Complex Intell. Syst. (2017) 3:41–66
DOI 10.1007/s40747-017-0036-x

REVIEW ARTICLE

Genetic programming for production scheduling: a survey with a unified framework

Su Nguyen[1] · Yi Mei[1] · Mengjie Zhang[1]

Received: 26 November 2016 / Accepted: 7 February 2017 / Published online: 24 February 2017

Abstract Genetic programming has been a powerful technique for automated design of production scheduling heuristics. Many studies have shown that heuristics evolved by genetic programming can outperform many existing heuristics manually designed in the literature. The flexibility of genetic programming also allows it to discover very sophisticated heuristics to deal with complex and dynamic production environments. However, as compared to other applications of genetic programming or scheduling applications of other evolutionary computation techniques, the configurations and requirements of genetic programming for production scheduling are more complicated. In this paper, a unified framework for automated design of production scheduling heuristics with genetic programming is developed. The goal of the framework is to provide the researchers with the overall picture of how genetic programming can be applied for this task and the key components. The framework is also used to facilitate our discussions and analyses of existing studies in the field. Finally, this paper shows how knowledge from machine learning and operations research can be employed and how the current challenges can be addressed.

Keywords Genetic programming · Job shop · Production scheduling · Hyper-heuristics

✉ Mengjie Zhang
mengjie.zhang@ecs.vuw.ac.nz

Su Nguyen
su.nguyen@ecs.vuw.ac.nz

[1] Evolutionary Computation Research Group, Victoria University of Wellington, Wellington, New Zealand

Introduction

Production scheduling has been one of the most popular research topics in operations research, management science, and artificial intelligence. Because of limited production resources, jobs or customer orders usually have to wait in the shop floor significantly longer than their actual processing times. Production scheduling is required to determine when a job needs to be processed, which machine to process, or which priority assigned to the job. The goal of production scheduling is to effectively utilise the available resources to achieve some organisational objectives such as minimising average time that jobs have to spend in the system and minimise penalties caused by late deliveries. Over the years, new production technologies have been adopted but production scheduling is still an essential task to help businesses coordinate production activities and become more competitive.

Production scheduling has a number of challenges. For example, production environments are dynamic and uncertain (e.g. job arrivals, job cancellations, machine breakdowns), which cause computational difficulties for most optimisation techniques. The complexity of the production systems caused by heterogeneous production processes (e.g. batching, sequence-dependent setup times, assembly) also makes scheduling tasks particularly hard. Moreover, production scheduling has to take into account multiple conflicting objectives to ensure that the obtained schedules are approved and applicable.

In the last few decades, a large number of studies in artificial intelligence (AI) and operations research (OR) have been conducted to develop new scheduling techniques for production scheduling. Many techniques to search for optimal solutions such as branch-and-bound and dynamic programming have been investigated in the literature but they are mainly restricted to small and special problems. However, these techniques are too time consuming and impractical to handle real-world production scheduling problems. Therefore, heuristics have been proposed to find "good enough" and "quick" solutions. Scheduling heuristics can be very simple such as simple dispatching rules First-In-First-Out (FIFO), shortest processing time (SPT), and earliest due date (EDD). Some heuristics also monitor the status of the shop and machine to decide which dispatching rule to be applied. For example, the rule FIFO/SPT will apply FIFO when the jobs in the queue of the considered machine have been waiting for more than a specific time and SPT will be applied otherwise. Heuristics can also be very sophisticated such as composite dispatching rules [119], which are combinations of simple rules basically in the form of sophisticated human-made priority functions of various scheduling parameters. Other heuristics based on understandings of problem domains have been also proposed in the literature such as shifting bottlenecks [5]. More general techniques based on meta-heuristics such as tabu search [106] and genetic algorithm [16] have been developed to deal with different production scheduling problems and show promising results. However, it is noted that designing a good heuristic is not a trivial task and it can be very time consuming and requires a lot of problem domain knowledge.

The field of automated heuristic design or hyper-heuristics [21,23] 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. In this survey, we focus on hyper-heuristics for heuristic generation to fabricate a new heuristic (or meta-heuristic) by combining various small components (normally common statistics/features or operations used in pre-existing heuristics) and these heuristics are trained on a training set and evolved to become more effective. 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. In the last decade, genetic programming [6,67] has been the dominating technique for automated design for production scheduling heuristics [20].

As compared to other hyper-heuristics based on supervised learning such as decision tree [107,127], logistic regression [57], support vector machine [129], and artificial neural networks [32,138], genetic programming (GP) has shown a number of key advantages. First, GP has flexible representations which allow various heuristics to be represented as different 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 of a heuristic without assuming any model based on a particular distribution or domain knowledge. 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 very important feature to enhance its applicability in practice.

In the last decade, there is a growing number of articles about automated heuristic design and its applications. In Burke et al. [22], a general genetic programming-based hyper-heuristic framework was presented and some studies were used to explain the idea. Burke et al. [24] provided a general survey of related studies on hyper-heuristics developed to deal with a wide range of scheduling and combinatorial optimisation problems. Both heuristic selection and heuristic generation are discussed in that survey. Brief discussions of hyper-heuristic applications were also provided. Although there are a number survey papers for production scheduling such as Ouelhadj and Petrovic [109] and Hart et al. [43], they just focused on traditional meta-heuristics to find optimal solutions for a set of static scheduling instances. Recently, Branke et al. [20] presented the most comprehensive survey of existing studies on hyper-heuristics and production scheduling. In the survey, critical design aspects such as attribute selection, representation, and fitness functions are presented. However, as the survey attempts to cover all existing hyper-heuristic approaches for production scheduling, only key general issues are provided.

GP-based hyper-heuristics have been applied to many production scheduling problems and many new algorithms have been developed. As compared to other evolutionary computation techniques, GP is more sophisticated because of its variable representations and special operators. Production scheduling problems themselves are also complicated and have special characteristics as compared to other combinatorial optimisation problems. To successfully apply genetic programming to production scheduling, researchers will need to understand technical aspects from these two research areas. The goal of this paper is to provide a comprehensive review of existing studies on using GP for automated design of production scheduling heuristics. A unified framework is presented in this paper to show how GP can be applied to design production scheduling heuristics and the key components that can influence the performance of GPHHs for these tasks. Each key component is described in detail and we also analyse how they are treated in the previous studies. Then we discuss the connections between GP and other AI and OR techniques. Finally we highlight the current issues and challenges. It is expected that the survey will help the beginning researchers have a good overview of this emerging and interesting research area and pick up key ideas and challenges for the future studies.

Background

Before moving to detailed discussions of GP for production scheduling, we provide a brief overview of production scheduling and genetic programming. This section targets researchers who are new to these two research areas. Those who are familiar with scheduling and GP concepts can safely skip this section.

Production scheduling

Production scheduling is about allocation of scarce manufacturing resources to tasks over time. Depending on the nature of production processes and customer demand, there are many different types of production environments. In the literature, a scheduling problem is described by the triplet $\beta|\gamma|\delta$, where β represents the machine environment, γ provides the processing characteristic (it may contain no entry at all or multiple entries), and δ describes the objective to be optimised [119]. The goal of production scheduling is to determine when a job (or a customer order) should be processed and which machine (i.e. production resource in β) is used to process that job to optimise δ, given that all process constraints γ are satisfied. For the single machine environment, a job only needs to go through one production process to be completed. For multi-stage (with multiple machines) environment, a job is a sequence of operations, each of which is to be performed on a particular machine.

