A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem

Article *in* IEEE Transactions on Evolutionary Computation · January 2012

DOI: 10.1109/TEVC.2012.2227326

Su Nguyen
La Trobe University
73 PUBLICATIONS **1,076** CITATIONS

SEE PROFILE

Mengjie Zhang
Victoria University of Wellington
678 PUBLICATIONS **9,466** CITATIONS

SEE PROFILE

Mark Johnston
University of Worcester
111 PUBLICATIONS **1,768** CITATIONS

SEE PROFILE

A Computational Study of Representations in Genetic Programming to Evolve Dispatching Rules for the Job Shop Scheduling Problem

Su Nguyen, Mengjie Zhang, *Senior Member, IEEE*
Mark Johnston, *Member, IEEE* and Kay Chen Tan, *Senior Member, IEEE*

***Abstract*—Designing effective dispatching rules is an important factor for many manufacturing systems. However, this time-consuming process has been performed manually for a very long time. Recently, some machine learning approaches have been proposed to support this task. In this paper, we investigate the use of genetic programming for automatically discovering new dispatching rules for the single objective job shop scheduling problem (JSP). Different representations of the dispatching rules in the literature and newly proposed in this work are compared and analysed. Experimental results show that the representation which integrates system and machine attributes can improve the quality of the evolved rules. Analysis of the evolved rules also provides useful knowledge about how these rules can effectively solve JSP.**

***Index Terms*—Genetic Programming, job shop scheduling, hyper-heuristic, dispatching rule.**

I. Introduction

In the field of sequencing and scheduling, Job Shop Scheduling is one of the most popular problems because of its complexity and applicability in real world situations. In the job shop scheduling problem (JSP), given a set of machines and a set of jobs with various pre-determined routes through the machines, the objective is to find the schedule of jobs that minimises certain criteria such as makespan, maximum lateness, total weighted tardiness, etc. Different approaches have been proposed to solve this problem and they can be classified into two main categories [1]: (1) theoretical studies of optimisation methods which are usually restricted to static problems and (2) experimental studies of heuristics or dispatching rules to deal with both static and dynamic problems. Although the dispatching rules do not guarantee to provide optimal solutions for the problems, they have been applied extensively in research and practice because of their simplicity and ability to cope with the dynamic environment. Different from optimisation approaches which represent the scheduling solution in a very sophisticated way in order to employ specialised techniques to solve the scheduling problem, a dispatching rule provides a way to perform a scheduling task which is understandable to shop floor operators. Normally, a dispatching rule is considered as a simple function that determines the priorities of jobs in the queue of a machine and decides which one should be processed next. The popularity of the dispatching rule is derived from the fact that it can be easily modified when real world aspects such as setup time, release time or parallel machines are considered. Another aspect that makes dispatching rules attractive to both researchers and practitioners is that they do not have the scalability problems which are a big issue of almost all optimisation methods. Moreover, effective dispatching rules can also be used to create good initial solutions for optimisation methods.

Su Nguyen, Mengjie Zhang, and Mark Johnston are with the Evolutionary Computation Research Group at Victoria University of Wellington, PO Box 600, Wellington, New Zealand. Kay Chen Tan is with the Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, 117576, Singapore.

Many studies have been done in order to discover new effective dispatching rules. One of the most straightforward ways to improve the performance of dispatching rules without affecting their simplicity is to use a combination of simple dispatching rules. Another way to employ different dispatching rules is to monitor the status of jobs in the system and make a change from one dispatching rule to another as planned. For example, FIFO/SPT will apply first-in-first-out (FIFO) when the jobs in the queue of the considered machine have been waiting for more than a specific time and shortest-processing-time (SPT) will be applied otherwise. This combination, even though very simple, can take advantage of each rule at proper decision making moments and is normally better than the application of a single dispatching rule. Another approach to improving the performance of dispatching rules is to create composite dispatching rules (CDR) [2], [3] which provide heuristic combinations of simple rules basically in the form of sophisticated human-made priority functions of various scheduling parameters (processing times, waiting times, etc.).

Although the combinations of different simple dispatching rules may create better rules, the task of developing such rules is complicated and time consuming. Recently, machine learning methods have been proposed to facilitate this task. Some genetic programming (GP) methods [4], [5], [6], [7], [8], [9], [10] have been proposed to evolve dispatching rules. In these methods, the GP programs are used to calculate the priority of jobs in the queue of each machine and the job with highest priority will be processed first. The experimental results showed that the dispatching rules evolved by GP can outperform simple dispatching rules. Li and Olafsson [11] applied a data mining technique to discover new dispatching rules based on production data. Data engineering was also considered in their research in order to create more useful attributes besides the ones recorded as part of the raw production data. The results show that the discovered decision rules can accurately replicate the dispatching list obtained by specific rules. Ingimundardottir and Runarsson [12] proposed a supervised learning approach which tries to discover new dispatching rules using the characteristics of optimal solutions. The learned linear priority dispatching rules showed better results than simple rules.

With its ability to evolve sophisticated rules and flexible representations, GP is a very promising method for automatically generating dispatching rules for the JSP. Moreover, if properly implemented, GP can evolve effective *reusable rules* to solve the JSP. This feature makes GP more attractive than other evolutionary computation and meta-heuristic methods where only a *disposable solution* is obtained for each problem instance. One of the key factors that can enhance the performance of dispatching rules is the incorporation of machines and shop status into the rules in order to make more informative sequencing decisions. Miyashita [8] proposed three models to evolve dispatching rules with GP and showed that the two models which evolved different dedicated dispatching rules for different machines in the shop are better than the model which only employed a common dispatching rule for all machines. One of the drawbacks with this method is that evolving different dedicated dispatching rules would significantly increase the computational time of the proposed GP system and can also lead to overfitting problems [8]. Another disadvantage is that the representation in these models required the specific knowledge of the environment which can be changed in real situations. For that reason, it is very difficult to create more generalised and effective rules to deal with such a complicated problem as JSP. Jakobovic and Budin [6] proposed a more dynamic method to incorporate machine attributes into the evolved dispatching rules. Their proposed GP-3 system evolved three components including two dispatching rules and a (discrimination) function to determine which dispatching rule the considered machine should employ based on the attributes of that machine. The results showed that GP-3 was better than the GP method that only evolved dispatching rules. Unfortunately, no analysis or description of the evolved dispatching rules were provided in the paper. It should be noted that these existing studies emphasised not only the importance of machine attributes in the evolved rules but also the importance of how to embed these attributes into the evolved dispatching rules. Although some GP systems for evolving dispatching rules have been proposed, no detailed comparison and analysis of those representations have been done, which is an important issue in GP.

In this study, we use GP to evolve *adaptive dispatching rules* (ADR) for the static JSP with makespan and total weighted tardiness as objective functions. Basically, an ADR is a combination of different dispatching rules and it is "adaptive" because the master rule will choose a specific dispatching rule to sequence jobs in the queue based on the status of the machines at each decision making step. Three representations of dispatching rules are proposed and tested with the GP system. The objectives of this study are to: (1) investigate the performance of the GP system with different types of representations; (2) compare the performance of the evolved rules with well known dispatching rules both on the training set and test set; and (3) analyse the performance of the proposed algorithm as well as the evolved rules and compare evolved rules with improvement heuristics.

The remainder of this paper is organised as follows. In the next section, the background of JSP and heuristic generation methods is given. Section III will provide a detailed description of the proposed method. The experimental setting is presented in Section IV and the results will be shown in Section V. Some insights regarding the proposed method and the evolved rules will be discussed further in Section VI. Section VII offers some conclusions and future research directions.

II. Background

This section provides a brief review of JSP and hyper-heuristic methods for heuristic generation which provides essential background for later sections.

A. Job Shop Scheduling Problem (JSP)

This study concentrates on minimisation of the makespan and total weighted tardiness, known as $Jm||C_{max}$ and $Jm||\sum w_j T_j$ in the JSP literature. In this case, the shop includes a set of $\mathcal{M}$ machines and $\mathcal{N}$ jobs that need to be scheduled. Each job j has its own pre-determined route through a sequence of machines to follow and its own processing time at each machine it visits. In this paper, jobs are only allowed to visit a machine at most once; and therefore no recirculation can occur. Some basic definitions and notations are presented below that will be used in the rest of this paper.

Parameters:

- $O_j = \{o_{j,1}, \ldots, o_{j,l}, \ldots, o_{j,N_j}\}$: the set of all operations of job j where $o_{j,l}$, is the l^{th} operation of job j and N_j is the number of operations of job j.
- w_j: the weight given to job j in the weighted tardiness objective function.
- d_j: the due date assigned to job j.
- $p(\sigma)$: the processing time of operation σ.
- $m(\sigma)$: the machine that processes operation σ.
- $next(\sigma)$: the next operation of the job that contains σ or $null$ if σ is the last operation of that job (if $\sigma = o_{j,l}$ then $next(\sigma) = o_{j,l+1}$).

Variables:

- R_k: the ready time of machine k, which is the time that the machine becomes idle; in this study, all machines are idle at the beginning.
- $r(\sigma)$: the ready time of operation σ, which is the release time of job j (the time that job j is allowed to start, considered to be zero in all instances in the experiments) for the first operation or the completion time of its preceeding operation for other operations.
- C_j: the completion time of job j. Makespan $C_{max} = max(C_1, \ldots, C_{\mathcal{N}})$.
- T_j: the tardiness of job j and $T_j = max(C_j - d_j, 0)$. The total weighted tardiness is $\sum w_j T_j$.

```
1: $\Omega \leftarrow \{o_{1,1}, o_{2,1}, \ldots, o_{\mathcal{N},1}\}$
2: **repeat**
3:     let $t(\Omega) = \min_{\sigma\in\Omega}\{max\{r(\sigma), R_{m(\sigma)}\} + p(\sigma)\}$
4:     let $\sigma^*$ be the operation that minimum is achieved, $m^* = m(\sigma^*)$, and $\Omega^* = \{\sigma \in \Omega | m(\sigma) = m^*\}$
5:     let $S(m^*) = max\{min_{\sigma\in\Omega^*}\{r(\sigma)\}, R_{m^*}\}$
6:     let $\Omega' = \{\sigma \in \Omega^* | r(\sigma) \leq S(m^*) + \alpha(t(\Omega) - S(m^*))\}$
7:     apply dispatching rule $\Delta$ on $\Omega'$ to find the next operation $\sigma'$ to be scheduled on $m^*$
8:     remove $\sigma'$ from $\Omega'$ and include $next(\sigma)$ into $\Omega$ if $next(\sigma) \neq null$
9: **until** all operations have been scheduled
```

Fig. 1. Generic procedure to construct a schedule for JSP.

1) Active schedules and non-delay schedules: In JSP, a schedule is called *active* if it cannot be altered to make some operations complete earlier without delaying the completion time of other operations. Active schedules were first proposed by Giffler and Thompson [13] in their seminal work and it has been proven that an optimal solution of JSP must be an active schedule. Schedules are called *non-delay* schedules where no machine is allowed to be idle when there are jobs in the queue. A non-delay schedule is more restricted than an active schedule and the set of non-delay solutions may not include the optimal solution. Non-delay schedules are often applied in experiments using simulation since a machine will immediately process all jobs in its queue in the order determined by a specific sequencing rule.

Figure 1 shows a generic procedure to construct an active schedule, a non-delay schedule or a hybrid of both active and non-delay schedules. In this procedure, Ω contains all the operations that are ready to be scheduled. The procedure will first determine the next operation that would complete first if scheduled and the corresponding machine m^* (ties are broken arbitrarily). The non-delay factor $\alpha \in [0, 1]$ controls the look ahead ability of the algorithm and decides which jobs should be considered by the dispatching rule Δ. If $\alpha = 0$, the procedure can only construct non-delay schedules and only operations that have already joined the queue are considered for scheduling. On the other hand, if $\alpha = 1$, the procedure will consider all the potential jobs that are ready to join the queue of machine m^* before the earliest completion time of m^*. In general, α determines the interval of time that a machine is allowed to wait even when there are operations in the queue. As a result, there are fewer operations to be considered in each step of the algorithm when α is small.

2) Job shop scheduling techniques: Over the last few decades, a large number of techniques have been applied to JSP, ranging from simple heuristics to artificial intelligence and mathematical techniques [14]. Because of the complexity of JSP, finding optimal solutions can be very time-consuming particularly for large and complex shops. The research on meta-heuristics approaches for scheduling has been very active in the last two decades, mostly with makespan as the objective function, i.e., $Jm||C_{max}$. Local search based approaches such as large step optimisation [15], tabu search [16], and guided local search [17] have shown very promising results in solving the static JSP. The focus of these approaches is on the development of efficient neighbourhood structures and diversifying strategies to escape from local optima. Since neighbourhood structures play an important role in these approaches, the neighbourhood structures and their related operators have to be redesigned in order to incorporate real world constraints; even then it is still questionable whether they produce superior results.

