean Difference with LSD that comparing SLGA with other algorithms.

solve FJSP problem is presented in this paper. In SLGA, Reinforcement Learning (RL) is adopted to precisely adjust *Pc* and *Pm* of GA, which significantly improves the efficiency for solving FJSP. In learning module of SLGA, SARSA algorithm and Q-learning algorithm are applied in different execution phases to merge their own advantages, so both faster learning speed and higher solution precision can be achieved. RL environment is subtly designed for SLGA, including the state set of GA, the action set and the reward method.

In order to verify the learning effect of the proposed SLGA, a large number of comparative experiments are conducted between GA-SARSA, GA-Q, conventional GA and SLGA by using Brandimarte's data instances. Meanwhile, SLGA is statistically compared with other state-of-the-art algorithms that existing in literatures. According to the comparison results, SLGA is capable of obtaining better solutions for most instances, the learning effect and the excellent performance of SLGA are confirmed.

Although the proposed SLGA shows an outstanding performance in solving FJSP, there is still unknown for solving different scheduling problems. Moreover, some priori parameters usage is not considered deeply. In our future work, we will continue to deeply research the promotion of RL and other algorithms for other different combinatorial optimization problems and the use of priori parameters.

CRediT authorship contribution statement

Ronghua Chen: Writing - original draft, Writing - review & editing, Formal analysis, Software. **Bo Yang:** Conceptualization, Methodology, Writing - review & editing, Funding acquisition. **Shi Li:** Validation, Investigation, Funding acquisition. **Shilong Wang:** Resources, Supervision, Funding acquisition.

Acknowledgments

The presented work was supported by the Key Technologies Research and Development Program (no. 2018AAA0101804), the National Defense Basic Scientific Research Program of China (no. JCKY2016204A502) and the Key Project of Technological Innovation and Application Development Plan of Chongqing (no. cstc2019jscx-mbdxX0056).

Appendix A. Supplementary material

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cie.2020.106778.

References

Acan, A., & Ünveren, A. (2015). A great deluge and tabu search hybrid with two-stage memory support for quadratic assignment problem. *Applied Soft Computing, 36*, 185–203.

Aissani, N., Bekrar, A., Trentesaux, D., & Beldjilali, B. (2011). Dynamic scheduling for multi-site companies: A decisional approach based on reinforcement multi-agent learning. *Journal of Intelligent Manufacturing, 23*, 2513–2529.

Al-Hinai, N. (2011). An efficient hybridized genetic algorithm architecture for the flexible job shop scheduling problem. *Flexible Services and Manufacturing Journal, 23*, 64–85.

Alipour, M. M., Razavi, S. N., Feizi Derakhshi, M. R., & Balafar, M. A. (2017). A hybrid algorithm using a genetic algorithm and multiagent reinforcement learning heuristic to solve the traveling salesman problem. *Neural Computing and Applications, 30*, 2935–2951.

Amjad, M. K., Butt, S. I., Kousar, R., Ahmad, R., Agha, M. H., Faping, Z., … Asgher, U. (2018). Recent research trends in genetic algorithm based flexible job shop scheduling problems. *Mathematical Problems in Engineering, 2018*, 1–32.

Bashir, M. B., & Nadeem, A. (2017). Improved genetic algorithm to reduce mutation testing cost. *IEEE Access, 5*, 3657–3674.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research, 41*(1993), 1157–1183.

Brucker, P., & Schlie, R. J. C. (1990). Job-shop scheduling with multi-purpose machines. *Computing, 45*, 369–375.

Chang, H. C., Chen, Y. P., Liu, T. K., & Chou, J. H. (2015). Solving the flexible job shop scheduling problem with makespan optimization by using a hybrid Taguchi-genetic algorithm. *IEEE Access, 3*, 1740–1754.

Cheraghalipour, A., Hajiaghaei-Keshteli, M., & Paydar, M. M. (2018). Tree Growth Algorithm (TGA): A novel approach for solving optimization problems. *Engineering Applications of Artificial Intelligence, 72*, 393–414.

Choi, I. C., & Choi, D. S. (2002). A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers & Industrial Engineering, 42*, 43–58.