Studies of production scheduling literature can be classified into two main streams. The first one focuses on static scheduling problems where information of all jobs is available. Previously studies on static problems try to develop efficient algorithms to find optimal solutions. Nevertheless, many scheduling problems are proven to be NP-hard. Thus, most proposed exact methods such as branch-and-bound and dynamic programming fail to find optimal solutions (or can only find optimal solutions for very small instances). As a result, a large number of scheduling heuristics, e.g. NEH for flows shop scheduling [136], and shifting bottleneck [119] for job shop scheduling, are developed to search for “good enough” solutions within a reasonable computational times. Meta-heuristics such as tabu search [106], genetic algorithm [16], particle swarm optimisation [126] have also been applied extensively to solve production scheduling problems.

For dynamic scheduling problems, jobs may arrive randomly over time and their information is not available before their arrivals. Dispatching rules is the most popular approach for dynamic scheduling problem. In most cases, dispatching rules are represented by priorities functions that assign priorities to jobs. Then the jobs with the highest priority will be processed first. Many rules have been developed by practitioners and researchers to cope with a wide range of production environments. Three attractive characteristics of the dispatching rules are their efficiency, reactiveness, and interpretability. GP plays a major role in dynamic scheduling, which is to be described below.

Genetic programming

GP [67] is an evolutionary computation (EC) method, inspired by biological evolution, to automatically find computer programs (i.e. scheduling heuristics in our case) for solving a specific task. In genetic programming, a population of computer programs (individuals) is created and these programs are evolved to gain higher fitnesses through an evolutionary process. In each generation of the evolutionary process, each program is evaluated using a pre-defined fitness function, which assesses the ability of the program to perform a specific computational task. The fitness values obtained by programs in the population decide the chance of each program to survive and reproduce (with genetic operators) in the next generation.

Different from genetic algorithms, each individual of a GP population is not represented by a fixed-length string of genes (bits, real numbers, or symbols). Because the shape and length of the final program is normally not known by the user, individuals in GP usually represent programs as tree structures which are constructed by a set of terminals and a set of functions. Basically, a GP individual is a specific combination of elements selected from these two sets. The terminal set consists of programs’ inputs (also referred to as features or attributes) or (ephemeral random) constants [67]. Meanwhile, the function set can contain arithmetic operators, logic operators, mathematical or specialised functions used to construct GP programs. Other representations are also developed in grammar-based GP [139], graph-based GP [120,124] and linear-based GP [17,69] and achieve very promising results. For each representation, special genetic operators (crossover, mutation) are developed to help GP create new individuals based on parent programs.

Genetic programming for production scheduling

GP for production scheduling has been very active in recent years. With flexible representations, GP can represent and evolve effective scheduling heuristics to deal with a wide range of scheduling problems. In addition, since GP does not rely on any specific assumptions, it can be easily extended to deal with different production scheduling problems. Figure 1 shows the number of published articles in this area since 2000. Miyashita [81] is probably the first study that used GP to evolve dispatching rules for job shop scheduling

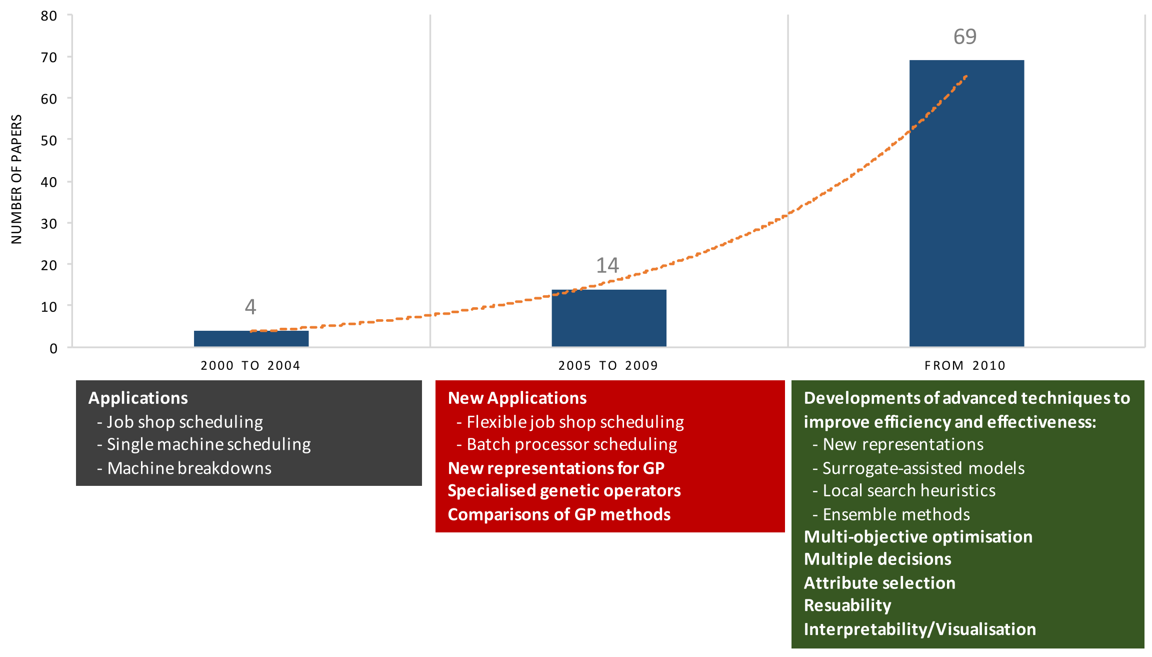


Fig. 1 Published articles on GP for production scheduling since 2000

and showed the effectiveness of evolved dispatching rules. The paper also analysed different ways dispatching rules can be learned in a general job shops. From 2000 to 2004, there were only four papers about this topic and mainly focused on applications of GP for classical production scheduling problems. From 2005 to 2009, GP is applied to more production scheduling problems and researchers become interested in improving the performance of GP. New representations and genetic operators were proposed to cope with specific scheduling problems [38]. Experiments to compare different GP methods were also conducted [59,73]. Since 2010, there have been a dramatic growth in the number of studies on this topic. These recent studies have focused on improving the effectiveness and efficiency of GP for production scheduling by developing new representations [89], new surrogate-assisted models [45], local search heuristics [97], and ensemble methods [42,113]. Practical issues such as multiple conflicting objectives [35,90], multiple decisions [95,104], attribute selection [79] are catching more attentions. Moreover, researchers have been interested in reusability of evolved dispatching rules as well as their interpretability [46,95]. Table 1 shows a list of major papers about automatic design of production scheduling heuristics via GP and their focuses.

Unified framework

Figure 2 shows a proposed unified framework for automated heuristic design of production scheduling with GP. Based on the scheduling problem of interest, the first step is to determine the meta-algorithm of scheduling heuristics, which explains how the heuristic will work. Based on the meta-algorithm, we need to identify which component(s) should be evolved by GP. Then the suitable representations, relevant features or attributes, and function sets used to evolve heuristics are decided. The evaluation models or evaluators are also needed help evaluate the performance of evolved heuristics during the evolution process. In the lower part of Fig. 2, the evolutionary process of GP is presented. Similar to other EC techniques, GP starts with a population of randomly generated heuristics (based on the representation, function sets, and terminals set defined previously). Each generated heuristics are then evaluated by the evaluation model to determine their quality, i.e. fitness. After all individuals in the GP population are evaluated, genetic operators are applied to generate new heuristics and potential heuristics are selected to form the population for the next generation. The population will be evolved over many generations and the evolution is stopped when the termination condition is met. Post-processing routines can also be applied to simplify and interpret the evolved heuristics. In the rest of this section, we will analyse each key component in this framework and the related existing studies.

