A Comprehensive Analysis on Reusability of GP-Evolved Job Shop Dispatching Rules

Yi Mei*, Mengjie Zhang*
* School of Engineering and Computer Science, Victoria University of Wellington
Wellington, 6012, New Zealand
{yi.mei, mengjie.zhang}@ecs.vuw.ac.nz

***Abstract*—Genetic Programming (GP) has been extensively used to automatically design dispatching rules for job shop scheduling problems. However, the previous studies only focus on the performance on the training instances. So far, there is no systematic investigation of the reusability of the GP-evolved rules on unseen instances. In practice, it is desirable to train the rules on smaller job shop instances, and apply them to larger instances with more jobs and machines to save training time. In this case, the reusability of the GP-evolved rules under different numbers of jobs and machines is an important issue. In this paper, a comprehensive investigation is conducted to analyse how the variation in the numbers of jobs and machines from the training set to the test set affects the reusability of the GP-evolved rules. It is found that in terms of minimizing makespan, the reusability of the GP-evolved rules highly depends on variation in the numbers of jobs and machines. A better reusability can be achieved by choosing training instances whose numbers of jobs and machines (or at least the ratio between the numbers of jobs and machines) are closer to that of the test instances. Furthermore, the ratio between the numbers of jobs and machines is demonstrated to be an important factor to reflect the complexity of an instance for dispatching rules. This study is the first systematic investigation on the reusability of GP-evolved dispatching rules.**

I. Introduction

Job Shop Scheduling (JSS) [1] is a classic scheduling problem with various applications in manufacturing industries and cloud computing. Due to its significance in practice, JSS has been extensively investigated in the past decades, and a large number of mathematical programming and heuristic algorithms have been proposed for solving it ([2], [3]).

Dispatching rules have been widely used in real-world JSS problems, due to their simplicity and ease of application. More importantly, dispatching rules have a much better scalability than the meta-heuristic methods, and are flexible to be used in any unseen instance without any modification. Briefly speaking, a dispatching rule can generate a schedule by choosing the next operation to be processed by an idle machine according to some criterion. There have been numerous dispatching rules designed manually for various job shop situations and objectives, which have achieved reasonably good performance [4], [5], [6]. In addition, there are some studies on composite rules [7] and adaptive rule selection [8] from a pool of rules to improve the robustness of the rules in different job shop situations.

Recently, automatic design of dispatching rules using Genetic Programming (GP) [9] is becoming prevalent, evidenced by the increasing number of studies in the recent decades ([10], [11], [12], [13]). GP formulates the rules as priority functions, which are represented as trees. By searching in the rule space, more hidden knowledge can be discovered by computer programs, and more complex and better rules can be obtained for JSS. It has been demonstrated that the rules generated by GP can perform much better than the existing manually designed rules on both static [12] and dynamic [13] JSS instances.

Traditionally, when evolving dispatching rules with GP, a set of JSS instances are selected (from existing benchmarks or randomly generated) for evaluating the rules. That is, during the optimization process, the rules that generate better schedules for the selected JSS instances are considered to have better fitness values. In this sense, the optimization process can be seen as a rule training process, in which the selected JSS instances are the training instances. Most of the existing work focused on improving the performance of the rules on the training instances (i.e. the *training performance*). However, the reusability of the rules on unseen instances (i.e. the *test performance*), especially those with different properties, has not been investigated systematically. Some work was done recently [14], [12], [13], [15] to evaluate the trained rules on unseen instances with different number of machines and distributions, showing the potential of the GP-evolved dispatching rules in reusability and scalability. However, there has been no guideline on how to form the training set to improve the reusability of the rules.

This paper aims to conduct a systematic study to analyse the reusability of the GP-evolved rules on unseen instances. In particular, motivated by training the rules on smaller instances and applying them to larger unseen instances to improve the training efficiency, we focus on the situation in which the test instances have larger numbers of jobs and machines than the training instances. The goal of this study is to discover how the variation in the numbers of jobs and machines from the training set to the test set affects the reusability of the GP-evolved dispatching rules, and identify the important factors that affect the relatedness of the training set to the test set. To this end, a number of experimental comparisons will be sophisticatedly designed to compare among the rules trained from different training sets on the same test set. Based on the knowledge learnt, we expect to propose a new guideline of selecting a more related training set so that the resultant GP-evolved rules are more reusable on the test set.

The rest of the paper is organized as follows. First, the background about JSS, automatic rule design with GP and relationship between shop scenario and rule performance are introduced in Section II. Then, the methodology design is described in III, including the training and test phases, and the reusability comparisons. After that, the experimental results and discussions are given in Section IV. Finally, Section V presents the conclusion and future work.

II. Background

A. Job Shop Scheduling and Dispatching Rule

In JSS, there are N *jobs* to be processed on M *machines*. Each job i has M *operations*, denoted as O_{ij}, $j = 1, \ldots, M$, each processed on a distinct machine. For each job, the operations must be processed in order, i.e. $O_{i,j+1}$ cannot be processed until the operation O_{ij} has been completed. Each operation O_{ij} has a processing time T_{ij} and a processing machine M_{ij}. The objective is to design a *schedule* to finish all the jobs with the machines, so that the predefined objectives (e.g. makespan and total flowtime) are optimized and the following constraints are satisfied:

1) Each operation O_{ij} must be processed on its corresponding machine M_{ij};
2) For each job i, the starting time of $O_{i,j+1}$ must be no earlier than the finishing time of O_{ij}, $\forall\, j = 1, \ldots, M-1$;
3) Each machine j cannot process more than one operations at the same time;
4) Once a process has started, it cannot be interrupted.

Note that there are many other JSS variants including missing operations, dynamic job arrival, parallel machines, etc. To facilitate analysis, the most basic model is considered in this paper.

A *dispatching rule* X can be seen as a heuristic that can generate a schedule for any JSS instance. A dispatching rule is essentially a *priority function*, which is used to select the next operation from the queue to be processed whenever a machine becomes idle. For example, in the commonly used shortest-processing-time rule, the priority function is defined as the processing time of the operation, and the operation with the minimal priority value is selected to be the next operation.

