Improving Job Shop Dispatching Rules Through Terminal Weighting and Adaptive Mutation in Genetic Programming

Michael Riley
School of Engineering
and Computer Science
Victoria University of Wellington
PO Box 600
Wellington, New Zealand
michael.roskill@gmail.com

Yi Mei
School of Engineering
and Computer Science
Victoria University of Wellington
PO Box 600
Wellington, New Zealand
yi.mei@ecs.vuw.ac.nz

Mengjie Zhang
School of Engineering
and Computer Science
Victoria University of Wellington
PO Box 600
Wellington, New Zealand
mengjie.zhang@ecs.vuw.ac.nz

***Abstract*—Automatic design of dispatching rules with Genetic Programming (GP) in job shop scheduling has become more prevalent in recent years. When evolving dispatching rules, choosing a proper terminal set is an important issue. There are a large number of attributes in the job shop that can be taken into account as terminals. However, not all of them are useful to be included. It is not a trivial task to identify the most important attributes out of the entire attribute pool. On the other hand, including all the attributes in the terminal set leads to a huge search space for GP, and makes it hard to find the promising regions of the search space. In this paper, we first demonstrate the differences in importance of attributes by frequency analysis. Then, we propose a terminal weighting algorithm to learn the importance of the terminals on-the-fly, and an adaptive mutation scheme to guide the search to concentrate on the more important terminals. The experimental studies show that the proposed algorithm outperformed its counterpart without terminal weighting and adaptive mutation, in the tested dynamic job shop scheduling, while optimising the mean weighted tardiness. This verifies that focusing on the important terminals will help to search inside more promising regions and lead to better solutions.**

I. Introduction

Job Shop Scheduling (JSS) [1] is an important problem with many real-world applications in cloud computing and manufacturing. It has been investigated intensively in the past decades, and both mathematical programming [2] and meta-heuristics [3]–[6] have been proposed for solving it.

Traditionally scheduling was determined by algorithms that would consider the whole set of available jobs and generate a schedule. This becomes computationally expensive and is increasingly infeasible as the size of scheduling problems increases. In addition, the practical environment is usually dynamic, and there are unpredictable events (e.g. new job arrivals and machine breakdown) happening during the decision horizon. Therefore, it is improper, if not impossible, to use the traditional optimisation methods to adapt to the changing environment due to their slow speed, and computational expense.

To address these issues, dispatching rules have become more commonly used as they can be implemented locally and scale well. Furthermore, dispatching rules can be used in a job shop configuration where the future jobs are not known. Dispatching rules work well in dynamic and stochastic environments. Due to this dispatching rules are found to be more relevant to the demands of highly flexible and responsive production scheduling environments. So far, there have been numerous dispatching rules designed for different job shop scenarios [7]–[11]. However, these manually designed dispatching rules are not good enough, and their performances can vary a lot from one job shop scenario to another.

In recent years, automatic design of dispatching rules using hyper-heuristics [12] has become more prevalent. Briefly speaking, hyper-heuristics are a set of methods that search for heuristics instead of solutions, so that the obtained heuristics can be applied to a series of problem instances rather than a particular one. In the context of JSS, a dispatching rule can be considered a *priority function*, which is used to decide which waiting job is to be processed next at each decision point.

Optimising the priority function can be seen as a symbolic regression [13], which can be solved by Genetic Programming (GP) [14]. So far, there have been a number of studies trying to design GP Hyper-Heuristics (GP-HH) for evolving dispatching rules to do job shop scheduling [15]–[19].

An important issue in evolving dispatching rules with GP-HH is the selection of the terminal set. There are a large number of attributes (e.g. the job-related, machine-related, job-floor-related, and "less myopic" ones) that can be selected as the terminals. Branke et. al have a summary of attributes that are commonly used in job shop scheduling problems [20]. However, different attributes may be suitable for different job shop scenarios, and the importance of the attributes also varies. For example, the due date attribute is often useless when minimising the flowtime-related objectives [15].

It is not trivial to identify the most important terminals before designing a dispatching rule. On the other hand, immediately including all the attributes in the terminal set leads to a huge search space, making it hard for GP to focus on the promising areas. Therefore, it is beneficial to estimate the importance of the terminals and choose them adaptively based on their discovered importance during the GP search process.

The goal of this paper is to develop an approach to learn the importance of the terminals, and guide the search to focus on the important terminals. The resultant algorithm can have

a stronger exploitation capability, and is more likely to find better dispatching rules in the promising regions of the search space. The detailed objectives are listed below:

1) Frequency analysis is conducted to demonstrate that the attributes have different importance/contribution to the dispatching rules. This motivates us to learn such importance and guide the search accordingly.
2) A new GP-HH algorithm is proposed with terminal weighting and adaptive mutation for the job shop attributes. The terminal weighting is used to learn the importance of different attributes, and the task of adaptive mutation is to guide the search using the terminal weights.
3) The new algorithm is compared with its counterpart without terminal weighting and adaptive mutation to show the efficacy of the proposed algorithm.

