Feature Selection in Evolving Job Shop Dispatching Rules with Genetic Programming

Yi Mei
School of Engineering and CS
Victoria University of Wellington
Wellington, New Zealand
yi.mei@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering and CS
Victoria University of Wellington
Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

Su Nyugen
Dep. of Business Admin.
Hoa Sen University
Ho Chi Minh City, Vietnam
nguyenphanbachsu@gmail.com

ABSTRACT

Genetic Programming (GP) has been successfully used to automatically design dispatching rules in job shop scheduling. The goal of GP is to evolve a priority function that will be used to order the waiting jobs at each decision point, and decide the next job to be processed. To this end, the proper terminals (i.e. job shop features) have to be decided. When evolving the priority function, various job shop features can be included in the terminal set. However, not all the features are helpful, and some features are irrelevant to the rule. Including irrelevant features into the terminal set enlarges the search space, and makes it harder to achieve promising areas. Thus, it is important to identify the important features and remove the irrelevant ones to improve the GP-evolved rules. This paper proposes a domain-knowledge-free feature ranking and selection approach. As a result, the terminal set is significantly reduced and only the most important features are selected. The experimental results show that using only the selected features can lead to significantly better GP-evolved rules on both training and unseen test instances.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Scheduling*

Keywords

Combinatorial Optimization, Job Shop Scheduling, Genetic Programming, Feature Selection

1. INTRODUCTION

Job Shop Scheduling (JSS) [20] is an important scheduling problem, and has many applications in the real world such as the manufacturing industries and resource allocation in cloud computing. Briefly speaking, JSS is to assign the order of a set of jobs on one or more machines to achieve some desired goals such as minimizing the makespan or mean flowtime.

In JSS, the environment can be *static* or *dynamic*. In a static environment, all the jobs are assumed to have arrived at the beginning of the scheduling horizon, and the information such as the number of operations, machine orders and processing time is known in advance. This situation is often considered when the scheduling plan is made periodically, e.g. weekly based on the orders arrived during the previous week. In a dynamic environment, the jobs can arrive randomly over time. For each job, the information can only be known upon its arrival. Although the dynamic environment is closer to reality than the static one, it is harder to solve as it is required to change to current schedule to adapt environment change such as new job arrivals.

Dispatching rules have been widely used in real-world JSS problems due to their simplicity, ease of application, and instantaneous response to the environment change. To put it simply, a dispatching rule can be represented as a priority function. During the scheduling horizon, whenever a machine becomes idle and there are operations waiting to be processed in its queue, then the priority value of each waiting operation is calculated by the priority function, and the operation with the best (largest or smallest) priority value is selected to be processed next. There have been a large number of manually designed dispatching rules so far. A comprehensive comparison between manual dispatching rules can be found in [23].

As shown in literatures (e.g. [12, 22, 24]), the manually designed dispatching rules are still not good enough, and their performance depends highly on the job shop situation and the objective to be optimized. In recent years, Genetic Programming (GP) has attracted more and more research interests for automatic design of dispatching rules. There have been a number of works proposed on evolving dispatching rules with GP (e.g. [4, 6, 8, 15, 19]), which successfully obtained much better rules than the manually designed rules. Therefore, in this paper, we also focus on evolving dispatching rules with GP.

When using GP to evolve good dispatching rules effectively, one important factor is the terminals to be used. So far, a large number of job shop features have been considered in different scenarios for optimizing different objectives. The commonly used features include global shop-level features (e.g. current time), job-related features (e.g. processing time, due date, waiting time) and machine-related features (e.g. machine ready time, work in the queue, uti-

GECCO '16, July 20-24, 2016, Denver, CO, USA

DOI: http://dx.doi.org/10.1145/2908812.2908822

lization level). Obviously, some of the features may be irrelevant with the performance of the rules. For example, if the objective is to minimize the mean flowtime, the due date of the jobs tends to be useless or even misleading [6]. Similarly, the weight of the jobs is supposed to be only useful if the objective is weighted, such as the mean weighted tardiness.

With such a large number of features, how to select the best terminal set for GP becomes an important issue. On the one hand, including irrelevant features will enlarge the search space, and make it harder to reach the promising areas. On the other hand, ignoring important features will rule out the promising areas in the search space. The same issue happens in features selection in machine learning, in which it has been shown that removing irrelevant features can help improve the classification performance.

In this paper, we aim to investigate how to select the best terminal set for GP without adopting domain knowledge. This is motivated by the reality in which any unknown shop scenario (e.g. static, dynamic, flexible job shop, etc) and objective can occur. In summary, the goals in this paper can be listed as below.

1. Design a feature selection method for selecting the terminal set of GP that does not require any domain knowledge;
2. Employ the designed feature selection method to determine the important features from all the possible features for the representative problems including minimizing makespan in static environment, and mean flowtime and mean weighted tardiness in dynamic environment;
3. Verify the correctness of the selected features by comparing the GP with the terminal sets before and after the feature selection.

