An Efficient Feature Selection Algorithm for Evolving Job Shop Scheduling Rules with Genetic Programming

Yi Mei, *Member, IEEE*, Su Nguyen, *Member, IEEE*, Bing Xue, *Member, IEEE* and Mengjie Zhang, *Senior Member, IEEE*

***Abstract*—Automated design of job shop scheduling rules using genetic programming as a hyper-heuristic is an emerging topic that has become more and more popular in recent years. For evolving dispatching rules, feature selection is an important issue for deciding the terminal set of genetic programming. There can be a large number of features, whose importance/relevance varies from one to another. It has been shown that using a promising feature subset can lead to a significant improvement over using all the features. However, the existing feature selection algorithm for job shop scheduling is too slow and inapplicable in practice. In this paper, we propose the first "practical" feature selection algorithm for job shop scheduling. Our contributions are twofold. First, we develop a niching-based search framework for extracting a diverse set of good rules. Second, we reduce the complexity of fitness evaluation by using a surrogate model. As a result, the proposed feature selection algorithm is very efficient. The experimental studies show that it takes less than 10% of the training time of the standard genetic programming training process, and can obtain much better feature subsets than the entire feature set. Furthermore, it can find better feature subsets than the best-so-far feature subset.**

***Index Terms*—Job shop scheduling, genetic programming, feature selection, hyper-heuristic**

I. Introduction

Job Shop Scheduling (JSS) [43] is an important problem in the scheduling and artificial intelligence fields. It has many applications in the real world such as manufacturing [7], [31], project scheduling [55] and resource allocation in cloud computing, and has attracted many research interests from both researchers and practitioners. In JSS, a number of *jobs* and *machines* are given, each job contains a sequence of *operations*, each of which has an eligible machine to process it. Then, the aim is to design a schedule to process the operations with the machines subject to the predefined constraints (e.g. the operations are processed in order by their eligible machines) to achieve the desired goals such as minimising the makespan, flowtime and tardiness.

In practice, the environment is usually *dynamic*, and jobs can arrive in real time with no prior information (e.g. its arrival time, due date and processing time). Traditional optimisation approaches such as mathematical programming and meta-heuristic methods can hardly tackle dynamic JSS (DJSS) well, since they cannot immediately react to the occurrences of unpredicted events such as new job arrivals and machine breakdowns. On the contrary, *dispatching rules* can respond to the environment change instantaneously. Therefore, dispatching rules have been widely adopted for solving DJSS [2], [14]. Briefly speaking, a dispatching rule consists of a priority function considering the features of the current job shop state and the operation to be processed. At each decision point when a machine becomes idle and there are operations waiting in its queue, the dispatching rule calculates the priority value for each waiting operation, and selects the operation with the highest priority value to be processed next. So far, there have been a large number of dispatching rules designed manually to tackle different job shop scenarios and objectives. A comprehensive comparison between manual dispatching rules can be found in [50].

It has been shown in literature (e.g. [25], [46], [51]) that the manually designed dispatching rules are still not satisfactory enough. Due to the complex interactions involving different waiting operations and job shop states, it is very hard, if not impossible, to manually capture all the underlying relationships to design a promising dispatching rule. Recently, Genetic Programming (GP) has attracted more and more research interests for automatically designing dispatching rules. There have been plenty of works proposed for evolving dispatching rules with GP (e.g. [10], [16], [21], [36], [42]), which successfully obtained much better rules than the manually designed rules.

A challenge for evolving dispatching rules using GP is the huge search space due to the variable-length representation. For example, as indicated in [48], if using the tree-based representation with the maximal depth of D and all the functions are binary, i.e. they take two arguments, then the size of GP search space is $|\mathcal{F}|^{2^D-1} \cdot |\mathcal{T}|^{2^D}$, where $\mathcal{F}$ and $\mathcal{T}$ stand for the function and terminal sets. It can be seen that the size of GP search space depends on the maximal depth of the tree, function set and terminal set. Therefore, in order to reduce the GP search space and improve efficiency, one can decrease the maximal tree depth, the number of function nodes and the number of terminals. In this paper, we will particularly focus on reducing the number of terminals by feature selection.

For GP-evolved dispatching rules, the candidate terminals include the global shop-level features (e.g. the current time), job-related features (e.g. the processing time and due date) and machine-related features (e.g. the machine ready time and work in the queue). In addition to the instantaneous

Yi Mei, Su Nguyen, Bing Xue and Mengjie Zhang are with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6012, New Zealand. e-mail: {yi.mei, su.nguyen, bing.xue, mengjie.zhang}@ecs.vuw.ac.nz. Su Nguyen is also with Department of Business, Hoa Sen University, Vietnam.

information, one can also look ahead to the future (e.g. the work in the next queue) and back to the past (e.g. the historical utilisation level and average waiting time of the previous 5 jobs) [20]. Obviously, not all these features are (strongly) relevant to the performance of rule, and the contributions of the features vary from one objective to another (e.g. minimising makespan and minimising total weighted tardiness require different features). It has been demonstrated that including irrelevant features can lead to negative effect in evolving GP rules [16]. To reduce the search space without losing promising regions, it is important to select a proper set of relevant features as the terminal set of GP. However, it is a challenging task due to the complex interactions between the features.

Intuitively, an important feature tends to make a *considerable contribution* to *good* dispatching rules. In most previous works, the contribution of a feature is estimated by the frequency it occurs in GP individuals with high fitness values [9], [40], [47] during the search process. However, such an estimation has two major issues. First, the estimation for each GP individual may not be accurate due to the redundant feature occurrences such as x/x and $x - x$. Second, the estimation may be biased to the individuals in the particular local optimal region that GP converges to. In this case, the features that contribute to other local optima but not in the current population will be missed.

To address the above issues, we proposed a new feature selection approach in [34]. To address the first issue, we proposed a more accurate and domain-knowledge-free measure based on a *feature removal* operation, and empirically verified its effectiveness in several job shop scheduling scenarios [34]. To address the second issue, we selected the best individuals from 30 independent runs of GP to reduce the estimation bias. However, the proposed feature selection approach is still not to applicable in practice, mainly because of its low efficiency. Specifically, a diverse set of good individuals is required to achieve a high accuracy of the feature selection. There is no efficient way for obtaining such a set of individuals so far. In [34], the 30 individuals were obtained from 30 independent runs of GP, requiring computational time of $30 \cdot Time(GP)$, where $Time(GP)$ is the computational time of a single GP run.

A. Goals

This paper aims to address the above issues and propose a better feature selection algorithm with higher efficiency and accuracy. It has the following research objectives:

1) To propose a more efficient method to obtain a diverse set of good individuals by employing niching and surrogate model.
2) To develop a new feature selection scheme that takes both the fitness of the individuals and the feature contribution to the individuals into account.
3) To verify the efficacy of the proposed feature selection algorithm in terms of number of selected features and the quality of the selected feature set.
4) To analyse the effect of the selection of each feature on the capability of GP in evolving promising dispatching rules.

B. Organisation

The rest of the paper is organised as follows: Section II gives the background introduction, including the problem description, automated dispatching rule design and feature selection. Then, the proposed feature selection algorithm (NiSuFS) is described in Section III. Experimental studies are carried out in Section IV to investigate the behaviour and performance of NiSuFS. Further analysis is conducted in Section V. Finally, Section VI gives the conclusions and future work.

II. Background

A. Job Shop Scheduling

A Job Shop Scheduling (JSS) problem aims to process a number of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ with a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j has a sequence of l_j ($l_j \leq m$) operations $\mathbf{O}_j = (O_{1j}, \ldots, O_{l_j,j})$. It has a release time $t_0(J_j)$ and a due date $\rho(J_j)$. Each operation O_{ij} can only be processed by machine $\pi(O_{ij}) \in \mathcal{M}$, and its processing time is $\delta(O_{ij})$. Then JSS is to find the best schedule subject to the following constraints:

1) For each job, $O_{i+1,j}$ cannot be processed before the completion of its preceding operations O_{ij}, $\forall\ j = 1, \ldots, n$. The first operation O_{1j} cannot be processed until the job is release at time $t_0(J_j)$.
2) Operation O_{ij} must be processed on machine $\pi(O_{ij})$.
3) Each machine can only process one operation at the same time.
4) The scheduling is non-preemptive, i.e. the processing of an operation cannot be disrupted or stopped until it is completed.

The commonly considered JSS objectives include minimising the makespan ($C_{\max}$), total flowtime ($\sum C_j$), total weighted tardiness ($\sum w_j T_j$), number of tardy jobs, etc [43].

B. Automated Design of Dispatching Rules

Dispatching rules are commonly used in DJSS due to its flexibility, scalability and ease of implementation. It generates the schedule in real time and can adapt to the environment change (e.g. new job arrivals) well. At each decision point, at least one machine is idle and the queue of that machine is not empty. Then, the dispatching rule uses some *priority function* to select the next operation among those waiting in the queue. The priority function assigns a priority value for each candidate operation, and the one with the highest priority value is selected. For example, in the Shortest Processing Time (SPT) rule, the priority function is defined as $-\delta(O_{ij})$.

Dispatching rules can be divided into two categories according to their definitions of the decision points. The first category is called the *non-delay* rules. In the non-delay rules, no delay is allowed as long as a machine is idle and its waiting queue is not empty, i.e. an operation has to be selected and processed immediately. On the other hand, the *active* rules allow some delay in case that a new urgent job will arrive soon. In this paper, we will focus on the non-delay category only, since it is simpler to use and have achieved much success in previous studies (e.g. [16], [36]).

A great amount of effort has been made to manually design more effective dispatching rules (e.g. [17], [23], [45]). However, due to the subtle interactions involving various factors in affecting the performance of the rule, the performance of existing manually designed rules are not satisfactory enough. Moreover, the best dispatching rule depends on the shop scenario and objective to be optimised. Therefore, it is difficult to manually obtain an effective dispatching rule for the given JSS. In this situation, hyper-heuristics [5] are promising approaches to search for dispatching rules automatically.

Hyper-heuristics optimise heuristics rather than solutions by searching in the heuristic space. Genetic Programming (GP) [26] has been demonstrated to be a powerful hyper-heuristic for DJSS ([10], [16], [20], [22], [36], [37], [42], [52]). For JSS, it is shown to outperform other methods such as neural network and linear combination [3]. Therefore, in this paper, we choose GP as the hyper-heuristic framework for evolving dispatching rules.

The framework of GP is given in Fig. 1. It is similar to the conventional genetic algorithm framework, except that it adopts special representations for individuals (i.e. GP programs), and evolves them using specific crossover and mutation operators. For evolving dispatching rules, GP represents each individual (candidate dispatching rule) as a syntax tree. An example of the well-known 2PT+WINQ+NPT rule [18] is given in Fig. 2. The crossover operator randomly selects a sub-tree from each parent, and then swaps them to produce two children. The mutation operator randomly selects a sub-tree from the parent, and replaces it with a newly generated sub-tree. For evaluating a dispatching rule, the rule is first applied to a set of JSS instances. The fitness value is then set to the average objective values of the resultant solutions.

