Collaborative Multi-fidelity Based Surrogate Models for Genetic Programming in Dynamic Flexible Job Shop Scheduling

Fangfang Zhang, *Graduate Student Member, IEEE,* Yi Mei, *Senior Member, IEEE,*
Su Nguyen, *Member, IEEE,* and Mengjie Zhang, *Fellow, IEEE*

***Abstract*—Dynamic flexible job shop scheduling has received widespread attention from academia and industry due to its practical application value. It requires complex routing and sequencing decisions under unpredicted dynamic events. Genetic programming, as a hyper-heuristic approach, has been successfully applied to evolve scheduling heuristics for job shop scheduling due to its flexible representation. However, the simulation-based evaluation is computationally expensive since there are many calculations based on individuals for making decisions in the simulation. To improve training efficiency, this paper proposes a novel multi-fidelity based surrogate-assisted genetic programming. Specifically, multi-fidelity based surrogate models are first designed by simplifying the problem expected to be solved. In addition, this paper proposes an effective collaboration mechanism with knowledge transfer for utilising the advantages of multi-fidelity based surrogate models to solve the desired problems. This paper examines the proposed algorithm in six different scenarios. The results show that the proposed algorithm can dramatically reduce the computational cost of genetic programming without sacrificing the performance in all scenarios. With the same training time, the proposed algorithm can achieve significantly better performance than its counterparts in most scenarios while no worse in others.**

***Index Terms*—Collaboration, Multi-fidelity Based Surrogate Models, Knowledge Transfer, Genetic Programming, Dynamic Flexible Job Shop Scheduling.**

I. Introduction

Job shop scheduling (JSS) [1] is one of the best-known combinatorial optimisation problems, which is of great scientific research value since it reflects the real-world applications such as manufacturing processes [2], [3] and cloud computing [4]. Given a number of jobs (each job consists of a sequence of operations) and a set of machines (each operation can only be processed by a specific machine), the purpose of JSS is to find an effective schedule which is an allocation of the operations to time intervals on the machines. Flexible JSS (FJSS) [5] is a common relaxation of JSS where each operation can be processed on multiple machines. For FJSS, we need to make two decisions simultaneously. One is *machine assignment* (i.e. assign an operation to a particular machine), and the other is *operation sequencing* (i.e. choose an operation as the next operation to be processed by an idle machine). Dynamic FJSS (DFJSS) focuses on optimising the machine resources under a dynamic environment with stochastic events, such as real-time new jobs arrivals [6], [7], [8] and machine breakdown [9], [10].

DFJSS is an NP-hard problem [11] and cannot be handled efficiently with *exact optimisation methods*, such as dynamic programming [12] and integer linear programming [13]. *Approximate solution optimisation methods*, such as simulated annealing [14], tabu search [15] and genetic algorithms [16], which aim to find a near-optimal solution, have been widely applied to JSS. These methods can obtain high-quality solutions in a reasonable time. However, most of them can hardly handle dynamic environments efficiently because the re-optimisation process is too time-consuming to react in real-time. *Scheduling heuristics* such as dispatching rules [17], [18], [19], might be the most popularly used heuristics for DFJSS. Scheduling heuristics make decisions according to the priority values of machines or operations only at the decision points. It is noted that a scheduling heuristic in DFJSS consists of a *routing rule* (i.e. for machine assignment) and a *sequencing rule* (i.e. for operation sequencing) in this paper. There are two main reasons for the success of scheduling heuristics in DFJSS [6], [7]. One is its ability to handle large scale problems efficiently. The other is its efficiency to make real-time decisions with dynamic events. However, the scheduling heuristics, such as SPT (i.e. shortest processing time) and some composite rules [20], are often *manually* designed by experts. The designing process is time-consuming, and the designed rules are typically too specific to be applied to different scenarios.

Genetic programming (GP) [21], as a hyper-heuristic approach, has been successfully applied to evolve scheduling heuristics *automatically* for JSS [8], [22], [23], [24], [25], [26]. The most advantageous feature of GP is its representation,

Manuscript received January 1, 2020; revised September 14, 2020; accepted January 1, 2021. This work was supported in part by the Marsden Fund of New Zealand Government under Contracts VUW1509 and VUW1614, the Science for Technological Innovation Challenge (SfTI) fund under grant E3603/2903, and the MBIE SSIF Fund under Contract VUW RTVU1914. This work of Fangfang Zhang was supported by the China Scholarship Council (CSC)/Victoria University Scholarship. This article was recommended by Associate Editor XXX. *(Corresponding author: Fangfang Zhang.)*

Fangfang Zhang, Yi Mei, and Mengjie Zhang are with the Evolutionary Computation Research Group, School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: fangfang.zhang@ecs.vuw.ac.nz; yi.mei@ecs.vuw.ac.nz; mengjie.zhang@ecs.vuw.ac.nz).

Su Nguyen is with the Centre for Data Analytics and Cognition, La Trobe University, Melbourne, VIC 3086, Australia (e-mail: P.Nguyen4@latrobe.edu.au).

This article has supplementary downloadable material available at XXX, provided by the authors.

Colour versions of one or more of the figures in this article are available online at XXX.

Digital Object Identifier XXX

which makes it a natural fit for generating scheduling heuristics. It is not necessary to define the structures of scheduling heuristics in advance. In fact, we usually do not know what the optimal structure is. However, the main drawback of GP is its high computational cost since the evaluation of GP individuals is time-consuming. Simulation [27] is a promising technique to investigate the complex real-world problems such as health care [28]. The simulation-based research in DFJSS further increases the need for computing cost since more calculations with GP individuals are involved to make decisions during the process of simulation.

To the best of our knowledge, little is yet known to improve the efficiency of GP in JSS. This paper groups the existing works related to surrogate-assisted GP for JSS into two categories according to the way of using the surrogate model to estimate the fitness of an individual. One is to use existing models such as KNN (i.e. K nearest neighbour) [29] to estimate the fitness of individuals [30], [31] by finding the most similar individual in the previous generation. The other is to use a simplified simulation model as a surrogate model, which is a problem approximation technique to estimate the fitness of individuals [32], [33]. In [30], [31] and [32], the surrogates are used in a pre-selection way that a larger number of offspring are generated, and only the top individuals are selected to be re-evaluated with real fitness evaluations. In [33], surrogates with different fidelities are used to evaluate the individuals directly by increasing the fidelity of applied surrogate model gradually along with the search process.

In general, studies mentioned above all show the superiority of introducing the surrogate technique in JSS. However, the performance of the algorithms highly rely on the accuracy of proposed surrogate models, either based on similar rules [30], [31] or simplified models [32], [33]. In addition, pre-selection is a way to speed up the convergence of an algorithm by adding more evaluations but with the low computational cost of extra individuals [30], [31], [32]. In fact, more evaluations of GP individuals are involved. Last but not least, in [33], although multi-fidelity surrogates are successfully introduced, each surrogate is used independently at different generations. On the one hand, the performance at a specific generation is sensitive to the surrogate used in that generation. On the other hand, it may not be an effective way since there is no communication between surrogates with multi-fidelity. More advanced techniques are worth investigating.

To address the above issues, this paper proposes to use multiple surrogate models with different fidelities collaboratively to improve the efficiency of GP without losing its accuracy performance to evolve scheduling heuristics for DFJSS automatically. Specifically, multi-fidelity surrogates are built by simplifying the DFJSS problem to be solved in this paper. Involving more surrogates aims to weaken the sensitive relationship between the applied surrogate and the investigated problem. In addition, an effective collaboration framework with knowledge transfer is proposed to utilise the advantages of surrogate models with different fidelities.

The goal of this paper is to *develop an effective multi-fidelity based surrogate models to improve the efficiency of GP for evolving scheduling heuristics automatically in the DFJSS problems.* The proposed algorithm is expected to both speed up the convergence and reduce the training time of GP for DFJSS. Specifically, this work has the following research objectives:

1) Develop multiple surrogate models with different fidelities by simplifying the problem expected to be solved according to its characteristics.
2) Propose an effective collaboration framework with knowledge transfer for the designed multi-fidelity based surrogate models to solve the desired problem.
3) Analyse the efficiency and effectiveness of the proposed algorithm in terms of the training time, the convergence speed, and the performance of evolved rules.
4) Analyse how the proposed algorithm influences the behaviour of GP in terms of the effect of the knowledge transfer mechanism.

The rest of this paper is organised as follows. Section II gives a background introduction. Section III gives a discussion of the related studies. Detailed descriptions of the proposed algorithm are given in Section IV. The experiment design is shown in Section V, followed by results and discussions in Section VI. Further analyses are conducted in Section VII. Finally, Section VIII concludes the paper.

II. Background

This section provides a brief introduction of DFJSS, scheduling heuristics for DFJSS, and how to use GP for solving the DFJSS problem.

A. Dynamic Flexible Job Shop Scheduling

In FJSS, n jobs $J = \{J_1, J_2, ..., J_n\}$ need to be processed by m machines $M = \{M_1, M_2, ..., M_m\}$. Each job J_j has a sequence of operations $O_j = (O_{j1}, O_{j2}, ..., O_{ji})$. Each operation O_{ji} can only be processed by one of its optional machines $\pi(O_{ji})$ and its processing time $\delta(O_{ji})$ depends on the machine that processes it. It implies that there are two decisions in FJSS which are routing decision and sequencing decision. In DFJSS, not only the two decisions need to be made simultaneously, but also dynamic events are necessary to be taken into account when making schedules. This paper focuses on one dynamic event, i.e. dynamically and continuously arriving new jobs. The jobs arrive at the job shop following a Poisson process (i.e. its arrival time to the job shop is randomly generated from a Poisson process). The information of a job (e.g. arrival time, processing time and due date) is unknown until it arrives at the shop floor. The following constraints must be satisfied in the DFJSS problem:

- A machine can process at most one operation at a time.
- Each operation can be processed only by one of its candidate machines.
- One cannot start processing an operation until all its precedent operations have been processed.
- The processing of an operation cannot be stopped or paused until it is completed.

The goal of DFJSS is to optimise certain objective functions while satisfying all the above constraints. We consider three commonly used objective functions which are shown as follows.

- Max-flowtime $= max\{C_1 - r_1, C_2 - r_2, ..., C_n - r_n\}$
- Mean-flowtime $= \frac{\sum_{i=1}^{n}\{C_i - r_i\}}{n}$
- Mean-weighted-flowtime $= \frac{\sum_{i=1}^{n} w_i * \{C_i - r_i\}}{n}$

where C_i is the completion time of job J_i, r_i is the release time of J_i, and w_i is the weight of J_i.