The rest of the paper is organized as follows: Section 2 gives the background introduction, including the problem description, automatic dispatching rule design and feature selection. Section 3 describes the proposed feature selection method for selecting the key terminal set for GP in JSS. Section 4 presents the experimental studies in different shop scenarios and objectives. Then, Section 5 gives the conclusions and future work.

2. BACKGROUND

2.1 Job Shop Scheduling

A Job Shop Scheduling (JSS) problem consists of a number of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ and machines $\mathcal{M} = \{M_1, \ldots, M_m\}$. Each job J_j has a sequence of l_j ($l_j \leq m$) operations $\mathbf{O}_j = (O_{1j}, \ldots, O_{l_j,j})$, as well as an release time $t_0(J_j)$ and a due date $\rho(J_j)$. Each operation O_{ij} has an eligible machine $\pi(O_{ij}) \in \mathcal{M}$ and a processing time $\delta(O_{ij})$. In a standard JSS problem, a schedule is to be made under the following constraints:

1. For each job, the order of the operations needs to be strictly followed. That is, the starting time of $O_{i+1,j}$ cannot be earlier than the finishing time of O_{ij}, $\forall\ j = 1, \ldots, n$. In addition, the starting time of O_{1j} cannot be earlier than the corresponding release 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. once an operation starts to be processed, its processing cannot be paused or stopped until it has been finished.

In JSS, a schedule X can be represented as a set of processing sequences, each for a machine. The processing sequence $\mathbf{P}(M_k)$ consists of the processings of all the operations $\{O_{ij} \mid \pi(O_{ij}) = M_k\}$, along with the starting time $\tau(O_{ij})$. The commonly used objectives include minimizing the makespan ($C_{\max}$), total flowtime ($\sum C_j$) and total weighted tardiness ($\sum w_j T_j$).

2.2 Automatic Design of Dispatching Rules

Dispatching rules are often used for constructing the job shop schedule on-the-fly, especially in the dynamic environment where new jobs keep arriving over time. Specifically, at a decision point in which at least one machine is idle and there are operations waiting to be processed on the idle machines, the dispatching rule is applied to calculate the priority of each waiting operation. Then, the operation with the highest priority is selected to be processed next. For example, in the well-known Shortest Processing Time (SPT) rule, the operation with the shortest processing time has the highest priority.

With respect to the definition of decision points, there are two main categories of dispatching rules, namely the *nondelay* rules and *active* rules. In nondelay rules, once a machine becomes idle and its waiting queue is not empty, a decision has to be made immediately and no delay is allowed. In contrast, in active rules, the decision can be delayed to some extent. In this paper, we only focus on the nondelay rules, since it is simpler to use, and has achieved success in many previous studies (e.g. [6, 15]). In this case, a dispatching rule is essentially a priority function.

There have been a variety of manually designed rules (e.g. [7, 10, 21]). However, their performance is not satisfactory enough, and depends on the shop scenario and objective to be optimized. In practice, it is difficult to choose the proper rule that most suits the given scenario.

Automatic design of dispatching rule with hyper-heuristics [2] is attracting more and more research interests recently as it is capable of searching in the space of dispatching rules. Being a hyper-heuristic, Genetic Programming (GP) [13] has been extensively adopted to evolve more flexible priority functions, and has achieved great success in dynamic job shop scheduling problems ([4, 6, 9, 15, 16, 19, 26]).

2.3 Feature Selection

Feature selection [5, 27] is a well-known problem in data mining and machine learning. It aims to reduce the number of features or attributes in the given data to remove redundant information, and thus to save the training time and cost to collect the feature values, and improve the classification accuracy. There have been more sophisticated feature selection methods such as the filter methods [18] and wrapper methods [28]. Xue et al. [27] gave a comprehensive survey on evolutionary computation approaches to feature selection.

In GP, Ok et al. [17] proposed an adaptive terminal selection scheme and corresponding adaptive mutation based on the updated terminal set. Friedlander et al. [3] proposed a

feature ranking method based on frequency analysis. However, one of the drawbacks of frequency analysis is that there may be many irrelevant occurrences of the features. For example, in the rule $(A-B)/(A-B)+C/D$, both features A and B occurred twice, which is more than C and D. However, the rule is $C/D+1$, and both A and B are actually irrelevant.

In this paper, a new feature ranking and selection method for evolving JSS dispatching rules is proposed. The method directly estimates the contribution of each feature to the performance of dispatching rules rather than the frequency. The details of the proposed approach is described in the next section.

3. A NEW FEATURE SELECTION METHOD FOR GENETIC PROGRAMMING IN JOB SHOP SCHEDULING

For the sake of convenience, we list all the useful notations in Table 1.

Table 1: The useful notations.

Notation	Description
$\vec{x}$	The vector of features.
$x_i \in \vec{x}$	A feature.
$g(\vec{x})$	The priority function of a dispatching rule.
I	A JSS instance.
$S(g(\vec{x}); I)$	The schedule obtained by applying $g(\vec{x})$ to I.
$f(S)$	The value of objective $f(\cdot)$ of the schedule S.
$\phi_{\mathcal{I}}(g(\vec{x}))$	The fitness of $g(\vec{x})$ on an instance set $\mathcal{I}$.