The remainder of the paper is organised as follows: Section II introduces the background, including the problem description and related works. Then, Section III describes the proposed GP with terminal weighting and adaptive mutation. Section IV carries out the experimental studies, comparing the proposed GP with its traditional counterpart to demonstrate the efficacy of the adaptive mutation. Finally, Section V gives conclusions and directions for future work.

II. Background

A. Job Shop Scheduling Problem

JSS problem can be formally defined by the following: given a set of machines $\mathcal{M} = \{M_1, \ldots, M_m\}$ and a set of jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$, each job J_j consists of several operations $\mathcal{O}_j = (O_{1j}, \ldots, O_{mj})$. Each operation O_{ij} has a processing time t_{ij} and a specific machine π_{ij} that it needs to be completed on. Jobs will only visit each machine once, and may not be processed at more than one machine at a time. Each operation in a job must be completed sequentially. Machines are not allowed to be pre-empted, i.e. a job can not be removed from a machine once it has started. In a dynamic environment, each job J_j has a release date $a_j \geq 0$ and a due date $d_j \geq a_j + \sum_{i=1}^{m} t_{ij}$. The job cannot be processed until its released, and there will be a penalty induced if a job is finished after its due date.

Given the above conditions, the aim of the JSS problem is to make a schedule to finish all the jobs, so that a predefined objective is optimised. The commonly considered objectives include the makespan $C_{\max}$, total flowtime $\sum C_j$ and total weighted tardiness $\sum w_j T_j$.

When applying a dispatching rule to a JSS instance, a schedule is generated through simulation, which is an iterative schedule construction framework. Starting from time zero and an empty schedule, whenever there are idle machines at present and there are available jobs waiting in the queue, the rule calculates the priority of each waiting job using its priority function, and selects the job with the highest priority to be processed. For example, in the well-known SPT rule, the priority function is $-$PT, where PT is the processing time of the imminent operation. Such a process is continued until all the jobs have been processed.

B. GP for Evolving Dispatching Rules

There have been a number of studies in designing GP for evolving dispatching rules. Geiger et al. [16] proposed a GP for a batch processor scheduling problem. Jakobovic and Budin [21] proposed a hybrid GP called GP-3, which evolves a decision tree together with two scheduling trees, one for the bottleneck machines, and the second for the other machines. The algorithm detects the bottleneck machines by the decision tree, and then applies the two scheduling trees accordingly.

Pickardt et al. [22] proposed a two-stage hyper-heuristic. First, a composite rule is obtained by GP. Then, a pool of dispatching rules is constructed with the GP-evolved rule and some other manually designed rules. In the second stage, an EA approach is adopted to select a rule from the pool for each machine. Nguyen et al. [18] considered different representations of dispatching rules, and designed a new GP tree structure that is comprised of decision trees and scheduling trees with both job shop attributes and manually designed rules.

Hildebrandt [15] proposed the dynamic job shop simulation, and suggested using a different random seed in each generation to avoid overfitting. A lookahead scheme was also proposed to allow idle machines to wait so that more flexible schedules can be generated.

Tan and Ho used GP to evolve dispatching rules to solve multi-objective flexible job shop scheduling problem [23] [24]. New terminals were designed to deal with the flexible assignment of the operations to a set of machines.

C. Adaptive Feature Ranking and Selection

Friedlander et al. [25] proposed a terminal weighting scheme and an adaptive mutation based on the terminal weights. Terminal weights are updated generation by generation based on frequency in the individual and the fitness of the individual. The adaptive mutation differs from the traditional mutation in that when sampling the terminals of the newly generated sub-tree, the probabilities of the terminals are set proportional to their weights (i.e. the roulette wheel sampling) rather than equal to each other. Ok et al. [26] proposed the same terminal weighting scheme, and added another phase of categorising the terminals into the relevant terminals and irrelevant terminals. Then, all the terminals of the mutated individual are replaced by only sampling from the relevant terminals.

III. Terminal Weighting and Adaptive Mutation

A. Motivation

After noticing how some human-designed rules (from [7]) could achieve good performance while utilising only a few features we want to confirm that some job shop attributes have more value to dispatching rule construction. In order to verify that some attributes are indeed more important than others, we conducted frequency analysis on the best dispatching rules for

dynamic JSS using mean weighted tardiness as the objective function. Thirty independent experiments were run. It should be noted that there are many parameters that could have influenced the relative importance of the attributes, such as objective function and utilisation level. We have carried out experiments on different scenarios (e.g. different objective functions and utilisation levels), and discovered the same pattern. That is, a small number of attributes appeared much more frequently than others. Fig. (1) shows the representative result for shop utilisation level of 0.9 and due date factor of 1.3, with lowest scoring attributes removed.

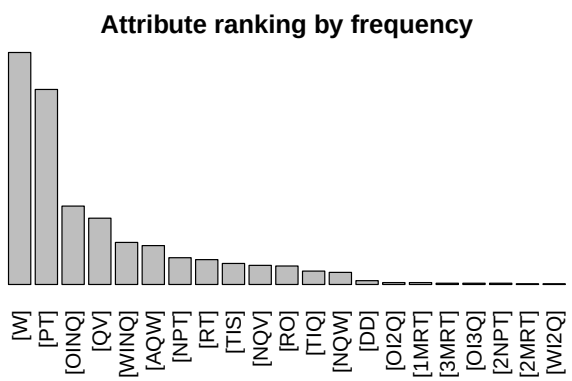

Fig. 1. The frequencies of the attributes in the best dispatching rules of 30 independent runs for the dynamic JSS that minimises the mean weighted tardiness. The utilisation level is 0.9.