B. Genetic Programming for Evolving Dispatching Rules

Unlike the manually designed rules, GP attempts to find better rules by evolving priority functions. During the training (evolution) process, the fitness function is defined based on a set of JSS training instances $\mathcal{I}$. Specifically, given a dispatching rule X, a solution $S(X; I)$ is generated by applying X on each instance $I \in \mathcal{I}$. Then, the corresponding objective value (e.g. makespan and tardiness) of the solutions are calculated and normalized across the instances (e.g. divided by the lower bound). Finally, the fitness of the dispatching rule X is defined as follows:

$$f(X; \mathcal{I}) = \frac{1}{|\mathcal{I}|} \sum_{I \in \mathcal{I}} \hat{f}\left(S(X; I)\right), \tag{1}$$

where $\hat{f}\left(S(X; I)\right)$ is the normalized objective value of $S(X; I)$, and $|\mathcal{I}|$ is the number of instances in $\mathcal{I}$. After the GP process has finished, the best rule X^* with the minimal $f(X; \mathcal{I})$ value is selected to be used on any unseen test instance.

Dimopoulos et al. [16] conducted a preliminary study on evolving dispatching rules with GP on a single-machine JSS problem using several sets of training instances with different number of jobs. They achieved much better results in terms of total tardiness. Geiger et al. also managed to obtain more efficient rules than the manually designed ones for the single-machine JSS problems [17]. For the more practical scenarios with multiple machines, much more work has been done (e.g. [10], [12], [13], [18]). Tay and Ho used GP to evolve dispatching rules to solve multi-objective flexible JSS problem [19]. Various training sets have been generated for training the rules, which have different numbers of jobs and machines (e.g. [10], [19], [12], [13]). Many of the GP-evolved dispatching rules have achieved competitive or even better results than the meta-heuristic methods, particularly on large and/or dynamic JSS problems [15].

C. Dependency of Rule Performance on Shop Scenario

There have been some studies about how different shop scenarios affect the performance of manually designed rules in the dynamic environment. According to the surveys conducted by Kiran and Smith (1984) [20] and Ramasesh (1990) [21], the relationship between shop scenario and relative performance of rules can be summarized as follows:

- The arrival rate distribution does not significantly affect the relative performance of scheduling rules [22];
- The distribution of the processing time (exponential, Erlang and uniform) significantly affected the relative performance of the rules;
- The number of machines seems not be an important factor that influences the relative performance of dispatching rules [23] [24];
- There can be various settings on the number of operations that each job has, and the sequence of machines of the operations, e.g. normal/uniform distributions. The impact of these configurations on the performance of rules is still unknown;
- In terms of the percentage of on time completion (percentage of tardy jobs), the relative performance of rules depends on the utilization level [25];
- In terms of the mean tardiness, the relative performance can be dependent on the due-date tightness and utilization level.

However, the existing relevant studies are all for the manually designed rules, and can by no means be comprehensive. There has been no study about when evolving dispatching rules with GP, how the selection of training instances will affect the reusability of the evolved rules. In this paper, we make the initial attempt by comparing the GP-evolved rules obtained from different training instances, which have different properties. As a starting point, we focus on the static JSS.

III. Methodology Design

Similar to the assumption in traditional machine learning and classification, the training and test instances must be in the same feature space and have the same distribution [26]. In JSS, the concept of feature space and its distribution cannot be trivially defined. In general, a JSS problem includes at least the following features that may change from instance to instance:

1) Number of jobs N;
2) Number of machines M;
3) Processing time T_{ij} of each operation;
4) Machine M_{ij} to process each operation.

Intuitively, following the same definition as that in traditional classification, two JSS instances are considered to have the same feature space with the same distribution, or belong to the same *domain*, if they have the same N and M, and the same joint distribution of all the T_{ij}'s and M_{ij}'s. However, this assumption is too restricted, since it only allows the dispatching rules to be reused on the instances with the same number of jobs and machines, which is usually not the case in practice. Here, an instance with different N and M is said to belong to a different but *related* domain. Then, it is desirable to reuse the rules to instances not only from the same domain, but also those from related domains.

In this study, we aim to investigate whether it is proper to relax the assumption on N and M. In other words, we attempt to answer the following research questions:

- Can the GP-evolved dispatching rules be reused on unseen instances with different numbers of jobs and machines?
- What is the relationship between the reusability of the GP-evolved dispatching rules and the change in the numbers of jobs and machines from the training set to the test set?

To answer the above questions, a number of experiments are designed to compare the reusability of the rules trained from different training sets on the same test set.

A. Datasets

The datasets used in the experiments must include instances with various numbers of jobs and machines. To this end, we selected the Taillard JSS benchmark set [27] (TA set), which contains eight groups. Each group includes ten instances with the same numbers of jobs and machines. The processing time and machine matrices are generated from the same distribution using different random seeds. For each operation, the processing time is randomly sampled from the uniform distribution between 1 and 99, and the processing machine is randomly picked from the machine set. However, the TA set only consists of two different numbers of machines (15 and 20), which are close to each other. To obtain more instances with a more widespread number of machines, we randomly generated another set of instances using the same method to generate the TA set. The newly generated set, namely the NEW set, consists of 20 groups. Each group has 20 instances with the same number of jobs and machines, but different processing time and machine matrices. In the NEW set, the number of jobs ranges from 20 to 100, and the number of machines ranges from 5 to 20. The features of the TA and NEW sets are given in Table I. In the table, "*ratio*" simply stands for the ratio between the numbers of jobs and machines, i.e. $ratio = \frac{\#J}{\#M}$, where #J and #M denote the number of jobs and machines, respectively.

TABLE I: The features of the TA and NEW groups.