In the table, the fitness $\phi_{\mathcal{I}}(g(\vec{x}))$ of a priority function $g(\vec{x})$ in terms of the objective $f(\cdot)$ on a set of instances $\mathcal{I}$ is generally defined as the average normalized objective value over the instances. That is,

$$\phi_{\mathcal{I}}(g(\vec{x})) = \frac{1}{|\mathcal{I}|}\sum_{I\in\mathcal{I}}\frac{f(S(g(\vec{x}); I))}{\hat{f}(I)}, \tag{1}$$

where $\hat{f}(I)$ is the reference objective value of the instance I. Ideally, $\hat{f}(I)$ should be the optimal objective value. In practice, since the optimal value is usually unknown, it can be set to the (1) theoretical lower bound, (2) best-known result, or (3) the objective value of some reference rule such as SPT, EDD and FDD.

Before describing the proposed feature selection method, we first define the *feature removal in priority function* as follows.

DEFINITION 1. *Given a priority function $g(\vec{x})$, where $\vec{x} = (x_1, \ldots, x_n)$ is the vector of features, then the removal of feature x_i from $g(\vec{x})$ is defined as fixing x_i to 1 in $g(\vec{x})$.*

For the sake of convenience, let $g(\vec{x} \setminus x_i)$ be the priority function after removing x_i, then

$$g(\vec{x} \setminus x_i) = g(\vec{x}|x_i = 1). \tag{2}$$

For example, for the PT+WINQ rule, after removing the WINQ feature, the priority function becomes PT+1. Note that the priority function is only used to sort the waiting operations. Thus, including any constant in the priority function will not change the order of the waiting operations, and thus not change the final solution. Therefore, the rule with the priority function of (PT+1) is equivalent to the SPT rule.

Here, the fixed value of 1 is chosen as a rule of thumb. Note that the basic arithmetic operators include addition, subtraction, multiplication and protected division (returning 1 for $a/0$), fixing the variables to 1 is suitable for multiplication and protected division, since $a \times 1 = a$, and $a/1 = a$. Obviously, $a+1$ and $a-1$ give the same order of the waiting operations as a. Thus, the value of 1 is also suitable for addition and subtraction.

Given a priority function $g(\vec{x})$, after removing a specific feature x_i, the following three possible situations may happen.

1. If x_i is not a variable of $g(\vec{x})$, then $g(\vec{x})$ keeps the same after the removal;
2. If x_i is one of the irrelevant variables of $g(\vec{x})$, then $g(\vec{x})$ is simplified after the removal, but is essentially the same;
3. If x_i is one of the relevant variables of $g(\vec{x})$, then $g(\vec{x})$ is changed after the removal.

To illustrate this, suppose we have a rule whose priority function is $g(A, B, C, D, E) = (A-B)/(A-B)+C/D$. First, feature E is not a variable of the priority function. Then the priority function will be the same after removing E. Second, since $g(A, B, C, D) = C/D+1$, which is equivalent to C/D, features A and B are irrelevant, and removing them will simplify the priority function without changing its behavior. Finally, the priority function will be changed after removing C or D.

Based on the above discussions, we define the *contribution* $\zeta_{\mathcal{I}}(x_i; g(\vec{x}))$ of a feature x_i to a priority function $g(\vec{x})$ on a set of JSS instances $\mathcal{I}$ as follows.

$$\zeta_{\mathcal{I}}(x_i; g(\vec{x})) = \phi_{\mathcal{I}}(g(\vec{x} \setminus x_i)) - \phi_{\mathcal{I}}(g(\vec{x})). \tag{3}$$

In other words, the contribution of the feature x_i to the priority function $g(\vec{x})$ is the difference between the fitness values before and after removing x_i from $g(\vec{x})$. A positive value of $\zeta_{\mathcal{I}}(x_i; g(\vec{x}))$ indicates that after removing x_i, the new dispatching rule will generate worse schedules for the tested JSS instances. Furthermore, a larger value implies that the performance of the dispatching rule deteriorates more after removing the feature x_i, i.e. x_i makes more contribution to $g(\vec{x})$. Similarly, a negative $\zeta_{\mathcal{I}}(x_i; g(\vec{x}))$ value implies that x_i makes a negative contribution to $g(\vec{x})$, and removing it can lead to an improvement on the performance. If $\zeta_{\mathcal{I}}(x_i; g(\vec{x})) = 0$, then x_i makes no contribution, and is irrelevant.

Based on Eq. (3), we can define the *relevance* $\Upsilon_{\Omega}(x_i)$ of a feature x_i for evolving dispatching rules under the given shop *scenario* Ω to be optimized. Here, a shop scenario stands for a certain job shop model and objective. For example, the static job shop with the objective of makespan is one scenario, and the dynamic job shop with the objective of mean flowtime is another scenario. A scenario consists of infinite instances.

