Instance Rotation Based Surrogate in Genetic Programming with Brood Recombination for Dynamic Job Shop Scheduling

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

***Abstract*—Genetic programming has achieved great success for learning scheduling heuristics in dynamic job shop scheduling. In theory, generating a large number of offspring for genetic programming, known as brood recombination, can improve its heuristic generation ability. However, it is time-consuming to evaluate extra individuals. Phenotypic characterisation based surrogates with K-nearest neighbours have been successfully used for genetic programming to preselect only promising individuals for real fitness evaluations in dynamic job shop scheduling. However, sample individuals used by surrogate are from only the current generation, since the fitness of individuals across generations are not comparable due to the rotation of training instances. The surrogate cannot accurately estimate the fitness of an offspring that is far away from all the limited sample individuals at the current generation. This paper proposes an effective instance rotation based surrogate to address the above issue. Specifically, the surrogate uses the samples extracted from individuals across multiple generations with different instances. More importantly, we propose a fitness mapping strategy to make the fitness evaluated by different instances comparable. The results show that the GP with brood recombination and the proposed surrogate can significantly improve the quality of scheduling heuristics. The results also reveal that the proposed algorithm has successfully reduced the number of omitted promising offspring due to the higher accuracy of the surrogate. The samples in the new surrogate spread better in the phenotypic space, and the nearest neighbour tends to be closer to the predicted offspring. This makes the estimated fitness more accurate.**

***Index Terms*—Surrogate, Instance Rotation, Genetic Programming, Brood Recombination, Dynamic Job Shop Scheduling.**

I. Introduction

Job shop scheduling (JSS) is an important combinatorial optimisation that captures practical and challenging issues in real-world scheduling tasks such as order picking in warehouses [1], designing manufacturing processes (e.g., shop floor control) [2] and managing grid/cloud computing [3]. With a number of jobs (i.e., a job consists of a sequence of operations, and each operation can be processed on a redefined machine), JSS aims to arrange the production of jobs with a set of machines efficiently. Dynamic JSS [4] is an extension of JSS but considers dynamic events such as job arrival over time [5] when executing schedules. Dynamic flexible job shop scheduling (DFJSS) [6], [7] is a typical case of JSS, where each operation can be processed on a set of machines. There are two decisions that need to be made in DFJSS. One is *machine assignment* for allocating a ready operation to a machine. The other is *operation sequencing* for choosing an operation to be processed next when a machine becomes idle. DFJSS is normally studied with a simulation that mimics the dynamic production environment in real-world applications [4]. However, as the scale or complexity of the scheduling problem increases, the approaches based on the simulation optimisation become more and more time-consuming due to the long run of simulation for evaluation. Instances (i.e., represented by simulations) are widely used in dynamic JSS to learn scheduling heuristics [8].

Genetic programming (GP) [9], [10], as a hyper-heuristic approach, has been successfully used to learn scheduling heuristics for JSS [11], [12]. Scheduling heuristics can be regarded as priority functions to prioritise operations or machines at the decision points (e.g., new jobs come or a machine becomes idle). Theoretically, generating and examining more individuals at each generation for GP can speed up its convergence. Typically, brood recombination has been applied to generate a large number of individuals to improve the performance of GP [13]–[15]. However, this requires much more fitness evaluation, and can make GP very slow, especially if the fitness evaluation is very time-consuming (e.g., with long dynamic JSS simulations).

There are two main strategies to handle the above issue. One is the rotation of training instances. Specifically, training instances of GP vary with different generations, i.e., each generation uses one instance, to improve the generalisation of the learned scheduling heuristics [16]–[19]. It is essentially a batch-based training strategy with a batch size of 1. [16] tested different batch sizes, and found that using a new training

Manuscript received XXX; revised XXX and XXX; accepted XXX. This work is supported by the Marsden Fund of New Zealand Government under Contract MFP-VUW1913 and Contract VUW1614; in part by the Science for Technological Innovation Challenge Fund under Grant 2019-S7-CRS; and in part by the MBIE SSIF Fund under Contract VUW RTVU1914. (*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, yi.mei, 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). Kay Chen Tan is with the Department of Computing, Hong Kong Polytechnic University (e-mail: kctan@polyu.edu.hk).

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

instance at each generation (i.e., batch size equals 1) is the best setting. The other is the surrogate techniques to improve the efficiency of individual evaluations [20]. However, most of the studies related to surrogates are conducted on continuous numeric optimisation problems with known optimal solutions [21]–[25]. The research of surrogates on discrete optimisation problems is still limited. In addition, most existing surrogate techniques are applied to the evolutionary algorithms with vector-based fixed-length representations such as genetic algorithms [26] and particle swarm optimisation [27]. However, the studies of surrogate techniques with tree-based variable-length representations such as GP, are still in an early stage.

Phenotypic characterisation based surrogates have been used in GP for dynamic JSS [28], [29]. For each GP individual, a fixed-length phenotypic characterisation is obtained based on a predefined list of decision situations to represent its behaviour. Based on the phenotypic characterisations, the fitness of newly produced offspring by brood recombination are estimated by the real fitness of its nearest neighbour (i.e., KNN with $k = 1$) among the individuals in the current population. This KNN-based surrogate maintains a pool of samples, where each sample is a pair of the phenotypic characterisation of an individual in the current generation and its real fitness. It is noted that the idea of brood recombination proposed in 1993 is very similar to generating a large pool of individuals for preselection in the area of surrogate developed in 2005 [20]. The surrogate is used to preselect only the promising individuals from a large number of newly produced offspring. Only the selected individuals are evaluated with the real instances in the next generation.

Due to the rotation of training instances in GP for dynamic job shop scheduling, the fitness of individuals across different generations are not comparable, and we cannot place them together directly to build KNN-based surrogates. Therefore, the existing studies [19], [28] only take the individuals from the same generation to build the KNN-based surrogate model (i.e., normally the current generation). This limits the capacity of the surrogate. If a newly produced offspring is still far away from the nearest neighbour in the surrogate, its fitness prediction can hardly be accurate. On the other hand, it is impractical to use all the individuals evaluated with different training instances in all the previous generations, since using a large number of samples in KNN-based surrogate indicates a high computational cost requirement.

To tackle this challenge, we aim to propose a fitness mapping strategy to convert the fitness of individuals across different generations into a comparable fitness space. Specifically, the fitness mapping strategy takes the populations from two consecutive generations. Then, it learns a mapping function from the fitness in the former generation to the fitness in the latter generation based on the common individuals in both generations (i.e., those produced by elitism and reproduction). Finally, for each unique individual in the former generation, it converts its fitness evaluated in the former generation to a comparable fitness in the latter generation with the learned mapping function.

The overall goal of this work is to develop an effective instance rotation based surrogate to preselect promising individuals in GP with brood recombination for learning scheduling heuristics for dynamic JSS efficiently. By adding more individual samples to the KNN-based surrogate, the proposed algorithm is expected to estimate the fitness of newly produced offspring more accurately and improve the quality of the learned scheduling heuristics. Specifically, the major contributions of this paper are shown as follows:

1) We developed an effective fitness mapping strategy to make the fitness of individuals obtained with different training instances comparable. The proposed fitness mapping strategy is the first attempt to use the information of GP individuals with fitness obtained from different instances. It also provides a fundamental technique that can broaden the study of synergising the information with instance rotation during the whole evolutionary process of GP for further performance improvement.
2) We proposed an effective instance rotation based surrogate GP with brood recombination that can include more samples from individuals across multiple generations. The results show that the proposed algorithm with the fitness mapping strategy can learn competitive scheduling heuristics for dynamic JSS by improving the accuracy of the fitness estimations of newly produced offspring. To the best of our knowledge, this is the first time to involve the information generated by different training instances to build surrogate for dynamic scheduling.
3) We showed that the proposed GP algorithm with the novel fitness mapping strategy outperforms the state-of-the-art KNN-based surrogate for dynamic scheduling, and can learn highly competitive scheduling heuristics for dynamic JSS. The way to standardise data from different instances can also provide guidance for extracting useful information from heterogeneous data sources and use them together in other problem domains.
4) Further analysis and discussions on the effect of the proposed algorithm reveal that the superiority of the proposed algorithm is realised by the improvement of the accuracy of the fitness estimations of newly produced offspring. Adding more samples with the proposed fitness mapping strategy successfully enlarges the area in the phenotypic space covered by the surrogate model, which improves the fitness estimation due to the ability to find closer samples.

II. Literature Review

A. Dynamic Flexible Job Shop Scheduling

In DFJSS, a number of jobs $\mathcal{J} = \{J_1, J_2, ..., J_n\}$ need to be processed by a set of machines $\mathcal{M} = \{M_1, M_2, ..., M_m\}$. Each job has a sequence of operations that need to be processed one by one. Each operation can be processed on a number of machines [30] (i.e., flexible machine resources). However, each operation will be processed on one of its candidate machines, and its processing time depends on the machine that processes it. This paper focuses on one dynamic event, i.e., dynamically and stochastically arriving new jobs [5], [31], since it is the most common dynamic event in real life. The information about a new job is not known until

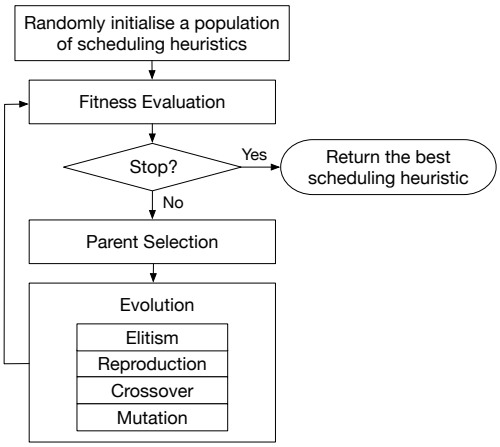


Fig. 1. The flowchart of genetic programming to learn scheduling heuristics for dynamic flexible job shop scheduling.

it arrives at the shop floor. The constraints of the DFJSS problems are shown as follows.

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

In real-world applications, it is common that production providers have different preferences on the optimised scheduling objectives [32]. The providers may prefer to minimise flowtime to reduce the total cost from the perspective of the overall production, or to deliver the products with a minimal delay. Two commonly used objectives are considered in this paper, which are shown below:

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

where C_j is the completion time of a 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 (importance) of job J_j, and n is the number of jobs.

B. Genetic Programming for DFJSS

GP has been widely used to learn scheduling heuristics for dynamic scheduling [17], [19], [33], [34]. GP has three main advantages which make it natural to be a good candidate to learn scheduling heuristics for dynamic scheduling. First, GP has flexible representation to represent various scheduling heuristics for JSS. The scheduling heuristics with the same behaviour can be represented in different genotypes, which provides more diverse genetic materials to generate promising scheduling heuristics during the evolutionary process. More importantly, with GP, we do not need to define the structures of the scheduling heuristics, which are normally unknown in advance. Second, scheduling heuristics represented by GP can be considered as priority functions, which makes it easy to incorporate domain knowledge into the scheduling heuristics.