This figure confirms that some attributes are more important than others. After the evolutionary process high performing dispatching rules are more likely to contain W and PT than all the rest. These attributes appear to be critical in achieving the highest performance in the run.

Given that some attributes are more important than others, we can do better than utilising the attributes with uniform probability. As the number of attributes increases so does the complexity of the search space. While each new attribute may have a small contribution to make to the performance of the dispatching rule, its contribution might only be useful when combined with other sets of attributes, or the contribution could be very small. Consequently, the search space becomes vast, with huge regions unlikely to have promising results. In this case, better performance can be achieved by focusing on promising areas of the search space. For example, each time an unimportant attribute is included in the dispatching rule it will hinder the GP's search. However, there are potential improvements to be made to the speed of the search, by focusing on the more important attributes and allowing the less important attributes to be ignored, we can search in an informed manner. We want to do this as the evolutionary search proceeds, without making any prior assumptions. This will allow the evolutionary system to reactively prioritise attributes and respond to different JSS problem instances.

On the other hand, when considering the importance of attributes, even the less important attributes have a small contribution to make to the dispatching rule. It might not be sensible to completely ignore this contribution. The optimal dispatching rule could be beyond the ability of the evolutionary system to generate if it prematurely excludes significant attributes.

Due to this tension a balance needs to be found where less important attributes are given a lower priority while highly relevant attributes are readily available to the dispatching rules. This motivates us to propose a better balance in selecting attributes for inclusion in dispatching rules. It is also beneficial to do this in a reactive way without making assumptions before the evolution begins. Based on this motivation, we propose a terminal weighting scheme, which is described in Section III-B.

B. Terminal Weighting

To determine the importance of the attributes, we maintain an attribute weight vector $\boldsymbol{w} = (w_1, w_2, ..., w_n)$ throughout the GP search process, where w_i stands for the weight of the i^{th} attribute. The vector represents the importance of the attributes, and a larger value indicates a higher importance. On initialisation, all the w_i's are set equally to $1/n$, where n is the number of attributes. Then, after each generation's evaluation is completed the weight vector is updated. The weight vector in subsequent generations is determined by the formula that is derived in [25] and [26]:

$$w_i(t) \leftarrow w_i(t-1) + \sum_{j=1}^{P} \frac{count(j,i) \times fitness_j}{\sum_{k=1}^{n} count(j,k)}, \quad (1)$$

where P is the population size $count(j,i)$ is the count of i^{th} attribute in the j^{th} dispatching rule in the population. $fitness_j$ is the standardised fitness of the j^{th} dispatching rule, which is always to be maximised. That is, a larger $fitness$ value indicates a better dispatching rule. Based on Eq. (1), an attribute will be assigned a larger weight if it appears more frequently in dispatching rules with better fitness values. As the search continues, attributes that appear many times in high-fitness individuals will have bigger increase in weight. This allows us to build an increasingly accurate picture of what attributes are important during the search process.

Note that the frequency can be misleading, since there can be many redundant occurrences of irrelevant attributes in a good dispatching rule. For example, if a rule has a priority function of PT + WINQ + $\min(\text{RT}, \text{RT})/\max(\text{RT}, \text{RT})$, the most important attributes are PT and WINQ, while RT is irrelevant. However, from the frequency, RT seems to be the most important attribute since it appears 4 times. Such a phenomenon is particularly more obvious in the early stage of the search due to the random initialisation. Therefore, the weight vector can fluctuate and contain inaccuracy in the early stages of the search. Using the inaccurate weight vector can have a negative effect on the subsequent search and influence it towards unpromising areas. To mitigate this risk and increase the confidence when using the weight vector, we allow 15 generations for the weight vector to stabilise before it is used to guide the search.

After the weights have settled they can be used to guide the search in a way that is sensitive to attribute importance. The result is that the more important attributes are frequently seen and utilized in the resultant dispatching rules. To make use of the attribute weight vector we use adaptive mutation.

C. Adaptive Mutation

There are two components for utilising the weight vector constructed in the previous step to influence the search. First, the weight vector is transformed into a probability vector where each entry represents the chance of an attribute being selected when a new sub-tree is constructed to replace the sub-tree selected for mutation. Second, when the sub-tree is generated during the mutation process, it uses the probability vector to determine how often attributes will be included in the newly generated sub-tree.

In the typical case of genetic programming, the individual will randomly generate a sub-tree to replace one of its existing sub-trees. This new sub-tree typically contains attributes selected from a uniform distribution of all available attributes. When making use of the weights vector, the newly generated sub-tree will contain attributes sampled from a probability distribution that depends on the weight vector. We include a tunable parameter $\lambda \in [0, \infty)$ to control the bias to the more important attributes. The transformation formula from the weight vector to the probability vector is given as follows.

$$p_i = \frac{w_i^\lambda}{\sum_1^k w_k^\lambda}, \tag{2}$$

where p_i is the probability to choose the i^{th} attribute. When $\lambda = 0$, all the attributes are uniformly selected, and the adaptive mutation is reduced to the traditional mutation. When $\lambda \to \infty$, the adaptive mutation tends to always select the attribute with the largest weight. Thus, increasing λ will enhance the bias towards the attributes with larger weights.