TA set							
Group	**#J**	**#M**	***ratio***	**Group**	**#J**	**#M**	***ratio***
T01	15	15	1.00	T03	20	20	1.00
T02	20	15	1.33	T05	30	20	1.50
T04	30	15	2.00	T07	50	20	2.50
T06	50	15	3.33	T08	100	20	5.00
NEW set							
Group	**#J**	**#M**	***ratio***	**Group**	**#J**	**#M**	***ratio***
M01	20	5	4.00	M11	20	15	1.33
M02	40	5	8.00	M12	40	15	2.67
M03	60	5	12.00	M13	60	15	4.00
M04	80	5	16.00	M14	80	15	5.33
M05	100	5	20.00	M15	100	15	6.67
M06	20	10	2.00	M16	20	20	1.00
M07	40	10	4.00	M17	40	20	2.00
M08	60	10	6.00	M18	60	20	3.00
M09	80	10	8.00	M19	80	20	4.00
M10	100	10	10.00	M20	100	20	5.00

B. Definition of Rule Fitness

In the experiments, a single objective is selected to eliminate potential effects of different objectives on the reusability. The makespan is considered, since it is one of the most commonly used objectives (e.g [4], [6]. Then, given a set $\mathcal{I}$ of JSS instances, the fitness of a rule X is defined as follows:

$$f(X;\mathcal{I}) = \frac{1}{|\mathcal{I}|}\sum_{I\in\mathcal{I}}\frac{\kappa(X;I)}{\kappa^*(I)} \times 100\%, \tag{2}$$

where $\kappa(X;I)$ is the makespan of the schedule obtained by applying rule X to the instance I, and $\kappa^*(I)$ is the best-known makespan value for the instance I. For the TA set, the best-known makespan values were directly obtained from the JSS website [28]. For the NEW set, since there is no best-known solution for it, the FDD rule (demonstrated to be effective in minimizing makespan [5]) was applied to the instances, and the resultant makespan values were taken as the best-known makespan values for the NEW instances.

C. Reusing Rules from a Training Set to a Test Set

Given a training set $\mathcal{I}_{\text{train}}$ and a test set $\mathcal{I}_{\text{test}}$, the experiment consists of a training phase and a test phase. During the training phase, the rules are trained by the standard GP using the fitness function $f(X;\mathcal{I}_{\text{train}})$ defined by Eq. (2). Here, the fitness value $f(X;\mathcal{I}_{\text{train}})$ is also called the *training performance* of X on the training set $\mathcal{I}_{\text{train}}$. The aim of the GP is to find a rule with the minimal $f(X;\mathcal{I}_{\text{train}})$ value.

In the GP system, the commonly used features were included in the terminal set, which are listed in Table II. The

$+$, $-$, $\times$ and protective $/$ (returning 1 if the denominator is 0) were used along with $\min$, $\max$ and $(\texttt{if} > 0)$ as the function set. The population size was set to 1024. The crossover and mutation rates were set to 90% and 10%, respectively. The maximal depth was set to 8, and the number of generations was set to 51. Since GP is a stochastic process, 50 independent runs were carried out. For each run r, the rule X_r^* with the best training performance was obtained.

TABLE II: The features (terminals) used in GP.

Feature	Description
PT	Processing time of the operation
RT	Ready time of the operation
RPT	Remaining processing time of the operation (inclusive)
MRT	Ready time of the machine
NRO	Number of remaining operations of the job (exclusive)
MNRO	Number of ready operations on the machine
CT	Current time
NOPT	Processing time of next operation
NOINQ	Number of operations in the queue of the next machine
IPT	Inverse processing time (1/PT)
FDD	Flow due date of the operation

During the test process, we apply each rule X_r^* obtained from $\mathcal{I}_{\text{train}}$ to $\mathcal{I}_{\text{test}}$. The *test performance* of X_r^* on $\mathcal{I}_{\text{test}}$ is defined as the fitness value $f(X_r^*; \mathcal{I}_{\text{test}})$, calculated by Eq. (2). Then, one can calculate the mean and standard deviation of the $f(X_r^*; \mathcal{I}_{\text{test}})$'s, $r = 1, \ldots, 50$.

D. Reusability Comparison

To investigate the influence of training set selection on reusability, we designed a number of *Reusability Comparisons* (RCs). In each RC, multiple training sets are compared with each other on the same test set in terms of the test performance of the rules obtained from the 50 independent runs.

Recalling that the goal is to train the rules on smaller instances and reuse them on larger instances. Therefore, in each RC, the training instances always have no more jobs and machines than the test instances, unless there are not enough training sets. In addition, to prevent mixing instances with different numbers of jobs and machines together, the training and test instances will be selected from the same group.

E. Results Illustration

To observe how the variation of the numbers of jobs and machines affects the reusability, the results will be illustrated as tables (Tables III–VII) and figures (Figs. 1–5). In the figures, the x-axis stands for the gap between the *ratio* values of the training set and the test set, and the y-axis presents the test performance (shown in percentage). Here, the *ratio* value is selected rather than the number of jobs or machines in order to normalize the features of the test sets which may have different numbers of jobs and machines. Each RC is represented as a curve consisting of several points with error bars. Each point corresponds to the test performance (mean $\pm$ standard deviation) of the rules obtained from one training set in the RC. For example, if the *ratio* values of the training and test sets are 1 and 2, respectively, and the mean test performance value is 98%, then the corresponding point will be (-1, 98).

IV. Results and Analysis

A. Comparisons with the Same Number of Machines

The first part of experiments investigates the effect of the number of jobs under the same number of machines. For the TA set, the T04, T06, T07 and T08 groups were selected to be the test sets. For each test set, there are at least two training sets with the same number of machines and fewer jobs. Similarly, for the NEW set, the M05, M10, M15 and M20 groups were selected to be the test sets. For each test set, there are four training sets to be compared with each other.

Table III shows the results of the RCs for the TA groups in which the training and test sets have the same number of machines. The "Fitness %" column stands for the fitness value of the rules on the test set, i.e. the test performance. From the table, it is clear that the fitness value decreases as the number of jobs in the training set increases. For all the four RCs for the TA groups, the training set with the largest number of jobs achieved the best test performance (smallest fitness values). Although the significance was only found on RC1, one can still see a consistent decreasing trend of the mean fitness value with the increase of the number of jobs.

