Surrogate-Assisted Evolutionary Multitasking Genetic Programming for Dynamic Flexible Job Shop Scheduling

Fangfang Zhang, *Student Member, IEEE,* Yi Mei, *Senior Member, IEEE,* Su Nguyen, *Member, IEEE,* Mengjie Zhang, *Fellow, IEEE,* and Kay Chen Tan, *Fellow, IEEE*

***Abstract*—Dynamic flexible job shop scheduling is an important combinatorial optimisation problem with complex routing and sequencing decisions under dynamic environments. Genetic programming, as a hyper-heuristic approach, has been successfully applied to evolve scheduling heuristics for job shop scheduling. However, its training process is time-consuming, and it faces the retraining problem once the characteristics of job shop scenarios vary. It is known that multitasking is a promising paradigm for solving multiple tasks simultaneously by sharing knowledge among the tasks. To improve the training efficiency and effectiveness, this paper proposes a novel surrogate-assisted evolutionary multitasking via genetic programming to share useful knowledge between different scheduling tasks. Specifically, we employ the phenotypic characterisation for measuring the behaviours of scheduling rules and building a surrogate for each task accordingly. The built surrogates are used not only to improve the efficiency of solving each single task but also for knowledge transfer in multitasking with a large number of promising individuals. The results show that the proposed algorithm can significantly improve the quality of scheduling heuristics for all scenarios. In addition, the proposed algorithm manages to solve multiple tasks collaboratively in terms of the evolved scheduling heuristics for different tasks in a multitasking scenario.**

***Index Terms*—Surrogate, Multitasking, Genetic Programming, Hyper-heuristics, Dynamic Flexible Job Shop Scheduling.**

I. Introduction

Job shop scheduling (JSS) [1] has been widely studied in both academia and industry such as manufacturing processes [2] and grid computing [3] due to its practical applications. In a job shop, a number of jobs (each with a sequence of operations) need to be processed by a set of machines (an operation can only be processed on a specific machine). The JSS is an NP-hard problem [4], and flexible JSS (FJSS) [5] is a general extension of JSS in which each operation can be processed by a set of candidate machines rather than a specific machine. There are two decisions that need to be made simultaneously in FJSS. 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) [6] is more challenging than JSS and FJSS, since it tends to optimise the machine resources under a dynamic environment with unpredicted events such as new jobs arriving at any time [7] and breakdown of machines [8]. In DFJSS, not only the machine assignment and operation sequencing decisions need to be made simultaneously, but also dynamic events are considered when making schedules.

Exact optimisation methods such as dynamic programming [9] and integer linear programming [10] are usually not applicable to large scale problems. *Approximate solution optimisation methods* such as simulated annealing [11], tabu search [12] and genetic algorithms [13], 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. *Heuristics* are strategies derived from previous experiences with similar problems [14], which are widely used in decision-making. *Scheduling heuristics* such as dispatching rules [15], [16] are perhaps the most popularly used heuristics for JSS. Typical scheduling heuristics make decisions by prioritising the operations or machines at the decision points. However, the common scheduling heuristics, such as SPT (shortest processing time) [17] and some composite rules [15], are usually *manually* designed by experts. The design highly relies on domain knowledge, especially for complex scenarios, which is often unavailable to the end users, and the designed heuristics are usually too specific to be reused in other scenarios.

Genetic programming (GP) [18], as a hyper-heuristic approach (GPHH), has been successfully applied to evolve scheduling heuristics *automatically* for JSS [7], [19], [20], [21]. The most advantageous feature of GP is its flexibility of representation. DFJSS is widely studied with simulation [22] which is a promising technique to examine complicated real-world applications such as healthcare [23] and manufacturing [24]. However, the main disadvantage of GPHH for DFJSS is its high demand for computing resources, mainly due to two reasons. One is that there are lots of priority calculations for candidates, e.g., machines and operations, with GP individuals for making decisions during the execution of the simulation. The other is that the scheduling heuristics evolved by GPHH are not always applicable to other scheduling scenarios, and new heuristics need to be retrained when facing new problems. This is because the characteristics of DFJSS such as the

The authors are with the Evolutionary Computation Research Group at the School of Engineering and Computer Science, Victoria University of Wellington, 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, Australia (e-mail: P.Nguyen4@latrobe.edu.au). Kay Chen Tan is with the Department of Computer Science, City University of Hong Kong (kaytan@cityu.edu.hk).

machine resources and order quantity may vary, which is common in practical applications [25].

Surrogate models [26], [27], [28] have been widely used to reduce the computational cost in evolutionary computation. The success lies in building computationally cheap models to approximate the fitness of individuals without requiring the original computationally expensive evaluations. In the past decades, surrogate-assisted GPHH has been successfully used in JSS. In terms of the surrogate model itself, existing studies are either based on k nearest neighbour (KNN) [29] or simplified simulation model [24], [30], [31]. In terms of applications, surrogates are used to replace the real evaluation directly [31] or assist pre-selection [24], [29], [30] that approximates the fitness of a large number of offspring and only revaluate the top individuals that have good estimated fitness with real fitness evaluations. All of these studies confirm the superiority of using surrogate techniques in JSS. However, these studies only focus on improving the efficiency and effectiveness of GPHH for a single JSS task.

The paradigm of evolutionary multitasking was given in [32], [33] for solving multiple tasks simultaneously. The success of evolutionary multitasking relies on the knowledge sharing mechanism between tasks during the evolutionary process. Evolutionary multitasking has been successfully applied to solve different problems such as continuous numeric optimisation [33], [34], [35], symbolic regression [36] and job shop scheduling [37], [38]. However, most of the existing approaches are only applied on continuous, numeric optimisation problems rather than discrete, combinatorial problems such as DFJSS. In real-world applications, each product has different demand at different times, which is a typical DFJSS problem. The complexities of job shops such as the frequencies of production orders for producing a specific product may vary [39]. For example, in the clothing industry, the orders for the down jacket in winter are usually higher than in summer. For the down jacket job shop, the frequencies of orders for producing down jacket vary between seasons. It is thus beneficial to have various kinds of scheduling heuristics for a company to handle different cases. Intuitively, giving multiple solutions simultaneously for a company is an effective way to improve the problem-solving capability.

Although surrogate and multitasking techniques can improve the efficiency and effectiveness of evolutionary optimisation in different aspects, they are often used independently. To the best of our knowledge, there are only a few studies on surrogate-assisted multitasking [40], [41], [42], [43]. In [40], Gaussian Process was introduced to build a surrogate model for the designed global search component of the algorithm. In [41], surrogates are applied to reduce the fitness evaluations for each task on benchmark problems. In [42], [43], computationally cheap models are applied to handle expensive optimisation problems by sharing its acquired knowledge. Although these studies have showed good performance for multitasking optimisation, the surrogate technique is only used to improve the effectiveness for a single task independently rather than multiple tasks simultaneously. In other words, these studies are not about using surrogate for the core mechanism of multitasking such as knowledge transfer between tasks.

Moreover, it is not easy to apply existing approaches [40], [41], [42], [43] in DFJSS with GPHH directly. Firstly, the way to allocate individuals for different tasks is computationally expensive for DFJSS. Secondly, the typical way for multitasking in [32], [33] that puts parent population and the newly generated offspring together and then allocates individuals to different tasks could be too greedy for GP to converge in evolution. Besides, existing approaches are mostly applied to benchmarks with continuous, numeric optimisation problems [36] rather than discrete, combinatorial optimisation problems, which are not applicable for DFJSS.

To address the above issues, we apply the surrogate technique to evolutionary multitasking with GP for DFJSS. The built surrogates are not only used for improving the effectiveness of solving a single task but also the effectiveness of sharing knowledge between tasks. The objective of this work is thus to develop an effective surrogate-assisted evolutionary multitasking approach to improving the effectiveness of GPHH for evolving scheduling heuristics automatically for multiple DFJSS tasks. The proposed algorithm is expected to both speed up the convergence and improve the quality of scheduling heuristics of GPHH for DFJSS. Specifically, the contributions of this paper are given as follows:

1) We developed a novel and effective evolutionary multitasking framework for GPHH to optimise multiple DFJSS tasks simultaneously.
2) This work built surrogates for different tasks accordingly by taking the behaviour of individuals and their fitness into account. The proposed surrogates can not only estimate the quality of individuals well but also speed up the convergence of the algorithm to obtain effective scheduling heuristics.
3) We proposed to transfer knowledge by allocating the newly generated individuals to appropriate tasks incorporated with the surrogate for multitasking. The results show that the proposed knowledge sharing mechanism can effectively transfer knowledge between tasks.
4) The proposed overall GPHH algorithm with surrogate-assisted multitasking outperforms the state-of-the-art algorithm, and can evolve highly-competitive scheduling heuristics for DFJSS for different tasks in all examined multitasking scenarios.

The rest of this paper is organised as follows. Section II introduces the background of this research work. Detailed descriptions of the proposed algorithm are given in Section III. The experiment designs are shown in Section IV. Simulation results and discussions are presented in Section V. Further analyses are conducted in Section VI. The conclusions are given in Section VII.

II. Background

This section provides a brief introduction of multitasking DFJSS, and how to use GPHH to evolve scheduling heuristics for solving DFJSS, which also includes the related work of GPHH for JSS. In addition, the related work of evolutionary multitasking, surrogate and knowledge transfer in GP is also briefly described here.

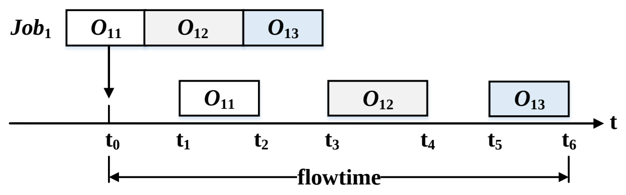

Fig. 1. An example of calculating the flowtime of a job (Job_1).