As mentioned in Section III-B, the adaptive mutation is not employed until 15 generations have passed. After this the weight vector tends to have stabilised. That is, λ is set to 0 for the first 15 generations.

This technique of attribute weighting is derived from the previous relevant works [26] and [25]. However, it is different from the existing studies in the following aspects.

- Unlike the technique in [26], which categorised the features, and completely removed the irrelevant features, we have a smoother terminal selection scheme by adopting the roulette wheel selection scheme. This way, the weakly relevant attributes still have chance to be included, and to help the search jump out of the local optimum.
- The adaptive mutation in [25] starts from generation 10. We allow a larger stabilising period of 15 generations to avoid misleading information in the early stage of the search.
- We introduced an extra tunable parameter λ while transforming the weight vector to the probability vector. This way, we have more control of the bias towards the heavily weighted attributes.

D. Tuning the Mutation Rate

In traditional GP, mutation is usually used for introducing new genetic materials and increasing diversity [27]. The mutation rate is normally set to a small value (e.g. 10%) to keep a good balance between exploration and exploitation. The existing studies have shown that a small mutation rate works well for many problems.

However, in adaptive mutation, the mutation is not only for increasing diversity, but also to introduce more bias to the important terminals. If the mutation rate is too small, then such a guidance will become almost invisible, and the adaptive mutation will be very close to the traditional mutation. To enhance the effect of adaptive mutation, and increase the frequency of the more important attributes, the mutation rate is increased. This will be examined further in the experimental studies.

IV. Experimental Studies

A. Experiment Design

A dynamic job shop is used to evaluate the dispatching rules generated by the GP process. This section details which parameters are used. During training each dispatching rule is simulated in 4 different job shop configurations, which is the same as that used in [15]. Parameters for the training and test instances are specified in Table I. The training instances are taken from the highest and lowest of the test instances. For each test configuration, 20 replications were randomly generated with different seeds. This allows the test instances to gain a much more trustworthy indication of a particular dispatching rule's performance. The GP process was run on the ECJ [28] system, and the simulation was conducted using the JASIMA simulation environment [15].

Job weights were selected from $\{1, 2, 4\}$ with probabilities (0.2, 0.6, 0.2) respectively. Each job is assigned a weight as it arrives at the job shop. Such a weight setting was inspired by the research in [29].

We used the objective function of mean weighted tardiness to determine the fitness of the dispatching rules, which is calculated as

$$\text{MWT} = \frac{1}{N}\sum_{i=1}^{N} w_i T_i,$$

where w_i is the weight of job i, and T_i is the tardiness of job i, which is defined as $T_i = \max(C_i - d_i, 0)$. Here, C_i and d_i are the completion time and due date of job i. N is the number of jobs.

Due to the stochastic nature of the simulations being initialised with different random seeds absolute fitness values would exhibit jitter. A reference rule was employed to provide a consistent measuring point regardless of the natural variation in the stochastic simulation process. We selected Weighted Shortest Processing Time (WSPT) as our reference rule. Fitness scores were determined by the formula:

$$fitness_j = \frac{1}{4}\sum \frac{\text{MWT}(R_j)}{\text{MWT(WSPT)}},$$

where $fitness_j$ is the fitness of individual j, and R_j is the dispatching rule individual j represents.

TABLE I
JOBSHOP PARAMETERS

Training	
utilization levels	0.8, 0.95
min/max operations	(2,10) missing operations (10,10) full operations
due date factor	1.3
time units per operation	uniform distribution [0-49]
job routing	random, no machine visited twice or simultaneously
job arrival interval	Poisson distribution to achieve desired shop utilization level
warm-up job count	500
end simulation after	2500 jobs completed
data collection	post-warmup 2000 jobs
Testing	
utilization levels	0.8, 0.85, 0.9, 0.95
min/max operations	(2,10) missing operations (10,10) full operations
due date factor	1.3
time units per operation	uniform distribution [0-49]
job routing	random, no machine visited twice or simultaneously
replication count	20
warm-up jobs count	1000
end simulation after	5000 jobs completed
data collection	post-warmup 4000 jobs

B. Parameter Settings

Table II gives an overview of evolutionary parameters. The function set includes: $+$, $-$, $*$, $\%$, $\min$, $\max$, ifte, among which $+$, $-$, $*$ are the standard arithmetic operators. The protective division $\%$ returns 1 if the denominator is 0. The $\min$ and $\max$ functions take two arguments and return the minimum and maximum respectively. The ifte operator takes three arguments, returns the second argument if the first argument is positive, and the third argument otherwise.

TABLE II
THE PARAMETER SETTING OF GP.