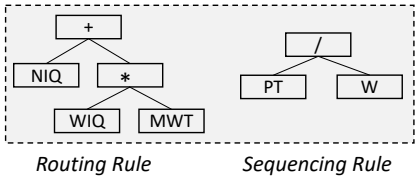


Fig. 2. An example of a genetic programming individual with a routing and sequencing rule for dynamic flexible job shop scheduling.

Last, the scheduling heuristics with tree-based structures are relatively easy to be interpreted, which is important for the applications in real-world.

Fig. 1 shows the flowchart of GP to learn scheduling heuristics for DFJSS. The main processes are the same as the typical GP. The major difference is that the individuals are evaluated with dynamic JSS instances to get their fitness. The output is the learned best scheduling heuristics.

1) Representation: Two decisions, i.e., machine assignment and operation sequencing, need to be made simultaneously in DFJSS. The routing rule and sequencing rule have shown their effectiveness in generating schedules in DFJSS. Using the multi-tree representation of GP has proved to be a good way to learn these two rules simultaneously [35], and we use the representation of GP with two trees in this paper. Fig. 2 shows an example of a GP individual to represent the routing rule and the sequencing rule for DFJSS. The GP individual consists of a routing rule and a sequencing rule, and these two rules work together to generate schedules. The routing rule prioritises machines based on NIQ + WIQ * MWT, where NIQ is the number of operations, WIQ is the total time for a machine to finish the operations in its queue, and MWT is the time for a machine to finish the operation which is currently processing. The sequencing rule is the well-known WSPT rule, which gives priority values to a ready operation according to PT / W, where PT is the processing time of an operation and W is the importance of an operation.

2) Decision Making with Scheduling Heuristics: In DFJSS, decision making is conducted at decision points, i.e., routing and sequencing decision points. Routing decision points are the situations that an operation is ready to be processed (i.e., the first operation of a newly arrived job or the operation whose precedent operations have been processed). Sequencing decision points are the cases that when a machine becomes idle and there are operations waiting in its queue. Taking the routing decision process as an example, Table I shows an example of how the machines are selected to allocate a ready operation. Table I assumes the ready operation can be processed on machines M_1, M_2, and M_3. The priority values of the three machines are calculated based on the routing rule shown in Fig. 2. The priority values of M_1, M_2, and M_3 are calculated as 6100, 5030, and 5040, respectively. As a result, the machine (i.e., M_2) with the smallest priority value (i.e., marked with an underline) is selected to process the operation.

C. Related Work

1) Methods for Job Shop Scheduling: Exact methods such as integer linear programming [36] and dynamic program-

TABLE I
AN EXAMPLE OF THE DECISION MAKING OF THE ROUTING RULE NIQ + WIQ * MWT AT A ROUTING DECISION POINT WITH THREE MACHINES.

Decision Situation	Machine Option	Feature (NIQ	WIQ	MWT)	Priority Value	Chosen Machine
1	M_1	100	200	30	6100	M_2
	M_2	30	100	50	5030	
	M_3	40	50	100	5040	

ming [37], have been used to handle the JSS problems with the goal of finding the optimal solution(s). However, exact methods are mainly applied for dealing with static and small scale JSS problems, and they are not efficient to solve the dynamic and large scale JSS problems with a large number of decision making points. *Heuristic methods* based on iterative improvement such as genetic algorithms [38] and particle swarm optimisation [39], can find good enough solutions efficiently for JSS. However, it is not easy to respond in time to dynamic events well, since they involve rescheduling processes. Dispatching rules [31] can make real-time decision efficiently, however, manually designing effective rules heavily relies on domain experts which are not always available. *Hyper-heuristic methods* aim to automatically select or generate heuristics for handling problems which explore a search space of low-level heuristics [40]–[42]. There are two types of commonly used hyper-heuristic approaches [43], i.e., heuristic selection [44] (i.e., select proper heuristics to use in different scheduling environment) and heuristic generation [45]. Heuristic generation methods such as GP, have been successfully used to generate comprehensive scheduling heuristics for JSS, especially in dynamic JSS. For DFJSS, we normally learn a routing rule for machine assignment, and a sequencing rule for operation sequencing. Only the sequencing rule was learned by fixing the routing rule in [46]. Cooperative coevolution was applied to learn the routing rule and the sequencing rule simultaneously in [47]. GP with multi-tree representation has also been successfully used to learn these two rules [35]. The results show that learning two rules simultaneously with GP is an effective approach for DFJSS.

2) Genetic Programming Hyper-heuristics: As a hyper-heuristic approach, GP has been used to learn rules in different domains. In dynamic JSS, GP has been successfully used to learn sequencing rules to prioritise operations in the queues of machines [48]. In DFJSS, GP has been used to learn routing rules for machine assignment and sequencing rules for operation sequencing simultaneously [49]. GP has been also used in other domains, e.g., arc routing [50], to learn routing policy for arranging the vehicles to serve the demand of edges.

Existing studies of GP to learn rules in different domains show that rotating training instances is an efficient way to learn effective rules [16]. With instance rotation, only one instance is used at each generation to evaluate individuals, this reduces the number of individual evaluations and maintains a good generalisation ability for GP in handling problems.

3) Surrogate-Assisted Genetic Programming: Surrogate-assisted evolutionary algorithms aim at dealing with the expensive individual evaluation in evolutionary algorithms [20], [51]. We focus on surrogate GP methods and group the existing studies into two categories based on the problems they work on, i.e., continuous numeric optimisation and discrete combinatorial optimisation. Note that surrogate-assisted evolutionary algorithms are used almost exclusively in the field of continuous numeric optimisation. This section will discuss the related studies with a focus on the way of building surrogates.

Continuous numeric optimisation: There are a number of commonly used surrogate techniques such as artificial neural networks [52], radial basis functions [53], and Gaussian process [54], which have been used to build surrogates for estimating the fitness of individuals. A surrogate model was built based on the information obtained from real evaluated individuals with radial basis functions to evaluate λ percentage individuals [55]. The extracted features in terms of the genotypes of GP individuals such as tree size and tree depth were used to build a surrogate with random forest for regression [56]. However, these kinds of surrogate models are implicitly or explicitly spatial with the assumption of spatial smoothness or continuity of the target model. This makes them naturally suited for continuous numeric optimisation. However, they are not suitable for the problem investigated in this paper, which is a discrete combinatorial optimisation problem.

Discrete combinatorial optimisation: Phenotypic characterisation was first developed to represent the behaviour of GP individuals in dynamic JSS in [28]. Then, a KNN-based surrogate was proposed by using the current real evaluated individuals (i.e., their phenotypic characterisations and real fitness) as samples to preselect a small number of promising individuals from a large number of individuals to do the real evaluation in the next generation. The results verified the effectiveness of the GP algorithm with a surrogate-assisted preselection strategy in dynamic JSS. Surrogate techniques for dynamic JSS by simplifying the simulation model were used in [57], [58]. Multi-fidelity based surrogates were also investigated based on simplifying the original simulation to different degrees [59]. The above studies all verified the effectiveness of the proposed surrogate techniques, either based on KNN or simplifying the original instance. KNN-based surrogates have the advantages of being easy to learn and computationally cheap to build. Therefore, this paper will focus on KNN-based surrogates. The KNN based surrogate models are limited to the use of individual information from the current (the same) generation, which is one of the major motivations of this paper.

The studies on surrogate GP for dynamic JSS are still in their infancy. This paper will focus on KNN-based surrogates to investigate how to use the data generated across different instances in dynamic JSS to further enhance the accuracy of fitness estimations by the surrogate. This paper will contribute to the development of surrogate-assisted evolutionary algorithms with variable-length tree-based representation on discrete combinatorial optimisation.

III. INSTANCE ROTATION BASED SURROGATE IN GENETIC PROGRAMMING WITH BROOD RECOMBINATION

A. Framework of the Proposed Algorithm

Fig. 3 shows the framework of the proposed algorithm, and its new components are highlighted. The main difference

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3180693

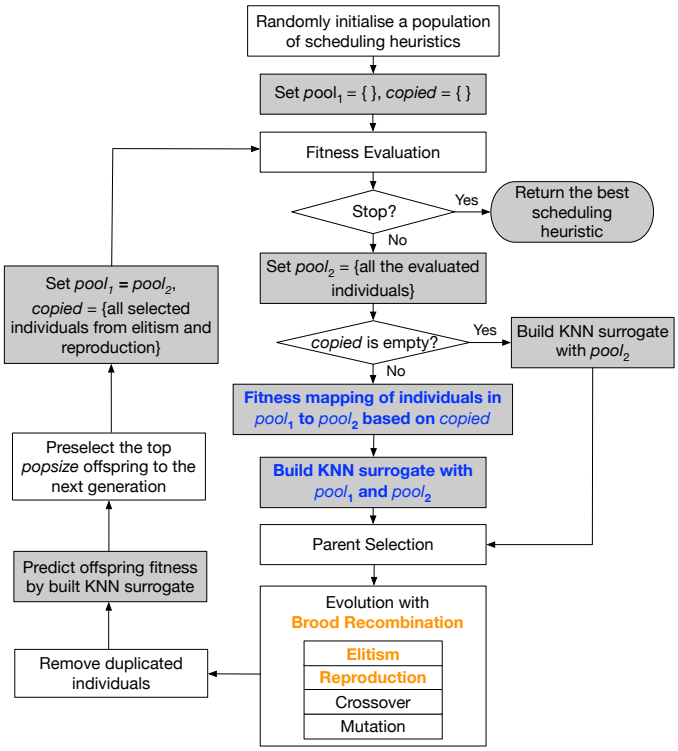


Fig. 3. The flowchart of the proposed algorithm.