A. Multitasking Dynamic Flexible Job Shop Scheduling

JSS focuses on improving production efficiency in a job shop. In JSS, 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 an arrival time $at(J_j)$ and a sequence of operations $O_j = (O_{j1}, O_{j2}, ..., O_{ji})$. The completion of the last operation for a job means the job has been finished. An example of calculating the flowtime of Job_1 with three operations can be found in Fig. 1. In FJSS, 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. Routing decision and sequencing decision need to be made simultaneously in FJSS. In DFJSS, not only the two decisions need to be made simultaneously, but also dynamic events are necessary to be taken into account when constructing schedules. This paper focuses on one dynamic event, i.e., dynamically and continuously arriving new jobs. The information of a job 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 at a time.
- 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 objective of the scheduling is the optimised performance criterion for a problem while satisfying all the above constraints. In this paper, we consider three commonly used objective functions. The calculations of the objectives are shown as follows:

- Mean-flowtime $= \frac{\sum_{j=1}^{n} \{C_j - r_j\}}{n}$
- Mean-tardiness $= \frac{\sum_{j=1}^{n} Max\{0, C_j - d_j\}}{n}$
- Mean-weighted-tardiness $= \frac{\sum_{j=1}^{n} w_j * Max\{0, C_j - d_j\}}{n}$

where C_j is the completion time of job J_j, r_j is the release time of J_j, d_j is the due date of J_j, w_j is the weight of J_j, and n is the number of jobs that are expected to be processed.

This paper defines the tasks with different utilisation levels (i.e., indicate different complexities) as related tasks to build a multitasking scenario. A multitasking scenario consists of k tasks $P = \{t_1, t_2, ..., t_k\}$ where tasks t_i and t_j ($i \neq j$) share the same problem parameters but only different in the utilisation level. The solution of a multitasking problem is a set of scheduling heuristics $P_s = \{h_1, h_2, ..., h_k\}$, each for a task. Note that each scheduling heuristic is composed of a routing rule and a sequencing rule.

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

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

B. Genetic Programming Hyper-heuristics for DFJSS

The optimal structures of heuristics are normally not known in real-world applications, which makes the heuristic learning process challenging. GP, as a hyper-heuristic method [44], has been successfully applied to evolve scheduling heuristics for different types of JSS [45], [46], [47]. Tree-based GP [48], [49] is a good candidate to learn heuristic for DFJSS due to its flexible representation. This implies that the structures of heuristics do not need to be defined in advance.

The pseudo-code of GPHH to learn heuristics for DFJSS is shown in Algorithm 1. The input of the proposed algorithm is a task that is expected to be solved, and the output is the learned heuristic h^* with a routing heuristic r^* and sequencing heuristic s^*. As a population-based algorithm, GP starts with a randomly initialised population (line 1). It is noted that each GP individual contains two trees [6]. The first tree represents the routing rule and the second tree represents the sequencing rule. Both trees are numerical priority functions, which are used to prioritise the machines or operations at different decision situations. When a new operation becomes ready, the routing rule will be applied to prioritise its candidate machines, and the operation will be assigned to the machine with the best priority (e.g., has the least workload under the least-work-in-queue rule). When a machine becomes idle, the sequencing rule will be triggered to prioritise the operations in its queue, and the operation with the best priority (e.g., the one with the shortest processing time if the shortest-processing-time rule is used) will be chosen to be processed next. The qualities of heuristics are measured based on the objective functions (from line 6 to line 10). Specifically, a simulation is run with the heuristic h_i that is expected to be examined to get a schedule $Schedule_i$ (line 8). The quality of the heuristic h_i is assigned by calculating the objective value of its obtained schedule $Schedule_i$ (line 9). A new population is generated by recombining the heuristics (crossover), mutating the heuristics (mutation), or copying the heuristics with good fitness directly (reproduction) (from line 11 to line 13) to the next generation.

Static JSS, dynamic JSS and DFJSS are three main types of JSS. GP was firstly introduced in [19] to evolve sequencing rules for static JSS and achieved good results. Due to the efficiency of making online decisions with rules, GP has been widely used to generate sequencing rules for dynamic JSS [46], [50], [51] and DFJSS [7], [52], [53]. For DFJSS, routing rule and sequencing rule are respectively used for machine assignment and operation sequencing. In [52], sequencing rule is evolved by GP with a fixed routing rule for DFJSS. Routing rule and sequencing rule are evolved simultaneously in [6], [53], which have been proven to achieve good performance.

C. Related Work

1) Evolutionary Multitasking: Evolutionary multitasking [32], [33] has recently received much research interests in optimising multiple tasks simultaneously. It has been applied to different fields, such as continuous numeric optimisation [33], [34], [54], and regression problems [36]. However, most of the approaches are only applied to benchmarks with continuous, numeric optimisation problems rather than discrete, combinatorial problems such as DFJSS. This thus limits its applications in practice. In terms of solution representation, most studies are conducted with the vector-based search space such as genetic algorithms [33]. In contrast, our work is based on tree-based search space, and the search consists of heuristics rather than direct solutions to the problem.

There are a few studies that use multitasking with GP, such as in team orienteering [55], dynamic JSS [56], and DFJSS [37]. The problem studied in [55] is a static problem, and the individuals are grouped to different islands for each task. Training instances are clustered for each island with individuals. Several individuals that have a good fitness at each island are transferred between islands to share the learned knowledge. In this work, the training instances are not available, since the research on DFJSS is based on simulation. In [56], a niched approach was proposed for dynamic JSS. However, the main drawback is that the niched individuals need further evaluations, which is not an efficient way. In [37], multitasking was applied to DFJSS in a multiple subpopulation manner, and the efficiency of solving multiple DFJSS problems was dramatically improved. However, the quality of the evolved scheduling heuristics was not improved.

2) Surrogate: Surrogate has been used in JSS with GPHH [29], [30], [24], [31]. Existing related surrogate works in JSS with GPHH are grouped into two categories based on the way it builds the surrogate model to estimate the fitness of an individual. One is to use existing techniques such as KNN for estimating the fitness of an individual by finding the most similar individual in the pool (e.g., the individuals evaluated in the previous generation) [29]. The other is to use a simplified simulation model (e.g., with smaller number of jobs and machines) as the surrogate [30], [24], [31].

Surrogate-assisted multitasking was proposed in [40] for the memetic algorithm with benchmark problems. However, the surrogate with Gaussian Process [57] was only used to assist the search process in the designed component of global search. In [41], surrogate models are built based on historical search information for each task to reduce the number of fitness evaluations in multitasking problems. These works [40], [41] use the surrogate solely to improve the search efficiency for each task independently or for a single component in a multitasking problem rather than enhancing the core multitasking mechanism such as knowledge sharing. In addition, the mechanism of individual allocation for tasks based on the original evaluations is computationally expensive if applied to DFJSS, since reallocated individuals would need to be revaluated with the simulation in the DFJSS.

3) Knowledge Transfer in Genetic Programming: In the field of transfer learning in GP, according to "what to transfer", there are two main schemes [58]. One is the "*FullTree*" that migrates a number of individuals with good quality from the source domain to the target domain. The other is the "*SubTree*" that is extracted from individuals in the source domain and adapted to the target domain.

Different from the traditional transfer learning in GP, there are no source and target domains in this work. The knowledge is transferred between different tasks directly without the knowledge extraction process from the source domain. In [36], assortative mating and vertical cultural transmission [33] were introduced to transfer information between different tasks in GP, which can be seen as a "*FullTree*" transfer. In [37], the knowledge transfer was realised by the crossover operator between the GP individuals for different tasks, which belongs to the "*Subtree*" transfer scheme. Compared with transferring "*SubTree*", the advantage of transferring "*FullTree*" is that the knowledge extraction process is not necessary. The key to transferring "*FullTree*" is that the chosen individuals must be of good quality for the problem that is expected to be solved. Otherwise, the transferred individuals will be eliminated subsequently during the evolutionary process, and lose the role of knowledge transfer [58]. On the contrary, compared with transferring "*FullTree*", the advantage of transferring "*SubTree*" is that the transferred knowledge might be more precise. However, the knowledge extraction of "*SubTree*" is complex and time-consuming. In this paper, we aim to propose an effective knowledge transfer mechanism via the surrogate that can share knowledge between tasks by taking the advantages of transferring "*FullTree*" and "*SubTree*".

In summary, the research on multitasking with surrogate mechanisms, especially for DFJSS is still in its early stage. In this paper, we propose to use the surrogate not only to improve the effectiveness of solving a single task but also for knowledge transfer in a multitasking problem. It is a good case study to illustrate the effectiveness of surrogate-assisted evolutionary multitasking for solving dynamically discrete and combinatorial optimisation problems.

III. The proposed algorithm

The idea of the proposed algorithm is to improve the knowledge sharing mechanism for multitasking by incorporating surrogate techniques. This section describes the proposed surrogate-assisted multitasking based on GPHH for DFJSS.

