A Hybrid Genetic Programming Algorithm for Automated Design of Dispatching Rules

Su Nguyen p.nguyen4@latrobe.edu.au
Centre for Data Analytics and Cognition, La Trobe University, Australia

Yi Mei yi.mei@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, New Zealand

Bing Xue bing.xue@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, New Zealand

Mengjie Zhang mengjie.zhang@ecs.vuw.ac.nz
Evolutionary Computation Research Group, Victoria University of Wellington, New Zealand

Abstract
Designing effective dispatching rules for production systems is a difficult and time-consuming task if it is done manually. In the last decade, the growth of computing power, advanced machine learning, and optimisation techniques has made the automated design of dispatching rules possible and automatically discovered rules are competitive or outperform existing rules developed by researchers. Genetic programming is one of the most popular approaches to discovering dispatching rules in the literature, especially for complex production systems. However, the large heuristic search space may restrict genetic programming from finding near optimal dispatching rules. This paper develops a new hybrid genetic programming algorithm for dynamic job shop scheduling based on a new representation, a new local search heuristic, and efficient fitness evaluators. Experiments show that the new method is effective regarding the quality of evolved rules. Moreover, evolved rules are also significantly smaller and contain more relevant attributes.

Keywords
Genetic programming, job shop, production scheduling, hyper-heuristics

1 Introduction

Scheduling is one of the important tasks in production planning and control (PPC), which can directly influence the production costs, delivery speed, and customer satisfaction (Wiendahl et al., 2005). In general, the goal of scheduling is to effectively allocate the production resources over time to process production orders (e.g. from forecasting or customer orders). In the literature, various production scheduling problems have been investigated such as parallel machine scheduling, flow shop scheduling, and job shop scheduling. Many solution methods have been developed (Pinedo, 2008) based on industrial practice, artificial intelligence (e.g. genetic algorithm, constraint programming), and operations research (e.g. mathematical programming, branch-and-bound). Due to their computational complexity, finding optimal solutions for these scheduling problems is very challenging (Baker and Trietsch, 2009) and can be impractical in

some cases. As a result, scheduling heuristics have been proposed to find good enough solutions in reasonable running times (Jones and Rabelo, 1998). Usually designing an effective scheduling heuristics is time-consuming and requires a lot of problem domain knowledge. However, because of the increasing dynamics and the turbulent environment, there are needs for a more responsive and adaptive production system (Ouelhadj and Petrovic, 2008; Bannat et al., 2011). Genetic Programming (GP) (Koza, 1992) has recently been used as a powerful technique for automated design of production scheduling heuristics and also known as GP based hyper-heuristics (Burke et al., 2009). The main goal of this approach is to automatically discover sophisticated and effective scheduling heuristics for complex production environments.

GP has been applied to evolve production scheduling heuristics for many production environments such as single machine scheduling (Dimopoulos and Zalzala, 2001; Geiger et al., 2006; Nie et al., 2010), parallel machine scheduling (Jakobovic et al., 2007; Durasevic et al., 2016), and (flexible) job shop scheduling (Miyashita, 2000; Nie et al., 2013; Nguyen et al., 2014a; Hunt et al., 2014; Hart and Sim, 2016). In these studies, evolved scheduling heuristics are usually in the form of priority dispatching rules which are used to assign priorities for jobs in the shop. The rules are encoded as GP programs by using different representations such as tree-based structure, fixed length linear structure, or neural network (Nguyen et al., 2013b; Branke et al., 2015). Then the evolved rules are evaluated by a number of simulated scenarios and new rules are generated based on genetic operators of GP and the fitness of rules in the population. Although the idea is relatively simple, GP can find very competitive rules which usually outperformed many rules manually designed in the past studies. Moreover, multiple PPC decisions (Yin et al., 2003; Nie et al., 2013) and multiple conflicting objectives (Nguyen et al., 2014a; Karunakaran et al., 2016) can also be handled by GP. Recently, more advanced techniques have also been proposed to improve both the effectiveness and efficiency of GP for automated heuristic design such as surrogate-assisted GP (Hildebrandt and Branke, 2015) and GP based ensemble methods (Park et al., 2016; Hart and Sim, 2016).

Despite the success of GP for automated design of production scheduling heuristics, there are a number of challenges. First, the computational costs of GP are high as many evolved rules are generated and evaluated by expensive computer simulation. Second, the search space of priority dispatching rules is quite large so finding effective rules is very difficult. Third, including a wide range of attributes (e.g. for jobs, work centre) can help to explore rules that have been discovered in the literature; however, it may further enlarge the search space and deteriorate GP performance. Finally, the obtained rules are usually large and include many redundant components, which causes difficulties for interpretation. While the first three challenges may restrict GP from searching for (near) optimal priority dispatching rules because of the limited computational budget and the complexity of the heuristic search space, the final challenge restricts us from understanding how the evolved rules work.

The goal of this paper is to tackle the current challenges by developing a hybrid GP for evolving dispatching rules for dynamic job shop scheduling (DJSS). The three key contributions are: (1) development of a new hybrid GP algorithm can evolve effective dispatching rules for DJSS, (2) development of a new representation which allows GP to remove less relevant attributes through the evolutionary process, and (3) development of a new efficient local search heuristic to improve the exploitation ability of GP. The core idea of the proposed method is to enhance the GP search mechanism by enhancing the effectiveness of genetic operators and adding an efficient local search

heuristic that can refine rules evolved by GP. This is motivated by the fact that the chance to produce bad dispatching rules by genetic operators of GP is quite high and it would be wasteful to evaluate these rules especially when the evaluations are computationally expensive. A multi-fidelity surrogate modeling approach is adopted to screen out poor-performing rules which are generated by the genetic operators and/or via the local search heuristic (i.e. rules in the neighborhood of an incumbent rule). Moreover, the new representation also allows GP to take attribute selection into account to narrow down the search space adaptively through generations and to improve the interpretability of evolved rules.

The rest of this paper is organised as follows. In Section 2, a brief introduction of job shop scheduling is given, and a review of recent studies on automated design of production scheduling heuristics is provided. The dynamic job shop problem investigated in this study and its simulation model is presented in Section 3. Detailed descriptions of the proposed algorithm are given in Section 4. The design of experiments is shown in Section 5 and the results and analyses are provided in Section 6. Finally, Section 7 concludes the paper.

2 Background

Job shop scheduling (JSS) is an interesting optimisation problem in artificial intelligence and operations research. The term job shop is used to indicate companies that produce customer-specific components in small batches (Land, 2004). In job shops, each customer order or job needs to be handled by some specific machines. Depending on the technical requirements, jobs can have different processing times and special routes through a set of machines. Because jobs will compete for the production resources, scheduling is needed to decide when jobs are processed to optimise the production speed and customer satisfaction. Because of the complexity of this production environment, effectively scheduling production resources is difficult. In past studies, researchers have developed many scheduling techniques to deal with both *static* and *dynamic* JSS problems. In static problems, all jobs are available at the beginning of the schedule, and the goal is to find the optimal sequence of these jobs on each machine. For dynamic problems, jobs will arrive randomly over time, and their information is only available upon their arrivals.

2.1 Solution methods for job shop scheduling

Past studies mainly focused on static JSS and treated them as combinatorial optimisation problems. Because JSS is an NP-hard problem, exact optimisation methods such as dynamic programming and branch-and-bound (Pinedo, 2008) can only be used to solve very small problem instances. Therefore, they are not suitable to handle large-scale practical scheduling problems. In the last few decades, many heuristics have been proposed for JSS to find *quick* and *good enough* solutions. Dispatching rules are among the most popular scheduling techniques to deal with both static and dynamic JSS. These rules prioritise jobs competing for some machine based on the information of jobs and machines and the one with the highest priority will be processed next. The advantages of dispatching rules are their ease of implementation and efficiency.

Other heuristics based on understandings of problem domains have also been proposed in the literature such as shifting bottlenecks (Applegate and Cook, 1991). More general techniques based on meta-heuristics such as tabu search (Nowicki and Smutnicki, 1996), simulated annealing (Aydin and Fogarty, 2004), and genetic algorithms (Yamada and Nakano, 1995; Bierwirth and Mattfeld, 1999; Cheng et al., 1999) have been

developed to deal with different production scheduling problems and show promising results. Many hybrid methods have also been developed in literature to combine the advantages of various heuristics and meta-heuristics (Goncalves et al., 2005; Kacem et al., 2002; Sha and Hsu, 2006; Xia and Wu, 2005; Zhou et al., 2009). However, most of these are designed for a particular static JSS problem.

2.2 Automated design of production scheduling heuristics

The field of automated heuristic design or hyper-heuristics (Burke et al., 2007, 2010) has been very active recently to facilitate the design of heuristics for hard computational problems. The goal of this approach is to explore the *"heuristic search space"* of the problems instead of the solution search space in the cases of heuristics and meta-heuristics. The motivation of this approach is to reduce the time needed to design heuristics from the human experts and to increase the chance to explore a wide range of powerful and undiscovered heuristics.

Different machine learning methods have also been applied for discovering new dispatching rules for JSS. Olafsson and Li (2010) used the decision tree method on optimal production data to generate dispatching rules. Attribute selection and instance selection have also been considered to improve the interpretability and accuracy of induced decision trees (Li and Olafsson, 2005). Meanwhile, Ingimundardottir and Runarsson (2011) proposed a logistic regression 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. Shiue (2009) proposed a GA/SVM method for dynamic dispatching rule selection in which a support vector machine (SVM) is used for classification to select suitable dispatching rules based on job and machine attributes and GA is used for attribute selection. Artificial neural network (Weckman et al., 2008; Eguchi et al., 2008) have also been applied to determine the priority of each given job based on the conditions of machines in the shop.

2.3 Genetic programming for production scheduling

In the last decade, genetic programming has been the dominating technique for automated design for production scheduling heuristics (Branke et al., 2016; Nguyen et al., 2017). As compared to other hyper-heuristics, genetic programming (GP) has shown some key advantages. First, GP has flexible representations which allow various heuristics to be represented as computer programs. Second, GP has powerful search mechanisms which can operate in the heuristic search space to find optimal or near-optimal scheduling heuristics. Different from the supervised learning methods mentioned above, GP can simultaneously explore both the structure and corresponding parameters. Moreover, many evolutionary multi-objective optimisation (EMO) techniques are also available in the literature to help GP design effective heuristics to deal with multiple conflicting objectives. Finally, heuristics obtained by GP can be partially interpretable and very efficient, which is a critical feature to enhance its applicability in practice.

GP has been applied to many production scheduling problems ranging from single machine scheduling (Nie et al., 2010; Geiger et al., 2006), parallel machine scheduling (Durasevic et al., 2016), to (flexible) job shop scheduling (Tay and Ho, 2008; Vazquez-Rodriguez and Ochoa, 2011; Nie et al., 2013; Nguyen et al., 2013b; Mei et al., 2016; Hart and Sim, 2016). In these studies, GP have been used to develop very competitive dispatching rules which outperformed the existing heuristics in most cases. For job shop scheduling, many aspects of automated heuristic design with GP have been

investigated. Different representations of dispatching rules have been compared in Nguyen et al. (2013b) and Branke et al. (2015). GP methods for evolving non-dominated dispatching rules for multiple conflicting objectives in JSS have been investigated in Nguyen et al. (2013c) and Karunakaran et al. (2016). The experimental results showed that GP could evolve non-dominated rules with very promising trade-offs which have not been explored in the previous studies. Other studies have also shown that GP can be extended to handle multiple planning and scheduling decisions in job shops simultaneously and dynamically (Nguyen et al., 2014a; Nie et al., 2013), which are great challenges for other methods.