between the proposed algorithm and the generic GP shown in Fig. 1 are highlighted with the grey background. The key idea of the proposed algorithm is to build an effective KNN surrogate for estimating the fitness of individuals produced by brood recombination efficiently. We use $pool_1$ and $pool_2$ to denote the individuals at the previous and current generations, respectively. The set $copied$ denotes the individuals produced by elitism and reproduction, which are essentially copied from the previous generation to the current generation. These individuals are the common individuals between two consecutive generations. *At the initialisation stage*, the GP population is randomly initialised. $pool_1$ and $copied$ are empty at the beginning. *During the fitness evaluation stage*, each individual is evaluated by applying it to a training instance to get the schedule and calculating the objective value based on the optimised objective function. All the evaluated individuals at the current generation are kept in $pool_2$. If $copied$ is empty (i.e., when it is the first generation), only $pool_2$ is used to build the KNN surrogate. Otherwise, the proposed fitness mapping strategy is used to map the fitness of individuals in $pool_1$ to $pool_2$ based on the individuals in $copied$. Then, the surrogate is built based on $pool_1$ and $pool_2$. *During the evolution stage*, brood recombination is applied to produce a large number of offspring based on selected parents with genetic operators, i.e., elitism, reproduction, crossover and mutation. After that, we remove the duplicated individuals which have the same behaviour (i.e., the behaviour is represented by phenotypic characterisation introduced in Section III-C). Then, we use the KNN surrogate to estimate the fitness of all newly produced offspring by brood recombination. The top $popsize$ (i.e., population size) individuals based on estimated fitness are selected

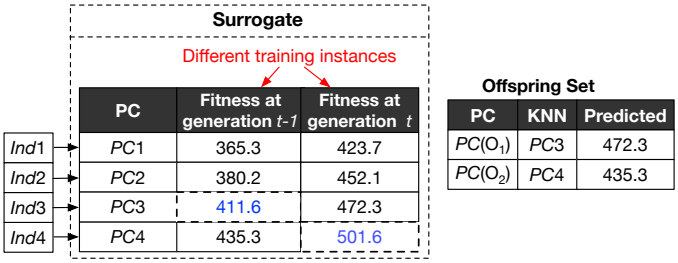


Fig. 4. An example for showing the issue of using the samples directly across different generation/instances.

to the next generation. $pool_1$ is set to $pool_2$ for building surrogate at the next generation. $copied$ is updated based on the selected individuals produced by elitism and reproduction (i.e., highlighted in orange). If the stopping criterion is met, the best learned scheduling heuristic at the current generation is reported as the output of the proposed algorithm.

There are two main steps in the proposed algorithm (i.e., highlighted in blue). One is the fitness mapping of individuals and the other is the building of the surrogate. The details of them are described in the following subsections.

B. Proposed Fitness Mapping Strategy

The individuals at different generations are evaluated with different instances, which results in incomparable fitness that are unreasonably put together for the use of building KNN-based surrogates directly. Fig. 4 shows an example to illustrate this issue. Note that $t-1$ and t indicate the previous generation and the current generation, respectively. In Fig. 4, the surrogate includes 4 individuals, where individuals 1 and 2 are evaluated with both instances at generations $t-1$ and t (i.e., they are common individuals inherited by elitism or reproduction). Individual 3 is evaluated with the only instance at generation t, and individual 4 is evaluated with the only instance at generation $t-1$. For convenience, their true fitness unknown to the surrogate at the corresponding generations are also given and highlighted with blue colour. For the two offspring O_1 and O_2, suppose we find that the nearest neighbours (KNN) of O_1 and O_2 are individual 3 and individual 4, respectively. If we use the real fitness of these two KNNs, then the predicted fitness of O_1 and O_2 are 472.3 and 435.3, respectively. Therefore, O_2 is considered to be better than O_1. However, this is contradictory with the fact that individual 3 is better than individual 4 ($411.6 < 435.3$, $472.3 < 501.6$). This mistake was made by comparing the real fitness of the individuals evaluated with different instances.

To address this issue, this paper proposes to utilise the common individuals between two consecutive generations to learn a fitness mapping function to map the fitness of individuals across different generations. Note that mapping across generations is essentially mapping fitness obtained on different instances. There are four main steps for the proposed fitness mapping strategy. The details are shown below with an example as shown in Fig. 5.

- Step 1: Find out the common individuals between two consecutive generations. The common individuals are

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3180693

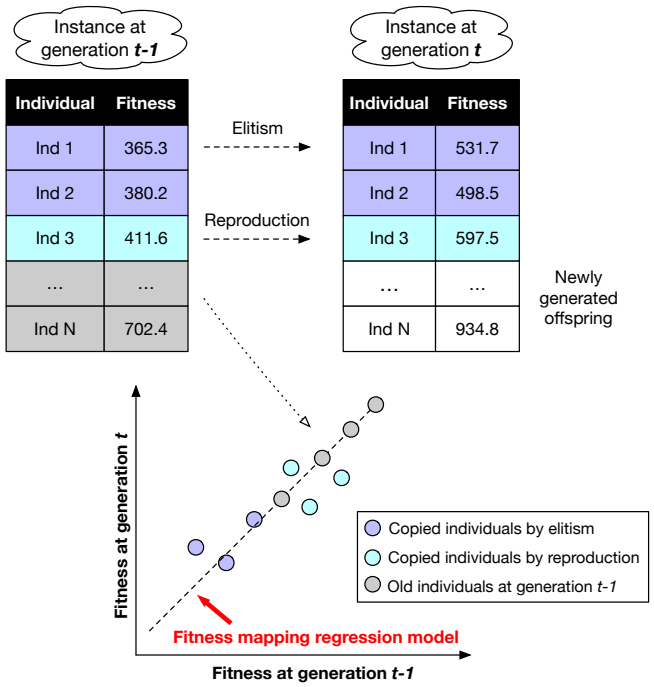


Fig. 5. An example of the proposed fitness mapping strategy.

from two sources. The first source is from the elites which are several top individuals copied from the previous generation to the current generation directly (i.e., marked as squares). The second is from the individuals produced by the reproduction operator which are selected based on the tournament selection method (i.e., marked as stars). The fitness of these common individuals will be used to learn the mapping from the fitness on the instance at generation $t-1$ to that at generation t. Note that we can use the instance at any generation between 1 and $t-1$ and learn its mapping to the current instance at generation t. However, to learn the mapping accurately, we require as many common individuals as possible, with fitness values that have been evaluated on both instances. Therefore, we select the generation t-1 to learn the mapping, since this is where common individuals, evaluated on the instances in subsequent generations, are the most frequent.

- Step 2: Build fitness sets of common individuals at the previous generation $t-1$ (i.e., $fit_{t-1}(ind1)$, $fit_{t-1}(ind2)$, ..., $fit_{t-1}(indn)$) and the current generation t (i.e., $fit_t(ind1)$, $fit_t(ind2)$, ..., $fit_t(indn)$) where n is the number of common individuals.

$$\begin{matrix} ind1 \\ ind2 \\ ..... \\ indn \end{matrix} \begin{pmatrix} fit_{t-1}(ind1) \\ fit_{t-1}(ind2) \\ ..... \\ fit_{t-1}(indn) \end{pmatrix} \longrightarrow \begin{pmatrix} fit_t(ind1) \\ fit_t(ind2) \\ ..... \\ fit_t(indn) \end{pmatrix}$$

- Step 3: Learn a linear regression model (i.e., $fit_t(fit_{t-1}(ind_i); \alpha_t, \beta_t) = \alpha_t * fit_{t-1}(ind_i) + \beta_t$) to map the fitness of individuals at the previous generation to the current generation. α and β are the slope and intercept, respectively. The linear regression model is learned by minimising the total squared error which is shown in Eq. (1).

$$fit_t(fit_{t-1}; \alpha, \beta) = \arg\min_{\alpha,\beta} \sum_{i=1}^{n} (\alpha * fit_{t-1}(ind_i) + \beta - fit_t(ind_i))^2 \quad (1)$$

The model can be learned from the $(fit_{t-1}(ind_i), fit_t(ind_i))$ data points generated from Step 2. We select a linear model here, since we have observed linear relationship between $fit_{t-1}(ind_i)$ and $fit_t(ind_i)$ for various generations in our preliminary study. In addition, if we use linear model, then the linear model structure is independent of the training instances used at different generations, since the linear relationship is symmetric and can be propagated (only with different coefficients). In other words, if we randomly shuffle the training instances across different generations, the linear model structure still holds. Linear regression also has a low computational cost requirement that can get the fitness mapping quickly.

- The sample individuals at generation $t-1$ which are unseen to generation t, are mapped to generation t based on the learned fitness mapping strategy. We use the phenotypic characterisations of those individuals at generation $t-1$ with their new fitness to build the surrogate at generation t. The details are given in the next subsection.

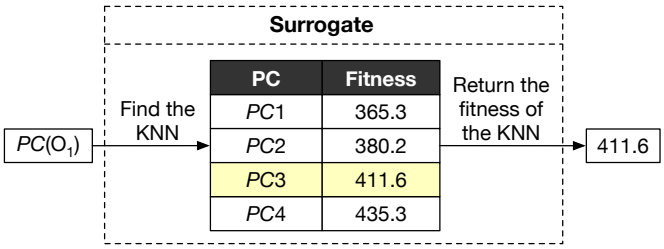


Fig. 6. An example of the working process of KNN-based surrogate in GP.

C. Surrogate Building and Updating

Surrogate is built and updated at each generation, and the samples of KNN-based surrogate consist of the phenotypic characterisations and the corresponding fitness of real evaluated individuals as shown in Fig. 6. When estimating the fitness of an offspring O_1, we find its nearest neighbour to the samples in the surrogate (i.e., $PC3$). The fitness of O_1 is estimated as the corresponding fitness of $PC3$ (i.e., 411.6).

1) Building: At each generation, the individuals in the population are evaluated and their fitness on the training instance at that generation is called the "real fitness", as opposed to the "estimated fitness" by the surrogate model. This is the information from the traditional evolutionary process of GP, and we utilise this information to build the surrogate without requiring much extra computational cost. One key to extract samples for KNN-based surrogate is to measure the behaviour of real evaluated individuals. We introduce to use phenotypic characterisation for this purpose due to the variable-length and tree-based representation of GP [28]. It

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

Decision Situation	WIQ	Routing Rule	PC_i
1 (M_1)	1	3	
1 (M_2)	3	2	2
1 (M_3)	2	1	
2 (M_1)	2	2	
2 (M_2)	1	3	3
2 (M_3)	3	1	

PC_i indicates the i^{th} dimension of phenotypic characterisation.

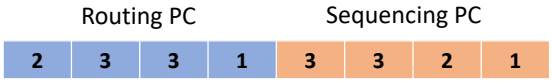


Fig. 7. An example of the phenotypic characterisation of a genetic programming individual which is a eight dimensional vector.

is noted that calculating the phenotypic characterisations for individuals is computationally cheap, which is negligible.

The phenotypic characterisation of a GP individual is a vector of the rank number of machines or operations which represents its decision making behaviour based on a set of decision situations. Each dimension of phenotypic characterisation is the corresponding rank of the first prioritised machine or operation which is decided by the examined routing or sequencing rule assigned by the reference rule.