B. Scheduling Heuristics for DFJSS

In DFJSS, one job consists of a sequence of operations. The operations of a job need to be processed one by one following the sequence constraint. The completion of a job means that all its operations have been processed. We use the term *ready operations* to define the candidate operations for the routing process. Naturally, two kinds of operations can become ready operations. One is the first operation of a job when it arrives at the job shop. This indicates that the jobs will be released once they arrive at the job shop. The other is the subsequent operation whose preceding operation is just finished.

In DFJSS, the decisions are only made at the *decision points* by scheduling heuristics based on the current information of the job shop. Once an operation becomes a ready operation (*routing decision point*), it will be allocated to the most prior machine according to the routing rule. When a machine becomes idle, and its queue is not empty (*sequencing decision point*), the sequencing rule will be applied to calculate the priority value of each operation in its queue. The most prior operation is then chosen as the next operation to be processed. It is noted that the queues of the machines act as intermediate storages. For a job, it will be moved to one of its candidate machines, if its current operation is finished and its next operation needs to be processed on another machine.

C. Genetic Programming Hyper-heuristics for DFJSS

A hyper-heuristic [34], [35], [36] is a heuristic search method that seeks to select or generate heuristics to efficiently solve hard computational search problems. The unique characteristic is that hyper-heuristic works on heuristic search space instead of solution search space. GP, as a hyper-heuristic method [37], has been successfully applied to evolve informative scheduling heuristics for combinational optimisation problems such as packing [38], [39], timetabling [40], [41], and JSS [42], [43], [44]. GP can automatically generate computer programs to solve problems without needing rich domain knowledge. The flexible representation of GP makes it natural to be a hyper-heuristic approach to evolving scheduling heuristics in JSS. This means that we do not need to define the structure of rules in advance. This is beneficial because we usually do not know what the optimal structures of the rules look like.

The pseudo-code of traditional GP to evolve scheduling heuristics for DFJSS is shown in Algorithm 1. The input is a problem, and the output is the evolved best routing and sequencing rule. The three main components in GP are initialisation, evaluation, and evolution. We start with initialising the population and evaluating all the GP individuals. If the stopping criterion is not met, offspring are generated by the genetic operators to form a new population. The best routing and sequencing rules (r^*, s^*) in the last generation of a GP run are the output of the algorithm. In Algorithm 1, each individual is evaluated by applying it to a *training instance* to get the objective value of the obtained schedule (from line 6 to line 9). During the DFJSS simulation, the GP individuals are used as schedule heuristics including routing and sequencing rules (i.e. priority functions) to prioritise machines and operations to make a schedule.

Algorithm 1: Pseudo-code of GP to learn routing and sequencing heuristics for DFJSS

```
   Input  : A problem
   Output: The learned scheduling heuristics h* = (r*, s*)
1: Initialisation: Randomly initialise the population
2: set h* ← (null, null), fitness(h*) ← +∞
3: gen ← 0
4: while gen < maxGen do
5:     // Evaluation: Evaluate the individuals in the population
6:     for i = 1 to popsize do
7:         Run a DFJSS simulation with h_i to get the schedule
             Schedule_i
8:         fitness(h_i) ← Obj(Schedule_i)
9:     end
10:    if gen < maxGen − 1 then
11:        Evolution: Generate a new population by crossover,
             mutation, and reproduction
12:    end
13:    gen ← gen + 1
14: end
15: for i = 1 to popsize do
16:    if fitness(h_i) < fitness(h*) then
17:        h* ← h_i
18:        fitness(h*) ← fitness(h_i)
19:    end
20: end
21: return h* = (r*, s*)
```

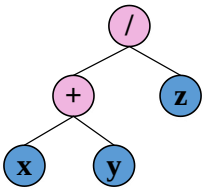


Fig. 1. An example of a routing rule for prioritising machines in a DFJSS simulation.

Fig. 1 shows an example of a routing rule for prioritising machines in a DFJSS simulation. The routing rule consists of three terminals (i.e. x, y, and z) and two functions (i.e. + and /), and it can be considered as a priority function $\frac{x+y}{z}$. Assume there are 3 candidate machines (i.e. M_1, M_2, M_3) for a job. If the function values of M_1, M_2, M_3 are 100, 300, and 200 according to the priority function, the job will be allocated to M_1 (i.e. A smaller function value leads to a higher priority).

Training and test are two phases of our research. In the training phase, GP is used to train heuristics based on a set of training instances. It is worth mentioning that the outputs of the training phase are general heuristics (i.e. routing and sequencing rules) rather than specific solutions in DFJSS. In the test phase, the evolved heuristics obtained in the training phase are tested on *unseen test instances* to generate the corresponding schedules. According to the information of schedules, the test performance of evolved heuristics (i.e. objective value such as flowtime) can be measured.

III. Surrogate Models in GP for JSS

In the past decades, surrogate-assisted evolutionary computation [45], [46], [47] has been widely studied to reduce the computational cost of evolutionary algorithms. The basic idea is to use computationally cheap surrogate models to replace part of the computationally expensive evaluations of individuals. The commonly used techniques for surrogate models include radial basis function networks [48] and kriging model [49]. However, there are some challenges which make these kinds of surrogate techniques not directly applicable in DFJSS. The task (i.e. prioritising operations or machines) and the training instances in DFJSS are different from the traditional machine learning tasks such as regression [50] and numerical optimisation [51]. The training data in DFJSS are dynamically generated along with the execution of simulation while the training data are available in advance in traditional machine learning tasks. In addition, although we can collect data during the process of simulation, it is not trivial to decide what kinds of information are useful for training the surrogate model, and how to represent the collected data.

There is little research about surrogate-assisted GP for solving the JSS problems. Hildebrandt and Branke [30] proposed to approximately estimate the fitness of individuals by finding the most similar rule in the previous generation. However, the best value in the last generation is an upper bound of estimated fitness, which is not able to totally reflect the quality of a new individual. The proposed surrogate approach was applied in a pre-selection way in which a larger number of individuals were generated to form an intermediate population, and only the top individuals were selected into the next generation for real fitness evaluations. Nguyen et al. [32] also used the pre-selection way for the surrogate with a simplified model based surrogate-assisted GP for dynamic JSS. The novelty of this work lied on the surrogate models based on simplified simulation models of the job shop. It showed the effectiveness of using simplified simulation models. Zhang et al. [33] further proposed to use an adaptive surrogate strategy with dynamic fidelities of simulation models over generation to estimate the fitness of individuals in the population directly rather than in the intermediate population for DFJSS.

The works mentioned above show the superiority of using surrogate-assisted GP to handle the JSS problems. However, there are still some limitations. *From the perspective of the way to incorporate surrogate into GP*, pre-selection technique is commonly used to speed up the convergence of GP by increasing the number of evaluations (i.e. cheap evaluation) of extra individuals which is conducted in an intermediate population [30], [32]. Then, only the individuals that achieve good fitness which are estimated by the surrogate model, are re-evaluated to get the real fitness. The other way is to use the surrogate model to evaluate the fitness of individuals directly [33]. Both ways are sensitive to the accuracy of surrogate models. The surrogate models greatly affect which individuals can be used to generate offspring for the next generation. It will further affect the quality of individuals in the population. A better way to apply the surrogate technique in JSS is worth studying further.

Algorithm 2: Framework of the Proposed Algorithm

Input : k multi-fidelity surrogate models S_1, S_2, ... , S_k
Output: The best evolved heuristics with each surrogate model ind_1^*, ind_2^*, ... , ind_k^*

1: **Initialisation**: Randomly initialise the population with k subpopulations
2: set ind_1^*, ind_2^*, ... , $ind_k^* \leftarrow null$
3: set $fitness(ind_1^*)$, $fitness(ind_2^*)$, ... , $fitness(ind_k^*) \leftarrow +\infty$
4: $gen \leftarrow 0$, $|Subpops| \leftarrow$ k
5: **while** $gen < maxGen$ **do**
6: // **Evaluation**: Evaluate the individuals in the population
7: **for** *i = 1 to* $|Subpops|$ **do**
8: **for** *j = 1 to* $|popsize|$ **do**
9: Run a DFJSS simulation with ind_j to get the schedule $Schedule_j$
10: $fitness(ind_j) \leftarrow Obj(Schedule_j)$
11: **end**
12: **for** *j = 1 to* $|popsize|$ **do**
13: **if** $fitness(ind_j) < fitness(ind_i^*)$ **then**
14: $ind_i^* \leftarrow ind_j$
15: $fitness(ind_i^*) \leftarrow fitness(ind_j)$
16: **end**
17: **end**
18: **end**
19: **Evolution**: Generate offspring for each subpopulation by genetic operators with the proposed *knowledge transfer* mechanism
20: $gen \leftarrow gen + 1$
21: **end**
22: **return** ind_1^*, ind_2^*, ... , ind_k^*

IV. The Proposed Algorithm

This paper develops a collaborative multi-fidelity based surrogate-assisted GP to solve the DFJSS problem. This section describes the framework of the proposed algorithm first. Then, the key components of the proposed algorithm are given.

A. The Framework of the Proposed Algorithm

The key idea of this paper is to solve the desired problem by utilising the advantages of surrogate models with different fidelities. The problem with lower fidelity surrogates is computationally cheaper but less accurate. On the contrary, the problem with higher fidelity surrogates is computationally more expensive but more accurate. This paper uses the degree of simplification of the problem to indicate the fidelity of the surrogate model. The smaller the scale of the problem, the less fidelity of the surrogate.

The main framework of the proposed algorithm is presented in Algorithm 2. The input is k designed surrogate models with different fidelities. The output is a set of best evolved rules obtained from subpopulations with different surrogate models. There are three main differences compared with the traditional GP for JSS, which is shown in Algorithm 1. *At the initialisation stage*, the population is formed with multiple subpopulations to incorporate multi-fidelity surrogate models into GP (line 1). *During the evaluation process*, the individuals in different subpopulations are evaluated with different surrogate models, respectively (from line 6 to line 17). *During the evolution stage*, the offspring of each subpopulation are generated for the next generation according to the proposed knowledge transfer mechanism.

According to the key idea of the proposed algorithm, there are three research questions in this paper. *The first* is how to

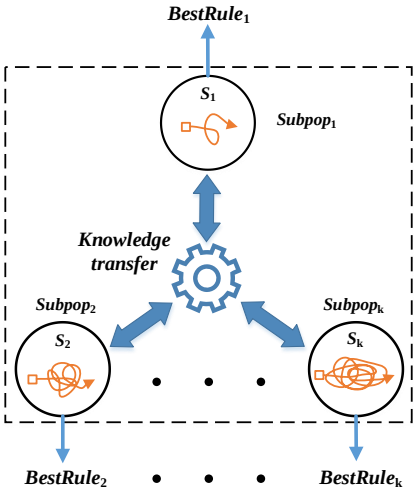


Fig. 2. The evolutionary framework of the proposed algorithm.

design the evolutionary framework to make it possible to collaborate between surrogate models with different fidelity. *The second* is how to define the representation of GP individuals to evolve routing and sequencing rules simultaneously with multi-fidelity surrogate models. *The last* is how to transfer knowledge between different subpopulations that incorporating with different surrogate models. The details of these three research questions with the corresponding principles and design points are presented in the next subsections.