Production scheduling problems

GP has been applied in a wide range of production scheduling problems, ranging from single machine scheduling [30,38, 59,100,142], parallel machine scheduling [31,60], to (flexible) job shop scheduling [42,53,59,63,79,81,88,89,95,102,

Table 1 Topics covered by previous studies

Paper	Meta-algorithm	Representation	Genetic operator	Search mechanism	Fitness function	Multiple decisions	Multi-objective	Attribute analysis	Interpretability	Generalisability
Miyashita [81]				✓						
Dimopoulos and Zalzala [30]									✓	✓
Yin et al. [142]	✓	✓	✓			✓				
Ho and Tay [50]										✓
Geiger et al. [38]			✓							✓
Jakobovic and Budin [59]	✓	✓								
Jakobovic et al. [60]	✓									
Tay and Ho [134]								✓		
Beham et al. [14]				✓						
Geiger and Uzsoy [37]									✓	✓
Baykasoglu [9]		✓					✓			
Li et al. [73]	✓	✓	✓							
Tay and Ho [135]					✓		✓		✓	
Yang et al. [141]	✓						✓			
Mucientes et al. [83]		✓			✓					
Baykasolu and Gken [12]	✓									
Kofler et al. [66]	✓									
Furuholmen et al. [36]	✓			✓						
Hildebrandt et al. [46]	✓				✓				✓	✓
Kuczapski et al. [68]	✓									
Nie et al. [100]	✓								✓	
Pickardt et al. [117]										✓
Baykasoglu et al. [11]		✓	✓							✓
Abednego and Hendratmo [1]										✓
Nguyen et al. [85]	✓	✓								✓
Nie et al. [101]				✓	✓		✓			
Nie et al. [102]	✓									
Vazquez-Rodriguez and Ochoa [136]	✓								✓	✓

Table 1 continued

Paper	Meta-algorithm	Representation	Genetic operator	Search mechanism	Fitness function	Multiple decisions	Multi-objective	Attribute analysis	Interpretability	Generalisability
Jakobovi and Marasovi [58]	✓			✓						
Nguyen et al. [86]	✓	✓		✓			✓			
Nguyen et al. [87]	✓									✓
Nie et al. [103]				✓		✓				
Han et al. [41]										✓
Nguyen et al. [88]	✓	✓							✓	✓
Nguyen et al. [89]	✓	✓			✓			✓	✓	✓
Nguyen et al. [90]							✓			✓
Nguyen et al. [91]	✓						✓			
Nie et al. [104]		✓				✓				
Park et al. [110]	✓									✓
Park et al. [111]	✓	✓								✓
Pickardt et al. [118]	✓	✓		✓		✓				✓
Qin et al. [122]				✓	✓					✓
Nie et al. [105]	✓									
Hildebrandt and Branke [45]				✓	✓			✓		
Hildebrandt et al. [47]								✓		✓
Hunt et al. [53]								✓		
Hunt et al. [54]		✓								✓
Nguyen et al. [93]	✓			✓	✓					✓
Nguyen et al. [92]	✓									✓
Nguyen et al. [94]	✓								✓	✓
Nguyen et al. [95]		✓		✓		✓	✓		✓	✓
Nguyen et al. [96]			✓	✓	✓					
Park et al. [112]										✓
Alsina et al. [3]	✓	✓								
Belisrio and Pierreval [15]	✓	✓							✓	
Sim and Hart [130]			✓	✓	✓					✓
Branke et al. [18]		✓		✓				✓	✓	✓

Table 1 continued

Paper	Meta-algorithm	Representation	Genetic operator	Search mechanism	Fitness function	Multiple decisions	Multi-objective	Attribute analysis	Interpretability	Generalisability
Hunt et al. [56]				✓	✓					
Hunt et al. [55]		✓						✓	✓	
Nguyen et al. [97]			✓	✓						
Nguyen et al. [98]				✓	✓		✓			
Park et al. [114]				✓						✓
Park et al. [113]				✓						✓
Shi et al. [128]				✓						
Wang et al. [137]				✓						
Baykasoglu and Ozbakr [10]	✓			✓			✓			✓
Chen et al. [26]				✓						✓
Durasevic et al. [31]	✓	✓		✓						
Freitag and Hildebrandt [35]				✓			✓			
Hart and Sim [42]			✓	✓	✓					✓
Karunakaran et al. [63]				✓			✓			
Park et al. [115]				✓	✓					
Park et al. [116]		✓		✓						
Riley et al. [123]			✓					✓		
Branke et al. [19]	✓									✓
Mei and Zhang [78]										✓
Karunakaran et al. [64]								✓		✓
Masood et al. [75]				✓			✓			
Mei et al. [79]				✓				✓		
Nguyen [84]	✓			✓						✓
Nguyen et al. [99]				✓	✓				✓	

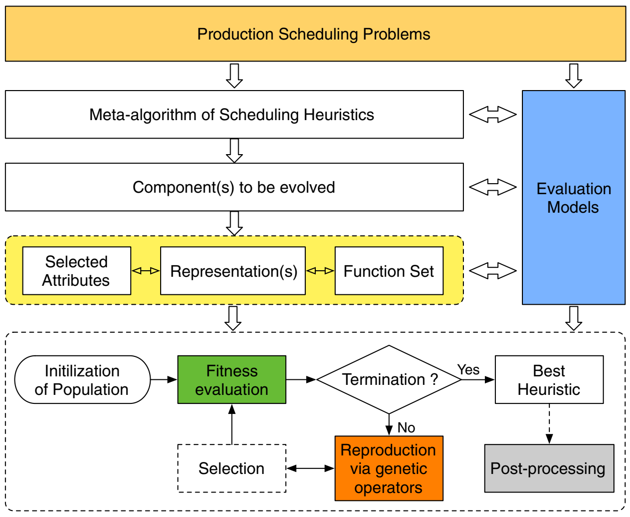

Production Scheduling Problems
Meta-algorithm of Scheduling Heuristics
Component(s) to be evolved
Selected Attributes
Representation(s)
Function Set
Evaluation Models
Initilization of Population
Fitness evaluation
Termination ?
Yes
No
Best Heuristic
Selection
Reproduction via genetic operators
Post-processing

Fig. 2 Unified framework

104, 135, 136]. Most machines considered in these problems are the same in terms of capability (eligibility to handle a job) and assumptions (e.g. utilisation level). Although some special cases are considered in the literature such as batching [37, 118], machine breakdowns [142], and unrelated parallel machines [31], these are very limited. In addition, most scheduling problems handled by GP are dynamic problems where jobs will arrive randomly over time and their information is only available upon their arrivals. For most of these problems, the main concern is to find the best way to prioritise or schedule jobs to optimise some objectives such as makespan, mean flowtime, maximum flowtime, mean tardiness, and total weighted tardiness.