Table II shows an example of how to calculate phenotypic characterisation for a routing rule with two decision situations, each with three candidate machines. PC_i indicates the value of the i^{th} dimension of phenotypic characterisation, which reflects the behaviour of a rule in decision situation i. We use the reference routing rule WIQ (i.e., least work in the queue) to rank all machines first. For the first decision situation, the routing rule chooses M_3, and therefore PC_1 is the rank of M_3 by WIQ (i.e., 2). Similarity, PC_2 is 3. It is noted that we can also get the phenotypic characterisation of the sequencing rule in the same way. The reference sequencing rule used in this paper is SPT (i.e., shortest processing time). Since we use GP with multi-tree representation (as shown in Fig. 2) to learn the routing and sequencing rule simultaneously for DFJSS, we combine the phenotypic characterisation of routing rule and sequencing rule together as the phenotypic characterisation of a GP individual. An example of the phenotypic characterisation which is a 8-dimensional vector (i.e., with four routing and four sequencing decision situations) of a GP individual for DFJSS can be found in Fig. 7.

2) Updating: The samples of the surrogate used at the current generation are extracted from the individuals at the current generation and the ones at the previous generation. To keep the samples in the KNN-based surrogate model identical, we put the samples extracted from the individuals at the current generation to the surrogate first. Then, we loop all the individuals at the previous generation $pool_1$ and only convert the ones which are different from any of the ones in the current generation $pool_2$ with the proposed fitness mapping strategy to the surrogate. In this case, more samples can be used for the surrogate, and newly produced offspring have more chances to be estimated by real-evaluated closer individuals to improve the accuracy of their fitness estimation.

Algorithm 1: Build and update surrogate

Input : $pool_1$, $pool_2$ and $copied$
Output: surrogate
surrogate = $\emptyset$
for *ind in* $pool_2$ **do**
  Calculate the phenotypic characterisation $PC(ind)$
  surrogate = surrogate $\cup$ $<PC(ind), fit(ind)>$
end
Build the fitness mapping function $g(x;\alpha,\beta)$ based on the data $\{(fit_{t-1}(ind), fit_t(ind)) \mid ind \in copied\}$ and Eq. (**??**)
for *ind in* $pool_1$ **do**
  if $ind \notin pool_2$ **then**
    Calculate the phenotypic characterisation $PC(ind)$
    $fit'(ind) = g(fit(ind);\alpha,\beta)$
    surrogate = surrogate $\cup$ $<PC(ind), fit'(ind)>$
  end
end
return surrogate

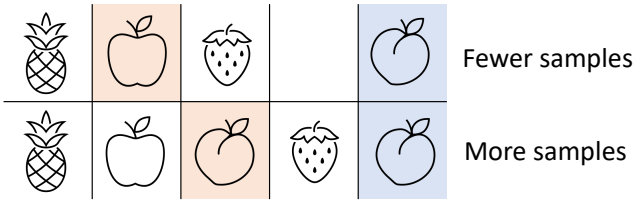


Fig. 8. An example of surrogate with less and more samples.

Algorithm 1 shows the details of the process of building a surrogate. The input is the common individuals $copied$, $pool_1$ and $pool_2$. The output is the built surrogate. First, the phenotypic characterisations of individuals in $pool_1$ are calculated, and the tuple $<PC(ind), fit(ind)>$ are added to the surrogate. Second, a fitness mapping function, which is a linear regression model is built based on the relationship between $fit_{t-1}(ind)$ and $fit_t(ind)$ where individuals belong to $copied$, by minimising the total squared error between them. Last, for the individuals in $pool_1$ which are unseen in $pool_2$, their phenotypic characterisations are calculated and their fitness are mapped with the learned fitness mapping function. The tuples of $<PC(ind), fit'(ind)>$ are added into the surrogate.

Fig. 8 shows an example of the difference between the surrogate with different numbers of samples. Assume we have a KNN-based fruit surrogate with a number of fruit samples, and we need to estimate the names of unknown fruits which are highlighted in blue. For the surrogate with a smaller number of fruits (i.e., pineapple, apple, strawberry), the unknown fruit will be compared with all samples in the fruit surrogate, and the apple will be selected (i.e., highlighted in orange) since they are most similar one. The unknown fruit is named as apple by the surrogate with fewer samples. However, if we use the surrogate with more samples (i.e., pineapple, apple, peach, and strawberry), the unknown fruit will be predicted as peach which is actually true. We can see that there is a higher chance of finding similar or the same samples with more samples to have a better estimation.

TABLE III
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

IV. EXPERIMENT DESIGN

A. Simulation Model

Refer to widely used DFJSS instances [46], [60], and following the settings in [61], the instances used in this paper assume that 5000 jobs need to be processed by 10 machines. New jobs will arrive over time according to a Poisson process with a rate λ. The number of operations of a job is 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. 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 set at 1.5 times of its processing time. The weights (importance) of 20%, 60%, and 20% of jobs are set as 1, 2, and 4, respectively [28]. To improve the generalisation of the evolved scheduling heuristics, the training instance used at each generation (i.e., one instance per generation) is changed by assigning a new random seed of the simulation [62].

Utilisation level (p) is a factor to simulate different job shop scenarios [63]. It is the proportion of time that a machine is expected to be busy, which is controlled by the λ in Poisson process. The utilisation level is calculated as $\lambda = \mu * P_M/p$, 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.

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

B. Design of Comparisons

We consider two different objectives, i.e., mean-flowtime (denoted as Fmean) and mean-weighted-tardiness (denoted as WTmean) and three utilisation levels (i.e., 0.75, 0.85, and 0.95) which are typical distinct configurations in DFJSS [64], [65] to generate examined scenarios. A scenario represents a specific problem to be solved, containing the instances generated by the same problem configuration, e.g., the same objective and utilisation level. For brood recombination, we use Rep to define the number of produced offspring, i.e., $popsize * Rep$ (i.e., brood size), to investigate the effect of the number of individuals on the proposed algorithm. 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 the true performance of the rule.

The GP system without surrogate, named GP, is the baseline algorithm. The state-of-the-art GP with KNN-based surrogate [28], named SGP, is compared to verify the effectiveness of the proposed algorithm. When including the samples across different instances, a straightforward approach is to estimate the fitness of an individual by finding the nearest neighbour from each generation, and average the fitness of the nearest neighbours from each generation as the final estimated fitness. This algorithm is named $\mathrm{SGP\_average}$ for comparison. We also compare with the algorithm that directly adds the samples from the previous generation without adjusting the fitness (i.e., named $\mathrm{SGP\_naive}$). The proposed algorithm is named $\mathrm{SGP\_mapping}$, since it involves a fitness mapping strategy for the individual information from different instances.

A main difference between SGP and $\mathrm{SGP\_mapping}$ is that more samples are used in $\mathrm{SGP\_mapping}$. The comparison between SGP and $\mathrm{SGP\_mapping}$ can help verify the idea of $\mathrm{SGP\_mapping}$ well. Neither $\mathrm{SGP\_average}$ nor $\mathrm{SGP\_naive}$ has fitness mapping strategy, $\mathrm{SGP\_average}$ treats the samples from generations separately, while $\mathrm{SGP\_naive}$ just simply ignores the fitness incomparable issue and treat the samples from different generations in the same way. Comparing $\mathrm{SGP\_mapping}$ with $\mathrm{SGP\_average}$ and $\mathrm{SGP\_naive}$ can verify the importance of fitness mapping strategy.

C. Parameter Settings

The features of the job shop are considered as the terminals of GP [29]. The features are commonly extracted based on the characteristics of machines (i.e., NIQ, WIQ, and MWT), operations (i.e., PT, NPT, and OWT), and jobs (i.e., WKR, NOR, W, and TIS) in the job shop floor. The details are shown in Table III. The function set is set to $\{+, -, *, /, max, min\}$, following the setting in [29]. Each function takes two arguments. The "/" function is the 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 as suggested in [28], [29], [66], are shown in Table IV. Following the suggestions in [28], we set the population size to 500, and the number of produced offspring as 500 * Rep (2, 3, 5, 10). Some parameter settings, i.e., the number of elites and parent selection, follow the suggestions in [29] which have been showing their effectiveness in GP for DFJSS. Note that a number of parameters such as method for initialising population and the number of generations, are commonly used parameter settings in GP [66].

In this paper, we pick 20 routing decision situations and 20 sequencing decision situations in a long simulation for getting the phenotypic characterisations of GP individuals. Thus, the dimension of the phenotypic characterisation of an individual is 40. The decision situations are fixed to calculate the phenotypic characterisations of all individuals. It is noted that the number of operations and machines is set to the

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3180693

TABLE IV
THE PARAMETER SETTINGS IN GP.

Ref.	Parameter	Value
[28]	Population size	500
	The number of produced offspring	500 * *Rep* (2, 3, 5, 10)
[29]	The number of elites	10
	Parent selection	Tournament selection with size 5
[66]	Crossover / Mutation / Reproduction rate	80% / 15% / 5%
	Method for initialising population	ramped-half-and-half
	Initial minimum / maximum depth	2 / 6
	Maximal depth of programs	8
	Terminal / non-terminal selection rate	10% / 90%
	The number of generations	51

same number (i.e., 7) to make the phenotypic characterisations between individuals comparable.

V. RESULTS AND DISCUSSIONS

Friedman's test and Wilcoxon rank-sum test with a significance level of 0.05 is used to examine the performance of the proposed algorithm with 30 independent runs. "Win, Draw, Lose" means the number of scenarios that the proposed algorithm SGP_mapping is statistically better, similar, or worse than a compared algorithm. "Average Rank" shows the average ranking of the algorithm on all the examined scenarios. An algorithm is also compared with the algorithm(s) before it one by one. In the following results, "–", "+", and "≈" indicate the corresponding result is statistically significantly better than, worse than, or similar to its counterpart. It is noted that we work on minimisation problems in this paper, and a smaller value indicates a better performance.

A. Quality of Evolved Scheduling Heuristics

1) Quality of the Evolved Best Scheduling Heuristics: Table V shows the mean and standard deviation of the objective values on test instances according to 30 independent runs of GP, SGP, SGP_average, SGP_naive, and SGP_mapping with replications of 2, 3, 5, and 10 in six scenarios. Based on the Friedman's test, we can see that the proposed algorithm SGP_mapping performs the best, since it achieves the smallest rank with a value of 2.06 among all the involved algorithms. SGP_mapping wins GP and SGP_naive in all of the scenarios, and outperforms SGP/SGP_average in 19/16 out of 24 scenarios. The results also show that the proposed algorithm SGP_mapping outperforms all the other algorithms in the experiments with different replications, since SGP_mapping wins the other algorithms in most of the scenarios. In general, SGP_average gets similar performance with SGP. In addition, SGP_naive is significantly worse than SGP_mapping in all scenarios and significantly worse than SGP half of the scenarios, which indicates that putting samples across different instances to KNN-based surrogate model directly brings noise for the surrogate and deteriorates the fitness estimations of individuals. This verifies the effectiveness of SGP_mapping from an opposite angle.