Algorithm 2: Framework of the Proposed Algorithm

```
Input : Tasks $t_1$, $t_2$, ... , $t_k$
Output: The best evolved heuristics for each task $h_1^*$, $h_2^*$, ... , $h_k^*$
1:  Initialisation: Randomly initialise the population with $k$ subpopulations
2:  set $h_1^*$, $h_2^*$, ... , $h_k^* \leftarrow null$
3:  set $fitness(h_1^*)$, $fitness(h_2^*)$, ... , $fitness(h_k^*) \leftarrow +\infty$
4:  $gen \leftarrow 0$, $|subpops| \leftarrow k$
5:  while $gen < maxGen$ do
6:      set $S \leftarrow null$
7:      set $newPop \leftarrow null$
8:      // Evaluation: Evaluate the individuals in the population
9:      for $i$ = 1 to $|subpops|$ do
10:         for $j$ = 1 to $subpopsize$ do
11:             Run a DFJSS simulation with $h_j$ according to task $t_i$ to
                  get the schedule $Schedule_j$
12:             $fitness_{h_j} \leftarrow Obj(Schedule_j)$
13:         end
14:         for $j$ = 1 to $subpopsize$ do
15:             if $fitness_{h_j} < fitness_{h_i^*}$ then
16:                 $h_i^* \leftarrow h_j$
17:             end
18:         end
19:         Calculate the phynotypic characterisation vector for each
              individual in $Subpop_i$
20:         Build surrogate model $S_i$ with the phenotypic characterisations
              and the corresponding fitness of indivudlas in $Subpop_i$
21:         $S \leftarrow S \cup S_i$
22:     end
23:     if $gen < maxGen - 1$ then
24:         // Evolution: generate new population
25:         Generate $n * subpopsize$ offspring for each subpopulation by
              genetic operators, respectively.
26:         Offspring Pool: Put the offspring of all subpopulations together
27:         Clearing the individuals in the offspring pool
28:         // Assign individuals to tasks
29:         for $i$ = 1 to $|subpops|$ do
30:             Estimate the fitness of individuals in the offspring pool
                  using the surrogate model $S_i$ built for $Subpop_i$
31:             $newInds$: Choose the top $subpopsize$ individuals from
                  offspring pool based on the estimated fitness
32:             $newPop \leftarrow newPop \cup newInds$
33:         end
34:     end
35:     $gen \leftarrow gen + 1$
36: end
37: return $h_1^*$, $h_2^*$, ... , $h_k^*$
```

A. The Framework of the Surrogate-Assisted Multitasking

The framework of the proposed algorithm is presented in Algorithm 2. The inputs are the k tasks to be solved. The output is a set of best evolved scheduling heuristics obtained from subpopulations for different tasks.

At the initialisation stage, the population is formed with k subpopulations to solve k tasks (line 1). *During the evaluation process*, the individuals in different subpopulations are evaluated with different training instances according to the tasks. In addition, the surrogate models are built with the phenotypic characterisations of individuals and their real fitness for each subpopulation (from line 8 to line 22). Note that there are k surrogates generated at each generation, one for each task and the surrogates are rebuilt at the next generation. *During the evolution stage*, $n*subpopsize$ number of offspring are generated for each subpopulation to build an offspring pool (from line 24 to line 26). Then, the duplicated individuals are removed from the offspring pool according to their phenotypic characterisations (line 27). To obtain the final offspring $newInds$ for $subpop_i$ for $task_i$, the fitness of all the individuals in the offspring pool are estimated by the surrogate S_i (line 30). Then, the individuals in the offspring pool are ranked according to the fitness estimated by S_i, and the top $subpopsize$ individuals are selected as the final offspring for $subpop_i$ (from line 28 to line 33). With surrogates, more generated knowledge carried by individuals can be evaluated efficiently. In addition, the surrogates can help with knowledge transfer between tasks by allocating the newly generated individuals from different subpopulations for proper tasks.

B. Surrogate Model

The main idea in this paper is to use surrogate to assist multitasking. Surrogate design itself is not the focus in this paper, thus we simply choose a straightforward one that is designed based on the characteristics of GP and job shop scheduling. KNN with phenotypic characterisation has been successfully proposed in [29] for dynamic JSS with GP. Since KNN is efficient and straightforward, it is chosen as the surrogate to estimate the fitness of individuals by finding the most similar individual based on the phenotypic characterisations of individuals with Euclidean distance as suggested in [59]. The Euclidean distance is a reasonable technique to measure the similarity of behaviour of individuals, since the phenotypic characterisation is a vector of ranking number that indicates the decision made by a rule, and it is not related to the search space of the algorithm, either continuous or discrete. The individuals with real fitness in the previous generation are used to build the KNN surrogate, and the surrogate is updated at each generation. Note that the phenotypic characterisation consists of ordinal numbers, which makes it not necessarily applicable to build accurate surrogates with other surrogate techniques such as Gaussian process.

The phenotypic characterisation of an individual is a decision vector based on a set of decision situations [29]. In this paper, decision situations are sampled from preliminary simulation runs with 5000 jobs on 10 machines using the reference rules (e.g., WIQ, work in the queue for routing, and SPT, shortest processing time for sequencing). The preliminary simulation generated about 50,000 routing and 50,000 sequencing decision situations. Following the steps in [29], we randomly sampled decision situations from the generated decision situations that contains 2 and 20 jobs. To balance the accuracy and complexity of the phenotypic characterisation, the number of candidates, i.e., machines for routing and operations for sequencing, in each decision situation, is set to 7 in this paper. In other words, from all the generated decision situations, a subset of 20 routing situations and 20 sequencing situations with the length of 7 is sampled for measuring the phenotypic characteristic of an individual. A smaller distance between the phenotypic characterisations of two individuals suggests that the two individuals are similar.

The phenotypic characterisation of a rule is a vector of rank numbers, where the number of dimensions equals the number of decision situations. The element in the ith dimension indicates the rank of the most prior candidate, i.e., operation or machine, by the characterised rule in the rank list of the

TABLE I
AN EXAMPLE OF CALCULATING THE PHENOTYPIC CHARACTERISATION OF A ROUTING RULE WITH FOUR DECISION SITUATIONS AND EACH WITH THREE CANDIDATE MACHINES.

Decision Situation	Reference Rule	Characterised Rule	PC_i
1 (M_1)	3	2	
1 (M_2)	2	3	1
1 (M_3)	$\underline{1}$	$\underline{1}$	
2 (M_1)	1	3	
2 (M_2)	$\underline{3}$	$\underline{1}$	3
2 (M_3)	2	2	
3 (M_1)	2	3	
3 (M_2)	3	2	1
3 (M_3)	$\underline{1}$	$\underline{1}$	
4 (M_1)	1	3	
4 (M_2)	$\underline{2}$	$\underline{1}$	2
4 (M_3)	3	2	

PC_i indicates the ith dimension of phenotypic characterisation.

reference rule (i.e., WIQ for routing, and SPT for sequencing). Table I shows an example of calculating the phenotypic characterisation of a routing rule with four decision situations, and each decision situation consists of three candidate machines. In the first decision situation, M_3 is the most prior machine by the characterised routing rule. When looking at the rank value of M_3 by the reference rule, we find that the rank value is 1. Therefore, the value of the phenotypic characterisation in the first situation PC_1 is set to 1. Similarly, PC_i can be obtained in other decision situations. The corresponding observation indicators for finalising the phenotypic characterisation are underlined in each decision situation. Finally, the phenotypic characterisation of this routing rule is [1, 3, 1, 2]. Note that the way to calculate the phenotypic characterisation of sequencing rule is the same as the routing rule, except that sequencing rule is examined with sequencing decision situations rather than routing decision situations.

This paper extends the idea in [60], [61] to calculate the phenotypic characterisation for an individual in DFJSS by concatenating the phenotypic characterisations of routing and sequencing rules. An example of the phenotypic characterisation of an individual in DFJSS is shown in Fig. 2. The phenotypic characterisation of an individual consists of the decision vectors of routing and sequencing heuristics. The individuals with both similar routing (left part) and sequencing (right part) phenotypic characterisations are considered to have similar behaviour.

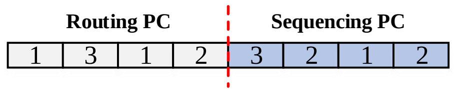

Fig. 2. An example of the phenotypic characterisation of an individual in DFJSS (PC indicates phenotypic characterisation).

Regarding the way of using the surrogate model, [29], [30], [24] show that pre-selection is an effective approach. Specifically, an intermediate population with a large number of offspring is generated at each generation, then the surrogate is used to estimate the fitness of all the offspring efficiently. Only individuals with good surrogate fitness are pre-selected to the next generation, and are revaluated with the real fitness evaluation. This paper uses the surrogate in a pre-selection manner at each generation but in a different way. Specifically, the surrogate is not only used to improve the effectiveness of solving a single task but also sharing knowledge between tasks. The details are shown in the next subsection.

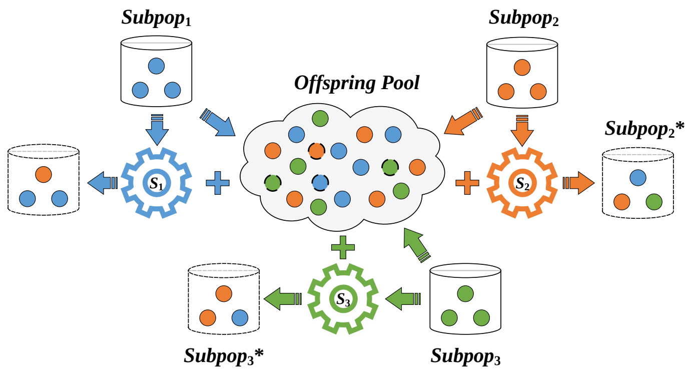

Fig. 3. An example of the proposed surrogate-assisted multitasking with three tasks in terms of the surrogate and knowledge transfer mechanism.

C. Knowledge Transfer with Surrogate for Multitasking

For simplicity, the proposed algorithm applies the approach of transferring "*FullTree*" between tasks. To find suitable individuals for knowledge transfer, we employ the KNN-based surrogate mechanism with phenotypic characterisation to help transfer knowledge between different tasks due to its effectiveness in distinguishing the behaviours of individuals.