TABLE III: The results of the RCs for the TA groups in which the training and test sets have the same number of machines.

RC	Train	Test	#J (train → test)	#M	*ratio* (train → test)	Fitness % (mean ± sd)
RC1	T01	T04	15 → 30	15	1.00 → 2.00	117.16 ± 0.95
	T02	T04	20 → 30	15	1.33 → 2.00	**116.48 ± 0.98**
RC2	T01	T06	15 → 50	15	1.00 → 3.33	112.36 ± 0.89
	T02	T06	20 → 50	15	1.33 → 3.33	111.89 ± 0.76
	T04	T06	30 → 50	15	2.00 → 3.33	111.65† ± 0.60
RC3	T03	T07	20 → 50	20	1.00 → 2.50	112.78 ± 1.50
	T05	T07	30 → 50	20	1.50 → 2.50	112.36† ± 0.74
RC4	T03	T08	20 → 100	20	1.00 → 5.00	105.35 ± 1.98
	T05	T08	30 → 100	20	1.50 → 5.00	104.42 ± 2.16
	T07	T08	50 → 100	20	2.50 → 5.00	104.22† ± 0.90

· For each RC, the t-test with significance level of 0.05 was conducted on each pair of the compared training sets. If the best training set is significantly better than the others, the corresponding entry is marked in bold. Otherwise, it is marked with †.

Fig. 1 illustrates the fitness value versus the gap in the *ratio* value of the four RCs shown in Table III, in which each RC is represented as a curve. It is clear that for all the RCs, the fitness value tends to decrease (improve) when the gap in the *ratio* value is getting closer to zero. Note that when comparing among different RCs, it seems that the fitness values in RC4 were better than those in RC1, although the gap in *ratio* value is larger in RC4. This is due to the different test sets (T08 in RC4 versus T04 in RC1), which may have different tightness of the lower and upper bounds. For example, the T08 instances might have a looser bounds (larger $\kappa^*(I)$) than T04, which lead to smaller fitness values. Therefore, we *only* compare between the points in the same RC rather than those across different RCs.

Table IV shows the results of the RCs for the NEW groups. The corresponding figure is in Fig. 2. It can be seen that for RC1 and RC2, the fitness values are almost the same

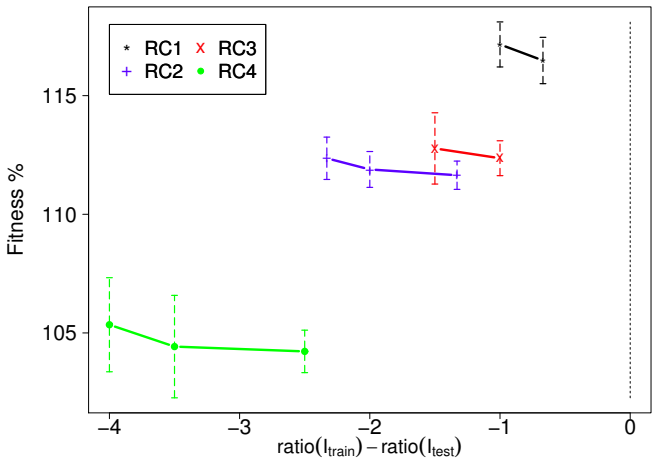

Fig. 1: The fitness value (standard deviation as the error bar) on the test set versus the gap in the *ratio* value (training minus test) of the RCs for the TA groups in which the training and test sets have the same number of machines. The RCs are defined in Table III.

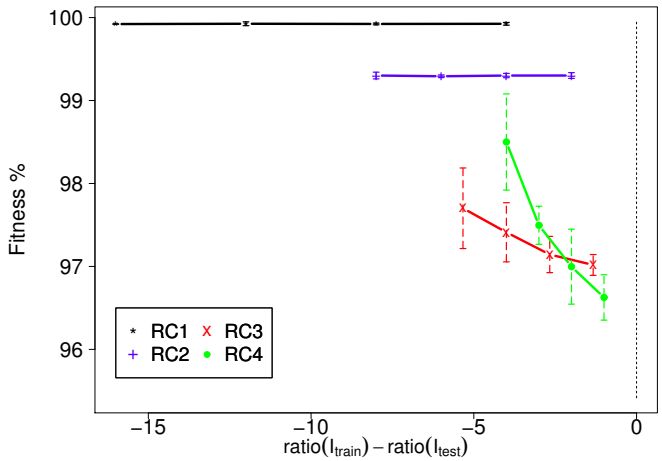

Fig. 2: The fitness value (standard deviation as the error bar) on the test set versus the gap in the *ratio* value (training minus test) of the RCs for the NEW groups in which the training and test sets have the same number of machines. The RCs are defined in Table IV.

regardless of the gap in the *ratio* value. However, in RC3 and RC4, the fitness value dramatically decreases as the *ratio* value of the training set is getting closer to that of the test set. In both RC3 and RC4, the best training set (M14 and M19, respectively) achieved *significantly* better fitness values on the test set than all the other compared training sets. Therefore, the difference is more significant on the test sets with fewer machines.

TABLE IV: The results of the RCs for the NEW groups in which the training and test sets have the same number of machines.