A more general alternative for solving JSP is the use of evolutionary computation methods. Genetic Algorithm (GA) is one of the most popular approaches in this line of research (refer to [18] for a review of GA approaches for JSP). Besides $Jm||C_{max}$, research on other objective functions have also been considered in the literature, especially due date related objectives due to the need to improve the delivery performance in modern manufacturing systems. For $Jm||\sum w_jT_j$, several methods have been proposed in the literature and shown promising results. For example, Kreipl [19] proposed an efficient large step random walk (LSRW) method, which is still one of the best local search methods to deal with $Jm||\sum w_jT_j$. Zhou et al. [20] introduced a general framework using a genetic algorithm and heuristic rules to solve a similar problem and showed that it is better than the pure genetic algorithm. Even though the proposed hybrid GA provides solutions that are inferior to those obtained by LSRW [19], it is still very promising since it is capable of solving a variety of scheduling problems without major redesign.

Although there have been many breakthroughs in the developments of exact and approximate approaches for JSP, these approaches are mainly focused on static problems and simplified job shop environments. General approaches like GA can be extended to solve problems with realistic constraints, but the major drawback is its weak computational efficiency. Moreover, as pointed out in McKay et al. [21], the conventional operational research and artificial intelligence approaches are often not applicable to the dynamic characteristics of the actual situation because these approaches are fundamentally based on static assumptions. For that reason, simple dispatching rules have been used consistently in practice because of their ability to cope with the dynamics of shop changes [22]. There have been a large number of rules proposed in the literature and they can be classified into three categories [14]: (1) simple priority rules, which are mainly based on the information related to the jobs; (2) combinations of rules that are implemented depending on the situation that exists on the shop floor; and (3) weighted priority indices which employ more than one piece of information about each job to determine the schedule. Composite dispatching rules (CDR) [2], [3] can also be considered as a version of rules based on weighted priority indices, where scheduling information can be combined in more sophisticated ways instead of linear combinations.

B. Hyper-Heuristics for Heuristic Generation

Hyper-heuristics (HH) are a new search method and have attracted a lot of attention from the research community. A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems [23]. Heuristic selection and heuristic generation are currently the two main research methodologies in HH [24]. The focus of this paper is on hyper-heuristics for heuristic generation. In order to generate a new heuristic, the HH framework must be able to combine various small components (normally common statistics or operators used in pre-existing heuristics) and these heuristics are trained on a training set and evolved to become more effective.

1) Genetic Programming based hyper-heuristics (GP-HH): Recently, Genetic Programming (GP) has become popular in the field of hyper-heuristics and it is known as genetic programming based hyper-heuristics (GP-HH) [25]. Because GP is able to represent and evolve complex programs or rules, it naturally becomes an excellent candidate for heuristic generation. Bolte and Thonemann [26] proposed a GP system to evolve annealing schedule functions in simulated annealing to solve the quadratic assignment problem (QAP). The experimental results showed that the method with GP as a meta-algorithm can find near optimal solutions for QAP. Fukunaga [27] used GP to evolve variable selection heuristics in each application of a local search algorithm for the satisfiability (SAT) problem and showed competitive results when compared with other heuristics. Bader-El-Den and Poli [28] also employed GP to evolve disposable local search heuristics to solve SAT for a specific set of instances and the evolved heuristics show very competitive results as compared to other well-known evolutionary or local search heuristics. Burke et al. [29], [30] proposed a GP-HH framework to evolve construction heuristics for online bin packing. The basic idea of this GP system is to generate a priority function from static and dynamic statistics of the problem to decide where the considered pieces should be placed. The experimental results showed that human designed heuristics can be obtained by GP. A similar idea was also applied to evolve two dimensional strip packing heuristics and showed very good results [31]. Keller and Poli [32], [33] proposed a grammar based linear genetic programming method to solve the travelling salesman problem. Several grammars are introduced, including ones with a looping construct. Bader-El-Den et al. [34] introduced a sophisticated grammar based GP for evolving timetabling heuristics. The evolved heuristics were able to produce competitive results when compared with some existing search methods in the literature. Burke et al. [35] proposed a grammatical evolution method for automatic design of local search heuristics for cutting and packing problems. The experimental results showed the competitiveness of the evolved local search heuristics.

2) GP-HH for scheduling problems: Dimopoulos and Zalzala [36] used GP to evolve reusable dispatching rules for the one-machine scheduling problem with a standard function set and a terminal set of scheduling statistics (processing time, release time, due date, number of jobs, etc.). The evolved dispatching rules were found to be better than traditional rules and these rules can be reused for a range of unseen problem instances. The best evolved rules were also examined and the analysis showed that these rules are interpretable. Jakobovic et al. [37] employed the same method for developing dispatching rules for the parallel machine scheduling problem in both static and dynamic environments. However, the evolved rules obtained from these studies have not considered the effects of different representations on the performance of the GP system and the machine and system attributes were not included in the evolved rules. Also, trying to learn new dispatching rules for the single machine environment, Geiger et al. [38] presented a learning system that combines GP with a simulation model of an industrial facility. The proposed GP system is used to create the priority rule for a single machine in static and dynamic environments. The terminal set of GP includes system attributes, job attributes, and machine attributes, and the function set consists of basic operators. The paper also proposed a method to learn dispatching rules for multiple machine problems in which GP will evolve multiple trees simultaneously with modified crossover and mutation operators. Comparison with the optimal rule in a simple two machine environment showed that the evolved rules are quite competitive. However, the use of an independent dispatching rule for each machine may rapidly increase the complexity of the scheduling systems and make it difficult to generate generalised rules for large manufacturing systems. Geiger and Uzsoy [39] also applied this system for learning dispatching rules for batch processor scheduling, with very promising results. For a stochastic single machine scheduling problem, Yin et al. [40] proposed a GP system employing bi-tree structured representation scheme to deal with machine breakdowns. The empirical results under different stochastic environments showed that GP can evolve high quality predictive scheduling heuristics.

Miyashita [8] developed an automatic method using GP to design customised dispatching rules for a job shop environment and viewed JSP as a model of a multi-agent problem where each agent represents a resource (machine or work station). Three multi-agent models are proposed: (1) a homogeneous model where all resources share the same dispatching rule, (2) a distinct agent model where each resource employs its own evolved rule, and (3) a mixed agent model where two rules can be selected to prioritise jobs depending on whether the resource is a bottleneck or not. Although the multi-agent models perform better, the use of these models depends on the some prior-knowledge of the job shop environment, which can be changed in dynamic situations. A similar system was also proposed by Atlan et al. [41] but the focus of their paper is on finding the solution for a particular problem instance. Jakobovic and Budin [6] applied GP to evolve dispatching rules for both single machine and job shop environments. The results for the single machine environment are shown to be better than existing rules. For the job shop environment, a meta-algorithm is defined to show how the evolved rules are used to construct a schedule. This study also proposed an interesting way to provide some adaptive behaviours for the evolved rules. They proposed a GP-3 system that evolves three components, a discriminant function and two dispatching rules. The discriminant function aims at identifying whether the considered machine to be scheduled is a bottleneck or not. This function serves as the classifier in binary classification problems. Based on the classification decision obtained from the discriminant function, one of two dispatching rules will be selected to sequence jobs in the queue of that machine. Even though the purpose of the discriminant function in this case is to identify the bottleneck machine, there is no guarantee that the classification can help indicate the bottleneck machine or just some useful attributes of the shop or machines. The results show that the GP-3 system performed better than traditional GP with a single tree. Unfortunately, no demonstrations or analyses of the evolved rules are given. Tay and Ho [5] proposed a GP system to evolve dispatching rules for a job shop environment. The proposed GP program can be considered as a priority function which is used to calculate the priority of operations in the queue of a machine. The set of instances was randomly generated and it was shown that the evolved dispatching rules can outperform other simple dispatching rules. However, they did not consider the use of machine attributes in the priority function. Hildebrandt et al. [7] re-examined the GP system proposed in [5] in different dynamic job shop scenarios and showed that rules evolved by Tay and Ho [5] are only slightly better than the earliest release date (ERD) rule and quite far away from the performance of the SPT rule. They explained that the poor performance of these rules is caused by the use of the *linear* combination of different objectives and the fact that the randomly generated instances cannot effectively represent the situations that happened in a long term simulation. For that reason, Hildebrandt et al. [7] evolved dispatching rules by training them on four simulation scenarios (10 machines with two utilisation levels and two job types) and only aimed at minimising the mean flow time. Some aspects of the simulation models were also discussed in their study. The experimental results indicated that the evolved rules were quite complicated but effective when compared to other existing rules. Moreover, these evolved rules are also robust when tested with another environment (50 machines and different processing time distributions). Pickardt et al. [42] applied this system to evolve dispatching rules for semiconductor manufacturing to minimize weighted tardiness. Vazquez-Rodriguez and Ochoa [43] proposed a GP system to learn variants of the Nawaz, En-score and Ham (NEH) procedure for flow shop scheduling. The results showed that the evolved heuristics can outperform the original and stochastic variants of NEH.

Other methods have also been proposed for generating new dispatching rules. Li and Olafsson [11] applied data mining techniques on production data to generate dispatching rules. For further improvement, the authors used data engineering to create more useful attributes besides ones recorded as part of the raw production data. The results show that the decision rules discovered can accurately replicate the dispatching list obtained by specific rules. However, since the rules are learned from the historical records of the production system, it is difficult to generate new dispatching rules and the performance of the evolved rules may only be as good as the actual rules. Most recently, Ingimundardottir and Runarsson [12] proposed a supervised learning approach which tries to discover new dispatching rules using the characteristics of optimal solutions. The learned linear priority dispatching rules showed better results than simple rules. One of the drawbacks is that the proposed supervised learning approach tries to learn from the optimal solutions which are not available in many cases.

Dispatching rules have been a very practical tool for scheduling in real world situations. However, manual design of new dispatching rules is still a very time consuming process. Several machine learning methods have been proposed to ease this task. GP is one of the more popular methods because of its flexibility which helps it easily cope with different problem environments. Since the design process is automated by GP, not only priority functions but also complicated issues such as incorporating machine and system attributes can be considered in order to improve the effectiveness of evolved rules. However, the existing GP systems mainly focus only on evolving priority functions. Also, bottleneck was the only system attribute used in the previous studies [6], [8]. In this study, we propose two new representations for GP which evolve rules with the ability to incorporate different machine and system attributes to make better sequencing decisions. A comparison between the proposed rules and the arithmetic representation used in previous studies is also presented. This study also examines other important issues, such as the influence of fitness functions and analysis of evolved rules, to gain more understanding of how GP can evolve effective dispatching rules.

III. Methodology

This section discusses a new GP system that is able to learn new dispatching rules which can adaptively sequence operations in the queue based on the status of the shop and the machines to be scheduled. In this section, the representations of a dispatching rule are presented first, followed by the fitness function used to measure the performance of an evolved dispatching rule.

A. Representations

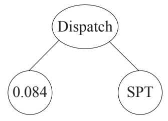
Three representations of dispatching rules are considered in this study. The first representation (R_1) provides a way to incorporate machine attributes into the GP program along with simple dispatching rules and the hybrid scheduling strategy (between non-delay and active scheduling). The second representation (R_2) is the traditional arithmetic representation like that employed in [5]; the purpose of this representation is to generate composite dispatching rules. The last representation (R_3) is a combination of R_1 and R_2, in which different composite dispatching rules exist and are logically applied to JSP based on the machine and system attributes.

1) Decision-tree like representation (R_1): The key idea of this representation is to provide the adaptive dispatching rules the ability to apply different simple dispatching rules based on machine attributes. In this case, the adaptive dispatching rules are represented in a decision-tree form. To make the rules more readable and explainable, the proposed grammar in Figure 2 is used when building the GP programs and performing the genetic operators (e.g. dispatching nodes must contain two arguments, which are a value of the non-delay factor and a single dispatching rule). Two example rules based on this grammar are shown in Figure 3. In Figure 3(a), the rule is similar to that of SPT which is applied with the non-delay factor $\alpha = 0.084$. The rule in Figure 3(b) is a bit more

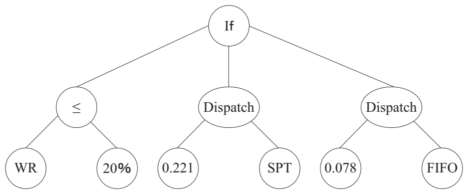
```
Start ::= <action>
<action> ::= <if> | <dispatch>
<if> ::= if <attributetype> <op> <threshold>
then <action> else <action>
<op> ::= ≤ | >
<attributetype> ::= WR | MP | DJ | CMI | CWR | BWR
<threshold> ::= 10%|20%|30%|40%|50%|60%|70%|80%|90%|100%
<dispatch> ::= assign <nondelayfactor> assign <rule>
<nondelayfactor> ::= uniform[0,1]
<rule> ::= FIFO | SPT | LPT | LSO | LRM | MWKR | SWKR |
MOPR | EDD | MS | WSPT
```

Fig. 2. Grammar for the proposed GP system with R_1.

sophisticated. The rule firstly checks the workload ratio WR (the ratio of the total processing times of jobs in the queue to the total processing times of all jobs that have to be processed at the machine) of the considered machine m^*; if the workload ratio is less than or equal to 20%, dispatching rule SPT is applied with $\alpha = 0.221$; otherwise, dispatching rule FIFO is applied with $\alpha = 0.078$. This ADR can be considered as a variant of FIFO/SPT, in which the workload of the machine is used as the switch instead of the waiting times of jobs in the queue. Different from other applications [44], [45] where a single non-delay factor is evolved, the proposed GP system using this representation allows different values of non-delay factors to be employed based on the status of the shop.