Fig. 3 illustrates the knowledge transfer process in a multitasking scenario with three tasks via surrogate. Assuming there is one population with three subpopulations, and each subpopulation consists of three individuals, each of which is designed to solve a single task. The individuals in $Subpop_1$, $Subpop_2$, and $Subpop_3$ are marked in solid blue, orange, and green circles, respectively. If there is no knowledge transfer between them, the three subpopulations can be considered as three independent evolutionary processes. For multitasking, a main consideration is to design the mechanism for transferring knowledge between these three subpopulations. Firstly, the phenotypic characterisations and the fitness of individuals in the subpopulations are used to build the corresponding KNN surrogate model for each subpopulation (i.e., S_1, S_2, S_3), respectively. Secondly, to get more potentially useful knowledge, a large number of offspring $n * subpopsize$ (where n is two in this example) are generated based on each subpopulation independently and put into the offspring pool. The offspring pool is composed of the offspring, from $Subpop_1$, $Subpop_2$ and $Subpop_3$. Then, the individuals in the offspring pool are cleared by removing the duplicated individuals according to the phenotypic characterisations. The clearing procedure for offspring pool is shown in Algorithm 3. The removed individuals are marked in dotted circles. Finally, the individuals in the offspring pool are evaluated based on S_1 (S_2 or S_3 respectively), and the best $subpopsize$ individuals according to fitness are selected as the final offspring for $Subpop_1$ ($Subpop_2$ or $Subpop_3$ respectively). The newly

Algorithm 3: Pseudo-code of clearing procedure of offspring pool

```
Input  : All the individuals in the offspring pool $Ind$
Output: Cleared individuals with different behaviours $NonRepInd$
1:  set $PC \leftarrow null$
2:  $distance \leftarrow \infty$
3:  for $i$ = 1 to $|Ind|$ do
4:      $PC_i$: Calculate the phenotypic characteristic of $Ind_i$
5:      $PC \leftarrow PC \cup PC_i$
6:  end
7:  for $i$ = 1 to $|PC|$ do
8:      for $j$ = $i$ +1 to $|PC|$ do
9:          $distance$: Calculate the distance between $PC_i$ and $PC_j$
10:         if $distance$ == 0 then
11:             remove $Ind_j$ from $Ind$
12:             $distance \leftarrow \infty$
13:         end
14:     end
15: end
16: $NonRepInd \leftarrow Ind$
17: return $NonRepInd$
```

generated subpopulations are shown as $Subpop_1{}^*$, $Subpop_2{}^*$, and $Subpop_3{}^*$. Note that an offspring in the offspring pool can be allocated to multiple subpopulations, since an offspring is possible to work well on multiple tasks.

From the perspective of transferred materials, the knowledge is transferred by a *FullTree* via the surrogate-assisted multitasking mechanism. However, in fact, the knowledge is also shared in a *SubTree* manner via the crossover operator in each subpopulation, since the chosen parents in each subpopulation can come from different subpopulations. As a result, the proposed approach can take advantage of transferring knowledge by both the *FullTree* and *SubTree*.

D. Summary

The proposed algorithm combines the advantage of surrogate and multitasking in GPHH for DFJSS. It improves the effectiveness of not only solving a single task but also sharing knowledge between tasks. From the perspective of each task, the surrogate is used in a pre-selection way by estimating the fitness of a large number of individuals in the offspring pool. From the perspective of all tasks, the surrogate plays a role to share knowledge between different tasks by reallocating the generated individuals to different tasks. The surrogate technique makes it possible for multitasking to examine more useful materials carried by the individuals in the offspring pool efficiently, while multitasking with the surrogate can make a better decision of individual allocations for tasks.

IV. Experiment Design

A. Simulation Model

The simulation assumes that 5000 jobs need to be processed by 10 machines. Each job consists of a different number of operations that are randomly generated from 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. In addition, the importance of jobs varies, and the weights of 20%, 60%, and 20% of jobs are set as 1, 2, and 4, respectively. The processing time of each operation is assigned by a uniform discrete distribution with the range [1, 99]. The due date of a job is the sum of its release time and 1.5 times its total processing time. Note that the total processing time of a job is defined as the sum of the average processing time of all the operations on their candidate machines.

New jobs will arrive over time according to a Poisson process with rate λ. *Utilisation level* (denoted as p) is an essential factor to simulate different job shop scenarios. It is the proportion of time that a machine is expected to be busy. The expression of λ is shown in Eq. (1), where μ is the average processing time of the machines, and P_M is the probability of a job visiting a machine. For example, P_M is 2/10 if each job has two operations. A larger utilisation level tends to lead to a busier job shop.

$$\lambda = \frac{p * P_M}{1/\mu} \tag{1}$$

To estimate the steady-state performance, the first 1000 jobs are considered as warm-up jobs and discarded in the objective calculations [7]. This work collects data from the next 5000 jobs. The simulation stops when the 6000th job is finished.

B. Comparisons

In this paper, each multitasking scenario consists of three tasks with different utilisation levels. We consider three multitasking scenarios, and each with a different objective, i.e., mean flowtime (denoted as Fmean), mean tardiness (denoted as Tmean), and mean weighted tardiness (denoted as WTmean). The utilisation levels (0.75, 0.85, and 0.95) are used since they are three typical distinct configurations in JSS [7], [45]. The details of the designed multitasking scenarios are shown in Table II. The relatedness between tasks is an important prerequisite for the effectiveness of multitasking algorithm, and measuring the relatedness between tasks is still an open question [62]. The correlation between the ranks of a set of individuals in terms of their fitness for different tasks was used to measure the relatedness of tasks in [63]. In addition, the data generated in the evolutionary process was successfully used to measure the relatedness between tasks to guide the knowledge transfer between tasks in an online fashion [64], [65], [66]. In this paper, we borrow the idea in [63] to measure the relatedness between the designed tasks. Briefly speaking, the ranks of 10 randomly selected individuals (vectors A and B for task T_1 and T_2, respectively) are based on the fitness obtained by the objective function f_1 and f_2. The ordinal correlation between A and B is defined as the relatedness of T_1 and T_2. The results show that the tasks in each multitasking scenario are related to each other with a correlation value around 0.8. The evolved rule is tested on 50 unseen instances, and the average objective value across the 50 test instances is reported as the test performance of the rule, which can be a good approximation of its true performance.

TABLE II
THE DESIGNED MULTITASKING SCENARIOS.

Task	Scenario 1	Scenario 2	Scenario 3
task 1	<Fmean,0.75>	<Tmean,0.75>	<WTmean,0.75>
task 2	<Fmean,0.85>	<Tmean,0.85>	<WTmean,0.85>
task 3	<Fmean,0.95>	<Tmean,0.95>	<WTmean,0.95>

TABLE III
THE TERMINAL SET OF GP.

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 IV
THE PARAMETER SETTINGS OF GP.

Parameter	Value
Number of subpopulations	3
Subpopulation size	400
The number of elites for each subpopulation	10
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 5
Terminal / non-terminal selection rate	10% / 90%
The number of generations	51
The number of offspring for each task	400*12
Intermediate population size of SMTGP / SMT^{2}GP	400*12*3 / 400*4*3

Four algorithms are taken into comparison here. The GP with k subpopulations to solve k tasks independently named MTGP (*GP with multi-tree representation*) [6], is selected as the baseline algorithm, since there is no surrogate and multitasking mechanism involved. The multitasking algorithm in [37] named M^{2}TGP, since it involves both *GP with multi-tree representation and multitasking*, is also applied in the designed multitasking scenarios. It is the state-of-the-art that introduces multitasking in DFJSS. The algorithm in [29] that uses KNN to build surrogate only, named SMTGP in this paper, is applied in a multitasking framework. The proposed surrogate-assisted multitasking algorithm in this paper is named SMT^{2}GP, since it involves *surrogate, multitasking, and multi-tree GP*.

To verify the effectiveness of the proposed SMT^{2}GP, the approaches of MTGP, M^{2}TGP and SMTGP are compared with SMT^{2}GP in terms of the test objective values on unseen instances. The effectiveness of the constructed surrogates in DFJSS is illustrated by the comparison between MTGP and SMTGP. The effect of multitasking mechanism of SMT^{2}GP is examined by the comparison between SMTGP and SMT^{2}GP.

C. Parameter Settings

The terminal set of GP is shown in Table III. 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 set to $\{+, -, *, /, max, min\}$, following the setting in [67]. Each function takes two arguments. The "/" function is protected division, returning one if divided by zero. The max and min functions take two arguments and return the

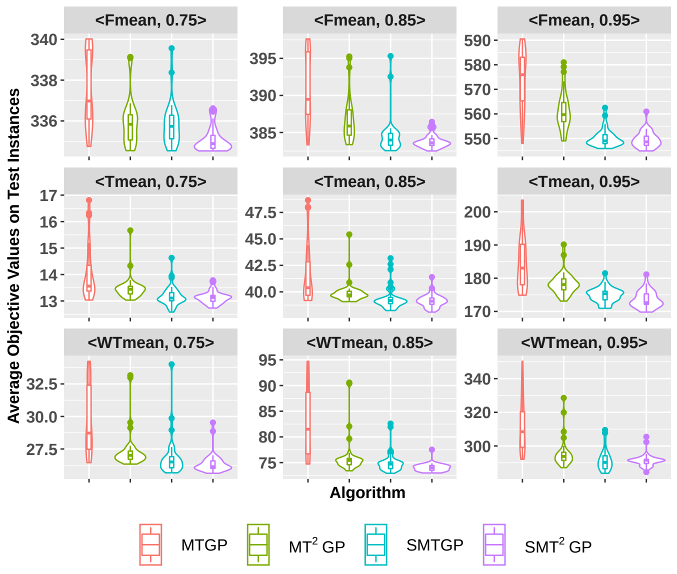

Fig. 4. The violin plot of average objective values on test instances of MTGP, M^{2}TGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios (each row is a multitasking scenario).

maximum and minimum of their arguments, respectively. The other parameter settings of GP are shown in Table IV. Our preliminary experiments show that 400*12 is a maximum intermediate subpopulation size for SMTGP, and no further improvement can be found if the size is further increased. Accordingly, the number of generated offspring from each subpopulation is set to 400*4 for SMT^{2}GP in order to obtain the same number of evaluations with surrogate (400*4*3 = 400*12) as SMTGP for each task.