Based on "Win / Draw / Lose", the results show that the advantage of SGP_mapping is more obvious in the experiments with larger replications (i.e., big brood size). This is consistent with our intuition that higher requirements for the accuracy of fitness estimation may be required when there is a larger number of individuals to be evaluated. In other words, it becomes increasingly harder for a surrogate model with lower accuracy than a surrogate with higher accuracy to select promising individuals from a larger number of individuals. Taking an extreme example, assume that surrogate S_1 has 100% prediction accuracy and surrogate S_2 has 0% prediction accuracy (i.e., random guess). If there are only two individuals for S_1 and S_2 to select the best one between them, the random surrogate S_2 still has 50% chance to select the right individual. However, if there are five individuals to select from, $S2$ will have only 20% chance to select the best one. On the other hand, SGP_mapping is significantly better than SGP_naive in all the scenarios, which verifies the effectiveness of the proposed algorithm with the developed fitness mapping strategy, and the importance of standardising the fitness of the individuals obtained with different instances.

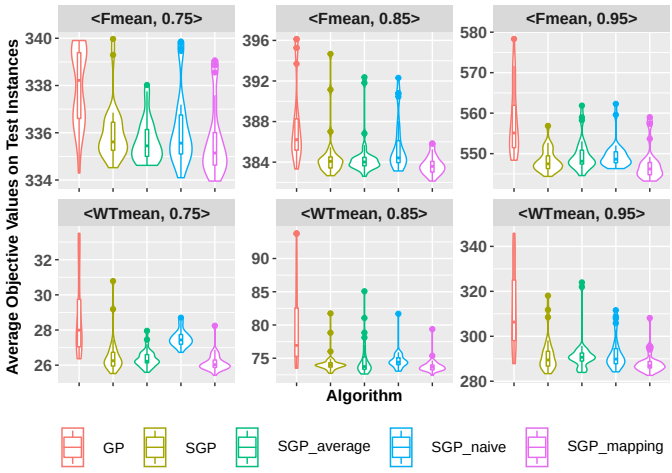


Fig. 9. The violin plots of the average objective values on test instances of GP, SGP, SGP_average, SGP_naive, and SGP_mapping based on 30 independent runs in six scenarios with replication values of 10.

2) Violin Plots of the Evolved Best Scheduling Heuristics: To have a further look at the objectives obtained by the algorithms, we take the experiments with a replication value of 10 as a case study. Fig. 9 shows the violin plot of the average objective values on test instance of GP, SGP, SGP_average, SGP_naive and SGP_mapping in six scenarios. It is clear that the proposed algorithm SGP_mapping shows its superiority with smaller objective values for all the examined scenarios among the involved five algorithms. This indicates that SGP_mapping can detect promising individuals among a large number of individuals, which verifies the effectiveness of the proposed fitness mapping strategy for bringing more samples to the KNN-based surrogate. It is noted that all surrogate-based algorithms (i.e., SGP, SGP_average, SGP_naive and SGP_mapping) perform better than GP (i.e., without surrogate). This indicates the effectiveness of the brood selection to improve the quality of individuals by preselecting a number of promising individuals with cheap cost (i.e., which normally can be ignored).

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3180693

TABLE V
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF GP, SGP, SGP_naive AND SGP_mapping INCLUDING THE COMPARISONS AMONG THEM OVER 30 INDEPENDENT RUNS WITH FOUR DIFFERENT REPLICATION SETTINGS IN SIX SCENARIOS.

Rep	Algorithm	<Fmean, 0.75>	<Fmean, 0.85>	<Fmean, 0.95>	<WTmean, 0.75>	<WTmean, 0.85>	<WTmean, 0.95>
	GP	337.97(1.63)	387.65(3.80)	557.75(8.15)	28.78(2.32)	79.89(6.17)	313.14(18.17)
2	SGP	336.37(1.48)(–)	386.58(4.13)(–)	553.40(7.24)(–)	27.08(1.49)(–)	75.78(4.19)(–)	298.12(12.74)(–)
	SGP_average	336.35(1.51)(–)(≈)	386.00(3.47)(–)(≈)	553.17(4.51)(–)(≈)	27.61(2.01)(–)(≈)	76.36(3.11)(–)(≈)	298.20(9.05)(–)(≈)
	SGP_naive	336.57(1.29)(–)(≈)(≈)	386.96(2.91)(≈)(+)(+)	553.78(5.27)(≈)(≈)(≈)	28.04(2.08)(–)(+)(≈)	76.43(4.55)(–)(≈)(≈)	298.73(8.02)(–)(≈)(≈)
	SGP_mapping	335.70(1.38)(–)(–)(–)(–)	385.91(3.04)(–)(≈)(≈)(–)	552.06(6.43)(–)(≈)(–)(–)	26.52(1.08)(–)(–)(–)(–)	75.61(4.52)(–)(–)(–)(–)	295.06(7.96)(–)(≈)(≈)(–)
		GP	SGP	SGP_average	SGP_naive	SGP_mapping	
	Win / Draw / Lose	6 / 0 / 0	3 / 3 / 0	4 / 2 / 0	6 / 0 / 0	N/A	
3	SGP	336.28(1.59)(–)	385.38(2.12)(–)	552.90(8.15)(–)	27.32(1.81)(–)	75.78(3.36)(–)	292.62(7.35)(–)
	SGP_average	336.49(1.63)(–)(≈)	385.91(3.00)(–)(≈)	552.47(5.82)(–)(≈)	27.11(1.65)(–)(≈)	75.89(3.72)(–)(≈)	294.04(8.17)(–)(≈)
	SGP_naive	337.07(1.51)(–)(+)(+)	385.93(2.72)(–)(≈)(≈)	553.36(5.10)(–)(+)(≈)	27.34(0.77)(–)(+)(+)	76.11(5.19)(–)(≈)(≈)	295.16(6.17)(–)(+)(≈)
	SGP_mapping	335.53(1.57)(–)(–)(–)(–)	384.74(2.62)(–)(–)(–)(–)	552.03(7.41)(–)(≈)(≈)(–)	26.66(1.37)(–)(–)(≈)(–)	74.56(3.04)(–)(–)(–)(–)	292.18(7.51)(–)(≈)(≈)(–)
		GP	SGP	SGP_average	SGP_naive	SGP_mapping	
	Win / Draw / Lose	6 / 0 / 0	4 / 2 / 0	3 / 3 / 0	6 / 0 / 0	N/A	
5	SGP	335.77(1.33)(–)	385.02(2.17)(–)	549.44(3.09)(–)	26.98(1.71)(–)	74.89(2.11)(–)	292.63(8.50)(–)
	SGP_average	336.25(1.51)(–)(≈)	385.05(2.15)(–)(≈)	549.90(3.94)(–)(≈)	26.83(1.37)(–)(≈)	74.77(1.83)(–)(≈)	292.76(9.64)(–)(≈)
	SGP_naive	336.54(1.43)(–)(+)(≈)	385.93(2.93)(–)(≈)(≈)	553.01(5.39)(–)(+)(+)	26.72(0.77)(–)(≈)(≈)	75.78(4.42)(–)(≈)(≈)	292.79(7.81)(–)(≈)(≈)
	SGP_mapping	335.07(0.91)(–)(–)(–)(–)	384.01(1.79)(–)(–)(–)(–)	548.43(6.44)(–)(–)(–)(–)	26.29(0.62)(–)(–)(–)(–)	74.49(3.02)(–)(–)(–)(–)	291.34(11.68)(–)(–)(≈)(–)
		GP	SGP	SGP_average	SGP_naive	SGP_mapping	
	Win / Draw / Lose	6 / 0 / 0	6 / 0 / 0	5 / 1 / 0	6 / 0 / 0	N/A	
10	SGP	335.97(1.18)(–)	384.62(2.46)(–)	548.26(2.73)(–)	26.55(1.06)(–)	74.40(1.75)(–)	292.12(8.09)(–)
	SGP_average	335.63(0.90)(–)(≈)	384.61(2.19)(–)(≈)	549.59(4.01)(–)(≈)	26.38(0.51)(–)(≈)	74.88(2.75)(–)(≈)	292.26(8.88)(–)(≈)
	SGP_naive	336.14(1.60)(–)(≈)(≈)	385.54(2.51)(–)(+)(+)	549.57(3.49)(–)(+)(≈)	27.52(0.46)(≈)(+)(+)	74.72(1.53)(–)(+)(≈)	292.66(7.44)(–)(≈)(≈)
	SGP_mapping	335.66(1.51)(–)(–)(≈)(–)	383.69(0.92)(–)(–)(–)(–)	547.41(4.16)(–)(–)(–)(–)	26.22(0.83)(–)(–)(–)(–)	74.04(2.04)(–)(–)(≈)(–)	288.20(4.92)(–)(–)(–)(–)
		GP	SGP	SGP_average	SGP_naive	SGP_mapping	
	Win / Draw / Lose	6 / 0 / 0	6 / 0 / 0	4 / 2 / 0	6 / 0 / 0	N/A	
		GP	SGP	SGP_average	SGP_naive	SGP_mapping	
Total	Win / Draw / Lose	24 / 0 / 0	19 / 5 / 0	16 / 8 / 0	24 / 0 / 0	N/A	
	Average Rank	4.23	2.67	2.78	3.27	**2.06**	

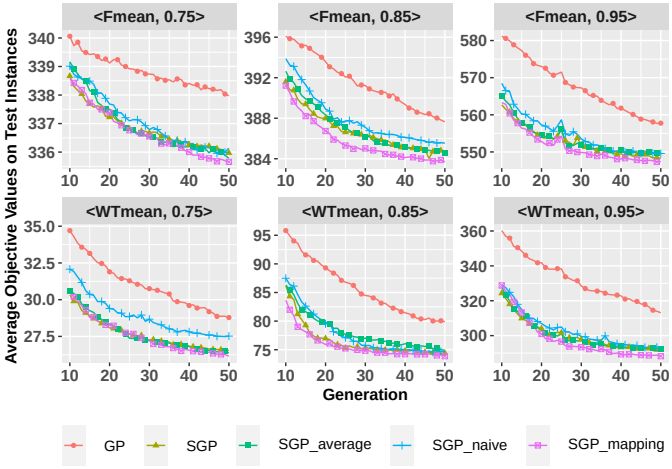


Fig. 10. The curves of the average objective values on test instances of GP, SGP, SGP_average, SGP_naive and SGP_mapping along with generations based on 30 independent runs in six scenarios with replication values of 10.