Representations of scheduling heuristics have been investigated in many papers. The tree-based representation of the traditional GP technique (Koza, 1992) is usually used in previous studies because it can easily represent priority functions in most scheduling heuristics (for both static and dynamic problems). The linear representation of gene expression programming (GEP) has also been applied to some studies and produced competitive results as compared to GP representations (Nie et al., 2013). Nguyen et al. (2013b) and Hunt et al. (2015) used grammar-based representations to restrict the search space of GP and improve the understandability of evolved dispatching rules. Also trying to improve the interpretability of evolved dispatching rules, Durasevic et al. (2016) applied dimensionally aware GP to ensure that the evolved rules are semantically correct. For complicated scheduling problems with more than one scheduling decisions to be made, specialised representations have been developed. For example, Jakobovic and Budin (2006) proposed the GP-3 method to simultaneously evolve two specialised rules and a discriminating function to check if the considered machine is a bottleneck.

Recently surrogate models have been proposed to reduce the computational costs of GP. These models have reduced the evaluation costs of GP and improved its convergence. Hildebrandt and Branke (2015) proposed surrogate model based on the phenotypic characterisation of evolved priority functions. In this technique, the phenotype of an evolved heuristic is characterised by a decision vector with the dimension of K where K is the number of decision situations (each decision situation includes a number of jobs to be prioritised). First, a reference rule (e.g. - 2(PT+WINQ+NPT)) is selected and applied to all decision situations. The ranks of jobs (smaller ranks for jobs with higher priorities) in each situation determined by the reference rule are recorded. For each evolved priority function, the corresponding ranks are also determined, and the decision value for each decision situation is the rank determined by the reference rule of the job whose corresponding rank obtained by the evolved priority function is 1. An archive is used to store past explored rules and their decision vectors which are recorded during GP evolution. During the reproduction process, the fitness of a newly generated rule is approximated by the fitness of the closest rule in the archive based on the distance between their corresponding decision vectors. This surrogate model, even though simple, can provide a good estimation of fitness and help to screen out bad rules created by crossover and mutation.

Also trying to reduce the computational times of GP for automated heuristic design, Nguyen et al. (2016) proposed a new technique to estimate the fitness of evolved rules using a simplified version of the original simulated shop. Instead of evaluating evolved heuristics with the model of the original shop which can be very large, a smaller model of the shop is created with a smaller number of machines, shorter simulation length while maintaining the same level of utilisation, due date tightness, etc. A set of benchmark rules are applied to different models, and their objective values are

recorded. The simplified model that has the highest rank correlation with the original model will be used to estimate the fitness of newly generated rules. Then, only the ones with highest estimated fitness are moved to the next generation and evaluated by the original model. The proposed GP technique based on this simplified models showed better results as compared to other GP methods.

In recent years, more advanced techniques have been applied to improve effectiveness and efficiency of GP for automated design of dispatching rules. Although the results of these studies are very promising, many practical issues have not been addressed. For example, attribute selection is an important issue to improve the performance and interpretability of automated heuristic design, but it has not been systematically investigated in GP. In most previous studies, only a small set of attributes are manually selected as the inputs for GP, but there are potentially a large number of attributes that need to be considered when dealing with complex job shops. In these cases, manual selection may not be accurate while maintaining a large attribute set can dramatically enlarge the search space of GP. Moreover, given the large and complex search of scheduling heuristics, discovering good heuristics is a very challenging task. Although GP has excellent exploration ability, it is not effective in refining potential evolved rules (i.e. exploitation ability). Therefore, a key challenge is to balance both exploration and exploitation abilities of GP. In Section 4, a hybrid GP is proposed to overcome these difficulties.

3 Simulation model

In this paper, a symmetrical job shop which has been used in previous studies (Nguyen et al., 2013c; Hildebrandt and Branke, 2015) is employed to evaluate the quality of dispatching rules. Below is the simulation configuration:

- 10 machines
- Each job has 2 to 14 operations (re-entry is allowed)
- Processing times follow discrete uniform distribution $U[1, 99]$
- Job arrivals follow Poisson process
- Due date = current time + allowance factor $\times$ total processing time (allowance factor of 4 is used in our experiments)
- Utilisation of the shop is 85% or 95%
- No machine break-down; preemption is not allowed
- Weights of jobs are assigned based on the $4 : 2 : 1$ rule (Kreipl, 2000; Pinedo and Singer, 1999), i.e. 20% of jobs with the weight of 1, 60% with the weight of 2, and 20% with the weights of 4.

In each simulation replication, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the 1000^{th} job is considered as the warm-up time and the statistics from the set $\mathbb{C}$ of the next completed 5000 jobs (Holthaus and Rajendran, 2000) will be used to calculate performance measures. Three scheduling performance measures examined in our experiments are (1) mean tardiness, (2) maximum tardiness, (3) total weighted tardiness. The details of these performance measures are provided in Table 1. In this table, the tardiness is $T_j = \max\{C_j - d_j, 0\}$

Table 1: Performance measures of dispatching rules

Mean Tardiness	$\texttt{MeanT} = \frac{1}{\lvert\mathbb{C}\rvert}\sum_{j\in\mathbb{C}} T_j$
Maximum Tardiness	$\texttt{MaxT} = \max_{j\in\lvert\mathbb{C}\rvert} T_j$
Total Weighted Tardiness	$\texttt{TWT} = \sum_{j\in\mathbb{C}} w_j T_j$

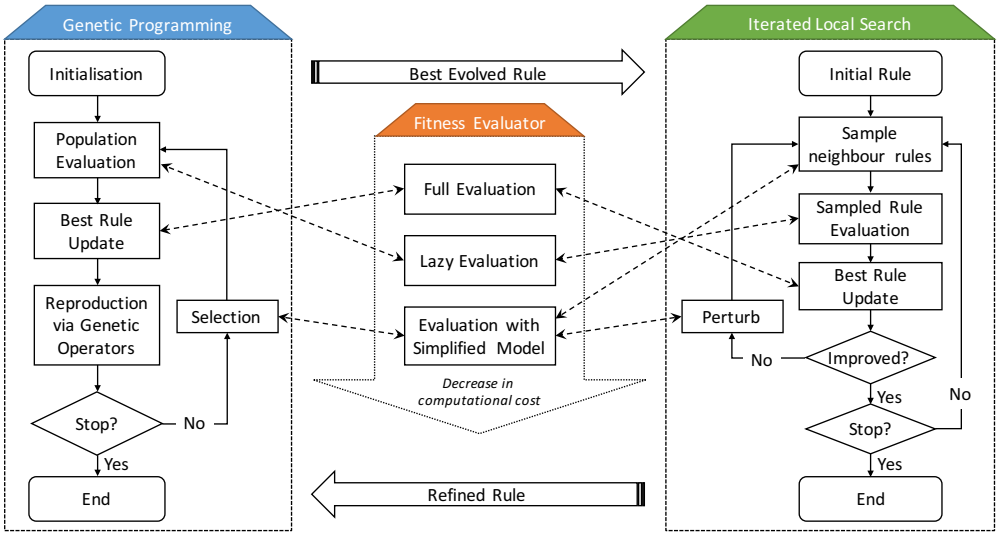


Figure 1: Hybrid genetic programming with multi-fidelity fitness evaluations.

where C_j and d_j are the completion time and the due date of job j. Although this simulation model is relatively simple, it still reflects key issues of real manufacturing systems such as dynamic changes and complex job flows. This section only considers the shop with utilisation of 95% and 85%, and tight due date (allowance factor of 4) because scheduling in this scenario is more challenging, which is easier to demonstrate the usefulness of GP.

4 Proposed method

Figure 1 presents an overview of the proposed hybrid genetic programming (HGP) method. The three main components of this method are (1) genetic programming, (2) fitness evaluator, and (3) iterated local search. HGP starts with initialisation to generate a population of random dispatching rules $P \leftarrow \{\Delta_1, \ldots, \Delta_n\}$. The quality of newly generated rules will be evaluated by using the fitness evaluator. Based on the obtained fitness, genetic operators (i.e. crossover, mutation) are applied to produce new rules which are then selected for the next generation of GP. GP will evolve its population through a number of generations and stop when the stopping condition is met.

During the evolution, the best rule Δ^* obtained by GP will be frequently sent to the iterated local search (ILS) for further refinement (see Section 5). From the initial (incumbent) rule Δ^c of ILS (the best found rule from GP) , a set of rules are sampled from its neighbourhood $\mathcal{N}(\Delta^c)$ (more details are provided in Section 4.3). The selected rules from the sample set will be evaluated, and the best rule will be updated if im-

Inputs: simulation model of DJSS, simplified model $\mathcal{S}$
Output: the best evolved rule Δ^*

randomly initialise the population $P \leftarrow \{\Delta_1, \ldots, \Delta_n\}$
set $\Delta^* \leftarrow null$ and the fitness $f^* \leftarrow +\infty$
setup a set of replications $\mathcal{R}$ for full evaluation
$generation \leftarrow 0$, select a replication π
while *stopping condition is reached* **do**
    $f_l^* \leftarrow +\infty$ and $\Delta_l^* \leftarrow null$
    for all $\Delta_i \in P$ **do**
        evaluate $f_l(\Delta_i)$ by applying a Δ_i to π
        if $(f_l(\Delta_i) < f_l^*)$ **then**
            $\Delta_l^* \leftarrow \Delta_i$
            $f_l^* \leftarrow f_l(\Delta_i)$
        end if
    end for
    obtain $f(\Delta_l^*)$ by applying a Δ_l^* to $\mathcal{R}$
    if $(f(\Delta_l^*) < f^*)$ **then**
        $\Delta^* \leftarrow \Delta_l^*$
        $f^* \leftarrow f(\Delta_l^*)$
    end if
    if local search is activated **then**
        $\Delta^*, f^* \leftarrow$ apply iterated local search to Δ^*
    end if
    $P' \leftarrow$ apply genetic operations to P
    for all $\Delta_i \in P'$ **do**
        obtain $f_s(\Delta_i)$ by applying a Δ_i to $\mathcal{S}$
    end for
    replace P with the top $|P|$ rules $\Delta_i \in P'$
    select a new π
    $generation \leftarrow generation + 1$
end while
return Δ^*

Figure 2: Proposed HGP algorithm.

provements are made. In the case that no improvement are made after some trials, a perturbation is applied to change the incumbent rule. The search of ILS will stop when the termination condition is met, and the best rule found by ILS Δ^* will be returned to GP.

To search for effective dispatching rules, the operators from both GP and ILS require information about the fitness of generated rules (e.g. crossover in GP or perturbation step in ILS). However, the full evaluations (based a large number of simulation replications) of dispatching rules are expensive and cannot be applied arbitrarily. For that reason, the fitness evaluator in this study provides a number of ways to estimate the quality of dispatching rules: (1) full evaluations $f(\cdot)$, (2) lazy evaluations $f_l(\cdot)$, and (3) evaluations with a simplified model $f_s(\cdot)$ (details are provided in Section 4.2). These evaluations are different in computational times and prediction accuracy. Depending on the requirements of each step in GP and ILS, suitable evaluations will be applied, as shown in Figure 1. The pseudo code for the proposed HGP is provided in Figure 2.

Table 2: Terminal and function sets of GP

Symbol	Description
rrJ	time-in-system of job ($\mathtt{t} - releasetime$)
rRJ	job queuing time ($\mathtt{t} - readytime$)
RO	number of remaining operation within the job.
RT	work remaining of the job
PR	operation processing time
rDD	time to due date $= (\mathtt{DD} - \mathtt{t})$
SJ	slack of the job $= \mathtt{DD} - (\mathtt{t} + \mathtt{RT})$
W	weight of the job
NPT	processing time of the next operation
WINQ	work in the next queue
APT	average operation processing time of jobs in the queue
NJIQ	number of jobs in the queue
MINPQ	minimum processing time of jobs in the queue
MAXPQ	maximum processing time of jobs in the queue
NJIS	number of jobs in the shop
NJNQ	number of jobs in the next queue of machine that job will visit next
MINDQ	minimum due date of jobs in the queue
MAXDQ	maximum due date of jobs in the queue
MAXWQ	maximum weight of jobs in the queue
WOR	aggregate workload at machines that the job has not yet visited
#	Random number from 0 to 1
Function set	$+$,$-$,$\times$, $\%$, min, max

*t is the time when the sequencing decision is made; DD is the due date of job. $releasetime$ and $readytime$ are the time the job arrived at the system and the considered machine respectively.

