Genetic Programming for Job Shop Scheduling

Su Nguyen[1], Mengjie Zhang[2], Mark Johnston[3], and Kay Chen Tan[4]

[1]La Trobe University, Australia
[2]Victoria University of Wellington, New Zealand
[3]University of Worcester, United Kingdom
[4]City University of Hong Kong, Hong Kong
p.nguyen4@latrobe.edu.au,mengjie.zhang@ecs.vuw.ac.nz
m.johnston@worc.ac.uk,kaytan@cityu.edu.hk

Abstract. Designing effective scheduling rules or heuristics for a manufacturing system such as job shops is not a trivial task. In the early stage, scheduling experts rely on their experiences to develop dispatching rules and further improve them through trials-and-errors, sometimes with the help of computer simulations. In recent years, automated design approaches have been applied to develop effective dispatching rules for job shop scheduling (JSS). Genetic programming (GP) is currently the most popular approach for this task. The goal of this chapter is to summarise existing studies in this field to provide an overall picture to interested researchers. Then, we demonstrate some recent ideas to enhance the effectiveness of GP for JSS and discuss interesting research topics for future studies.

Keywords: genetic programming, job shop scheduling, heuristic

1 Introduction

Scheduling deals with assigning manufacturing resources to process tasks over time. It has a big impact on the cost of operating a manufacturing system, as well as on other performance measures such as on-time delivery. Good scheduling is therefore an important factor for a company to be competitive. Because of their complexity, many scheduling problems are considered NP-hard [49], and thus exact methods are usually unable to solve them within a reasonable computational time. Job shop scheduling (JSS) is a good example of such problems in the literature and many heuristics such as dispatching rules [25, 26, 49], tabu search [45], genetic algorithm [8], ant colony optimisation [57], and particle swarm optimisation [54], have been proposed to find quick and acceptable solutions for JSS. Unfortunately, these heuristics are normally problem specific while designing an effective scheduling heuristics is a time-consuming and complicated task.

Dispatching rules is one of the most popular forms of scheduling heuristics that have been investigated since the earliest research on JSS. Basically, dispatching rules aim to prioritise jobs waiting in the shop or in front of a specific machine which then processes jobs based on their assigned priorities (usually jobs with the highest priorities are preferred). Due to their simplicity and ability to react quickly to dynamic changes, dispatching rules have received a lot

of attentions from both researchers and practitioners. However, similar to other scheduling heuristics for JSS, designing effective dispatching rules for a particular environment is not a trivial task and involves a lot trials and errors. To handle this issue, different automated approaches have been proposed in the last decade to facilitate the design process. The core of these approaches is machine learning [22, 11] and/or optimisation techniques [18, 6, 24, 37] which aim to learn and search for effective and robust dispatching rules.

Genetic programming (GP) [29, 3] is an evolutionary computation (EC) approach usually used to evolve computer programs for solving a specific task. In recent years, GP has been successfully applied to automatically generate dispatching rules for different scheduling problems. Many different variants of GP such as tree-based GP [29], grammar-based GP [3], and gene expression programming (GEP) [13] are capable of evolving rules that outperform most rules manually designed by human experts in the literature. Some reasons that make GP particularly suitable for these designing tasks are: (1) dispatching rules are usually priority functions [25, 49] which can be easily represented as a mathematical expression by GP trees/programs; (2) GP covers a large (heuristic or program) search space which enables us to discover unknown and effective dispatching rules; (3) representations of GP individuals are generally flexible, which allows sophisticated rules to be encoded and evolved; and (4) GP can take advantages of available EC search mechanisms to enhance the quality of obtained dispatching rules.

Different GP methods have been proposed to deal with JSS, in both static and dynamic environments. The research topics in previous studies are quite diverse and focus on different aspects of GP and JSS, ranging from representations of dispatching rules [43, 6, 37], learning/searching mechanisms [18, 48, 40], fitness evaluations [37, 19], attributes/features analyses [6, 37, 21], and interpretability [18, 6, 37, 21]. The goals of these studies are mainly to: (1) evolve competitive and robust dispatching rules, (2) improve the efficiency of GP for evolving dispatching rules, and (3) satisfy practical requirements in complex environments. Many promising results have been reported in these studies which make GP for JSS an interesting research topic for both operations research and automated heuristic design (hyper-heuristic [7]) communities.

In this chapter, we provide an overview of the current research on GP and JSS in order to help the readers who are interested in this field understand the key aspects, related applications, challenges and opportunities for future studies. The next section will give a brief description of JSS. Section 3 reviews the literature on GP and JSS. In Section 4, we revisit some useful ideas that have been proposed to enhance the quality of evolved dispatching rules. Further discussions and conclusions are provided in Section 5.

2 Background

The general JSS problem could be simply defined as the scheduling of different jobs to be processed on different machines to satisfy certain objectives. In this

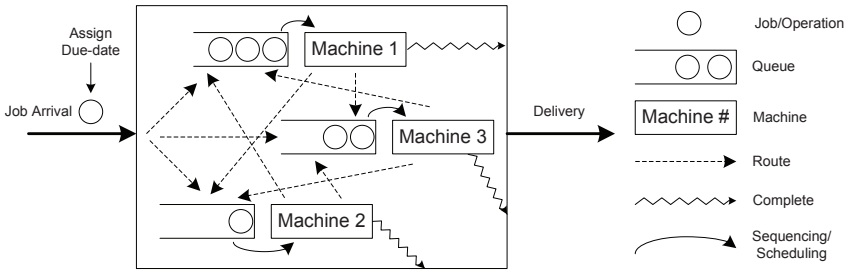

Fig. 1. Job Shop Scheduling (shop with 3 machines).

case, a job is a sequence of operations, each of which is to be performed on a particular machine. In JSS, the routes of jobs are fixed, but not necessarily the same for each job [49]. An example of a job shop production system is shown in Fig 1. For the static JSS problem, the shop (or the working/manufacturing environment) includes a set of m machines and n jobs that need to be scheduled. Each job has its own pre-determined route through a sequence of machines to follow and its own processing time at each machine it visits. In static JSS, processing information of all jobs is available. In the dynamic JSS problem, jobs arrive randomly over time and the processing information of jobs is unknown before their arrival.

Over the last few decades, a large number of methods have been developed and applied to JSS, ranging from simple heuristics to artificial intelligence and mathematical optimisation methods. Dispatching rules are perhaps the most straightforward method to deal with both static and dynamic JSS problems [53, 25]. Meanwhile, optimisation is the main research stream to deal with the static JSS problems [49]. A review of these methods is presented in this section. For a broader review of scheduling methods, the readers are encouraged to read Ouelhadj and Petrovic [46] and Potts and Strusevich [52].

2.1 Dispatching rules

Although there have been many breakthroughs in the developments of exact and approximate methods for JSS; these methods are mainly focused on static problems and simplified job shop environments. General methods like genetic algorithm (GA) can be extended to solve problems with realistic constraints, but the major drawback is its weak computational efficiency. Moreover, as pointed out in [34], the conventional operations research and artificial intelligence methods are often not applicable to the dynamic characteristics of the actual situation because these methods 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 dynamic changes of the shop.

There have been a large number of rules proposed in the literature and they can be classified into three categories: (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) [25, 26, 49] can also be considered a version of rules based on weighted priority indices, where scheduling information can be combined in more sophisticated ways instead of linear combinations. Pinedo [49] also showed various ways to classify dispatching rules based on the characteristics of these rules. The dispatching rules in this case can be classified as *static* and *dynamic* rules, where dynamic rules are time dependent (e.g. minimum slack) and static rules are not time dependent (e.g. shortest processing time). Another way to categorise these rules is based on the information used by these rules (either local or global information) to make sequencing decisions. A *local* rule only uses the information available at the machine where the job is queued. A *global* rule, on the other hand, may use the information from other machines.