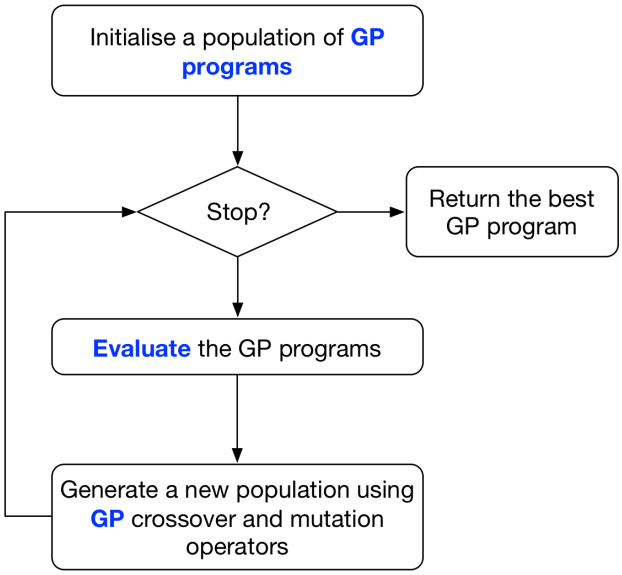


Fig. 1. The GP framework.

For evolving dispatching rules with GP, feature selection is an important and challenging issue. It has been demonstrated that including irrelevant features in the terminal set of GP can deteriorate the performance [16]. On the contrary, carefully selecting relevant features can lead to a significant improvement

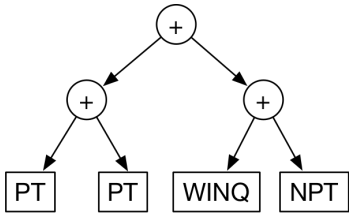


Fig. 2. The tree-based representation of the 2PT+WINQ+NPT rule.

[34]. Feature selection in GP for evolving dispatching rules has been overlooked so far. This paper conducts an in-depth study on this important topic.

C. Feature Selection

Feature selection is an important and necessary process in machine learning and data mining tasks [12], [56]. By removing irrelevant, redundant and misleading features, feature selection can reduce the number of features (i.e. dimensionality) of the data, increase the learning performance, reduce the training time, and improve the interpretability of the learned models [12], [56]. There have been a large number of feature selection methods proposed in the past decades [12], [28], mainly for classification and regression tasks [8], [32], [57]. Based on the evaluation criterion, existing feature selection methods can be classified into three categories [28], which are wrapper, filter and embedded approaches. Despite its importance, feature selection has seldom been used in designing dispatching rules for JSS.

Traditional feature selection approaches are not applicable to GPHH for dynamic JSS. First, the task (i.e. prioritising the operations in the queue) in GPHH for dynamic JSS is completely different from the traditional machine learning/data mining tasks (e.g. classification, regression and clustering), which makes traditional feature selection approaches inapplicable. Second, the instances in GPHH for dynamic JSS have dependencies on each other, while they are assumed to be independent in traditional machine learning/data mining.

GP is able to perform embedded feature selection, also called terminal selection, since GP can simultaneously identify relevant features and evolve the best GP tree using the relevant features in an adaptive evolutionary process. The relevance of the features/terminals are typically evaluated by counting their occurrence frequencies in the good individuals. During the evolutionary process, the probabilities of selecting features change accordingly based on the updated relevance. A typical example is the adaptive mutation [9], [40], [47], which selects the features to create a sub-tree based on their relevance rather than randomly, so that the more relevant features are more likely to be selected.

However, the commonly used frequency analysis has a major drawback that there may be many unnecessary or even meaningless occurrences of the features. For example, given a rule represented as $(A-B)/(A-B)+C/D$, A and B occur twice, while C and D occur only once. According to frequency analysis, A and B are more relevant than C and D. However, after simplification, the rule becomes $C/D+1$.

This means that both A and B are in fact not needed and may be useless features. In [34], a feature ranking and selection approach is proposed to improve the performance of GP for job shop scheduling, which is based on the contribution of features to the fitness. That approach has been demonstrated to be able to identify the relevant features more accurately. However, it requires a diverse set of good dispatching rules, which cannot be obtained in a trivial way. It simply extracts these rules by running GP for multiple times with different random seeds. As a result, it is very time consuming (about 70 hours on Intel(R) Core(TM) i7 CPU @3.60GHz for obtaining 30 rules) and thus inapplicable in practice. A new efficient feature selection algorithm is highly desired.

III. Proposed Feature Selection Algorithm

In this section, we propose an efficient feature selection algorithm (called NiSuFS) to select relevant features to improve the effectiveness of GP in evolving more promising dispatching rules. The proposed algorithm has the following two advantages:

1) It employs niching techniques to obtain a diverse set of good individuals (local optima) in a single GP run.
2) It replaces the original fitness evaluation with a surrogate model, which is based on a simplified and much less computationally expensive simulation. This way, the fitness evaluation becomes much more efficient without losing much accuracy.

A. Niching-GP Feature Selection Framework

The niching techniques [30], [49] have been demonstrated to be very effective to locate multiple local optimal solutions in the search space simultaneously. Niching-based evolutionary algorithms have been demonstrated to perform competitively for solving multi-modal [44] and multi-objective [19], [60] optimisation problems. There have been a variety of niching methods proposed, such as the fitness sharing methods [11], [35], the crowding methods (e.g. restricted tournament selection [13]) and the clearing method [41]. Here, we adopt the clearing method [41] since it is easier to control than the crowding methods and more efficient than the fitness sharing methods. The niching-GP feature selection framework of NiSuFS is described in Algorithm 1. The framework follows a standard GP process [26], except that it adjusts the fitness of the individuals using the clearing method before generating the offsprings (line 5). This way, the population is expected to converge to a number of different local optima instead of a single optimum. After that, a diverse set of good rules $\tilde{\mathcal{R}}$ are selected from the population $\boldsymbol{P}$ by `BestDiverseSet`$(\cdot)$ (line 9). Finally, a feature subset $\tilde{\mathcal{T}}$ is selected based on the selected rules $\tilde{\mathcal{R}}$ (line 10).

1) The Clearing Method: The basic idea of the clearing method is to "clear" the poor individuals in the crowding areas. The pseudo code of `Clearing`$(\cdot)$ is described in Algorithm 2. It has two parameters: the radius σ controlling the range of each niche, and the capacity κ which determines the number of individuals in each niche. Note that the fitness $fit(\cdot)$ is to be minimised in JSS (e.g. makespan and tardiness). Therefore, in line 10, an individual is cleared by setting its fitness to ∞.

Algorithm 1: $\tilde{\mathcal{T}} \leftarrow$ `NiSuFS`$(\mathcal{T}, N)$

Input: The feature set $\mathcal{T}$.
Output: A selected feature subset $\tilde{\mathcal{T}}$.
1 Initialise a population $\boldsymbol{P}$ randomly, set $gen \leftarrow 0$;
2 **while** $gen < N$ **do**
3 $gen \leftarrow gen + 1$;
4 `Evaluate`$(\boldsymbol{P})$;
5 $\boldsymbol{P'} \leftarrow$ `Clearing`$(\boldsymbol{P}, \sigma, \kappa)$;
6 $\boldsymbol{Q} \leftarrow$ `GenerateOffsprings`$(\boldsymbol{P'})$;
7 $\boldsymbol{P} \leftarrow \boldsymbol{Q}$;
8 **end**
9 $\tilde{\mathcal{R}} \leftarrow$ `BestDiverseSet`$(\boldsymbol{P}, \sigma, K)$;
10 $\tilde{\mathcal{T}} \leftarrow$ `FeatureSelection`$(\mathcal{T}, \tilde{\mathcal{R}})$;

Algorithm 2: $\boldsymbol{P'} \leftarrow$ `Clearing`$(\boldsymbol{P}, \sigma, \kappa)$ [41]

Input: A population $\boldsymbol{P}$, a radius σ, a capacity κ.
Output: A population $\boldsymbol{P'}$ with adjusted fitness values.
1 $\boldsymbol{P'} \leftarrow$ `SortFitness`$(\boldsymbol{P})$; `// from best to worst`
2 **for** $i = 1 \rightarrow |\boldsymbol{P'}|$ **do**
3 **if** $fit(\boldsymbol{P'}[i]) < \infty$ **then**
4 $size \leftarrow 1$;
5 **for** $j = i + 1 \rightarrow |\boldsymbol{P'}|$ **do**
6 **if** $fit(\boldsymbol{P'}[j]) < \infty$ *and* $\Delta(\boldsymbol{P'}[j], \boldsymbol{P'}[i]) < \sigma$ **then**
7 **if** $size < \kappa$ **then**
8 $size \leftarrow size + 1$;
9 **else**
10 $fit(\boldsymbol{P'}[j]) \leftarrow \infty$;
11 **end**
12 **end**
13 **end**
14 **end**
15 **end**
16 **return** $\boldsymbol{P'}$;

In Algorithm 2, a distance measure $\Delta(r_1, r_2)$ between two rules r_1 and r_2 is needed. In GP, a rule is represented as a GP tree. Unlike traditional tree distance measures such as tree edit distance [1], [59], two dispatching rules with different tree structures can essentially have exactly the same behaviour in making decisions. For example, the rules with the priority functions of x, $2x$, and $x + 1$ will always make the same decision. In addition, the redundant branches in the tree evolved during the GP process can also generate duplicated GP trees, such as $x + y$ and $x + \max\{y, y\}$). For comparing between two dispatching rules for JSS, the distance measure $\Delta(\cdot, \cdot)$ should reflect the difference in phenotypic behaviour instead of genotypic structure.

To this end, we adopt a phenotypic characterisation approach [15] to characterise the behaviour of a dispatching rule r as a fixed-length numeric vector. The phenotypic characterisation is described in Algorithm 3. It is based on a reference dispatching rule $\acute{r}$ and a set of *decision situations* $\boldsymbol{\Omega}$. Each decision situation $\Omega \in \boldsymbol{\Omega}$ includes a set of candidate operations and a job shop state, i.e. an instantiation of all the feature values (e.g. processing time, due date, current time, work remaining, etc.) of these candidate operations. In the algorithm, for the ith decision situation, the reference rule $\acute{r}$ is first applied to obtain the ranking vector $\acute{\boldsymbol{k}}$ of all the candidate

operations (line 2). Then, the index j of the operation with the highest priority assigned by $\acute{r}$ is identified (line 3). Finally, the ranking vector $\boldsymbol{k}$ of the operations assigned by rule r is obtained, and the rank of the jth operation in $\boldsymbol{k}$ is set to the ith element of the characteristic vector (lines 4 and 5).

Algorithm 3: $\boldsymbol{d} \leftarrow \texttt{PhenoCharacterisation}(r; \acute{r}, \boldsymbol{\Omega})$ [15]

Input: A dispatching rule r to be characterised, a reference dispatching rule $\acute{r}$, a set of decision situations $\boldsymbol{\Omega}$.
Output: A numeric (integer) characteristic vector $\boldsymbol{d}$.