In the rest of this section, we will introduce the representation of dispatching rules for GP along with the genetic operators, the fitness evaluator, and the iterated local search heuristic.

4.1 Representation

To make the evolved rules easier for interpretation and to help GP select relevant attributes during the evolution, a new representation is proposed. A rule Δ_i is represented in GP by two parts: (1) the priority function in the tree structure $ptree_i$ and (2) a attribute vector $avec_i$. Similar to previous studies, $ptree_i$ is constructed from a set of attributes $\mathcal{A} \leftarrow \{a_1, a_2, \ldots, a_A\}$ (where A is the number of attributes) and a set of function $\mathcal{F}$. Whenever scheduling decisions need to be made, the $ptree_i$ will be evaluated with the attributes values extracted from considered jobs and machines. The description of attributes and functions used in this study is shown in Table 2.

The outputs of $ptree_i$ are priorities of jobs, and the one with the highest priority is selected to process. Previous studies have shown that the tree-based representation of priority function is very useful to evolve powerful scheduling heuristics (Hildebrandt et al., 2010; Nguyen et al., 2013a,c). One drawback of the tree-based representation (and other variable-length representations) is that the evolved rules or functions are large and complicated. They can also include many attributes that are not useful to determine job priorities (Nguyen et al., 2013b; Branke et al., 2015). Mei et al.

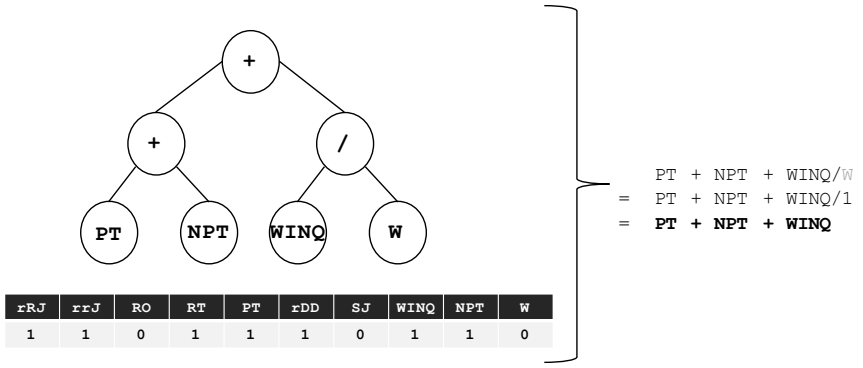


Figure 3: Example of a dispatching rule with the proposed representation.

(2016, 2017) have shown that GP can find significantly better dispatching rules if a good attribute subset is applied. However, no practical attribute selection technique is developed in their study. In our algorithm, the attribute vector $avec_i$ is incorporated and coevolved with $ptree_i$. The reason for including $avec_i$ into the representation is to help GP automatically identify relevant features on-the-fly. The attribute vector $avec_i = \{x_{i1}, x_{i2}, \ldots, x_{iA}\}$ is a binary array and its dimensionality is A. If x_{ik} is 1, the $ptree_i$ will be evaluated as usual. However, if x_{ik} is 0, the value of attribute a_k, if included in $ptree_i$, will be fixed to 1; and therefore, the attribute a_k has no effect on the priorities of jobs set by the rule $\Delta_i \leftarrow \{ptree_i, avec_i\}$. An example of a dispatching rule with this representation is showed in Figure 3. In this example, the $ptree_i$ represents the function `PT + NPT + WINQ/W`. Meanwhile, the $avec_i$ in the lower part of Figure 3 shows that there are ten attributes used to construct priority functions, but only seven attributes are active for the $ptree_i$ under consideration. The other three attributes, `RO`, `SJ`, and `W`, will be treated as 1 if they are included in $ptree_i$. Thus, the rule can be simplified to `PT + NPT + WINQ`. It is noted that replacing attribute a_k with $x_{ik} = 0$ by the constant 1 is just one simple approach to remove the impact of a_k on the considered rule. Alternatively, the replacement value can be adapted based on the actual operation's neutral value or by the outputs of automatically-defined functions (Koza, 1992) without a_k.

4.1.1 Initialisation

In the proposed HGP algorithm, the population P includes a set of dispatching rules $\Delta_i \leftarrow \{ptree_i, avec_i\}$. In the first generation, the population is randomly initialised. The ramped-half-and-half technique (Koza, 1992) is used to generate the random expression tree $ptree_i$ for each rule. For the attribute vector $avec_i$, all initial dispatching rules possess the same $avec_i \leftarrow \{1, 1, \ldots, 1\}$, which means that all attributes will be active when rules are evaluated. The reason for making all attributes active in the first generation of HGP is that it is very likely that random generated $ptree_i$ will result in poor performance with or without relevant attributes. Thus, using random $avec_i$ in the early generations may make HGP accidentally ignore relevant and important attributes before it can search for good $ptree_i$.

4.1.2 Genetic operators

For the proposed representations, new genetic operators are also developed to generate both $ptree_i$ and $avec_i$ for a dispatching rule. These operators will be used in step

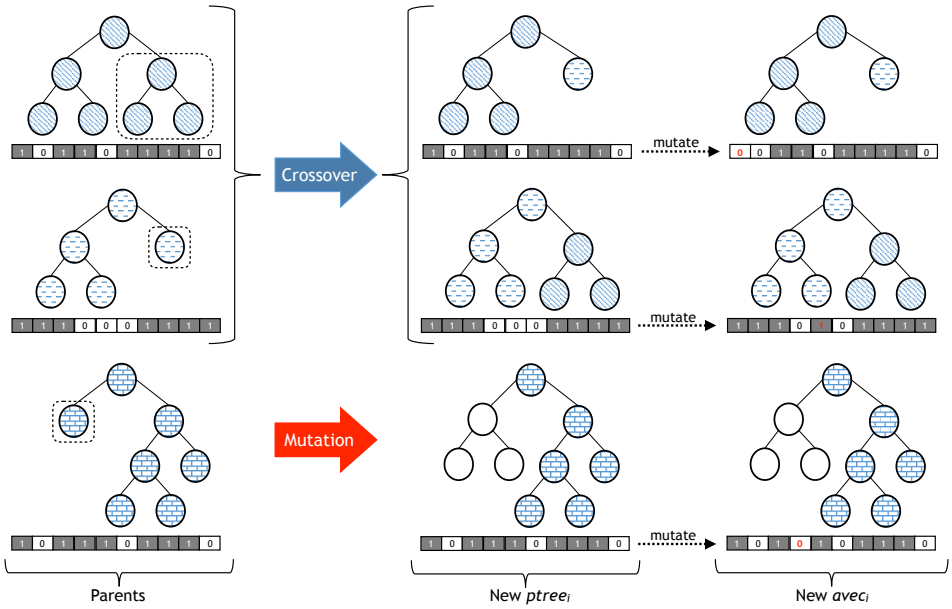


Figure 4: Illustration of genetic operators in HGP.

22 of Figure 2 and treated as the search operators in the ILS algorithm (described in Section 4.3). As a rule Δ_i has two components $ptree_i$ and $avec_i$, we apply a two-step procedure to use genetic operators as illustrated in Figure 4. In the first step, the subtree crossover and the subtree mutation (Koza, 1992) are applied to selected parents (by tournament selection) to generate new $ptree_i$. The subtree crossover creates new individuals for the next generation by randomly recombining subtrees from two selected parents. Meanwhile, the subtree mutation is performed by selecting a node of a chosen individual and replacing the subtree rooted at that node with a newly randomly-generated subtree. In this step, $avec_i$ is copied directly from the parent. After a new $ptree_i$ is generated, another mutation can be applied to the copied $avec_i$ with the probability of p_a. This mutation operator only randomly picks one attribute $k \in \mathcal{A}$ and the value of x_{ik} in $avec_i$ will be flipped. If $p_a = 0$, the reproduction process here will be the same as one in the traditional GP because only the trees (priority functions) are evolved. More subsets of attributes will be examined by HGP as p_a increases.

4.2 Fitness evaluator

Three approaches are used to estimate the fitness of evolved rules in different stages of the algorithms. In the proposed algorithm in Figure 2, $f(\Delta_i)$ is the real fitness calculated based on objective values obtained by applying Δ_i to a large number of simulation replications with a long simulation length (completions of 5000 jobs as shown in Section 3). The goal of $f(\Delta_i)$ is to measure the steady state performance of Δ_i for a given job shop scenario (Law and Kelton, 1999; Holthaus and Rajendran, 2000).

Because of expensive simulation costs, using a large number of replications to obtain all $f(\cdot)$ of all Δ_i generated in the population are very slow. To improve the efficiency of HGP, multi-fidelity fitness evaluation approach is used across the HGP algorithm. Within each generation of HGP, we only use one replication to evaluate the

quality of evolved rules $f_l(\Delta_i)$ (the subscript l is for *lazy evaluation*). This strategy has been shown to be useful to improve the effectiveness of evolved rules and the diversity in the population (Hildebrandt et al., 2010) and significantly reduce the computational times of GP. The rule with the best $f_l(\Delta_i)$ is then fully evaluated to obtain $f(\Delta_i)$.

After an intermediate population P' (with $|P'| > |P|$) generated by genetic operators and the estimated fitness $f_l(\Delta_i)$ (for selecting parents), the fitnesses of newly generated rules are quickly evaluated by using the simplified model $\mathcal{S}$. In this study, the simplified model $\mathcal{S}$ is the $HalfShop$ model proposed in (Nguyen et al., 2016), which has shown to be an effective and cheap model to estimate the quality of dispatching rules. The simplified $HalfShop$ is only half the scale of the original shop discussed in Section 3. In $HalfShop$, the number of machines is 5, and the maximum number of operations of a job is 7 (instead of 14). The warm-up time and the simulation length are governed by the arrival of the 100^{th} job and the completion of 500 jobs respectively. Nguyen et al. (2016) showed that this simplified model provides a good estimation $f_s(\Delta_i)$ (the subscript s is for *simplified model*) of the quality of evolved rules without significant increase in the computational cost of GP.

The use of these three approaches to estimating fitness of evolved rules makes the algorithm slightly more complicated but they allows us to utilise the computational budgets more effectively. The analyses from Hildebrandt and Branke (2015) (with $f(\Delta_i)$ and a surrogate model), Nguyen et al. (2014b) (with $f(\Delta_i)$, $f_l(\Delta_i)$, and a surrogate model), and Nguyen et al. (2016) (with $f(\Delta_i)$, $f_l(\Delta_i)$, and $f_s(\Delta_i)$) showed that such combinations can significantly improve the convergence of GP algorithms. The fitness $f_s(\Delta_i)$ determines the rough quality of generated heuristics. Because the likelihood to produce bad dispatching rules via crossover and mutation by GP is very high, GP may waste a lot of time evaluating bad heuristics. The fitness estimated efficiently by the simplified model helps the algorithm screen out rules with poor performance (steps 23–26 in Figure 2) and reduce the computational costs. Meanwhile $f_l(\Delta_i)$ helps the algorithm identify the most potential rules (step 8) for full evaluations (step 14) and improve the diversity of HGP. Moreover, $f_l(\Delta_i)$ is used within tournament selection to select parent rules for genetic operations discussed in Section 4.1.2. If $Cost(\cdot)$ and $Accuracy(\cdot)$ are the computational costs and accuracy of an estimation approach, we will have $Cost(f(\Delta_i)) > Cost(f_l(\Delta_i)) > Cost(f_s(\Delta_i))$ and $Accuracy(f(\Delta_i)) > Accuracy(f_l(\Delta_i)) > Accuracy(f_s(\Delta_i))$. In this paper, we try to improve the effectiveness of the search mechanism by maintaining good trade-offs between $Cost(\cdot)$ and $Accuracy(\cdot)$ across the algorithm.