(a) SPT with $\alpha = 0.084$



(b) If ($WR \leq 20\%$) then use SPT with $\alpha = 0.221$ else use FIFO with $\alpha = 0.078$

Fig. 3. Example program trees based on representation R_1.

In this study, we will consider six attributes which indicate the status of machines in the shop. Let Λ be the set of operations that are planned to visit the considered machine m^*, and K and I are the sets of all operations that have and have not yet been processed by m^*, respectively ($\Lambda = K \cup I$). In the shop, we call a machine critical if it has the greatest total remaining processing time $\sum_{\sigma \in I} p(\sigma)$ and a machine is called bottleneck if it has the largest workload $\sum_{\sigma \in \Omega'} p(\sigma)$ in Ω'. The following definitions of the machine and system attributes are used in this study:

- *Workload ratio,* $WR = \frac{\sum_{\sigma \in \Omega'} p(\sigma)}{\sum_{\sigma \in I} p(\sigma)}$: indicates the workload in Ω' compared to the total remaining workload that m^* has to process (including the operations in the queue and operations that have not yet visited m^*).
- *Machine progress,* $MP = \frac{\sum_{\sigma \in K} p(\sigma)}{\sum_{\sigma \in \Lambda} p(\sigma)}$: indicates the progress of m^*, calculated as the ratio of the total processing time that m^* has processed to the total processing time of all operations in Ω' that have to visit m^*.
- *Deviation of jobs,* $DJ = \frac{\min_{\sigma \in \Omega'}\{p(\sigma)\}}{\max_{\sigma \in \Omega'}\{p(\sigma)\}}$: is a simple ratio of minimum processing time to the maximum processing time of operations in Ω'.
- *Critical machine idleness,* CMI: is the workload ratio WR of the critical machine.
- *Critical workload ratio,* $CWR = \frac{\sum_{\sigma \in \Omega^c} p(\sigma)}{\sum_{\sigma \in \Omega'} p(\sigma}$: is the ratio of the workload of operations in Ω^c to the workload in Ω' where $\Omega^c \subset \Omega'$ is the set of operations belonging to the jobs that have operations that still need to be processed at the critical machine after being processed at m^*.
- *Bottleneck workload ratio,* $BWR = \frac{\sum_{\sigma \in \Omega^b} p(\sigma)}{\sum_{\sigma \in \Omega'} p(\sigma)}$: is the ratio of the workload of operations in Ω^b to the workload in Ω' where $\Omega^b \subset \Omega'$ is the set of operations belonging to the jobs that have operations that still need to be processed at the bottleneck machine after being processed at m^*.

While the first three attributes provide the local status at m^*, the last three attributes indicate the status of the shop with a special focus on the critical and bottleneck machines. The machine and system attributes here appear in the scheduling literature in different forms. The key difference between our attributes and the attributes used in other studies is that our attributes have been scaled from 0 to 1. The scaled (normalized) attribute values aim to enhance the generality of the evolved rules and also make the evolved rules easier to understand. Jakobovic and Budin [6] employed attributes similar to ours without normalization (e.g. remaining work at the machine is similar to workload ratio in our study). The definition of attributes for bottleneck and critical machines are adapted from the bottleneck concept [46] for static problems and these attributes are used to adjust the rules to react appropriately to the changes of the shop.

For representation R_1, eleven simple dispatching rules are considered as the candidate rules in the ADR. The aim of these rules is to determine which operation σ in Ω' will be processed next. Let $n(\sigma)$ be the job which operation σ belongs to, $j = n(\sigma)$ and $o_{j,h} = \sigma$. The following are brief descriptions of the candidate dispatching rules. Detailed discussion of these rules can be found in [1] and [47].

- *FIFO*: operations are sequenced *first-in-first-out*.
- *SPT*: select the operation with the *shortest processing time* $p(\sigma)$.
- *LPT*: select the operation with the *longest processing time* $p(\sigma)$.
- *LSO*: select the operation belonging to the job that has the *longest subsequent operation* $p(next(\sigma))$
- *LRM*: select the operation belonging to the job that has the *longest remaining processing time* (excluding the operation under consideration) $\sum_{l=h+1}^{N_j} p(o_{j,l})$.
- *MWKR*: select the operation belonging to the job that has the *most work remaining* $\sum_{l=h}^{N_j} p(o_{j,l})$.
- *SWKR*: select the operation belonging to the job that has

the *smallest work remaining* $\sum_{l=h}^{N_j} p(o_{j,l})$.

- *MOPR*: select the operation belonging to the job that has the *largest number of operations remaining* $N_j - h + 1$.
- *EDD*: select the operation belonging to the job that has the *earliest due date* d_j.
- *MS*: select the operation belonging to the job that has the *minimum slack* $MS_j = d_j - \sum_{l=h}^{N_j} p(o_{j,l}) - t$. Value $t = R_{m^*}$ is the time at which the sequencing decision needs to be made.
- *WSPT*: select the operation that has the maximum *weighted shortest processing time* $w_j/p(\sigma)$.

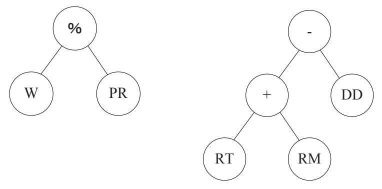
The non-delay factor is treated as Ephemeral Random Constants (ERC) [48]. The values of the non-delay factor will initially be a random number from 0 to 1. Meanwhile, attribute type, attribute threshold and dispatching rule terminals are randomly chosen from their candidate values as described in the previous section with equal probabilities.

2) Arithmetic representation (R_2): For this representation, the focus is to formulate composite dispatching rules that include different pieces of information from jobs and machines. Basically, the GP programs are priority functions that can be used to calculate the priorities for operations in Ω' and the operation with the highest priority will be scheduled to be processed next on the machine m^*. Different from R_1 rules that become more effective by logical choices of single dispatching rules, R_2 rules create their sophisticated behaviour by arithmetically combining various terms into the priority functions. The advantage of this representation is that more information can be directly considered to determine the priorities of operations when sequencing decisions need to be made.

TABLE I
TERMINAL SET FOR R_2 ($j = n(\sigma)$ AND $o_{j,h} = \sigma$)

Notation	Description	Value
RJ	operation ready time	$r(\sigma)$
RO	number of remaining operations of job j	$N_j - h + 1$
RT	work remaining of job j	$\sum_{l=h}^{N_j} p(o_{j,l})$
PR	operation processing time	$p(\sigma)$
W	weight	w_j
DD	due date	d_j
RM	machine ready time	R_{m^*}
#	constant	Uniform[0,1]

In our GP system, the four basic arithmetic operators ($+, -, *$, protected division $\%$) are used in the function set. Since we evolve a function to prioritise operations, it seems useful to include a single-argument function $(-1*)$ in the function set to provide a more convenient way to create priority functions. The terminal set contains popular terms that are used in developing existing dispatching rules. The descriptions of the terminals used for calculating the priority of operation σ are shown in Table I. Figure 4 shows two simple examples when WSPT and MS are represented by R_2 rules. The non-delay scheduling strategy will be used along with this representation like common applications of composite dispatching rules.



(a) $W\%PR$ (b) $(RT + RM) - DD$

Fig. 4. Example program trees based on representation R_2.

3) Mixed representation (R_3): This representation tries to combine the advantages of R_1 and R_2 to create sophisticated adaptive dispatching rules. Within R_3, the incorporation of both system/machine status and composite dispatching rules are considered. The representation R_3 inherits the grammar in R_1 and the composite dispatching rules will be used to calculate the priorities of operations for sequencing decisions besides the use of simple dispatching rules. An example of an R_3 rule is shown in Figure 5.

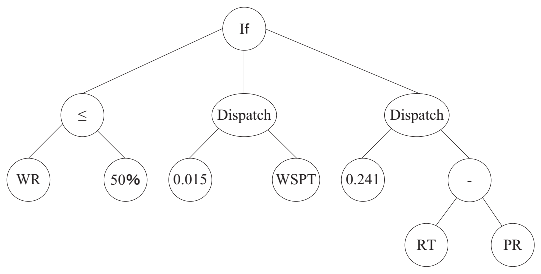


Fig. 5. An example program tree based on representation R_3.

B. Fitness Evaluation

The focus of this study is to learn effective rules for JSP to minimise the makespan or total weighted tardiness. In order to estimate the effectiveness of an evolved dispatching rule, it will be applied within the construction procedure in Figure 1 to solve a set of instances in the training set and the resulting objective values from all instances are recorded. Since the objective values obtained by a dispatching rule Δ for each instance are very different, we will measure the quality of an obtained schedule by the relative deviation of its objective value from its corresponding reference objective value as shown in equation (1).

$$dev(\Delta, I_n) = \frac{Obj(\Delta, I_n) - Ref(I_n)}{Ref(I_n)} \quad (1)$$

In this equation, $Obj(\Delta, I_n)$ is the objective value obtained when applying rule Δ to instance I_n, and $Ref(I_n)$ is the reference objective value for instance I_n. The fitness of rule Δ on the training set is calculated by equation (2). The fitness function $dev_{average}(\Delta)$ will measure the average performance of Δ across T instances in the data set.

$$dev_{average}(\Delta) = \frac{\sum_{I_n \in \{I_1,\ldots,I_T\}} dev(\Delta, I_n)}{T} \quad (2)$$

The objective of the GP system is to minimise this fitness function. In the case of $Jm||C_{max}$, the reference objective value is the lower bound obtained by other approaches (refer to [49] for a list of lower bound values obtained for popular benchmark instances). Since lower bounds are used in this case, the fitness values for the GP programs are always non-negative. If the fitness value is close to zero, it indicates that the evolved rules can provide near optimal solutions. For $Jm||\sum w_jT_j$, because the lower bound values are not available for all instances, we will use objective values obtained by EDD using non-delay scheduling as the reference objective values for all instances in the data set since it is a widely used dispatching rule for due date related problems. Because EDD is just a simple rule, it can be dominated by more sophisticated rules. For that reason, the fitness value of the GP programs for $Jm||\sum w_jT_j$ can be negative, which means that the evolved rules perform better than EDD with a given fitness function.

C. Proposed GP algorithm

Figure 6 shows the GP algorithm used in this study to evolve dispatching rules for JSP. The GP system first sets up the training set D and randomly initialises the population. At a generation, each dispatching rule (or individual) Δ_i will be applied to solve all instances in the training set D to find the relative deviation $dev(\Delta_i, I_n)$ for each instance. Then, the fitness value of each rule is calculated by using $dev_{average}(\Delta_i)$. If the evaluated rule is better (has smaller fitness value) than the best rule Δ^*, it will be assigned to the best rule Δ^* and the best fitness value $fitness(\Delta^*)$ is also updated. After all individuals in the population are evaluated, the GP system will apply genetic operators such as reproduction (elitism), crossover and mutation to the programs in the current population to generate new individuals for the next generation. More details of the genetic operators used in this study will be provided in the next section. When the maximum number of generations is reached, the GP algorithm will stop and return the best found rule Δ^*, which will be applied to the test set to evaluate the performance of the GP system.

```
Inputs: training instances $D \leftarrow \{I_1, I_2, \ldots, I_T\}$
Output: the best evolved rule $\Delta^*$
1: randomly initialise the population $P \leftarrow \{\Delta_1, \ldots, \Delta_S\}$
2: set $\Delta^* \leftarrow null$ and $fitness(\Delta^*) = +\infty$
3: $generation \leftarrow 0$
4: while $generation \leq maxGeneration$ do
5:     for all $\Delta_i \in P$ do
6:         for all $I_n \in D$ do
7:             $dev(\Delta_i, I_n) \leftarrow$ solve $I_n$ with $\Delta_i$
8:         end for
9:         evaluate $fitness(\Delta_i) \leftarrow dev_{average}(\Delta_i)$
10:        if $fitness(\Delta_i) < fitness(\Delta^*)$ then
11:            $\Delta^* \leftarrow \Delta_i$
12:            $fitness(\Delta^*) \leftarrow fitness(\Delta_i)$
13:        end if
14:    end for
15:    $P \leftarrow$ apply reproduction, crossover, mutation to $P$
16:    $generation \leftarrow generation + 1$
17: end while
18: return $\Delta^*$
```

Fig. 6. GP algorithm to evolve dispatching rules for JSP.