1 **for** $i = 1 \rightarrow |\boldsymbol{\Omega}|$ **do**
2 $\quad \acute{\boldsymbol{k}} \leftarrow \texttt{ranks}(\acute{r}, \boldsymbol{\Omega}[i])$;
3 $\quad j \leftarrow \texttt{index}(1, \acute{\boldsymbol{k}})$;
4 $\quad \boldsymbol{k} \leftarrow \texttt{ranks}(r, \boldsymbol{\Omega}[i])$;
5 $\quad \boldsymbol{d}[i] \leftarrow \boldsymbol{k}[j]$;
6 **end**
7 **return** $\boldsymbol{d}$;

Obviously, if r and $\acute{r}$ have the same behaviour, then they will always give rank 1 to the same operation. Thus, $\boldsymbol{d}(r) = \boldsymbol{d}(\acute{r}) = \boldsymbol{1}$. Otherwise, the more different r behaves from $\acute{r}$, the more distant $\boldsymbol{d}(r)$ is from $\boldsymbol{d}(\acute{r})$. Then, the distance between two rules r_1 and r_2 can be defined as the Euclidean distance between the two characteristic vectors, as shown in Eq. (1).

$$\Delta(r_1, r_2) = \sqrt{\sum_{i=1}^{|\boldsymbol{\Omega}|} (\boldsymbol{d}_i(r_1) - \boldsymbol{d}_i(r_2))^2}. \tag{1}$$

According to Algorithm 3, a set of decision situations $\boldsymbol{\Omega}$ and a reference rule $\acute{r}$ are needed to calculate $\Delta(\cdot)$. In this study, $\boldsymbol{\Omega}$ is generated as follows:

Step 1. Apply the Shortest Processing Time rule on a JSS simulation with 10 machines, 2500 jobs, full operations, utilisation level of 0.95, and record all the decision situations as $\boldsymbol{\Omega}'$.
Step 2. Remove all the decision situations from $\boldsymbol{\Omega}'$ whose number of candidate operations are smaller than 10.
Step 3. Randomly select 20 decision situations from the remaining decision situations in $\boldsymbol{\Omega}'$ to form $\boldsymbol{\Omega}$.

The characteristic vector only considers the ranking of the operation selected by the reference rule. Therefore, a better reference rule tends to lead to a more accurate and representative characterisation. To this end, at each generation, we reset $\acute{r}$ to the best individual in the population.

2) Final Best Diverse Set: Given the final population $\boldsymbol{P}$, a diverse set of K good individuals are to be selected. For this purpose, we continue to use the niching technique, and select only the best individual in each niche. The function $\texttt{BestDiverseSet}(\boldsymbol{P}, \sigma, K)$ is described in Algorithm 4. First, the set $\tilde{\mathcal{R}}$ is set to empty, and the individuals in $\boldsymbol{P}$ are sorted from the best to the worse. Then, for each individual $\boldsymbol{P}'[i]$ in the sorted population, we examine whether it is in the niche of any individual in $\tilde{\mathcal{R}}$ (lines 4–10). If not, then it is added into $\tilde{\mathcal{R}}$. The above steps are repeated until the number of individuals $\tilde{\mathcal{R}}$ reaches K or all the individuals in the population have been examined.

3) Feature Selection: A more important feature tends to make higher contribution to more good individuals.

Algorithm 4: $\tilde{\mathcal{R}} \leftarrow \texttt{BestDiverseSet}(\boldsymbol{P}, \sigma, K)$

Input: A population $\boldsymbol{P}$, a radius σ, the number of selected individuals K.
Output: A diverse set of good individuals $\tilde{\mathcal{R}}$.

1 Set $\tilde{\mathcal{R}} \leftarrow \emptyset$;
2 $\boldsymbol{P}' \leftarrow \texttt{SortFitness}(\boldsymbol{P})$;
3 **for** $i = 1 \rightarrow |\boldsymbol{P}'|$ **do**
4 $\quad inNiche \leftarrow false$;
5 $\quad$ **foreach** $r \in \tilde{\mathcal{R}}$ **do**
6 $\quad\quad$ **if** $\Delta(\boldsymbol{P}'[i], r) < \sigma$ **then**
7 $\quad\quad\quad inNiche \leftarrow true$;
8 $\quad\quad\quad$ **break**;
9 $\quad\quad$ **end**
10 $\quad$ **end**
11 $\quad$ **if** *not* $inNiche$ **then**
12 $\quad\quad \tilde{\mathcal{R}} \leftarrow \tilde{\mathcal{R}} \cup \boldsymbol{P}'[i]$;
13 $\quad\quad$ **if** $|\tilde{\mathcal{R}}| = K$ **then**
14 $\quad\quad\quad$ **return** $\tilde{\mathcal{R}}$;
15 $\quad\quad$ **end**
16 $\quad$ **end**
17 **end**
18 **return** $\tilde{\mathcal{R}}$;

In other words, the importance of a feature depends on both *individual fitness* and its *contribution to individuals*. Therefore, given the selected individuals $\tilde{\mathcal{R}}$, the function $\texttt{FeatureSelection}(\mathcal{T}, \tilde{\mathcal{R}})$ takes both these two aspects into account to select features out of the feature set $\mathcal{T}$. It is described in Algorithm 5. It is a weighted majority voting process, in which the selected rules in $\tilde{\mathcal{R}}$ vote for the features in $\mathcal{T}$. First, the contribution $\zeta(t, \tilde{r})$ of each feature $t \in \mathcal{T}$ to each rule $\tilde{r} \in \tilde{\mathcal{R}}$ is calculated by Eq. (2) developed in [34], where $(\tilde{r}|t = 1)$ stands for the rule obtained by fixing the feature t to 1 in the rule $\tilde{r}$ (lines 1–5). For example, $(\text{PT} + \text{WINQ}|\text{PT} = 1) = (1 + \text{WINQ})$.

$$\zeta(t, \tilde{r}) = fit(\tilde{r}) - fit(\tilde{r}|t = 1). \tag{2}$$

Since the fitness is to be minimised, a rule with a smaller fitness should have a larger voting weight. Then, the voting weight of each rule $\tilde{r}$ is defined as a a monotonically decreasing function of its fitness, given by Eqs. (3)–(6).

$$w(r) = \max\left\{\frac{g(r) - g_{\min}}{g_{\max} - g_{\min}}, 0\right\}, \tag{3}$$

$$g(r) = \frac{1}{1 + fit(r)}, \tag{4}$$

$$g_{\max} = \frac{1}{1 + \min\{fit(r)|r \in \mathcal{R}^*_{gen}\}}, \tag{5}$$

$$g_{\min} = \frac{1}{1 + \max\{fit(r)|r \in \mathcal{R}^*_{gen}\}}, \tag{6}$$

where $\mathcal{R}^*_{gen}$ is the set of the best rules in the population in each generation of the niching-based GP process.

After calculating the voting weights for the rules (lines 6–8), the weighted majority voting process starts. Specifically, a rule $\tilde{r} \in \tilde{\mathcal{R}}$ votes for a feature $t \in \mathcal{T}$ if $\zeta(t, \tilde{r})$ is greater than the contribution threshold ϵ (lines 13–15). The contribution threshold is set to a small positive value to prevent selecting

weakly relevant features. Here it is set to $\epsilon = 0.001$[1]. Then, a feature t is selected if the total weight voting for it is larger than the total weight not voting for it (lines 17–19).

Algorithm 5: $\tilde{\mathcal{T}} \leftarrow$ `FeatureSelection`$(\tilde{\mathcal{R}})$

Input: A selected set of individuals $\tilde{\mathcal{R}}$.
Output: The selected terminals $\tilde{\mathcal{T}}$.

```
1  foreach t ∈ 𝒯 do
2      foreach r̃ ∈ ℛ̃ do
3          Calculate the contribution ζ(t, r̃) by Eq (2);
4      end
5  end
6  foreach r̃ ∈ ℛ̃ do
7      Calculate w(r̃) by Eqs. (3)–(6);
8  end
9  Set 𝒯̃ ← ∅;
10 foreach t ∈ 𝒯 do
11     Set the total voted weight W(t) ← 0;
12     foreach r̃ ∈ ℛ̃ do
13         if ζ(t, r̃) > ϵ then
14             W(t) ← W(t) + w(r̃);
15         end
16     end
17     if W(t) ≥ (Σ_{r̃∈ℛ̃} w(r̃)) /2 then
18         𝒯̃ ← 𝒯̃ ∪ t;
19     end
20 end
21 return 𝒯̃;
```

B. Surrogate Model for Fitness Evaluation

The main computational complexity of GP comes from the fitness evaluation. For evolving dispatching rules using GP, the fitness evaluation is very computationally expensive, as a fitness evaluation requires one or more complex JSS simulations (e.g. 10 machines and 2500 jobs, each with 10 operations). Therefore, it is highly desired to improve the efficiency of fitness evaluation. Surrogate model [24], [54], [58] is a common strategy to this end.

In DJSS, an intuitive surrogate model is to use simplified instances or simulations with fewer jobs and machines. For static JSS, we have demonstrated that under the same or similar ratio between the numbers of jobs and machines, the dispatching rules trained on smaller instances can be reused on larger instances [33]. For DJSS, Nguyen *et al.* [39] proposed a number of simplified surrogate models for the original simulation model, which have much fewer jobs and machines than the original one. In our work, we adopt the HalfShop surrogate model, since it showed the best evaluation accuracy among all the surrogate models proposed in [39]. Following the settings in [39], the HalfShop surrogate model sets the number of jobs and warmup jobs to 500 and 100 respectively.

Based on the surrogate model, when evaluating each individual r (line 4 in Algorithm 1), the fitness function is defined as follows:

$$fit(r) = \frac{1}{|\mathcal{I}_{\text{train}}|} \sum_{I \in \mathcal{I}_{\text{train}}} \frac{f(r; HS(I))}{f_{\text{ref}}(HS(I))}, \tag{7}$$

[1]In preliminary experiment, we have compared different values including 0.001, 0.005, 0.0005, and found no significant difference between them.

where $\mathcal{I}_{\text{train}}$ is the training set (simulation replications), $HS(I)$ stands for the HalfShop surrogate model of the simulation I. $f(r; HS(I))$ is the objective value (e.g. makespan or tardiness) of the schedule obtained by applying the dispatching rule r to $HS(I)$. $f_{\text{ref}}(I)$ is the reference objective value for training instance $HS(I)$, obtained by the WATC rule, which is an effective dispatching rule to minimise the mean weighted tardiness. The only difference between Eq. (7) and the original fitness function is that each original simulation I is replaced by its HalfShop surrogate model $HS(I)$.

IV. Experimental Studies

The configuration of the DJSS simulation used in the experimental studies is described in Table I. It is the same as that in [16] except that the due date factor is set to 1.3 (which has been used in previous studies, e.g. [27]) to generate tighter due dates. For the number of operations per job, *missing* means that the number of operations for each job is randomly sampled between 2 and 10, while *full* indicates that the number of operations is 10 for all the jobs.

TABLE I
THE DJSS SIMULATION SYSTEM CONFIGURATION.