B. The Evolutionary Framework

Introducing multi-fidelity surrogate models implies that some individuals are evaluated with the simpler surrogate, and some individuals are evaluated with more complex surrogate. This paper groups individuals to different surrogate models by dividing the entire population into several subpopulations. The individuals in the same or different subpopulation are evaluated with the surrogate with the same or different fidelity. From the perspective of the evolutionary process, the subpopulations can be considered as several independent evolutionary processes that are evolved simultaneously.

Fig. 2 shows the evolutionary framework of the proposed algorithm. Assuming k surrogate models (S_1, S_2, ... , S_k) with different fidelities from simple to complex are used to improve the efficiency of GP to solve the problem with S_k which is the desired problem to be solved in a collaborative way. The population of GP is divided into k subpopulations (e.g. $Subpop_1$, $Subpop_2$, ... , $Subpop_k$) and each subpopulation will evolve scheduling heuristics based on the corresponding surrogate model. In addition, during the evolutionary process, different subpopulations tend to assist with each other by sharing their knowledge with others. It is noted that all the populations are evolved in parallel. Therefore, the output of a GP run consists of k best evolved rules (e.g. $BestRule_1$, $BestRule_1$, ... , $BestRule_k$). However, we only focus on the best evolved rule $BestRule_k$ obtained from $Subpop_k$ with s_k since it is the problem we aim to solve.

There are some advantages of using the proposed evolutionary framework to realise collaborative multi-fidelity

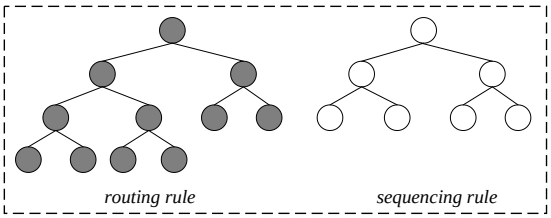


Fig. 3. An example of an individual with multi-tree representation of GP for DFJSS.

based surrogate-assisted GP for DFJSS. First, the evolutionary framework facilitates the collaboration between the surrogate models with different fidelities since they are in the same population. In addition, the proposed algorithm is not sensitive to the accuracy of one of the surrogate model since multiple surrogate models with different fidelities are involved at each generation. Finally, as a by-product, problems with different scales are solved simultaneously rather than using a single run to handle a problem at a time. It is an efficient way to make use of computational resources, if one prefers to solve multiple problems with different scales simultaneously.

C. Representation

According to the characteristics of DFJSS, a routing rule and a sequencing rule are needed to make two decisions simultaneously. To date, there have been three main frameworks to handle DFJSS. Tay et al. [3] proposed to use GP to only evolve sequencing rule by fixing the routing rule as a manually designed rule for FJSS. It is a straightforward way to solve the FJSS problems with scheduling heuristics. Yska et al. [6] introduced a cooperative coevolution framework with GP for the first time to evolve routing and sequencing rules simultaneously by applying two subpopulations. The proposed method showed its superiority due to the coevolution mechanism. Zhang et al. [7] introduced GP with multi-tree representation (MTGP) for evolving two rules within an individual in one population. The method MTGP is promising in terms of the effectiveness, efficiency, and the sizes of evolved rules. Genetic expression programming with multi-chromosome was also introduced to evolve two rules simultaneously [52]. This paper conducts on the MTGP framework to evolve routing and sequencing rules simultaneously. It is noted that this paper does not choose the cooperative coevolution framework in [6] or genetic expression programming with multi-chromosome in [52] to evolve two rules simultaneously since it can make the design of the algorithm a bit too complicated — there are already multiple subpopulations for incorporating multi-fidelity surrogate models in the proposed algorithm.

Fig. 3 shows an example of an individual with multi-tree representation of GP for DFJSS. Each individual consists of two trees. One is designed for evolving routing rule, and the other is for evolving sequencing rule. The routing rule and sequencing rule work together to make a schedule for a specific job shop scenario. The fitness of an individual depends on the performance of the schedule made by its routing and sequencing rule.

D. Knowledge Transfer

The key idea of this paper is to introduce collaborative mechanism to utilise the information of surrogate models with multi-fidelity. "How" and "when" to transfer knowledge, and "what" to transfer are important research questions in this section [53], [54], [55]. In the field of transfer learning in GP, based on "what to transfer" [56], there are two main schemas. The first schema is called *FullTree*, which tends to migrate a number of promising individuals at the last generation of the source problem to the target problem. The second is named as *SubTree*, which is implemented by randomly choosing a subtree in each individual at the last generation in the source problem. The selected subtrees are transferred at the first generation as individuals for the target problem.

Different from the traditional transfer learning in GP, this paper does not involve the source problem and the target problem. It implies that there is no knowledge extraction process from the source problem in this paper. Therefore, transferring full trees between the problems with multi-fidelity surrogates is more likely to have negative transfer since individuals are not well evolved without the source problem, especially at the beginning of the evolutionary process of GP. On the contrary, transferring subtrees can not only share knowledge between different problems but also preserve the knowledge for the current problem. Accordingly, this paper introduces knowledge transfer based on subtrees rather than full trees. In addition, to ensure the effectiveness of knowledge transfer between the problems with different fidelity models, the transfer in this paper is based on crossover rather than treating the subtrees as individuals for other problems.

How and when to transfer. Crossover is an important genetic operator in GP to produce offspring, which can be an effective carrier for knowledge transfer. Crossover can occur between individuals from the same subpopulation or different subpopulations. The proposed knowledge transfer mechanism is shown in Algorithm 3. We define a transfer ratio tr to control when to transfer knowledge from other subpopulations in each generation. A larger (smaller) tr value indicates the knowledge transfer between different subpopulations is (not) encouraged. If the knowledge transfer mechanism is triggered, the first parent $parent_1$ will be selected from the current subpopulation (line 8). The other parent $parent_2$ will be selected from one of the other subpopulations (line 9). Only the offspring derived from $parent_1$ is kept in the current subpopulation in the new generation (line 10 to line 13). If the knowledge mechanism is not triggered, two parents will be selected from the current subpopulation (i.e. the same subpopulation) to produce two offspring for generating the new subpopulation (from line 15 to line 23).

It is noted that the knowledge can not only be transferred from simpler to more complex subpopulation with a higher fidelity surrogate model but also from more complex to a simpler subpopulation with a lower fidelity surrogate model. From a knowledge transfer perspective, the subpopulation with a lower fidelity (simpler problem) can find promising individuals faster than the subpopulation with a higher fidelity (more complex problem). For a subpopulation with high fidelity surrogate models, introducing knowledge from a subpopulation with lower fidelity surrogate models can speed up its convergence. For a subpopulation with lower fidelity surrogate models, learning knowledge from a subpopulation with a higher fidelity surrogate model can help to increase the quality of individuals since the evolved rules with higher fidelity surrogate models are more reliable. In general, the collaboration with knowledge transfer is supposed to benefit all of the involved problems.

Algorithm 3: Generating Offspring with Knowledge Transfer

Input : A population with k subpopulations $P\{Subpop_1, Subpop_2, ..., Subpop_k\}$
Output: A new population with k subpopulations $P^{'}\{Subpop_1^{'}, Subpop_2^{'}, ..., Subpop_k^{'}\}$

1: set P, $P^{'} \leftarrow null$
2: $gen \leftarrow 0$, $|Subpops| \leftarrow$ k
3: **while** $gen < maxGen$ **do**
4: // **Evaluation**: Evaluate the individuals in each subpopulation, respectively
5: // **Evolution**
6: **for** *i = 1 to* $|Subpops|$ **do**
7: **if** $rand <= tr$ **then**
8: $parent_1 \leftarrow$ Select the first parent from $Subpops_i$
9: $parent_2 \leftarrow$ Select the second parent from $Subpops_{\neg i}$
10: $point_1$: the crossover point of $parent_1$
11: $point_2$: the crossover point of $parent_2$
12: $offspring$: replace $point_1$ of $parent_1$ by $point_2$
13: $Subpop_i^{'} \leftarrow Subpop_i^{'} \cup offspring$
14: **else**
15: $parent_1 \leftarrow$ Select the first parent from $Subpops_i$
16: $parent_2 \leftarrow$ Select the second parent from $Subpops_i$
17: $point_1$: the crossover point of $parent_1$
18: $point_2$: the crossover point of $parent_2$
19: $offspring_1$: replace $point_1$ of $parent_1$ by $point_2$
20: $offspring_2$: replace $point_2$ of $parent_2$ by $point_1$
21: $Subpop_i^{'} \leftarrow Subpop_i^{'} \cup offspring_1 \cup offspring_2$
22: **end**
23: $P^{'} \leftarrow P^{'} \cup Subpop_i^{'}$
24: **end**
25: $gen \leftarrow gen + 1$
26: **end**
27: **return** $P^{'}\{Subpop_1^{'}, Subpop_2^{'}, ..., Subpop_k^{'}\}$

What to transfer. It is critical to decide what kinds of knowledge are useful to be transferred. Intuitively, the knowledge carried by the promising individuals is beneficial. In this paper, we use the knee point technique [57] to decide the set of promising individuals. The individuals with smaller fitness values (i.e. our problem is minimising problem) than the fitness of the knee point individual are selected as promising individuals. Only the knowledge carried by the promising individuals is allowed to be transferred to other subpopulations.

The pseudo-code of selecting promising individuals for knowledge transfer is shown in Algorithm 4. Firstly, the individuals in the population are sorted in ascending order based on their fitness values (line 1). Secondly, one line (L) is generated between the two points with maximal and minimal fitness. Then, the distance between each individual and L is calculated (from line 6 to line 12), and the knee point which has the highest distance to L is detected. Finally, the individuals whose fitness values are smaller than the fitness of knee point are chosen as the *promising individuals* for knowledge transfer (line 13 to line 17).