RC	Train	Test	#J (train → test)	#M	*ratio* (train → test)	Fitness % (mean ± sd)
RC1	M01	M05	20 → 100	5	4.00 → 20.00	99.92† ± 0.00
	M02	M05	40 → 100	5	8.00 → 20.00	99.93 ± 0.02
	M03	M05	60 → 100	5	12.00 → 20.00	99.92 ± 0.01
	M04	M05	80 → 100	5	16.00 → 20.00	99.92 ± 0.02
RC2	M06	M10	20 → 100	10	2.00 → 10.00	99.30 ± 0.04
	M07	M10	40 → 100	10	4.00 → 10.00	99.29† ± 0.02
	M08	M10	60 → 100	10	6.00 → 10.00	99.30 ± 0.03
	M09	M10	80 → 100	10	8.00 → 10.00	99.30 ± 0.03
RC3	M11	M15	20 → 100	15	1.33 → 6.67	97.70 ± 0.49
	M12	M15	40 → 100	15	2.67 → 6.67	97.41 ± 0.36
	M13	M15	60 → 100	15	4.00 → 6.67	97.14 ± 0.22
	M14	M15	80 → 100	15	5.33 → 6.67	**97.02 ± 0.13**
RC4	M16	M20	20 → 100	20	1.00 → 5.00	98.50 ± 0.58
	M17	M20	40 → 100	20	2.00 → 5.00	97.50 ± 0.23
	M18	M20	60 → 100	20	3.00 → 5.00	97.00 ± 0.45
	M19	M20	80 → 100	20	4.00 → 5.00	**96.63 ± 0.27**

B. Comparisons with the Same Number of Jobs

The second part of the experiments was only carried out on the NEW set, since the TA set only has two different numbers of machines. Five RCs were carried out, in which the test sets were the five groups with 20 machines (M16, M17, M18, M19 and M20). For each test set, the training sets with the same number of jobs but fewer machines (5, 10 or 15 machines) were compared with each other. Table V and Fig. 3 show the corresponding results on the five RCs. It is obvious that the reusability is much improved as the *ratio* of the training set is getting closer to that of the test set. For all the RCs, the best training set significantly outperformed the others under the t-test with significance level of 0.05. It is also clear that the standard deviation is much larger when the *ratio* value of the training set is more distant from that of the test set, implying that the dispatching rules were much less stable on the test set. This indicates a worse reusability in terms of stability.

TABLE V: The results of the RCs for the NEW groups in which the training and test sets have the same number of jobs.

RC	Train	Test	#J	#M (train → test)	*ratio* (train → test)	Fitness % (mean ± sd)
RC1	M01	M16	20	5 → 20	4.00 → 1.00	95.25 ± 1.33
	M06	M16	20	10 → 20	2.00 → 1.00	94.43 ± 0.85
	M11	M16	20	15 → 20	1.33 → 1.00	**93.57 ± 0.59**
RC2	M02	M17	40	5 → 20	8.00 → 2.00	98.83 ± 4.56
	M07	M17	40	10 → 20	4.00 → 2.00	93.66 ± 0.99
	M12	M17	40	15 → 20	2.67 → 2.00	**93.34 ± 0.34**
RC3	M03	M18	60	5 → 20	12.00 → 3.00	104.58 ± 5.36
	M08	M18	60	10 → 20	6.00 → 3.00	96.12 ± 1.28
	M13	M18	60	15 → 20	4.00 → 3.00	**95.07 ± 0.58**
RC4	M04	M19	80	5 → 20	16.00 → 4.00	105.36 ± 5.58
	M09	M19	80	10 → 20	8.00 → 4.00	97.40 ± 1.56
	M14	M19	80	15 → 20	5.33 → 4.00	**96.00 ± 0.62**
RC5	M05	M20	100	5 → 20	20.00 → 5.00	105.76 ± 6.05
	M10	M20	100	10 → 20	10.00 → 5.00	98.01 ± 0.93
	M15	M20	100	15 → 20	6.67 → 5.00	**97.20 ± 0.66**

C. Comparisons with Different Numbers of Jobs and Machines

Based on the previous experimental analysis, one can clearly see that the gap in the *ratio* value plays an important role in affecting the test performance of the dispatching rules. Therefore, in this section, we relaxed the restriction on both the number of jobs and machines, and directly investigated the relationship between the gap in the *ratio* value and the reusability of the dispatching rules.

For the TA set, four RCs were conducted. The corresponding test sets were the TA groups with 20 machines (T03, T05,

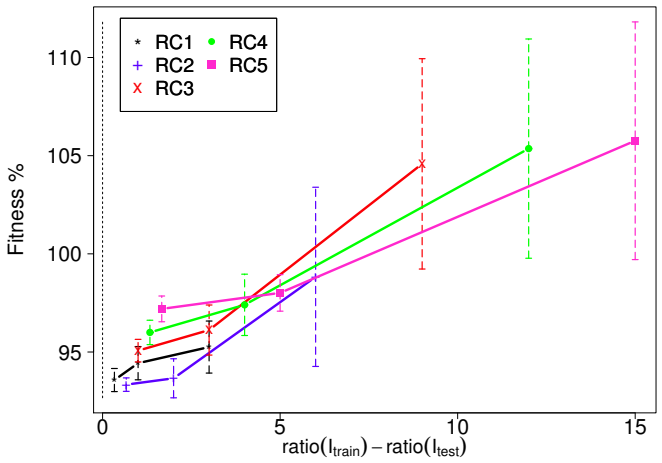

Fig. 3: The fitness value (standard deviation as the error bar) on the test set versus the gap in the *ratio* value (training minus test) of the RCs for the NEW groups in which the training and test sets have the same number of jobs. The RCs are defined in Table V.

T07 and T08). For each test set, the four groups with 15 machines (T01, T02, T04 and T06) were compared with each other. Note that in some cases, the training set can have more jobs than the test set (e.g. trained on T06 and tested on T03). However, the results were still included in the RCs to keep enough number of training sets. For the NEW set, the four largest groups (M14, M15, M19 and M20) were selected as test sets. For each test set, two or three RCs were conducted, depending on its number of machines.

Table VI and Fig. 4 show the results of the RCs for the TA sets. One can see an obvious trend that both the mean and standard deviation of the fitness decreases as the two *ratio*s are getting closer to each other. Significance was found for 3 out of the 4 RCs. Table VII gives the results of the RCs for the NEW sets, and Fig. 5 shows the representative results on M19 and M20 test sets. To make the figure tidy, the error bars of the results in the crowded area around zero were omitted without losing much information. Fig. 5 shows the same pattern as Fig. 4. Significance was found in 8 out of the total 10 RCs (except RC7 and RC10). In addition, the reusability was less sensitive when the gap lied within a certain region around zero (e.g. $[-2, 2]$).