3) Curves of Average Objective Values on Test Instances: Fig. 10 shows the curves of the average objective values on test instances of GP, SGP, SGP_average, SGP_naive, and SGP_mapping along with generations in six scenarios with the replication value of 10. The results show that SGP_mapping can achieve better scheduling heuristics than GP, SGP, and SGP_naive from an early stage, and keep this advantage during the whole evolutionary process. This further verifies the effectiveness of the algorithm SGP_mapping with the proposed fitness mapping strategy by building a mapping function based on the common individuals between the generations next to each other.

B. The Number of Omitted Individuals by Surrogate Model

In GP with brood recombination, $popsize$ individuals are selected from $popsize * Rep$ newly produced offspring. The key is to find the top $popsize$ individuals. We introduce to use the *number of omitted individuals* by the surrogate model (the top individuals based on real fitness that are missed by the surrogate model) to measure the effectiveness of the surrogate models. A smaller number of the omitted individuals indicates better effectiveness of surrogate. Specifically, we record the number of omitted individuals of SGP and SGP_mapping by looking at the top 30% and 100% (i.e., the entire population) ordered individuals with estimated fitness. The reason to look at the top 30% individuals is that our preliminary study shows that the parent selection (i.e., tournament selection with size 5) typically selects the top 30% individuals as parents (i.e., the top 30% individual have a big impact on the performance of the algorithm). We use SGP(30%), SGP_mapping(30%), SGP(100%) and SGP_mapping(100%) as the abbreviations to represent these observations of the algorithms for the convenience of comparison.

Fig. 11 shows the curves of the number of omitted individuals by the surrogate at each generation in six scenarios. There are three main observations based on the results. First,

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3180693

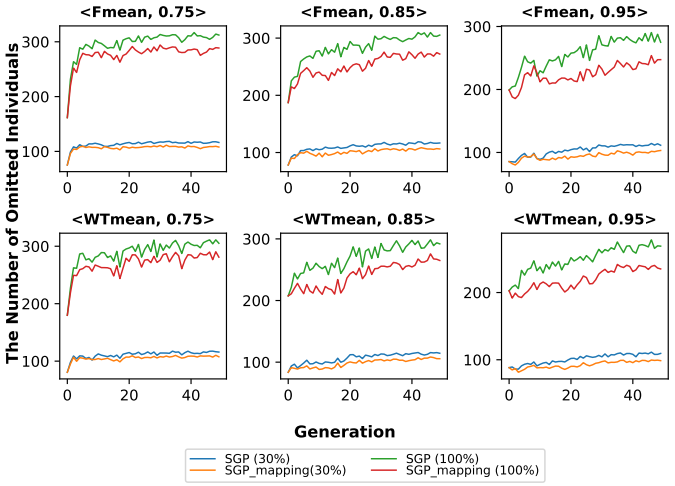


Fig. 11. The curves of the number of omitted individuals of SGP and SGP_mapping based on 30 independent runs in six scenarios with replication values of 10.

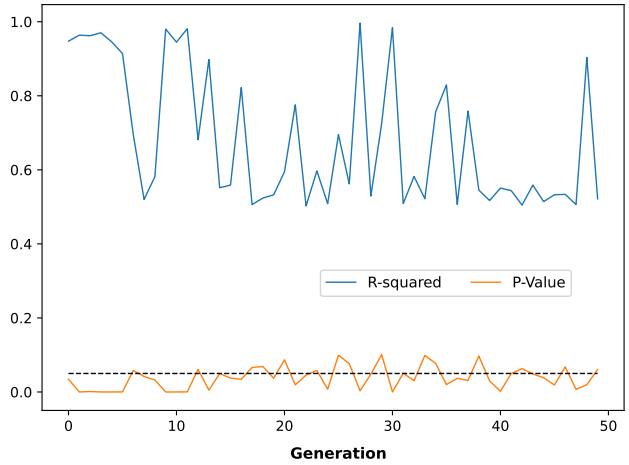


Fig. 12. The curves of R-squared and significance of the learned linear model of SGP_mapping in the scenario <WTmean, 0.95> of one run with a replication value of 10.

the accuracy of fitness estimation of SGP_mapping is higher (with a smaller number of omitted individuals) than that of SGP_naive in terms of both 30% or 100% individuals, which indicates the effectiveness of the proposed algorithm SGP_mapping. Second, the number of omitted individuals observed based on all individuals is much larger than that of based on top 30% individuals. This is consistent with our intuition that it is more challenging to select good individuals from a larger number of individuals. This finding is also consistent with the findings in Section V-A1. Last, the number of omitted individuals increases for both SGP and SGP_mapping along with generations. One possible reason is that the individuals become better and better over generations, and they normally have similar or the same behaviours which makes the samples in the KNN shrink to a small region to some extent. This might mislead the fitness estimations of individuals.

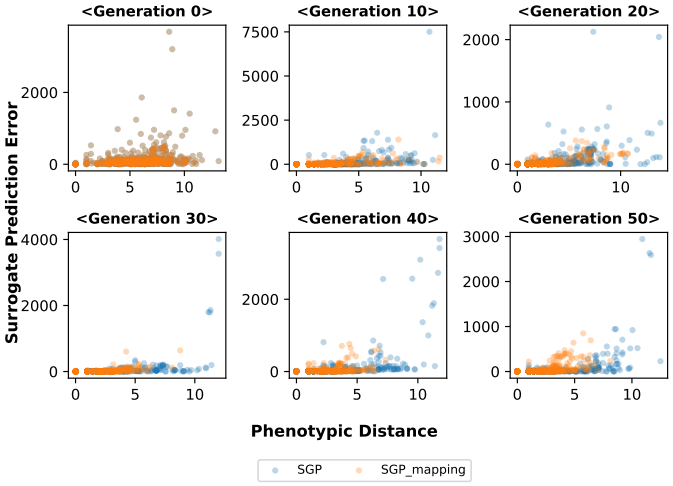


Fig. 13. The scatter plots of the distance and the gap between real and estimated fitness of SGP_mapping in the scenario <WTmean, 0.95> of one run with a replication value of 10.

C. Effectiveness of Learned Fitness Mapping Function

We use the R-Squared measure to analyse the effectiveness of the fitness mapping function. R-Squared is a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variables(s) in a regression model [67]. If the R^2 of a model is 0.5, then approximately half of the observed variation can be explained by the model. In addition, significance (P-Value) is the probability of rejecting the null hypothesis (i.e., saying that there is no correlation between the independent and the dependent variable) when it is true. A significance that is smaller than the significance level (usually 0.05) indicates the model fits the data well. Fig. 12 shows the curves of R-Squared and significance values in the scenario <WTmean, 0.95> of one run of SGP_mapping with a replication value of 10. The results show that the built linear model can match the fitness of individuals at the previous generation to the current generation well with a R-Squared value that is larger than 0.6 at most generations. In addition, the significance values are smaller than 0.05 (dashed line) at most generations, which shows a linear relationship between the fitness of common individuals at consecutive generations. This verifies the effectiveness of the proposed algorithm from the perspective of the core of building a surrogate model. In addition, it is not surprising that there are fluctuations of R-Squared values, since the degrees of similarities between consecutive training instances vary.

D. Phenotypic Distance Versus Surrogate Prediction Error

It is reasonable to believe that the surrogate prediction error depends on the phenotypic distance (i.e., we use Euclidean distance in this paper) between the predicted individual and its nearest neighbour in the surrogate. We expect that SGP_mapping can find the nearest neighbour with a smaller phenotypic distance, and thus obtain a smaller prediction error.

To verify this, Fig. 13 shows the scatter plots of the phenotypic distance and the surrogate prediction error, i.e., the gap between real and estimated fitness, at six different generations of SGP_mapping in the scenario <WTmean, 0.95> of one

This article has been accepted for publication in IEEE Transactions on Evolutionary Computation. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2022.3180693

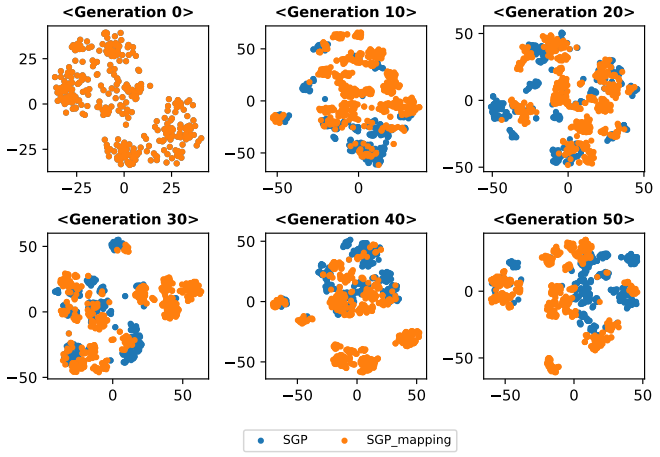


Fig. 14. Visualisations of the samples in the surrogate models of SGP and SGP_mapping based on phenotypic characterisations of one run in the scenario <WTmean, 0.95> with a replication value of 10.

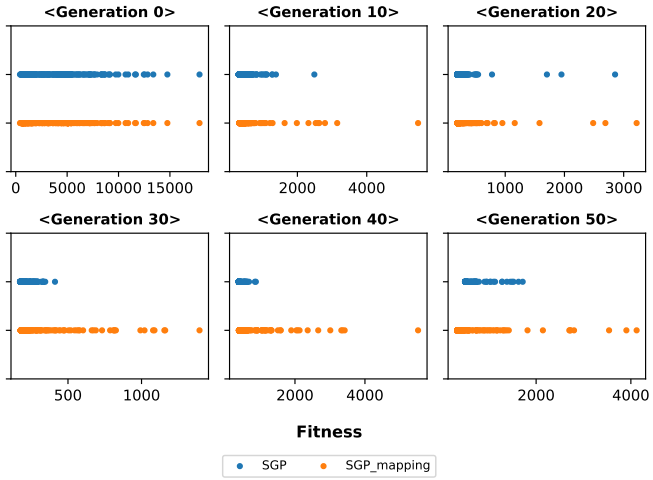


Fig. 15. Visualisations of the samples in the surrogate models of SGP and SGP_mapping based on fitness of one run in the scenario <WTmean, 0.95> with a replication value of 10.

run with a replication value of 10. We select the investigated generations in every 10 generations (i.e., generation 0, 10, 20, 30, 40 and 50). At the beginning (i.e., generation 0), the scatter points of SGP and SGP_mapping are exactly the same, since the samples of the surrogate of SGP and SGP_mapping are identical (i.e., only has the samples extracted from generation 0). As the evolution processes, we can see that SGP_mapping can find closer samples/individuals in the KNN-based surrogate model than SGP. For example, at generation 40, the largest distance of the estimated individual with the samples in KNN-based surrogate of SGP_mapping is around 7, which is about 12 of SGP. This verifies the effectiveness of SGP_mapping in terms of the mechanism of the KNN-based surrogate model with more samples. In other words, in general, a fitness estimation with a smaller distance is more reliable. This is also the reason why SGP_mapping can outperform the other algorithms.