Parameter	Value
#machines	10
#jobs	2500 for training, 5000 for test
#warmup jobs	20% of the total number of jobs
#operations per job	*missing*, *full*
Job arrival process	Poisson process
Utilisation level	{0.95, 0.8} for training 0.95, 0.9, 0.85 and 0.8 for testing
Due date	1.3 times the total processing time
Eligible machine	Uniform discrete distribution
Processing time	Uniform discrete distribution between 1 and 49

The training set consists of two utilisation levels (0.95 and 0.8) and two settings of the number of operations (missing and full). Thus, there are 4 different training configurations. We generate only one replication for each configuration, but change the random seed for generating the replications in each generation of the GP process to improve the generalisability of the rules. Therefore, 4 training instances are used for evaluating the rules. For the sake of convenience, the training set is called the *H2010* set, since it is the same as that used by Hildebrandt *et al.* in 2010 [16]. For testing the evolved rules, we use 8 test sets with different job shop scenarios (4 utilisation levels $\times$ 2 configurations on the number of operations, i.e. missing and full). For each test set, 20 random replications were generated independently. The 8 test sets can well represent a wide range of dynamic job shop scenarios.

In the experiment, the objective is set to minimising the mean weighted tardiness.

A. Parameter Settings of Genetic Programming

First, the terminal set of GP is to be determined. An overview of the promising features is given in [4]. For the basic JSS model considered in this study, we selected 16 commonly used features, which are listed in Table II.

TABLE II
THE 16 COMMONLY USED FEATURES SELECTED IN THE TERMINAL SET OF GP.

Notation	Description
NOW	The current time.
NOIQ	Number of operations in the current queue.
WIQ	Work in the current queue.
MRT	Ready time of the machine.
PT	Processing time of the operation.
NOPT	Processing time of the next operation.
ORT	Ready time of the operation.
NMRT	Ready time of the next machine.
WKR	Work remaining (including the current operation).
NOR	Number of operations remaining.
WINQ	Work in the next queue.
NOINQ	Number of operations in next queue.
FDD	Flow due date of the operation.
DD	Due date of the job.
W	Weight of the job.
AT	Arrival time of the job.

The function set is $\{+, -, \times, /, \min, \max, \text{ifte}\}$. The first three are traditional arithmetic operations. The function $/$ is the protected division, which returns 1 if the denominator is 0. The function $\min$ ($\max$) takes two arguments and returns the smaller (larger) one. The ifte function takes three arguments a, b and c in order. If $a > 0$, then it returns b. Otherwise, it returns c.

The parameter setting of GP is given in Table III. Most parameters take the standard values in this field [6], [34] and also GP research [53]. The algorithm was implemented by ECJ [29].

TABLE III
THE PARAMETER SETTING OF GP.

Notation	Description	Value
$popsize$	Population size	1024
$depth$	Maximal depth	8
$\Pr_x$	Crossover rate	0.85
$\Pr_m$	Mutation rate	0.1
$\Pr_r$	Reproduction rate	0.05
N	Number of generations	51

B. Parameter Analysis for Feature Selection

NiSuFS employs the niching technique, more specifically the clearing method [41]. Therefore, its performance depends on the two parameters σ (radius) and κ (capacity) of `Clearing`$(\cdot)$. In addition, σ is also used in Algorithm 4 to find the best diverse set. These two parameters interact with each other, and determine the balance between convergence and diversity of the population in Algorithm 1. The population will become more diverse with the increase of σ and the decrease of κ. In the experiments, we fix $\kappa = 1$ to have a finer control of the diverse set of individuals. Specifically, with $\kappa = 1$, we can ensure that for *each* individual in the diverse set, no other individual is within the radius σ around it. It has also been demonstrated that $\kappa = 1$ can reach more peaks [41]. Then, we compared between different σ values. In the experiments, we tested the following σ values:

- $\sigma = 0$: For each considered individual, all the remaining individuals with zero distance (i.e. "clones") are cleared;
- $\sigma = 1$: For each considered individual, all the remaining individuals with distance of 1 are cleared;
- $\sigma = 5$: For each considered individual, all the remaining individuals with distance of 5 are cleared.

For each of the three σ value, we conducted 30 independent runs of GP (with the parameter settings given in Table III and the terminal set given in Table II) on both the original H2010 training set and the HalfShop surrogate model of the H2010 training set. As a result, we obtained 6 sets of results (3 σ values $\times$ 2 training sets).

Figs. 3 and 4 show the distribution of the fitness of the 30 selected individuals in Algorithm 1 with $\kappa = 1$ and different σ values in 30 independent runs. In Fig. 3, the training set is the original H2010 set. In Fig. 4, the training set is the half-shop surrogate model of the H2010 set.

From Fig. 3, it can be seen that when using the original H2010 training set, both the mean and standard deviation of the distribution of the fitnesses increase with the increase of σ. This implies that if σ is larger, then the selected individuals are more diversely distributed. This is consistent with our expectation, as a larger σ value indicates more bias to diversity, and thus a larger standard deviation in the distribution of the fitnesses. However, the search tends to be less capable of refining the promising regions and finding better solutions. As a result, the GP obtained much worse fitness values with $\sigma = 5$ than with $\sigma = 0$ and $\sigma = 1$. Note that when $\sigma = 0$, the 30 selected individuals have very similar fitness values (e.g. reflected by the small standard deviations in runs 8, 13, 14 and 15). This indicates that the selected individuals may not be diverse enough and the resultant feature selection may be too biased.

Fig. 4 shows similar patterns as Fig. 3. As σ increases, both the mean and standard deviation of the distribution of the fitnesses increase. One can see that when using the HalfShop surrogate training set, the selected individuals seems to be more diversely distributed (with a larger standard deviation), especially for $\sigma = 0$ and $\sigma = 1$.

Given the distributions of the selected individuals, it is important to know how these different distributions affect the results of feature selection, which is the ultimate goal of this work. To this end, we plot the feature selection results of Algorithm 1 with different σ values for both using the original training set and using the HalfShop surrogate training set in Figs. 5 and 6. Each figure includes three matrices, each corresponding to the results obtained by one σ value (0, 1 or 5). In each matrix, each row indicates a run (from 1 to 30), and each column represents a feature (details given in Table II). If a feature X is selected in the ith run, then we draw a point in the location corresponding to the feature X and the ith run.

From Figs. 5 and 6, one can see that the 6 feature selection results are similar to each other (especially the results obtained by $\sigma = 0$ and $\sigma = 1$). To be more specific, we have the following observations:

- In all the 6 matrix plots, PT and W were selected in all the 30 runs. This is consistent with our domain knowledge

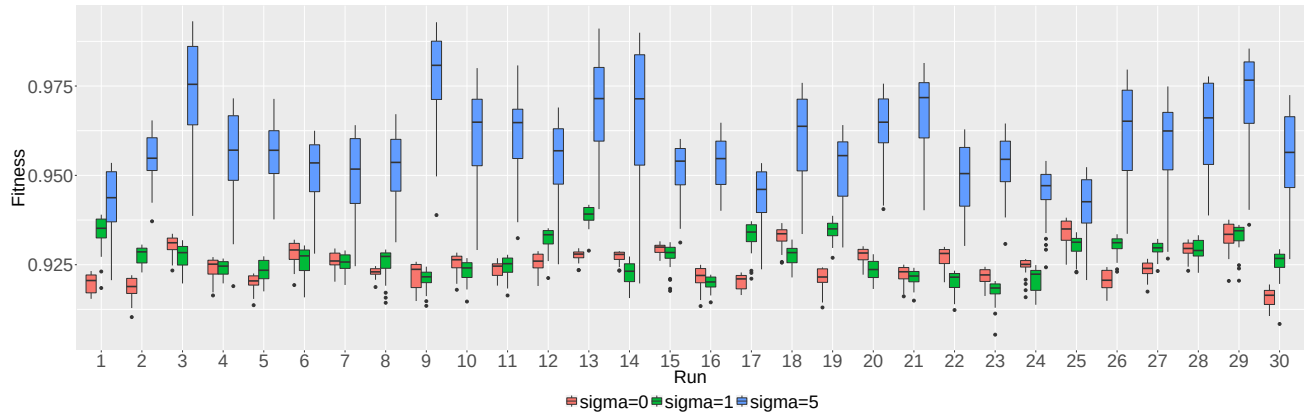


Fig. 3. The distribution of the fitness of the 30 selected individuals in Algorithm 1 with $\kappa = 1$ and different σ values in 30 independent runs. The training set is the original H2010 set.

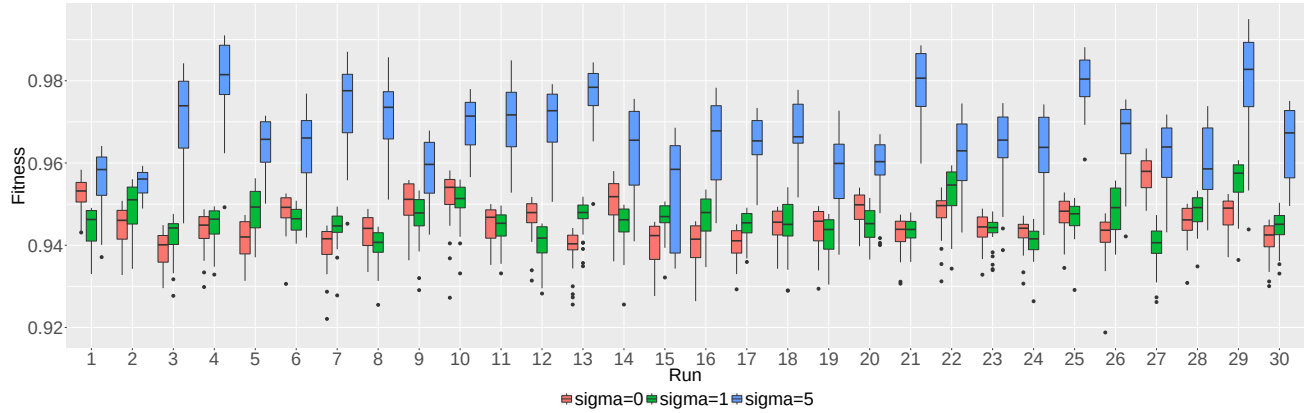


Fig. 4. The distribution of the fitness of the 30 selected individuals in Algorithm 1 with $\kappa = 1$ and different σ values in 30 independent runs. The training set is the HalfShop surrogate model of the H2010 set.

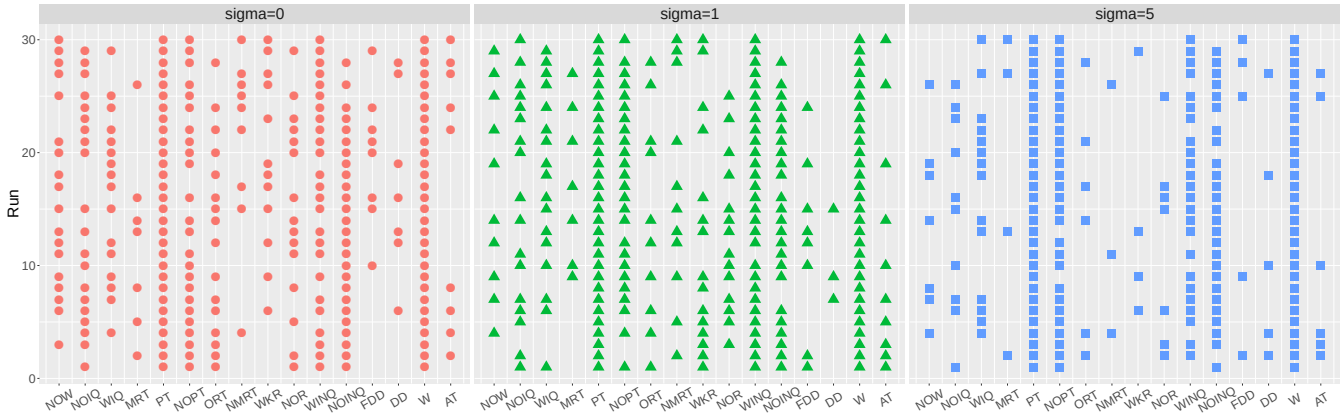


Fig. 5. The matrix plot of the feature selection results of Algorithm 1 when using the original training set and different σ values. If a feature X is selected in the ith run, then a point is added in the location corresponding to the feature X (column) and the ith run (row).

that PT and W are extremely important features to minimise the mean weighted tardiness.