Meta-algorithm of scheduling heuristics

As the scheduling problems are formulated, one of the key steps is to identify the meta-algorithm of scheduling heuristics. This step provides the basic concepts of the scheduling heuristics and explains how scheduling decisions will be made. It is expected that the meta-algorithm is general enough, ideally can lead to optimal scheduling decisions. In this step, it is important to (1) identify the *fixed* and *variable* components of the meta-algorithm, and (2) understand the *complexity* of the scheduling heuristics based on the meta-algorithm.

For example, Fig. 3 shows a generalized schedule construction algorithm [16, 89, 119] to construct an active schedule, a non-delay schedule or a hybrid of both active and non-delay schedules with a specific dispatching (priority) rule. This algorithm was based on the Giffler and Thompson [39] and has been widely used in the scheduling literature to deal with different production scheduling problems. The algorithm first identifies the machine `m`* to be scheduled based on the earliest completion time of all available operations `P`. Then a subset **`P'`** $\in$ `P` including candidate operations to be scheduled next is determined by checking if the ready times of these operations are smaller than **`S`***+**`alpha`**(**`C`***-**`S`***). The parameter **`alpha`** is the non-delay factor $\in$ [0, 1] to control the look-ahead ability of the algorithm by restricting operations included in `P'` (the algorithm generates non-delay schedules if **`alpha`** = 0 and active schedules if **`alpha`**=1). A **`dispatching rule`** is applied to determine the next operation in `P'` to schedule next. It is clear that performance of the algorithm depends on how the subset `P'` is determined and how the next operation is picked. This algorithm is very efficient because the next operation can be determined easily by calculating priorities for jobs in `P'`. These two decisions are governed by the non-delay factor **`alpha`** and the **`dispatching rule`**. In this algorithm, **`alpha`** and **`dispatching rule`** are the two *variable* components and the rest are *fixed*. When designing scheduling heuristics based on the algorithm in Fig. 3, we need to decide **`alpha`** and **`dispatching rule`** to apply to obtain optimal or near optimal schedules. These two are candidate components which can be evolved using GP.

It is noted that the above algorithm and its variants have been used in most previous studies on automated design of production scheduling heuristics. Nguyen et al. [88] proposed iterative dispatching rules (IDR) which are able to create multiple schedules iteratively and the new schedule is generated based on the information of the previous generated schedule (e.g. completion times of jobs). Although their meta-algorithm is slightly different from one in Fig. 3 (only small change in step 2 and step 7), two variable components to be designed are still **`alpha`** and **`dispatching rule`**. In the *variable neighborhood search* with IDR [88], *k* iterative dispatching rules can be used to improve the quality of the final schedule. In this case, the *variable* components are the *k* dispatching rules and the non-delay factor.

Usually meta-algorithms of scheduling heuristics are developed by studying existing heuristics in the literature. Other meta-algorithms have also been investigated such as beam search heuristics [93, 111], ensembles of heuristics [42], adaptive scheduling heuristics based on bottleneck machines [59], and NEH heuristics [136]. These evolved heuristics are very different in terms of computational costs and how they build schedules. While the majority of scheduling heuristics investigated in the literature are construction heuristics [23], i.e. step-by-step construct the schedule, some have also investigated improvement heuristics that iteratively refine the schedule [74, 88, 110, 136]. One of the reasons is that the improvement heuristics are usually much more computationally expensive as compared to construction heuristics. Although improvement heuristics developed by GP show very promising results, they are still restricted to static scheduling problems. The studies of applying evolved

Fig. 3 Example meta-algorithm of schedule construction heuristics

```
1. Initialize P with unscheduled operations (with no precedence operation)
2. From the set of operations P, find the operation with earliest completion
   time C*, and its corresponding machine m*
3. Find the earliest start time S* of operations in P that need to be processed
   by m*
4. Create the set P with all operations in P with ready time smaller than
   S*+alpha(C*-S*)
5. Apply a dispatching rule to find the next operation O in P to be scheduled
   next on m*
6. Remove O from P and add its successor operation (if available) to P
7. If P is not empty, return to step 2
```

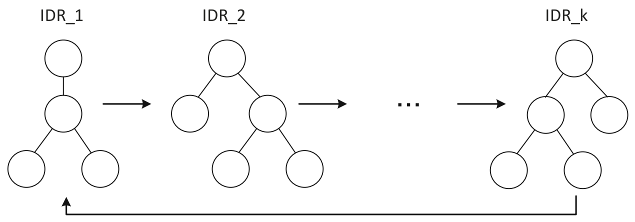


Fig. 4 Variable neighborhood search with IDR [88]

improvement heuristics to dynamic environments will be an interesting research topic in future studies. When dealing with different planning and scheduling decisions, different meta-algorithms can also be used [95,103].

Component(s) to be evolved

The meta-algorithms discussed above help us understand how scheduling decisions are made and its basic (variable) components. Depending on the production environments, one or more components will need to be designed. Below are some popular components that have been investigated in the literature:

- *Dispatching rule or priority rule* is used for sequencing tasks in a scheduling problem. At the moments when a sequencing decision needs to be made, dispatching rules will prioritise the jobs in the queue of a considered machine. Then, the job with the highest priority is processed next.
- *Routing rule or machine assignment rule* is used to decide which machine from a pre-determined set of machines to process the considered operation. Routing rules are usually investigated when dealing with flexible job shop scheduling problems.
- *Due date assignment rule* is used to determine the due dates for arriving jobs by estimating the job flowtimes (the time taken from the arrival until the completion of a job).
- *Batch formulation rule* is used to determine how to group the individual jobs into batches. This is mainly investigated for shops with batching processes.
- *Performance/processing time estimation*: a model is obtained to estimate processing times or performance measures for planning purposes.
- *Inserted idle time* a model is obtained to estimate the idle times to be inserted into the schedule to absorb disruptions.
- *Non-delay factor* is used to govern the look-ahead ability of dispatching rules, i.e. to what extent upstream jobs will be considered when making scheduling decisions.
- *Improvement/greedy heuristics* is used to iteratively improve the quality of schedules in static scheduling problems.

Table 2 summarises the basic components evolved by GP in the literature. Dispatching rules are the most popular component investigated in previous studies as sequencing and scheduling decisions are required in most production scheduling problems. Other components are more problem specific and only investigated when dealing with production systems with special processes or requirements. Because the structure of these components are usually unknown in advance, their corresponding search spaces are large and finding optimal or near-optimal solutions is very challenging. Moreover, evaluating the quality of evolved heuristics is not straightforward because of the complex and dynamic production environments. Thus, evolving scheduling components is challenging and time consuming. More discussions about these challenges and proposed techniques to overcome them will be provided in the upcoming sections.

However, it is noted that we do not need to evolve all those components. There are a number of reasons that evolving all components is not always a good idea. First, it can be very time consuming to evolve multiple components at the same time because the evaluation costs (for fitness evaluations) will be higher. Second, the search space of scheduling heuristics is also much larger as evolved components can be very different and can use different function sets and feature sets (will be discussed more in “Representations, function sets, and terminal sets”). Finally, some good alternatives are available for the variable components in some specific cases.