4.3 Iterated local search

The previous discussions showed how GP could be used to evolve dispatching rules for DJSS. To further improve the quality of the final evolved rule, an ILS heuristic is developed. As compared to other hybrid methods that combine EC and local search, the frequency of applying ILS in HGP is quite low because of the high computational cost. Through the evolution of HGP, ILS is applied every N_{ils} generations and only the best dispatching rule Δ^* is refined by ILS. The pseudo code of ILS is presented in Figure 5. This search heuristic is extended from APRILS developed in Nguyen et al. (2015). However, APRILS is too computationally expensive to be used within our proposed HGP. Moreover, APRILS was not designed to deal with a large number of attributes. In this paper, we improve the efficiency and effectiveness of APRILS by using multi-fidelity fitness evaluations.

At the beginning, the best rule Δ^* is assigned as the initial rule $\Delta^c \leftarrow$

Inputs: initial rule Δ^c, simulation model of DJSS, simplified model $\mathcal{S}$
Output: the best rule Δ^*

set $\Delta^* \leftarrow \Delta^c$ and the fitness $f^* \leftarrow f(\Delta^c)$
setup a set of replications $\mathcal{R}$ for full evaluation
$iteration \leftarrow 0$, select a replication π
while $iteration \leq maxIteration$ **do**
    sample a set $\mathcal{M} \leftarrow \{\Delta_1, \ldots, \Delta_{n_s}\}$ from the neighbourhood of Δ^c
    for all $\Delta_i \in \mathcal{M}$ **do**
        obtain $f_s(\Delta_i)$ by applying a Δ_i to $\mathcal{S}$
    end for
    create $\mathcal{M}'$ from the top n'_s rules $\Delta_i \in \mathcal{M}'$
    $f_l^* \leftarrow +\infty$ and $\Delta_l^* \leftarrow null$
    for all $\Delta_i \in \mathcal{M}'$ **do**
        evaluate $f_l(\Delta_i)$ by applying a Δ_i to π
        if $(f_l(\Delta_i) < f_l^*)$ **then**
            $\Delta_l^* \leftarrow \Delta_i$
            $f_l^* \leftarrow f_l(\Delta_i)$
        end if
    end for
    obtain $f(\Delta_l^*)$ by applying a Δ_l^* to $\mathcal{R}$
    if $(f(\Delta_l^*) < f^*)$ **then**
        $\Delta^* \leftarrow \Delta_l^*$
        $f^* \leftarrow f(\Delta_l^*)$
        $\Delta^c \leftarrow \Delta^*$
    end if
    if no improvement for f^* is made in NI iterations **then**
        $\Delta^c \leftarrow perturb(\Delta^*, \Delta_l^*)$
    end if
    select a new π
    $iteration \leftarrow iteration + 1$
end while
return Δ^*

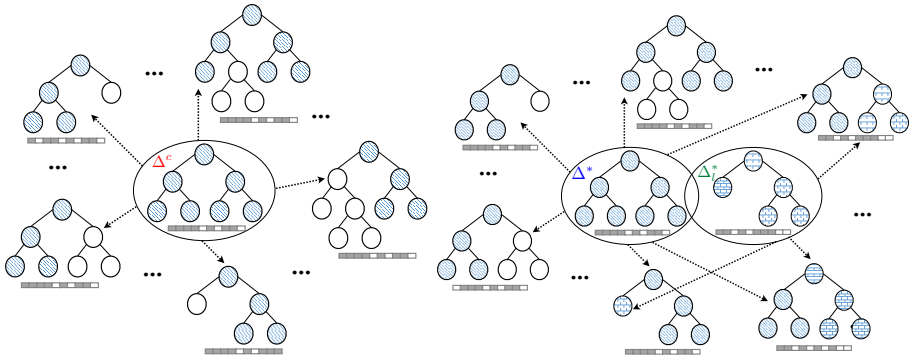
† $|P'| = k \times |P|$

Figure 5: Proposed Iterated Local Search.

$\{ptree_c, avec_c\}$ for ILS. An important aspect of local search that strongly influences its performance is the neighbourhood structure. Similar to APRILS, a restricted subtree mutation (RSM) is applied to generate the neighbour priority functions of Δ^c. RSM is similar to the subtree mutation presented in Section 4.1.2 but the depth of the randomly generated subtree is restricted to 2. This restriction tries to avoid large mutations which make the neighbour priority functions too different from the current $ptree_c$ in Δ^c. Meanwhile, RSM still allows new components (e.g., attributes, functions) to be introduced into the newly generated rules. In the proposed ILS, we do not put any constraint on RSM and it is possible for the root node of an evolved rule to be selected. The chance to select the root node is high when the rule is small (early generations) but it will be reduced as rules in the population evolve. An interesting advantage of this naive mutation approach is that it can help ILS find compact rules without significantly increasing their lengths.

Different from APRILS, attribute mutation is also applied to $avec_c$ to generate the attribute vector for the new neighbour dispatching rules. Because the neighbourhood $\mathcal{N}(\Delta^c)$ is very large (because of the tree structure, the number of attributes, and the number of functions), it is impossible to examine all neighbour rules. Therefore, ILS only samples a number of rules in $\mathcal{N}(\Delta^c)$. A two-stage sampling procedure is applied to help ILS search the neighbourhood effectively. In the first stage (steps 5–8 of Figure 5), n_s rules are sampled from $\mathcal{N}(\Delta^c)$, as illustrated in Figure 6(a). The sample $\mathcal{M} \leftarrow \{\Delta_1, \Delta_2, \ldots, \Delta_{n_s}\}$ is then evaluated with $\mathcal{S}$. Then a truncated sample $\mathcal{M}'$ is created from the top rules in $\mathcal{M}$, based on $f_s(\cdot)$. All $\Delta_i \in \mathcal{M}'$ are evaluated to calculate $f_l(\Delta_i)$. The best rule Δ_l^* is selected for a full evaluation and the best rule Δ^* and Δ^c are updated if an improvement in f^* is made.

This searching process (steps 5–23 in Figure 5) will be repeated until the there is no improvement for f^* made in the last NI iterations. In this case, it is assumed that Δ^c is trapped at a local optimum, and perturbation or a "*kick*" is applied to generate a new Δ^c. However, to avoid ILS from wasting time search from a poor solution, the new Δ^c will be produced from the current best rule Δ^* and the best-sampled rule Δ_l^*. In the perturbation step, a large number (2000) of rules are generated by applying mutation (to Δ^*) and crossover operators (to Δ^* and Δ_l^*) with equal probability, as illustrated in Figure 6(b). The rule with the best $f_s(\cdot)$ is then assigned to Δ^c. ILS will stop when the maximum iteration ($maxIteration$) is reached.



(a) Sampling neighbour rules (b) Perturbation

Figure 6: Search operators in ILS.

5 Experiment design

This section shows the experiments conducted our study and the parameter settings for HGP.

5.1 Experiment settings

The job shop simulation model presented in Section 3 is used to test the performance of the proposed HGP. Six simulation scenarios considered from our experiments are based on the combinations of three objectives (mean tardiness, maximum tardiness, and total weighted tardiness) and two utilisation levels (85% and 95%). The goal of HGP is to learn/evolve the dispatching rules for each scenario. For testing, the rule obtained by each independent run of HGP is applied to 50 independent simulation replications of the corresponding simulation scenario (in training) with different random seeds from

the ones of the training simulations. The test performance (average objective values) is the quality of the evolved rules (rules with lower average objective values are better). For testing, all rules are applied to the same set of simulation replications (Hildebrandt and Branke, 2015; Nguyen et al., 2016).

Two other GP methods are also applied and compared to HGP. The first one is the simple GP (SimGP) method (Hildebrandt et al., 2010). The representation of dispatching rules in SimGP is similar to HGP except that the attribute vectors $avec_i$ are not considered. Similar to HGP, SimGP also uses one simulation replication (changed for each generation) to evaluate the performance of rules in the population before sending the best rule Δ_l^* for full evaluations. A key difference is that SimGP did not use the simplified model to estimate $f_s(\cdot)$ in the reproduction stage. To investigate the impact of feature selection mechanism, local search heuristic ILS (lines 19–21 of Figure 2), and fitness evaluators, we examine three other GP methods in our experiments: (1) SimGP with feature selection mechanism (SimGPFS), (2) HGP without ILS (HGP-*nols*), (3) HGP without feature selection mechanism (HGP-*nofs*). All methods are compared in terms of effectiveness of evolved rules and length (number of nodes) of evolved rules, and number of attributes selected in evolved rules. Time limits are used as the stopping condition for the five GP methods. Particularly, 30 minutes and 60 minutes are used to evolve rules for scenarios with utilisation of 85% and 95% respectively as the GP methods almost converged after these time limits. Evolved dispatching rules are also compared with a set of popular benchmark dispatching rules as explained in Table 3 (Sels et al., 2011).

Table 3: Benchmark dispatching rules

Rules	Descriptions
RR	Raghu and Rajendran
COVERT	(weighted) cost over time
ATC	(weighted) apparent tardiness cost
PT+WINQ+NPT+WSL	PT+WINQ plus NPT and waiting slack
PT+WINQ+SL	processing time plus WINQ and slack
2PT+WINQ+NPT	double processing time plus WINQ and NPT
SPT+PW+FDD	shortest processing time +PW plus earliest flow due date
Slack/OPN	slack per remaining operations
Slack	slack
FIFO	first in first out
EDD	earliest due date
CR	critical ratio
CR+SPT	critical ratio plus processing time
WSPT	weighted shortest processing time
LWKR	least work remaining

*PT: processing time; WINQ: work in next queue; PW: process waiting time; NPT: next processing time

5.2 Parameter settings

The terminal and function sets for the five GP methods are presented in Table 2. Table 4 shows the parameters used in the proposed HGP algorithm. The upper part in this table are parameters used in the HGP algorithm in Figure 2 and the lower part shows the parameters for the ILS algorithm in Figure 5. For SimGP and HGP-*nofs*, the pa-

Table 4: Parameter settings

	Parameter	Description
GP	Initialisation	ramped-half-and-half
	Subtree crossover/mutation	80%/15%
	Attribute mutation probability p_a	{0.1,0.5,0.9}
	Elitism rates	5%
	Maximum depth	8
	Population size	200
	Size of intermediate population P'	2000
	Selection	tournament selection (size = 5)
ILS	Sample size n_s	500
	Sample size n'_s	50
	Number of non-improvement iterations NI	5
	Maximum iteration $maxIteration$	10

rameter p_a, the size of P', and ILS related parameters are irrelevant. Similarly, SimGP, SimGPFS, and HGP-*nols* will not use the parameters used in the lower part of Table 4. In ILS, the attribute mutation probability p_a is the same as the one used in HGP. To reduce the computation cost, HGP and HGP-*nofs* only activates ILS every ten generations (including the first generation).

The GP parameters in Table 4 have been used in the previous studies (Nguyen et al., 2015, 2016) and showed good results. The new parameter p_a has not been investigated before and we need to conduct some pilot experiments to determine the suitable value. In this pilot experiments, only ten attributes, in the upper part of Table 2, are used and ILS is not applied (i.e. we use HGP-*nols*). The results of our pilot experiments, with 30 independent runs for each p_a value, are presented in Figure 7 and Figure 8. In these figures, the label on the top of each subplot indicates the simulation scenarios under consideration $\langle obj, uti, allow\rangle$, where obj, uti, and $allow$ are respectively the objective, the utilisation, and the allowance factor. HGP-*nols* with $p_a = 0$ means that the attribute vector is not considered, and no explicit attribute selection mechanism is applied (this setting is the same as the surrogate-assisted GP proposed in Nguyen et al. (2016)). Figure 7 and Figure 8 showed the test performance and the number of selected attributes of the best rule evolved by HGP. It is noted that $p_a = 0$ is not presented in Figure 8 because it does not have an explicit attribute selection mechanism.