The comparisons of different dispatching rules have been continuously done in many studies [53, 20, 18]. The comparison was usually performed under different characteristics of the shop because it is well-known that the characteristics of the shop can significantly influence the performance of the dispatching rules. Different objective measures were also considered in these studies because they are the natural requirements in real world applications. Although many dispatching rules have been proposed, it is still a challenge for scheduling researchers to develop rules that can perform well on multiple objectives.

2.2 Meta-heuristic methods

Since the static JSS is a NP-hard problem [14], finding optimal solutions by mathematical programming methods can be very time-consuming even for reasonable small instances. The research on meta-heuristics for scheduling has been very active in the last two decades, mostly with makespan as the objective. Local search based methods such as simulated annealing [32], large step optimisation [33], tabu search [45], and guided local search [2] have shown very promising results. The focus of these methods is on the development of efficient neighbourhood structures (mainly based on the concept of critical paths and critical blocks) and diversifying strategies to escape from local optima. Since the neighbourhood structures play an important role in these methods, they and their related operators have to be redesigned in order to incorporate real world constraints; even then it is still questionable whether they produce good results.

A more general alternative for solving JSS problems is the use of evolutionary computation methods. GA is one of the most popular methods in this line of research (refer to [8] for a review of GA methods for JSS). More recently, many hybrid algorithms have been proposed to combine the advantages of GA and local search heuristics. Yamada and Nakano [58] presented a GA with multi-step crossover (MSX) for JSS. In this method, MSX was used in combination with a local search heuristic. The preliminary experiments using benchmark instances showed promising performance of the proposed approach. Goncalves

et al. [17] proposed a hybrid GA method for JSS to minimise makespan. In this method, the chromosome representation is based on random keys and represents the priorities of operations. An active/non-delay parameter is also applied to restrict the delay time of operations. During the GA search, the schedule is further improved by the neighbourhood local search procedure from [45].

Swarm intelligence methods [4, 5] have also been applied to JSS problems and show very promising results. Sha and Hsu [54] developed a hybrid PSO algorithm (HPSO) that modified the particle position based on preference list-based representation and employed Giffler and Thompson algorithm [16] to decode particle positions into schedules. Moreover, tabu search is also applied to further improve the solution quality. The experimental results showed that HPSO is competitive compared to other meta-heuristics proposed in the literature. Xing et al. [57] proposed a sophisticated ant colony optimisation method in which a knowledge model is used to learn some available knowledge from the optimisation of ACO. The existing knowledge will be used to guide the current heuristic searching. The proposed knowledge-based ant colony optimisation (KBACO) algorithm outperformed some current approaches.

Research on other objectives 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. Pinedo and Singer [50] presented a heuristic to minimise the total weighted tardiness in JSS, which was based on the shifting bottleneck procedure [49] that schedules one machine at a time and used a branching tree to find a good order to schedule the machines. Every node of the tree represents a partial order in which the machines are scheduled. From the experiments, this method yielded solutions that were near optimum on some problems with 10 jobs and 10 machines. Asano and Ohta [1] introduced another heuristic for the minimisation of the total weighted tardiness in JSS that is based on the tree search procedure, also with very promising results. Kreipl [30] proposed an efficient large step random walk (LSRW) method for minimising total weighted tardiness. This method employed different neighbourhood sizes to perform a small step or a large step. The small step consists of iterative improvement while the large step consists of a Metropolis algorithm (similar to the simulated annealing algorithm but with a constant temperature). Essafi et al. [12] proposed a hybrid genetic algorithm which employed an iterative local search and a hybrid scheduling construction procedure to solve this problem. The results showed that the proposed method is very competitive as compared to LSRW [30]. Dispatching rules based meta-heuristics [9, 47, 51] have also been investigated in the literature in order to utilise effective dispatching rules to narrow down the solution search space.

3 Genetic programming for job shop scheduling

Recently, GP based hyper-heuristics (GPHHs) [7] has become increasingly popular because of their ability to evolve a wide range of program structures corresponding to different types of scheduling rules and heuristics. Another reason

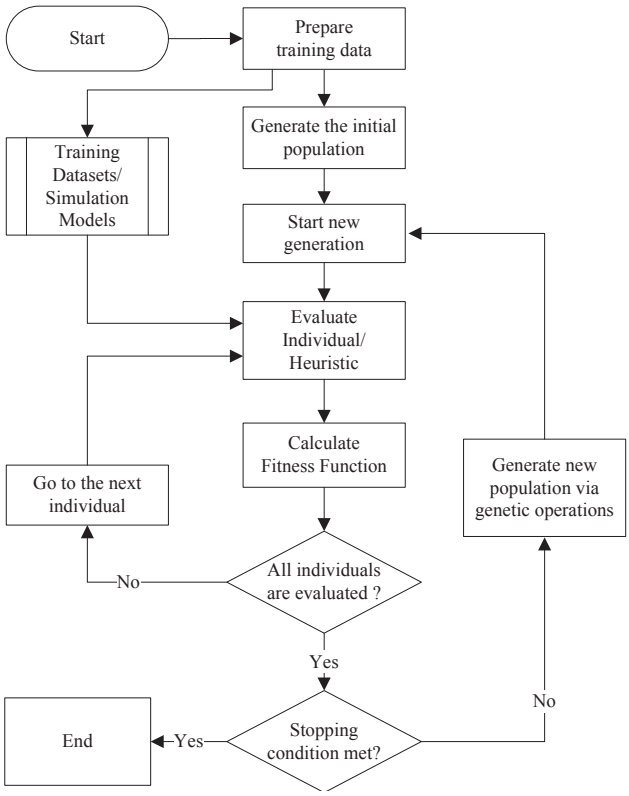

Fig. 2. Evolving dispatching rules with genetic programming.

that makes GPHHs suitable for this task is that the rules or heuristics evolved by GP can be (partially) interpretable. This is an important characteristic in order to apply obtained heuristics into practical applications. Fig. 2 presents the basic framework for evolving dispatching rules with GP. The procedure in this figure starts by preparing training data (datasets or simulation models) for the considered scheduling problems. The initial population of heuristics is then randomly generated (e.g., ramp-half-and-half [3]). In each generation of the evolutionary process, each heuristic is used to provide scheduling decisions for different instances/scenarios in the training datasets or simulation models. The obtained scheduling performance is used to calculate the fitness of each evolved rule. The fitness values obtained by rules in the population decide the chance of each rule to survive and reproduce (with genetic operations) in the next generation. The same routine is applied until the termination condition is met. In the rest of this section, we will review related studies on GP and JSS, which are categorised based on their representations and evolutionary search mechanisms.

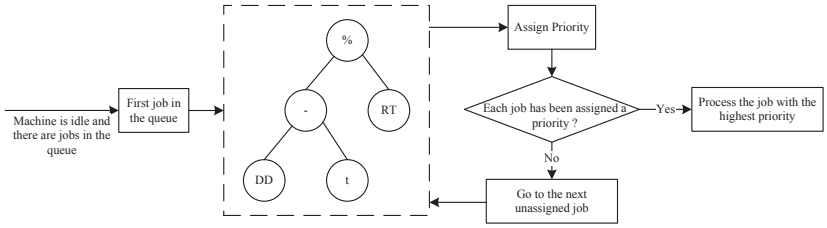

Fig. 3. Tree-based representation of the critical ratio rule $\frac{(\mathrm{DD}-\mathrm{t})}{\mathrm{RT}}$.

3.1 Representations of dispatching rules

Representations in GP are very important because they not only decide how evolved rules look like but they also determine how GP can evolve those rules. It is safe to say that most key aspects in GP are governed by the choice of representations. In this section, we will review popular representations employed in the literature and their corresponding genetic operators.