Algorithm 4: Pseudo-code of selecting promising individuals for knowledge transfer

Input : The current subpopulation with a set of individuals Ind
Output: Promising individuals Ind^* for knowledge transfer

```
sort(population)
minPoint(0, fitness(ind_0)
maxPoint(popsize -1, fitness(ind_{popsize-1}))
set maxDistance ← 0 and kneePointIdx ← 0
get a line L based on minPoint and maxPoint points
for i = 0 to popsize − 1 do
    calculate the distance (d) from Point(i, fitness(ind_i)) to line L
    if d > maxDistance then
        maxDistance ← d
        kneePointIdx ← i
    end
end
for i = 0 to popsize − 1 do
    if i <= kneePointIdx then
        Ind* ← Ind[i]
    end
end
return Ind*
```

It is noted that the number of promising individuals varies in different generations, which can capture promising individuals efficiently. In addition, the number of selected promising individuals is not necessary to define in advance (i.e. *parameter-free*) because Algorithm 4 can decide it adaptively.

E. Summary

Overall, involving lower fidelity surrogate models with simplified DFJSS will reduce the computational cost of GP. However, the evaluations of individuals with a less fidelity surrogate model are often inaccurate. The proposed algorithm deftly solve this conflicting issue by collaborating the surrogate models with different fidelities. An effective knowledge transfer strategy realises the collaboration mechanism.

V. Experiment design

To investigate the *efficiency* (the training time) and *effectiveness* (the test objective value) of the proposed algorithm in DFJSS, a set of experiments have been conducted. This section describes the simulation model, the comparison design, and the parameter setting for GP.

A. Simulation Model

Simulation is a common method to investigate real-world complex problems [27]. In this paper, we aim to solve a range of problem instances/simulations with a sufficiently large number of random jobs. In the DFJSS simulation, there are 10 machines. We aim to estimate the steady-state performance of the scheduling heuristics by considering two factors. First, the simulation should be long enough so that the performance becomes stable and not be changed by more job arrivals. Second, the initial stage of the simulation should be ignored, since the machines start with empty workload, and are less busy at the beginning than the steady-state situation. To address the first issue, we record 5000 completed jobs in the simulation, as it is a sufficiently large number. To address the second issue, we ignore the data of the first 1000 "warm-up" completed jobs. We collect the data from the next 5000 jobs, and the simulation stops when the 6000th job is finished. This setting strategy has been commonly used in previous studies, e.g. [30], [58], [59].

New jobs will arrive over time following a Poisson process with rate λ. Each job consists of a different number of operations that are randomly generated by a uniform discrete distribution between 1 and 10. The number of candidate machines for an operation follows a uniform discrete distribution between 1 and 10. The importance of jobs varies, which is indicated by weights. The weights of 20%, 60%, and 20% of jobs are set as one, two, and four, respectively. This is because usually in practice 20% of jobs are less important jobs, 60% of jobs are important median jobs, and 20% of jobs are significant jobs [30]. The processing time of each operation is assigned by a uniform discrete distribution with the range [1, 99].

To verify the robustness of the proposed algorithm, scenarios with different settings (i.e. different objectives and utilisation levels) are examined. It is noted that the *utilisation level* is an essential factor to simulate different scenario environments. It is the proportion of time that a machine is busy. A larger utilisation level leads to a busier job shop.

B. Comparison Design

The goal of this paper is to improve the efficiency of GP with collaborative multi-fidelity based surrogate models for DFJSS. Two algorithms are involved in this paper. The GP with multi-tree representation [7] (MTGP) algorithm is selected as the baseline algorithm because it can evolve two rules simultaneously, and its framework is suitable for applying collaborative multi-fidelity based surrogate models. The proposed algorithm with collaborative multi-fidelity based surrogate models, is named as $\mathrm{M^3GP}$ since it involves *multi-population framework, multi-tree representation, and multi-fidelity surrogate models*. It is noted that MTGP works with one population with 1024 individuals while $\mathrm{M^3GP}$ operates with two subpopulations with 1024 individuals (i.e. 512 individuals for each subpopulation). $\mathrm{M^3GP_1}$ and $\mathrm{M^3GP_2}$ can be considered as the algorithms to measure the performance of the evolved rules with the lower surrogate model S_1 and the original model S_2 with the problem to be solved.

The performance of the proposed algorithm is first measured by the comparison between MTGP and $\mathrm{M^3GP_2}$ since we mainly focus on solving the desired problem. The state-of-the-art algorithms in [30] and [32], which is named as $\mathrm{SGP\_K}$ and $\mathrm{SGP\_H}$ for convenience in this paper, are further compared with the proposed algorithm with a fixed training time. In addition, the proposed algorithm with more than two surrogates with different fidelities is also studied. In order to verify the effectiveness of the proposed knowledge transfer mechanism for GP, the performance of $\mathrm{M^3GP_1}$ without the knowledge transfer and with the knowledge transfer is investigated. Similarly, the effectiveness of knowledge transfer on $\mathrm{M^3GP_2}$ is also further analysed.

In this paper, we focus on using the objective function and the utilisation level to construct multiple problems because the

TABLE I
THE TERMINAL SET.

Notation	Description
NIQ	The number of operations in the queue
WIQ	Current work in the queue
MWT	Waiting time of a machine
PT	Processing time of an operation on a specified machine
NPT	Median processing time for the next operation
OWT	The waiting time of an operation
WKR	Median amount of work remaining for a job
NOR	The number of operations remaining for a job
W	Weight of a job
TIS	Time in system

TABLE II
THE PARAMETER SETTING OF GP.

Parameter	Value
Number of subpopulations	2
Subpopulation size	512
Method for initialising population	ramped-half-and-half
Initial minimum/maximum depth	2 / 6
maximal depth of programs	8
Crossover/Mutation/Reproduction rate	80% / 15% / 5%
Parent selection	Tournament selection with size 7
Terminal/non-terminal selection rate	10% / 90%
The number of generations	51
Transfer Ratio tr	0.6
*The number of jobs in surrogate model S_1	2500
*The number of jobs in original model S_2	5000

* for $\mathrm{M^3GP}$ only

performance of evolved rules is influenced significantly by these two factors. Max-flowtime, mean-flowtime and mean-weighted-flowtime are three commonly used objectives. To verify the effectiveness of the proposed algorithm, we expect to test the proposed algorithm on complex scenarios. We set utilisation level as 0.85 and 0.95, since they can lead to complex job shop scenarios which can reflect the performance of the proposed algorithm well. These two utilisation levels have been commonly used in traditional studies on scheduling rules [18], [30], [32]. The proposed algorithms are tested on *six different scenarios* with these three objectives (e.g. max flowtime, mean flowtime, and mean weighted flowtime) and the two utilisation levels (e.g. 0.85 and 0.95). For the sake of convenience, Fmax, Fmean, and WFmean are used to indicate max flowtime, mean flowtime, and mean weighted flowtime, respectively. In this study, all experiments are run on an Arch Linux OS with an Intel (R) Core (TM) i7-4770 CPU at 3.40GHz, with 8-GB RAM. The algorithms are encoded with the java programming language.

C. Parameter Setting

All the parameter values are set to the commonly used values. In addition, all the same parameters of the compared algorithms are set to the same to get a fair comparison. In our experiment, the terminal set of GP is shown in Table I. The features indicate the characteristics related to machines (e.g. NIQ, WIQ, and MWT), operations (e.g. PT, NPT, and OWT), and jobs (e.g. WKR, NOR, W, and TIS). The function set is $\{+, -, *, /, max, min\}$, following the setting in [60]. The arithmetic operators take two arguments. The "$/$" operator is protected division, returning one if divided by zero. The max and min functions take two arguments and return the maximum and minimum of their arguments, respectively.

The other parameter settings of GP are shown in Table II. For simplicity, only two models with different fidelities are mainly considered in this paper. Therefore, the population of GP consists of two subpopulations and the number of individuals are set to 512 for each subpopulation. Borrowing the idea in [32], the designed surrogate model (S_1) is generated by creating a "half shop" job shop with 2500 (i.e. 5000 * 0.5 = 2500) jobs, which reduces the number of jobs to half of the original model but without reducing the number of machines to keep the characteristics of DFJSS. The other model (S_2) is designed by the job shop scenario with 5000 jobs, which is the original model (i.e. can be considered as a surrogate model with 100% accuracy). The only difference of S_1 and S_2 is the number of jobs, and the original model S_2 is more accurate than the surrogate model S_1, since S_2 reflects the problem itself. We set the transfer ratio tr to 0.6 since it is the best setting according to our preliminary work which will be introduced in subsection VII-B.

In addition, the sizes of intermediate population of $\mathrm{SGP\_H}$, $\mathrm{SGP\_K}$ are set as two times of the population, as suggested in [30]. The half shop surrogate model based on problem approximation is set the same as in [32] but with the maximum number of operations for a job as five for applying the idea properly in the investigated problem in this paper. In addition, the number of neighbours for KNN in $\mathrm{SGP\_K}$ is set to 1.

VI. RESULTS AND DISCUSSIONS

Wilcoxon rank-sum test with a significance level of 0.05 is used to verify the performance of the algorithms with 30 independent runs. In the following results, "–", "+", and "≈" indicate the corresponding result is significantly better than, worse than or similar to its counterpart. It is worth mentioning that this paper works on minimisation problems. The smaller the value, the better the corresponding criterion is.

A. Training Time

The training time is an important criterion to measure the efficiency of the algorithms. Less training time means that the algorithm can get good solutions or models efficiently.

Table III shows the mean and standard deviation of the training time of MTGP and $\mathrm{M^3GP}$ based on 30 independent runs in six DFJSS scenarios. The training time of the proposed algorithm $\mathrm{M^3GP}$ is significantly shorter than that of MTGP in all examined scenarios. For example, the training time of $\mathrm{M^3GP}$ is reduced the most by 23.40% in scenario <Fmean, 0.95>. In general, $\mathrm{M^3GP}$ is more efficient than its counterpart, and the training time of $\mathrm{M^3GP}$ is roughly 78.5% of that of MTGP taking all scenarios into account.

Fig. 4 shows the curve of training time of MTGP and $\mathrm{M^3GP}$ during the training process in six different scenarios. It shows that the training time of $\mathrm{M^3GP}$ is shorter than its counterpart at all generations in all scenarios. It indicates that the proposed

TABLE III
THE MEAN (STANDARD DEVIATION) OF THE **TRAINING TIME** (IN MINUTES) OF MTGP AND M^3GP ACCORDING TO 30 INDEPENDENT RUNS IN SIX DFJSS SCENARIOS.