TABLE II
PARAMETERS OF THE PROPOSED GP SYSTEM

Population Size	1024
Crossover rate	95%,90%,85%
Mutation rate	0%,5%,10%
Reproduction rate	5%
Generations	50
Max-depth	6
Function set (R_1)	*If, Dispatch,* $\leq,>$
Terminal set (R_1)	*attribute type, attribute threshold, non-delay factor, dispatching rule*
Function set (R_2)	$+,-,*,\%,(-1*)$
Terminal set (R_2)	*as shown in Table I*
Function set (R_3)	*Function set (R_1) and Function set (R_2)*
Terminal set (R_3)	*Terminal set (R_1) and Terminal set (R_2)*
Fitness	$dev_{average}(\Delta)$

IV. Design of Experiments

This section discusses the configuration of the GP system and the data sets used for training and testing.

A. Parameter setting

The GP system for learning ADRs is developed based on the ECJ20 library [50] (a java-based evolutionary computation research system). The parameter settings of the GP system used in the rest of this paper are shown in Table II. The population size of 1024 is used to ensure that there is enough diversity in the population. The initial GP population is created using the ramped-half-and-half method [48]. Since the rules are created based on the grammar in Figure 2, we use strongly typed GP to ensure that the GP nodes will provide proper return types as determined by the grammar. In this case, the crossover and mutation operators of GP are only allowed if they do not violate the grammar. For crossover, the GP system uses the subtree crossover [48], which creates new individuals for the next generation by randomly recombining subtrees from two selected parents. Meanwhile, mutation is performed by subtree mutation [48], which randomly selects a node of a chosen individual and replaces the subtree rooted by that node by a newly randomly-generated subtree. The combinations of three levels of crossover rates and mutation rates will be tested in our experiments to examine the influence of these genetic operators on the performance of GP. When generating random initial programs or applying crossover/mutation, the maximum depth of six is used to restrict the program from becoming too large. Greater maximum depths can also be used here to extend the search space of GP; however, we choose this maximum depth to reduce the computational times of the GP system and make the evolved rules easier to analyse. Tournament selection with the tournament size of seven is used to select individuals for genetic operations.

B. Data sets

There are many data sets in the JSP literature which are generated by different scheduling researchers [51], [52], [53], [54] to measure the performance of different heuristic and

TABLE III
JSP DATA SETS

Data set	Notation	# of instances	Problem size ($\mathcal{N} \times \mathcal{M}$)	Reference
LA	la01-la40	40	from 10×5 to 15×15	Lawrence [51]
ORB	orb01-orb10	10	10×10	Applegate and Cook [52]
TA	ta01-ta80	80	from 15×15 to 100×20	Taillard [53]
DMU	dmu01-dmu80	80	from 20×15 to 50×20	Demirkol et al. [54]

optimisation methods. The instances in these data sets are still very useful because they include a wide range of instances with different levels of difficulty. Moreover, lower bounds for these instances are available and can be used to calculate the fitness of the evolved dispatching rules as described in Section III-B. Descriptions of the data sets used for the experiments are shown in Table III. In this study, we combine these data sets and distribute them into the training set and test set used by the proposed GP system. The training set and test set are created to include halves of the instances of each individual data set in Table III. In particular, the training set will contain {la01, la03, …, la39}, {orb01, …, orb09},{ ta01, …, ta79}, {dmu01, …, dmu79}. The other (even index) instances will be included in the test set. This allows a fair distribution of problems with different instance sizes into both training set and test set. There are 105 instances in each of the training set and test set. For the case of $Jm||\sum w_j T_j$, the due dates for jobs in each instance will be generated (following Baker [55]) by a due date assignment rule:

$$d_j = r_j + h \times \sum_{l=1}^{N_j} p(o_{j,l}) \tag{3}$$

The parameter h is used to indicate the tightness of due dates. We choose $h = 1.3$ for all instances in the training set and test set because it is the common value used in previous research [19], [20]. For the weights of jobs, we employ the $4:2:1$ rule which has been used in [19] and [20]. This rule is inspired by Pinedo and Singer [56] when their research showed that 20% of the customers are very important, 60% are of average importance and the remaining 20% are of less importance. For that reason, in $Jm||\sum w_j T_j$, the weights of 4 are assigned to the first 20% of jobs, the next 60% are assigned a weight of 2 and the last 20% of jobs are assigned a weight of 1.

V. RESULTS

The proposed GP systems with different settings are now applied to evolve new dispatching rules. This section shows the results obtained from the GP system with three representations, three levels of crossover/mutation rates, and two objective values of JSP. In total, we need $3 \times 3 \times 2 = 18$ experiments. For each experiment, 30 independent runs are performed with different random seeds. Table IV and Table VI show the means and standard deviations of fitness values (of the best evolved rules from each run) obtained from all experiments on the training set and test set. The triple $\langle c, m, r\rangle$ indicates the GP parameters used in a specific experiment. For example, $\langle 85, 10, 5\rangle$ represents the experiment where the crossover rate is 85%, the mutation rate is 10% and the reproduction rate is 5%. All statistical tests discussed in this section are the standard z-tests and they are considered significant if the obtained p-value is less than 0.05 [1].

A. Makespan

As shown in Table IV, the evolved rules based on R_1 show a performance close to those obtained by R_2 and R_3 on the training set. It is also noted that the R_1 rules evolved with higher mutation rates are significantly better than those evolved without lower mutation (all p-values < 0.0174) on the training set. Since R_1 rules contain many possible terminals higher mutation rates seem quite useful to improve the performance of the GP system with the R_1 representation. However, the performance of R_1 rules are quite poor on the test set. This indicates the overfitting issue of R_1 rules when learning with the training set. The reason for this problem comes from the fact that the candidate rules used in R_1 are too simple, and therefore the rules have to depend strongly on the machine and system status to provide better sequencing decisions on the training instances. However, the overuse of the machine and system attributes make R_1 rules less effective when dealing with unseen instances in the test set.

Evolved R_2 rules show a more consistent performance on both the training set and test set. Different from R_1, the statistical tests indicate that the choice of GP parameters does not have significant influence (all p-values > 0.1434) when R_2 is used as the representation of the dispatching rule. These results indicate that mutation is not really useful in this case and the crossover operator is sufficient for the GP system to evolve good individuals. Since R_2 provides only one way to sequence operations, the effectiveness is obtained by good combinations of different terms. Hence, they are also less affected when working with unseen instances like R_1 rules.

Taking the advantages of R_1 and R_2, the evolved R_3 rules show very promising performance. Different from R_1, the incorporation of the machine and system attributes into the R_3 rules are supported by better composite dispatching rules, and therefore they do not need to depend heavily on the use of machine and system attributes to be effective. The performance on the test set shows that the evolved R_3 rules also have good generalisation qualities like that of the R_2 rules. Mutation does not affect the GP system with R_3 as strongly as the GP system with R_1. The significant difference is only observed between the experiment with no mutation and the experiment with the mutation rate of 10% (p-value < 0.0031). Although there is no obvious difference in the performance of evolved rules from different configurations on the training set, R_2 rules and R_3 rules (evolved with non-zero mutation rate) are significantly

[1]Wilcoxon tests are also performed and the results from these tests are consistent with those obtained z-tests.

TABLE IV
PERFORMANCE OF EVOLVED RULES FOR $Jm||C_{max}$ ON TRAINING SET AND TEST SET WITH DIFFERENT SETTINGS
$(mean \pm standard\ deviation)$

Setting		R_1		R_2		R_3	
		Training	Testing	Training	Testing	Training	Testing
$dev_{average}(\Delta)$	$\langle 95, 0, 5\rangle$	0.188 ± 0.008	0.197 ± 0.007	0.181 ± 0.003	0.187 ± 0.004	0.183 ± 0.005	0.186 ± 0.005
	$\langle 90, 5, 5\rangle$	0.181 ± 0.003	0.192 ± 0.005	0.181 ± 0.003	0.188 ± 0.005	0.181 ± 0.005	0.184 ± 0.006
	$\langle 85, 10, 5\rangle$	0.180 ± 0.003	0.191 ± 0.006	0.180 ± 0.003	0.188 ± 0.004	0.179 ± 0.005	0.184 ± 0.004

TABLE V
RELATIVE DEVIATIONS OBTAINED BY EVOLVED RULES AND OTHER RULES FOR $Jm||C_{max}$

Rules		Training Set			Test Set		
		Min	Mean	Max	Min	Mean	Max
FIFO	Active	0.012	0.325	0.691	0.000	0.325	0.654
	Non-Delay	0.012	0.325	0.691	0.000	0.325	0.654
SPT	Active	0.322	0.694	1.252	0.316	0.711	1.091
	Non-Delay	0.029	0.292	0.576	0.092	0.312	0.664
LRM	Active	0.020	0.321	0.745	0.000	0.319	0.723
	Non-Delay	0.000	0.224	0.556	0.000	0.225	0.529
MWKR	Active	0.047	0.323	0.736	0.000	0.328	0.713
	Non-Delay	0.000	0.253	0.590	0.000	0.254	0.584
Evolved	$\Delta^{c1}_{R_1}$	0.000	0.173	0.448	0.000	0.192	0.525
	$\Delta^{c2}_{R_1}$	0.000	0.174	0.428	0.000	0.183	0.479
	$\Delta^{c1}_{R_2}$	0.000	0.174	0.479	0.000	0.190	0.442
	$\Delta^{c2}_{R_2}$	0.000	0.175	0.457	0.000	0.191	0.446
	$\Delta^{c1}_{R_3}$	0.000	0.171	0.490	0.000	0.185	0.572
	$\Delta^{c2}_{R_3}$	0.000	0.172	0.447	0.000	0.177	0.428

better than R_1 rules on the test set (all p-values < 0.0361 over $(3+2) \times 3 = 15$ statistical tests). Also, the evolved R_3 rules from the GP system with non-zero mutation rates are significantly better than R_1 and R_2 rules on the test set (all p-values < 0.014 over $2 \times (3+3) = 12$ statistical tests).

Table V shows the performance of the evolved rules with four selected existing dispatching rules for $Jm||C_{max}$. The values in this table are the statistics of relative deviations using equations (1) obtained when applying the evolved rules to the instances in the training set and test set. The values of $Mean$ can be calculated by using equation (2) to measure the average while Min and Max values show the best-case and worst-case performance of a dispatching rule on a given set of instances. From each representation, two evolved rules that show the best fitness in the training stage are used here for comparison. Rule $\Delta^{yv}_{R_x}$ is the v^{th} best rule that was evolved by using the representation R_x to minimise objective function y for the JSP. For example, $\Delta^{c2}_{R_2}$ is the second best rule evolved with the representation R_2 and C_{max} as the objective function (t will be used to indicate the total weighted tardiness).

Each simple dispatching rule is used to generate both active and non-delay schedules for all instances in the training set and test set. The results show that LRM with non-delay scheduling strategy is better than other simple rules. The performance of the evolved rules are better than all of the simple dispatching rules on the training set and test set. However, not all evolved rules can produce consistent results when dealing with unseen instances. Rules $\Delta^{c2}_{R_1}$ and $\Delta^{c2}_{R_3}$ are the rules that provide the best performance on the test set even though they are not the best rules on the training set. Generally, it seems quite difficult to develop a rule that produces good average and worst-case performance. One explanation is that the rules evolved with $dev_{average}(\Delta)$ focus on the overall performance, therefore they may ignore some extreme cases. Evolved rule $\Delta^{c2}_{R_3}$ is one specific case when the best average performance and very good worst-case performance is achieved.

B. Total weighted tardiness

In Table VI, the experiments with R_3 as the representation can produce rules with better fitness than experiments with R_1 or R_2 as the representation. The statistical tests show that evolved R_3 rules from the GP system with non-zero mutation rates are significantly better than other evolved R_1 and R_2 rules on both training set and test set (all p-values < 0.0009 over $2 \times (3+3) \times 2 = 24$ statistical tests). These results again confirm the effectiveness of the R_3 representation. Also, the R_1 rules from the GP system with non-zero mutation rates are significantly better than R_2 rules in this case (all p-values $< 2 \times 10^{-16}$ over $2 \times 3 \times 2 = 12$ statistical tests). This suggests that the machine attributes and scheduling strategies are quite important when dealing with $Jm||\sum w_jT_j$.