Most previous studies focused on only one component and fixed all other components to reduce the complexity. For example, Tay and Ho [135] applied GP to evolve dispatching

Table 2 Component(s) to be evolved by GP

Component	References
Dispatching rule or priority rule	Miyashita [81], Dimopoulos and Zalzala [30], Yin et al. [142], Ho and Tay [50], Geiger et al. [38], Jakobovic and Budin [59], Jakobovic et al. [60], Tay and Ho [134], Beham et al. [14], Tay and Ho [135], Yang et al. [141], Baykasoglu et al. [11], Kofler et al. [66], Hildebrandt et al. [46], Kuczapski et al. [68], Nie et al. [100], Pickardt et al. [117], Abednego and Hendratmo [1], Nie et al. [102], Jakobovi and Marasovi [58], Nguyen et al. [86], [103], Nguyen et al. [89–91], Nie et al. [104,105], Park et al. [110,111], Pickardt et al. [118], Qin et al. [122], Hildebrandt and Branke [45], Hildebrandt et al. [47], Hunt et al. [53,54], Nguyen et al. [96], Park et al. [112], Branke et al. [18], Chen et al. [26], Han et al. [41], Hunt et al. [55,56], Nguyen et al. [97,98], Park et al. [113,114], Shi et al. [128], Sim and Hart [130], Wang et al. [137], Branke et al. [19], Karunakaran et al. [64], Durasevic et al. [31], Freitag and Hildebrandt [35], Hart and Sim [42], Karunakaran et al. [63], Masood et al. [75], Mei et al. [79], Nguyen [84], Nguyen et al. [99], Park et al. [115,116], Riley et al. [123], Mei and Zhang [78]
Routing rule	Nie et al. [103,104]
Due date assignment rule	Baykasolu and Gken [12], Nguyen et al. [86,87,94,95]
Batch formulation rule	Geiger and Uzsoy [37], Li et al. [72]
Performance/processing time estimation	Baykasoglu [9], Mucientes et al. [83]
Inserted idle time	Yin et al. [142]
Non-delay factor	Nguyen et al. [88,89]
Improvement/greedy heuristic	Nguyen et al. [85], Vazquez-Rodriguez and Ochoa [136], Mascia et al. [74], Nguyen et al. [88,92,93]
Others	Alsina et al. [3], Baykasoglu and Ozbakr [10], Belisrio and Pierreval [15], Furuholmen et al. [36], Li et al. [73]

rules for flexible job shop scheduling and use the least waiting time assignment [49] to find a suitable machine to process an operation. Similarly, Pickardt et al. [117] evolved dispatching rules for semiconductor manufacturing and used two existing heuristics, i.e. minimum batch size (MBS) and larger batches first (LBF), to control batch formulation. Nguyen et al. [94] used GP to evolve the due date assignment rules while fixing the dispatching rules.

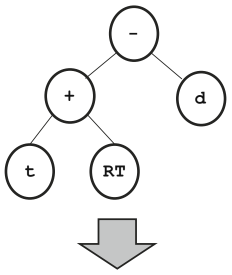
A limited number of studies focused on multiple components simultaneously. For example, Nie et al. [103,104] considered both dispatching rules and routing rules simultaneously when designing scheduling heuristics for flexible job shop scheduling. Nguyen et al. [95] developed a GP technique to deal with both sequencing decisions and due date assignment decisions. Yin et al. [142] aimed at designing predictive scheduling heuristics using GP to evolve two components, i.e. a dispatching rule and an estimation function to calculate the inserted idle time. Although the above studies showed benefits of evolving multiple components together, it may not be always the case. In the research on order acceptance and scheduling, Park et al. [111] showed that evolving both acceptance rules and dispatching rules is less effective than only focusing on dispatching rules and using a simple rule to reject orders. Therefore, selecting components to be evolved will be problem specific and depends on costs and benefits of the selection.

Representations, function sets, and terminal sets

After determining which component(s) will be evolved by GP, the next critical step is to select the suitable representation(s) for the component(s). In this section, we describe the most popular GP representations employed in the previous studies, especially those used to represent the dispatching rules because they are the main focus of previous studies (as shown in Table 2).

Evolving priority function

The most popular representations for scheduling heuristics are those used to evolve the priority functions. An example of this representation is presented in Fig. 5. In this example, the well-known minimum slack (MS) rule [119] is represented in the form of expression tree, i.e. a priority function. Based on the inputs (attributes of a job) such as the current time `t`, the remaining processing time `RT`, and the due date `d`, the function will calculate the priority of the corresponding job. Whenever a sequencing decision needs to be made at a machine (or in step 5 of Fig. 3), this function is applied to all jobs in the considered queue and the job with the highest priority will be selected to process next. Although this representation is very simple, it allows GP to explore a very diverse set of scheduling heuristics to handle different scenarios



```
Priority = - [d - (t + RT) ]
→ Priority = -[due date -(current time + remaining
              processing time)]
```

Fig. 5 Arithmetic representation

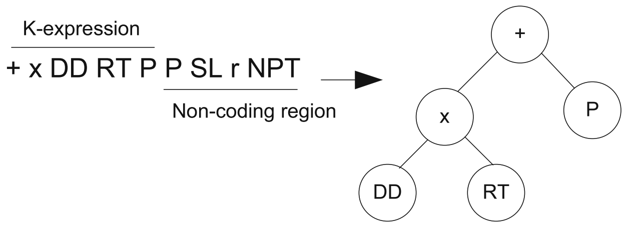


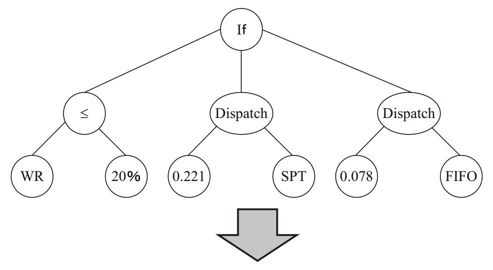
Fig. 6 GEP representation

and has shown to be effective in designing very competitive scheduling heuristics. To create heuristics, a function set and a terminal (attribute) set need to be defined. Usually basic arithmetic operators (`addition, subtraction, multiplication, and protected division`) are almost always included in the function set. Other operators such as `if, min, max` are also commonly used to evolve heuristics and have shown to be particularly useful when dealing with difficult scheduling objectives such as maximum tardiness and total weighted tardiness. The attributes used to construct scheduling heuristics can be classified as job attributes, work center attributes, and global or system attributes. A comprehensive list of attributes used in the literature can be found in [20].

The tree-based representation of the traditional GP technique [6,67] and the linear representation in gene expression programming (GEP) [34] are usually applied in the previous studies. For the tree-based GP, there are many genetic operators available in the literature [6,67]. Subtree crossover and subtree mutation are commonly used to evolve scheduling heuristics. 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 by that node with a newly randomly generated subtree.

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. 6. The gene can be translated into an expression tree using K-expression (similar to the tree-based representation in the traditional GP). 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. To ensure that a valid K-expression can be obtained, the 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. [100] showed that GEP was very competitive as compared to tree-based GP. The genetic operators in GEP can be considered as hybrids between those of genetic algorithm (GA) and the tree-based GP. The subtree crossover and subtree mutation mentioned above can also be applied to GEP. However, because of the difference in data structure (linear and 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 [103,104]. Special transposition operators are also employed in GEP to randomly select a fragment of the chromosome and insert it into the head.