Tree-based representation The most popular representation of dispatching rules is the tree-based representation which is commonly used in the conventional GP, often referred to as tree-based GP (TGP) [29, 3]. This is easy to understand as the tree-based GP is the most established method in the GP literature and the tree-based representation can easily be used to represent any (existing) priority functions with appropriate choices of functions and terminals (attributes). An example dispatching rule in the tree structure is shown in Fig. 3. When converted into its mathematical form, it is the same as the critical ratio rule [49] where the terminals t, DD, RT are the decision moment (current time), the due date and remaining processing time of the considered job, respectively. The figure also describes how an evolved rule makes dispatching decisions. When a machine is idle and a new job arrives at the machine, this job will be processed immediately. In the case that a machine has just completed a job and there are still jobs waiting in the queue to be processed at that machine, the dispatching rule will be applied. To assign a priority to a waiting job, the information related to that job will be extracted to be used in the corresponding terminals of the rule. Then, the tree representing the dispatching rule will be evaluated and the output from this evaluation will be assigned to the considered job as its priority (refer to [29] for detailed discussion on how a tree program is evaluated). This procedure will be applied until priorities are assigned to all waiting jobs and the job with the highest priority will be processed next.

Due to the complexity of scheduling problems, a single tree may not be sufficient to generate effective and comprehensive scheduling heuristics. Therefore, more sophisticated tree-based representations have been developed. Geiger et al. [15] propose a multiple tree representation to evolve different dispatching rules (trees) for different machines or groups of machines. The goal of this ap-

proach is to generate specialised rules to cope with particular requirements of each machine.

In order to create more effective scheduling heuristics for job shops, Jakobovic and Budin [23] present the GP-3 method in which three program trees represent one discriminating function and two priority functions. A special terminal set is used to build the discriminating function which is employed to determine whether the considered machine is a bottleneck. Based on this decision, one of the two priority functions (for bottleneck and non-bottleneck machines) is applied to make scheduling decisions. Nguyen et al. [36] represent the scheduling heuristics by two program trees. The first one is the priority function (the same as previous studies) while the second one represents the look-ahead strategy based on the Giffler and Thompson algorithm [16] to decide how much time idle machines can delay before jobs can be processed. The experiments show that these extended representations can help GPHHs evolve significantly better scheduling heuristics as compared to GPHHs with the single tree representation.

For the tree-based representation, there are many genetic operators available in the GP literature [3]. Subtree crossover and subtree mutation are probably the most commonly used genetic operators in GP to explore new dispatching rules. 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. Depending on the considered scheduling problems and the structures of the evolved rules, some specialised genetic operators are also applied. For example, Geiger et al. [15] employ restricted crossover and mutation operators to generate dispatching rules for parallel machine scheduling problems. In their approach, only rules from the same machine are allowed to exchange genetic materials. Similarly, Yin et al. [59] also restrict their subtree crossover to be carried out only between the subtrees of similar functions (priority functions and idle-time estimation function).

GEP representation The linear representation of GEP has been applied to construct priority functions [42, 44, 43, 27], similar to those evolved by GP with the tree-based representation. GEP genes are also constructed based on the function set and the terminal set. In GEP, the priority function is represented as a chromosome of one or more genes. Each gene in the chromosome represents a fixed length symbolic string which represents a mathematical function. A gene can be divided into two parts: head and tail. While the head can contain both functions and terminals, the tail can only contain terminals. An example GEP chromosome with a single gene is shown in Fig. 4. The gene can be translated into an expression tree by using K-expression. In this example, the first element in the gene + is the root of the tree whose arguments can be obtained from the next elements in the gene. It is noted that the first five elements in the gene have already formed a valid K-expression and the rest of the gene will be ignored in this case. In order to ensure that a valid K-expression can be obtained, the

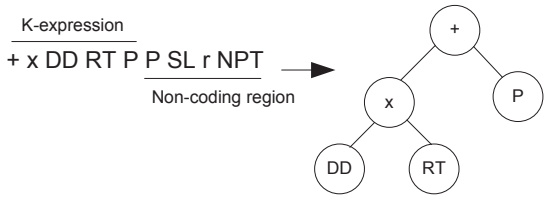

Fig. 4. GEP representation of the rule DD × RT + P .

length of the gene will be set such that $t = h(n-1)+1$, where h, t, and n are respectively the length of the head, the length of the tail, and the maximum number arguments of a function. In their experiments with the dynamic single machine scheduling problems, Nie et al. [43] show that GEP was very competitive with TGP (better than GP in some cases) and the rules obtained by TGP and GEP were better than all the benchmark heuristics.

Similar to the tree-based representation, the GEP representation can also be extended to cope with multiple scheduling decisions. Nie et al. [44, 42] develop GEP methods to deal with flexible job shop scheduling problems. In their methods, each GEP individual contains two chromosomes for making sequencing and routing decisions or a chromosome will contain two genes representing the two scheduling rules. The results show that the new GEP methods can evolve scheduling heuristics that outperform heuristics in the literature and the GEP method that deals with a single scheduling decision.

In order to evolve more sophisticated rules, multiple genes can also be used to represent multiple functions which can be combined by using a simple summation of these functions [43] or explicitly using a control gene to combine the outputs from these functions [44]. In the latter approach, the control gene is a dedicated gene which is used to characterise the relationship between outputs obtained from other genes. The control gene used the same function set as other genes and the terminal set consisting of outputs from other genes. While this representation can help GEP evolve more sophisticated rules, it will also increase the computational time as well as the search space of GEP.

The genetic operators in GEP can be considered as hybrids between those of genetic algorithm (GA) and TGP. The subtree crossover and subtree mutation from TGP can also be applied to GEP. However, because of the difference in data structure (linear vs tree), GEP needs to explicitly transverse through elements in a gene to identify the subtree. Because the length of a GEP gene is fixed, the same genetic operators such as the point mutation and the one-point/two-point crossover in GA can also be applied [44, 42]. Special transposition operators are also employed in GEP to randomly select a fragment of the chromosome and insert it into the head.

Grammar-based representation Different from the tree-based representations which mainly focus on evolving priority functions (dispatching rules),

grammar-based representations are usually used to construct high-level heuristics composed of several low-level heuristics and solution attributes. Although grammar-based GP has been developed to evolve heuristics for many hard combinatorial problems , their applications in manufacturing scheduling are still very limited. Nguyen et al. [37] develop a grammar-based representation for GP to evolve adaptive dispatching rules for job shop scheduling. The heuristics evolved with this representation is quite similar to decisions trees which try to find out which (available) candidate dispatching rule should be applied and what non-delay factor should be used given some specific machine/system status. The advantage of this representation is that the obtained rules can be interpreted easier as compared to evolved priority functions previously discussed. Also, by using a set of candidate rules which have been readily coded, the evaluations of these rules are faster than those of rules with the tree-based representation. On the other hand, the disadvantage of this representation is that it depends a lot on the available problem-domain knowledge to choose appropriate machine/system attributes and candidate rules. If the candidate rules cannot cover all situations, the evolved adaptive rules may not provide satisfactory results.

3.2 Search mechanisms

The traditional search mechanism in Fig. 2 is currently the most common technique to deal with scheduling problems. Regardless of its simplicity, this framework is able to discover very effective scheduling heuristics. However, in order to deal with more complicated design issues such as multiple scheduling decisions and multiple objectives, specialised search mechanisms will be needed.

Evolutionary multi-objective optimisation Multiple conflicting objectives are a natural characteristic in real world applications and the design of new scheduling heuristics also need to consider this issue. One advantage of using GPHHs for designing heuristics is that their search mechanisms are very flexible and many advanced techniques have been developed to cope with multiple objectives.