- In all the 6 matrix plots, WINQ and NOINQ were selected in most runs. Furthermore, in all the runs, at least one of WINQ and NOINQ are selected. This implies that the workload information (total processing time and/or number of operations) in the next queue is very important to minimise the mean weighted tardiness.
- The feature NOPT was selected in most runs (more than 70% of the time), especially when $\sigma = 1$. This shows that

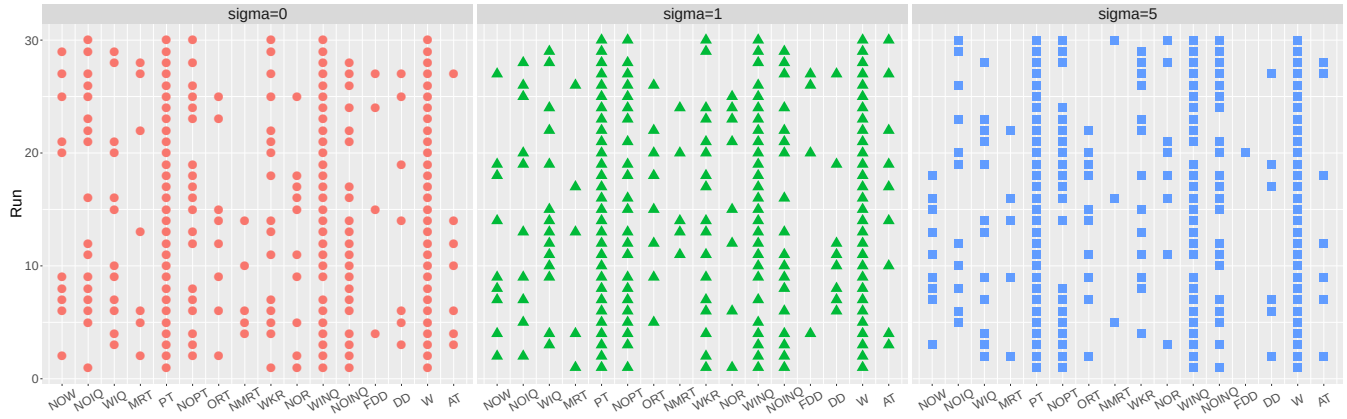


Fig. 6. The matrix plot of the feature selection results of Algorithm 1 when using the HalfShop surrogate training set and different σ values. If a feature X is selected in the ith run, then a point is added in the location corresponding to the feature X (column) and the ith run (row).

NOPT is an important feature for minimising the mean weighted tardiness.

- MRT, NMRT, FDD and DD were not selected in most runs, indicating that they may be redundant or irrelevant for minimising the mean weighted tardiness. The elimination of DD looks counter-intuitive, since DD seems to be closely related to tardiness. However, this actually makes sense considering the configuration of the experiments, in which the due date factor is so tight that almost all the jobs are tardy. With such a tight due date factor, the problem becomes completing the jobs, especially those with higher weights, as soon as possible. In this situation, minimising the mean weighted tardiness is nearly reduced to minimising the mean weighted flowtime, where DD is actually irrelevant. This is consistent with the findings in [38], [50], where the best rules for minimising mean tardiness are 2PT+WINQ+NPT and 2PT+LWKR+FDD, which do not have due date information. The results clearly show the advantage of the NiSuFS, as it can identify DD as an unimportant feature in this particular case, while humans tend to include DD due to a wrong intuition that it is always important in minimising tardiness.
- It is not very clear whether the remaining features (NOW, NOIQ, WIQ, ORT, WKR, NOR, AT) are important or not.

Overall, Figs. 5 and 6 show that using the HalfShop surrogate model in fitness evaluation can maintain a very high accuracy in terms of feature selection, and the results are not sensitive to the parameter settings, at least when σ is not too large. This verifies the efficacy of NiSuFS.

On average, NiSuFS selected 7.73, 7.50 and 7.17 features for $\sigma = 0$, $\sigma = 1$ and $\sigma = 5$ respectively, which are much smaller than the original feature set consisting of 16 features.

In the subsequent experiments, we choose the feature selection results obtained by $\sigma = 1$ for NiSuFS. The reason for selecting $\sigma = 1$ is that from Fig. 6, it is observed that NOPT, which is known to be an important feature based on our prior knowledge, was selected more often with $\sigma = 1$ than with other σ values.

C. Efficiency

Since one of the major goals of this paper is to improve the efficiency of the feature selection, it is important to analyse the computation time. Table IV gives the mean and standard deviation of the computational time of the 30 independent runs of the standard GP and the feature selection algorithm 1 with the original training set and the HalfShop surrogate training set.

TABLE IV
THE COMPUTATIONAL TIME (MEAN AND STANDARD DEVIATION) OF THE STANDARD GP AND THE FEATURE SELECTION ALGORITHM 1 WITH THE ORIGINAL TRAINING SET AND THE HALFSHOP SURROGATE TRAINING SET.

Algorithm	Time (second) mean (std)
Standard GP	8449.32 (863.55)
Algorithm 1 + Original	8767.96 (629.15)
Algorithm 1 + Surrogate (NiSuFS)	740.74 (43.80)

From the table, it is clear that Algorithm 1 with the original training set has similar computational time as the standard GP. This implies that the niching component (line 5 of Algorithm 1) does not significantly increase the computational time of GP. The computational time of Algorithm 1 with the surrogate training set is much smaller than (less than 10% of) that of both other algorithms. This is consistent with our expectation. In the original training set, a full job shop has 2500 jobs, each with 10 operations. Therefore, there are $2500 \times 10 = 25000$ decision points, each for an operation. In the HalfShop surrogate model, a full job shop consists of only 500 jobs, each with 5 operations. This leads to $500 \times 5 = 2500$ decision points. In addition, at each decision point, the queue in the surrogate model tends to be shorter than that in the original model. Therefore, the fitness evaluation when using the HalfShop surrogate model is expected to take less 10% computational time of that when using the original model.

D. Quality of Selected Features – Test Performance

In this sub-section, we investigate the effectiveness of NiSuFS in terms of the quality of the selected feature sets. First

we obtain the 30 feature sets selected by the 30 independent runs of NiSuFS. For the sake of convenience, these feature sets are denoted as FSx ($x = 1, \ldots, 30$) hereafter. Then, for each feature set FSx, its quality is evaluated by the test performance of the dispatching rules evolved by GP using FSx as the terminal set. To this end, for each FSx, 30 independent runs of the standard GP (parameter settings given in Table III) are carried out on the H2010 training set. Then the test performance of the 30 evolved rules are evaluated and compared with the rules evolved by GP with the compared feature sets.

To verify the effectiveness of NiSuFS as a feature selection algorithm, a fair comparison should be with other feature selection algorithms. However, there is no existing *practical* feature selection algorithm for evolving dispatching rules so far. The only feature selection algorithm proposed in [34] is too time consuming (70 hours even when running on power i7 CPUs) and thus impractical. In this case, we compare with the following two *feature sets* rather than feature selection algorithms:

- The entire feature set given in Table II, denoted as "All". The entire feature set can be seen as a baseline feature set.
- The best-so-far feature set {MRT, PT, NOPT, WINQ, NOINQ, W} that was preselected in [34], denoted as "BSF".

Note that additional 740 seconds (shown in Table IV) are required to obtain the feature sets by NiSuFS, which is around the computational time of 4 GP generations. To make a fair comparison, we give 55 generations to the GP with "All", and 51 generations to the GP with the FSx's, so that they will have comparable computational time.

Fig. 7 shows the boxplots of the test performance of the rules evolved by GP (in 30 independent runs) using all the features ("All"), the best-so-far feature set ("BSF"), and the 30 feature sets selected by NiSuFS on the 8 test sets. For each test set and each feature set selected by NiSuFS, the Wilcoxon rank sum test was conducted to compare with "All" and "BSF" under significance level of 0.05. Given that the result of "BSF" is significantly better than that of "All", we fill the boxplots in Fig. 7 based on the two tests as follows:

- If the result of a feature set is significantly worse than that of "All", then the boxplot is filled in red;
- If there is no statistical difference between the results of a feature set and of "All", then the boxplot is filled in white ("All" is filled in white);
- If the result of a feature set is significantly better than that of "All", then
  - if it is significantly worse than that of "BSF", then the boxplot is filled in orange;
  - if there is no statistical difference between the results of a feature set and of "BSF", then the boxplot is filled in blue ("BSF" is filled in blue);
  - if it is significantly better than that of "BSF", then the boxplot is filled in green.

From Fig. 7, we have the following observations:

- About 15 to 20 of the 30 feature sets showed significantly better test performances than "All" (the boxplots in orange, blue and green), especially when the utilisation level is high. Almost all the feature sets performed no worse than "All" statistically. In other words, NiSuFS can select much better feature sets than the entire feature set in most cases, and can almost guarantee to select feature sets that are not significantly worse than the entire feature set.
- Only 7 out of the total 30 feature sets (FS10, FS14, FS17, FS20, FS22, FS27 and FS29) showed significantly worse test performances than "All" on part of the test sets (the red boxplots). When looking into the details, it is found that these 7 feature sets missed at least one important feature (e.g. FS10, FS14, FS17, FS20, FS22 and FS29 missed NOPT, and FS27 missed WINQ). This shows the importance of keeping all the important features, although challenging, while removing the redundant/irrelevant features.
- At least half of the 30 feature sets performed significantly better than or the same as the best-so-far feature set "BSF" (the blue and green boxplots). This indicates that NiSuFS can often select very promising feature sets (at least the same as the best-so-far feature set).
- There are 2 to 5 feature sets that significantly outperformed "BSF" (the green boxplots) in almost all cases (except for 0.95-missing). In other words, NiSuFS managed to find better feature sets than the best-so-far feature set {MRT, PT, NOPT, WINQ, NOINQ, W} for minimising the mean weighted tardiness.

Fig. 8 shows the computational time of the GP with "All" (55 generations), "BSF" (51 generations) and the 30 feature sets (51 generations) plus the time for NiSuFS. Wilcoxon rank sum test was conducted between the computational time of each feature set and that of "All". If a feature set has a significantly shorter computation time (under significance level of 0.05), then the boxplot is filled in green. If there is no statistical difference between the two computational times, then the boxplot is filled in white.

From Fig. 8, it can be seen that the computational time of the GP with the selected feature sets (51 generation) plus the feature selection time is at least statistically the same as the computation time of the GP with "All" (55 generations).