Du, Y., Fang, J., & Miao, C. (2014). Frequency-domain system identification of an unmanned helicopter based on an adaptive genetic algorithm. *IEEE T Ind Electron, 61*, 870–881.

Emary, E., Zawbaa, H. M., & Grosan, C. (2018). Experienced gray wolf optimization through reinforcement learning and neural networks. *IEEE Trans Neural Netw Learn Syst, 29*, 681–694.

Y.G. Fei Chen, Zhao-qian Chen, and Shi-fu Chen, SCGA: Controlling Genetic Algorithms with Sarsa(0), Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference

on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05), (2005).
Gao, K., Yang, F., Zhou, M., Pan, Q., & Suganthan, P. N. (2019). Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm. *IEEE Transactions on Cybernetics, 49*, 1944–1955.
M. Gyoung Hwan Kim, Genetic Reinforcement Learning Approach to the Heterogeneous Machine Scheduling Problem, IEEE Transactions On Robotics and Automation, 14 (1998) 879-893.
Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research, 1*, 117–129.
Henchiri, A., & Ennigrou, M. (2013). Particle swarm optimization combined with Tabu search in a multi-agent model for flexible job shop problem. *Proc. ICS I, 385–394*, 2013.
Ho, N. B., Tay, J. C., & Lai, E. M. K. (2007). An effective architecture for learning and evolving flexible job-shop schedules. *European Journal of Operational Research, 179*, 316–333.
Hsieh, Y.-Z., & Su, M.-C. (2015). A Q-learning-based swarm optimization algorithm for economic dispatch problem. *Neural Computing and Applications, 27*, 2333–2350.
Jiang, T. H., & Zhang, C. (2018). Application of grey wolf optimization for solving combinatorial problems: Job shop and flexible job shop scheduling cases. *IEEE Access, 6*, 26231–26240.
Kacem, I., Hammadi, S., Borne, P., & Man, P. C. (2002). Cybernetics, Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. *IEEE Transactions on Systems, Man, 32*, 1–13.
Karimi, H., Rahmati, S. H. A., & Zandieh, M. (2012). An efficient knowledge-based algorithm for the flexible job shop scheduling problem. *Knowledge-Based Systems, 36*, 236–244.
Lei, D. M. (2012). Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. *Applied Soft Computing, 12*, 2237–2245.
Li, X. Y., & Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics, 174*, 93–110.
Li, X. X., Peng, Z., Du, B. G., Guo, J., Xu, W. X., & Zhuang, K. J. (2017). Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems. *Computers & Industrial Engineering, 113*, 10–26.
Li, T. H., Su, Y. T., Lai, S. W., & Hu, J. J. (2011). Walking motion generation, synthesis, and control for biped robot by using PGRL LPI, and fuzzy logic. *IEEE Transactions on Systems, Man, and Cybernetics. Part B, Cybernetics, 41*, 736–748.
Liouane, N., Saad, I., Hammadi, S., & Borne, P. (2007). Ant systems & local search optimization for flexible job shop scheduling production. *International Journal of Computers, Communications & Control, II, I*, 174–184.
Liu, T. K., Chen, Y. P., & Chou, J. H. (2014). Solving distributed and flexible job-shop scheduling problems for a real-world fastener manufacturer. *IEEE Access, 2*, 1598–1606.
B. Marzouki, O.B. Driss, K. Ghedira, Multi Agent model based on Chemical Reaction Optimization with Greedy algorithm for Flexible Job shop Scheduling Problem, 21th International Conference on Knowledge Based and Intelligent Information and Engineering Systems, KES2017, 6-8 September 2017, Marseille, France, (2017) 81-90.
Meng, T., Pan, Q. K., & Sang, H. Y. (2018). A hybrid artificial bee colony algorithm for a flexible job shop scheduling problem with overlapping in operations. *International Journal of Production Research, 56*, 5278–5292.
Nouiri, M., Bekrar, A., Jemai, A., Niar, S., & Ammari, A. C. (2018). An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing, 29*, 603–615.