Evolution Parameters	
populations size	1024
max tree depth	8
generations	51
function set	$+$, $-$, $*$, $\%$, min, max, ifte
terminal set	(see Table III)
crossover rate	0.8
mutation rate	0.1
reproduction rate	0.1

The terminal set is shown in Table III, which consists of 24 attributes. Note that xMRT, xNPT, QIxQ, WIxQ stand for a series of attributes (depending on the range of x) instead of a single one. The crossover, mutation and reproduction rates are set for the baseline GP. As discussed in Section III-D, when adopting the adaptive mutation, a larger mutation rate is tested as well. Specifically, in additional to the values given in Table II, the GP with adaptive mutation is also tested with the crossover rate of 0.65, mutation rate of 0.3, and reproduction rate of 0.05. For the sake of convenience, the original GP is referred to as "Baseline", while the GP with adaptive mutation is denoted as AM-GP. It has two parameters, λ and P_m, where λ is the power coefficient used in Eq. (2), and P_m is the mutation rate.

TABLE III
THE TERMINAL SET USED IN GP.

Attribute	Description
NOW	Current simulation time
W	Weight of job
PT	Processing time of current operation
RO	Remaining operations in job
RT	Remaining time in job
DD	Due date of job
TIQ	Time in current queue
TIS	Time in system
NPT	Next operation's processing time
WINQ	Work in job's next queue
OINQ	Number of operations in job's next queue
NQW	Average wait time of last 5 jobs completed at the next machine
AQW	Average wait time of last 5 jobs completed anywhere in the shop
QV	Ratio of shortest to longest job in current queue
NQV	Ratio of shortest to longest job in next queue
xMRT	$[1 \leq x \leq 3]$ Ready time of machine x steps ahead
xNPT	$[2 \leq x \leq 3]$ Processing time of operation x steps ahead
OIxQ	$[2 \leq x \leq 3]$ Operation in queue x steps ahead
WIxQ	$[2 \leq x \leq 3]$ Work in queue x steps ahead

C. Results and Discussions

Table IV shows the mean and standard deviation (in parenthesis) of the test performance of the compared algorithms in different test sets. From the table, we can see a trend where every rule designed by GP can perform better than WPST. This margin increases as the utilization level of the job shop increases.

The standard deviation also increases as the utilization level increases. Since the problem becomes harder when the utilization rate increases, the results show that tougher job shop conditions reduce the stability of results. Across all the λ values, a larger mutation rate tends to lead to a better result. This demonstrates the importance of increasing the mutation rate while using the adaptive mutation. It can be seen that the AM-GP with $\lambda = 1$ and $P_m = 0.3$ obtained the best average test performance in 5 out of the 8 test sets, followed by $(\lambda, P_m) = (5, 0.3)$. The parameters $(2, 0.3)$ performed the best on the remaining one test set, whose test performance is very close to that of $(1, 0.3)$. The AM-GP with $\lambda = 10$ achieved the worst results among all the parameter settings. This shows that setting λ to 10 makes the search too greedy, and the population loses its diversity.

Wilcoxon's rank sum test was conducted between each AM-GP and the baseline GP, and the p-value of the test results are shown in Table V. Unfortunately, the p-values do not show much significance on the differences, p-values less than 0.05 were only found in (0.95, miss) and (0.95, full) of AM-GP with $(5, 0.3)$. When taking a closer look at Table IV, we found that the reason is that the standard deviation is usually very large compared to the difference (about 0.02 for the difference of at most 0.01 in most cases). To understand the distribution of

TABLE IV
THE MEAN AND STANDARD DEVIATION (IN PARANTHESIS) OF THE TEST PERFORMANCE OF THE COMPARED ALGORITHMS IN DIFFERENT TEST SETS. THE BEST MEAN RESULTS ARE MARKED IN BOLD.

(util, ops)	Baseline	(λ, P_m) for AM-GP							
		(1,0.1)	(1,0.3)	(2,0.1)	(2,0.3)	(5,0.1)	(5,0.3)	(10,0.1)	(10,0.3)
(0.8, miss)	0.978 (0.014)	0.978 (0.014)	**0.972** (0.011)	0.980 (0.013)	0.973 (0.013)	0.982 (0.019)	0.977 (0.012)	0.982 (0.013)	0.978 (0.015)
(0.8, full)	0.977 (0.015)	0.977 (0.015)	**0.972** (0.012)	0.979 (0.015)	0.972 (0.014)	0.981 (0.020)	0.977 (0.013)	0.981 (0.015)	0.976 (0.016)
(0.85, miss)	0.978 (0.017)	0.979 (0.016)	**0.972** (0.014)	0.983 (0.017)	0.972 (0.016)	0.983 (0.021)	0.973 (0.016)	0.983 (0.015)	0.976 (0.016)
(0.85, full)	0.972 (0.019)	0.972 (0.018)	0.966 (0.016)	0.976 (0.019)	**0.965** (0.016)	0.976 (0.022)	0.971 (0.017)	0.977 (0.018)	0.971 (0.018)
(0.9, miss)	0.961 (0.023)	0.962 (0.023)	**0.953** (0.018)	0.967 (0.022)	0.954 (0.021)	0.965 (0.022)	0.953 (0.021)	0.968 (0.020)	0.959 (0.021)
(0.9, full)	0.958 (0.025)	0.960 (0.023)	**0.951** (0.017)	0.964 (0.024)	0.952 (0.021)	0.963 (0.024)	0.956 (0.021)	0.966 (0.022)	0.957 (0.022)
(0.95, miss)	0.956 (0.027)	0.957 (0.023)	0.944 (0.018)	0.960 (0.023)	0.950 (0.023)	0.956 (0.025)	**0.935** (0.028)	0.963 (0.022)	0.954 (0.022)
(0.95, full)	0.954 (0.028)	0.957 (0.025)	0.944 (0.019)	0.961 (0.025)	0.947 (0.024)	0.954 (0.027)	**0.939** (0.028)	0.960 (0.024)	0.951 (0.023)