The comparison of the evolved rules and other dispatching rules are shown in Table VII. It is easy to see that non due date related rules such as FIFO and LRM have very poor performance when dealing with $Jm||\sum w_jT_j$ even though LRM achieves good performance on $Jm||C_{max}$. MS and WSPT show better performance than FIFO and LRM because information about the due date and the weight of a job is considered in these rules. While MS is still much worse than EDD, WSPT shows good performance even though it still cannot totally beat EDD. Sophisticated due-date related rules W(CR+SPT), W(S/RPT+SPT), COVERT and ATC (see [47],

TABLE VI
PERFORMANCE OF EVOLVED RULES FOR $Jm||\sum w_jT_j$ ON TRAINING SET AND TEST SET WITH DIFFERENT SETTINGS
$(mean \pm standard\ deviation)$

Setting		R_1		R_2		R_3	
		Training	Testing	Training	Testing	Training	Testing
$dev_{average}(\Delta)$	$\langle 95,0,5\rangle$	-0.227 ± 0.004	-0.221 ± 0.005	-0.216 ± 0.003	-0.203 ± 0.005	-0.240 ± 0.015	-0.233 ± 0.015
	$\langle 90,5,5\rangle$	-0.235 ± 0.006	-0.223 ± 0.003	-0.216 ± 0.004	-0.205 ± 0.006	-0.245 ± 0.012	-0.233 ± 0.014
	$\langle 85,10,5\rangle$	-0.237 ± 0.006	-0.222 ± 0.003	-0.216 ± 0.004	-0.204 ± 0.006	-0.247 ± 0.013	-0.235 ± 0.013

negative values mean the evolved rules are better than EDD

TABLE VII
RELATIVE DEVIATIONS OBTAINED BY EVOLVED RULES AND OTHER RULES FOR $Jm||\sum w_jT_j$

Rules		Training Set			Test Set		
		Min	Mean	Max	Min	Mean	Max
FIFO	Active	−0.318	0.455	1.600	−0.159	0.442	1.196
	Non-Delay	−0.318	0.455	1.600	−0.159	0.442	1.196
LRM	Active	0.193	0.832	2.129	−0.125	0.836	1.813
	Non-Delay	−0.189	0.507	1.571	−0.236	0.494	1.425
MS	Active	0.106	0.601	1.519	0.035	0.607	1.321
	Non-Delay	−0.040	0.387	1.205	−0.133	0.363	0.822
WSPT	Active	−0.133	0.148	0.874	−0.154	0.160	0.946
	Non-Delay	−0.394	−0.169	0.168	−0.459	−0.161	0.253
W(CR+SPT)	Active	−0.133	0.140	0.670	−0.234	0.147	0.858
	Non-Delay	−0.398	−0.173	0.177	−0.459	−0.165	0.435
W(S/RPT+SPT)	Active	−0.110	0.149	0.867	−0.154	0.161	0.853
	Non-Delay	−0.394	−0.168	0.168	−0.459	−0.161	0.253
COVERT	Active	−0.171	0.146	0.722	−0.235	0.147	0.853
	Non-Delay	−0.394	−0.173	0.177	−0.459	−0.160	0.253
ATC	Active	−0.258	0.145	0.757	−0.260	0.140	0.795
	Non-Delay	−0.394	−0.168	0.168	−0.459	−0.163	0.253
Evolved	$\Delta^{t1}_{R_1}$	−0.586	−0.247	0.020	−0.560	−0.220	0.150
	$\Delta^{t2}_{R_1}$	−0.531	−0.244	0.003	−0.507	−0.224	0.018
	$\Delta^{t1}_{R_2}$	−0.467	−0.223	−0.003	−0.517	−0.199	0.326
	$\Delta^{t2}_{R_2}$	−0.529	−0.223	0.151	−0.523	−0.206	0.116
	$\Delta^{t1}_{R_3}$	−0.506	−0.265	−0.033	−0.616	−0.246	−0.002
	$\Delta^{t2}_{R_3}$	−0.581	−0.265	−0.022	−0.555	−0.253	0.031

[57] for a detailed description of these rules) which have not been included in the list of candidate dispatching rules are also presented here. The expected waiting time in COVERT and ATC is calculated based on the standard method [47] in which the expected waiting time $W = b \times PR$, where PR is the operation processing time. For each method, two parameters needed to be specified which are k (look-ahead parameter) and b. 25 combinations of $b \in \{0.5, 1.0, 1.5, 2.0, 2.5\}$ and $k \in \{1.5, 2.0, 2.5, 3.0, 3.5\}$ are examined on the training set, and the combination that gives the best average performance is selected for comparison in the paper ($k = 2.5$, $b = 2.5$ for COVERT and $k = 1.5$, $b = 0.5$ for ATC). The performance of these rules are quite good since they are customised to deal with weighted tardiness problems. However, in the worst case, they still cannot provide better schedules than those obtained by EDD.

The best two evolved rules obtained by the GP system with different settings are now compared with these human-made dispatching rules. On both the training set and the test set, all evolved rules show better average performance than the existing rules. Similar to what has already been stated for $Jm||C_{max}$, R_1 and R_3 rules here are also much better than R_2 rules. However, it is still not easy to find a rule that totally dominates EDD. Among all the evolved rules, the evolved R_3 rules are the most promising ones. The best two evolved R_3 rules obtained very good average relative deviations for both training set and test set compared to those obtained by R_1 and R_2 rules. Rule $\Delta^{t1}_{R_3}$ is also the only evolved adaptive dispatching rule that totally dominates EDD on all training and testing instances. Meanwhile, $\Delta^{t2}_{R_3}$ produces a very good average performance, even better than $\Delta^{t1}_{R_3}$ on the test set and it is just slightly worse than $\Delta^{t1}_{R_3}$ in the worst case. The promising results of R_3 rules again confirm the need for integrating machine and system attributes with sophisticated dispatching rules to produce generalised and effective dispatching rules.

VI. ANALYSIS AND DISCUSSION

The previous section shows the results obtained by the proposed GP system and it is noted that the evolved rules can effectively provide good results for $Jm||C_{max}$ and $Jm||\sum w_jT_j$. In this section, we will further investigate the evolved rules to figure out how they can produce good performance. An analysis on the components of the evolved rules is given in order to gain more understanding of each factor within the proposed representations that may influence the ability of the GP system to generate better rules. A comparison between the evolved rules and some meta-heuristic approaches is then provided. Finally, the dispatching rules evolved by the proposed GP system are tested under a simulated dynamic JSP environment.

A. Insights from the evolved rules

In the previous experiment, there are 540 rules evolved with different GP parameters and representations for $Jm||C_{max}$ and $Jm||\sum w_jT_j$. The performance of the two best evolved rules for each representation and each objective function of JSP were shown in Table V and Table VII. As an example, we pick one evolved rule from each representation among these rules based on their overall performance on the training set and test set for further analysis (other evolved rules have a similar pattern).

1) Evolved rules for $Jm||C_{max}$: Here, $\Delta^{c1}_{R_1}$, $\Delta^{c1}_{R_2}$ and $\Delta^{c2}_{R_3}$ are chosen for analysis. The detailed representations of these rules are shown in Figure 7. For $\Delta^{c1}_{R_1}$, the first observation is that even though the rule looks complicated, it is just a combination of four simple dispatching rules (LRM, SPT, LPT and WSPT) and three machine attributes (CMI, CWR, DJ). Since WSPT is less relevant to $Jm||C_{max}$ and LPT is not very effective in this case, they only appear once in the entire adaptive dispatching rule. The root condition of $\Delta^{c1}_{R_1}$ checks the critical machine idleness and it is noted that LRM is the main dispatching rule when the idleness of the critical machine is greater than 10%. When CMI is small, the rule is more complicated and rules that favour small processing time operations like SPT and WSPT occur more in this case. This rule suggests that when the critical machine is idle, the considered machine should focus on completing operations with small processing times in order to feed more work to the critical machine and keep it busy; otherwise LRM should be used to prevent certain jobs from being completed so late and increase the makespan.

Different from $\Delta^{c1}_{R_1}$, $\Delta^{c1}_{R_2}$ (Figure 7(b)) is a pure mathematical function. In order to make it easy for analysis, we will simplify the whole function by eliminating terms, which appear to be less relevant. The simplification step is as follows:

$$\begin{aligned}\Delta^{c1a}_{R_2} &= \tfrac{(PR+RM)W}{PR} - \tfrac{(RJ+RM)PR}{RT} + RT\\ &\quad + \tfrac{DD}{(PR+RM)}RT\tfrac{W}{(PR^2)} + \tfrac{RT}{PR}(RJ+RM)\\ &\approx RT + \tfrac{RT}{PR}\tfrac{DD}{(PR+RM)}\tfrac{W}{(PR)} + \tfrac{RT}{PR}(RJ+RM)\\ &\approx RT + k \times \tfrac{RT}{PR}\end{aligned}$$

Since the first and second terms of $\Delta^{c1}_{R_2}$ do not make much sense in this case, we just drop them from the priority function[2]. The rest of the function can be grouped in two parts. The first part has RT, like rule MWKR, and the second part contains $\frac{RT}{PR}$, which is a combination of SPT and MWKR. When considering other terms in the second part as a constant k, we have the approximation of $\Delta^{c1}_{R_2}$ as a linear combination of MWKR and SPT/MWKR. This rule is actually not new. If we omit RT in the approximation function, the rest is known as shortest processing time by total work (SPTtwk) rule in the literature.

[2]The revised rules after dropping these irrelevant terms have been tested and it is noted that the performance is not significantly changed.

```
(IF (> CMI 10%)
  (IF (> CWR 20%)
  (IF (> CWR 80%) (DISPATCH 0.131 LRM)
   (IF (≤ DJ 30%) (DISPATCH 0.198 SPT) (DISPATCH 0.102 LRM)))
  (IF (> CWR 10%) (DISPATCH 0.102 LRM) (DISPATCH 0.131 LRM)))
 (IF (> CWR 10%)
  (IF (> CWR 80%) (DISPATCH 0.014 WSPT)
   (IF (≤ DJ 30%) (DISPATCH 0.198 SPT) (DISPATCH 0.131 LRM)))
  (IF (> CWR 80%) (DISPATCH 0.830 LPT)
   (IF (≤ DJ 20%) (DISPATCH 0.198 SPT) (DISPATCH 0.102 LRM)))))
```

(a) Evolved rule $\Delta^{c1}_{R_1}$

```
(+ (+ (-
          (*(+ PR RM) (% W PR))
          (%(+ RJ RM) (% RT PR)))
       RT)
  (-(*
          (% DD(+ PR RM))
          (*(% RT PR) (% W PR)))
  ((-1*) (*(% RT PR) (+ RJ RM)))))
```

(b) Evolved rule $\Delta^{c1}_{R_2}$

```
(IF (> CWR 90%)
 (DISPATCH 0.069 (% (* (+ RJ 0.594) (+ RT PR)) (+ W PR)))
  (IF (≤ MP 100%) (DISPATCH 0.128 (% (- RT PR) (+ W PR)))
  (IF (≤ CWR 100%) (IF (≤ MP 100%)
   (DISPATCH 0.166 WSPT) (DISPATCH 0.282 LPT))
   (IF(≤ MP 100%)(DISPATCH 0.044 LRM) (DISPATCH 0.736 FIFO)))))
```

(c) Evolved rule $\Delta^{c2}_{R_3}$

Fig. 7. Selected evolved rules for $Jm||C_{max}$.

Rule $\Delta^{c2}_{R_3}$ (see Figure 7(c)) is the most interesting rule in this case because both arithmetic rules and simple dispatching rules are employed. However, with the condition that MP has to be less than 100% at the second level, we can totally eliminate the subtrees that contain the simple dispatching rules. After some simplification steps, it is also noted that the arithmetic rules are also variants of SPTtwk. With the support of the machine attribute, the obtained arithmetic rules become much easier to analyse. The simplified version of $\Delta^{c2}_{R_3}$ is as follows:

$$\Delta^{c2a}_{R_3} = \begin{cases} \langle \frac{(RJ+0.594845)(RT+PR)}{W+PR}, \alpha = 0.069\rangle & \text{if } CWR > 90\% \\ \langle \frac{(RT-PR)}{(W+PR)}, \alpha = 0.128\rangle & \text{otherwise} \end{cases}$$

$$\approx \begin{cases} \langle \frac{(RJ+0.594845)(RT+PR)}{PR}, \alpha = 0.069\rangle & \text{if } CWR > 90\% \\ \langle \frac{(RT-PR)}{PR}, \alpha = 0.128\rangle & \text{otherwise} \end{cases}$$

The notation $\langle\cdot,\cdot\rangle$ indicates the dispatching rule and α value as in the middle and right subtrees of Figure 5. Although both priority functions of $\Delta^{c2}_{R_3}$ are variants of SPTtwk, the non-delay factor α is smaller for the case when $CWR > 90\%$. One explanation is that when Ω' contains many critical operations, it is reasonable to start the available operations right away instead of waiting for the operations that will be ready after the ready time of m^*.