V. RESULTS AND DISCUSSIONS

Due to the stochastic nature of evolutionary algorithms, Wilcoxon rank-sum test with a significance level of 0.05 is used to examine 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. A smaller value indicates a better performance in minimisation problem.

A. Quality of the Evolved Scheduling Heuristics with Surrogate-Assisted Multitasking

Fig. 4 shows the violin plot of average objective values on test instances of MTGP, M^{2}TGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios. Compared with MTGP, both M^{2}TGP and SMTGP achieve better performance in all scenarios. This shows the effectiveness of using surrogate and multitasking techniques in DFJSS in an independent way, which is consistent to the conclusion drawn in [29], [37]. Compared with M^{2}TGP and SMTGP, SMT^{2}GP achieves better performance with smaller average objective values and standard deviations, illustrating the effectiveness of SMT^{2}GP.

Fig. 5 shows the curves of the average objective values on test instances based on 30 independent runs of MTGP,

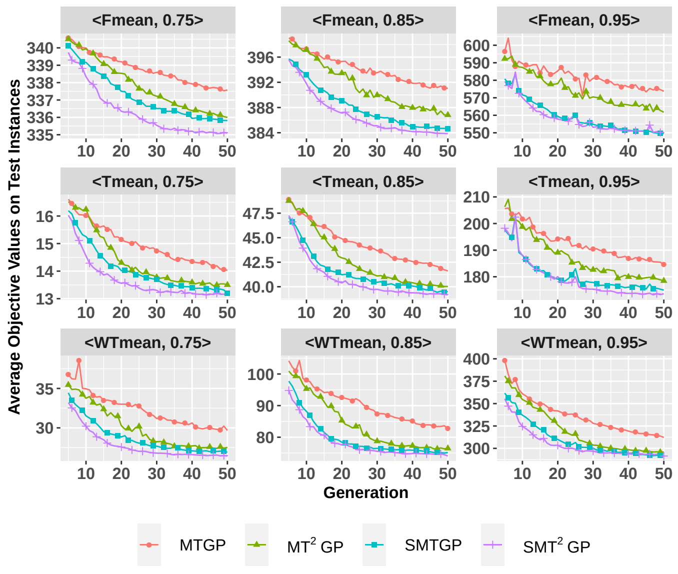

Fig. 5. The curves of the average objective values on test instances based on 30 independent runs of MTGP, M^{2}TGP, SMTGP, and SMT^{2}GP in three multitasking scenarios (each row is a multitasking scenario).

M^{2}TGP, SMTGP, and SMT^{2}GP in three multitasking scenarios with totally nine tasks. The results show that SMT^{2}GP converges faster than MTGP, M^{2}TGP, and SMTGP during the evolutionary process in all multitasking scenarios. As stated earlier, the only difference between SMTGP and SMT^{2}GP is the allocation of individuals for different tasks with the surrogate. This means that the proposed surrogates are capable of sharing knowledge between different tasks by assigning appropriate individuals from the offspring pool for different tasks. This shows the effectiveness of the proposed surrogate-assisted multitasking algorithm in terms of convergence speed and quality of the evolved rules. This also shows that tasks with the same objective but different utilisation levels in DFJSS can be solved in a mutually reinforcing way.

B. *Effectiveness of the Constructed Surrogate in Multitasking*

A proper surrogate for DFJSS is important for the success of surrogate-assisted multitasking. Table V shows the mean and standard deviation of the objective values on unseen instances of MTGP and SMTGP according to 30 independent runs in three multitasking scenarios with nine DFJSS tasks. The results show that SMTGP outperforms MTGP with a significant difference for all the tasks of the examined multitasking scenarios. This shows the effectiveness of the surrogates for DFJSS in multitasking scenarios. In addition, for both MTGP and SMTGP, the objective values in each multitasking task increase along with the utilisation level. This shows that a higher utilisation level leads to a more complex task, which is harder to optimise.

Fig. 6 shows the curves of the average objective values on test instances based on 30 independent runs of MTGP and SMTGP in three multitasking scenarios with totally nine tasks. For all the scenarios, SMTGP performs much better than MTGP after a few generations, i.e., roughly five generations, for all tasks with a higher convergence speed in the examined

TABLE V
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF MTGP AND SMTGP OVER 30 INDEPENDENT RUNS IN THREE MULTITASKING SCENARIOS.

Scenario	Task	MTGP	SMTGP
1	<Fmean,0.75>	337.57(1.80)	335.60(1.20)(–)
	<Fmean,0.85>	391.04(4.65)	384.35(1.86)(–)
	<Fmean,0.95>	573.70(12.17)	548.77(4.60)(–)
2	<Tmean,0.75>	14.03(1.01)	13.15(0.48)(–)
	<Tmean,0.85>	41.61(2.73)	39.16(0.81)(–)
	<Tmean,0.95>	184.60(8.04)	173.32(1.25)(–)
3	<WTmean,0.75>	29.67(2.55)	26.53(0.59)(–)
	<WTmean,0.85>	82.75(6.71)	74.84(2.63)(–)
	<WTmean,0.95>	312.26(15.86)	289.48(6.73)(–)

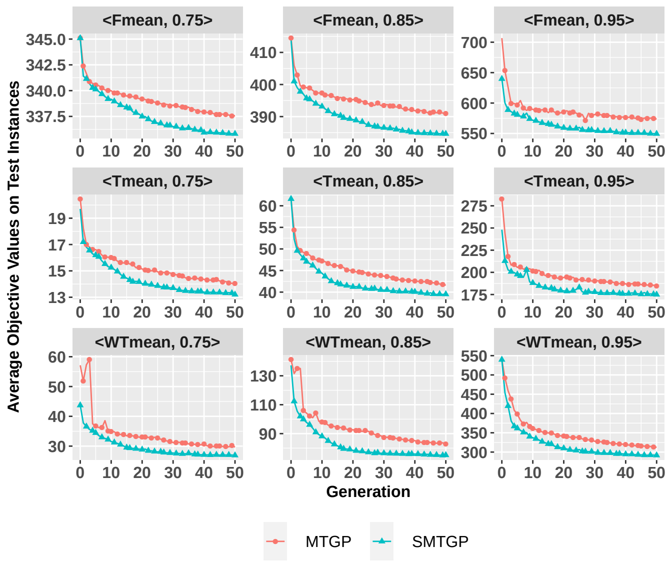

Fig. 6. The curves of the average objective values on test instances based on 30 independent runs of MTGP and SMTGP in three multitasking scenarios (each row is a multitasking scenario).

multitasking scenarios. In most cases, before generation five roughly, the difference of MTGP and SMTGP is not clear. One possible reason is that the sample data in the surrogate with KNN are not accurate before generation five, since the individuals in the population are not well evolved at the beginning of the evolutionary process. In summary, the simulation results show the effectiveness of the way that surrogates are constructed for the DFJSS problems in multitasking scenarios.

C. *Diversity of Individuals for Tasks*

The individuals in the offspring pool are cleared (as shown in Algorithm 3) to reduce the duplicated individuals based on their behaviour before assigning to different subpopulations by the surrogates. Among the four algorithms, only SMTGP and SMT^{2}GP are designed with the clearing process. It is interesting to see the effect of the proposed SMT^{2}GP on the number of cleared individuals in the offspring pool, since the number of cleared individuals is a good indicator for measuring the diversity of the individuals. A smaller number of cleared individuals generally indicates a higher diversity of individuals in the offspring pool.

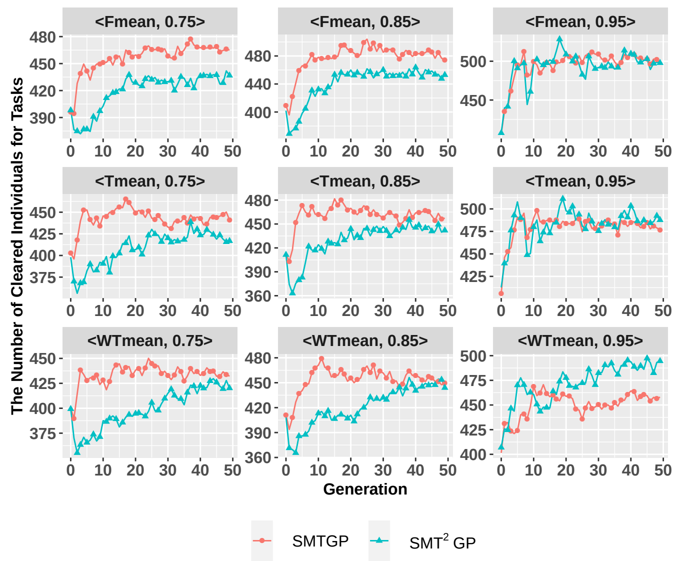

Fig. 7. The curves of the average number of cleared individuals for tasks of SMTGP and SMT^{2}GP over 30 independent runs in three multitasking scenarios (each row is a multitasking scenario).

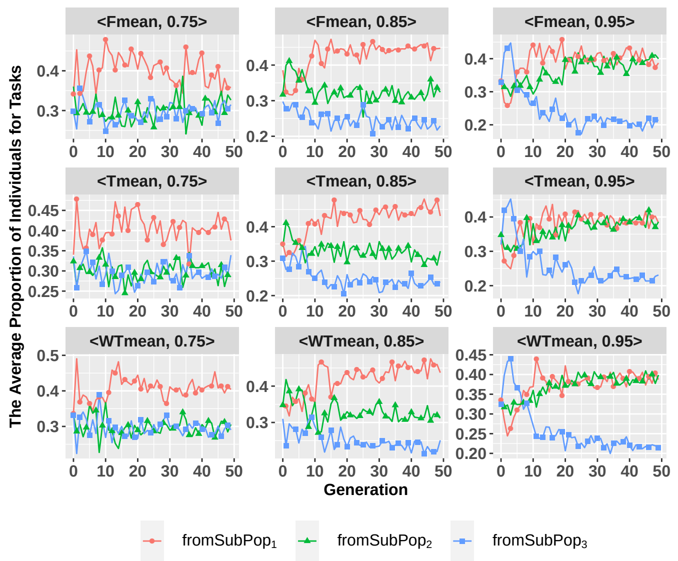

Fig. 8. The curves of the average number of assigned individuals for a specific task from different tasks of SMT^{2}GP over 30 independent runs in three multitasking scenarios (each row is a multitasking scenario).