Intuitively, if a feature x_i is more relevant, then it tends to make more contribution to the rules that perform well in the shop scenario. Therefore, $\Upsilon_{\Omega}(x_i)$ is defined based on the contribution to the best rules found by GP. To increase our confidence, we calculate $\Upsilon_{\Omega}(x_i)$ based on multiple runs instead of a single run as follows.

Step 1. Select a set of training instances $\mathcal{I}_{\text{train}} \subseteq \Omega$;

Step 2. Apply GP to evolve dispatching rules using all the features $\vec{x}$ as the terminal set;

Step 3. Conduct λ independent GP runs, and obtain λ best rules $g_1^*(\vec{x}), \ldots, g_\lambda^*(\vec{x})$, each for a single run;

Step 4. For each rule $g_k^*(\vec{x})$ and each feature $x_i \in \vec{x}$, calculate the contribution $\zeta_{\mathcal{I}_{\text{train}}}(x_i, g_k^*(\vec{x}))$ of x_i to $g_k^*(\vec{x})$ by Eq. (3);

Step 5. Set $\Upsilon_\Omega(x_i)$ to the *median* value of all the $\zeta_{\mathcal{I}_{\text{train}}}(x_i, g_k^*(\vec{x}))$ values ($k = 1, \ldots, \lambda$).

The number of runs λ is user-defined, and we set it to 30 here as a rule of thumb. The relevance value of each feature is set to the median of the contribution values over multiple runs (Step 5). This way, a zero relevance implies that the feature makes no contribution for at least half of the time in the best rule found by GP.

Then, one can select the key terminals to be used in GP based on the $\Upsilon_\Omega(x_i)$ for each $x_i \in \vec{x}$. For example, one can simply select the features whose relevance is larger than zero, as they make a positive contribution for at least half of the best rules found by GP.

3.1 An Offline Feature-Selection-based Genetic Programming

Based on the proposed feature ranking and selection approach, we propose an offline Feature-Selection-based GP (FS-GP). The framework of FS-GP consists of three stages. In the first stage, 30 independent runs of the standard GP with the entire feature set are conducted to get 30 best rules. Then, in the second stage, the relevance $\Upsilon_\Omega(x_i)$ of each feature x_i is calculated based on the 30 best rules, and a new key terminal set $\vec{x}^*$ is formed by selecting the features whose relevance is larger than zero. Finally, in the third stage, the standard GP with the terminal set of $\vec{x}^*$ is conducted again to obtain the new rules. The algorithm is offline, since the feature selection is carried out in the first two stages, which can be considered as a preprocessing phase.

To distinguish from the general term of GP, the GP with the Entire Terminal set is referred to as ET-GP hereafter.

4. EXPERIMENTAL STUDIES

In this section, three commonly investigated job shop scenarios [6, 15] are considered as the testbed. The first scenario is the static job shop scheduling that minimizes the makespan. The second scenario is the dynamic job shop scheduling that minimizes the mean flowtime. The third scenario is the dynamic job shop scheduling that minimizes the mean weighted tardiness. The scenarios are denoted as *sjs-ms*, *djs-mft* and *djs-mwt*, respectively.

For the sjs-ms scenario, the Taillard benchmark [25] instances were used to form the training and test sets. Specifically, the total 80 instances were divided into two groups, one with the odd indices $(1, \ldots, 79)$, and the other with the even indices $(2, \ldots, 80)$. For the sake of convenience, the two groups are denoted as *T-odd* and *T-even*, respectively.

For the djs-mft and djs-mwt scenarios, a dynamic JSS simulation system is adopted to generated the job arrival events. In the simulation system, the key parameters include the number of machines, the number of jobs before stopping, the number of warmup jobs, the number of operations of each job, the arrival time and due date of each job and the eligible machine and processing time of each operation. Here, the parameter settings are described in Table 2. The setting is the same as that in [6] except that the due date factor is set to 1.3 rather than 4.0 to generate tighter due dates.

Table 2: The parameters in the dynamic JSS simulation system.

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

In the table, there are two utilization levels (0.95 and 0.8) and two settings of the number of operations (missing and full) in the training set. Thus, there are four different training configurations. Here we generate a single replication for each configuration, but change the random seed in each generation of the GP process to improve the generalizability of the rules. The training set is the same as that used by Hildebrandt et al. [6]. Thus we call it the *H2010* set. For each test configuration, 20 replications were generated to reflect the real distribution better.

Given the above parameter settings, we created a set of experiments whose training and test sets are described in Table 3.

Table 3: The designed experiments.