Tay and Ho [55] aim to tackle three objectives (makespan, mean tardiness, and mean flowtime) when using GP to evolve dispatching rules for a flexible job shop. In order to simplify the design problem, the three objectives are aggregated by using the weighted sum approach with the same weight for each objective. However, because the scale of each objective as well as the knowledge about the objective search space is unknown, this approach can lead to unsatisfactory results. For that reason, the rules evolved by their GP method are sometimes worse than simple rules such as FIFO [55]. When these evolved rules are examined in a long simulation [18], they are only slightly better than the earliest release date (ERD) rule and worse than the shortest processing time (SPT) rule with respect to mean tardiness. This suggests that using the weighted aggregated objective to deal with multi-objective design problem is not a good approach if the prior knowledge about the objective functions is not available.

Nguyen et al. [38] develop a multi-objective genetic programming based hyper-heuristic (MO-GPHH) for dynamic job shop scheduling. In this work, they aim to evolve a set of non-dominated dispatching rules for five common objective functions in the literature. By relying on the Pareto dominance rather than any specific objective, the proposed MO-GPHH was able to evolve very competitive rules as compared to existing benchmark rules in the literature. Their results show that it is very easy for MO-GPHH to find rules that totally dominate simple rules such as FIFO and SPT regarding all five considered objectives. The proposed MO-GPHH can also find rules that dominate more sophisticated rules such as apparent tardiness cost (ATC), RR, 2PT+WINQ+NPT, and cost over time (COVERT) [25, 26, 53] in most of its runs. The analyses show that evolving the Pareto front is more beneficial as compared to evolving a single rule as many unknown and helpful trade-offs are be discovered. Similar methods have been applied to evolve comprehensive scheduling policies for dynamic job shop scheduling [40] and order acceptance and scheduling [39] and showed promising results.

Coevolution The advantages of automatic design of scheduling heuristics have been demonstrated in the previous study. However, the drawback of this approach is the high computational time. Even though the obtained heuristics are very fast, training hundreds or thousands of these heuristics under different training examples can be very time consuming. To cope with this issue, GP can evolve heuristics in parallel to reduce the computational times and hopefully the complexity of the problem.

Miyashita [35] proposes three multi-agent learning structures based on GP to evolve dispatching rules for JSS. The first one is a homogeneous agent model which is basically the same as other GP methods which evolves a single dispatching rule for all machines. The second model treated each machine (resource) as a unique agent which requires distinct heuristics to prioritise jobs in the queue. In this case, each agent has its own population to evolve heuristics with GP. Finally, this research proposed a mixed agent model in which resources are grouped based on their status. Two types of agents in this model are bottleneck agent and non-bottleneck agent. Because of the strong interactions between agents, credit assignment is difficult. Therefore, the performance of each agent is directly measured by the quality of the entire schedule. The experimental results show that the distinct model has better training performance compared to the homogeneous model. However, the distinct model has overfitting issues because of the too specialised rules (for single/local machines). The mixed agent model shows the best performance among the three when tested under two different shop conditions. The drawback of this model is that it depends on some prior-knowledge (i.e. bottleneck machines) of the job shop environment, which can be changed in dynamic situations.

To deal with multiple scheduling decisions (sequencing and due date assignment) in job shops, Nguyen et al. [40] develop a GP based cooperative coevolution approach in which scheduling rules for a scheduling decision are evolved

in their own subpopulation. Similar to [35], the fitness of each rule is measured by the overall performance obtained through cooperation. Specialised crossover, archiving and representation strategies are also developed in this study to evolve the Pareto front of non-dominated scheduling heuristics. The results show that the cooperative coevolution approach is very competitive with some other evolutionary multi-objective optimisation approaches. The analysis also indicates that the proposed cooperative coevolution approach can generate more diverse sets of non-dominated scheduling heuristics.

Multi-stage learning/optimising In order to further utilise the outputs of GPHHs for enhancing scheduling performance, some multi-stage learning/optimising approaches have been proposed. Kuczapski et al. [31] develop two hyper-heuristics to generate initial populations of GA for JSS. The first hyper-heuristic uses GP to evolve composite dispatching rules similar to previous studies [55, 18, 10]. The second one tries to find composite rules through simple weighted sum of priorities generated by some existing dispatching rules. Another GA is used in this GPHH to search for the weight of each existing dispatching rule. The comparison showed that rules generated by GA is better than those generated by GP. A reason for the poor performance of GP in this case may be that the population size (20) is too small and does not provide GP with enough genetic materials to synthesise effective rules. The experimental results showed that the two hyper-heuristics can find rules to generate good initial solutions for JSS and significantly enhance the performance of GA, particularly improving the quality of solutions up to 10% and reducing the computational time up 50% [31]. One drawback of this approach is that the proposed GPHHs have to be applied for each instance and reusability of evolved dispatching rules has not been investigated.

The multi-stage approach is not only applied to static scheduling problems but also to dynamic scheduling problems. Pickardt et al. [48] propose a two-stage approach to evolving dispatching rule sets for semiconductor manufacturing. In the first stage, GP is used to evolve general dispatching rules. The best obtained dispatching rule is combined with a list of benchmark dispatching rules to generate a set of candidate rules. In the second stage, a $\mu + \lambda$ evolutionary algorithm (EA) [48] is used to select the most suitable dispatching rule in the set of candidate rules for each work centre in the shop. The experiments in this paper compare the performance of the two-stage hyper-heuristics with the pure GP and EA hyper-heuristics. The results show that the three hyper-heuristics outperformed benchmark dispatching rules and the two-stage hyper-heuristics produced significantly better performance than the other two hyper-heuristics.

4 Performance enhancement revisited

The previous section has shown some successful applications of GP for JSS. Several clever ideas have been introduced in previous studies that can be used to enhance the performance of GP for JSS. Although many ideas are proposed for specific scheduling problems (e.g., flexible job shops with routing and sequencing

decisions), some ideas are quite generic and can be easily applied to enhance the quality of GP for JSS. In this section, we will revisit these generic ideas to demonstrate how they can be applied and their usefulness. All experiments are conducted using the same GP system and simulation environments. Different performance measures of JSS are examined to verify the general effectiveness of the considered ideas.

4.1 Experimental settings

All experiments in this section are based on the simulation model of a symmetrical job shop which has been used in previous studies on dispatching rules [25, 38, 6]. Here are the simulation configuration:

- 10 machines
- Each job has 2 to 14 operations (re-entry is allowed)
- Processing time follows discrete uniform distribution $U[1, 99]$
- Job arrivals follow Poisson process
- Due date = current time + allowance factor × total processing time (allowance factor of 4 is used in our experiments)
- Utilisation of the shop is 95%
- No machine break-down; preemption is not allowed
- Weights of jobs are assigned based on the $4 : 2 : 1$ rule [30, 50].

In each simulation replication, we start with an empty shop and the interval from the beginning of the simulation until the arrival of the 500^{th} job is considered as the warm-up time and the statistics from the next completed 2000 jobs [20] will be used to calculate performance measures. Three scheduling performance measures examined in our experiments are: (1) mean flowtime, (2) mean tardiness, (3) total weighted tardiness. Although this simulation model are 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 high utilisation (95%) 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. Table 1 shows the terminal set and function set used in our experiments. The parameters used in GP are presented in Table 2. The results for each GP method in this section are based on 30 independent runs.

4.2 Training simulation replications

Using discrete event simulation is a conventional and suitable approach to assess the performance of dispatching rules. In order to reliably measure the effectiveness of evolved rules, a large number of simulation replications are needed (e.g., 30 to 50 simulation replications are usually needed to accurately estimate the performance of rules in the scenario described in Section 4.1). However, using simulation to evaluate fitness of evolved rules is also the most time-consuming

Table 1. Terminal and function sets of GP

Symbol	Description
rJ	job release time (arrival time)
RJ	operation ready time
RO	number of remaining operation within the job.
RT	work remaining of the job
PR	operation processing time
DD	due date of the job
RM	machine ready time
SL	slack of the job = DD − (t + RT)
WT	is the current waiting time of the job = max(0, t − RJ)
#	Random number from 0 to 1
NPR	processing time of the next operation
WINQ	work in the next queue
APR	average operation processing time of jobs in the queue
Function set	+,−,×, %, min, max

*`t` is the time when the sequencing decision is made.