Fig. 7 shows the curves of the number of cleared individuals for tasks of SMTGP and SMT^{2}GP over 30 independent runs in three multitasking scenarios. The results show that the number of cleared individuals in the tasks with the utilisation level 0.75 and 0.85 of SMT^{2}GP is much smaller than that of SMTGP during the evolutionary process. This indicates that SMT^{2}GP can improve the diversity of generated offspring of $Subpop_1$ for task 1 and $Subpop_2$ for task 2. One possible reason is that the final offspring for task 1 in $Subpop_1$ and task 2 in $Subpop_2$ contain different kinds of individuals from all subpopulations. In this case, the newly generated offspring for task 1 in $Subpop_1$ and task 2 in $Subpop_2$ vary due to the diversity of selected parents. However, it is not the case for task 3 in $Subpop_3$ that optimises the task with a utilisation level 0.95, especially in <WTmean,0.95>.

D. Individual Allocation for Tasks

One of the main ideas in this work is to allocate individuals to appropriate tasks. It is interesting to see the allocation of individuals for each task, i.e., the individuals originally generated for which task. Fig. 8 shows the curves of the average number of assigned individuals for tasks of SMT^{2}GP over 30 independently runs in three multitasking scenarios. The results show that the utilisation level is an important factor for the individual allocation rather than the objective. The tasks with the same utilisation level but with different objectives (the same column) have a similar trend of allocated individuals.

For the subpopulation of task 1 with utilisation level 0.75, the number of individuals originally generated for task 1 fluctuates around 40% over generations. The number of individuals originally generated for task 2 and task 3 shares a similar trend during the evolutionary process, which is 30% roughly. For task 2 with utilisation level 0.85, the number of individuals from $Subpop_1$ for task 1 is the largest, and is approximately 45%. The number of individuals generated in $Subpop_2$ for task 2 ranks second, fluctuating between 30% and 35%. The least number of individuals is generated in $Subpop_3$ for task 3, which is lower than 25% roughly. For task 3 with utilisation level 0.95, the trend of the number of individuals generated for task 1 and task 2 is quite similar, fluctuating between 35% and 40%. However, compared with the number of individuals originally generated for task 1 and task 2, the number of individuals generated for task 3 is quite small. In other words, the individuals for task 3 are not well allocated. This might be the reason that the diversity of individuals in $Subpop_3$ for task 3 of SMT^{2}GP is not as high as expected, which is shown in the last column of Fig. 7. In addition, the offspring generated by the current subpopulation are not guaranteed to perform well in the corresponding problem.

The samples in the pools of the surrogates contain individuals from different subpopulations because of the knowledge transfer. According to the mechanism of KNN, the newly generated individuals in the intermediate population tend to be selected into the next generation if they have similar phenotypic behaviour with the top-ranked individual samples in the surrogate pool. Taking task 3 as an example, based on our investigation which is shown in Section III of the supplementary file, we find that among the individuals in the surrogate pool, the ones from task 1 and task 2 have better ranks than that generated from task 3. On the other hand, the newly generated individuals tend to have similar phenotypic characterisation with the samples in the surrogate pool that are originally for the same task. Therefore, the offspring from task 1 and 2 tend to have better predicated fitness than that from task 3, and thus more likely to be selected into the next generation. In addition, Fig. 7 shows that the number of removed individuals increases from task 1 to task 3. This indicates that task 3 has the smallest number of individuals to be allocated. These explain why the proportion of individuals from $Subpop_3$ is consistently smaller than that from other subpopulations.

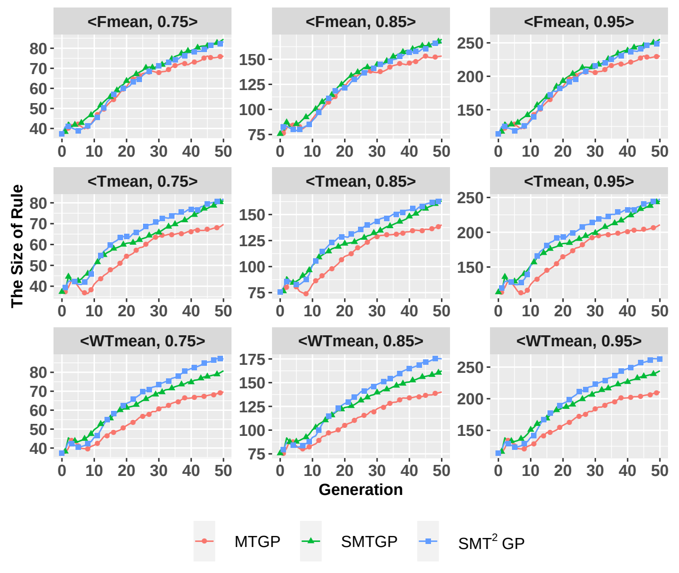

Fig. 9. The curves of the average rule sizes (routing plus sequencing rule) for tasks of MTGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios (each row is a multitasking scenario).

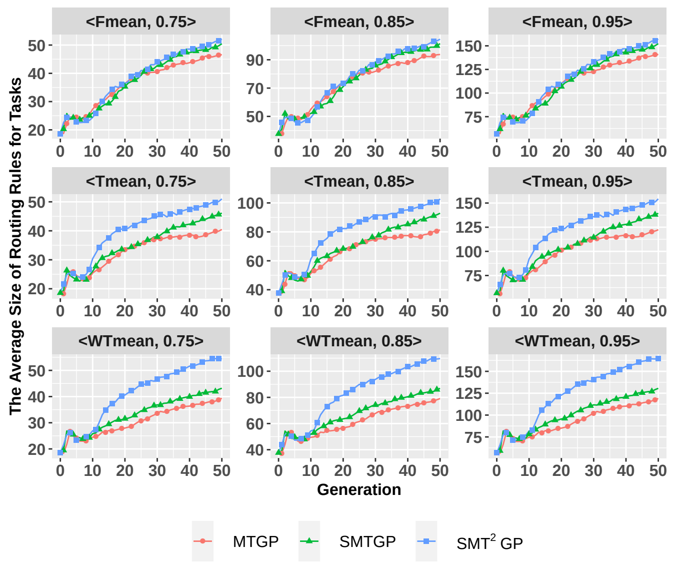

Fig. 10. The curves of the average *routing* rule sizes for tasks of MTGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios (each row is a multitasking scenario).

VI. Further Analyses

A. The Sizes of Evolved Scheduling heuristics

Without considering the bloat problem of GP, the quality of a rule is normally conflicting with its size (the number of nodes). A larger scheduling rule tends to have a better quality than a smaller rule. Therefore, SMTGP and SMT^{2}GP are expected to obtain scheduling heuristics with better qualities and larger rule sizes than MTGP from an early stage of the evolutionary process.

Fig. 9 shows the curves of the average rule sizes including routing and sequencing rules in different tasks of MTGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios. In terms of the complexity of the task, the results show that a more complex task requires a larger rule size for all the algorithms. For example, the rule sizes of the tasks with utilisation level 0.95 are larger than that of the tasks with utilisation level 0.75 and 0.85. The rule sizes of SMTGP and SMT^{2}GP are similar over generations, and the rule sizes of SMTGP and SMT^{2}GP are larger than MTGP in all scenarios from the early stage. This indicates that the inclusion of surrogate and multitasking techniques tend to increase sizes of the evolved scheduling heuristics as compared to MTGP. For SMTGP, one possible reason is that it tends to choose scheduling heuristics that have large sizes due to their good quality in the intermediate population. For SMT^{2}GP, another possible reason is that the parents may have large sizes, since they may come from other subpopulations for complex tasks with large rules. This might also be a reason that the scheduling heuristics with good qualities can be obtained by SMTGP and SMT^{2}GP from the early stage.

To further study the effect of the proposed SMT^{2}GP on the rule size, the routing and sequencing rule are examined respectively due to the similarity of rule sizes of SMTGP and SMT^{2}GP. Fig. 10 and Fig. 11 show the curves of the average routing and sequencing rule sizes for tasks of MTGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios with totally nine tasks, respectively. For the sizes of both routing and sequencing rules, the sizes in each multitasking (i.e., the tasks with the same objective but with different utilisation level) show a similar trend but with different scales. It may be due to the tasks in each multitasking scenario are handled simultaneously by one population with three subpopulations, and the rule sizes highly interact between subpopulations. This is different from the individual allocations for tasks as shown in Fig. 9. The trend of individual allocation is highly related to the utilisation level, while the trend of the rule size, either routing or sequencing, is highly related to the objective examined.

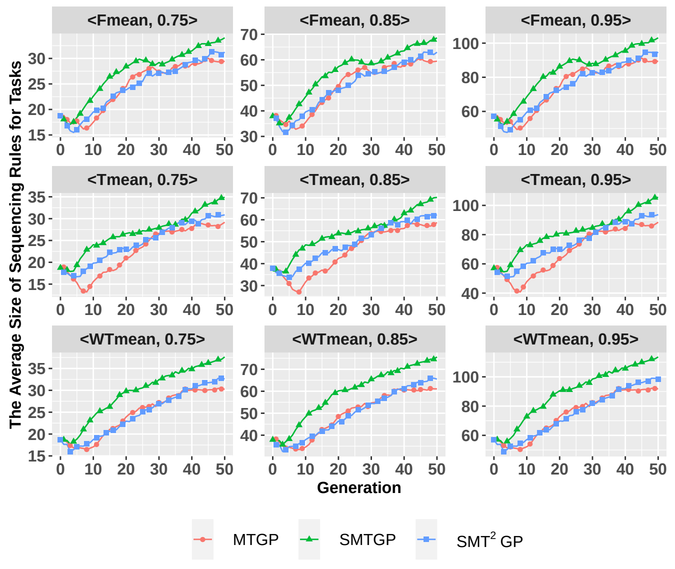

Fig. 11. The curves of the average *sequencing* rule sizes for tasks of MTGP, SMTGP, and SMT^{2}GP over 30 independent runs in three multitasking scenarios (each row is a multitasking scenario).