Nouiri, M., Bekrar, A., Jemai, A., Trentesaux, D., Ammari, A. C., & Niar, S. (2017). Two stage particle swarm optimization to solve the flexible job shop predictive scheduling problem considering possible machine breakdowns. *Computers & Industrial Engineering, 112*, 595–606.
Pezzella, F., Morganti, G., & Ciaschetti, G. (2008). A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research, 35*, 3202–3212.
Qin, J., Li, M., Shi, Y., Ma, Q., & Zheng, W. X. (2019). Optimal synchronization control of multiagent systems with input saturation via off-policy reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems, 30*, 85–96.
Shahrabi, J., Adibi, M. A., & Mahootchi, M. (2017). A reinforcement learning approach to parameter estimation in dynamic job shop scheduling. *Computers & Industrial Engineering, 110*, 75–82.
Shao, G. F., Shangguan, Y. L., Tao, J. P., Zheng, J. W., Liu, T. D., & Wen, Y. H. (2018). An improved genetic algorithm for structural optimization of Au-Ag bimetallic nanoparticles. *Applied Soft Computing, 73*, 39–49.
Sutton, Richard S., & Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press.
Wang, Y., & de Silva, C. W. (2008). A machine-learning approach to multi-robot coordination. *Engineering Applications of Artificial Intelligence, 21*, 470–484.
Wang, H., Gu, M., Yu, Q., Tao, Y., Li, J., Fei, H., … Hong, T. (2019). Adaptive and large-scale service composition based on deep reinforcement learning. *Knowledge-Based Systems, 180*, 75–90.
Wang, Y.-H., Li, T.-H. S., & Lin, C.-J. (2013). Backward Q-learning: The combination of Sarsa algorithm and Q-learning. *Engineering Applications of Artificial Intelligence, 26*, 2184–2193.
Wang, Y. M., Yin, H. L., & Qin, K. D. (2013). A novel genetic algorithm for flexible job shop scheduling problems with machine disruptions. *The International Journal of Advanced Manufacturing Technology, 68*, 1317–1326.
Wang, F., Zhou, L., Ren, H., & Liu, X. (2017). Search improvement process-chaotic optimization-particle swarm optimization-elite retention strategy and improved combined cooling-heating-power strategy based two-time scale multi-objective optimization model for stand-alone microgrid operation. *Energies, 10*.
Wei, X.-K., Zhang, C., Wang, B.-Z., Li, J.-L., & Shao, W. (2014). Improved self-adaptive genetic algorithm with quantum scheme for electromagnetic optimisation. *IET Microwaves, Antennas & Propagation, 8*, 965–972.
Xing, L. N., Chen, Y. W., Wang, P., Zhao, Q. S., & Xiong, J. (2010). Knowledge-based ant colony optimization for flexible job shop scheduling problems. *Applied Soft Computing, 10*, 888–896.
Xing, L. N., Chen, Y. W., & Yang, K. W. (2009). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering, 56*, 1735–1736.
Yuan, Y., Xu, H., & Yang, J. D. (2013). A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing, 13*, 3259–3272.
Zandieh, M., & Karimi, N. (2010). An adaptive multi-population genetic algorithm to solve the multi-objective group scheduling problem in hybrid flexible flowshop with sequence-dependent setup times. *Journal of Intelligent Manufacturing, 22*, 979–989.
Zhang, G. H., Gao, L., & Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications, 38*, 3563–3573.
Zhang, G. H., Shao, X. Y., Li, P. G., & Gao, L. (2009). An effective hybrid particle swarm optimization algorithm for multi-objective flexible job-shop scheduling problem. *Computers & Industrial Engineering, 56*, 1309–1318.
Zhang, C. Y., Wang, X. J., & Gao, L. (2010). An Improved Genetic Algorithm for Multi-objective Flexible Job-shop Scheduling Problem. *Manufacturing Science and Engineering, 97–101*, 2449–2454.
Zhicong Zhang, L. Z., Li, Na., & Wang, Weiping (2012). Shouyan Zhong, Kaishun Hu, Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning. *Computers & Operations Research, 39*, 1315–1324.
Ziaee, M. (2014). A heuristic algorithm for solving flexible job shop scheduling problem. *The International Journal of Advanced Manufacturing Technology, 71*, 519–528.