Scenario	MTGP	M^3GP
<Fmax, 0.85>	64(9)	51(9)(–)
<Fmax, 0.95>	67(12)	53(9)(–)
<Fmean, 0.85>	61(11)	48(8)(–)
<Fmean, 0.95>	64(13)	49(6)(–)
<WFmean, 0.85>	62(13)	49(7)(–)
<WFmean, 0.95>	63(11)	49(6)(–)

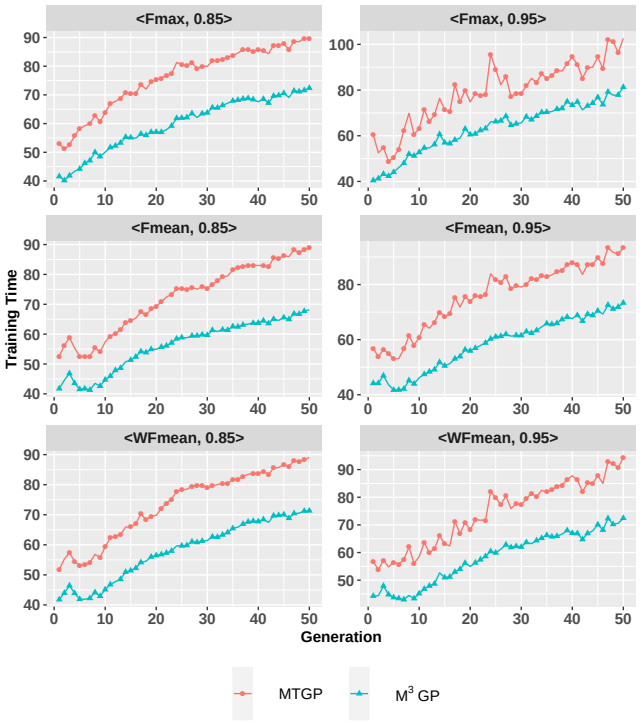


Fig. 4. The curve of *training time* (in minutes) of MTGP and M^3GP during the training process over 30 independent runs in six DFJSS scenarios.

algorithm can successfully save more computational cost in the whole evolutionary process. In addition, the training time of M^3GP is increasing more slowly than that of MTGP. As the number of generation increases, the training time of M^3GP grows from 40 to 70 roughly in all scenarios. However, the training time of MTGP increases from about 50 to 90 in the scenarios with utilisation level as 0.85 (e.g. <Fmax, 0.85>, <Fmean, 0.85> and <WFmean, 0.85>) while rises from 55 to 95 roughly in the scenarios with utilisation level as 0.95 (e.g. <Fmax, 0.95>, <Fmean, 0.95> and <WFmean, 0.95>).

More training time is normally needed in the scenarios with higher utilisation levels since the corresponding job shop environments are more complicated as it shows with MTGP in different scenarios. Specifically, compared with the training time needed in scenarios with utilisation levels as 0.85, for MTGP, more training time is needed in the scenarios with utilisation levels as 0.95. However, this is not the case for M^3GP. This indicates that the training time of M^3GP is not sensitive to the utilisation level of the job shop. One possible reason is that the surrogate models with lower fidelities weaken the relationship between utilisation level and training time since the corresponding problem is simpler. The simpler problem with even a higher utilisation level does not have a significant impact on the training time.

TABLE IV
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF MTGP AND M^3GP_2 **WITH THE SAME NUMBER OF GENERATIONS** OVER 30 INDEPENDENT RUNS IN SIX DFJSS SCENARIOS.

Scenario	MTGP	M^3GP_2
<Fmax, 0.85>	1235.73(41.27)	1232.39(31.70)(≈)
<Fmax, 0.95>	1967.24(65.18)	1932.22(42.22)**(–)**
<Fmean, 0.85>	384.55(1.04)	385.98(2.98) (≈)
<Fmean, 0.95>	555.32(9.91)	552.50(5.07)(≈)
<WFmean, 0.85>	831.30(7.32)	831.24(5.22)(≈)
<WFmean, 0.95>	1114.93(13.80)	1114.36(8.88)(≈)

B. The Performance of Evolved Rules

The goal of this paper is to improve the efficiency of GP to evolve effective scheduling heuristics for DFJSS without scarifying its performance. Table IV shows the mean and standard deviation of the objective values on unseen instances of MTGP and M^3GP_2 with the same number of generations over 30 independent runs in six different scenarios. It shows that there is no statistical difference in the performance between MTGP and M^3GP_2 in five out of the six scenarios. In addition, M^3GP_2 performs significantly better than its counterpart in scenario <Fmax, 0.95>. This shows that M^3GP_2 can achieve similar or even better performance than MTGP with a less computational cost.

It is also interesting to know whether the performance of M^3GP_2 can be better than MTGP if the same computational time is given. To answer this question, we set a fixed training time for all the algorithms. If the training time of the algorithm exceeds the given computational time budget, we stop the running of the algorithm. In this case, the number of generations in each run for one algorithm can be different, and we can not compare with the algorithms based on generations any more. To make a fair comparison, we process the results by choosing the compared data properly before comparison. We aim to choose data at the same or similar time points from each run of the compared algorithms. In addition, the number of chosen data in each run for all the compared algorithms is expected to be equal. We use $time$ to indicate the training time budget, and the results are equally divided into k groups. The average period time in each group is $\frac{time}{k}$, and the demarcation points of training time of k groups are $\frac{time}{k} * 1$, $\frac{time}{k} * 2$, ..., $\frac{time}{k} * k$. According to the demarcation points, the closest recorded time is identified to map the compared data. Note that there are inevitable errors to measure the performance of the algorithm in this way since the compared data is not obtained with exactly the same number of evaluations. Fortunately, these data are still representative to measure the performance of the algorithms since the sampling time is similar for different runs of one algorithm and different algorithms.

We limit the training time to 80 minutes for MTGP and M^3GP since the training time of the baseline MTGP with

TABLE V
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF MTGP, SGP_H, SGP_K, AND M^3GP_2 **WITH THE SAME TRAINING TIME** ACCORDING TO 30 INDEPENDENT RUNS IN SIX DFJSS SCENARIOS.

Scenario	MTGP	SGP_H	SGP_K	M^3GP_2
<Fmax, 0.85>	1225.58(44.21)	1267.49(40.76)(+)	1239.48(40.73)(≈)	1212.25(28.60)(–)(–)(–)
<Fmax, 0.95>	1963.85(61.53)	1981.81(52.60)(≈)	1956.72(29.60)(≈)	1925.87(28.98)(–)(–)(–)
<Fmean, 0.85>	384.20(0.93)	387.24(4.22)(+)	386.74(3.32)(+)	384.61(1.25)(≈)(–)(–)
<Fmean, 0.95>	554.62(9.79)	554.75(6.71)(≈)	554.07(8.18)(≈)	550.56(3.32)(–)(–)(–)
<WFmean, 0.85>	830.36(7.09)	834.79(8.13)(+)	830.40(5.52)(≈)	829.25(3.37)(–)(–)(≈)
<WFmean, 0.95>	1112.40(11.34)	1115.75(13.46)(≈)	1110.21(11.15)(≈)	1109.14(5.47)(≈)(–)(≈)

51 generations is about 80 minutes. We set the number of groups k to 20 that can get enough data for investigating the objective values along with training time of MTGP, SGP_H, SGP_K, and M^3GP_2. The objective values around 80 minutes are used to measure the performance of the involved algorithms. Table V shows the mean and standard deviation of the objective values on unseen instances of MTGP, SGP_H, SGP_K, and M^3GP_2 over 30 independent runs in six scenarios. First, SGP_H, SGP_K, and M^3GP_2 are compared with MTGP, respectively. Second, M^3GP_2 is compared with SGP_H and SGP_K, respectively. With the same training time, compared with MTGP, SGP_H and SGP_K perform significantly worse in three and two scenarios, respectively. However, M^3GP_2 can achieve significantly better performance than MTGP in four out of six scenarios (e.g. <Fmax, 0.85>, <Fmax, 0.95>, <Fmean, 0.95> and <WFmean, 0.85>). In addition, M^3GP_2 is not worse than MTGP in other scenarios. This verifies the effectiveness of the proposed algorithm.

Fig. 5 shows the curve of average objective values according to 30 independent runs on unseen instances of MTGP, SGP_H, SGP_K, and M^3GP_2 with the same training time in six DFJSS scenarios. With the same training time, the proposed algorithm M^3GP_2 can converge faster than its counterparts in all the scenarios. The performance of M^3GP_2 becomes better than its counterparts after about 10 minutes in scenario <Fmean, 0.85> and <Fmean, 0.95>. In addition, M^3GP_2 performs better than its counterparts after 20 minutes roughly in scenario <Fmax, 0.95> and <WFmean, 0.85>. Note that SGP_H performs worse than SGP_K, which is contrary to the conclusion in [32]. However, this is consistent with our expectations since the evaluations with half shop surrogate in [32] is more time consuming than the KNN surrogate in [30].

Overall, the proposed algorithm can improve the efficiency of GP in DFJSS by reducing the computational cost without losing its performance, and thus speeding up its convergence. Given the same training time, the proposed algorithm can achieve significantly better performance than its counterparts in most scenarios while no worse in all the scenarios.

C. The Effectiveness of Knowledge Transfer Mechanism

Multiple surrogate models with different fidelities are used to improve the efficiency of GP in a collaborative way to evolve scheduling heuristics for DFJSS. In order to examine the effectiveness of the proposed knowledge transfer mechanism, several experiments are conducted in this subsection.

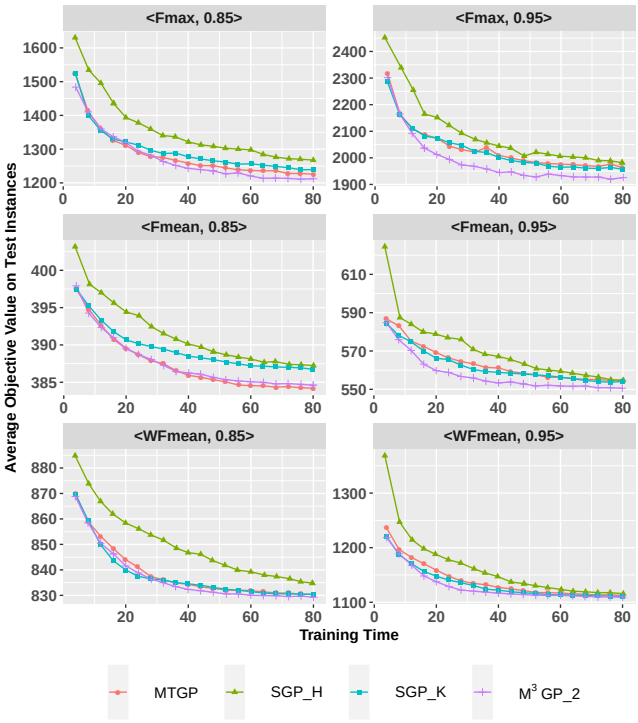


Fig. 5. The curve of average objective values according to 30 independent runs on test instances of MTGP, SGP_H, SGP_K, and M^3GP_2 *with the same training time* (in minutes) in six different scenarios.