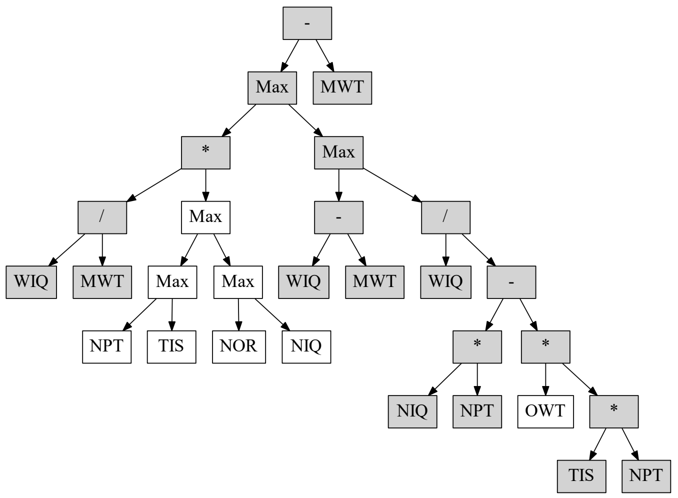

Fig. 12. One of the best evolved routing rules for task 1 <WTmean,0.75> in multitasking scenario 3.

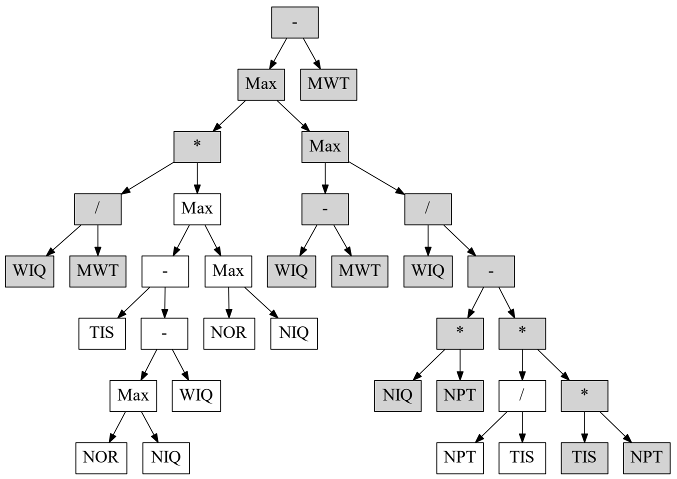

Fig. 13. One of the best evolved routing rules for task 2 <WTmean,0.85> in multitasking scenario 3.

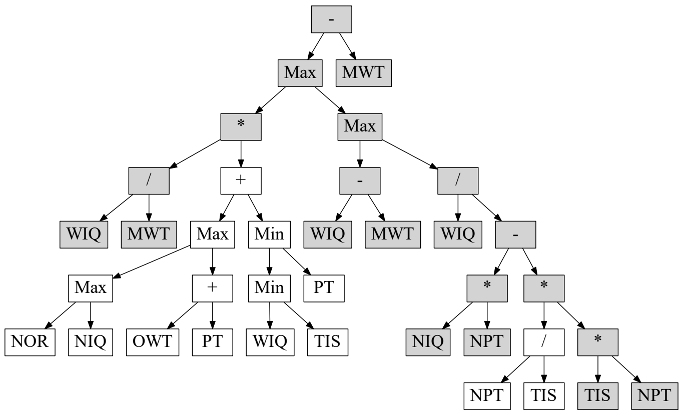

Fig. 14. One of the best evolved routing rules for task 3 <WTmean,0.95> in multitasking scenario 3.

For the sizes of routing rules, as shown in Fig. 10, SMT^2GP tends to evolve larger rules than that of SMTGP for most tasks (e.g., <Tmean,0.75>, <Tmean,0.85>, <Tmean,0.95>, <WTmean,0.75>, <WTmean,0.85>, and <WTmean,0.95>). In addition, the sizes of the evolved routing rules by SMTGP is similar to that of MTGP. For the sizes of sequencing rules, as shown in Fig. 11, SMTGP obtains larger rule sizes than that of SMT^2GP for all the tasks. The sizes of the evolved sequencing rules of SMT^2GP and MTGP are similar. This indicates that the sizes of rules for tasks can be similar but with various size combinations of routing and sequencing. In addition, SMT^2GP improves the quality of evolved scheduling heuristics via routing rule, while SMTGP increases its performance by sequencing rule. It shows that enhancing the quality of routing rule may be an effective way to improve effectiveness of the final schedules in DFJSS.

B. Insight of Evolved Scheduling Heuristics

As stated in the previous subsection, the proposed algorithm SMT^2GP has a significant impact on the routing rule. In this subsection, we choose three routing rules evolved by SMT^2GP for each task in the multitasking scenario 3 that are related to WTmean for further analysis. Figs. 12-14 show the best routing rules for task <WTmean,0.75>, <WTmean,0.85> and <WTmean,0.95> in a multitasking scenario, respectively. These three routing rules are evolved together in the same multitasking scenario.

In terms of the structures of the routing rules, we can see that the major part of the structure of the three routing rules are the same (as shown in grey). One possible reason is that the optimised objective in a multitasking scenario is the same, and the rules for different tasks have inherent similarities. For the evolved building-blocks, except for the building-blocks shown in grey, the routing rule for task 1 shares the same component (i.e., Max{*, Max{NOR, NIQ}}) with the routing rule for task 2. Similarly, "Max{NOR, NIQ}" and "NPT / TIS" are evolved building-blocks of both the routing rule for task 2 and task 3. In addition, "Max{NOR, NIQ}" is a common constructed feature shared by the routing rule of task 1, task 2 and task 3. This is a sign that the tasks in a multitasking scenario learn from each other, which is as expected. In terms of the rule size, the sizes (the number of nodes) of routing rules for task 1, task 2, and task 3 are 29, 35 and 37, respectively. We can see that complex tasks often require larger rules, which is consistent with the observation, as shown in Subsection VI-A.

To make it easy for analysis, we simplify the routing rules by calculating its different components. Note that a machine with a smaller priority value is considered to be more prior. The routing rule for task 1 in Fig. 12 can be further simplified as shown in Eq. (2).

$$\begin{aligned} R_1 =& Max\{\frac{WIQ}{MWT} * Max\{NPT, TIS, NOR, NIQ\}, \\ & WIQ - MWT, \\ & \frac{WIQ}{NIQ * NPT - OWT * TIS * NPT}\} - MWT \\ \approx& Max\{\frac{WIQ}{MWT} * TIS, WIQ - MWT, \\ & \frac{WIQ}{NPT(NIQ - OWT * TIS)}\} - MWT \\ =& \frac{WIQ}{MWT} * TIS - MWT \end{aligned} \tag{2}$$

The Max function connects the main parts of this rule, and the key of this priority function is the component with the largest value. "Max{NPT, TIS, NOR, NIQ}" is simplified as TIS from step 1 to step 2, since TIS is almost always larger than NPT, NOR, and NIQ. Similarly, this rule can be further simplified as $\frac{WIQ}{MWT} * TIS - MWT$. We can see that this routing rule suggests the choice of machine with a small workload (small WIQ) and a long idle time (large MWT). Note than TIS is a constant here which will not make a significant impact on the final decision, since TIS is a property of operation and it is the same for all the candidate machines of an operation.

The routing rule for task 2 in Fig. 13 can be simplified, as shown in Eq. (3).

$$\begin{aligned} R_2 =& Max\{\frac{WIQ}{MWT} * Max\{ \\ & TIS - Max\{NIQ, NOR\} + WIQ, NOR, NIQ\}, \\ & WIQ - MWT, \frac{WIQ}{NPT(NIQ-NPT)}\} - MWT \\ \approx& Max\{\frac{WIQ}{MWT} \\ & * Max\{TIS - Max\{NIQ, NOR\} + WIQ\}, \\ & WIQ - MWT, \frac{WIQ}{NPT(NIQ-NPT)}\} - MWT \\ =& \frac{WIQ}{MWT} * (TIS + WIQ - Max\{NIQ, NOR\}) \\ & - MWT \end{aligned} \tag{3}$$

From step 1 to step 2, "Max{TIS-Max{NIQ, NOR} + WIQ, NOR, NIQ}" can be simplified as "TIS - Max{NOR, NIQ} + WIQ", since "TIS - Max{NOR, NIQ} + WIQ" is more likely to be larger than NOR and NIQ. Compared with Eq. (2), except for WIQ and MWT, this routing rule takes NIQ or NOR into consideration. If NIQ is larger than NOR, this rule will be simplified as $\frac{WIQ}{MWT} * (TIS + WIQ - NIQ) - MWT$. Compared with the routing rule for task 1, except for WIQ and MWT, this rule takes the number of operations in the queue (NIQ) into consideration. This rule suggests selecting the machine that has a larger number of operations but a smaller work in queue. This means that this rule prefers to select the machine with operations that need a short processing time. In this way, the newly allocated operations have a high probability of being processed earlier, since the number of sequencing decision point is increased. If NOR is larger than NIQ, this rule will be simplified as $\frac{WIQ}{MWT} * (TIS + WIQ - NOR) - MWT$. Like TIS, NOR (the number of operations remaining of a job) is a characteristic of the operation, which is a constant for candidate machine and will not affect the routing decision too much. In addition, compared with the routing rule for task 1, WIQ plays an important role in this rule since it appears twice.