Fig. 6 shows the scatter plot of all the results obtained for the NEW test sets, in which each point represents the mean fitness value of the rules obtained from one training set on one test set. The figure demonstrates a very strong correlation between the gap in *ratio* values and the mean fitness value of the dispatching rules, resulting in a correlation coefficient of 0.92.

D. Further Analysis

In order to obtain a deeper understanding about how different training sets affect the reusability of the GP-evolved rules, we analysed the total occurrences of the terminals in the priority functions of the best rules obtained in the 50 independent runs. Based on such occurrences, we made

TABLE VI: The results of the RCs for the TA groups in which the training and test sets have different numbers of jobs and machines.

RC	Train	Test	#J (train → test)	#M (train → test)	*ratio* (train → test)	Fitness % (mean ± sd)
RC1	T01	T03	15 → 20	15 → 20	1.00 → 1.00	**116.08 ± 1.15**
	T02	T03	20 → 20	15 → 20	1.33 → 1.00	116.91 ± 1.18
	T04	T03	30 → 20	15 → 20	2.00 → 1.00	117.25 ± 0.79
	T06	T03	50 → 20	15 → 20	3.33 → 1.00	119.82 ± 3.05
RC2	T01	T05	15 → 30	15 → 20	1.00 → 1.50	117.96 ± 1.01
	T02	T05	20 → 30	15 → 20	1.33 → 1.50	117.50 ± 0.80
	T04	T05	30 → 30	15 → 20	2.00 → 1.50	**117.08 ± 0.77**
	T06	T05	50 → 30	15 → 20	3.33 → 1.50	120.32 ± 2.71
RC3	T01	T07	15 → 50	15 → 20	1.00 → 2.50	112.85 ± 0.87
	T02	T07	20 → 50	15 → 20	1.33 → 2.50	112.42 ± 0.98
	T04	T07	30 → 50	15 → 20	2.00 → 2.50	112.30† ± 0.67
	T06	T07	50 → 50	15 → 20	3.33 → 2.50	112.40 ± 1.17
RC4	T01	T08	15 → 100	15 → 20	1.00 → 5.00	105.35 ± 1.09
	T02	T08	20 → 100	15 → 20	1.33 → 5.00	104.66 ± 1.47
	T04	T08	30 → 100	15 → 20	2.00 → 5.00	104.14 ± 0.89
	T06	T08	50 → 100	15 → 20	3.33 → 5.00	**103.22 ± 0.82**

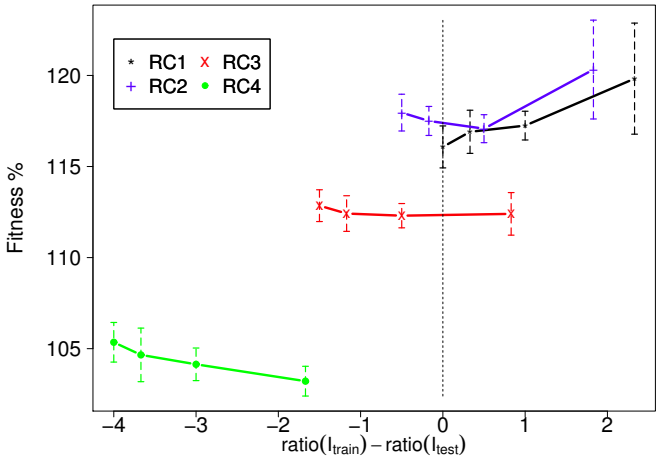

Fig. 4: The test performance (standard deviation as the error bar) versus the gap in the *ratio* value (training minus test) of the RCs for the TA groups in which the training and test sets have different numbers of jobs and machines. The RCs are defined in Table VI.

a comparison between M01, M05 and M20. M01 has the smallest problem size (20 jobs and 5 machines). M05 is larger than M01 (100 jobs and 5 machines). M20 is the largest group (100 jobs and 20 machines). On the other hand, the *ratio* value of M05 (20.00) is much different from that of both M01 (4.00) and M20 (5.00).

The results are shown in Fig. 7. The figure shows that in the rules obtained from both M01 and M20, there are much more terminal occurrences than those trained from M05. Note that the terminal occurrences can reflect the size of the priority function, and thus the complexity of the rule to some extent. Fig. 7 implies that the rules obtained from M05 are much simpler than those obtained from M01 and M20. In contrast to M05, the rules obtained from M01 and M20 have similar terminal occurrences, indicating similar complexities. The rules for M20 are slightly more complex than those for M01, especially in the occurrences of the terminals `PT`, `NOINQ`, `MNRO` and `CT`.

From the observations in Fig. 7, we can see that the

TABLE VII: The results of the RCs for the NEW groups in which the training and test sets have different numbers of jobs and machines.