We set the transfer ratio tr to zero in M^3GP to pervert the knowledge transfer between different subpopulations. $M^3GP_1(\text{without})$ and $M^3GP_2(\text{without})$ indicate that there is no knowledge transfer between subpopulations while $M^3GP_1(\text{with})$ and $M^3GP_2(\text{with})$ indicate that there is knowledge transfer between subpopulations with a tr of 0.6.

Table VI shows the mean and standard deviation of the objective values of M^3GP without and with knowledge transfer on test instances according to 30 independent runs in the six DFJSS scenarios. With knowledge transfer, both the performance of the evolved rules with different fidelity based surrogate models are significantly better than its counterpart without knowledge transfer in all examined scenarios. To be specific, the performance of $M^3GP_1(\text{with})$ is better than $M^3GP_1(\text{without})$ while the performance of $M^3GP_2(\text{with})$ is better than $M^3GP_2(\text{without})$ in all the scenarios. This indicates that the knowledge transfer can benefit both involved problems with different complexities. The knowledge obtained with the simpler surrogate model is useful for more complex

TABLE VI
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES OF M^3GP_1 AND M^3GP_2 **WITH AND WITHOUT KNOWLEDGE TRANSFER** WITH THE SAME NUMBER OF GENERATIONS ON TEST INSTANCES ACCORDING TO 30 INDEPENDENT RUNS IN SIX DFJSS SCENARIOS.

Scenario	M^3GP_1(without)	M^3GP_1(with)	M^3GP_2(without)	M^3GP_2(with)
<Fmax, 0.85>	1201.44(60.42)	1170.31(29.48)(–)	1261.85(65.40)	1232.39(31.70)(–)
<Fmax, 0.95>	1815.72(76.82)	1758.22(34.88)(–)	2001.56(80.43)	1932.22(42.22)(–)
<Fmean, 0.85>	390.24(4.21)	388.14(2.98)(–)	387.98(4.10)	385.98(2.98)(–)
<Fmean, 0.95>	566.98(7.81)	561.28(5.30)(–)	558.10(7.38)	552.50(5.07)(–)
<WFmean, 0.85>	840.19(9.50)	835.37(5.21)(–)	836.31(9.29)	831.24(5.22)(–)
<WFmean, 0.95>	1149.64(25.43)	1135.75(9.23)(–)	1130.06(25.95)	1114.36(8.88)(–)

TABLE VII
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF MTGP AND M^3GP_2(without) ACCORDING TO 30 INDEPENDENT RUNS IN SIX DFJSS SCENARIOS.

Scenario	MTGP	M^3GP_2(without)
<Fmax, 0.85>	1235.73(41.27)	1261.85(65.40)(+)
<Fmax, 0.95>	1967.24(65.18)	2001.56(80.43)(+)
<Fmean, 0.85>	384.55(1.04)	387.98(4.10)(+)
<Fmean, 0.95>	555.32(9.91)	558.10(7.38)(+)
<WFmean, 0.85>	831.30(7.32)	836.31(9.29)(+)
<WFmean, 0.95>	1114.93(13.80)	1130.06(25.95)(+)

surrogate model. The knowledge derived from complex surrogate model is beneficial to the simpler surrogate model. This confirms the effectiveness of the proposed knowledge transfer mechanism.

Table VII shows the mean and standard deviation of the objective values on unseen instances of MTGP and M^3GP_2 without knowledge transfer according to 30 independent runs in the six DFJSS scenarios. The results show that the performance of M^3GP_2 without knowledge transfer is significantly worse than that of MTGP in all scenarios. It verifies the effectiveness of the proposed knowledge transfer mechanism. In addition, it is in line with our expectation since more computational resources are used for solving the desired problem in MTGP. To be specific, without knowledge transfer, the number of individuals for solving the desired problem in M^3GP_2(without) is 512, which is only half of the number of individuals in MTGP.

Fig. 6 shows the curve of average objective values on unseen instances of M^3GP_1(without) and M^3GP_1(with) based on 30 independent runs in the six different DFJSS scenarios. It is obvious that the performance of M^3GP_1(with) is better than that of M^3GP_1(without) after generation 5 roughly in the max-flowtime related scenarios (i.e. <Fmax, 0.85> and <Fmax, 0.95>) and after about 10 generations in mean-flowtime and weighted mean-flowtime related scenarios (i.e. <Fmean, 0.85>, <Fmean, 0.95>, <WFmean, 0.85> and <WFmean, 0.95>). It may be because the individuals in the population before generation 5 or generation 10 have not reached good quality yet, and the transferred knowledge does not have sufficient contribution to the other subpopulation. Fortunately, the transferred knowledge before generation 5 or 10 does not have a negative effect on the other subpopulation. The same trend is also found between M^3GP_2(without) and M^3GP_2(with), which is shown in Fig. 7.

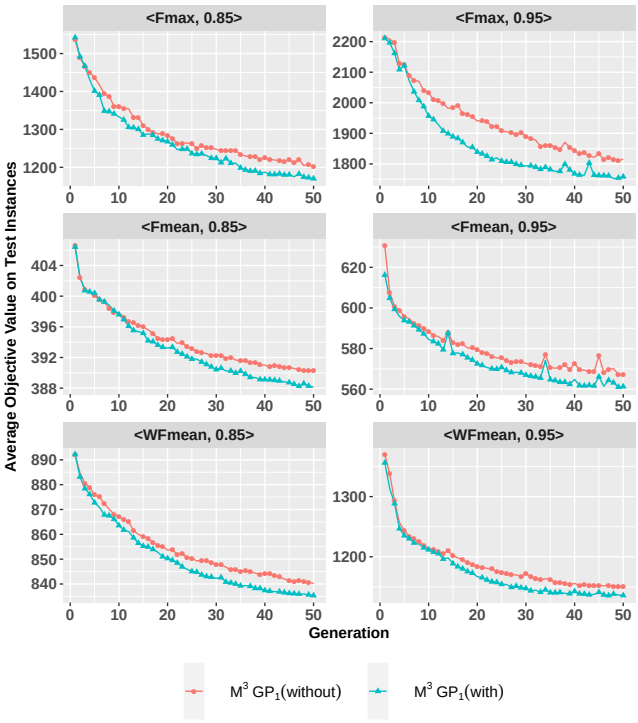


Fig. 6. The curve of average objective values on test instances of M^3GP_1(without) and M^3GP_1(with) according to 30 independent runs in six different DFJSS scenarios.

VII. FURTHER ANALYSIS

To deeply understand the effect of the proposed algorithm, the proposed algorithm with more surrogate models with different fidelities and the sensitivity analysis of knowledge transfer ratio, are further analysed in this section.

A. Collaboration with More Surrogate Models

We have conducted an in-depth analysis of the proposed algorithm. It is interesting to investigate whether the collaboration between more models with different fidelities can benefit problem-solving or not.

To ensure the performance of the algorithm for the problem to be solved, half individuals in the population are kept for optimising the desired problem. Other individuals are divided equally for solving other simplified problems with different surrogate models. The number of jobs between surrogate models with different fidelities follows an arithmetic sequence with an upper bound as 5000. M^3GP_2, M^3GP_3, M^3GP_4, and

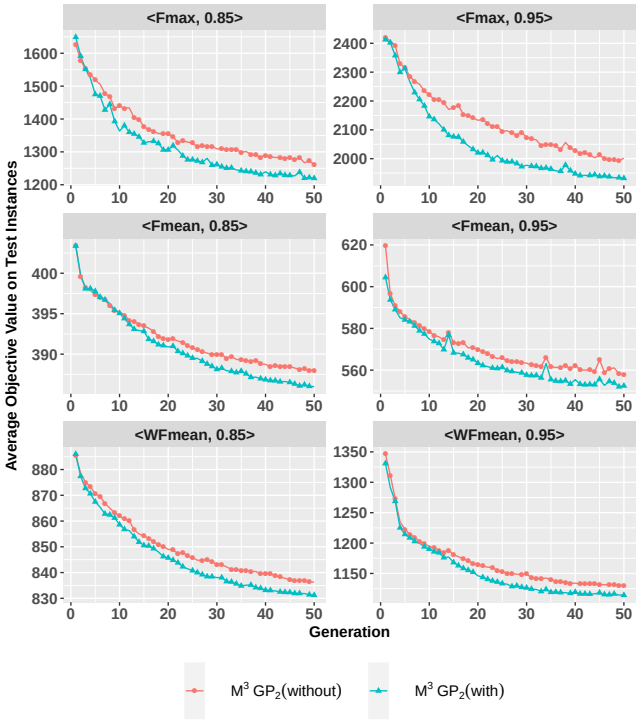


Fig. 7. The curve of average objective values on test instances of $\mathrm{M^3GP_2}$(without) and $\mathrm{M^3GP_2}$(with) according to 30 independent runs in six different DFJSS scenarios.

TABLE VIII
THE SETTINGS OF THE NUMBER OF INDIVIDUALS/JOBS OF THE PROPOSED ALGORITHM WITH ONE, TWO, THREE, AND FOUR SURROGATES.

Algorithm	$Subpop1$	$Subpop2$	$Subpop3$	$Subpop4$	$Subpop5$
$\mathrm{M^3GP_2}$	512/2500	512/5000	–	–	–
$\mathrm{M^3GP_3}$	256/1667	256/3333	512/5000	–	–
$\mathrm{M^3GP_4}$	170/1250	171/2500	171/3750	512/5000	–
$\mathrm{M^3GP_5}$	128/1000	128/2000	128/3000	128/4000	512/5000

$\mathrm{M^3GP_5}$ can be considered as the corresponding algorithms on the desired problem, respectively. Table VIII shows the settings of the number of individuals and jobs of the proposed algorithm with one, two, three, and four surrogates.

Table IX shows the mean and standard deviation of the training time of the involved algorithms with the same number of generations according to 30 independent runs in six different scenarios. Compared with $\mathrm{M^3GP_2}$, as the number of surrogate models increases, the training time of $\mathrm{M^3GP_3}$, $\mathrm{M^3GP_4}$, and $\mathrm{M^3GP_5}$ has no significant difference in most scenarios.