In summary, Figs. 7 and 8 show that NiSuFS can help GP evolve significantly better rules within similar or even shorter computational time.

V. Further Analysis

A. Generalisability

First, we investigate the effect of the selected feature set on the generalisation of the dispatching rules evolved by GP. To this end, we plot the training and test performances of the dispatching rules evolved by 30 independent runs of GP with different feature sets in Fig. 9. Here, we select (a) the entire feature set "All", (b) the feature set preselected in [34], and (c) the feature sets FS10, FS16, FS21 and FS28 selected by NiSuFS. FS10 is a representative poor feature set that does

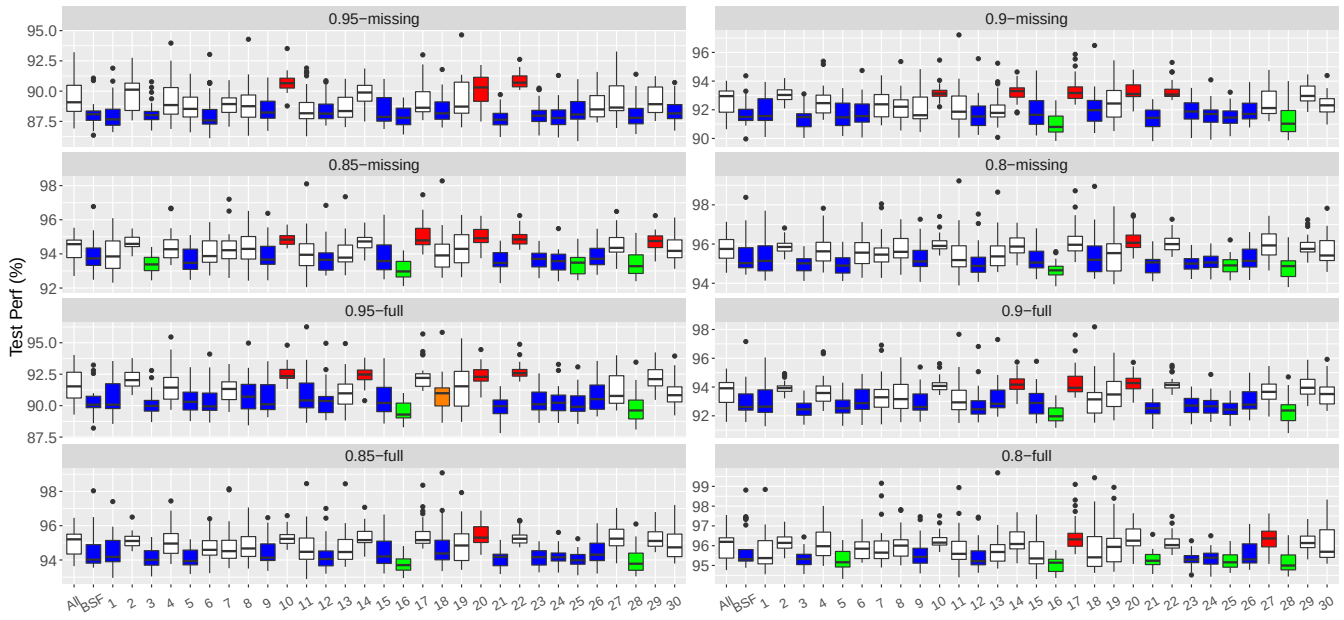


Fig. 7. The boxplots of the test performance of the rules evolved by GP (in 30 independent runs) using all the features (All), the best-so-far feature set (BSF), and the 30 feature sets selected by NiSuFS. The Wilcoxon rank sum test was conducted between each feature set selected by NiSuFS and "All" ("BSF") feature set under significance level of 0.05. Based on the test results, the filled colours from best to worse are green, blue, orange, white, red.

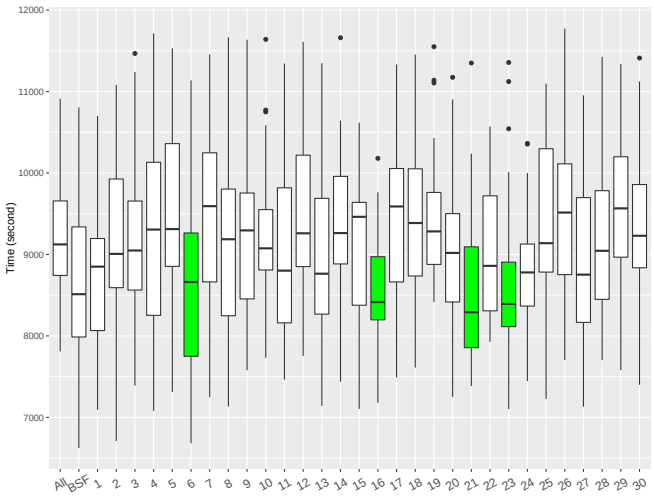


Fig. 8. Computational time of the GP with "All" (55 generations), "BSF" (51 generations) and the 30 feature sets (51 generations) plus the time for NiSuFS. If one feature set has a significantly shorter computational time than that of the GP with "All", then the boxplot is filled in green. If the two computational times are statistically the same, then the boxplot is filled in white.

not select the feature NOPT. FS16, FS21 and FS28 represent three promising feature sets that included all the important features (i.e. PT, NOPT, WINQ, NOINQ and W), and achieved significantly better test performances than "BSF" on some test sets. However, they have different selected features. For the sake of convenience, the elements in these feature sets are given below:

- "BSF" = {MRT, PT, NOPT, WINQ, NOINQ, W},
- FS10 = {WIQ, PT, WINQ, NOINQ, DD, W, AT},
- FS16 = {PT, NOPT, WINQ, NOINQ, W},
- FS21 = {PT, NOPT, WKR, NOR, WINQ, NOINQ, W},
- FS28 = {NOIQ, WIQ, PT, NOPT, WINQ, NOINQ, W}.

In other words, based on FS16, "BSF" has an additional feature MRT, FS21 has two additional features {WKR, NOR}, and FS28 has two additional features {NOIQ, WIQ}.

Note that we changed the random seed for the training simulations in each generation, and set different maximal number of generations for "All" (55 generations) and other feature sets (51 generations). To eliminate the effect of different training simulations on the training performance, we use the training performance in generation 51 for all the feature sets.

From Fig. 9, we can have the following observations:

- For all the selected feature sets, there is a strong positive correlation between the training and test performances of the dispatching rules. That is, given the same feature set, the rule with a better training performance tends to perform better on the test sets.
- Compared to other feature sets, the rules obtained by "All" have larger standard deviations in their training and test performances. This demonstrates that a larger feature set with redundant/relevant features makes it harder for GP to search in the larger search space, and thus less capable of finding promising regions.
- FS16, FS21 and FS28 and "BSF" have much better training and test performances than "All" (their results are located towards the bottom-left areas). The training and test performances of FS10 are much worse than that of "All" (located towards the top-right areas) due to the lack of NOPT in the terminal set.
- Compared to "BSF", FS16 achieved both better training and test performances, as well as better generalisability, i.e. smaller difference between training and test performances. Note that the only difference between FS16

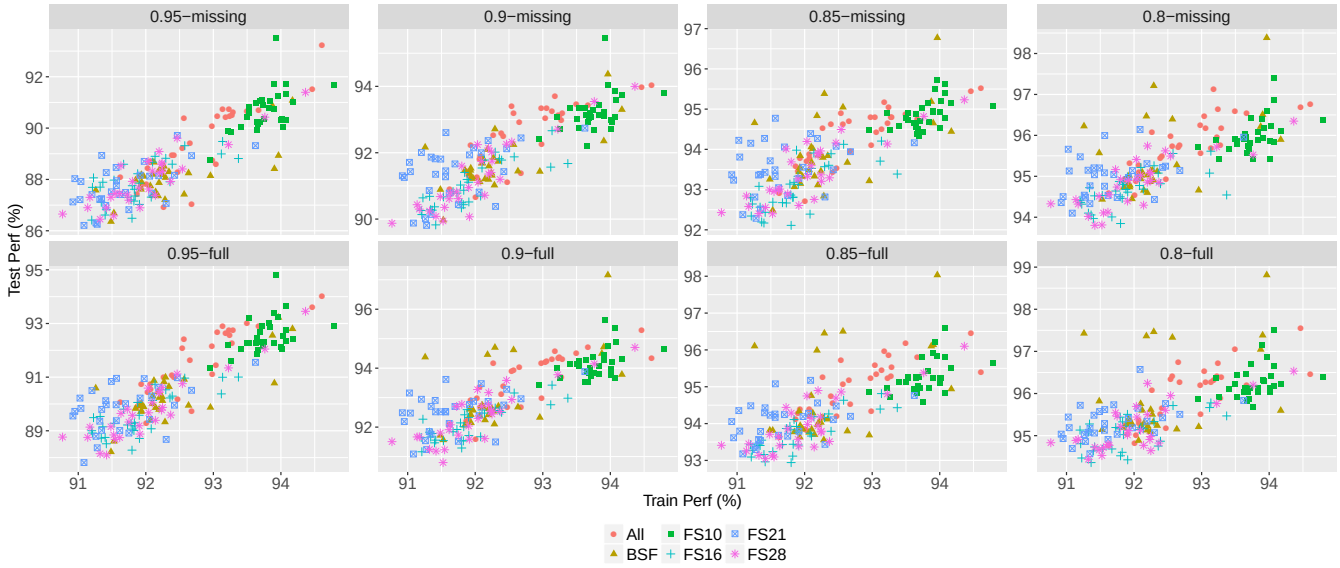


Fig. 9. The training versus test performance scatter plots of the dispatching rules evolved by 30 independent runs of GP with the entire feature set (All), the best-so-far feature set obtained in [34] (BSF), and several feature sets selected by NiSuFS (FS10, FS16, FS21 and FS28) on the 8 test sets.

and "BSF" is that FS16 removes the feature MRT from "BSF". This demonstrates that MRT might be redundant given the feature set {PT, NOPT, WINQ, NOINQ, W}. Moreover, it shows that even a slight modification of the terminal set can significantly affect the search effectiveness of GP and the generalisability of the evolved dispatching rules.

- Although FS21 obtained slightly better training performance than FS16, its generalisability is much worse than that of FS16. The difference between the training and test performances is much larger for FS21 than for FS16, especially when the training performance is extremely good (the most left areas). Compared to FS16, FS21 has two extra features WKR and NOR, which are both related to the work remaining (total processing time and number of operations). These two features might be useful in minimising the mean weighted tardiness (more than 20 runs selected at least one of them according to Fig. 6). However, a larger feature set tends to result in more complex dispatching rules, and thus worse generalisability.
- FS28 showed similar generalisability and test performance as that of FS16, which are better than FS21. However, it has two outliers on the top-right side, with much worse training and test performances. This indicates that the additional NOIQ and WIQ features may not provide extra useful information, but may mislead GP to poor local optima sometimes.

B. Feature Selection vs Training Performance

Fig. 9 shows a strong correlation between the training and test performances. Here we conduct further analysis to investigate the relationship between feature selection and training performance.