RC	Train	Test	#J (train → test)	#M (train → test)	*ratio* (train → test)	Fitness % (mean ± sd)
RC1	M01	M14	20 → 80	5 → 15	4.00 → 5.33	**97.22 ± 0.37**
	M02	M14	40 → 80	5 → 15	8.00 → 5.33	99.30 ± 2.19
	M03	M14	60 → 80	5 → 15	12.00 → 5.33	102.63 ± 3.96
RC2	M06	M14	20 → 80	10 → 15	2.00 → 5.33	97.04 ± 0.20
	M07	M14	40 → 80	10 → 15	4.00 → 5.33	**96.84 ± 0.28**
	M08	M14	60 → 80	10 → 15	6.00 → 5.33	97.14 ± 0.29
RC3	M01	M15	20 → 100	5 → 15	4.00 → 6.67	**97.50 ± 0.28**
	M02	M15	40 → 100	5 → 15	8.00 → 6.67	99.01 ± 1.67
	M03	M15	60 → 100	5 → 15	12.00 → 6.67	101.45 ± 3.11
	M04	M15	80 → 100	5 → 15	16.00 → 6.67	103.20 ± 4.72
RC4	M06	M15	20 → 100	10 → 15	2.00 → 6.67	97.38 ± 0.21
	M07	M15	40 → 100	10 → 15	4.00 → 6.67	**97.24 ± 0.27**
	M08	M15	60 → 100	10 → 15	6.00 → 6.67	97.38 ± 0.17
	M09	M15	80 → 100	10 → 15	8.00 → 6.67	97.60 ± 0.53
RC5	M01	M19	20 → 80	5 → 20	4.00 → 4.00	**96.95 ± 0.65**
	M02	M19	40 → 80	5 → 20	8.00 → 4.00	99.61 ± 2.59
	M03	M19	60 → 80	5 → 20	12.00 → 4.00	103.69 ± 4.52
RC6	M06	M19	20 → 80	10 → 20	2.00 → 4.00	96.70 ± 0.39
	M07	M19	40 → 80	10 → 20	4.00 → 4.00	**96.24 ± 0.65**
	M08	M19	60 → 80	10 → 20	6.00 → 4.00	96.98 ± 0.66
RC7	M11	M19	20 → 80	15 → 20	1.33 → 4.00	96.96 ± 0.49
	M12	M19	40 → 80	15 → 20	2.67 → 4.00	96.91 ± 0.62
	M13	M19	60 → 80	15 → 20	4.00 → 4.00	95.99† ± 0.51
RC8	M01	M20	20 → 100	5 → 20	4.00 → 5.00	**97.69 ± 0.53**
	M02	M20	40 → 100	5 → 20	8.00 → 5.00	100.03 ± 2.36
	M03	M20	60 → 100	5 → 20	12.00 → 5.00	103.54 ± 4.08
	M04	M20	80 → 100	5 → 20	16.00 → 5.00	105.27 ± 5.25
RC9	M06	M20	20 → 100	10 → 20	2.00 → 5.00	97.45 ± 0.41
	M07	M20	40 → 100	10 → 20	4.00 → 5.00	**97.10 ± 0.48**
	M08	M20	60 → 100	10 → 20	6.00 → 5.00	97.72 ± 0.54
	M09	M20	80 → 100	10 → 20	8.00 → 5.00	97.98 ± 0.95
RC10	M11	M20	20 → 100	15 → 20	1.33 → 5.00	97.99 ± 0.56
	M12	M20	40 → 100	15 → 20	2.67 → 5.00	97.49 ± 0.67
	M13	M20	60 → 100	15 → 20	4.00 → 5.00	96.86 ± 0.43
	M14	M20	80 → 100	15 → 20	5.33 → 5.00	96.73† ± 0.34

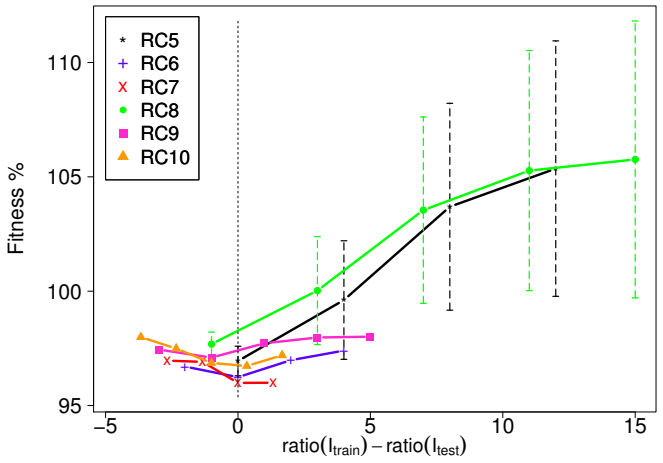

Fig. 5: The test performance (standard deviation as the error bar) versus the gap in the *ratio* value (training minus test) of the RCs for the NEW groups in which the training and test sets have different numbers of jobs and machines. The RCs are defined in Table VII.

M05 instances are simple, since they can be tackled well by relatively simple rules. However, the M01 and M20 instances

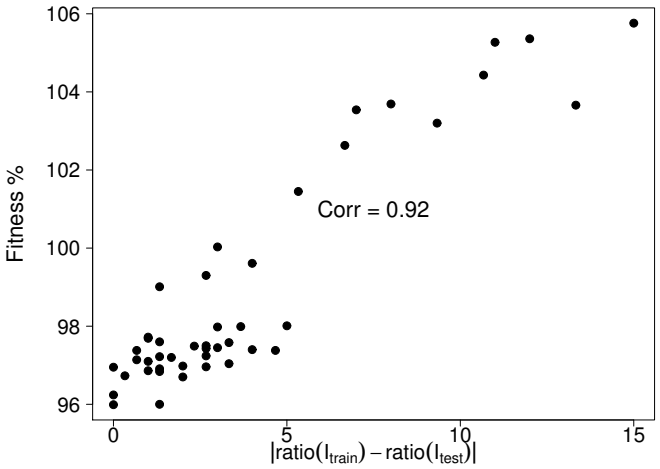

Fig. 6: The mean reusability versus the absolute gap in the *ratio* value for the NEW set.

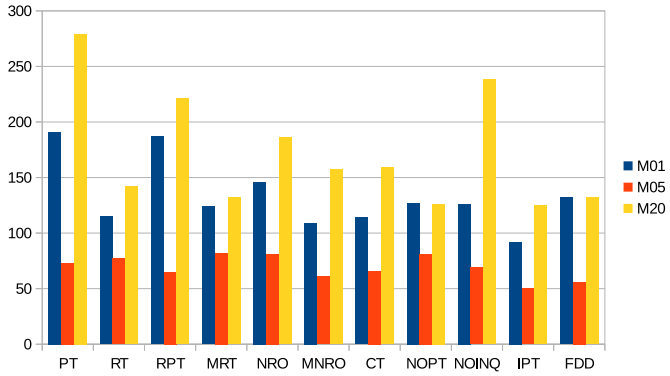

Fig. 7: The total occurrences of the terminals in the priority functions of the best rules obtained by the 50 independent runs of M01, M05 and M20.

are much more complex, requiring more sophisticated rules. This indicates that the complexity of the JSS instances for dispatching rules depends more on the *ratio* value rather than simply the problem size. An instance with a smaller *ratio* value tends to be much more complex. The results of RC8 in Table VII and RC5 in Table IV show that on the test set of M20, the reusability of the rules obtained from M01 were much better than those obtained from M05. It is not surprising that the simple rules obtained from M05 is likely to ignore much information and fail to distinguish many complex conditions in M20.