Basically, other popular GP representations in the literature such as graph representations in strongly typed GP [82], Cartesian GP [80], linear GP [17], and grammar-based GP [77,139] can also be applied to evolve priority functions for scheduling heuristics. The choice of representations will depend on the requirements of scheduling heuristics. For example, Nguyen et al. [89] used grammars to evolve scheduling heuristics (as shown in Fig. 7) that can select appropriate priority functions based on the shop conditions. In this case, the proposed grammar is used to ensure that system attributes are used to set the conditions while job and work centre attributes are used to create the priority functions. Durasevic et al. [31] used dimensionally aware genetic programming [65] (similar to strongly typed GP) to improve the interpretability of scheduling heuristics by ensuring that evolved priority functions are semantically correct (e.g. the addition operator can be performed only on the nodes whose values are in the same unit). Similarly, Hunt et al. [55] designed a grammar to help evolve dispatching rules with better understandability for dynamic job shop scheduling.



```
If (workload ratio is less than or equal to 20%)
       Use the SPT rule with non-delay factor of 0.221
Else
       Use the FIFO rule with non-delay factor of 0.078
```

Fig. 7 Decision tree-like representation

Evolving multiple components

Due to the complexity of production scheduling problems, evolving a single component may not be sufficient to generate effective and comprehensive scheduling heuristics. Therefore, more sophisticated representations have been developed. Geiger et al. [38] propose a multiple tree representation to evolve different dispatching rules (trees) for different machines or groups of machines. The goal of this approach is to generate specialised rules to cope with particular requirements of each machine.

To create more effective scheduling heuristics for job shops, Jakobovic and Budin [59] presented the GP-3 method in which three program trees represent one discriminating function and two priority functions. The conceptual illustration of this representation is shown in Fig. 8. A special terminal set is used to build the discriminating function which is employed to determine whether the considered machine is 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. [88] represented the scheduling heuristics by two program trees. The first one is the priority function (the same ones described in the previous section) while the second represents the look-ahead strategy based on the Giffler and Thompson algorithm [119] to decide how much idle time machines can delay before jobs can be processed. The experiments show that these extended representations can help GP evolve significantly better scheduling heuristics as compared to those focusing only on priority functions.

Multiple tree representations are also useful for representing different scheduling decisions such as acceptance/rejection [111], due date assignment [95] and maintenance [142]. Figure 9 shows the representation of scheduling policies developed by Nguyen et al. [95]. This is motivated by the fact that scheduling and sequence decisions are directly influenced by other related production planning and control decisions such as due date assignment. The proposed representation allows the due date assignment rule and dispatching rules to be evolved at the same time, which provides the chance to optimise the overall performance of the dynamic production systems. In Yin et al. [142], a GP technique is proposed to evolve predictive scheduling heuristics to deal with stochastic machine breakdowns. The goal of the research is to handle job tardiness and stability. The two GP trees are used to represent the priority function, i.e. to determine the sequence of jobs, and to represent the idle time to be inserted before processing a particular job.

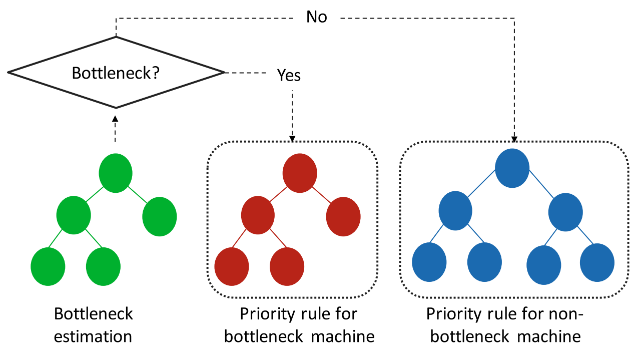

No
Bottleneck?
Yes
Bottleneck estimation
Priority rule for bottleneck machine
Priority rule for non-bottleneck machine

Fig. 8 Multiple components for bottleneck-guided dispatching rules [59]

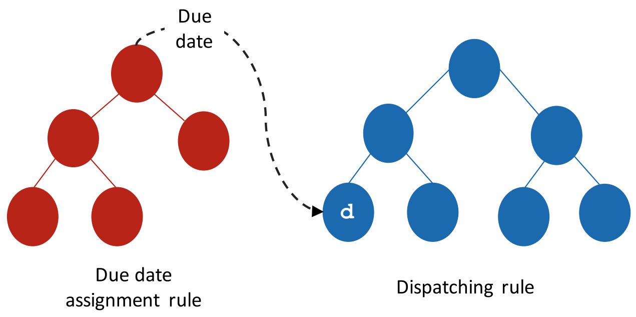


Fig. 9 Representation of scheduling policies [95]

Estimate quality of evolved scheduling heuristics

Similar to any EC methods, GP needs to estimate the quality of heuristics in its population. Here we discuss how evolved heuristics are evaluated and how fitnesses of heuristics are calculated. Recent developed techniques to improve the efficiency of GP evaluations are also presented.

Evaluation models

GP guides the search based on the quality of evolved scheduling heuristics, i.e. fitness values. To calculate the fitness function, a evaluation model or evaluator needs to be developed. Ideally, the evaluation model has to be a good representation of the real-world problems or the environment in which the obtained scheduling heuristics will be applied to. For static scheduling problems, the quality of a heuristic is determined by applying the heuristic to a set of static problem instances, obtained from real-world situations. In the literature, instances from popular benchmark datasets, e.g. OR-library [13], or randomly generated based on some assumptions [59,135], are usually applied to test the quality of different GP methods.

Meanwhile, for dynamic scheduling problems, simulation models are used to determine the steady-state performance of scheduling heuristics. Discrete event simulation (DES) [70] was the main simulation technique to estimate the performance of scheduling heuristics [46,51,95]. In previous studies, different theoretical simulation models have been employed. For example, the ten-machine symmetrical job shop model [51] is commonly used to evaluate performance of evolved dispatching rules. Although this model is relatively simple, it can reflect important characteristics of job shops (which is suitable for studies on scheduling decisions) and its scale is reasonable for evaluation purpose. More complex simulation models such as simulation models of semi-conductor production systems [118] have also been used to evaluate scheduling heuristics.

To deal with real-world dynamic production systems, it is recommended that the evaluation or simulation models should be developed by the researchers after carefully investigating the real systems. Essential steps for simulations studies can be found in [70]. Before incorporating the simulation models into the GP framework (in Fig. 2), following steps (adopted from [70]) are expected to be done by the researchers:

– Formulate the problem.
– Collect data and define a model.
– Check if the model assumptions are correct and complete.
– Construct and verify a computer program (simulator).
– Make pilot runs and validate the programmed model.

To ensure that the performance of evolved heuristics is accurately estimated, the simulation model needs to be a good representation of the real system. Otherwise, evolved rules are not applicable. Fortunately, techniques in computer simulation has been quite mature and useful tools are available to help researchers develop and validate their models.

Although DES has been shown to be a better way to evaluate the scheduling heuristics in dynamic environments, it is computationally much more expensive. As thousands of evolved scheduling heuristics need to be evaluated by GP, time-consuming simulation will dramatically increase the running times of GP. In the next sections, we will discuss some techniques that can be used to efficiently utilise the computational budget.