2) Evolved rules for $Jm||\sum w_jT_j$: Here, $\Delta^{t1}_{R_1}$, $\Delta^{t2}_{R_2}$ and $\Delta^{t1}_{R_3}$ are selected to represent the evolved rules for $Jm||\sum w_jT_j$. The three full rules obtained by the GP are shown in Figure 8. For $\Delta^{t1}_{R_1}$, it is quite interesting that this rule can obtain such a good result (as shown in Table VII) without any due-date related components. The two main simple dispatching rules used in this case are WSPT and LPT. While WSPT can be considered as a suitable rule for $Jm||\sum w_jT_j$, it does not make sense to include LPT in this case. The result when we replace LPT by WSPT shows that the refined rule can still produce the results as good as $\Delta^{t1}_{R_1}$. For this reason, the contribution for the success of the rule comes from WSPT and other factors instead of the combination of different rules as we observed in the previous section. It is noted that most values of α in this case are about 0.4. Using these values for the WSPT alone show that WSPT with appropriate choice of α can produce the results much better than the case when the non-delay scheduling strategy is used. In this rule, the contribution of the machine and system attributes are not very important and they are mainly employed to improve the worst-case performance.

```
(IF (>DJ 80%)
  (IF (>BWR 90%)
  (IF (≤ DJ 90%) (DISPATCH 0.426 WSPT)
  (IF (≤ MP 10%) (DISPATCH 0.436 WSPT) (DISPATCH 0.364 LPT)))
   (DISPATCH 0.065 WSPT))
  (IF (> BWR 20%)
  (IF (≤ DJ 30%) (DISPATCH 0.436 WSPT)
   (IF (≤ DJ 30%) (DISPATCH 0.364 LPT) (DISPATCH 0.389 WSPT)))
  (IF (≤ DJ 30%) (DISPATCH 0.436 WSPT)
   (IF (> DJ 80%) (DISPATCH 0.364 LPT) (DISPATCH 0.181 WSPT)))))
```

(a) Evolved rule $\Delta^{t1}_{R_1}$

```
(- (+ (*
          (* PR (* 0.614577 PR))
          (- ((-1*) RM) (% RM W)))
      ((-1*) (* (* RT PR) (% RT W))))
   (+ (-
          (% (* RT PR) (- W 0.5214191))
          (* (% RM W) (* 0.614577 PR)))
      (* (% (* RT PR) (- W 0.5214191)) (+ (% RM W) (% RM W)))))
```

(b) Evolved rule $\Delta^{t2}_{R_2}$

```
(IF (≤ DJ 50%)
      (DISPATCH 0.331 ((-1*) (* DD (% PR W))))
      (DISPATCH 0.163 ((-1*) (* (% DD W) (% RT W)))))
```

(c) Evolved rule $\Delta^{t1}_{R_3}$

Fig. 8. Selected evolved rules for $Jm||\sum w_jT_j$.

For $\Delta^{t2}_{R_2}$, we perform the following steps to simplify the evolved rule:

$$\begin{aligned}\Delta^{t2a}_{R_2} &= 0.614PR^2(-RM - \tfrac{RM}{W}) - RT \times PR \times \tfrac{RT}{W} \\ &\quad -(RT\tfrac{PR}{W} - 0.521) - 0.614\tfrac{RM}{W}PR + 2RT\tfrac{PR}{(W-0.5214191)}\tfrac{RM}{W} \\ &\approx -0.614RM \times PR^2(1+\tfrac{1}{W}) - RT\tfrac{PR}{W}(RT + 2\tfrac{RM}{(W-0.5214191)}) \\ &\approx -k_1 \times PR^2(1+\tfrac{1}{W}) - k_2 \times RT\tfrac{PR}{W}\end{aligned}$$

The simplified rule is a linear combination of two sophisticated variants of WSPT where the first part includes PR^2 instead of PR and the second part includes RT. Repeating the experiment on this simplified rule shows that it can perform better than sophisticated rules like ATC and COVERT regarding the average relative deviation with appropriate choice of k_1 and k_2 (with $k_2 > k_1$, similar to the original rule).

It is very surprising that the best evolved rule for $Jm||\sum w_jT_j$ with the R_3 representation is also the smallest rule. Rule $\Delta^{t1}_{R_3}$ can be formally described as follows:

$$\Delta^{t1a}_{R_3} = \begin{cases} \langle -\frac{DD \times PR}{W}, \alpha = 0.331\rangle & \text{if } DJ \leq 50\% \\ \langle -\frac{DD \times RT}{W^2}, \alpha = 0.163\rangle & \text{otherwise} \end{cases}$$

The dispatching rule following the first priority function is a combination of EDD and WSPT and it is applied when the deviation of processing times of operations in Ω' is less than 50% (which means that the minimum processing time is less than half of the maximum processing time). When DJ is larger than 50% (which means that the gap between the minimum and maximum processing time is small), RT is used instead of PR and W^2 is used instead of W to increase the priority of jobs with small remaining processing times and high weights.

In general, even though the evolved rules can be very complicated sometimes, they always contain some good patterns which are very useful for creating good dispatching schedules. While R_1 rules can be easily explained with the support of the machine and system attributes, R_2 rules are quite sophisticated and often possess very interesting properties. It is surprising that the R_3 rules presented here are quite straightforward although they contain both machine attributes and composite dispatching rules. Rule $\Delta^{t1}_{R_3}$ could be quite tricky to represent as a pure mathematical function and it could be more difficult to discover its useful patterns; however, the use of machine and system attributes makes it much easier to identify and explain the rule.

B. Aggregate view of the evolved rules

Figure 9 shows the frequency that machine and system attributes have occurred in all of the evolved R_1 and R_3 rules. For $Jm||C_{max}$, critical machine related attributes are the most frequent ones while the bottleneck workload ratio has the lowest frequency. This result indicates that the information related to the critical machine is very important for the construction of a good dispatching rule to minimise the makespan. As shown in the previous section, suitable dispatching rules can be selected based on the idleness of the critical machine as well as the critical workload of m^*. In the case of $Jm||\sum w_jT_j$, the critical machine related attributes are still employed very often, and CWR along with DJ are the ones with the highest occurrence frequency. However, the distribution of attributes in $Jm||\sum w_jT_j$ are more uniform than that in $Jm||C_{max}$. This observation suggests that different attributes should be used to construct good dispatching rules for $Jm||\sum w_jT_j$.

For the priority functions (for composite dispatching rules) within R_2 and R_3 rules, Figure 10 shows that RT and PR

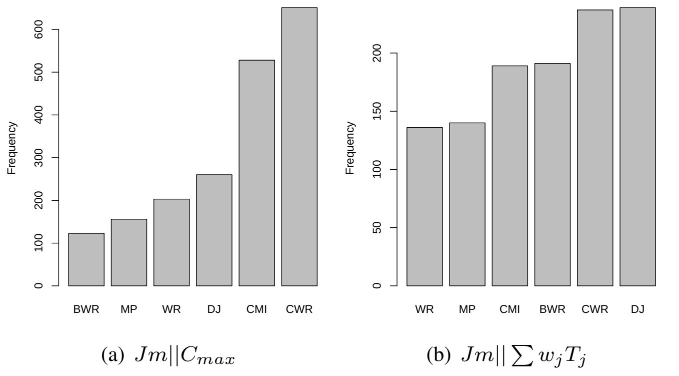


Fig. 9. Frequency of attributes in the evolved R_1 and R_3 rules

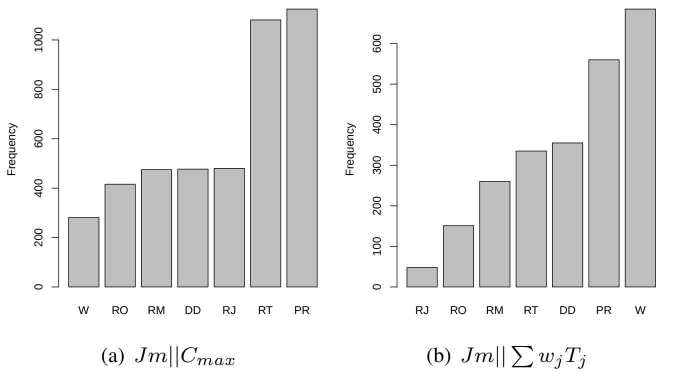


Fig. 10. Frequency of terminals in the evolved R_2 and R_3 priority function

are the most used terms for $Jm||C_{max}$. This explains the occurrence of SPTtwk variants in the best rules in the previous section. While W is the least used term for $Jm||C_{max}$, it is the most used term in $Jm||\sum w_jT_j$. This emphasises the importance of weights for determining the priority of operations in $Jm||\sum w_jT_j$. It is quite surprising that the second most frequent term in evolved rules for $Jm||\sum w_jT_j$ is PR instead of DD. Even though due-dates of jobs also depends on the processing times in aggregate form, the results here suggest that such local information as PR is still very useful for the dispatching tasks for due date related problems.

C. Comparison with meta-heuristic methods

This section compares the performance of the evolved dispatching rules to some improvement heuristics. Even though it seems unfair to the proposed GP method, since most meta-heuristic methods were especially developed for the static environment and iteratively explore the solution search space of each instance for better solutions, the comparison can provide an indicator about what kind of performance the evolved dispatching rules can achieve. In this section, we only compare the evolved rules with meta-heuristic methods for $Jm||\sum w_jT_j$ because it is still a very challenging problem and total weighted tardiness is one of the most important and practical objective functions in the scheduling literature.

Table VIII shows the total weighted tardiness obtained by the three best evolved rules for the three representations mentioned above and dispatching rules ATC and COVERT with fine-tuned α, and those obtained by the hybrid genetic algorithm [20] and the large step random walk [19] on selected problem instances from [20] and [19]. In the table, GA-(R&M/COVERT) and GA-(R&R) are the two versions of hybrid GA proposed by [20] (with population size of 100 and 2000 generations) while LSRW(t) is the large step optimisation method with running time equal to t seconds. The instances with bold names are ones that have been used in the training set. Instance la21' to la24' are sub-problems of instance la21-la24 where the last five jobs were omitted here. In Table VIII, rule $\Delta^{t1}_{R_3}$ produced the best performance in most instances compared to other evolved rules. Although $\Delta^{t1}_{R_3}$ is a very simple rule, it is still able to beat the hybrid GAs in several cases. Among 19 instances, $\Delta^{t1}_{R_3}$ outperforms GA-(R&M/COVERT) in five cases and GA-(R&R) in 12 cases, including ones that have not been included in the training set. Although the objective values obtained by the hybrid GAs are still far from the optimal solutions and those found by LSRW, they are still very promising because they can be directly applied to the real dynamic system without major redesign [20]. The hybrid GAs are also shown to dominate the GA approach [58]. The fact that the GP evolved rules in this work can beat the hybrid GAs in some cases and be competitive in other cases shows the effectiveness of these rules. Also, while the hybrid GAs need 200,000 evaluations and LSRW requires about more than 15 seconds to solve an instance, the GP evolved rules can solve the whole set of 210 instances in less than one second. This suggests new rules can produce better performance and still maintain the efficiency of a dispatching rule. This characteristic makes the GP approach particularly useful for real-time applications. Another advantage of the proposed GP method is that it is able to automatically incorporate information and constraints of different job shop environments into the rules; this it is a very hard task for conventional optimisation methods.

D. The performance of evolved rules in a dynamic JSP environment

This section examines the performance of the evolved rules in a dynamic environment. A simulation model of a simple job shop environment is built for the evaluation of the evolved rules. Table IX shows the parameters of the simulation model. With this configuration, the utilisation of each machine $\frac{\lambda}{\mu}$ is equal to the arrival rate λ (since $\mu = 1$). The due-date assignment rule is the same as that used in the previous section. Each simulation experiment consists of 30 replications and the *common random seeds* are used for each experiment in order to reduce the variance of the obtained results. The interval from the beginning of the simulation until the arrival of job 50,000 is considered as the warm-up time and all statistics are collected for the next 100,000 jobs. The results obtained from the experiments are shown in Figure 11. Since the JSP instances in the data set do not consider the release times of jobs (all release times of jobs are assumed to be zero in all instances) which are very important for the dynamic $Jm||C_{max}$, the evolved rules for $Jm||C_{max}$ are not suitable for these experiments. So, only the evolved rules for $Jm||\sum w_jT_j$ presented in Figure 8 are investigated here. For

TABLE VIII
COMPARISON OF EVOLVED RULES WITH META-HEURISTIC METHODS FOR $Jm||\sum w_jT_j$ ($h = 1.3$)

instance	optimal	$\Delta^{t1}_{R_1}$	$\Delta^{t2}_{R_2}$	$\Delta^{t1}_{R_3}$	ATC$^\alpha$	COVERT$^\alpha$	GA-1	GA-2	LSRW(15)	LSRW(200)
la16	1170	2000$^{(a)}$	2521	2477	2360	2360	1623	1619	$*$	$*$
la17	900	1623$^{(c)}$	2107	1555$^{(ac)}$	1936	1777	1464	1779	977	$*$
la18	929	1764	1662	1325$^{(a)}$	1439	1694	1248	1248	946	$*$
la19	948	1884	2284	1505$^{(bc)}$	1439$^{(ab)}$	2189	1696	1671	966	951
la20	809	2702	2898	1784$^{(a)}$	1866	2069	1402	1402	819	$*$
la21'	464	1280$^{(c)}$	2293	1047$^{(ac)}$	1530	1530	1044	1286	$*$	$*$
la22'	1068	1931	2270	2600	2960	1870$^{(a)}$	1550	1909	1149	1086
la23'	837	2053$^{(a)}$	2100	2131	2605	2605	1094	1094	875	875
la24'	835	1948	2692	1136$^{(abc)}$	1875	1819	1403	1479	844	$*$
orb01	2568	5247	5680	4210$^{(ac)}$	6289	4583	4005	4865	2726	2616
orb02	1412	2526	2461$^{(a)}$	2797	3022	2908	2143	2075	1434	1434
orb03	2113	4276	3968	2880$^{(ac)}$	4215	4215	2866	4647	2289	2204
orb04	1623	3285	3060	2280$^{(abc)}$	3945	3945	2326	3188	1816	1674
orb05	1593	3360	4031	1986$^{(abc)}$	3511	3573	2533	2673	1802	1662
orb06	1792	3759	4098	3371$^{(c)}$	3146$^{(a)}$	3146$^{(a)}$	3047	3689	1852	1802
orb07	590	1169	834$^{(abc)}$	922$^{(bc)}$	1324	1239	927	982	619	618
orb08	2429	2554$^{(abcd)}$	5147	4570	3777$^{(bc)}$	4404	3792	4103	2717	2554
orb09	1316	2388$^{(abc)}$	2949	2415$^{(c)}$	2679	3190	2061	2628	1449	1334
orb010	1679	3075	3186	2802$^{(ac)}$	3418	3109	2217	2913	1837	1775