Table 2. Parameter settings

Parameter	Description
Initialisation	ramped-half-and-half
Crossover rate	80%
Mutation rate	15%
Reproduction rate	5%
Maximum depth	8
Number of generations	100
Population size	500
Selection	tournament selection (size = 5)

part in GP for JSS. Therefore, only a small number of replications are usually used for fitness evaluations during the evolution (training) process.

How to effectively use the computational budget, i.e. total number of simulation replications, is an interesting and important research question. Hildebrandt et al. [18] has investigated different trade-offs between numbers of training replications and maximum numbers of generations when trying to evolve dispatching rules for a semiconductor manufacturing systems. From their analyses, the best setting is to use only one replication to evaluate fitness of rule; however, different replications (different random seeds for the simulator) should be used in different generations. In their method, the best rule obtained from each generation is fully evaluated to validate its real performance. The experimental results of *fixed* simulation replication and *changing* simulation replication strategies for

our scenario are shown in Fig. 5 (for presentation purpose, the total weighted tardiness is normalised by dividing it by the number of jobs). The values in these boxplots are the average performance measures over 50 simulation replications.

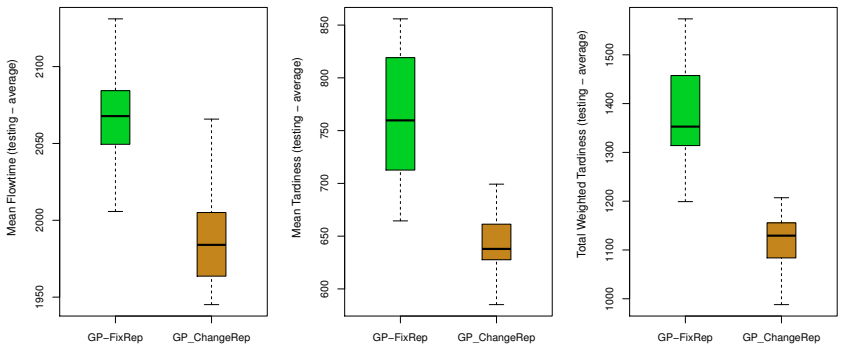

Fig. 5. *Fixed* simulation replications vs. *changing* simulation replications.

In this experiment, GP-FixRep uses only two fixed simulation replications during the evolution (thus, only 50 generations are performed for a fair comparison) and GP-ChangeRep changes the simulation replication when moving to the next generation). From the experiments, it is quite clear that changing simulation replications can help GP evolve more effective dispatching rules regardless of performance measures. As changing simulation replications can be easily implemented, using this strategy is a good way to enhance the quality of evolved rules when dealing with a single objective (i.e., performance measure), especially when the computational times are restricted or the simulation is computationally too expensive.

4.3 Smooth and nonsmooth evolved dispatching rules

Selecting an appropriate function set is important in GP as it decides the search space of evolved rules. Since most popular rules in the literature are relatively simple and can be easily constructed by using basic arithmetic operators ($+, -, \times, \%$), it is unclear if using more sophisticated functions such as $if, \max, \min$ is necessary. In this section, we examine GP with two different function sets (1) GP-Smooth where only basic arithmetic operators are used and (2) GP-Nonsmooth where arithmetic operators and $\max, \min$ are used. The comparison of the two settings are shown in Fig. 6.

In the case with mean flowtime as the performance measure, there is no significant difference between the two settings and GP-Smooth is even slightly better than GP-Nonsmooth. It is understandable because most effective existing rules in the literature for minimising flowtime are usually simple (usually linear combinations of different attributes) such as shortest processing time (SPT), or

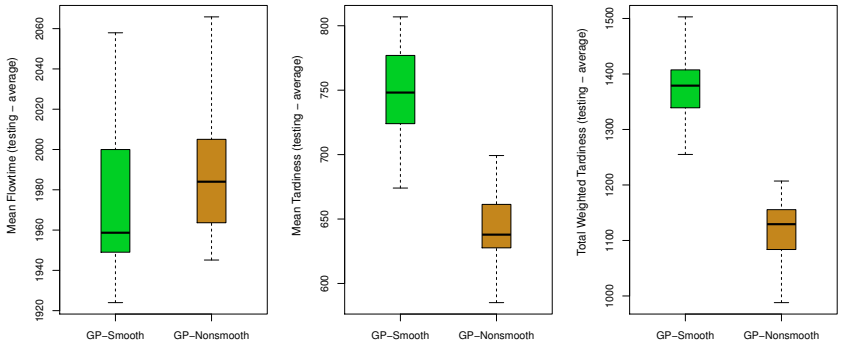

Fig. 6. *Smooth* vs. *nonsmooth* evolved dispatching rules.

2PT+WINQ+NPT [20]. Therefore, min, max seems to be redundant here and may deteriorate the performance of GP.

For minimising mean tardiness and total weighted tardiness, GP-Nonsmooth convincingly outperform GP-Smooth. These results suggest that basic arithmetic operators are not sufficient to construct rules to deal with complex scheduling problems (e.g., complex performance measures). The smooth evolved rules generated by GP-Smooth cannot possess complex behaviours to cope with different situations in the simulation. As we examine the best evolved rules from GP-Nonsmooth, it is easy to see that max, min functions create a complex behaviour for evolved rules which cannot be easily created using arithmetic operators. However, it is noted that we also try to avoid too complex functions such if or max, min with multiple arguments as they can significantly increase the search space without making any real contributions. From our past experiments, max, min along with basic arithmetic operators are reasonably sufficient to generate very complex (nonsmooth) rules. These issues also need to be taken into account when developing new representations for dispatching rules.

4.4 Surrogate model

Surrogate models have been employed in many EC applications, especially when the fitness evaluations are expensive. However, compared to traditional EC methods, using surrogate models in GP is a lot more complicated. Approximating the fitness of GP individuals is challenging because it is difficult to capture the behaviours of evolved programs, or dispatching rules in our applications.

Hildebrandt and Branke [19] investigated two surrogate models for evolving dispatching rules to minimise mean flowtime. In their approach, the fitness of an evolved rule is approximated by using the fitness of the most similar rules generated in the previous generations. Based on this idea, two similarity measures are proposed. The first measure is based on the genotype similarity of evolved rules, i.e. the similarity in their structure. The second measure is calculated based on the similarity of evolved rules, i.e the similarity in the way they prioritise jobs.

Their (simplified) proposed surrogate-assisted GP (SGP) can be summarised as follows:

i. Initialise the population with N rules
ii. Evaluate and determine (real) fitness for all evolved rules
iii. Apply genetic operator to generate $m \times N$ new rules
iv. Approximate the fitness of newly generated rules
v. Put rules with the best fitness into the population of the next generation
vi. Stop if the termination condition is met; otherwise, back to step (ii)

SGP in [19] used fixed simulation replications and utilised individuals in the last two generations to approximate fitness of newly generated rules. For the surrogate model, the behaviour of an evolved rule is characterised by a decision vector based on a reference rule (2PT+WINQ+NPT) and the similarity of two rules is measured by the distances of their corresponding decision vectors (see [19] for a detailed description). In our experiments, we further examine the performance of SGP with changing simulation replications and different reference rules. Different from [19], because fitness of rules in different generations is not compatible (as different replications are used), we only use rules in the most recent generation for fitness approximation. The results of our experiments are shown in Fig. 7.