TABLE V
THE p-VALUES OF THE WILCOXON'S RANK SUM TEST BETWEEN EACH AM-GP AND THE BASELINE GP. THE ONES SMALLER THAN THE SIGNIFICANCE LEVEL OF 0.05 IS MARKED IN BOLD.

(util, ops)	(1,0.1)	(1,0.3)	(2,0.1)	(2,0.3)	(5,0.1)	(5,0.3)	(10,0.1)	(10,0.3)
(0.8, miss)	0.58	0.07	0.90	0.06	0.77	0.65	0.40	0.79
(0.8, full)	0.74	0.14	0.76	0.11	0.79	0.84	0.40	0.68
(0.85, miss)	1.00	0.06	0.57	0.06	0.46	0.10	0.25	0.38
(0.85, full)	0.92	0.19	0.46	0.07	0.74	0.81	0.23	0.90
(0.9, miss)	0.89	0.10	0.36	0.18	0.54	0.18	0.14	0.82
(0.9, full)	0.65	0.34	0.31	0.43	0.40	0.98	0.11	0.89
(0.95, miss)	0.95	0.05	0.65	0.41	0.76	**0.00**	0.28	0.69
(0.95, full)	0.66	0.10	0.33	0.30	0.86	**0.00**	0.30	0.68

the results better, we made the boxplots of all the algorithms on the test sets, which is shown in Fig. 2.

From the boxplots, we can see clearly that all the algorithms have a very large variance. This may be due to the large number of attributes included in the terminal set of GP. In our experiments, there are 24 attributes in the terminal set, which can lead to a huge search space. In such a huge search space, it is difficult to consistently identify the promising regions of the search space and obtain high-quality solutions.

Additionally, it is observed that when $P_m = 0.3$ and $\lambda \geq 2$, there are many outliers in most cases. In other words, although the adaptive mutation can help GP find better solutions most of the time, its behaviour is still unstable. It is known that the frequency may not be accurate enough, since the irrelevant attributes can have many redundant occurrences. This can increase the risk of emphasizing irrelevant attributes by mistake, and consequently reduce the stability. Despite the outliers, when comparing the boxplots of Baseline (the first one) and $(1, 0.3)$ (the third one), we can see that the distribution of $(1, 0.3)$ is always below that of Baseline. However, $(1, 0.3)$ still suffered from an outlier, which is shown as the dot for (0.85, miss), (0.85, full), (0.9, miss), (0.9, full), (0.95, miss) and (0.95, full).

To understand why the outlier occurred, we plot the curve of the weights of the attributes throughout the corresponding run. The result is shown in Fig. 3. It can be seen that the weights of PT and W, which are the two most important attributes as shown in Fig. 1, are not substantially distinguished from the other attributes (e.g. AQW and WI2Q). This means that the adaptive mutation was not properly guided by the weight vector, which has not emphasized PT and W enough. Fig. 4 shows the weight curves of the runs of Baseline and (1, 0.3) that produce the rules with high test performance. It can be seen that in these two runs, the weights of W and PT are much higher than that in Fig. 3. This observation is consistent with our intuition, which shows that the test performance of the GP-evolved rules largely depends on whether the search successfully identifies the correct direction and focuses on the more important attributes (W and PT in this case).

Table VI shows the best test performance obtained by the algorithms on the test sets. It can be seen that (5, 0.3) performed the best, outperforming the baseline GP in 6 out of the 8 test sets. This demonstrates that the adaptive mutation can enhance the strength of the search in the best case.

In summary, we have the following observations and discussions from the results.

- Using adaptive mutation tends to lead to better test performance, although the difference is not significant in most cases.
- The variation of the test performance of all the algorithms is relatively high. This may be due to the large number of terminals (24 in the experiment), which led to a huge search space.
- In such a huge search space, it is hard to identify important attributes, either by uniform sampling or adaptive mutation. As a result, we see outliers for AM-GP.
- The formula to update the weights can also be problematic. First, the frequency can be misleading, since the less relevant attributes can have many redundant occurrences in the priority function. Second, in Eq. (1), the fitness and frequency are of different magnitude. The scale of frequency is much larger than that of the fitness. This will enhance the misleading effect of frequency.