Figure 7 shows that the attribute mutation probability did not have clear influence on the test performance of the evolved rules. The performance is slightly better with $p_a = 0.5$ in the scenarios $\langle tmean, 85, 4\rangle$ and $\langle tmax, 95, 4\rangle$. However, from Figure 8, p_a clearly influences attribute selection in HGP. For most scenarios, $p_a = 0.5$ and 0.9 lead to rules with a number of selected attributes which is significantly smaller than that obtained with $p_a = 0.1$. There is no clear difference between $p_a = 0.5$ and $p_a = 0.9$. These pilot experiments show that HGP is able to select relevant attributes through its evolution process without deteriorating the performance of evolved rules. In our experiments in the next section, we will use $p_a = 0.5$ as it provides good results in terms of both the test performance and the number of selected attributes in most scenarios.

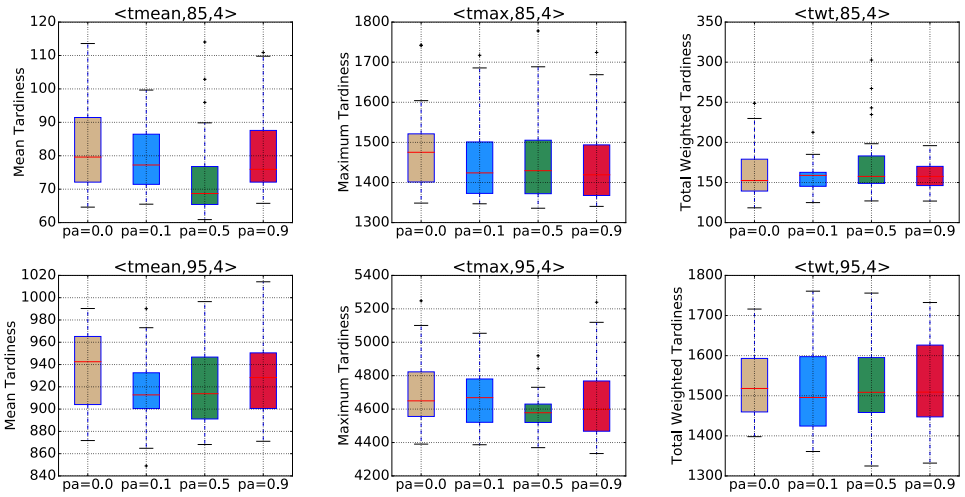


Figure 7: Influence of attribute probability p_a on the test performance.

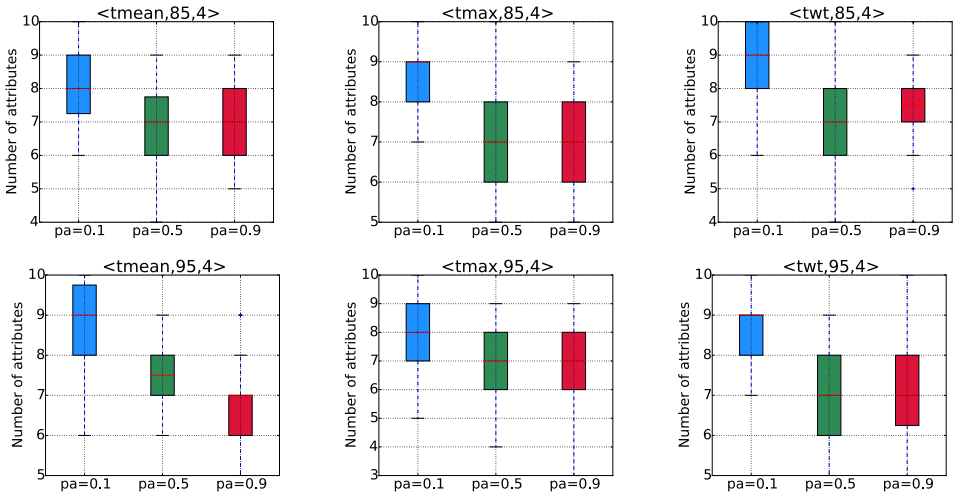


Figure 8: Influence of attribute probability p_a on the number of selected attributes.

6 Results and analyses

In this section, the results of our experiments are presented. As mentioned earlier, the proposed HGP are compared to SimGP, SimGPFS, HGP-*nols*, and HGP-*nofs* to verify its effectiveness and efficiency. The two key performance metrics used for the comparison are the test performance and the lengths of evolved rules of the five GP methods. For the two metrics, smaller values are better and Wilcoxon rank sum test, with significant level of 0.05, is used for statistical significance tests.

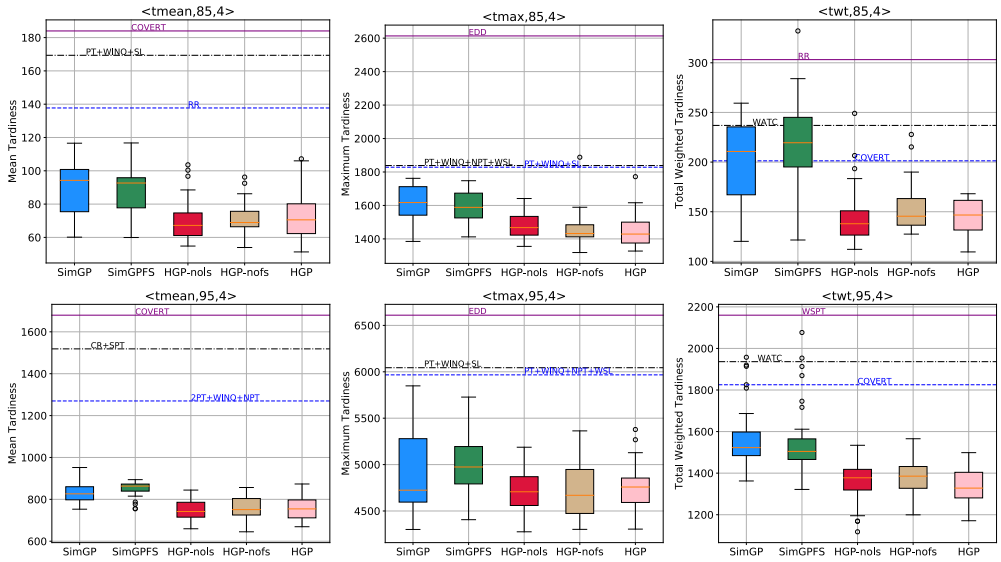


Figure 9: Test performance of dispatching rules evolved by SimGP, HGP-*nols*, and HGP.

6.1 Test performance of evolved rules

Figure 9 shows the test performance (boxplots) of the best dispatching rules evolved by GP methods through the 30 independent runs. The horizontal lines show the top three existing rules in the Table 3. Overall, it is easy to see that HGP-*nols*, HGP-*nofs*, and HGP outperform SimGP and SimGPFS in all simulated scenarios (lower boxplots are better). The rules evolved by HGP variants can dominate the majority of rules evolved with SimGP and SimGPFS. The gaps between SimGP, SimGPFS and our proposed methods are much clearer when we deal with more complex scenarios. It is also noted that HGP-*nols* and HGP have lower variances as compared with SimGP and SimGPFS in most scenarios except for $\langle tmean, 95, 4\rangle$. For the scenarios with total weighted tardiness as the objective, SimGP and SimGPFS result in very high variance and many (poor) outliers. This clearly indicates the limitation of a simple GP method when dealing with complex scenarios. In general, there is no significant difference between, HGP-*nols*, HGP-*nofs*, and HGP regarding test performance.

The evolved rules show superior performance as compared with existing rules in the literature. In the extreme case $\langle tmean, 85, 4\rangle$, the mean tardiness obtained by the best existing rule **RR** is 200% larger than those obtained with rules evolved by HGP variants. For $\langle tmean, 95, 4\rangle$, the rules evolved by the five GP methods are also a lot better than the best existing rule **2PT+NPT+WINQ** (a very competitive rule in the literature for mean tardiness and mean flowtime). A similar pattern can also be observed for the scenarios with maximum tardiness as the objective although the gaps are not as extremely large. For $\langle twt, 85, 4\rangle$, the best existing rule **COVERT** is better than a majority of rules evolved by SimGP and SimGPFS. This means that SimGP and SimGPFS are unable to find competitive rules within the time limit. However, most rules evolved by HGP variants can outperform **COVERT**. For $\langle twt, 95, 4\rangle$, the rules evolved by SimGP and SimGPFS are much more competitive as compared to **COVERT** but there are some evolved rules that are outperformed by **COVERT**. HGP variants

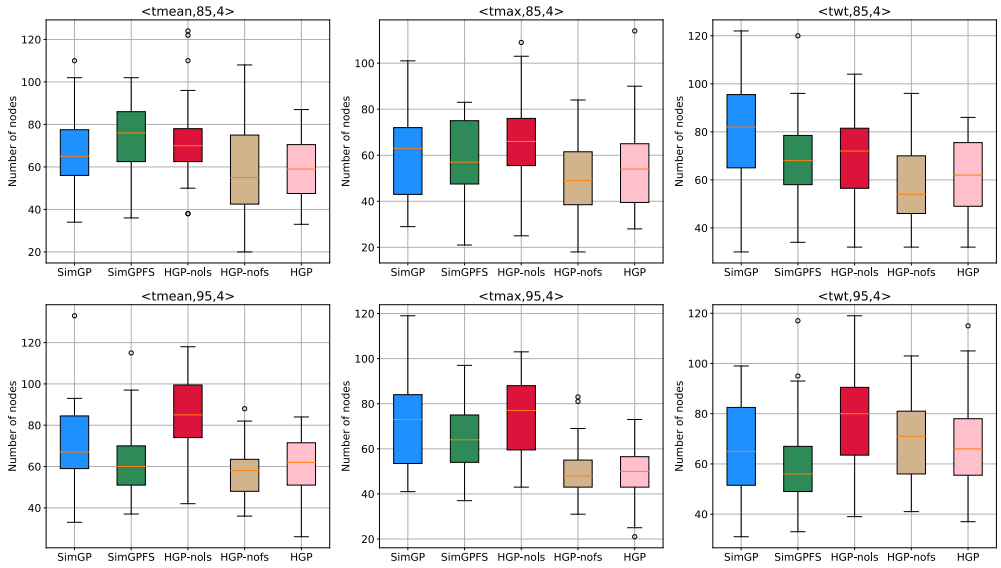


Figure 10: Length of dispatching rules evolved by SimGP, HGP-*nols*, and HGP.

still demonstrate robust results in this scenario.

6.2 Program length

In terms of rule lengths (i.e. the number of nodes in the evolved rules), there is no significant difference between SimGP and SimGPFS but they both produce smaller rules as compared to HGP-*nols* in most scenarios with utilisation of 95%, as shown in Figure 10. This is consistent with findings in previous studies (e.g. Nguyen et al. (2016)) which revealed that more effective rules evolved by GP methods are usually larger. In our experiments, SimGP and SimGPFS are significantly better than HGP-*nols* in three out of the six considered scenarios with the utilisation of 95%. For scenarios with the utilisation of 85%, good rules can be found more quickly; therefore, the lengths of these rules are only slightly different.

While HGP is very similar to HGP-*nols*, the experiment results show that HGP can produce more compact rules (with a smaller number of nodes). In terms of program length, HGP is significantly better than HGP-*nols* in five scenarios except for $\langle tmax, 85, 4\rangle$. As compared to SimGP and SimGPFS, HGP is significantly worse in one scenario $\langle twt, 95, 4\rangle$ and significantly better in three scenarios $\langle tmean, 85, 4\rangle$, $\langle twt, 85, 4\rangle$, and $\langle tmax, 95, 4\rangle$ (there is no significant difference in the remaining scenarios). Given that HGP is significantly better than SimGP and SimGPFS regarding test performance in all scenarios, the fact that HGP can effectively control the rule length makes it a more attractive method for designing dispatching rules. HGP-*nofs* is similar to HGP in all scenarios. This competitive advantage of HGP and HGP-*nofs* is achieved by the inclusion of the proposed ILS. More analyses will be provided in Section 6.3 to support this claim.