First we focus on the training curves. Fig. 10 shows the training curves of GP using feature sets of "All", "BSF", FS10, FS16, FS21 and FS28 (note that "All" has 55 generations). From the figure, one can see that during the early stage of GP (before the 10th generation), the GP with "All" converged more slowly than the GPs with other feature sets. This demonstrates the efficacy of using smaller feature sets in improving convergence speed. Then, the disadvantage of not including NOPT in the terminal set became more obvious, and the GP with FS10 started to be stuck in poor local optima. It was finally outperformed by the GP with "All". On the other hand, the GPs with "BSF", FS16, FS21 and FS28 continued to improve, and finally achieved much better training performances than the GP with "All". This shows that these more compact feature sets led to both faster convergence speed and better final results.

Then, for each feature, we investigate the relationship between whether it is selected in the terminal set and the resultant training performance of GP. Fig. 11 shows the distributions of the training performances of the 30 independent runs of GP, categorised by whether each feature is selected or not.

From Fig. 11, it is obvious that when NOPT is not selected, the training performance is much worse than when it is selected. Similar (but less obvious) phenomena can be found for NOR and WKR, indicating that these two features are likely to be important features. On the other hand, for AT, DD, FDD, MRT, NMRT, NOIQ, NOW, ORT and WIQ, the best training performance can be achieved without selecting them. Therefore, they tend to be redundant given the remaining features (i.e. NOINQ, NOPT, NOR, PT, W, WINQ, WKR).

C. Program Size

Fig. 12 shows the program size (number of nodes) of the best individual in each generation of the GPs with "All", "BSF", FS10, FS16, FS21 and FS28. From the figure, one can hardly distinguish the effect of different feature sets on the program size of the best individual. We expect more compact

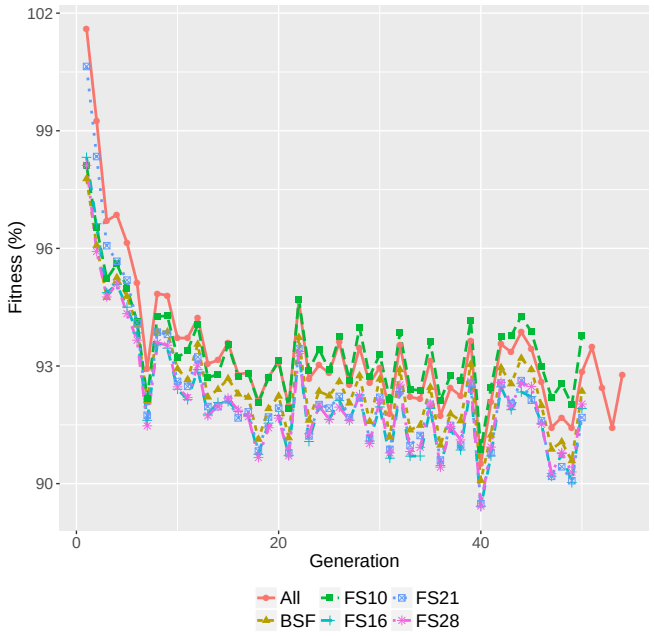


Fig. 10. The training curves of GP using feature sets of "All", "BSF", FS10, FS16, FS21 and FS28.

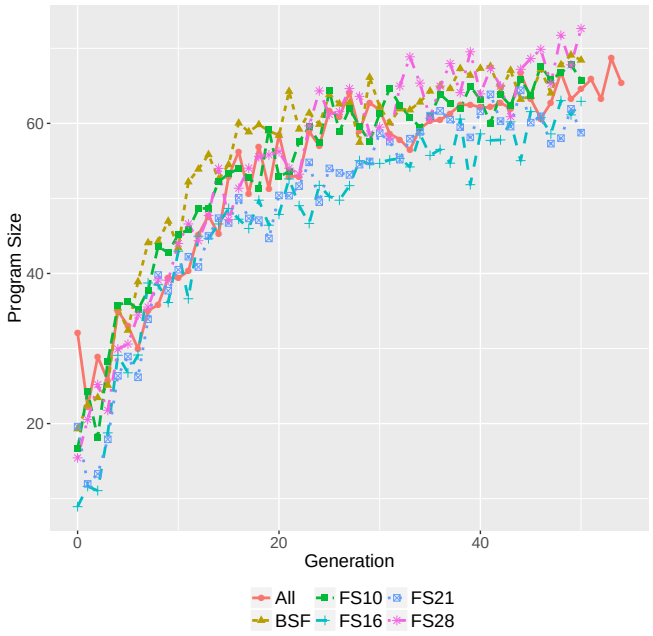


Fig. 12. The program size (number of nodes) of the best individual in each generation of the GPs with "All", "BSF", FS10, FS16, FS21 and FS28.

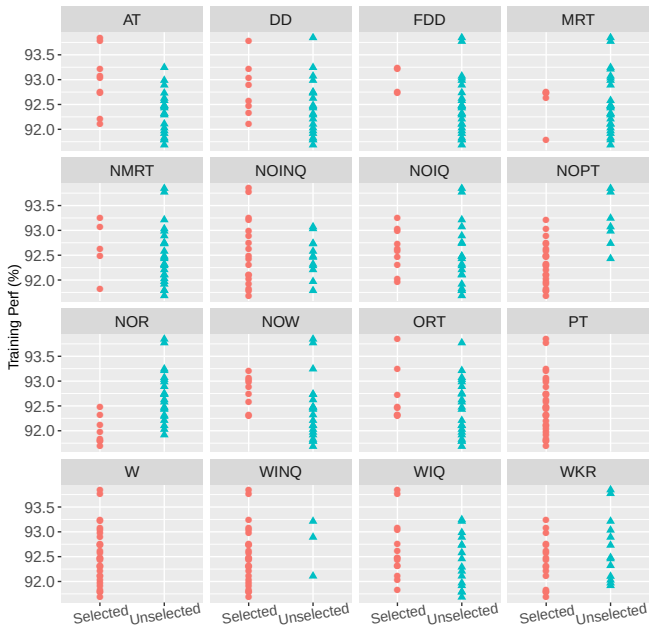


Fig. 11. The distributions of the training performances of the 30 independent runs of GP, categorised by whether each feature is selected or not.

feature sets (e.g. FS16, FS21, FS28 and "BSF") to help GP obtain simpler and shorter rules. However, this phenomenon is not obvious from Fig. 12. The reason may be that GP individuals need a large fraction of redundant branches to protect its truly useful building blocks from being destroyed regardless of the terminal set of GP. In order to evolve shorter and more understandable rules, we need more sophisticated search process such as grammar-based GP. We will do more investigation on this direction in our future work.

VI. Conclusions and Future Work

In this paper, we proposed the first *practical* feature selection algorithm to evolve dispatching rules for JSS with GP. So far, there is only one existing feature selection algorithm [34], which is computationally impractical (70 hours on power CPUs). The proposed feature selection algorithm, called NiSuFS, employs a niching-based GP framework to obtain a diverse set of good rules efficiently. In addition, it uses a surrogate model to significantly decrease the complexity of the fitness evaluation. The efficacy of NiSuFS is demonstrated by the experimental studies on the H2010 DJSS simulation [16] that minimises the mean weighted tardiness. NiSuFS is shown to be very efficient (only take less than 10% of the standard GP training time to obtain the selected features). Furthermore, the quality of the feature sets obtained by NiSuFS is verified. It has been demonstrated that about half of the feature sets selected by NiSuFS can help GP evolve significantly better rules than using the entire feature set within similar or even shorter computational time. Almost $66\%(2/3)$ of the selected features are statistically no worse than the best-so-far "BSF" feature set ({MRT, PT, NOPT, WINQ, NOINQ, W}) in terms of the test performance of the rules evolved by GP by using that feature set as terminals. Moreover, NiSuFS managed to identify feature sets (e.g. {PT, NOPT, WINQ, NOINQ, W} and {PT, NOPT, WKR, NOR, WINQ, NOINQ, W}) that are significantly better than the best-so-far "BSF" feature set.

Further analysis shows that a better feature set than "BSF" can lead to both better test performance and generalisation (difference between training and test performances) of the rules evolved by GP, as well as a consistently better convergence curve. In other words, GP can achieve both faster

convergence speed and better final solution than "BSF" when using a better feature set (e.g. {PT, NOPT, WINQ, NOINQ, W}). Finally, we observed that simply using a more compact feature set cannot effectively reduce the program size and complexity of the evolved rules.

In the future, more investigations will be conducted on further improving the accuracy of the feature selection. NiSuFS sometimes failed to identify all the important features (e.g. NOPT and WINQ in the experimental studies), and the corresponding selected feature set led to a significantly worse performance than other selected feature sets (and the best-so-far "BSF" set). A possible future direction is to transform the output of feature selection from binary values (select or not) to the probability of selecting each feature in the terminal set. Then, one can adaptively change the terminal set according to the individuals in the current population. In addition, we will verify our feature selection method on more job shop scenarios with other objectives (e.g. makespan and total flowtime).

REFERENCES

[1] P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1):217–239, 2005.

[2] J. H. Blackstone, D. T. Phillips, and G. L. Hogg. A state-of-the-art survey of dispatching rules for manufacturing job shop operations. *The International Journal of Production Research*, 20(1):27–45, 1982.

[3] J. Branke, T. Hildebrandt, and B. Scholz-Reiter. Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. *Evolutionary Computation*, 23(2):249–277, 2015.

[4] J. Branke, S. Nguyen, C. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124, 2016.

[5] E. 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*, 64(12):1695–1724, 2013.

[6] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automating the packing heuristic design process with genetic programming. *Evolutionary computation*, 20(1):63–89, 2012.

[7] J. Ceberio, E. Irurozki, A. Mendiburu, and J. A. Lozano. A distance-based ranking model estimation of distribution algorithm for the flow-shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 18(2):286–300, 2014.

[8] Q. Chen, B. Xue, and M. Zhang. Improving generalisation of genetic programming for high-dimensional symbolic regression with feature selection. In *IEEE Congress on Evolutionary Computation*. IEEE, 2016.

[9] A. Friedlander, K. Neshatian, and M. Zhang. Meta-learning and feature ranking using genetic programming for classification: Variable terminal weighting. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, pages 941–948. IEEE, 2011.

[10] C. Geiger and R. Uzsoy. Learning effective dispatching rules for batch processor scheduling. *International Journal of Production Research*, 46(6):1431–1454, 2008.

[11] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49. Hillsdale, NJ: Lawrence Erlbaum, 1987.

[12] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[13] G. R. Harik. Finding multimodal solutions using restricted tournament selection. In *Proceedings of International Conference on Genetic Algorithms*, pages 24–31, 1995.

[14] R. Haupt. A survey of priority rule-based scheduling. *Operations-Research-Spektrum*, 11(1):3–16, 1989.

[15] T. Hildebrandt and J. Branke. On using surrogates with genetic programming. *Evolutionary computation*, 23(3):343–367, 2015.

[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 Genetic and Evolutionary Computation Conference*, pages 257–264. ACM, 2010.

[17] O. Holthaus and C. Rajendran. Efficient dispatching rules for scheduling in a job shop. *International Journal of Production Economics*, 48(1):87–105, 1997.