The routing rule for task 3 in Fig. 14 can be simplified, as shown in Eq. (4). "Max{NOR, NIQ, OWT + PT}" is denoted as "OWT + PT", since "OWT + PT" is almost always larger than NOR and NIQ. PT is used to indicate "Min{WIQ, TIS, PT}", since PT is usually the smallest one among them. Different from R_1 and R_2, R_3 suggests choosing the machine that has high processing efficiency for a specific operation. It is consistent with our intuition that operations tend to be assigned to the most efficient machine for processing it. In addition, operation waiting time (OWT) can be considered as a constant as it is the same for all its candidate machines of an operation that are expected to be allocated.

$$\begin{aligned} R_3 =& Max\{\frac{WIQ}{MWT} * \{Max\{NOR, NIQ, OWT + PT\} \\ & + Min\{WIQ, TIS, PT\}\}, WIQ - MWT, \\ & \frac{WIQ}{NPT(NIQ-NPT)}\} - MWT \\ \approx& Max\{\frac{WIQ}{MWT} * (OWT + PT + PT), \\ & WIQ - MWT, \frac{WIQ}{NPT(NIQ-NPT)}\} - MWT \\ =& \frac{WIQ}{MWT} * (OWT + 2PT) - MWT \end{aligned} \tag{4}$$

In summary, the routing rules evolved in a multitasking scenario that optimises the same objective have similarities. For all rules, they are highly related to WIQ and MWT, and "WIQ / MWT" is a shared pattern of the routing rules. However, they differ according to the complexities of tasks. For the slightly complex task (task 2), the routing rule pays more attention to the workload of the machine. For the most complex task (task 3), the routing rule focuses on the processing efficiency of the machine. This shows the effectiveness of the proposed surrogate-assisted multitasking algorithm in DFJSS, since it not only shares the knowledge between different tasks but also keeps the unique characteristics for each task.

VII. Conclusions

This paper has proposed a new surrogate-assisted evolutionary multitasking approach for genetic programming hyper-heuristics in solving the dynamic flexible job shop scheduling problems by improving both the effectiveness of solving a single task and transferring knowledge between tasks.

The proposed surrogate-assisted multitasking algorithm has three main features as compared to traditional multitasking framework. Firstly, a large number of new offspring are generated for providing useful materials for tasks rather than combining the parent population and the offspring population. Secondly, the newly generated individuals are evaluated with surrogate rather than actual simulation evaluations. Thirdly, individuals are assigned to optimise tasks based on the estimated fitness by surrogates directly rather than computationally expensive simulation revaluations. The results show that the proposed SMT^{2}GP can evolve highly-competitive scheduling heuristics for dynamic flexible job shop scheduling with high convergence speed for all the examined multitasking scenarios. The effectiveness of SMT^{2}GP are examined by comparing the convergence speed, the quality of evolved scheduling heuristics, the analyses of the diversity of individuals for tasks, structures and behaviours of the evolved scheduling heuristics. It has also been observed that the individual allocations for tasks are highly related to the utilisation level. This implies that the complexities of tasks have a significant impact on knowledge transfer between tasks in a multitasking scenario. In addition, we found that the sizes of the evolved rules over generations for tasks are highly related to the objective examined rather than the utilisation level. This indicates that the rule evolving processes for different tasks highly interact with each other in a multitasking scenario.

Some interesting directions can be further studied in future. It is still not clear what the optimal number of the transferred individuals is, and how the tasks can help each other. In addition, whether tasks with different objectives can be defined as a multitasking scenario can be further examined. We also hope to build a more effective surrogate model such as Gaussian processes and neural networks instead of the KNN.

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] 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.

[4] 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.

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

[6] 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.

[7] ——, "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*. ACM, 2019, pp. 347–355.

[8] 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.

[9] 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.

[10] 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.

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

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

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

[14] J. Pearl, *Heuristics - intelligent search strategies for computer problem solving*, ser. Addison-Wesley series in artificial intelligence. Addison-Wesley, 1984.

[15] 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.

[16] 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.

[17] R. Haupt, "A survey of priority rule-based scheduling," *Operations-Research-Spektrum*, vol. 11, no. 1, pp. 3–16, 1989.

[18] 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.

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

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] M. E. Leusin, E. M. Frazzon, M. Uriona Maldonado, M. Kück, and M. Freitag, "Solving the job-shop scheduling problem in the industry 4.0 era," *Technologies*, vol. 6, no. 4, p. 107, 2018.

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

[27] 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.

[28] Y.-S. Ong, Z. Zhou, and D. Lim, "Curse and blessing of uncertainty in evolutionary algorithm using approximation," in *2006 IEEE International Conference on Evolutionary Computation*. IEEE, 2006, pp. 2928–2935.

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

[30] 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.

[31] 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.

[32] A. Gupta, Y. Ong, L. Feng, and K. C. Tan, "Multiobjective multifactorial optimization in evolutionary multitasking," *IEEE Transactions on Cybernetics*, vol. 47, no. 7, pp. 1652–1665, 2017.

[33] A. Gupta, Y.-S. Ong, and L. Feng, "Multifactorial evolution: toward evolutionary multitasking," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 3, pp. 343–357, 2015.

[34] G. Li, Q. Lin, and W. Gao, "Multifactorial optimization via explicit multipopulation evolutionary framework," *Information Sciences*, vol. 512, pp. 1555–1570, 2020.

[35] L. Zhou, L. Feng, K. C. Tan, J. Zhong, Z. Zhu, K. Liu, and C. Chen, "Toward adaptive knowledge transfer in multifactorial evolutionary computation," *IEEE Transactions on Cybernetics*, 2020.

[36] J. Zhong, L. Feng, W. Cai, and Y.-S. Ong, "Multifactorial genetic programming for symbolic regression problems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2018.

[37] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "A preliminary approach to evolutionary multitasking for dynamic flexible job shop scheduling via genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 2020, pp. 107–108.

[38] Y. Yuan, Y.-S. Ong, A. Gupta, P. S. Tan, and H. Xu, "Evolutionary multitasking in permutation-based combinatorial optimization problems: Realization with tsp, qap, lop, and jsp," in *Proceedings of the IEEE Region 10 Conference*. IEEE, 2016, pp. 3157–3164.

[39] M. Fisher and A. Raman, "Reducing the cost of demand uncertainty through accurate response to early sales," *Operations research*, vol. 44, no. 1, pp. 87–99, 1996.

[40] D. Liu, S. Huang, and J. Zhong, "Surrogate-assisted multi-tasking memetic algorithm," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2018, pp. 1–8.

[41] S. Huang, J. Zhong, and W. Yu, "Surrogate-assisted evolutionary framework with adaptive knowledge transfer for multi-task optimization," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[42] J. Ding, C. Yang, Y. Jin, and T. Chai, "Generalized multitasking for evolutionary optimization of expensive problems," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 44–58, 2017.

[43] A. T. W. Min, Y.-S. Ong, A. Gupta, and C.-K. Goh, "Multiproblem surrogates: transfer evolutionary multiobjective optimization of computationally expensive problems," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 15–28, 2017.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*. Springer, 2005, pp. 127–164.

[49] 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.

[50] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," in *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning*. Springer, nov 2017, pp. 435–447.

[51] R. Hunt, M. Johnston, and M. Zhang, "Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming," in *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014*, 2014, pp. 927–934.

[52] 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.

[53] 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.

[54] J. Ding, C. Yang, Y. Jin, and T. Chai, "Generalized multitasking for evolutionary optimization of expensive problems," *IEEE Transations on Evolutionary Computation*, vol. 23, no. 1, pp. 44–58, 2019.

[55] D. Karunakaran, Y. Mei, and M. Zhang, "Multitasking genetic programming for stochastic team orienteering problem with time windows," in *Proceedings of IEEE Symposium Series on Computational Intelligence*. IEEE, sep 2019.

[56] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "Evolutionary multitask optimisation for dynamic job shop scheduling using niched genetic programming," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2018, pp. 739–751.

[57] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.

[58] 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.

[59] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.

[60] S. Nguyen, M. Zhang, D. Alahakoon, and K. C. Tan, "Visualizing the evolution of computer programs for genetic programming," *IEEE Computational Intelligence Magazine*, vol. 13, no. 4, pp. 77–94, 2018.

[61] ——, "People-centric evolutionary system for dynamic production scheduling," *IEEE transactions on cybernetics*, 2019.

[62] Y.-S. Ong and A. Gupta, "Evolutionary multitasking: a computer science view of cognitive multitasking," *Cognitive Computation*, vol. 8, no. 2, pp. 125–142, 2016.

[63] B. Da, Y.-S. Ong, L. Feng, A. K. Qin, A. Gupta, Z. Zhu, C.-K. Ting, K. Tang, and X. Yao, "Evolutionary multitasking for single-objective continuous optimization: Benchmark problems, performance metric, and baseline results," *arXiv preprint arXiv:1706.03470*, 2017.

[64] Y. Chen, J. Zhong, L. Feng, and J. Zhang, "An adaptive archive-based evolutionary framework for many-task optimization," *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2019, Doi: 10.1109/TETCI.2019.2916051.

[65] K. K. Bali, Y.-S. Ong, A. Gupta, and P. S. Tan, "Multifactorial evolutionary algorithm with online transfer parameter estimation: Mfea-ii," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 1, pp. 69–83, 2019.

[66] K. K. Bali, A. Gupta, Y.-S. Ong, and P. S. Tan, "Cognizant multitasking in multiobjective multifactorial evolution: Mo-mfea-ii," *IEEE Transactions on Cybernetics*, 2020, Doi: 10.1109/TCYB.2020.2981733.

[67] 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.