From the experiment results, we can see that the proposed HGP and its variant HGP-*nofs* is effective regarding the test performance and the program length when dealing with different job shop conditions as compared to SimGP, SimGPFS, HGP-*nols*, and existing dispatching rules. Also, different from other GP methods which tend to

produce large rules to be effective, HGP can control the length of the evolved rules quite well. This will make the evolved rules easier to interpret, and evaluations of such rules are also faster.

6.3 Further analyses

In the rest of this section, we will further investigate how HGP can achieve good results by looking at its evolution process and the attributes selected.

6.3.1 Behaviours of HGP through generations

Figures 11–13 show the behaviours of HGP, HGP-*nols*, and SimGP through the evolutionary process. The lines in Figures 11–13 are the average values from 30 independent runs and the shaded areas represent the standard errors. Because all GP methods start with the same population, their starting points are the same. Figure 11 shows the fitness $f(\Delta^*)$ of the best evolved rule through generations for different simulation scenarios. First, it is clear that HGP has a very distinctive pattern. The HGP lines for $f(\Delta^*)$ fall sharply after a certain period. This is because ILS is activated very ten generations as mentioned in Section 5. For all scenarios, it only takes HGP usually less than ten generations (about 10 minutes for scenarios with utilisation of 85% and 30 minutes for scenarios with utilisation of 95%) to obtain rules with the same quality as the best rule Δ^* evolved by SimGP at the end of the evolution. These results clearly demonstrate the superiority of HGP over SimGP for all scenarios. Although the running times of HGP are higher than those of SimGP when evolving rules for the same number of generations, HGP can find effective rules much faster than SimGP. This explains why HGP can easily outperform SimGP given the same time limit.

Between HGP and HGP-*nols*, Figure 11 shows that HGP can improve $f(\Delta^*)$ much faster at the beginning of the evolution because the best rules Δ^* obtained by GP usually perform poorly in the first few generations and can be easily improved with the local search heuristic. However, HGP-*nols* can catch up with HGP performance in the later generations since the evolved rules have been more sophisticated and the local search heuristic becomes less efficient (the neighborhood $\mathcal{N}(\Delta)$ is much larger).

As discussed previously, HGP can control the length of evolved rules quite well as compared to SimGP and HGP-*nols*. Figure 12 shows the length (number of nodes) of Δ^* across generations ($ptree_i$ for HGP-*nols* and HGP). In most scenarios, HGP evolves significantly larger rules in early generations. This is because ILS tried to evolve more effective rules by searching neighbourhood with restricted subtree mutations. Therefore, the rules refined by ILS tends to be large as compared to rules evolved by GP only (e.g. SimGP and HGP-*nols*). Then, the length of Δ^* grows slowly in later generations, which is different from SimGP and HGP-*nols*. Probably, rules refined by ILS help HGP find good building blocks within the rules with little redundancy (or introns); therefore, the evolved rules in the later generations will be more compact and contain fewer redundant components. In most cases, HGP-*nols* tends to grow much larger rules as compared to SimGP and HGP, especially in $\langle tmean, 95, 4\rangle$ and $\langle tmax, 95, 4\rangle$. It suggests that handling a large search space (with $ptree_i$ and $avec_i$) without the support of ILS can significantly increase the length of evolved rules.

Figure 13 shows the number of active attributes in $avec_i$ of the best evolved rule Δ_i. Since SimGP does not contain $avec_i$, we did not show SimGP lines in Figure 13. n general, the number of attributes selected by HGP and HGP-*nols* reduces quite smoothly through generations, which suggests that HGP and HGP-*nols* can successfully reduce the number of attributes to be included in the evolved rules. However, as HGP-*nols*

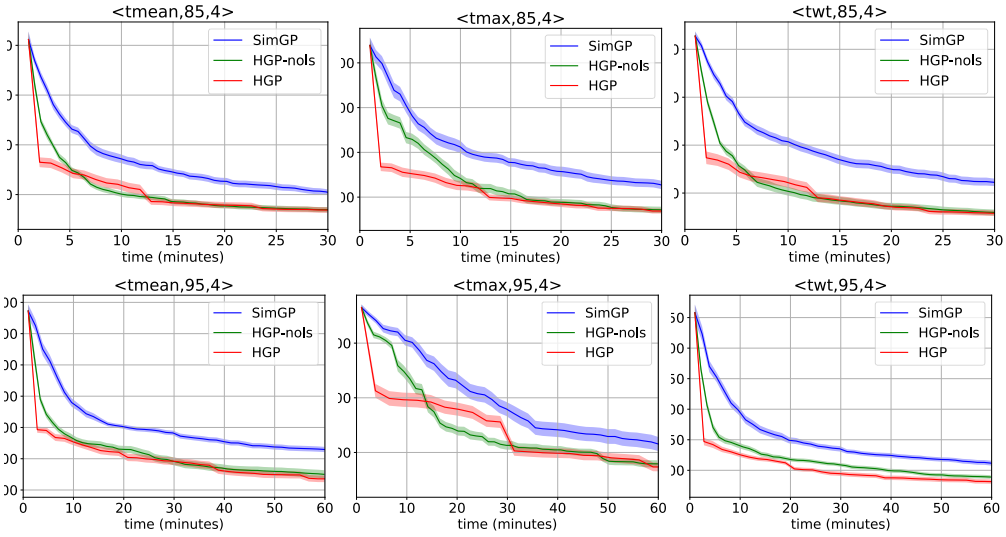


Figure 11: Fitness of the best evolved dispatching rules through generations.

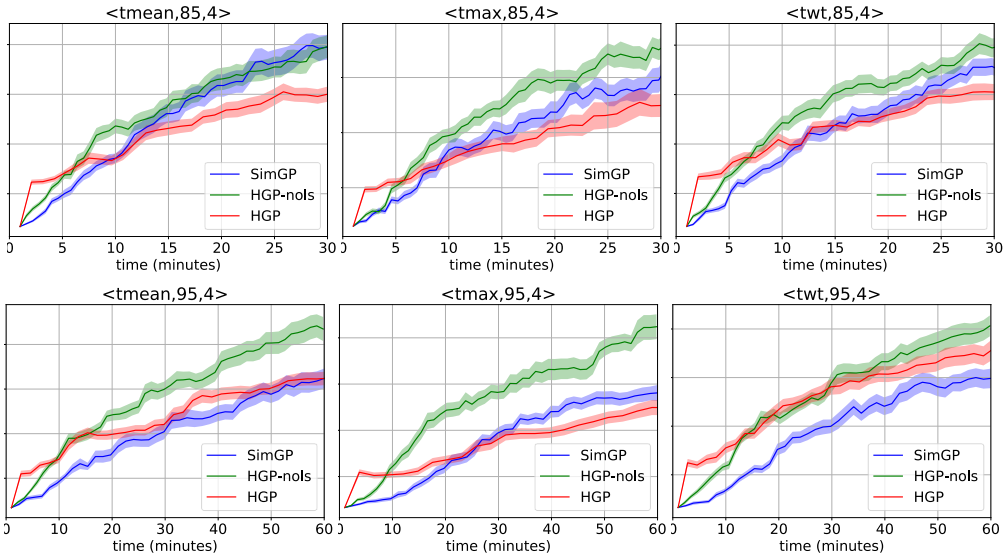


Figure 12: Length of the best evolved dispatching rules through generations.

does not use ILS, it can normally evolve through more generations than HGP given the same time limit and the evolved rules will have fewer active attributes.

The above analyses provide some useful insights about the role of ILS. While ILS can help HGP find more compact and competitive rules, it does not help HGP reduce the number of active attributes although the same mutation operator (and with the same p_a) is applied to the attribute vector in ILS. It means that the attribute mutation operator is very unlikely to produce improved dispatching rules and the improvements achieved through the local search heuristic are mainly obtained by the restricted

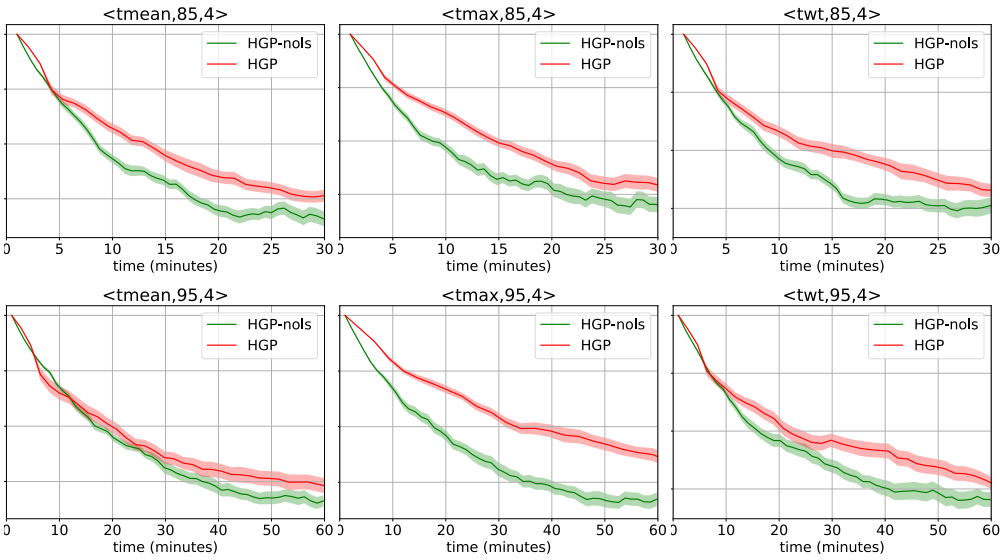


Figure 13: Number of features selected in dispatching rules through generations.

subtree mutation. Here, we see an interesting trade-off between selecting small attribute subsets and producing compact dispatching rules. Obviously, achieving these two goals simultaneously is not trivial and requires the algorithm to maintain a balance between exploration (via GP evolution) and exploitation (via ILS). The periodic application of ILS in the proposed HGP is certainly not the perfect option, and a more intelligent approach should be developed in future studies.

6.3.2 Attribute selection

Here we also want to compare attribute selection ability of HGP and HGP-*nols* to the natural attribute selection of SimGP. For all rules Δ^* obtained by the five GP methods, we determine the number of unique attributes $N_a < A$ (constants are not counted). For example, N_a for the tree in Figure 3 is 4. For SimGP, N_a is the number of attributes needed for the evolved rules. For HGP-*nols* and HGP, we determine N'_a which is the number of active attributes, i.e. the number of unique attributes with the corresponding $x_{ik} \in avec_i$ equal to 1. For the example in Figure 3 , N'_a is 3. It is noted that N'_a is always smaller than or equal to N_a. For SimGP, $N_a = N'_a$ as no explicit attribute selection mechanism is used. Therefore, N_a represents the baseline for the natural attribute selection of GP. Figure 14 shows values of N'_a and $(N_a - N'_a)$, i.e. number of inactive attributes, from rules evolved by the five GP methods. Although the numbers of included attributes N_a (the red part and the blue part) of HGP variants are slightly higher than SimGP and SimGPFS, the numbers of active attributes N'_a for HGP-*nols* and HGP are slightly smaller (except for $\langle tmean, 95, 4\rangle$). HGP-*nofs* performs poorly in terms of the number of selected attributes. Overall, the average numbers of attributes selected by HGP-*nols* and HGP are smaller than 10 in most scenarios, i.e. more than 50% of attributes are not used. Also, it is interesting that the number of included attributes for HGP-*nols* is smaller than that of HGP although HGP-*nols* tends to produce larger rules (see Figure 10 and Figure 12). A possible explanation is that HGP-*nols* tends to reuse attributes (or duplicate important subtree via the subtree crossover) at differ-

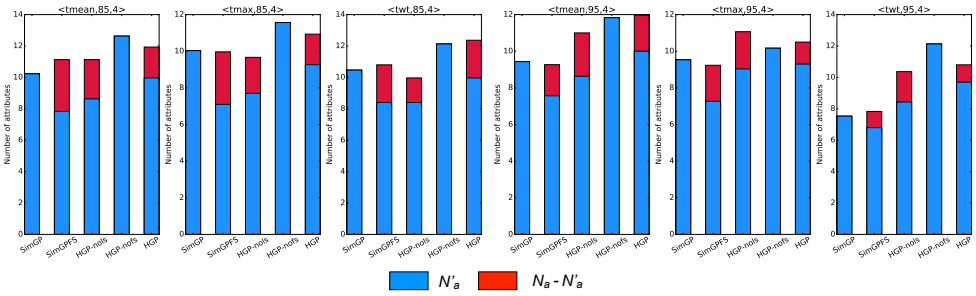


Figure 14: Attribute selection in SimGP, SimGPFS, HGP-*nols*, HGP-*nofs*, and HGP.