D. Further Analysis

Fig. 5 shows the detail of the priority function obtained by the outlier point of $(1, 0.3)$. One can see that the first argument of the ifte operator is $\min(\mathrm{W/T}, \mathrm{AQW} + \mathrm{WI2Q})$, which is positive most of the time. Then, the rule can be mostly simplified to $\min(\mathrm{W/T}, \mathrm{OINQ} + \mathrm{AQW})$, which is very close to $\mathrm{W/PT}$. Therefore scheduling decisions made by this rule will match those made by WSPT in the vast majority of cases. This explains why the test performance of this outlier

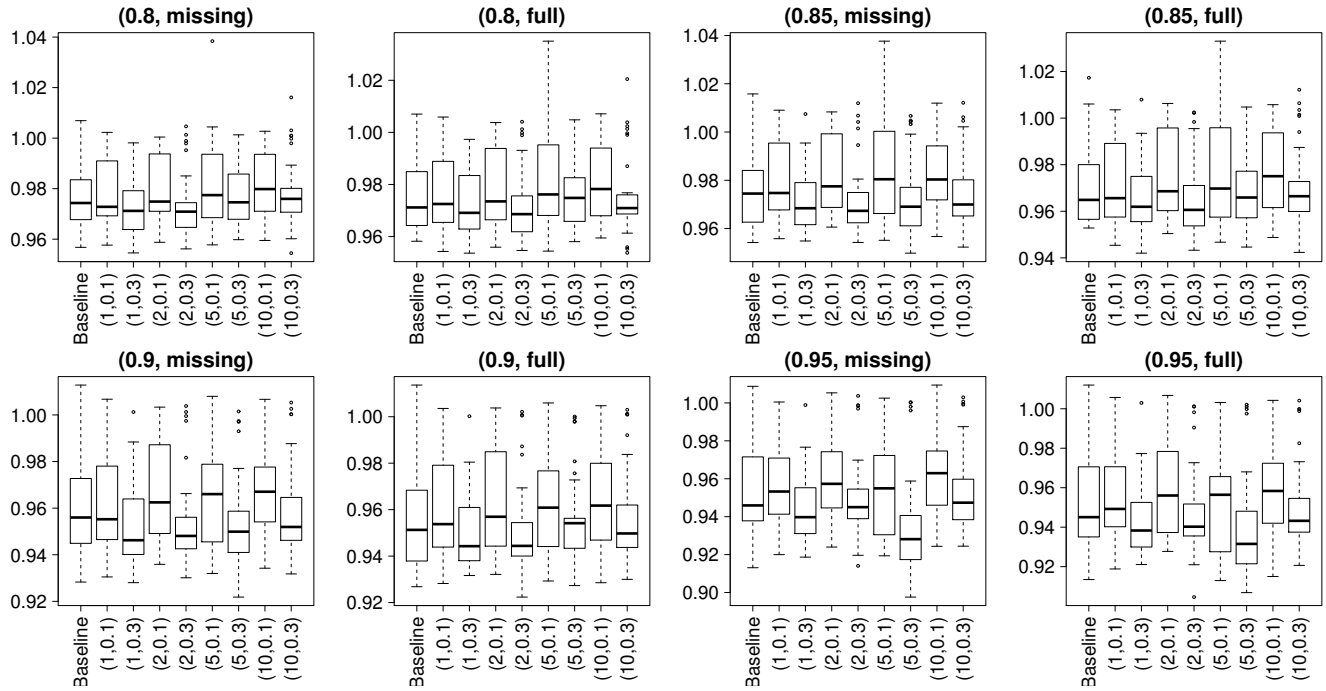

Fig. 2. Boxplots of the algorithms for all the test sets.

TABLE VI
THE BEST TEST PERFORMANCE FOUND BY ALL THE ALGORITHMS ON THE TEST SETS. THE BEST VALUES ARE MARKED IN BOLD.

(util, ops)	Baseline	(1,0.1)	(1,0.3)	(2,0.1)	(2,0.3)	(5,0.1)	(5,0.3)	(10,0.1)	(10,0.3)
(0.8, miss)	0.9568	0.9576	0.9546	0.9588	0.9562	0.9577	0.9598	0.9595	**0.9545**
(0.8, full)	0.9582	0.9543	**0.9536**	0.9559	0.9547	0.9544	0.9581	0.9595	0.9538
(0.85, miss)	0.9542	0.9558	0.9548	0.9606	0.9542	0.9551	**0.9498**	0.9567	0.9523
(0.85, full)	0.9528	0.9454	**0.9420**	0.9504	0.9433	0.9467	0.9446	0.9487	0.9423
(0.9, miss)	0.9283	0.9305	0.9280	0.9359	0.9301	0.9320	**0.9218**	0.9342	0.9318
(0.9, full)	0.9268	0.9282	0.9316	0.9321	**0.9224**	0.9293	0.9273	0.9285	0.9300
(0.95, miss)	0.9131	0.9200	0.9187	0.9240	0.9141	0.9194	**0.8976**	0.9244	0.9244
(0.95, full)	0.9135	0.9188	0.9211	0.9277	**0.9046**	0.9129	0.9068	0.9150	0.9206

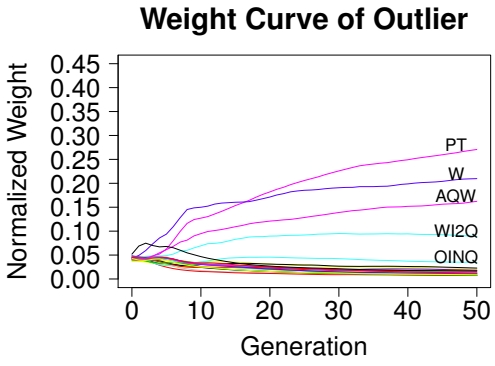

Fig. 3. The weight curves of the run that produced the outlier for AM-GP-$(1, 0.3)$.