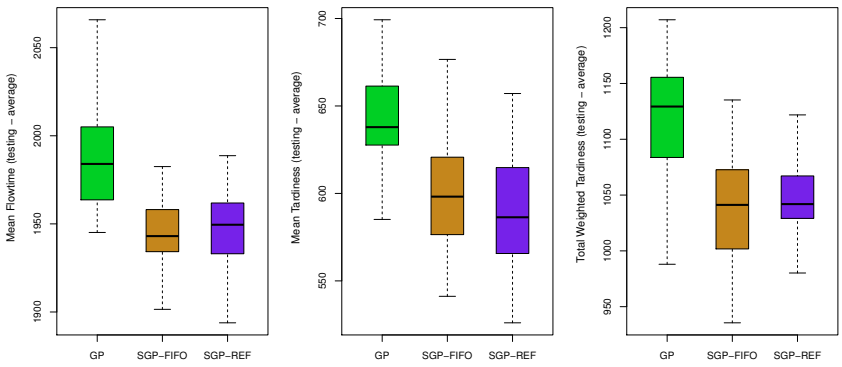

Fig. 7. Performance of *surrogate-assisted* GP methods for JSS.

In Fig. 7, GP represents simple GP method with changing replications and the extended function set. SGP-FIFO and SGP-REF are our SGP versions with FIFO and 2PT+WINQ+NPT as reference rules respectively. The results show that surrogate approaches are more effective as compared to the simple GP approach when the same computational budget is used. It is noted that SGP-FIFO and SGP-REF are slightly slower than GP because of fitness approximation. However, as the complexity of scheduling environments increases, the time for fitness approximation will be negligible as compared to the time for full fitness evaluations. More detailed analyses of SGP are presented in Fig. 8 to Fig. 10.

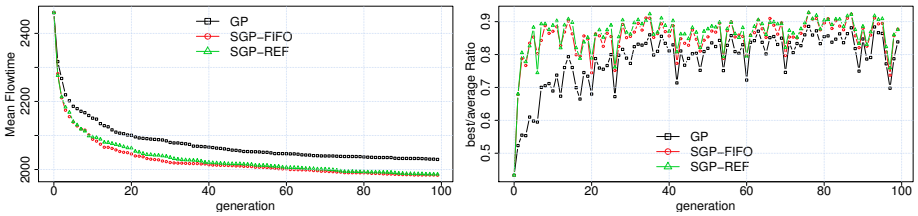

Fig. 8. Behaviours of *surrogate-assisted* GP – Minimise mean flowtime.

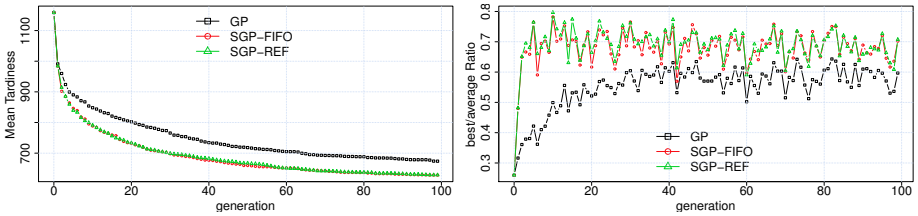

Fig. 9. Behaviours of *surrogate-assisted* GP – Minimise mean tardiness.

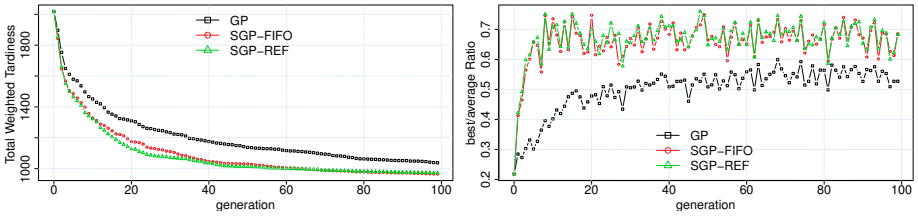

Fig. 10. Behaviours of *surrogate-assisted* GP – Minimise total weighted tardiness.

In each figure, the left part shows the progress of the best performance measure (based on 50 simulation replications) across generations (averaged over 30 independent runs). The right part shows the ratio between the best fitness and average fitness in each generation. The behaviour in the right part is not stable because the fitness function changes across generations. For all three performance measures, it is easy to see that SGP methods can find good rules a lot faster than GP. In three cases, SGP methods can find the best rules evolved by GP by using only 50 generations (two times faster than GP). Although, SGP methods employ elitism in their selection, they did not suffer from premature convergence. For the right parts of Fig. 8 to Fig. 10, it is obvious that SGP can also find better rules for each training replication as compared to GP. This helps confirm that surrogate model is still be very useful even when the simulation replication changes across generations. Therefore, we can take advantages of both changing simulation replications and surrogate models.

4.5 Multi-objective

One big advantages of using GP or EC for automated design of dispatching rules is that evolutionary multi-objective optimisation (EMO) has been mature over the last decade. Many effective and efficient EMO techniques have been developed to deal with hard multi-objective problems. Therefore, there is no reason not to utilise EMO to find the set of non-dominated rules, which help us understand better about possible trade-offs before selecting appropriate dispatching rules to apply. Fig. 11 demonstrates the usefulness of evolving non-dominated rules. This figure presents the non-dominated rules for five performance measures found by MO-GPHH [41] for our considered scenario. The black circles represent evolved non-dominated rules while the red crosses represent 31 rules developed in the literature. It is not hard to see that there are many interesting trade-offs that have been ignored in the literature. For example, we can find very effective rules to minimise both maximum flowtime (Fmax) and percentage of tardy jobs (%T); however, these rules have never been discovered.

One problem with evolving heuristics for multi-objective problems is overfitting. The chance of creating overfitted heuristics through crossover and mutation is actually quite high when Pareto dominance is used as the criteria for individual selection in GPHHs. A heuristic in the Pareto front accidentally generated may not be dominated by other good non-dominated heuristics, especially when many objectives are considered, even though the heuristic can contain many useless components. This issue also occurs in single objective methods; however, it is less severe because only one objective is considered and later generation can find more compact heuristics to replace the unnecessarily lengthy heuristics. It would be interesting to further extend the performance enhancement strategies mentioned in previous sections to improve the quality of evolved non-dominated rules.

4.6 Simplification

Evolving more compact dispatching rules is important in GP for JSS. More compact rules are easier to understand, especially when rules are in the form of priority functions. Moreover, compact rules will made the fitness evaluations faster and will significantly reduce the computational times of GP. In this section, we try to perform online simplification to hopefully help GP explore more compact rules through the evolution process. Because we are dealing with mathematical expression, a straightforward simplification technique is to apply simple symbolic simplification rules [56, 28].

In our experiments, we examine three methods to incorporate online simplification into GP. GPSimFit only simplifies evolved rules for fitness evaluations and the original rules are returned for genetic operations. GPSimALL simplifies all generated rules in the population. Finally, GPSimHalf randomly simplifies a half of rules in the population. The results from the three simplification strategies and the simple GP method are shown in Fig. 12. Basically, there is no significant difference between the three methods in term of performance measures.