Table 3 Performance measures of scheduling heuristics

Mean flowtime	$F = \frac{\sum_{j\in\mathbb{C}} f_j}{\lvert\mathbb{C}\rvert}$
Maximum flowtime	$F_{\max} = \max_{j\in\mathbb{C}}\{f_j\}$
Percentage of tardy jobs	$\%T = 100 \times \frac{\lvert\mathbb{T}\rvert}{\lvert\mathbb{C}\rvert}$
Mean tardiness	$T = \frac{\sum_{j\in\mathbb{T}}(C_j - d_j)}{\lvert\mathbb{T}\rvert}$
Maximum tardiness	$T_{\max} = \max_{j\in\mathbb{T}}\{C_j - d_j\}$
Makespan	$C_{\max} = \max_{j\in\mathbb{C}}\{C_j\}$
Total weighted tardiness	$\text{TWT} = \max_{j\in\mathbb{T}}\{w_j \times (C_j - d_j)\}$

Fitness function

Table 3 shows some common performance measures of scheduling heuristics in the literature. Following are the definitions of notations used in Table 3:

– r_j: the release time when job j is available to be processed.
– w_j: the weight of job j in the weighted tardiness objective function.
– d_j: the due date assigned to job j.
– C_j: the completion time of job j.
– f_j: the flowtime of job j calculated by $f_j = C_j - r_j$.
– T_j: the tardiness of job j calculated by $T_j = \max(C_j - d_j, 0)$.
– $\mathbb{C}$: the collection of jobs recorded to calculate the objective values. ($\mathbb{C}$ is all the jobs in static JSS problem instances or a set of jobs recorded after the warm-up period of the simulation of the dynamic job shops).
– $\mathbb{T} = \{j \in \mathbb{C} : C_j - d_j > 0\}$: the collection of tardy jobs.

The quality of evolved scheduling heuristics depends on its corresponding performance measure under interested shop conditions. The application of a scheduling heuristic $\mathcal{H}$ to a number of training instances (static instances or simulation replications) $T = \{1, 2, \ldots, |T|\}$ results in performance measures $z_i(\mathcal{H})$, the objective value reached by the heuristic on instance i. These measures have to be integrated by means of a fitness function $fitness(\cdot)$ to determine the overall fitness of the heuristic. The following fitness functions have been proposed in the literature:

– Sum [or average] of objective values
$fitness(\mathcal{H}) = [\frac{1}{|T|}]\sum_{i=1}^{|T|} z_i(\mathcal{H})$
– Average relative objective value
$fitness(\mathcal{H}) = \frac{1}{|T|}\sum_{i=1}^{|T|} \frac{z_i(\mathcal{H})}{z_i(\text{ref})}$
– Sum [or average] of relative deviations
$fitness(\mathcal{H}) = [\frac{1}{|T|}]\sum_{i=1}^{|T|} \frac{z_i(\mathcal{H})-z_i(\text{ref})}{z_i(\text{ref})}$

where $z_i(\text{ref})$ denotes a reference objective value for instance i, obtained by some other solution method. Depending on

Decision situation	Attribute Set			Reference Rule -(2PT+WINQ+NPT)	Rule 1 (-PT/NPT-WINQ)		Rule 2 (-PT*NPT-WINQ)	
	PT	*WINQ*	*NPT*	Ranking	Ranking	Decision Vector	Ranking	Decision Vector
1	10	50	5	**2**	3	3	1	2
1	15	20	10	**1**	2		2	
1	30	14	12	**3**	1		3	
2	13	44	5	**2**	2	1	1	2
2	22	6	9	**1**	1		2	
…	…	…	…	**…**	…	…	…	…
K	6	30	8	**1**	4	3	1	1
K	25	25	15	**4**	3		3	
K	20	15	22	**3**	1		4	
K	15	15	7	**2**	2		2	

Fig. 10 Decision vectors used in the surrogate models [45]

the practical requirements, the researcher may choose the suitable fitness function. Sum (or average) of objective values concentrates on performing well on problem instances with a large potential for improvement while largely ignoring their performance on other instances. On the other hand, the average relative objective value or the sum (or average) of relative deviations try to measure the quality of evolved heuristics weighted by the difficulty of training instances. This fitness function is less opportunistic as compared to the fitness function based on sum of objective values.

As discussed previously, the evaluations of scheduling heuristics can be very time consuming, a full evaluation with a large number of training instances to calculate the fitness function is slow. Because GP usually requires a large population (hundreds to thousands of individuals), full evaluations for the whole population could be computationally too expensive. Therefore, most past studies did not use full evaluations to calculate the fitness function and replaced them with much cheaper evaluations as discussed in "Evaluation models". Hildebrandt et al. [46] showed that it is possible to evolve effective dispatching rules for dynamic job shop using a single simulation replication per generation. The random seed for the simulation will be changed in each generation to improve the diversity in the population. They also gave high penalties for heuristics causing instability in the simulated shop as they will slow down the evaluation process and usually are bad scheduling heuristics. This is particularly true in early generations of GP since the chance to generate bad scheduling heuristics are very high.

Here are a number of techniques proposed to reduce the computational times of GP for automated heuristic design for production scheduling:

– Early termination of the simulation: stop the simulation when the number of jobs in the system exceed some predefined threshold [46].
– Use a small number of simulation replications but change the random seed for each replication when moving to a new generation [46,99].
– Avoid evaluating the same evolved rules [45].
– Surrogate models: reduce the evaluation costs caused by expensive simulation [45,99].

Surrogate-assisted models

Recently, surrogate models have been proposed to reduce the computational costs of GP [45,96,99]. These models have reduced the evaluation costs of GP and improved its convergence. Hildebrandt and Branke [45] proposed a 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). Figure 10 gives an example of how decision vector is determined. First, a reference rule (e.g. -2PT-WINQ-NPT) is selected and applied to all decision situation. 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 rank obtained by the evolved priority function is 1. In Fig. 10, the decision vectors for rule 1 and rule 2 are $\langle 3, 1, \ldots, 3\rangle$ and $\langle 2, 2, \ldots, 1\rangle$, respectively. An archive is used to store past explored rules and their decision vector and are recorded during GP evolution. During the reproduction process, the fitness of a new 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 good estimation of fitness and help screening out bad rules created by crossover and mutation.

Also trying to reduce the computational times of GP for automated heuristic design [99] proposed a new technique to estimate the fitness of evolved rules using a simplified version of the original simulated shop, as shown in Fig. 11. Instead of evaluating evolved heuristics with the model of

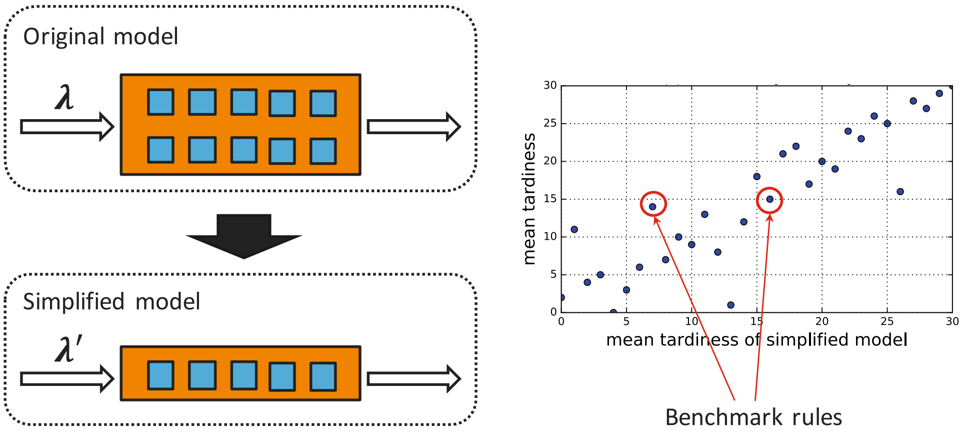


Fig. 11 Simplified models of the original simulated shop [99]

the original shop which can be very large, a smaller model of the shop is created with a smaller number of machines, smaller 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 the highest estimated fitness are moved to the next generation and estimated by the original model. The proposed GP technique based on this simplified model showed better results as compared to other GP techniques.