E. Distribution of Samples in Surrogate

The samples in the surrogate models of SGP_mapping are expected to cover a bigger area in the phenotypic space, since more samples are included in the surrogate models. This is the key factor to the success of SGP_mapping. We choose the samples of surrogate of SGP_mapping in the scenario <WTmean, 0.95> of one run with a replication value of 10 to investigate the sample distribution in this section.

1) From the Perspective of Phenotypic Characterisation: The phenotypic characterisation represents the behaviour of the individual, which is a 40-dimensional vector in this paper. To visualise the phenotypic characterisations of the samples in the surrogate models, we use t-SNE to reduce the dimensions of the phenotypic characterisations. Fig. 14 shows the visualisations of phenotypic characterisations of the samples in KNN-based surrogate models of SGP and SGP_mapping at generation 0, 10, 20, 30, 40 and 50. At generation 0, the samples in the surrogate models of SGP and SGP_mapping only contain the extracted samples from generation 0, and they are identical. For later generations, we can see that the samples in the surrogate models of SGP_mapping cover a larger area than SGP. With a larger range of the samples, SGP_mapping have a higher chance to find closer samples to the estimated individuals and get a better estimation of the fitness of the individual. This verifies the effectiveness of the proposed algorithm SGP_mapping from the perspective of sample distribution in terms of phenotypic characterisation.

2) From the Perspective of Fitness: The fitness distribution of samples in the surrogate can also provide the information of the covered areas of the surrogate. A good surrogate is expected to have a wide range of fitness. Fig. 15 shows the visualisations of fitness of samples in surrogate models of SGP and SGP_mapping at generation 0, 10, 20, 30, 40 and 50 of one run in the scenario <WTmean, 0.95> with a replication value of 10. The same as the visualisation of surrogate samples in terms of phenotypic characterisation, the fitness distributions at generation 0 are identical. However, the fitness distributions of SGP_mapping show a larger range than that of SGP at all other generations. This verifies the effectiveness of the proposed algorithm SGP_mapping from the perspective of sample distribution in terms of fitness.

VI. Further Analysis

A. Comparison with Real Fitness-Evaluation Preselection

In the ideal case, if each offspring from brood recombination is evaluated with the real fitness evaluation for preselection, the fitness estimation is 100% accurate. To understand how close our fitness mapping surrogate is from the real fitness evaluation, we compare the performance of SGP_mapping and the algorithm using real fitness evaluation to do preselection, which is named SGP_real.

Table VI shows the mean and standard deviation of the test performance of SGP_mapping and SGP_real according to 30 independent runs with four replication settings, each with six scenarios. The results show that SGP_mapping can achieve similar performance with SGP_real with small replication

TABLE VI
THE MEAN (STANDARD DEVIATION) OF THE OBJECTIVE VALUES ON TEST INSTANCES OF SGP_mapping AND SGP_real OVER 30 INDEPENDENT RUNS WITH FOUR DIFFERENT REPLICATION SETTINGS IN SIX SCENARIOS.

Rep	Scenario	SGP_mapping	SGP_real
2	<Fmean, 0.75>	335.70(1.38)	335.59(1.82)(≈)
	<Fmean, 0.85>	385.91(3.04)	385.71(2.82)(≈)
	<Fmean, 0.95>	552.06(6.43)	551.82(5.95)(≈)
	<WTmean, 0.75>	26.52(1.08)	26.98(1.29)(≈)
	<WTmean, 0.85>	75.61(4.52)	75.52(5.41)(≈)
	<WTmean, 0.95>	295.06(7.96)	291.91(6.15)(≈)
3	<Fmean, 0.75>	335.53(1.57)	334.83(1.28)(–)
	<Fmean, 0.85>	384.74(2.62)	384.36(1.57)(≈)
	<Fmean, 0.95>	552.03(7.41)	549.77(5.53)(≈)
	<WTmean, 0.75>	26.66(1.37)	26.31(0.37)(≈)
	<WTmean, 0.85>	74.56(3.04)	73.25(0.91)(–)
	<WTmean, 0.95>	292.18(7.51)	291.01(6.91)(≈)
5	<Fmean, 0.75>	335.07(0.91)	334.68(1.00)(–)
	<Fmean, 0.85>	384.01(1.79)	383.90(1.52)(≈)
	<Fmean, 0.95>	548.43(6.44)	545.84(5.63)(–)
	<WTmean, 0.75>	26.29(0.62)	25.94(0.28)(–)
	<WTmean, 0.85>	74.49(3.02)	73.83(0.51)(≈)
	<WTmean, 0.95>	291.34(11.68)	287.34(5.67)(–)
10	<Fmean, 0.75>	335.11(1.51)	334.89(1.02)(–)
	<Fmean, 0.85>	383.69(0.92)	383.22(0.67)(–)
	<Fmean, 0.95>	547.41(4.16)	545.39(2.65)(–)
	<WTmean, 0.75>	26.22(0.83)	25.93(0.35)(–)
	<WTmean, 0.85>	74.04(2.04)	73.47(0.67)(–)
	<WTmean, 0.95>	288.20(4.92)	286.83(5.49)(–)

settings, however, this is not the case when the replication value becomes large. To be specific, SGP_mapping obtains similar performance with SGP_real in all six scenarios with replication values of 2, and performs similar with SGP_real in four out of six scenarios with replication values of 3. However, SGP_mapping only gets similar performance with SGP_real in two out of six scenarios with replication values of 5, and performs worse than SGP_real in all the scenarios with replication values of 10. We can see that there is still potential for improvement and we will consider it as future work. On the other hand, we can see that the existing KNN-based surrogate models are even further away from the ideal. The proposed algorithm is better than the existing KNN-based surrogate algorithms.

B. Training Time

Table VII shows the mean and standard deviation of the training time of GP, SGP, SGP_average, SGP_naive, SGP_mapping and SGP_real in minutes over 30 independent runs in six scenarios with replication values of 2. The results show that there is no significant difference between the training time of GP and the GP algorithms with surrogate-assisted preselection (i.e., SGP, SGP_average, SGP_naive, SGP_mapping). In other words, the GP algorithms with surrogate-assisted preselection can achieve better performance than GP without requiring an extra computational cost. Combining with the results with replication value of 2 in Table VI, we can find that the proposed algorithm SGP_mapping can reduce the training time to half to achieve similar performance as the one using real fitness evaluation for preselection. In terms of the training time, we also find the same pattern in

TABLE VII
THE MEAN (STANDARD DEVIATION) OF TRAINING TIME (IN MINUTES) OF GP, SGP, SGP_average, SGP_naive, SGP_mapping AND SGP_real OVER 30 INDEPENDENT RUNS IN SIX SCENARIOS WITH REPLICATION OF 2.

Scenarios	GP	SGP	SGP_ average	SGP_ naive	SGP_ mapping	SGP_ real
<Fmean, 0.75>	50(11)	52(12)(≈)	53(10)(≈)	53(9)(≈)	51(7)(≈)	101(18)(+)
<Fmean, 0.85>	51(7)	51(10)(≈)	52(8)(≈)	52(7)(≈)	51(8)(≈)	103(14)(+)
<Fmean, 0.95>	53(7)	54(6)(≈)	53(8)(≈)	53(6)(≈)	54(8)(≈)	122(24)(+)
<WTmean, 0.75>	52(7)	53(8)(≈)	54(10)(≈)	54(11)(≈)	53(7)(≈)	107(26)(+)
<WTmean, 0.85>	54(7)	55(9)(≈)	55(11)(≈)	56(12)(≈)	55(6)(≈)	114(24)(+)
<WTmean, 0.95>	55(10)	55(8)(≈)	56(10)(≈)	56(9)(≈)	56(7)(≈)	115(12)(+)

the experiments with replication values of 3, 5, and 10. Due to the page limit, we do not show the results here.

VII. CONCLUSIONS

The goal of this paper was to develop an effective genetic programming algorithm with surrogate-assisted preselection for dynamic job shop scheduling. The goal has been successfully achieved by proposing an effective fitness mapping strategy to include the individual samples from multiple different generations to enlarge the number of samples of the KNN-based surrogate.

The results showed that the proposed SGP_mapping can achieve significantly better scheduling heuristics for all the examined dynamic job shop scenarios. The effectiveness of the proposed fitness mapping strategy was also examined by comparing the accuracy of fitness estimation, the effect of used samples in KNN surrogate on the gap of real and estimated fitness of individuals, and the characterisations of built fitness mapping functions. It has been observed that the proposed SGP_mapping manages to enhance its performance due to the improvement of the surrogate technique with the proposed fitness mapping strategy. The samples in the KNN-based surrogate models are further analysed from the perspective of their phenotypic characterisations and fitness, respectively. The results show that the improved samples can cover a more comprehensive area in the phenotypic space by adding more samples from multiple generations.

The study in this paper can significantly contribute to the development of the evolutionary computation community. First, the proposed algorithm broadens the study of using data obtained from different instances. Specifically, it proposes a fitness mapping strategy based on common individuals to make it effective to use different data in a comparable standard. Second, the proposed algorithm promotes the studies of surrogates on discrete optimisation problems. Last, the way of extracting useful information from the data with different training instances in dynamic job shop scheduling can provide guidance for extracting more training data for the problems which have limited training data to learn from.

Some interesting directions can be further studied in future. We would like to investigate how to use all the information from the previous generations to enhance the effectiveness of KNN-based surrogates. We will investigate more effective ways to map the information from different training instances to a comparable standard. Advanced surrogate models will be

designed with a goal to achieve the same performance as using the real evaluation in a wide range of scenarios. In addition, we plan to investigate the proposed algorithms on other combinatorial optimisation problems such as vehicle routing and cloud computing. The effectiveness of the proposed idea on other types of GP will be also investigated.

REFERENCES

[1] S. Winkelhaus, E. H. Grosse, and S. Morana, "Towards a conceptualisation of order picking 4.0," *Computers & Industrial Engineering*, vol. 159, p. 107511, 2021.

[2] B. Illés, P. Tamás, P. Dobos, and R. Skapinyecz, "New challenges for quality assurance of manufacturing processes in industry 4.0," in *Solid State Phenomena*, vol. 261. Trans Tech Publ, 2017, pp. 481–486.

[3] S. Bera, S. Misra, and J. J. Rodrigues, "Cloud computing applications for smart grid: A survey," *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 5, pp. 1477–1494, 2014.