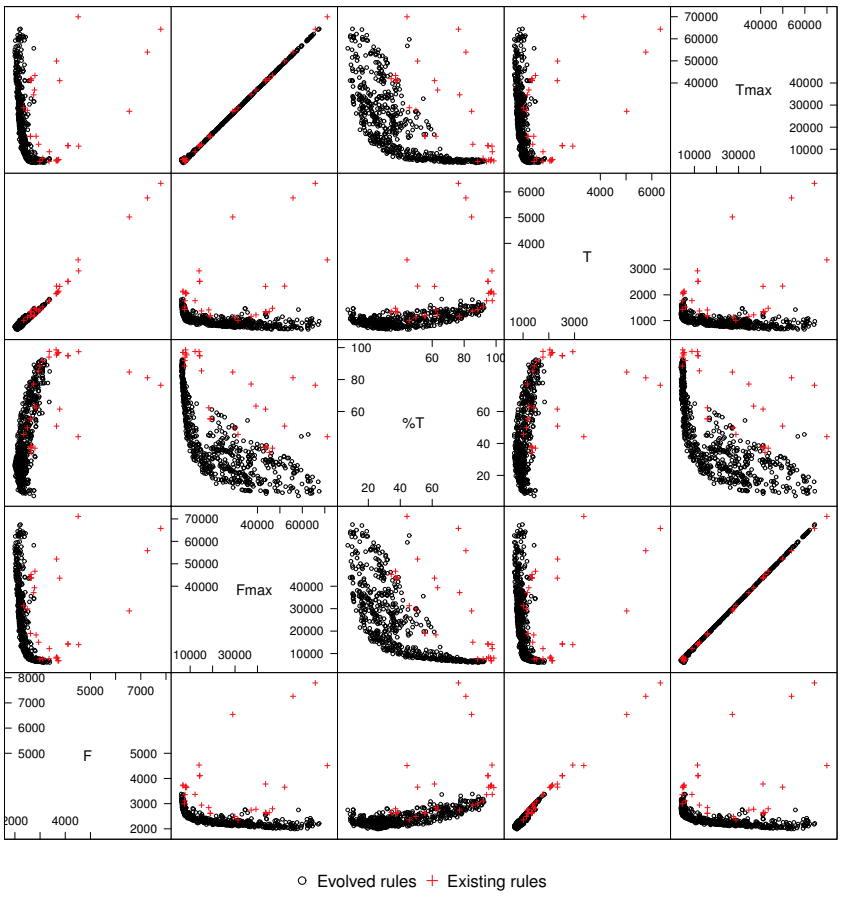

Fig. 11. Distribution of rules on the evolved Pareto front.

Simplification also does not ensure that more compact rules will be obtained at the end of GP runs. When examining the rules during the evolution process, we observe that simplification only make evolved rules compact temporarily. Over time, more complex rules are still generated with subtrees that cannot be further simplified. Therefore, the final rules are still quite complex. The only strategy with some potential benefit is GPSimFit where the computational times can be significantly reduced when times for fitness evaluations are reduced due to simplification. Since the original building blocks are preserved in this case, simplification is still useful even at the latter stage of the evolution.

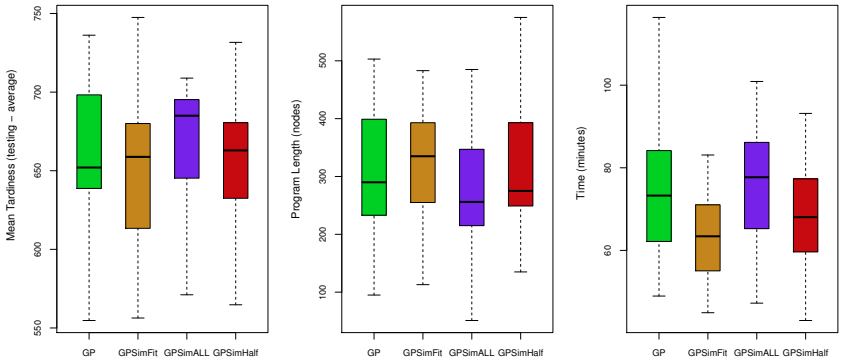

Fig. 12. Simplfication in GP.

5 Conclusions

In this chapter, we have provided an overview of current research on GP and JSS. Although JSS has been studied for decades, automated design of dispatching rules is still a relatively new research direction. The use of GP for evolving dispatching rules has made the design process a lot easier and more productive. Instead of spending time fabricating or improving dispatching rules, scheduling experts can focus more on investigating behaviours of obtained rules, comparing the trade-offs between different rules, and handling real-world constraints.

In future studies, representations of dispatching rules are still a key research topic to further enhance the effectiveness of GP for JSS. Since practical manufacturing systems can be very complex, many aspects need to be considered. Smart function and feature/attribute selection is worth investigating to make GP search more effective. Multiple scheduling decisions and multiple objectives are interesting research topics; however, they are still challenges for GP in order to effectively handle these issues. Surrogate-assisted GP is promising and still has a lot of room for further improvements. Finally, interpretability of rules needs to receive more attentions. Evolved rules are still quite complicated and we still have to manually simplify and transform evolved rules to an understandable forms. Therefore, there is a need of more systematic and effective approaches (e.g., visualisation, automatic analyses) to help us interpret evolved rules. This is also a key issue to gain the confidence of users on GP systems.

Because there are many optimisation problems which share similar characteristics of job shop scheduling, advances gained from this field can also be applied to other applications, especially ones with sequencing related decisions. The combination of GP with other optimisation methods should be an interesting topic to explore in future studies.

References

1. Asano, M., Ohta, H.: A heuristic for job shop scheduling to minimize total weighted tardiness. Computers and Industrial Engineering 42, 137–147 (2002)
2. Balas, E., Vazacopoulos, A.: Guided local search with shifting bottleneck for job shop scheduling. Management Science 44, 262–275 (1998)
3. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: Genetic Programming: An Introduction. Morgan Kaufmann (1998)
4. Beni, G., Wang, J.: Swarm intelligence in cellular robotic systems. In: Dario, P., Sandini, G., Aebischer, P. (eds.) Robots and Biological Systems: Towards a New Bionics?, NATO ASI Series, vol. 102, pp. 703–712. Springer Berlin Heidelberg (1993), http://dx.doi.org/10.1007/978-3-642-58069-7_38
5. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence: from natural to artificial systems. Oxford University Press, Inc., New York, NY, USA (1999), http://portal.acm.org/citation.cfm?id=328320
6. Branke, J., Hildebrandt, T., Scholz-Reiter, B.: Hyper-heuristic evolution of dispatching rules: A comparison of rule representations. Evolutionary Computation (2014), in press (DOI:10.1162/EVCO_a_00131)
7. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: Mumford, C., Jain, L. (eds.) Computational Intelligence, Intelligent Systems Reference Library, vol. 1, pp. 177–201. Springer Berlin Heidelberg (2009)
8. Cheng, V.H.L., Crawford, L.S., Menon, P.K.: Air traffic control using genetic search techniques. In: McClamroch, N.H., Sano, A., Gruebel, G. (eds.) Proceedings of the 1999 IEEE International Conference on Control Applications, vol. 1, pp. 249–254. IEEE Press, Piscataway, NJ (1999)
9. Chiang, T.C., Shen, Y.S., Fu, L.C.: A new paradigm for rule-based scheduling in the wafer probe centre. International Journal of Production Research 46(15), 4111–4133 (2008)
10. Dimopoulos, C., Zalzala, A.M.S.: Investigating the use of genetic programming for a classic one-machine scheduling problem. Advances in Engineering Software 32(6), 489–498 (2001)
11. El-Bouri, A., Balakrishnan, S., Popplewell, N.: Sequencing jobs on a single machine: a neural network approach. European Journal of Operational Research 126(3), 474–490 (2000)
12. Essafi, I., Mati, Y., Dauzère-Pérès, S.: A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. Computer & Operations Research 35(8), 2599–2616 (2008)
13. Ferreira, C.: Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence. Springer-Verlag, Germany, 2 edn. (2006)
14. Garey, M.R., Johnson, D.S., Sethi, R.: The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1(2), 117–129 (1976)
15. Geiger, C.D., Uzsoy, R., Aytuğ, H.: Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. Journal of Scheduling 9(1), 7–34 (2006)
16. Giffler, B., Thompson, G.L.: Algorithms for solving production-scheduling problems. Operations Research 8(4), 487–503 (1960)
17. Goncalves, J.F., de Magalhaes Mendes, J.J., Resende, M.G.C.: A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 167(1), 77–95 (2005)