Table X shows the mean and standard deviation of the objective values of $\mathrm{M^3GP_2}$, $\mathrm{M^3GP_3}$, $\mathrm{M^3GP_4}$, and $\mathrm{M^3GP_5}$ on unseen instances with the same number of generations according to 30 independent runs in six different scenarios. In terms of the objective values on unseen data, there is no significant difference among the compared algorithms in most scenarios. In one of the scenarios of $\mathrm{M^3GP_3}$, $\mathrm{M^3GP_4}$, and $\mathrm{M^3GP_5}$, the performance is significantly worse than that of $\mathrm{M^3GP_2}$. In terms of the mean and standard deviation, the performance of $\mathrm{M^3GP_2}$ is better than other compared algorithms in half of the scenarios (e.g. <Fmax, 0.95>, <Fmean, 0.95>, and <WFmean, 0.95>) as shown in bold. In addition, the examined problems are not very sensitive to the number of surrogates as the performance of the proposed algorithm with different number of surrogates achieve similar performance.

TABLE IX
THE MEAN (STANDARD DEVIATION) OF THE TRAINING TIME (IN MINUTES) OF THE INVOLVED ALGORITHMS **WITH THE SAME NUMBER OF GENERATIONS** BASED ON 30 INDEPENDENT RUNS IN SIX SCENARIOS.

Scenario	$\mathrm{M^3GP_2}$	$\mathrm{M^3GP_3}$	$\mathrm{M^3GP_4}$	$\mathrm{M^3GP_5}$
<Fmax, 0.85>	51(9)	48(9)(≈)	46(8)(≈)	48(8)(≈)
<Fmax, 0.95>	53(9)	48(7)**(–)**	47(7)**(–)**	49(7)**(–)**
<Fmean, 0.85>	48(8)	43(4)**(–)**	47(8)(≈)	47(7)(≈)
<Fmean, 0.95>	49(6)	47(7)(≈)	48(6)(≈)	46(7)(≈)
<WFmean, 0.85>	49(7)	48(7)(≈)	48(6)(≈)	47(8)(≈)
<WFmean, 0.95>	49(6)	47(9)(≈)	47(8)(≈)	47(7)(≈)

Overall, the results show that the proposed algorithm with two surrogates achieves the best performance with the settings in this paper. A possible reason is that the additional surrogate models in the experiments were not accurate enough, and thus introduced more noise than the first two surrogate models. The accuracy of surrogates can be different for different problem complexities, such as utilisation levels. In our case, we observe that two surrogate models is a proper choice. There are several interesting but challenging questions are worth studying in the future. First, how to decide the optimal number of surrogates. Second, how to design efficient surrogate models according to the domain knowledge or information from the evolutionary process. Last but not least, how to design effective knowledge transfer mechanisms for a large number of surrogates since the interaction between more surrogates is even complex. However, this is out of the scope of this study, and we would like to investigate it in the future.

B. The Sensitivity Analysis of Knowledge Transfer Ratio

The transfer ratio which decides the frequency to transfer knowledge between different problems at each generation is further analysed in this subsection.

Fig. 8 shows the curve of average objective values on unseen instances of $\mathrm{M^3GP_2}$ with different transfer ratios according to 30 independent runs in six different scenarios. The performance of $\mathrm{M^3GP_2}$ with different transfer ratios is almost the same in scenario <Fmax, 0.95>, <WFmean, 0.85>, and <WFmean, 0.95>. In scenario <Fmax, 0.85>, <Fmean, 0.85>, and <Fmean, 0.95>, there are some slight differences between the algorithms with different transfer ratios. From an overall perspective, $\mathrm{M^3GP_2}$ with transfer ratio as 0.6 has slightly better performance than its counterparts. Therefore, this paper sets the transfer ratio to 0.6 to $\mathrm{M^3GP}$ to compare with other algorithms, as we mentioned earlier. However, in general, the performance of $\mathrm{M^3GP_2}$ is not sensitive to the transfer ratio.

VIII. CONCLUSIONS

The goal of this paper was to develop an effective strategy that collaborates multi-fidelity based surrogate models to

TABLE X
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF M^3GP_2, M^3GP_3, M^3GP_4, AND M^3GP_5 **WITH THE SAME NUMBER OF GENERATIONS** ACCORDING TO 30 INDEPENDENT RUNS IN SIX DFJSS SCENARIOS.

Scenario	M^3GP_2	M^3GP_3	M^3GP_4	M^3GP_5
<Fmax, 0.85>	1232.39(31.70)	1228.03(37.98)(≈)	**1222.44(29.73)**(≈)	1230.54(33.47)(≈)
<Fmax, 0.95>	**1932.22(42.22)**	1948.53(45.67)(≈)	1960.67(120.22)(≈)	1954.50(68.28)(≈)
<Fmean, 0.85>	385.98(2.98)	385.33(2.06)(≈)	**384.93(1.73)**(≈)	385.59(3.11)(≈)
<Fmean, 0.95>	**552.50(5.07)**	553.43(5.30)(≈)	554.02(4.84)(≈)	555.82(6.76)**(+)**
<WFmean, 0.85>	831.24(5.22)	830.83(6.50)(≈)	831.36(6.56)(≈)	**830.79(4.27)**(≈)
<WFmean, 0.95>	**1114.36(8.88)**	1119.21(15.79)**(+)**	1122.82(17.39)**(+)**	1117.59(12.98)(≈)

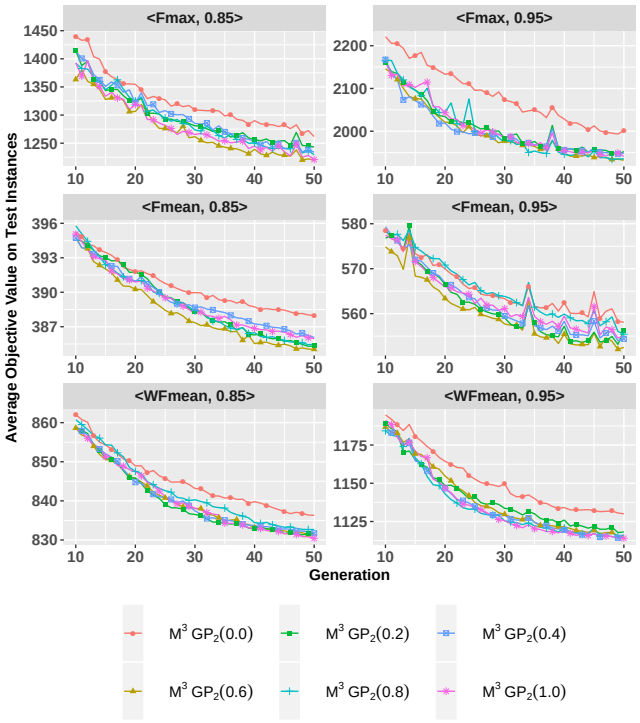


Fig. 8. The curve of average objective values on test instances of M^3GP_2 with different transfer ratios over 30 independent runs in six DFJSS scenarios.

improve the efficiency of GP to evolve scheduling heuristics automatically for DFJSS. The goal was successfully achieved by proposing an effective collaboration framework in GP that made multiple surrogates can learn from each other, and an effective knowledge transfer mechanism.

The results show that the proposed algorithm M^3GP_2 can dramatically reduce the computational time of GP without losing its performance. Within the same training time, M^3GP_2 can achieve significantly better performance in most of the scenarios, while no worse than its counterpart in all the scenarios. The efficiency and effectiveness of the proposed algorithm are verified by comparing the training time, the performance with both the same number of generations and training time, and the analysis of knowledge transfer mechanism. In general, the proposed algorithm M^3GP_2 can successfully improve the efficiency of GP, and achieve effective scheduling heuristics automatically for DFJSS. The proposed algorithm shows its superiority compared with the state-of-the-art algorithms related to surrogate for the job shop scheduling problems.

Some interesting directions can be further investigated in the near future. We plan to apply the proposed algorithm to the same problem but with more different objectives and experimental factors such as various processing time distributions or other problems such as vehicle routing and timetabling. It is still not clear what is the optimal number of the surrogate models with multi-fidelity for one specific problem, and how to effectively define different multi-fidelity surrogate models. In addition, what kinds of knowledge are transferred, and how the problems help each other. We also would like to investigate the effects of different representations on the proposed algorithm. Last but not least, we plan to further improve our simulation based on the investigation of the industry practice to make it closer to the real-world applications.

REFERENCES

[1] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[2] C. D. Geiger, R. Uzsoy, and H. Aytuğ, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Scheduling*, vol. 9, no. 1, pp. 7–34, 2006.

[3] J. C. Tay and N. B. Ho, "Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems," *Computers & Industrial Engineering*, vol. 54, no. 3, pp. 453–473, 2008.

[4] S. Bennett, S. Nguyen, and M. Zhang, "A hybrid discrete particle swarm optimisation method for grid computation scheduling," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2014, pp. 483–490.

[5] P. Brucker and R. Schlie, "Job-shop scheduling with multi-purpose machines," *Computing*, vol. 45, no. 4, pp. 369–375, 1990.

[6] D. Yska, Y. Mei, and M. Zhang, "Genetic programming hyper-heuristic with cooperative coevolution for dynamic flexible job shop scheduling," in *European Conference on Genetic Programming*. Springer, 2018, pp. 306–321.

[7] F. Zhang, Y. Mei, and M. Zhang, "Genetic programming with multi-tree representation for dynamic flexible job shop scheduling," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 472–484.

[8] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Guided subtree selection for genetic operators in genetic programming for dynamic flexible job shop scheduling," in *European Conference on Genetic Programming*. Springer, 2020, pp. 262–278.

[9] J. Xiong, L.-n. Xing, and Y.-w. Chen, "Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns," *International Journal of Production Economics*, vol. 141, no. 1, pp. 112–126, 2013.

[10] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "Investigating a machine breakdown genetic programming approach for dynamic job shop scheduling," in *Proceedings of the European Conference on Genetic Programming*. Springer, 2018, pp. 253–270.

[11] Y. N. Sotskov and N. V. Shakhlevich, "Np-hardness of shop-scheduling problems with three jobs," *Discrete Applied Mathematics*, vol. 59, no. 3, pp. 237–266, 1995.

[12] H. Chen, C. Chu, and J.-M. Proth, "An improvement of the lagrangean relaxation approach for job shop scheduling: a dynamic programming method," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 5, pp. 786–795, 1998.

[13] F. Y.-P. Simon *et al.*, "Integer linear programming neural networks for job-shop scheduling," in *Proceedings of the IEEE International Conference on Neural Networks*. IEEE, 1988, pp. 341–348.

[14] P. J. Van Laarhoven and E. H. Aarts, "Simulated annealing," in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.

[15] F. Glover and M. Laguna, "Tabu search," in *Handbook of combinatorial optimization*. Springer, 1998, pp. 2093–2229.

[16] G. V. Conroy, "Handbook of genetic algorithms," *The Knowledge Engineering Review*, vol. 6, no. 4, pp. 363–365, 1991.