[18] O. Holthaus and C. Rajendran. Efficient jobshop dispatching rules: further developments. *Production Planning & Control*, 11(2):171–178, 2000.

[19] J. Horn, N. Nafpliotis, and D. E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 82–87. IEEE, 1994.

[20] R. Hunt, M. Johnston, and M. Zhang. Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In *Proceedings of the 2014 conference on Genetic and evolutionary computation*, pages 927–934. ACM, 2014.

[21] D. Jakobović and L. Budin. Dynamic scheduling with genetic programming. In *Genetic Programming*, pages 73–84. Springer, 2006.

[22] D. Jakobović and K. Marasović. Evolving priority scheduling heuristics with genetic programming. *Applied Soft Computing*, 12(9):2781–2789, 2012.

[23] M. Jayamohan and C. Rajendran. New dispatching rules for shop scheduling: a step forward. *International Journal of Production Research*, 38(3):563–586, 2000.

[24] Y. Jin. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation*, 1(2):61–70, 2011.

[25] A. S. Kiran and M. L. Smith. Simulation studies in job shop sheduling—i a survey. *Computers & Industrial Engineering*, 8(2):87–93, 1984.

[26] J. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[27] S. Kreipl. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 3(3):125–138, 2000.

[28] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502, 2005.

[29] S. Luke et al. A java-based evolutionary computation research system. https://cs.gmu.edu/~eclab/projects/ecj/.

[30] S. W. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1995.

[31] M. K. Marichelvam, T. Prabaharan, and X. S. Yang. A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *IEEE transactions on evolutionary computation*, 18(2):301–305, 2014.

[32] Y. Mei, B. Xue, and M. Zhang. Fast bi-objective feature selection using entropy measures and bayesian inference. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 469–476. ACM, 2016.

[33] Y. Mei and M. Zhang. A comprehensive analysis on reusability of gp-evolved job shop dispatching rules. In *Proceedings of Congress on Evolutionary Computation*. IEEE, 2016.

[34] Y. Mei, M. Zhang, and S. Nyugen. Feature selection in evolving job shop dispatching rules with genetic programming. In *Proceedings of Genetic and Evolutionary Computation Conference*, pages 365–372. ACM, 2016.

[35] B. L. Miller and M. J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 786–791. IEEE, 1996.

[36] S. Nguyen, M. Zhang, M. Johnston, and K. Tan. A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. *IEEE Transactions on Evolutionary Computation*, 17(5):621–639, 2013.

[37] S. Nguyen, M. Zhang, M. Johnston, and K. Tan. Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. *IEEE Transactions on Evolutionary Computation*, 18(2):193–208, 2014.

[38] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan. Dynamic multi-objective job shop scheduling: A genetic programming approach. In *Automated Scheduling and Planning*, pages 251–282. Springer, 2013.

[39] 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*, pages 1–15, DOI: 10.1109/TCYB.2016.2562674, 2016.

[40] S. Ok, K. Miyashita, and S. Nishihara. Improving performance of gp by adaptive terminal selection. In *PRICAI 2000 Topics in Artificial Intelligence*, pages 435–445. Springer, 2000.

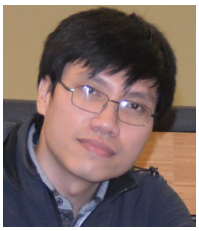
[41] A. Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 798–803. IEEE, 1996.

[42] C. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter. Evolutionary generation of dispatching rule sets for complex dynamic

scheduling problems. *International Journal of Production Economics*, 145(1):67–77, 2013.
[43] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.
[44] B. Qu, J. J. Liang, and P. N. Suganthan. Niching particle swarm optimization with local search for multi-modal optimization. *Information Sciences*, 197:131–143, 2012.
[45] C. Rajendran and O. Holthaus. A comparative study of dispatching rules in dynamic flowshops and jobshops. *European Journal of Operational Research*, 116(1):156–170, 1999.
[46] R. Ramasesh. Dynamic job shop scheduling: a survey of simulation research. *Omega*, 18(1):43–57, 1990.
[47] M. Riley, Y. Mei, and M. Zhang. Feature selection in evolving job shop dispatching rules with genetic programming. In *IEEE Congress on Evolutionary Computation*. IEEE, 2016.
[48] J. Rosca and D. Ballard. Genetic programming with adaptive representations. Technical report, University of Rochester, Computer Science Department, Rochester, NY, USA, 1994.
[49] B. Sareni and L. Krähenbühl. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106, 1998.
[50] V. Sels, N. Gheysen, and M. Vanhoucke. A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270, 2012.
[51] V. Subramaniam, T. Ramesh, G. Lee, Y. Wong, and G. Hong. Job shop scheduling with dynamic fuzzy selection of dispatching rules. *The International Journal of Advanced Manufacturing Technology*, 16(10):759–764, 2000.
[52] J. Tay and N. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.
[53] K. Veeramachaneni, O. Derby, D. Sherry, and U.-M. O'Reilly. Learning regression ensembles with genetic programming at scale. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1117–1124. ACM, 2013.
[54] H. Wang, Y. Jin, and J. O. Janson. Data-driven surrogate-assisted multi-objective evolutionary optimization of a trauma system. *IEEE Transactions on Evolutionary Computation*, 20(6):939–952, 2016.
[55] J. Xiong, J. Liu, Y. Chen, and H. A. Abbass. A knowledge-based evolutionary multiobjective approach for stochastic extended resource investment project scheduling problems. *IEEE Transactions on Evolutionary Computation*, 18(5):742–763, 2014.
[56] B. Xue, M. Zhang, W. Browne, and X. Yao. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 20(4):606–626, 2016.
[57] B. Xue, M. Zhang, and W. N. Browne. Particle swarm optimization for feature selection in classification: A multi-objective approach. *IEEE Transactions on Cybernetics*, 43(6):1656–1671, 2013.
[58] B. Yuan, B. Li, T. Weise, and X. Yao. A new memetic algorithm with fitness approximation for the defect-tolerant logic mapping in crossbar-based nanoarchitectures. *IEEE Transactions on Evolutionary Computation*, 18(6):846–859, 2014.
[59] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.
[60] Q. Zhang and H. Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

Yi Mei (S'09–M'13) is a Lecturer at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand. He received his BSc and PhD degrees from University of Science and Technology of China in 2005 and 2010, respectively. His research interests include evolutionary computation in scheduling, routing and combinatorial optimisation, as well as evolutionary machine learning, genetic programming, feature selection and dimensional reduction.

Dr Mei has more than 50 fully referred publications, including the top journals in EC and Operations Research (OR) such as IEEE TEVC, IEEE Transactions on Cybernetics, European Journal of Operational Research, ACM Transactions on Mathematical Software. He is an Editorial Board Member of International Journal of Bio-Inspired Computation. He currently serves as a Vice-Chair of the IEEE CIS Emergent Technologies Technical Committee, and a member of three IEEE CIS Task Forces. He is a guest editor of a special issue of the Genetic Programming Evolvable Machine journal. He serves as a reviewer of over 25 international journals including the top journals in EC and OR.



Su Nguyen received the B.E. degree in Industrial and Systems Engineering from the Ho Chi Minh City University of Technology, Vietnam, in 2006, the M.E. degree in Industrial Engineering and Management from the Asian Institute of Technology (AIT), Bangkok, Thailand, in 2008, and the Ph.D. degree in Operations Research and Data Analytics from Victoria University of Wellington (VUW), Wellington, New Zealand, in 2013. He is currently a David Myers Research Fellow attached to the Centre for Research in Data Analytics and Cognition, La Trobe University, Australia.

Su Nguyen has taken different research positions focusing on quantitative methods for operations management. He was a Research Associate in Industrial and Manufacturing Engineering at the School of Engineering and Technology, AIT from 2009 to 2010 and the Research Assistant at VUW from 2011 to 2013. He was a postdoctoral research fellow at VUW from 2013 to 2016, focusing on automated design of production scheduling heuristics. From 2014 to 2016, he was also the lecturer at International University, VNU-HCMC and Hoa Sen University in Vietnam. His primary research interests include evolutionary computation, optimization, data analytics, large-scale simulation, and their applications in operations management. Su Nguyen is a member of IEEE and IEEE Computational Intelligence Society and the Chair of IEEE Task Force on Evolutionary Scheduling and Combinatorial Optimization.

Bing Xue (M'10) received the B.Sc. degree from the Henan University of Economics and Law, Zhengzhou, China, in 2007, the M.Sc. degree in management from Shenzhen University, Shenzhen, China, in 2010, and the PhD degree in computer science in 2014 at Victoria University of Wellington, New Zealand. She is currently a Senior Lecturer in School of Engineering and Computer Science at Victoria University of Wellington. Her research focuses mainly on evolutionary computation, feature selection, feature construction, multi-objective optimisation, data mining and machine learning.

Dr Xue is an Associate Editor/member of Editorial Board for five international journals including IEEE Computational Intelligence Magazine, Applied Soft Computing, International Journal of Swarm Intelligence, and International Journal of Computer Information Systems and Industrial Management Applications. She is a Guest Editor for the Special Issue on Evolutionary Feature Reduction and Machine Learning for the Springer Journal of Soft Computing. She is also a Guest Editor for Evolutionary Image Analysis and Pattern Recognition in Journal of Applied Soft Computing.

She has also been a program chair, special session chair, tutorial chair, symposium chair, and publicity chair for a number of international conferences. She is serving as a reviewer of over 20 international journals and a program committee member for over 50 international conferences. She is currently the Chair of the IEEE Task Force on Evolutionary Feature Selection and Construction. She is chairing the IEEE CIS Graduate Student Research Grants Committee and the Secretary of the IEEE Computational Intelligence Chapter in New Zealand.

Mengjie Zhang (M'04–SM'10) received the B.E. and M.E. degrees from Artificial Intelligence Research Center, Agricultural University of Hebei, Hebei, China, and the Ph.D. degree in computer science from RMIT University, Melbourne, VIC, Australia, in 1989, 1992, and 2000, respectively.

Since 2000, he has been with the Victoria University of Wellington, Wellington, New Zealand, where he is currently Professor of Computer Science, Head of the Evolutionary Computation Research Group, and the Associate Dean (Research and Innovation) in the Faculty of Engineering. His current research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, feature selection and reduction, job shop scheduling, and transfer learning. He has published over 350 research papers in refereed international journals and conferences.

Prof. Zhang has been serving as an Associated Editor or Editorial Board Member for ten international journals (including IEEE Transactions on Evolutionary Computation, IEEE Transactions on Cybernetics, Evolutionary Computation Journal, and IEEE Transactions on Emergent Topics in CI) and as a Reviewer of over 20 international journals. He has been serving as a Steering Committee Member and a Program Committee Member for over 100 international conferences. He has supervised over 50 postgraduate research students. He is the Chair of the IEEE CIS Intelligent Systems and Applications Technical Committee, a member of the IEEE CIS Evolutionary Computation Technical Committee, a Vice-Chair of the IEEE CIS Task Force on Evolutionary Computer Vision and Image Processing, a Vice-Chair of the IEEE CIS Task Force on Evolutionary Computation for Feature Selection and Construction, a member of IEEE CIS Task Force of Hyper-heuristics, and the Founding Chair for IEEE Computational Intelligence Chapter in New Zealand.