Experiment	Training set	Test set
sjs-ms-1	T-odd (40 instances)	T-even (40 instances)
sjs-ms-2	T-even (40 instances)	T-odd (40 instances)
djs-mft-1	H2010 (4 replications)	0.95-missing (20 replications)
djs-mft-2	H2010 (4 replications)	0.9-missing (20 replications)
djs-mft-3	H2010 (4 replications)	0.85-missing (20 replications)
djs-mft-4	H2010 (4 replications)	0.8-missing (20 replications)
djs-mft-5	H2010 (4 replications)	0.95-full (20 replications)
djs-mft-6	H2010 (4 replications)	0.9-full (20 replications)
djs-mft-7	H2010 (4 replications)	0.85-full (20 replications)
djs-mft-8	H2010 (4 replications)	0.8-full (20 replications)
djs-mwt-1	H2010 (4 replications)	0.95-missing (20 replications)
djs-mwt-2	H2010 (4 replications)	0.9-missing (20 replications)
djs-mwt-3	H2010 (4 replications)	0.85-missing (20 replications)
djs-mwt-4	H2010 (4 replications)	0.8-missing (20 replications)
djs-mwt-5	H2010 (4 replications)	0.95-full (20 replications)
djs-mwt-6	H2010 (4 replications)	0.9-full (20 replications)
djs-mwt-7	H2010 (4 replications)	0.85-full (20 replications)
djs-mwt-8	H2010 (4 replications)	0.8-full (20 replications)

4.1 Parameter Setting of Genetic Programming

First, the terminal set of GP is to be determined. An overview of the promising features for generating priority functions is given in [11]. For the basic JSS model, the commonly used features are listed in Table 4.

The non-terminal 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

Table 4: The commonly used features for designing dispatching rules for the basic JSS model.

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

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.

As described in Eq. (1), the fitness of dispatching rules depends on the reference rule $\hat{g}$. Here, $\hat{g}$ is set to the FDD rule for minimzing the makespan and mean flowtime, and WATC for minimizing the mean weighted tardiness.

Then, the other parameter settings of GP is given in Table 5. The algorithm was implemented by ECJ [14], and the standard parameter settings were adopted.

Table 5: The parameters of GP.

Parameter	Value
Population size	1024
Number of generations	51
Maximal depth	8
Crossover rate	0.85
Mutation rate	0.1
Reproduction rate	0.05

4.2 Feature Relevance

First, we investigate the relevance of the features in different shop scenarios and objectives. Fig. 1 shows the relevance of each feature over the 30 independent runs on the T-odd training set in terms of makespan. One can see that WKR has the largest relevance (nearly 0.12), followed by PT, which is about 0.05. There are 8 features whose relevance is almost zero (NOIQ, WIQ, IPT, NOPT, ORT, MNRT, NOR, and W), indicating that they made no contribution to the best rule in at least half of the time.

Fig. 2 shows the relevance of the features on the T-even training set in terms of makespan. A slightly different pattern on the relevance of the features is shown. Specifically, WKR, PT and WINQ to be the most important three features. There are more irrelevant features (NOINQ and FDD) for the T-even training set. This might be because the T-even instances are simpler than the T-odd instances.

Based on Figs. 1 and 2, we select the features that are relevant for at least one of the T-odd and T-even training sets. Note that in the Taillard instances, all the jobs arrive at time 0, and the total processing time of each job is the sum of the flow due date (FDD) and the work remaining (WKR) minus the processing time. That is, DD = 1.3 × (FDD + WKR - PT). Thus, DD is considered to be redundant

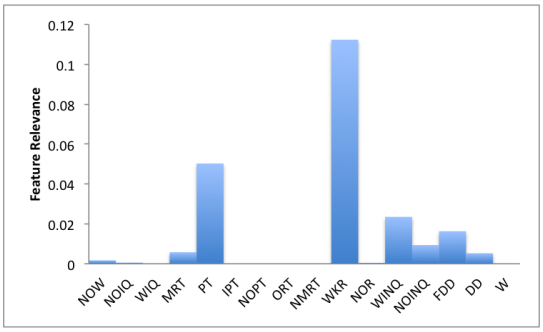

Figure 1: The relevance of each feature over the 30 independent runs of ET-GP on the *T-odd* training set in terms of makespan.

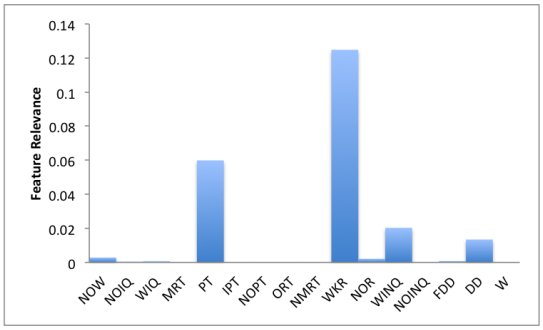

Figure 2: The relevance of each feature over the 30 independent runs of ET-GP on the *T-even* training set in terms of makespan.

when FDD, WKR and PT are all selected. In addition, the relevance of NOR is considered to be too small, and thus NOR is removed as well. As a result, we select NOW, MRT, PT, WKR, WINQ, NOINQ, and FDD for static job shop scheduling that minimizes the makespan.

Fig. 3 shows the relevance of the features for the dynamic job shop scheduling that minimizes the mean flowtime. One can see that in this scenario, only 5 out of the total 16 features made solid contribution to the best rules. PT, NOPT and WINQ are much more relevant than WKR and NOINQ. All the other 11 features are irrelevant (with zero relevance value). Therefore, we select PT, NOPT, WKR, WINQ and NOINQ in this case.