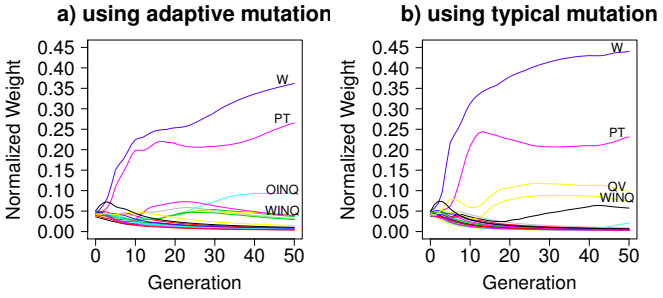

Fig. 4. Example weight curves of high performance dispatching rules as evolution progresses.

is around 1.0. In other words, in this case, GP failed to find better rules than the reference rule.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we investigated using GP to evolve dispatching rules for dynamic JSS. Specifically, we studied the importance of different terminals in GP to evolve high-quality rules, and proposed a terminal weighting scheme to learn the importance

```
(ifte (min (% W PT) (+ AQW WI2Q))
(min (% W PT) (+ OINQ AQW))
(min (% (% W PT) PT) (% W PT)))
```

Fig. 5. Outlier dispatching rule

of different terminals using an adaptive mutation to guide the search, based on the terminal weights. The experimental results showed that the terminal weighting and adaptive mutation schemes can improve the test performance of the GP-evolved rules. This demonstrates the efficacy of treating the terminals differently based on their contribution rather than randomly selecting them.

The proposed algorithm still has some drawbacks. Its behaviour is unstable, and this resulted in many outliers. This is due to the large number of attributes and the use of frequency in the terminal weighting, which is not accurate enough and may mislead the search to focus on the irrelevant attributes with many redundant occurrences in the individuals. To overcome the drawbacks, we will design a more accurate terminal weighting scheme that can truly reflect the importance of attributes, and improve the adaptive mutation by increasing the influence of the terminal weighting in a proper way. In addition, other job shop scenarios will be examined to verify the generality of the proposed algorithm.

REFERENCES

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

[2] P. Brucker, B. Jurisch, and B. Sievers, "A branch and bound algorithm for the job-shop scheduling problem," *Discrete applied mathematics*, vol. 49, no. 1, pp. 107–127, 1994.

[3] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of an international conference on genetic algorithms and their applications*, vol. 140. Carnegie-Mellon University Pittsburgh, PA, 1985.

[4] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian, "Ant system for job-shop scheduling," *Belgian Journal of Operations Research, Statistics and Computer Science*, vol. 34, no. 1, pp. 39–53, 1994.

[5] G. Zhang, L. Gao, and Y. Shi, "An effective genetic algorithm for the flexible job-shop scheduling problem," *Expert Systems with Applications*, vol. 38, no. 4, pp. 3563–3573, 2011.

[6] 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.

[7] 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*, vol. 50, no. 15, pp. 4255–4270, 2012.

[8] 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.

[9] T. Raghu and C. Rajendran, "An efficient dynamic dispatching rule for scheduling in a job shop," *International Journal of Production Economics*, vol. 32, no. 3, pp. 301–313, 1993.

[10] 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.

[11] A. P. Vepsalainen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management science*, vol. 33, no. 8, pp. 1035–1047, 1987.

[12] 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*, vol. 64, no. 12, pp. 1695–1724, 2013.

[13] E. J. Vladislavleva, G. F. Smits, and D. Den Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 2, pp. 333–349, 2009.

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

[15] 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*. ACM, 2010, pp. 257–264.

[16] C. 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.

[17] D. Jakobović and K. Marasović, "Evolving priority scheduling heuristics with genetic programming," *Applied Soft Computing*, vol. 12, no. 9, pp. 2781–2789, 2012.

[18] 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*, vol. 17, no. 5, pp. 621–639, 2013.

[19] R. Hunt, M. Johnston, and M. Zhang, "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 618–625.

[20] B. J., S. Nguyen, C. 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

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

[22] 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*, vol. 145, no. 1, pp. 67–77, 2013.

[23] N. Ho and J. Tay, "Evolving dispatching rules for solving the flexible job-shop problem," in *IEEE Congress on Evolutionary Computation*, vol. 3. IEEE, 2005, pp. 2848–2855.

[24] J. Tay and N. 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.

[25] 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)*. IEEE, 2011, pp. 941–948.

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

[27] D. R. Munroe, "Genetic Programming The ratio of Crossover to Mutation as a function of time," 2004. [Online]. Available: http://mro.massey.ac.nz/handle/10179/4429

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

[29] M. Pinedo and M. Singer, "A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop," *Naval Research Logistics*, vol. 46, no. 1, pp. 1–17, 1999.