[4] R. Ramasesh, "Dynamic job shop scheduling: a survey of simulation research," *Omega*, vol. 18, no. 1, pp. 43–57, 1990.

[5] 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.

[6] X.-N. Shen and X. Yao, "Mathematical modeling and multi-objective evolutionary algorithms applied to dynamic flexible job shop scheduling problems," *Information Sciences*, vol. 298, pp. 198–224, 2015.

[7] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming based generative hyper-heuristics: A case study in dynamic scheduling," *IEEE Transactions on Cybernetics*, 2021, Doi: 10.1109/TCYB.2021.3065340.

[8] A. Baykasoğlu, M. Göçken, and L. Özbakir, "Genetic programming based data mining approach to dispatching rule selection in a simulated job shop," *Simulation*, vol. 86, no. 12, pp. 715–728, 2010.

[9] W. B. Langdon and R. Poli, *Foundations of genetic programming*. Springer Science & Business Media, 2013.

[10] F. Zhang, S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: An evolutionary learning approach," in *Machine Learning: Foundations, Methodologies, and Applications*. Springer, 2021, DOI: 10.1007/978-981-16-4859-5, pp. XXXIII+338 pages.

[11] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2013.

[12] E. Hart and K. Sim, "A hyper-heuristic ensemble method for static job-shop scheduling," *Evolutionary Computation*, vol. 24, no. 4, pp. 609–635, 2016.

[13] W. A. Tackett, "Recombination, selection, and the genetic construction of computer programs," Ph.D. dissertation, University of Southern California Los Angeles, 1994.

[14] W. A. Tackett and A. Carmi, "The unique implications of brood selection for genetic programming," in *Proceedings of the IEEE Conference on Evolutionary Computations*. IEEE, 1994, pp. 160–165.

[15] M. Zhang, X. Gao, W. Lou, and D. Qian, "Investigation of brood size in gp with brood recombination crossover for object recognition," in *Proceedings of the Pacific Rim International Conference on Artificial Intelligence*. Springer, 2006, pp. 923–928.

[16] 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 Conference on Genetic and Evolutionary Computation*. ACM, 2010, pp. 257–264.

[17] 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.

[18] S. Nguyen, Y. Mei, B. Xue, and M. Zhang, "A hybrid genetic programming algorithm for automated design of dispatching rules," *Evolutionary Computation*, pp. 1–31, 2018.

[19] F. Zhang, Y. Mei, S. Nguyen, K. C. Tan, and M. Zhang, "Multitask genetic programming-based generative hyper-heuristics: A case study in dynamic scheduling," *IEEE Transactions on Cybernetics*, 2021, Doi: 10.1109/TCYB.2021.3065340.

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

[21] Q. Zhou, J. Wu, T. Xue, and P. Jin, "A two-stage adaptive multi-fidelity surrogate model-assisted multi-objective genetic algorithm for computationally expensive problems," *Engineering with Computers*, vol. 37, no. 1, pp. 623–639, 2021.

[22] D. Lim, Y. Jin, Y.-S. Ong, and B. Sendhoff, "Generalizing surrogate-assisted evolutionary computation," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 3, pp. 329–355, 2009.

[23] X. Cai, L. Gao, and X. Li, "Efficient generalized surrogate-assisted evolutionary algorithm for high-dimensional expensive problems," *IEEE Transactions on Evolutionary Computation*, vol. 24, no. 2, pp. 365–379, 2019.

[24] H. Wang, L. Feng, Y. Jin, and J. Doherty, "Surrogate-assisted evolutionary multitasking for expensive minimax optimization in multiple scenarios," *IEEE Computational Intelligence Magazine*, vol. 16, no. 1, pp. 34–48, 2021.

[25] 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, Doi: 10.1109/TETC.2019.2945775.

[26] L. Liu, Y. Cheng, and X. Wang, "Genetic algorithm optimized taylor kriging surrogate model for system reliability analysis of soil slopes," *Landslides*, vol. 14, no. 2, pp. 535–546, 2017.

[27] H. Yu, Y. Tan, J. Zeng, C. Sun, and Y. Jin, "Surrogate-assisted hierarchical particle swarm optimization," *Information Sciences*, vol. 454, pp. 59–72, 2018.

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

[29] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted evolutionary multitask genetic programming for dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, vol. 25, no. 4, pp. 651–665, 2021.

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

[31] 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.

[32] M. Pinedo, *Scheduling*. Springer, 2012, vol. 29.

[33] M. Dumić and D. Jakobović, "Ensembles of priority rules for resource constrained project scheduling problem," *Applied Soft Computing*, p. 107606, 2021.

[34] S. Nguyen, D. Thiruvady, M. Zhang, and D. Alahakoon, "Automated design of multipass heuristics for resource-constrained job scheduling with self-competitive genetic programming," *IEEE transactions on cybernetics*, 2021, DOI: 10.1109/TCYB.2021.3062799.

[35] 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.

[36] M. M. Van Hulle, "A goal programming network for mixed integer linear programming: a case study for the job-shop scheduling problem," *International Journal of Neural Systems*, vol. 2, no. 03, pp. 201–209, 1991.

[37] 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.

[38] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European Journal of Operational Research*, vol. 167, no. 1, pp. 77–95, 2005.

[39] D. Sha and C.-Y. Hsu, "A hybrid particle swarm optimization for job shop scheduling problem," *Computers & Industrial Engineering*, vol. 51, no. 4, pp. 791–808, 2006.

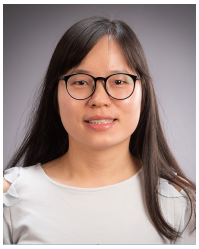
[40] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Proceedings of the Computational intelligence*. Springer, 2009, pp. 177–201.

[41] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[42] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of metaheuristics*. Springer, 2003, pp. 457–474.

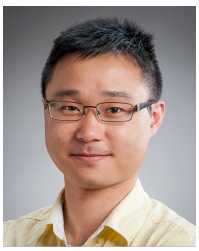
[43] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of metaheuristics*. Springer, 2010, pp. 449–468.

[44] J. Lin, "Backtracking search based hyper-heuristic for the flexible job-shop scheduling problem with fuzzy processing time," *Engineering Applications of Artificial Intelligence*, vol. 77, pp. 186–196, 2019.
[45] M. Durasević and D. Jakobović, "Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment," *Applied Soft Computing*, vol. 96, p. 106637, 2020.
[46] 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.
[47] 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.
[48] S. Nguyen, "Automatic design of dispatching rules for job shop scheduling with genetic programming," 2013.
[49] Y. Zhou, 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.
[50] S. Wang, Y. Mei, M. Zhang, and X. Yao, "Genetic programming with niching for uncertain capacitated arc routing problem," *IEEE Transactions on Evolutionary Computation*, 2021, Doi: 10.1109/TEVC.2021.3095261.
[51] Y. Jin, H. Wang, and C. Sun, "Data-driven evolutionary optimization," 2021.
[52] M. H. Hassoun *et al.*, *Fundamentals of artificial neural networks*. MIT press, 1995.
[53] R. G. Regis, "Evolutionary programming for high-dimensional constrained expensive black-box optimization using radial basis functions," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 3, pp. 326–347, 2013.
[54] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*. Springer, 2003, pp. 63–71.
[55] A. Kattan and Y.-S. Ong, "Surrogate genetic programming: A semantic aware evolutionary search," *Information Sciences*, vol. 296, pp. 345–359, 2015.
[56] M. Pilát and R. Neruda, "Feature extraction for surrogate models in genetic programming," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2016, pp. 335–344.
[57] 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.
[58] 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.
[59] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative multi-fidelity based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, 2021. Doi: 10.1109/TCYB.2021.3050141.
[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] 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*, vol. 51, no. 4, pp. 1797–1811, 2021.
[62] 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.
[63] 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.
[64] 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.
[65] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Correlation coefficient based recombinative guidance for genetic programming hyper-heuristics in dynamic flexible job shop scheduling," *IEEE Transactions on Evolutionary Computation*, 2021. Doi: 10.1109/TEVC.2021.3056143.
[66] J. R. Koza and R. Poli, "Genetic programming," in *Search Methodologies*. Springer, 2005, pp. 127–164.
[67] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 329.



Fangfang Zhang (Member, IEEE) received the B.Sc. and M.Sc. degrees from Shenzhen University, Shenzhen, China, and the Ph.D. degree in Computer Science from Victoria University of Wellington, New Zealand, in 2014 and 2017, and 2021, respectively.

She is currently a Research Fellow in Artificial Intelligence with the School of Engineering and Computer Science, Victoria University of Wellington. She has over 30 papers in refereed international journals and conferences. Her current research interests include evolutionary computation, hyper-heuristic learning, job shop scheduling, surrogate, and multitask learning.

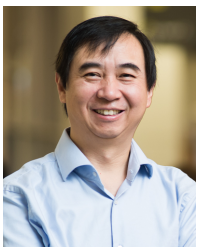


Yi Mei (M'09-SM'18) received the B.Sc. and Ph.D. degrees from the University of Science and Technology of China, Hefei, China, in 2005 and 2010, respectively.

He is 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. His research interests include evolutionary scheduling and combinatorial optimisation, machine learning, genetic programming, and hyper-heuristics.

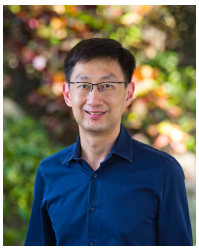
Su Nguyen (M'13) received his Ph.D. degree in Artificial Intelligence from Victoria University of Wellington, New Zealand, in 2013.

He is a Senior Lecturer and an Algorithm Lead with the Centre for Data Analytics and Cognition, La Trobe University, Melbourne, VIC, Australia. He has over 70 peer-reviewed papers. 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 advanced machine-learning algorithms.



Mengjie Zhang (M'04-SM'10-F'19) received the B.E. and M.E. degrees from Artificial Intelligence Research Center, 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, Victoria University of Wellington, New Zealand. She has published over 500 research papers in refereed international journals and conferences. He is a Fellow of the Royal Society, and Fellow of Engineering of New Zealand, a Fellow of IEEE and an IEEE Distinguished Lecturer.



Kay Chen Tan (SM'08-F'14) received the B.Eng. (First Class Hons.) degree in electronics and electrical engineering and the Ph.D. degree from the University of Glasgow, Glasgow, U.K., in 1994 and 1997, respectively.

He is currently a Chair Professor with the Department of Computing at the Hong Kong Polytechnic University. He has published over 300 refereed articles and 7 books. He is an IEEE Fellow, an Honorary Professor at University of Nottingham in UK, and the Chief Co-Editor of Springer Book Series on Machine Learning: Foundations, Methodologies, and Applications.