Fig. 4 shows the feature relevance for the dynamic job shop scheduling minimizing the mean weighted tardiness. In this case, MRT, PT, NOPT, WINQ, NOINQ and W are the important features to be selected, of which PT and W are the most important ones. All the other 10 features are irrelevant in terms of the definition of $\Upsilon_\Omega(x_i)$.

In summary, we have the following observations based on the results:

1. Most of the candidate features are irrelevant to the best rules (priority functions) found by ET-GP;

2. The relevance of features depends on the job shop scenario and objective to be optimized. For example, WKR and FDD are the key features when minimiz-

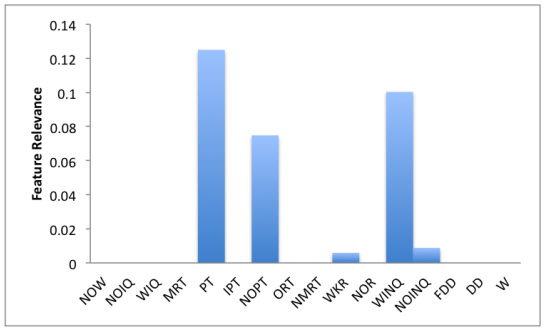

Figure 3: The relevance of each feature over the 30 independent runs of ET-GP on the H2010 training set in terms of mean flowtime.

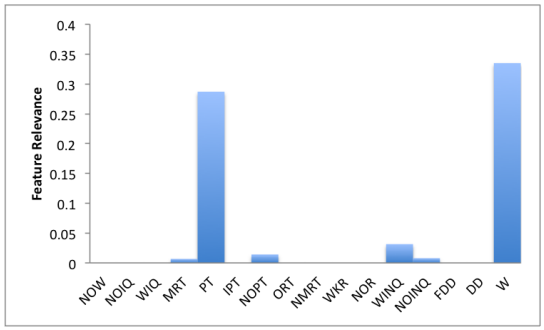

Figure 4: The relevance of each feature over the 30 independent runs of ET-GP on the H2010 training set in terms of mean weighted tardiness.

ing makespan for static job shop scheduling, while is weakly relevant or irrelevant in the case of dynamic job shop scheduling;

3. There are still some key features (e.g. PT and WINQ) that are important for all the tested scenarios.

As a result, the selected key terminal sets for the above three job shop scenarios are summarized in Table 6.

Table 6: The key terminal set for the three job shop scenarios.

Scenario	Key Terminal Set
sjs-ms	NOW, MRT, PT, WKR, WINQ, NOINQ, FDD
djs-mft	PT, NOPT, WKR, WINQ, NOINQ
djs-mwt	MRT, PT, NOPT, WINQ, NOINQ, W

4.3 Results with Key Terminal Sets

After obtaining the key terminal sets, we conduct GP again to evolve rules that only consists of the key terminals. By using fewer terminals, the evolved rules are expected to be simpler, more understandable and generalizable. As described in Section 3.1, the resultant GP is referred to as FS-GP.

Tables 7–9 show the test performance of ET-GP, FS-GP and some other representative benchmark rules in the three investigated job shop scenarios. The test performance of a rule A is defined as the percentage deviation from the reference value, i.e. $100 \cdot (\phi_{\mathcal{I}_{\text{test}}}(A) - 1)$, where $\phi_{\mathcal{I}_{\text{test}}}(A)$ is defined in Eq. (1). For sjs-ms, the reference makespan values of the Taillard instances were obtained from the lower bounds given on the website of JSS [1]. For djs-mft and djs-mwt, the reference values were set to the test performance of the FDD and WATC rules, respectively. In the tables, Wilcoxon rank sum test was conducted between the results of ET-GP and FS-GP, and the statistically significantly better one is marked in bold.

From the tables, one can see that both ET-GP and FS-GP achieved much better results than the compared benchmark rules, which demonstrates the advantage of automatically evolving dispatching rules using GP (negative value indicates an advantage over the reference rule). In addition, FS-GP performed significantly better than ET-GP on most of the experiments (sjs-ms-2, 6 out of the total 8 djs-mft experiments and all the 8 djs-mwt experiments). This shows that selecting only the key features in GP can significantly improve the test performance of the GP-evolved rules.

Since FS-GP achieved better test performance than ET-GP, it is interesting to know whether such advantage is caused by better training performance or better generalizability (difference between training and test performances). To this end, Table 10 shows the average training and test performance of ET-GP and FS-GP in the three experiment sets. For the djs-mft and djs-mwt experiments, we combined the 8 experiments together, and only reported the mean test performance, since they used the same training set, and thus obtained the same rules. It can be seen that the better test performance of FS-GP is due to the better training performance. This indicates that using a smaller number of terminals can reduce the search space, and restrict the search within more promising areas.

4.4 Analysis on the Evolved Dispatching Rules

To investigate the complexity and interpretability of the GP-evolved rules, we first simplify the rules according to the following principles.