"GA-1" and "GA-2" represent GA-(R&M/COVERT) and GA-(R&R) [20]; ATC$^\alpha$ and COVERT$^\alpha$ showed the results from ATC and COVERT with $\alpha = 0.4$.
"*" means the method found the optimal solution; "a" means that it is the best rule among the three evolved rules for the instance;
"b" means that the evolved rule is better than GA-(R&M/COVERT) for the instance; "c" means that the evolved rule is better than GA-(R&R) for the instance;
"d" means that the evolved rule is better than LSRW(15) for the instance; and "e" means that the evolved rule is better than LSRW(200) for the instance

ease of presentation, the total weighted tardiness is normalised using the formula proposed by [47]. Thus, the normalised weighted tardiness $= \frac{\sum w_jT_j}{\mathcal{N}\times\mathcal{M}\times\frac{1}{\mu}\times\bar{w}}$, where the number of jobs $\mathcal{N} = 100000$, the number of machines $\mathcal{M} = 6$, the average processing time of an operation $\frac{1}{\mu} = 1$ and average weight of a job $\bar{w} = 2.2$.

In Figure 11, the performance of the evolved rules are quite consistent with what has been shown when dealing with the static problem. The results show that simple rules such as FIFO and EDD are easily dominated by the sophisticated rules like ATC, WSPT, COVERT and the evolved rules. When using pairwise z-tests to compared the performance of these rules, it is noted that rule $\Delta^{t1}_{R_1}$ and rule $\Delta^{t1}_{R_3}$ are significantly better (p-value $\ll 0.05$) than existing rules ATC, COVERT and WSPT when the utilisation is less than or equal to 0.8, but they are beaten by the existing rules when utilisation is 0.9. On the other hand, the R_2 rule is significantly worse than the existing rules for all levels of utilisations. The R_2 rule is significantly worse than R_1 and R_3 rules when the utilisation is less than or equal to 0.8 and better than R_1 when utilisation is 0.9. One of the reasons for the inferiority of the evolved rules when the utilisation of the shop is high is that the evolved rules are

TABLE IX
PARAMETER SETTING OF THE SIMULATION MODEL

Parameter	Value
Number of machines	6
Arrival process	Possion with λ from 0.5 to 0.9
Processing time	Exponential with mean $\frac{1}{\mu} = 1$
Number of operations per job	6
Visiting order of jobs	randomly generated with no machine being revisited
Weight	randomly sampled from $\{4, 2, 1\}$ with the probabilities $\{0.2, 0.6, 0.2\}$

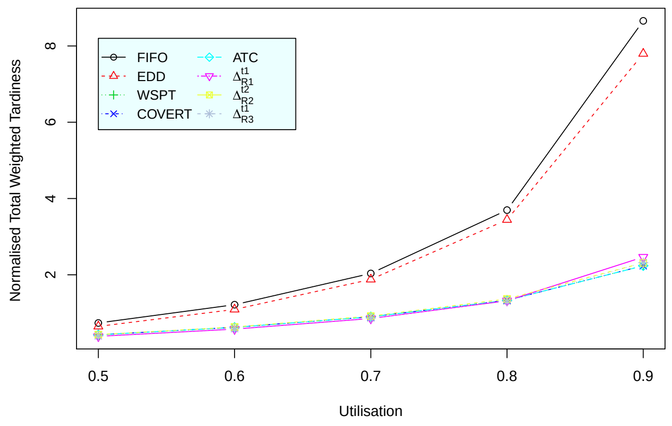


Fig. 11. Performance of the evolved rules in the dynamic environment (FIFO and EDD curves are above all other curves)

trained on a set of static instances which usually reflect the situations in the dynamic job shop with a low utilisation.

The investigation of the evolved rules in the dynamic environment shows that there is still a difference in the characteristics between the static problem and the dynamic problem which sometimes prevent the evolved rules from being effective in the dynamic environment. Although the evolved rules show good results in the dynamic $Jm||\sum w_jT_j$, the fact that they are inferior to the existing rules suggests the lack of generality of the data set used to train the dispatching rules. The deteriorated performance of the evolved rules when utilisations of machines are high is not because the GP system is not able to evolve effective rules but because the static training instances cannot represent all possible scenarios happening in a dynamic environment. Thus, if the ultimate objective of the dispatching rules is to be applied to the dynamic environment, it seems reasonable to train the rules in the dynamic situation (e.g. via a simulation model) in order to ensure that the evolved rules can capture all critical features of the dynamic system.

E. Further discussions

Many analyses have been performed and it has been noted that the GP system proposed in this paper can find effective dispatching rules for the JSP. Different from traditional applications of evolutionary computation and meta-heuristic methods which can only find a particular solution for a problem instance, the GP system can evolve reusable rules to solve different problem instances without the need to rerun the search algorithm. Since a dispatching rule only provides one schedule for a static problem instance, it is very difficult for these rules to outperform meta-heuristic methods which perform improving steps by exploring thousands of schedules to find better schedules. However, the fact that the evolved rules can beat GA-(R&R) in many instances shows that it is possible to generate powerful dispatching rules for the JSP that are competitive with meta-heuristic methods. It is noted that the program trees are restricted to the maximum depth of six in this study and many other machine attributes as well as terminals (scheduling parameters) are not considered when the GP system is used to evolve new dispatching rules. These restrictions are made to reduce the computational times of the GP system but they also prevent the GP system from finding better dispatching rules. If a better choice of function set, terminal set, and machine attributes is made, it may be possible that the GP system can evolve rules that are as good as effective meta-heuristic methods but much more efficient.

One major advantage of the evolved rules is that they are quite ready to be applied to the dynamic environment as shown in the previous subsection, which makes these rules more practical than meta-heuristic methods. Therefore the evolved rules are suitable for shops with rapid dynamic changes [59], [22], [60]. In these shops, even when mathematical programming methods are able to find optimal schedules, these schedules can quickly become sub-optimal or even infeasible under dynamic conditions [22]. Besides, with its flexibility, the proposed GP system can easily be applied to generate good rules for complex manufacturing processes (e.g. batch processing, assembly station, etc.), which are difficult to be handled by conventional optimisation methods. Another advantage of the evolved dispatching rules is that they are understandable to the users (managers, operators, and researchers), and therefore it is much easier for these people to incorporate these rules with other planning decisions compared to the detailed schedules produced by other methods.

Most of the proposed GP systems [5], [8], [40], [39] in the literature is very similar to ours when the R_2 representation is used. Miyashita [8] and Jakobovic and Budin [6] are the only works that focus on the use of system attributes, specifically bottleneck machines, to support the sequencing decisions. However, it is noted that the GP system proposed by Miyashita [8] is only suitable when we try to evolve dispatching rules for a specific shop with a small number of machines in which the bottleneck machine is known. Therefore, it is difficult to generalise the rules based on on this GP system. Jakobovic and Budin [6] improved this GP system [8] by using a dedicated GP tree to detect the bottleneck machine to apply suitable dispatching rules. The problem is that the bottleneck machine is not always a good feature to help decide which dispatching rules to be applied. When there are multiple bottleneck machines in the shop (especially with a symmetrical shop), applying a dedicated rule for the detected bottleneck is not very effective since the temporary bottleneck machine can change rapidly before that rule shows any noticeable effect. This explains why the proposed GP-3 system [6] performs significantly better than the simple GP system, but the gaps between the objectives obtained by GP-3 and the simple GP are not large. When examining the GP-3 system with our training and testing instances (using the same terminal sets for dispatching rules in R2 and R3), the experimental results show that there is no significant difference between the rules evolved by GP-3 and rules evolved with the R_2 representation. The analysis in Section VI-B also showed that the workload ratio WR (an indicator for the bottleneck level of the considered machine) is not a major attribute in the best evolved rules. This suggests that different machine and system attributes should be used instead of depend solely on bottleneck machines.

Compared to the existing methods, the novelties of this study are: (1) the introduction of new representations (R_1 and R_3) that allow the GP system to evolve effective adaptive dispatching rules by incorporating machine and system attributes; (2) the comparison of different representations for GP; and (3) the analysis of the evolved dispatching rules, which explain how they solve JSP instances effectively.

VII. Conclusions

Dispatching rules are one of the most crucial tools in production planning and control systems and a large number of studies have been conducted to develop more effective rules. However, the development process is still very complicated and time consuming. The improvement in computing power provides some alternatives to facilitate this process. In this paper, we investigated the use of genetic programming as a hyper-heuristic method for automatically discovering new dispatching rules for the job shop scheduling problem. New representations of dispatching rules which take advantage of the machine and system attributes to support the sequencing decisions were developed and examined. The experimental results suggest that the GP system can evolve effective dispatching rules for $Jm||C_{max}$ and $Jm||\sum w_jT_j$ with the representations that integrate system attributes and suitable composite dispatching rules. The evolved rules are also shown to outperform the existing rules on the training set and test set. Moreover, the best evolved rules produce competitive performances when compared to the hybrid GAs of [20]. Although the performance of the evolved rules is still far from the optimal solutions in the static JSP, they are still very promising because they are quite ready to be applied to a dynamic system whereas the solutions provided by the exact and meta-heuristic methods have trouble dealing with the dynamics of the shop change. The simulation experiments have confirmed the effectiveness of the evolved rules compared to well-known dispatching rules in the dynamic environment. Also, compared to the solutions obtained by the conventional meta-heuristic approaches, the evolved rules in this work provide much better interpretability of the sequencing decisions. Another advantage of the proposed GP system is that it only requires one run to generate dispatching rules which can be used to solve multiple problem instances.

In general, this is the first study that has compared different representations of dispatching rules used in a GP system and investigated how representations influence the performance of the GP system when evolving dispatching rules for the JSP. With their flexibility, the proposed representations in this study can be easily extended to deal with different manufacturing environments. Moreover, these representations provide a convenient way to incorporate special system features of real world environments into the adaptive dispatching rules. Therefore, the GP system proposed in this study can also be employed as a good tool to automatically discover effective dispatching rules for a particular manufacturing system. The analysis in the previous section has shown very interesting characteristics of the evolved rules.

For future studies, there are still many problems that need to be further investigated. This study has shown that the explicit incorporation of machine and system attributes into the decision making process can provide favourable results. Therefore, it would be very useful to have a comprehensive study of potential attributes and their effects on each type of manufacturing environment. Also, since the dispatching rules will be applied to dynamic situations, whether it is necessary to evolve these rules in the dynamic environment through some simulation model would be an important question for the future research. In addition, while the technique of dispatching rules can quickly react to changes encountered on the shop floor, some practical issues (cognitive and organisational) [59] need to be considered. It would be interesting to study these issues when designing the dispatching rules, such as by developing a multi-objective GP-HH method to evolve rules that can satisfy multiple practical requirements. Fitness landscapes defined by different fitness functions would be very interesting to explore and will be a good topic for future research. Finally, even though the proposed representation is useful, it also creates a large search space for the GP system to explore. For that reason, it would be worthwhile paying more attention to the design of specialised genetic operators to be able to find the good rules.

ACKNOWLEDGMENT

This work is supported in part by the Marsden Fund of New Zealand Government (VUW0806 and 12-VUW-134), administrated by the Royal Society of New Zealand, and the University Research Fund (200457/3230) at Victoria University of Wellington.

REFERENCES

[1] S. S. Panwalkar and W. Iskander, "A survey of scheduling rules," *Operations Research*, vol. 25, no. 1, pp. 45–61, 1977.

[2] M. S. Jayamohan and C. Rajendran, "Development and analysis of cost-based dispatching rules for job shop scheduling," *European Journal of Operational Research*, vol. 157, no. 2, pp. 307–321, 2004.