[17] M. Durasevic and D. Jakobovic, "A survey of dispatching rules for the dynamic unrelated machines environment," *Expert Systems with Applications*, vol. 113, pp. 555–569, 2018.

[18] F. Zhang, Y. Mei, and M. Zhang, "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*. IEEE, 2019, pp. 1366–1373.

[19] X. Li and L. Gao, "Gep-based reactive scheduling policies for dynamic fjsp with job release dates," in *Proceedings of the Effective Methods for Integrated Process Planning and Scheduling*. Springer, 2020, pp. 405–428.

[20] M. Jayamohan and C. Rajendran, "New dispatching rules for shop scheduling: a step forward," *International Journal of Production Research*, vol. 38, no. 3, pp. 563–586, 2000.

[21] J. R. Koza, *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems*. Stanford University, Department of Computer Science Stanford, CA, 1990, vol. 34.

[22] K. Miyashita, "Job-shop scheduling with genetic programming," in *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2000, pp. 505–512.

[23] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Genetic programming for evolving due-date assignment models in job shop environments," *Evolutionary Computation*, vol. 22, no. 1, pp. 105–138, 2014.

[24] F. Zhang, Y. Mei, and M. Zhang, "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*. IEEE, 2019, pp. 347–355.

[25] Y. Zhou, J.-J. Yang, and L.-Y. Zheng, "Multi-agent based hyper-heuristics for multi-objective flexible job shop scheduling: A case study in an aero-engine blade manufacturing plant," *Ieee Access*, vol. 7, pp. 21 147–21 176, 2019.

[26] J. Lin, L. Zhu, and K. Gao, "A genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem," *Expert Systems with Applications*, vol. 140, p. 112915, 2020.

[27] J. P. Davis, K. M. Eisenhardt, and C. B. Bingham, "Developing theory through simulation methods," *Academy of Management Review*, vol. 32, no. 2, pp. 480–499, 2007.

[28] G. Lamé and M. Dixon-Woods, "Using clinical simulation to study how to improve quality and safety in healthcare," *BMJ Simulation and Technology Enhanced Learning*, 2018.

[29] L. E. Peterson, "K-nearest neighbor," *Scholarpedia*, vol. 4, no. 2, p. 1883, 2009.

[30] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evolutionary Computation*, vol. 23, no. 3, pp. 343–367, 2015.

[31] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Selection schemes in surrogate-assisted genetic programming for job shop scheduling," in *Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, 2014, pp. 656–667.

[32] S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2951–2965, 2017.

[33] F. Zhang, Y. Mei, and M. Zhang, "Surrogate-assisted genetic programming for dynamic flexible job shop scheduling," in *Proceedings of the Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 766–772.

[34] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2016.

[35] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, 2017.

[36] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, 2020. Doi: 10.1109/TCYB.2020.3024849.

[37] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational Intelligence*. Springer, 2009, pp. 177–201.

[38] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward, "A genetic programming hyper-heuristic approach for evolving 2-d strip packing heuristics," *IEEE Trans. Evolutionary Computation*, vol. 14, no. 6, pp. 942–958, 2010.

[39] M. R. Hyde, "A genetic programming hyper-heuristic approach to automated packing," Ph.D. dissertation, University of Nottingham, UK, 2010.

[40] M. B. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memetic Computing*, vol. 1, no. 3, pp. 205–219, 2009.

[41] N. Pillay and W. Banzhaf, "A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem," in *Proceedings of the Portuguese Conference on Aritficial Intelligence*, 2007, pp. 223–234.

[42] F. Zhang, Y. Mei, and M. Zhang, "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*. Springer, 2019, pp. 33–49.

[43] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 1–14, 2015.

[44] M. Durasevic and D. Jakobovic, "Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, pp. 9–51, 2018.

[45] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[46] X. Sun, D. Gong, Y. Jin, and S. Chen, "A new surrogate-assisted interactive genetic algorithm with weighted semisupervised learning," *IEEE Transactions on Cybernetics*, vol. 43, no. 2, pp. 685–698, 2013.

[47] Y. Sun, H. Wang, B. Xue, Y. Jin, G. G. Yen, and M. Zhang, "Surrogate-assisted evolutionary deep learning using an end-to-end random forest-based performance predictor," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 350–364, 2019.

[48] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural computation*, vol. 3, no. 2, pp. 246–257, 1991.

[49] T. Chugh, Y. Jin, K. Miettinen, J. Hakanen, and K. Sindhya, "A surrogate-assisted reference vector guided evolutionary algorithm for computationally expensive many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 1, pp. 129–142, 2018.

[50] Q. Chen, M. Zhang, and B. Xue, "Structural risk minimization-driven genetic programming for enhancing generalization in symbolic regression," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 4, pp. 703–717, 2019.

[51] K. Chen, B. Xue, M. Zhang, and F. Zhou, "Novel chaotic grouping particle swarm optimization with a dynamic regrouping strategy for solving numerical optimization tasks," *Knowledge-Based Systems*, p. 105568, 2020.

[52] Y. Zhou, J.-j. Yang, and Z. Huang, "Automatic design of scheduling policies for dynamic flexible job shop scheduling via surrogate-assisted cooperative co-evolution genetic programming," *International Journal of Production Research*, vol. 58, no. 9, pp. 2561–2580, 2020.

[53] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

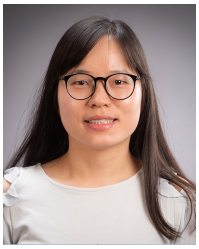
[54] M. Iqbal, B. Xue, H. Al-Sahaf, and M. Zhang, "Cross-domain reuse of extracted knowledge in genetic programming for image classification," *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 4, pp. 569–587, 2017.

[55] K. Chen, B. Xue, M. Zhang, and F. Zhou, "An evolutionary multitasking-based feature selection method for high-dimensional classification," *IEEE Transactions on Cybernetics*, 2020. Doi: 10.1109/TCYB.2020.3042243.

[56] T. T. H. Dinh, T. H. Chu, and Q. U. Nguyen, "Transfer learning in genetic programming," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2015, pp. 1145–1151.

[57] X. Zhang, Y. Tian, and Y. Jin, "A knee point-driven evolutionary algorithm for many-objective optimization," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 6, pp. 761–776, 2015.

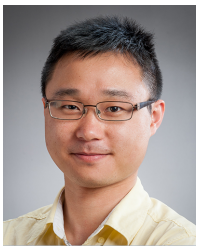
[58] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *Proceedings of the Annual Conference on Genetic and Evolutionary Computation*. ACM, 2010, pp. 257–264.
[59] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, "An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 5, pp. 339–353, 2017.
[60] Y. Mei, M. Zhang, and S. Nguyen, "Feature selection in evolving job shop dispatching rules with genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*. IEEE, 2016, pp. 365–372.



Fangfang Zhang (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees from Shenzhen University, Shenzhen, China, in 2014 and 2017, respectively. She is currently pursuing the Ph.D. degree in computer science with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand.

She has over 20 journal and conference papers. Her current research interests include evolutionary computation, hyper-heuristic, job shop scheduling, and multitask optimization.

Ms. Zhang is a member of the IEEE Computational Intelligence Society and Association for Computing Machinery, and has been severing as reviewers for top international journals such as the IEEE Transactions on Evolutionary Computation and the IEEE Transactions on Cybernetics, and conferences including the Genetic and Evolutionary Computation Conference and the IEEE Congress on Evolutionary Computation. She is also a committee member of the IEEE NZ Central Section.



Yi Mei (M'09-SM'18) received the B.Sc. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively.

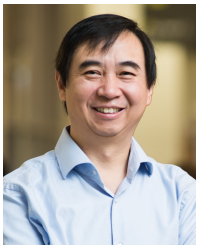
He is currently a Senior Lecturer with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He has more than 100 fully referred publications, including the top journals in EC and Operations Research, such as IEEE Transactions on Evolutionary Computation, IEEE Transactions on Cybernetics, Evolutionary Computation, European Journal of Operational Research, and ACM Transactions on Mathematical Software. His research interests include evolutionary scheduling and combinatorial optimization, machine learning, genetic programming, and hyperheuristics.

Dr. Mei serves as a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee and a member of the Intelligent Systems Applications Technical Committee. He is an Editorial Board Member/Associate Editor of three international journals, and a Guest Editor of a special issue of the Genetic Programming and Evolvable Machines journal. He serves as a reviewer of over 30 international journals.

Su Nguyen (M'13) received his Ph.D. degree in Artificial Intelligence and Data Analytics from Victoria University of Wellington, New Zealand, in 2013.

He is a Senior Research Fellow and an Algorithm Lead with CDAC, La Trobe University, Melbourne, VIC, Australia. His expertise includes evolutionary computation (EC), simulation optimization, automated algorithm design, interfaces of AI/OR, and their applications in logistics, energy, and transportation. He has more than 70 publications in top EC/OR peer-reviewed journals and conferences. His current research focuses on novel people-centric artificial intelligence to solve dynamic and uncertain planning tasks by combining the creativity of evolutionary computation and power of advanced machine-learning algorithms.

Dr. Nguyen was the Chair of IEEE Task Force on Evolutionary Scheduling and Combinatorial Optimisation from 2014 to 2018 and is a member of the IEEE CIS Data Mining and Big Data Technical Committee. He delivered technical tutorials about EC and AI-based visualization at Parallel Problem Solving from Nature Conference in 2018 and IEEE World Congress on Computational Intelligence in 2020. He served as an Editorial Member of Complex and Intelligence Systems and the Guest Editor of the special issue on Automated Design and Adaption of Heuristics for Scheduling and Combinatorial Optimization in Genetic Programming and Evolvable Machines journal.



Mengjie Zhang (M'04-SM'10-F'19) received the B.E. and M.E. degrees from Artificial Intelligence Re- search Centre, Agricultural University of Hebei, Baoding, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively.

He is currently a Professor of Computer Science, the Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) with the Faculty of Engineering, Victoria University of Wellington, Wellington, New Zealand. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, feature selection and reduction, job shop scheduling, and transfer learning. She has published over 500 research papers in refereed international journals and conferences.

Prof. Zhang was the Chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, the IEEE CIS Emergent Technologies Technical Committee, and the Evolutionary Computation Technical Committee, and a member of the IEEE CIS Award Committee. He is a Vice-Chair of the Task Force on Evolutionary Computer Vision and Image Processing and the Founding Chair of the IEEE Computational Intelligence Chapter in New Zealand. She is also a Committee Member of the IEEE NZ Central Section. He is a Fellow of the Royal Society of New Zealand and an IEEE Distinguished Lecturer.