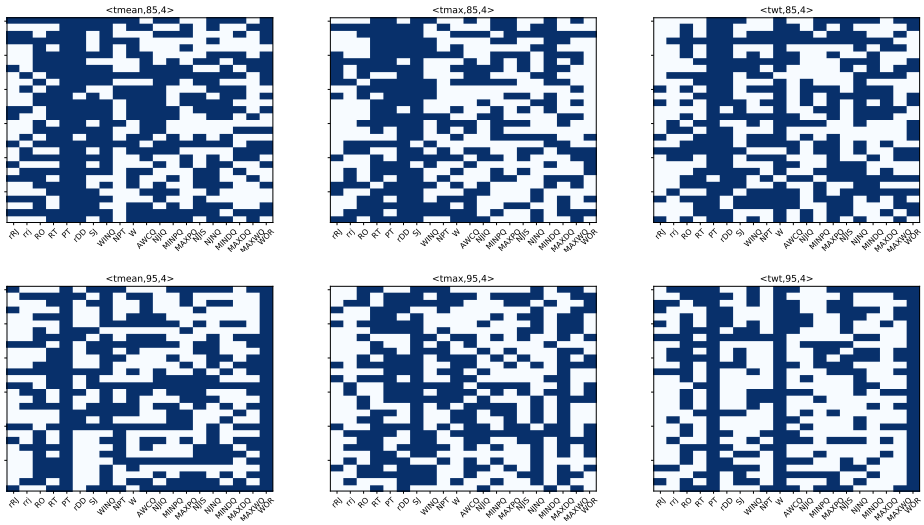


Figure 15: Selected attributes from SimGP across 30 independent runs.

ent places in the evolved rules to explore effective rules. Meanwhile, ILS in HGP tends to maintain the main structure of the rules and combine new genetic materials with the restricted subtree mutation (possibly with new attributes) while performing local search.

In Figure 15 and Figure 16, we visualise the active attributes obtained by SimGP and HGP-*nols*. HGP has a similar pattern with HGP-*nols*, and thus is not be presented here. The row in each subplot represents active attributes for a particular run of SimGP and HGP-*nols*. Each active attribute is represented by a blue cell; otherwise, the corresponding cell is left empty. Ideally, we want the columns corresponding to irrelevant attributes to be entirely empty and the columns corresponding to important attributes to be filled.

For minimising mean tardiness, `PT`, `RT`, `RO`, and `rDD` are the most frequently included attributes in the evolved rules, especially when the utilisation is 85%. More global attributes such as `NJIS`,`NJNQ`, and `WOR` are also usually included in the evolved rules. When the utilisation is 95%, only `PT`, `RO`, `rDD`, and `WOR` are included in most rules. Scenarios $\langle twt, 85, 4\rangle$ and $\langle twt, 95, 4\rangle$ also show similar patterns and additional emphasis on `W`. For $\langle tmax, 85, 4\rangle$ and $\langle tmax, 95, 4\rangle$, only due-date related attributes such as `rDD` and `SJ`, and `RT` are clearly needed for all evolved rules. In $\langle tmax, 95, 4\rangle$, more

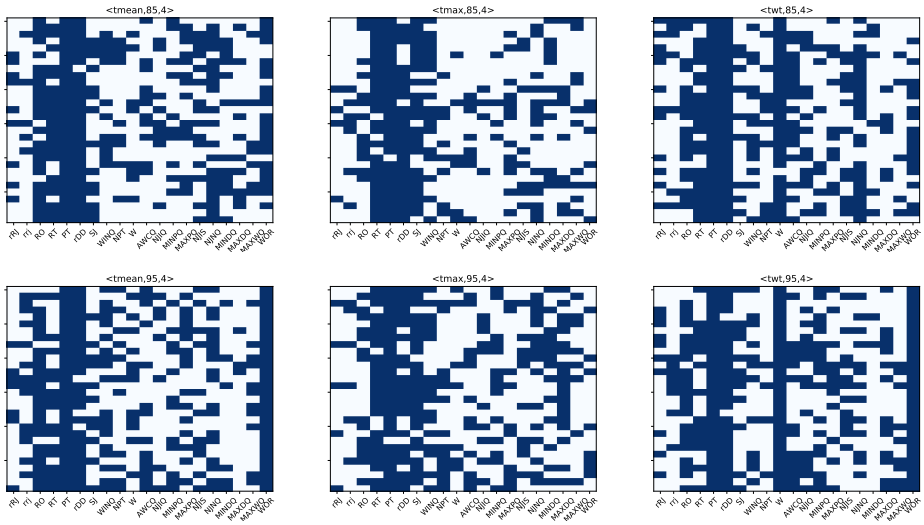


Figure 16: Selected attributes from HGP-*nols* across 30 independent runs.

attributes are needed but there is no clear pattern. In general, the patterns in HGP-*nols* are clearer than those in SimGP. For $\langle tmean, 85, 4\rangle$ and $\langle tmean, 95, 4\rangle$, while active attributes in HGP-*nols* are quite consistent, it is not the case with SimGP. For example, in $\langle tmean, 85, 4\rangle$ and $\langle tmean, 95, 4\rangle$, it seems that SimGP have troubles identifying important attributes as the selected attributes are relatively random.

Because of the complexity of DJSS, it is not necessary to have a unique good rule for a particular scenario. In fact, Nguyen et al. (2016) has shown that rules with very different behaviours (ways to assign priorities) can results in similar test performance. Thus, it is possible to have multiple *"optimal"* attribute subsets. As HGP performs online attribute selection, the attribute subset can be different in different independent runs, which explains why the patterns in Figure 15 and Figure 16 are not perfectly clear. While the number of selected attributes in the two methods are similar, HGP-*nols* can identify key attributes more effectively than SimGP.

6.4 An Example evolved rule

Figure 17 presents an example rule evolved by HGP for $\langle twt, 95, 4\rangle$. In the raw form, the rule is complicated, and there is no clear pattern to explain how it can achieve good results. To analyse the rule, we first remove some redundant components by (1) replacing attributes with $avec_i = 0$ by a constant "1", and (2) performing numerical simplification technique proposed by Nguyen et al. (2016). The simplification algorithm will traverse through the tree and replaces each subtree with a constant "1". If the modified rule is not significantly worse than the original rule (using Wilcoxon rank sum test), it will replace the original rule. For $\min, \max$ functions in the rule, each argument will, in turn, replace the root $\min, \max$ node and the modified rule will replace the original rule if it is not significantly worse than the original rule. The simplification continues until no new rule is found. As shown in Figure 17, the simplified rule is much smaller than the original rule. The next step is to rearrange the expression manually. We can see that the first component of the rule is a composite rule which combines weighted shortest processing time (WSPT), minimum slack, min-

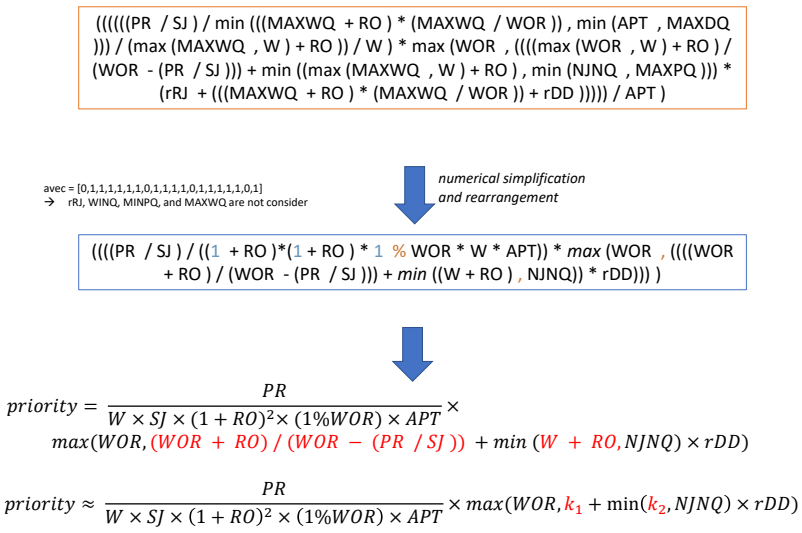


Figure 17: Selected attributes from HGP-*nols* across 30 independent runs.

imum remaining operations, and minimum workload of machines on the route (to be visited). The second term is much more complicated and contains some nonsense combinations such as: $(WOR + RO)/(WOR - PR/SJ)$ and $W + RO$. However, these components are necessary as they cannot be eliminated through the simplification algorithm. Our assumption is that they are included in the rules to create some adaptive behaviours for the evolved rules, which are usually critical for superior rules. The component $W + RO$ is used in $\min(W + RO, NJNQ)$ to make the rule adapt its behaviour depending on $NJNQ$ (number of jobs in the next queue). So $W + RO$ plays the role of a threshold. Actually when we replace $W + RO$ with their expected value $\mathbf{E}[W + RO] = \mathbf{E}[W] + \mathbf{E}[RO] = (0.2 + 0.6 + 0.2) + (2 + 14)/2 = 10.2$ (see the simulation settings in Section 3), the modified rule provides similar results. Interpreting $(WOR + RO)/(WOR - PR/SJ)$ is not so straightforward, but it also adapts the rule based on two dynamic attributes WOR and SJ. In general, we can use k_1 and k_2 as the two adaptive parameters for the rule. In this final form, we can see that this is a clever composite rule what combines many attributes to make scheduling decisions. Similar to ATC or COVERT (Pinedo, 2008), some extra experiments and analyses are needed to understand how the parameters can be set.

7 Conclusions

This paper proposed a new hybrid genetic programming for automated design of dispatching rules for dynamic job shop scheduling. In this method, a new representation for dispatching rules is developed to construct effective rules and select relevant attributes simultaneously. An efficient iterated local search heuristic employing a multi-fidelity surrogate modeling approach is also developed to search for competitive rules in the large heuristic search space. The experimental results have confirmed that the proposed method can evolve rules that are much better than those obtained with the simple GP methods. In some simulated scenarios, the proposed method can find rules with the same quality as the best rules evolved by the simple GP method within the

first few generations. Moreover, the hybrid method is also useful regarding controlling the length of rules and selecting relevant attributes and the proposed method performs well with more complex job shop conditions. These characteristics make the proposed method a powerful and attractive approach for automated design of dispatching rules.

The hybrid genetic programming framework proposed in this paper can be extended to other production scheduling as well as combinatorial optimisation problems. To successfully extend this approach to deal with a wide range of problems, it is important to automate the whole selection process for fitness evaluators (including surrogate models) to support the genetic operators and the local search heuristic. This can be done in many ways such as coevolving the fitness evaluator in the evolutionary process of genetic programming and applying machine learning method to learn efficient and accurate fitness evaluators based on historical searching data. These will be potential research directions in the future studies.