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

[4] L. Nie, X. Shao, L. Gao, and W. Li, "Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems," *The International Journal of Advanced Manufacturing Technology*, vol. 50, no. 5, pp. 729–747, 2010.

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

[6] D. Jakobovic and L. Budin, "Dynamic scheduling with genetic programming," in *EuroGP'06: Proceedings of the 9th European Conference on Genetic Programming*, 2006, pp. 73–84.

[7] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach," in *GECCO'10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, 2010, pp. 257–264.

[8] K. Miyashita, "Job-shop scheduling with GP," in *GECCO'00: Proceedings of the Genetic and Evolutionary Computation Conference*, 2000, pp. 505–512.

[9] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems," in *CEC '12: Proceedings of the IEEE Congress on Evolutionary Computation*, 2012, pp. 3261–3268.

[10] ——, "Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming," in *EuroGP'12: Proceedings of the 15th European Conference on Genetic Programming*, 2012, pp. 121–133.

[11] X. Li and S. Olafsson, "Discovering dispatching rules using data mining," *Journal of Scheduling*, vol. 8, no. 6, pp. 515–527, 2005.

[12] H. Ingimundardottir and T. Runarsson, "Supervised learning linear priority dispatch rules for job-shop scheduling," in *Learning and Intelligent OptimizatioN (LION 5)*, 2011, pp. 263–277.

[13] B. Giffler and G. L. Thompson, "Algorithms for solving production-scheduling problems," *Operations Research*, vol. 8, no. 4, pp. 487–503, 1960.

[14] A. Jones, L. C. Rabelo, and A. T. Sharawi, *Survey of Job Shop Scheduling Techniques*. John Wiley & Sons, Inc., 2001.

[15] H. R. Lourenco, "Job-shop scheduling: Computational study of local search and large-step optimization methods," *European Journal of Operational Research*, vol. 83, no. 2, pp. 347–364, 1995.

[16] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996.

[17] E. Balas and A. Vazacopoulos, "Guided local search with shifting bottleneck for job shop scheduling," *Management Science*, vol. 44, no. 2, pp. 262–275, 1998.

[18] R. Cheng, M. Gen, and Y. Tsujimura, "A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies," *Computers & Industrial Engineering*, vol. 36, no. 2, pp. 343–364, 1999.

[19] S. Kreipl, "A large step random walk for minimizing total weighted tardiness in a job shop," *Journal of Scheduling*, vol. 3, no. 3, pp. 125–138, 2000.

[20] H. Zhou, W. Cheung, and L. C. Leung, "Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm," *European Journal of Operational Research*, vol. 194, no. 3, pp. 637–649, 2009.

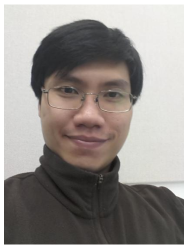
[21] K. N. McKay, F. R. Safayeni, and J. A. Buzacott, "Job-shop scheduling theory: What is relevant?" *Interfaces*, vol. 18, no. 4, pp. 84–90, 1988.

[22] S. C. Sarin, A. Varadarajan, and L. Wang, "A survey of dispatching rules for operational control in wafer fabrication," *Production Planning & Control*, vol. 22, no. 1, pp. 4–24, 2011.

[23] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*, ser. International Series in Operations Research and Management Science, vol. 146. Springer, 2010, pp. 449–468.

[24] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," School of Computer Science and Information Technology, University of Nottingham, Tech. Rep. Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747, 2010.

[25] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. R. Woodward, "Exploring hyper-heuristic methodologies with genetic programming," in *Computational Intelligence*, ser. Intelligent Systems Reference Library, C. Mumford and L. Jain, Eds. Springer Berlin Heidelberg, 2009, vol. 1, pp. 177–201.
[26] A. Bolte and U. W. Thonemann, "Optimizing simulated annealing schedules with genetic programming," *European Journal of Operational Research*, vol. 92, no. 2, pp. 402–416, 1996.
[27] A. Fukunaga, "Automated discovery of composite SAT variable-selection heuristics," in *Eighteenth National Conference on Artificial Intelligence*, 2002, pp. 641–648.
[28] M. B. Bader-El-Den and R. Poli, "Generating SAT local-search heuristics using a gp hyper-heuristic framework," in *Proceedings of the 8th International Conference on Artificial Evolution*, 2007, pp. 37–49.
[29] E. K. Burke, M. R. Hyde, and G. Kendall, "Evolving bin packing heuristics with genetic programming," in *Parallel Problem Solving from Nature (PPSN)*, 2006, pp. 860–869.
[30] E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward, "The scalability of evolved online bin packing heuristics," in *CEC'07: IEEE Congress on Evolutionary Computation*, 2007, pp. 2530–2537.
[31] ——, "A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 14, pp. 942–958, 2010.
[32] R. Keller and R. Poli, "Linear genetic programming of parsimonious metaheuristics," in *CEC'07: IEEE Congress on Evolutionary Computation*, 2007, pp. 4508–4515.
[33] ——, "Cost-benefit investigation of a genetic-programming hyperheuristic," in *Proceedings of the 8th International Conference on Artifcial Evolution*, 2007, pp. 13–24.
[34] M. B. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memetic Computing*, vol. 1, no. 3, pp. 205–219, 2009.
[35] E. K. Burke, M. R. Hyde, and G. Kendall, "Grammatical evolution of local search heuristics," *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 3, pp. 406–417, 2012.
[36] C. Dimopoulos and A. M. S. 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.
[37] D. Jakobovic, L. Jelenkovic, and L. Budin, "Genetic programming heuristics for multiple machine scheduling," in *EuroGP'07: Proceedings of the 10th European Conference on Genetic programming*, 2007, pp. 321–330.
[38] C. D. Geiger, R. Uzsoy, and H. Aytug, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *Journal of Heuristics*, vol. 9, no. 1, pp. 7–34, 2006.
[39] 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.
[40] W. J. Yin, M. Liu, and C. Wu, "Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming," in *CEC'03: IEEE Congress on Evolutionary Computation*, 2003, pp. 1050–1055.
[41] L. Atlan, J. Bonnet, and M. Naillon, "Learning distributed reactive strategies by genetic programming for the general job shop problem," in *Proceedings of the 7th Annual Florida Artificial Intelligence Research Symposium*, 1994.
[42] C. Pickardt, J. Branke, T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness," in *Proceedings of the 2010 Winter Simulation Conference (WSC)*, 2010, pp. 2504–2515.
[43] J. A. Vazquez-Rodriguez and G. Ochoa, "On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming," *Journal of the Operational Research Society*, vol. 62, no. 2, pp. 381–396, 2011.
[44] D. J. John, "Co-evolution with the Bierwirth-Mattfeld hybrid scheduler," in *GECCO '02: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, 2002, p. 259.
[45] S. Petrovic and E. Castro, "A genetic algorithm for radiotherapy pre-treatment scheduling," in *Applications of Evolutionary Computation*, 2011, pp. 454–463.
[46] W. J. Hopp and M. L. Spearman, *Factory Physics: Foundations of Manufacturing Management*. McGraw-Hill, 2000.
[47] A. P. J. Vepsalainen and T. E. Morton, "Priority rules for job shops with weighted tardiness costs," *Management Science*, vol. 33, no. 8, pp. 1035–1047, 1987.
[48] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
[49] P. Pardalos and O. Shylo, "An algorithm for the job shop scheduling problem based on global equilibrium search techniques," *Computational Management Science*, vol. 3, no. 4, pp. 331–348, 2006.
[50] S. Luke, *Essentials of Metaheuristics*. Lulu, 2009. [Online]. Available: http://cs.gmu.edu/∼sean/book/metaheuristics/
[51] S. Lawrence, "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques," Ph.D. dissertation, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.
[52] D. Applegate and W. Cook, "A computational study of the job-shop scheduling instance," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149–156, 1991.
[53] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.
[54] E. Demirkol, S. Mehta, and R. Uzsoy, "Benchmarks for shop scheduling problems," *European Journal of Operational Research*, vol. 109, no. 1, pp. 137–141, 1998.
[55] K. R. Baker, "Sequencing rules and due-date assignments in a job shop," *Management Science*, vol. 30, no. 9, pp. 1093–1104, 1984.
[56] 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.
[57] P. Mizrak and G. M. Bayhan, "Comparative study of dispatching rules in a real-life job shop environment," *Applied Artificial Intelligence*, vol. 20, pp. 585–607, 2006.
[58] D. C. Mattfeld and C. Bierwirth, "An efficient genetic algorithm for job shop scheduling with tardiness objectives," *European Journal of Operational Research*, vol. 155, no. 3, pp. 616–630, 2004.
[59] J. Riezebos, J. M. Hoc, N. Mebarki, C. Dimopoulos, W. Wezel, and G. Pinot, "Design of scheduling algorithms: Applications," in *Behavioral Operations in Planning and Scheduling*, J. C. Fransoo, T. Waefler, and J. R. Wilson, Eds., 2011, pp. 371–412.
[60] T. Sloan, "Shop-floor scheduling of semiconductor wafer fabs: exploring the influence of technology, market, and performance objectives," *IEEE Transactions on Semiconductor Manufacturing*, vol. 16, no. 2, pp. 281–289, 2003.



Su Nguyen received a BE degree in Industrial and Systems Engineering at Ho Chi Minh City University of Technology, Vietnam in 2006 and a ME degree in Industrial Engineering and Management at Asian Institute of Technology (AIT), Thailand. He is currently a PhD candidate in Evolutionary Computation Research Group at Victoria University of Wellington (VUW), New Zealand. Prior to VUW, he was a research associate in Industrial and Manufacturing Engineering at the School of Engineering and Technology, AIT. His primary research interests include evolutionary computation, discrete-event simulation and their applications in operations planning and scheduling.

Mengjie Zhang received a BE and an ME in 1989 and 1992 from Artificial Intelligence Research Center, Agricultural University of Hebei, China, and a PhD in computer science from RMIT University, Australia in 2000.

Since 2000, he has been working at Victoria University of Wellington, New Zealand. He is currently Professor of Computer Science and heads the Evolutionary Computation Research Group. His research is mainly focused on evolutionary computation, particularly genetic programming and particle swarm optimisation with application areas of image analysis, multi-objective optimisation, classification with unbalanced data, and feature selection and dimension reduction for classification with high dimensions. He has published over 200 academic papers in refereed international journals and conferences. He has been serving as an associated editor or editorial board member for five international journals (including IEEE Transactions on Evolutionary Computation and the Evolutionary Computation Journal) and as a reviewer of over fifteen international journals. He has been serving as a steering committee member and a program committee member for over eighty international conferences. He has supervised over thirty postgraduate research students.

Prof Zhang is a senior member of IEEE, a member of the IEEE Computer Society, the IEEE CI Society and the IEEE SMC Society. He is also a member of the IEEE CIS Evolutionary Computation Technical Committee, a member of the IEEE CIS Intelligent Systems Applications Technical Committee, a vice-chair of the IEEE CIS Task Force on Evolutionary Computer Vision and Image Processing, and a committee member of the IEEE New Zealand Central Section. He is a member of ACM and the ACM SIGEVO group.

Mark Johnston holds a BSc in mathematics and computer science and a PhD in operations research from Massey University, New Zealand. He is a senior lecturer at Victoria University of Wellington, New Zealand, where he teaches various operations research courses. His research is primarily in combinatorial optimisation and evolutionary computation, with particular interest in scheduling models, genetic programming and multiple-objective optimisation. Dr Johnston is a member of IEEE.

Kay Chen Tan is currently an Associate Professor in the Department of Electrical and Computer Engineering, National University of Singapore. Dr Tan has published over 100 journal papers, over 100 papers in conference proceedings, co-authored 5 books and co-edited 4 books. Dr Tan has been invited to be an invited keynote/pleanary speaker for over 30 international conferences. He served in the international program committee for over 100 conferences and involved in the organizing committee for over 40 international conferences, including the General Co-Chair for IEEE Congress on Evolutionary Computation 2007 in Singapore and the General Co-Chair for IEEE Symposium on Computational Intelligence in Scheduling 2009 in Tennessee, USA.

Dr Tan is an IEEE Distinguished Lecturer of IEEE Computational Intelligence Society since 2011. Dr Tan is currently the Editor-in-Chief of IEEE Computational Intelligence Magazine (CIM). He also serves as an Associate Editor / Editorial Board member of over 15 international journals, such as IEEE Transactions on Evolutionary Computation, IEEE Transactions on Systems, Man and Cybernetics: Part B Cybernetics, IEEE Transactions on Computational Intelligence and AI in Games, Evolutionary Computation (MIT Press) etc. Dr Tan is the awardee of the 2012 IEEE Computational Intelligence Society (CIS) Outstanding Early Career Award. He also received the Recognition Award (2008) from the International Network for Engineering Education & Research (iNEER). He was also a winner of the NUS Outstanding Educator Awards (2004), the Engineering Educator Awards (2002, 2003, 2005), the Annual Teaching Excellence Awards (2002, 2003, 2004, 2005, 2006), and the Honour Roll Awards (2007).