18. Hildebrandt, T., Heger, J., Scholz-Reiter, B.: Towards improved dispatching rules for complex shop floor scenarios — a genetic programming approach. In: Pelikan, M., Branke, J. (eds.) GECCO '10: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 257–264. ACM Press, New York (2010)
19. Hildebrandt, T., Branke, J.: On using surrogates with genetic programming. Tech. rep., Warwick Business School (2014)
20. Holthaus, O., Rajendran, C.: Efficient jobshop dispatching rules: Further developments. Production Planning & Control 11(2), 171–178 (2000)
21. Hunt, R., Johnston, M., Zhang, M.: Evolving "less-myopic" scheduling rules for dynamic job shop scheduling with genetic programming. In: GECCO'14: Proceedings of Genetic and Evolutionary Computation Conference (2014), (to appear)
22. Ingimundardottir, H., Runarsson, T.P.: Supervised learning linear priority dispatch rules for job-shop scheduling. In: Coello Coello, C.A. (ed.) Learning and Intelligent Optimization, LNCS, vol. 6683, pp. 263–277. Springer, Berlin and Heidelberg (2011)
23. Jakobović, D., Budin, L.: Dynamic scheduling with genetic programming. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) Genetic Programming, LNCS, vol. 3905, pp. 73–84. Springer, Berlin and Heidelberg (2006)
24. Jakobović, D., Marasović, K.: Evolving priority scheduling heuristics with genetic programming. Applied Soft Computing 12(9), 2781–2789 (2012)
25. Jayamohan, M.S., Rajendran, C.: New dispatching rules for shop scheduling: a step forward. International Journal of Production Research 38, 563–586 (2000)
26. Jayamohan, M.S., Rajendran, C.: Development and analysis of cost-based dispatching rules for job shop scheduling. European Journal of Operational Research 157(2), 307–321 (2004)
27. Jedrzejowicz, P., Ratajczak-Ropel, E.: Agent-based gene expression programming for solving the rcpsp/max problem. In: Kolehmainen, M., Toivanen, P., Beliczynski, B. (eds.) Adaptive and Natural Computing Algorithms, Lecture Notes in Computer Science, vol. 5495, pp. 203–212. Springer Berlin Heidelberg (2009)
28. Johnston, M., Liddle, T., Zhang, M.: A relaxed approach to simplification in genetic programming. In: Genetic Programming, LNCS, vol. 6021, pp. 110–121. Springer (2010)
29. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
30. Kreipl, S.: A large step random walk for minimizing total weighted tardiness in a job shop. Journal of Scheduling 3, 125–138 (2000)
31. Kuczapski, A.M., Micea, M.V., Maniu, L.A., Cretu, V.I.: Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling. Information Technology and Control 39(1), 32–37 (2010)
32. van Laarhoven, P.J.M., Aarts, E.H.L., Lenstra, J.K.: Job shop scheduling by simulated annealing. Operations Research 40(1), 113–125 (1992)
33. Lourenco, H.R.: Job-shop scheduling: Computational study of local search and large-step optimization methods. European Journal of Operational Research 83(2), 347–364 (1995)
34. McKay, K.N., Safayeni, F.R., Buzacott, J.A.: Job-shop scheduling theory: What is relevant? Interfaces 18, 84–90 (1988)
35. Miyashita, K.: Job-shop scheduling with genetic programming. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.G. (eds.) GECCO 2000: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 505–512. Morgan Kaufmann, San Francisco (2000)

36. Nguyen, S., Zhang, M., Johnston, M., Tan, K.: Learning iterative dispatching rules for job shop scheduling with genetic programming. The International Journal of Advanced Manufacturing Technology 67(1–4), 85–100 (2013)
37. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. IEEE Transactions on Evolutionary Computation 17(5), 621–639 (2013)
38. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Dynamic multi-objective job shop scheduling: a genetic programming approach. In: Etaner-Uyar, A.Ş., Özcan, E., Urquhart, N. (eds.) Automated Scheduling and Planning, Studies in Computational Intelligence, vol. 505, pp. 251–282. Springer, Berlin and Heidelberg (2013)
39. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Learning reusable initial solutions for multi-objective order acceptance and scheduling problems with genetic programming. In: Krawiec, K., Moraglio, A., Hu, T., Etaner-Uyar, A.Ş., Hu, B. (eds.) Genetic Programming, LNCS, vol. 7831, pp. 157–168. Springer, Berlin and Heidelberg (2013)
40. Nguyen, S., Zhang, M., Johnston, M., Tan, K.C.: Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. IEEE Transactions on Evolutionary Computation 18(2), 193–208 (2014)
41. Nguyen, S.: Automatic Design of Dispatching Rules for Job Shop Scheduling with Genetic Programming. Ph.D. thesis, Victoria University of Wellington (2013)
42. Nie, L., Gao, L., Li, P., Li, X.: 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 (2013)
43. Nie, L., Shao, X., Gao, L., Li, W.: Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. The International Journal of Advanced Manufacturing Technology 50(5–8), 729–747 (2010)
44. Nie, L., Bai, Y., Wang, X., Liu, K.: Discover scheduling strategies with gene expression programming for dynamic flexible job shop scheduling problem. In: Tan, Y., Shi, Y., Ji, Z. (eds.) Advances in Swarm Intelligence, vol. 7332, pp. 383–390 (2012)
45. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. Management Science 42, 797–813 (1996)
46. Ouelhadj, D., Petrovic, S.: A survey of dynamic scheduling in manufacturing systems. Journal of Scheduling 12(4), 417–431 (2009)
47. Petrovic, S., Fayad, C., Petrovic, D., Burke, E., Kendall, G.: Fuzzy job shop scheduling with lot-sizing. Annals of Operations Research 159, 275–292 (2008)
48. Pickardt, C.W., Hildebrandt, T., Branke, J., Heger, J., Scholz-Reiter, B.: Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. International Journal of Production Economics 145(1), 67–77 (2013)
49. Pinedo, M.L.: Scheduling: Theory, Algorithms, and Systems. Springer, New York, 3rd edn. (2008)
50. Pinedo, M., Singer, M.: A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. Naval Research Logistics 46(1), 1–17 (1999)
51. Ponnambalam, S.G., Ramkumar, V., Jawahar, N.: A multiobjective genetic algorithm for job shop scheduling. Production Planning and Control 12(8) (2001)
52. Potts, C.N., Strusevich, V.A.: Fifty years of scheduling: a survey of milestones. Journal of the Operational Research Society 60(Supplement 1), 41–68 (2009), http://www.palgrave-journals.com/jors/journal/v60/ns1/abs/jors20092a.html

53. Sels, V., Gheysen, N., Vanhoucke, M.: 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 (2011)
54. Sha, D., Hsu, C.Y.: A hybrid particle swarm optimization for job shop scheduling problem. Computers & Industrial Engineering 51(4), 791 – 808 (2006)
55. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers & Industrial Engineering 54(3), 453–473 (2008)
56. Wong, P., Zhang, M.: Algebraic simplification of gp programs during evolution. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. pp. 927–934. GECCO '06 (2006)
57. Xing, L.N., Chen, Y.W., Wang, P., Zhao, Q.S., Xiong, J.: A knowledge-based ant colony optimization for flexible job shop scheduling problems. Applied Soft Computing 10(3), 888 – 896 (2010)
58. Yamada, T., Nakano, R.: 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. pp. 146–151 (1995)
59. Yin, W.J., Liu, M., Wu, C.: 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., Gedeon, T. (eds.) The 2003 Congress on Evolutionary Computation (CEC 2003), vol. 2, pp. 1050–1055. IEEE Press, Piscataway, NJ (2003)