When taking a closer look at Fig. 7, one can see that when the instances become more complex, the occurrences of the terminals `PT`, `RPT`, `NOINQ`, `NRO`, `CT` and `MNRO` increased much more than the other terminals. This indicates that these terminals are more important in effectively distinguishing among complex conditions.

In summary, from the experimental analysis, we can obtain the following observations:

1) The complexity of a JSS instance for dispatching rules largely depends on the *ratio* value. If the *ratio* value is

smaller, then there are more complex conditions to be distinguished by more sophisticated rules;

2) Ceteris paribus, if the *ratio* value changes a lot from the training set to the test set, the reusability of the GP-evolved dispatching rules will dramatically deteriorate, mainly due to the change of the complexity of the instances for dispatching rules;
3) The reusability is more sensitive to the gap in the *ratio* value when the gap is larger. On the contrary, it is tolerable to use the training instances with similar *ratio* values without losing much reusability;
4) When the *ratio* value is the same, a better reusability can be achieved by choosing a training set whose numbers of jobs and machines are closer to that of the test set.

Based on the above observations, we suggest the following two criteria in selecting the training set to achieve better reusability:

- (*Strong criterion*): The training set has the same (or similar) numbers of jobs and machines as the test set;
- (*Weak criterion*): The training set has the same (or close) *ratio* value as the test set.

Considering that in practice, the size of the test instances will be unknown in advance. In this case, one can evolve and store a set of GP rules, each for a different *ratio* value. Then, during the test phase, the rule with the closest *ratio* value to that of the test instances will be chosen.

V. Conclusion and Future Work

In this paper, a comprehensive experimental analysis is conducted to investigate the reusability of the GP-evolved dispatching rules when applied to unseen test instances with different numbers of jobs and machines. In addition to adopting a commonly used job shop benchmark set (the TA set), a new dataset was generated to cover a wider range of number of jobs and machines. A number of reusability comparisons were designed to discover useful knowledge about how the training set selection affects the reusability of GP-evolved rules. The analysis showed that the reusability of the GP-evolved rules dramatically deteriorates if the test set has a considerably different *ratio* value from the training set. This suggests that it would be unwise to mix the instances with a wide range of number of jobs and machines in the training set, which may mislead the evolution of the dispatching rules.

In the future, the reusability of the rules for dynamic JSS will be examined. Moreover, new GP algorithms will be designed to take the reusability issue into account (e.g. including the number of jobs and machines and the *ratio* value in the terminal set) to produce dispatching rules that are more flexible and reusable when the number of jobs and machines of the test instances are unknown in advance.

References

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

[2] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[3] B. Peng, Z. Lü, and T. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Computers & Operations Research*, vol. 53, pp. 154–164, 2015.

[4] O. Holthaus and C. Rajendran, "Efficient dispatching rules for scheduling in a job shop," *International Journal of Production Economics*, vol. 48, no. 1, pp. 87–105, 1997.

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

[6] O. Holthaus and C. Rajendran, "Efficient jobshop dispatching rules: further developments," *Production Planning & Control*, vol. 11, no. 2, pp. 171–178, 2000.

[7] M. E. Aydin and E. Öztemel, "Dynamic job-shop scheduling using reinforcement learning agents," *Robotics and Autonomous Systems*, vol. 33, no. 2, pp. 169–178, 2000.

[8] V. Subramaniam, G. Lee, G. Hong, Y. Wong, and T. Ramesh, "Dynamic selection of dispatching rules for job shop scheduling," *Production planning & control*, vol. 11, no. 1, pp. 73–81, 2000.

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

[10] D. Jakobović and L. Budin, "Dynamic scheduling with genetic programming," in *Genetic Programming*. Springer, 2006, pp. 73–84.

[11] C. D. Geiger and R. Uzsoy, "Learning effective dispatching rules for batch processor scheduling," *International Journal of Production Research*, vol. 46, no. 6, pp. 1431–1454, 2008.

[12] 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," *Evolutionary Computation, IEEE Transactions on*, vol. 17, no. 5, pp. 621–639, 2013.

[13] ——, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *Evolutionary Computation, IEEE Transactions on*, vol. 18, no. 2, pp. 193–208, 2014.

[14] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. ACM, 2010, pp. 257–264.

[15] B. Jürgen, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated design of production scheduling heuristics: A review," *IEEE Transactions on Evolutionary Computation*, 2015. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2015.2429314

[16] C. Dimopoulos and A. M. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Advances in Engineering Software*, vol. 32, no. 6, pp. 489–498, 2001.

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

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

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

[20] A. S. Kiran and M. L. Smith, "Simulation studies in job shop sheduling—i a survey," *Computers & Industrial Engineering*, vol. 8, no. 2, pp. 87–93, 1984.

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

[22] D. A. Elvers, "The sensitivity of the relative effectiveness of job shop dispatching rules with respect to various arrival distributions," *AIIE Transactions*, vol. 6, no. 1, pp. 41–49, 1974.

[23] C. Baker and B. P. Dzielinski, "Simulation of a simplified job shop," *Management Science*, vol. 6, no. 3, pp. 311–323, 1960.

[24] Y. R. Nanot, "An experimental investigation and comparative evaluation of priority disciplines in job shop-like queueing networks." DTIC Document, Tech. Rep., 1963.

[25] D. A. Elvers and L. R. Taube, "Time completion for various dispatching rules in job shops," *Omega*, vol. 11, no. 1, pp. 81–89, 1983.

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

[27] E. D. Taillard, "Benchmarks for basic scheduling problems," *european journal of operational research*, vol. 64, no. 2, pp. 278–285, 1993.

[28] Job shop scheduling. [Online]. Available: http://optimizizer.com/TA.php