- Replace $a+0$ $(0+a)$, $a-0$, $a \times 1$ $(1 \times a)$, $a/1$, $\min(a, a)$, $\max(a, a)$, $\text{ifte}(*, a, a)$ with a;
- Replace $a - a$ with 0, replace $a/0$ with 1;
- Replace a/IPT with $a \times \text{PT}$, replace $\text{PT} \times \text{IPT}$ $(\text{IPT} \times \text{PT})$ with 1, replace $a \times \text{IPT}$ $(\text{IPT} \times a)$ with a/PT;
- Replace $\text{ifte}(a, b, c)$ with b if a is always positive (e.g. PT, W, WKR), and with c if a is always non-positive (e.g. $0-\text{PT}$);

After the above simplification, we analyze the rules by calculating their depth, size (number of nodes) and leaves (number of terminals). Table 11 gives the mean and standard deviation of the depth, size and leaves of the best rules obtained by the 30 independent runs of ET-GP and FS-GP in the djs-mwt experiment. Surprisingly, it can be seen that FS-GP seems to obtain more complex rules than ET-GP, with higher values on depth, size and leaves.

Eqs. (4) and (5) shows two example rules obtained by ET-GP and FS-GP in the djs-mwt experiments, respectively. It can be seen that the two rules have similar complexity after the simplification. However, g_1 adopts many features

Table 7: The test performance (% relative to the lower bounds) of the GP-evolved rules with and without terminal selection along with the benchmark rules in the static job shop scenario that minimizes the makespan.

Experiment	SPT	OPFSLK/PT;FDD	2PT+LWKR+FDD	2PT+WINQ+NPT	ET-GP	FS-GP
sjs-ms-1	26.88	19.98	14.81	25.38	12.48 ± 0.65	12.37 ± 0.52
sjs-ms-2	26.74	20.12	16.90	25.50	13.76 ± 0.58	**13.47 ± 0.32**

Table 8: The test performance (% relative to that of FDD) of the GP-evolved rules with and without terminal selection along with the benchmark rules for minimizing the mean flowtime of different dynamic job shop instances.

Experiment	Util.	Ops.	SPT	OPFSLK/PT;FDD	2PT+LWKR+FDD	2PT+WINQ+NPT	ET-GP	FS-GP
djs-mft-1	0.95	missing	-6.37	2.68	20.67	-19.43	-27.29 ± 0.92	-27.67 ± 0.80
djs-mft-2	0.9	missing	-6.58	5.30	24.27	-14.35	-20.67 ± 0.54	**-21.14 ± 0.67**
djs-mft-3	0.85	missing	-6.30	4.45	18.02	-9.83	-14.75 ± 0.36	**-15.11 ± 0.42**
djs-mft-4	0.8	missing	-4.24	4.26	13.04	-6.51	-10.24 ± 0.33	**-10.42 ± 0.23**
djs-mft-5	0.95	full	-4.30	3.72	24.16	-13.52	-21.90 ± 0.59	-22.15 ± 0.83
djs-mft-6	0.9	full	-4.85	5.01	22.92	-9.38	-15.28 ± 0.55	**-15.68 ± 0.55**
djs-mft-7	0.85	full	-4.09	4.49	16.13	-6.98	-11.07 ± 0.47	**-11.45 ± 0.31**
djs-mft-8	0.8	full	-3.67	3.48	11.11	-5.44	-8.40 ± 0.38	**-8.62 ± 0.22**

Table 9: The test performance (% relative to that of WATC) of the GP-evolved rules with and without terminal selection along with the benchmark rules for minimizing the mean weighted tardiness of different dynamic job shop instances.

Experiment	Util.	Ops.	WEDD	WMDD	WSPT	W(CR+SPT)	WCOVERT	ET-GP	FS-GP
djs-mwt-1	0.95	missing	30.97	4.55	0.04	-0.44	-0.52	-11.38 ± 1.58	**-12.65 ± 1.10**
djs-mwt-2	0.9	missing	39.63	8.28	0.35	0.40	0.92	-8.33 ± 1.04	**-9.12 ± 0.96**
djs-mwt-3	0.85	missing	36.49	9.27	0.37	0.32	0.27	-5.81 ± 0.85	**-6.58 ± 0.94**
djs-mwt-4	0.8	missing	33.35	10.71	0.23	0.32	0.04	-4.11 ± 0.77	**-4.76 ± 0.79**
djs-mwt-5	0.95	full	37.75	6.77	0.18	0.02	-0.32	-9.32 ± 1.46	**-10.50 ± 1.11**
djs-mwt-6	0.9	full	39.22	9.44	0.18	1.08	0.51	-6.63 ± 0.99	**-7.41 ± 1.04**
djs-mwt-7	0.85	full	34.69	10.33	-0.13	0.28	0.08	-5.10 ± 0.80	**-5.73 ± 1.02**
djs-mwt-8	0.8	full	31.98	11.24	0.08	0.38	0.01	-4.00 ± 0.72	**-4.53 ± 0.85**

Table 10: The average training and test performance of ET-GP and FS-GP for different experiment sets.