Attribute selection is an important task in automated design of dispatching rules and it is particularly critical when dealing with complex production systems and the availability of a large number of attributes from jobs and machines. The proposed method in this study tries to perform explicit attribute selection by developing a new representation for dispatching rules. In future research, it is important to further investigate this representation, especially in terms of rule initialisation, search operators, and related parameters. An adaptive way to evaluate programs with different attribute vectors is also needed to improve the performance of GP. For example, automatically-defined functions can be used to replace the irrelevant attributes decided by the attribute vector. Also, our future studies will focus more on the feature selection mechanism to understand when it is useful and how to make it more efficient. Different benchmark scheduling problems (with more attributes) will be needed to justify the importance of attribute selection.

Iterated local search proposed in this study has successfully refined rules evolved by genetic programming. Although it may increase the computational costs for evolving dispatching rules, its advantages outweigh the disadvantages. Different from other simple local search previously used to support genetic programming, the proposed iterated local search has good exploration and exploitation ability; therefore, it is not easily be trapped at the local optima. Future studies can further investigate the behaviour of this heuristic to reduce the number of required parameters and make it more adaptive.

References

Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156.

Aydin, M. E. and Fogarty, T. C. (2004). A simulated annealing algorithm for multi-agent systems: a job-shop scheduling application. *Journal of Intelligent Manufacturing*, 15(6):805–814.

Baker, K. R. and Trietsch, D. (2009). *Principles of Sequencing and Scheduling*. Wiley Publishing.

Bannat, E., Bautze, T., Beetz, M., Blume, J., Diepold, K., Ertelt, C., Geiger, F., Gmeiner, T., Gyger, T., Knoll, A., Lau, C., Lenz, C., Ostgathe, M., Reinhart, G., Roesel, W., Ruehr, T., Schuboe, A., Shea, K., Wersborg, I. S. G., Stork, S., Tekouo, W., Wallhoff, F., Wiesbeck, M., and Zaeh, M. F. (2011). Artificial cognition in production systems. *IEEE Trans. on Automation Science and Engineering*, pages 148–174.

Bierwirth, C. and Mattfeld, D. C. (1999). Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation*, 7(1):1–17.

Branke, J., Hildebrandt, T., and Scholz-Reiter, B. (2015). Hyper-heuristic evolution of dispatching rules: a comparison of rule representations. *Evolutionary Computation*, 23(2):249–277. in press (doi: 10.1162/EVCO_a_00131).

Branke, J., Nguyen, S., Pickardt, C. W., and Zhang, M. (2016). Automated Design of Production Scheduling Heuristics: A Review. *IEEE Transactions on Evolutionary Computation*, 20(1):110–124.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., zcan, E., and Woodward, J. R. (2010). A classification of hyper-heuristic approaches. In Gendreau, M. and Potvin, J.-Y., editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, pages 449–468. Springer, New York, 2nd edition.

Burke, E. K., Hyde, M., Kendall, G., and Woodward, J. (2007). Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In Lipson, H. and Thierens, D., editors, *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1559–1565, New York. ACM Press.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Ozcan, E., and Woodward, J. R. (2009). Exploring Hyper-heuristic Methodologies with Genetic Programming. In Mumford, C. and Jain, L., editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer Berlin Heidelberg.

Cheng, R., Gen, M., and Tsujimura, Y. (1999). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers and Industrial Engineering*, 36(2):343–364.

Dimopoulos, C. and Zalzala, A. M. S. (2001). Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, 32(6):489–498.

Durasevic, M., Jakobovic, D., and Knezevic, K. (2016). Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, 48:419–430.

Eguchi, T., Oba, F., and Toyooka, S. (2008). A robust scheduling rule using a neural network in dynamically changing job-shop environments. *International Journal of Manufacturing Technology and Management*, 14(34):266–288.

Geiger, C. D., Uzsoy, R., and Aytu, H. (2006). Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *Journal of Scheduling*, 9(1):7–34.

Goncalves, J. F., de Magalhaes Mendes, J. J., and Resende, M. G. C. (2005). A hybrid genetic algorithm for the job shop scheduling problem. *European Journal of Operational Research*, 167(1):77–95.

Hart, E. and Sim, K. (2016). A hyper-heuristic ensemble method for static job-shop scheduling. *Evolutionary Computation*, 24(4):609–635.

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

Hildebrandt, T., Heger, J., and Scholz-Reiter, B. (2010). Towards improved dispatching rules for complex shop floor scenarios a genetic programming approach. In Pelikan, M. and Branke, J., editors, *GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, pages 257–264, Portland, Oregon, USA. ACM Press.

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

Hunt, R., Johnston, M., and Zhang, M. (2014). Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In Igel, C. and Arnold, D. V., editors, *GECCO '14: Proceedings of the 2014 Conference on Genetic and Evolutionary Computation*, pages 927–934, New York. ACM Press.

Hunt, R., Johnston, M., and Zhang, M. (2015). Evolving dispatching rules with greater understandability for dynamic job shop scheduling. Technical Report ECSTR15-06, Victoria University of Wellington.

Ingimundardottir, H. and Runarsson, T. P. (2011). Supervised learning linear priority dispatch rules for job-shop scheduling. In Coello Coello, C. A., editor, *Learning and Intelligent Optimization*, volume 6683 of *LNCS*, pages 263–277, Berlin and Heidelberg. Springer.

Jakobovic, D. and Budin, L. (2006). Dynamic scheduling with genetic programming. In Collet, P., Tomassini, M., Ebner, M., Gustafson, S., and Ekrt, A., editors, *Genetic Programming*, volume 3905 of *LNCS*, pages 73–84, Berlin and Heidelberg. Springer.

Jakobovic, D., Jelenkovic, L., and Budin, L. (2007). Genetic programming heuristics for multiple machine scheduling. In Ebner, M., O'Neill, M., Ekrt, A., Vanneschi, L., and Esparcia-Alczar, A. I., editors, *Genetic Programming*, volume 4445 of *LNCS*, pages 321–330, Berlin and Heidelberg. Springer.

Jones, A. and Rabelo, L. C. (1998). Survey of job shop scheduling techniques. Technical report, NISTIR, National Institute of Standards and Technology, Gaithersburg, USA.

Kacem, I., Hammadi, S., and Borne, P. (2002). Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and Computers in Simulation*, 60(3-5):245–276.

Karunakaran, D., Chen, G., and Zhang, M. (2016). Parallel Multi-objective Job Shop Scheduling Using Genetic Programming. In Ray, T., Sarker, R., and Li, X., editors, *Artificial Life and Computational Intelligence*, Lecture Notes in Computer Science, pages 234–245. Springer International Publishing. DOI: 10.1007/978-3-319-28270-1_20.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA.

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

Land, M. J. (2004). *Workload Control in Job Shops, Grasping the Tap*. PhD thesis, University of Groningen, The Netherlands.

Law, A. M. and Kelton, D. M. (1999). *Simulation Modeling and Analysis*. McGraw-Hill Higher Education, 3rd edition.

Li, X. and Olafsson, S. (2005). Discovering dispatching rules using data mining. *Journal of Scheduling*, 8(6):515–527.

Mei, Y., Nguyen, S., Xue, B., and Zhang, M. (2017). An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(5):339–353.

Mei, Y., Zhang, M., and Nguyen, S. (2016). Feature selection in evolving job shop dispatching rules with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 365–372.

Miyashita, K. (2000). Job-shop scheduling with genetic programming. In Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *GECCO 2000: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 505–512, San Francisco. Morgan Kaufmann.

Nguyen, S., Mei, Y., and Zhang, M. (2017). Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, 3(1):41–66.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. (2013a). Learning iterative dispatching rules for job shop scheduling with genetic programming. *The International Journal of Advanced Manufacturing Technology*, 67(14):85–100.

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. (2015). Automatic Programming via Iterated Local Search for Dynamic Job Shop Scheduling. *IEEE Transactions on Cybernetics*, 45(1):1–14.

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

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2013c). Dynamic Multi-objective Job Shop Scheduling: A Genetic Programming Approach. In Uyar, A. S., Ozcan, E., and Urquhart, N., editors, *Automated Scheduling and Planning*, number 505 in Studies in Computational Intelligence, pages 251–282. Springer Berlin Heidelberg. DOI: 10.1007/978-3-642-39304-4_10.

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

Nguyen, S., Zhang, M., Johnston, M., and Tan, K. C. (2014b). Selection schemes in surrogate-assisted genetic programming for job shop scheduling. In *Simulated Evolution and Learning - 10th International Conference, SEAL 2014, Dunedin, New Zealand, December 15-18, 2014. Proceedings*, pages 656–667.

Nguyen, S., Zhang, M., and Tan, K. C. (2016). Surrogate-Assisted Genetic Programming With Simplified Models for Automated Design of Dispatching Rules. *IEEE Transactions on Cybernetics*, pages 1–15.

Nie, L., Gao, L., Li, P., and Li, X. (2013). A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. *Journal of Intelligent Manufacturing*, 24(4):763–774.

Nie, L., Shao, X., Gao, L., and Li, W. (2010). Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 50(58):729–747.

Nowicki, E. and Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6):797–813.

Olafsson, S. and Li, X. (2010). Learning effective new single machine dispatching rules from optimal scheduling data. *International Journal of Production Economics*, 128(1):118–126.

Ouelhadj, D. and Petrovic, S. (2008). A survey of dynamic scheduling in manufacturing systems. *Journal of Scheduling*, 12(4):417.

Park, J., Mei, Y., Nguyen, S., Chen, G., Johnston, M., and Zhang, M. (2016). Genetic programming based hyper-heuristics for dynamic job shop scheduling: Cooperative coevolutionary approaches. In Heywood, I. M., McDermott, J., Castelli, M., Costa, E., and Sim, K., editors, *Genetic Programming: 19th European Conference, EuroGP 2016, Porto, Portugal, March 30 - April 1, 2016, Proceedings*, pages 115–132, Cham. Springer International Publishing.

Pinedo, M. and Singer, M. (1999). A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 46(1):1–17.

Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer, New York, 3rd edition.

Sels, V., Gheysen, N., and Vanhoucke, M. (2011). A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. *International Journal of Production Research*, 50(15):4255–4270.

Sha, D. and Hsu, C.-Y. (2006). A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, 51(4):791 – 808.

Shiue, Y.-R. (2009). Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach. *International Journal of Production Research*, 47(13):3669–3690.

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

Vazquez-Rodriguez, J. A. and Ochoa, G. (2011). On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. *Journal of the Operational Research Society*, 62(2):381–396.

Weckman, G. R., Ganduri, C. V., and Koonce, D. A. (2008). A neural network job-shop scheduler. *Journal of Intelligent Manufacturing*, 19(2):191–201.

Wiendahl, H.-H., Cieminski, G. V., and Wiendahl, H.-P. (2005). Stumbling blocks of ppc: Towards the holistic configuration of ppc systems. *Production Planning & Control*, 16(7):634–651.

Xia, W. and Wu, Z. (2005). An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Computers & Industrial Engineering*, 48(2):409–425.

Yamada, T. and Nakano, R. (1995). A genetic algorithm with multi-step crossover for job-shop scheduling problems. In *GALESIA: First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications*, pages 146–151.

Yin, W.-J., Liu, M., and Wu, C. (2003). Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In Sarker, R., Reynolds, R., Abbass, H., Tan, K. C., McKay, B., Essam, D., and Gedeon, T., editors, *The 2003 Congress on Evolutionary Computation (CEC 2003)*, volume 2, pages 1050–1055, Piscataway, NJ. IEEE Press.

Zhou, H., Cheung, W., and Leung, L. C. (2009). Minimizing weighted tardiness of job-shop scheduling using a hybrid genetic algorithm. *European Journal of Operational Research*, 194(3):637–649.