Experiment	Algorithm	Training Perf.	Test Perf.
sjs-ms-1	ET-GP	12.13 ± 0.41	12.48 ± 0.65
	FS-GP	12.07 ± 0.35	12.37 ± 0.52
sjs-ms-2	ET-GP	10.74 ± 0.43	13.76 ± 0.58
	FS-GP	10.67 ± 0.26	13.47 ± 0.32
djs-mft	ET-GP	-19.41 ± 0.60	-16.20 ± 0.52
	FS-GP	-19.66 ± 0.57	-16.53 ± 0.50
djs-mwt	ET-GP	-10.37 ± 0.96	-6.83 ± 1.03
	FS-GP	-11.05 ± 0.76	-7.66 ± 0.98

Table 11: The mean and standard deviation of the depth, size and leaves of the best rules obtained by the 30 runs of ET-GP and FS-GP in the djs-mwt experiment.

	Depth	Size	Leaves
ET-GP	7.8(0.4)	55.2(19.4)	28.8(10.2)
FS-GP	7.9(0.3)	61.2(19.5)	32.5(10.7)

that are out of the key feature set (e.g. FDD, NOIQ, NOR, NMRT). Thus, the actual part that contributes to the priority function in g_1 is much smaller than that in g_2. In addition, although looking complex, g_2 contains many meaningful building blocks such as PT/W and WINQ/W that are known to perform well for minimizing the mean weighted tardiness for dynamic JSS. As a result, the performance of g_1 is worse than that of g_2 on both the training set (-9.16 versus -10.46) and test set (e.g. -8.44 versus -10.47 in the djs-mwt-5 experiment).

$$g_1 = \text{ifte}(((\text{NOPT} - \text{FDD}) + (\text{FDD} - \text{NOIQ})), (\text{NOINQ} + (\text{PT+NOINQ})), (\text{PT+NOINQ})) \times (\max(((\text{NOIQ+MRT})/\text{W}), (((\text{FDD} \times \text{NOIQ})/(\text{NOR}/\text{W})) \times (\max(\text{PT}, \text{IPT})/\text{NMRT}))) - \min(((\text{MRT}/\text{W})/\text{W}), (\min(\text{NOINQ}, \text{FDD}) + (\text{PT+NOINQ})))), \quad (4)$$

$$g_2 = (((\min((\text{WINQ}/\text{W}), (\text{NOPT} \times \text{W})) + (\text{PT} + (\text{PT}/\text{W})))/\text{W}) \times (\text{MRT}/(\text{W}/\text{PT}))) \times \max((((\text{PT}/\text{W}) + (\text{PT} + (\text{NOPT} \times \text{W})))) \times (\text{ifte}(\text{NOINQ}, \text{W}, \text{PT}) \times (((\text{WINQ}/\text{W})/\text{ifte}(\text{NOINQ}, \text{WINQ}, \text{MRT}))/\text{ifte}(\text{NOINQ}, \text{WINQ}, \text{MRT})))), ((\text{PT} + (\text{PT}/\text{W})) + (\text{WINQ}/\text{W}))). \quad (5)$$

In summary, the analysis on the evolved rules obtained by ET-GP and FS-GP shows that by selecting the key feature set, FS-GP focuses more on the promising regions in the search space, and is easier to find meaningful building blocks than ET-GP.

5. CONCLUSION AND FUTURE WORK

In this paper, we designed a domain-knowledge-free feature selection method for evolving dispatching rules using GP. The proposed method is based on the contribution or relevance of each feature to the priority function of the evolved rules. Based on the proposed feature selection method,

we further designed a feature-selection-based GP (FS-GP), which learns the importance of the features offline. The experimental results show that selecting only the identified key terminals can lead to significantly better performance on both training and test sets. The analysis on the evolved rules reveals that FS-GP is more capable of finding more useful building blocks that contributes more to the performance of the rules.

In the future, to improve training efficiency, we will investigate online learning (estimating the importance of the features during the GP process), and adaptively select the terminals during the GP process. In addition, identifying useful building blocks (sub-trees) into the terminal set will further help the search to find more promising regions.

6. REFERENCES

[1] Job shop scheduling. http://optimizizer.com/TA.php.

[2] 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*, 64(12):1695–1724, 2013.

[3] 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.

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

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

[6] 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.

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

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

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

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

[11] B. Jürgen, S. Nguyen, C. W. Pickardt, and M. Zhang. Automated design of production scheduling heuristics: A review. *IEEE Transactions on Evolutionary Computation*, 2015.

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

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

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

[15] 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*, 17(5):621–639, 2013.

[16] S. Nguyen, M. Zhang, M. Johnston, and K. C. 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.

[17] 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.

[18] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[19] C. W. 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.

[20] M. L. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer Science & Business Media, 2012.

[21] 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.

[22] R. Ramasesh. Dynamic job shop scheduling: a survey of simulation research. *Omega*, 18(1):43–57, 1990.

[23] 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.

[24] 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.

[25] E. D. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[26] J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54(3):453–473, 2008.

[27] B. Xue, M. Zhang, W. Browne, and X. Yao. A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation*, 2015.

[28] 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.