In general, full evaluations to calculate the real fitness for each evolved scheduling heuristics are too expensive. Therefore, it is important to utilise evaluations efficiently within the restricted computational budget. Here are some fitness functions (classified by their accuracy and usage) which have been employed in the literature:

– Real fitness function: requires a lot of simulation replications and it is the most expensive fitness function. This should be used to validate the performance of selected evolved heuristics.
– Generational fitness function: identify the most potential heuristics for real fitness evaluations. The generational fitness function can use a small number of replications to reduce the computational costs but new training instances are used for each generation to prevent GP from overfitting and improve the diversity of the population [46].
– Fitness estimated by surrogate models: determines the rough quality of generated heuristics. Because the likelihood to produce bad heuristics via crossover and mutation by GP is very high, GP may waste a lot of time evaluating bad heuristics. The fitness estimated efficiently by surrogate models [45,99] helps the algorithm screen out heuristics with poor performance and reduce the computational costs as well as improve the convergence of GP.

Figure 12 illustrates the trade-offs between the accuracy and computational costs of different models used to evaluate the performance of evolved scheduling heuristics. In this figure, full evaluations are the one with the best accuracy and the highest computational times while evaluations with static training instances are the most efficient ones but may cause overfitting issues [46,89]. Surrogate-assisted GP can be designed to effectively use these models in the algorithm. Currently, surrogate models are only used as the pre-selection strategy [62]. Other applications of surrogate models in GP can be also investigated in future studies (e.g. individual-based, generation-based and population-based techniques).

Search mechanisms

Most GP techniques proposed in the literature for automated design of production scheduling heuristics imitate Darwinian biological evolution by maintaining and evolving a large population. Genetic operators inspired by natural evolution such as crossover, mutation, and elitism (as discussed in "Representations, function sets, and terminal sets") are used to generate new individuals. Despite its simplicity, this mechanism is able to discover very effective scheduling heuristics. However, to deal with more complicated design issues such as multiple scheduling decisions and multiple conflicting 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 GP for designing heuristics is that their search mechanisms are very flexible and many advanced EC techniques [27,61,133] have been developed to cope with multiple objectives.

Tay and Ho [135] aimed to tackle three objectives (makespan, mean tardiness, and mean flowtime) when using

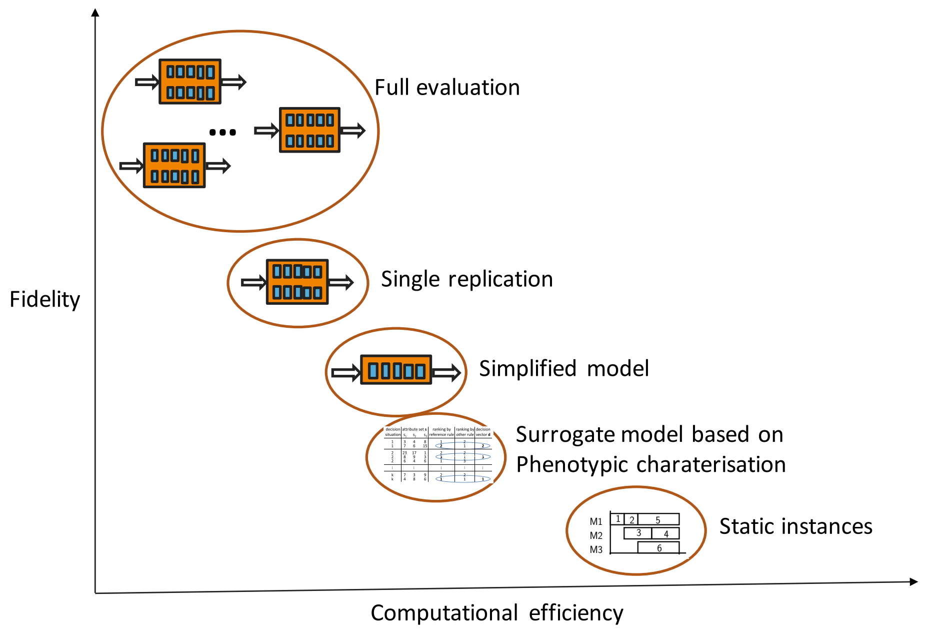


Fig. 12 An illustration of a trade-off between fidelity (approximation accuracy) and computational cost for evaluations of scheduling heuristics (adopted from [62])

GP to evolve dispatching rules for a flexible job shop. To simplify the design problem, the three objectives are aggregated using the weighted sum approach with the same weight for each objective. However, because the scale of each objective and the knowledge about the objective search space may be unknown in advance, this approach can lead to unsatisfactory results. For this reason, the rules evolved by their GP method are sometimes worse than simple rules such as FIFO Freitag and Hildebrandt [135]. When these evolved rules are examined in a long simulation [46], they are only slightly better than the earliest release date (ERD) rule and worse than the SPT rule with respect to mean tardiness. It 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 individual objective is not available.

Nguyen et al. [90] developed a multi-objective genetic programming-based hyper-heuristic (MO-GPHH) for dynamic job shop scheduling. In this work, the goal is 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 showed 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 ATC, RR, 2PT+WINQ+NPT, and COVERT [125] in most of its runs. The experimental results showed that the obtained Pareto front contains many dispatching rules with good trade-offs that have not been explored earlier in the literature (e.g. percentage of tardy jobs %T can be reduced greatly without significantly deteriorating other objectives). Similar observations for a complex semiconductor manufacturing system are found by [35]. Thus, evolving the Pareto front is more beneficial as compared to evolving a single rule generally. Similar methods have been applied to evolve comprehensive scheduling policies for dynamic job shop scheduling [95] and order acceptance and scheduling [84] and showed promising results.

Masood et al. [75] proposed to combine the advantage of GP and NSGA-III to evolve a set of Pareto-optimal dispatching rules for many-objective job shop scheduling. The proposed algorithm uses the tree-based representation and evolutionary operators of GP and the fitness assignment scheme (i.e. non-dominated sorting and reference points) of NSGA-III. They further extended their work [76] by taking the discrete and possibly non-uniform Pareto front into account. To search for the non-uniform Pareto front more efficiently, they proposed a scheme to adaptively adjust the positions of the reference points using particle swarm optimisation.

Coevolution

Miyashita [81] proposes three multi-agent learning structures based on GP to evolve dispatching rules for dynamic job shop scheduling. The first one is a homogeneous agent model, which is basically the same as other GP techniques which evolve 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 co-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 the bottleneck agent and the 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. [95] develop a GP-based cooperative coevolution approach in which scheduling rules and due date assignment rules are evolved in their own subpopulation. Similar to Miyashita [81], 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 compared to 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.

In another study, Beham et al. [14] utilised parallel technologies to evolve dispatching rules for a flexible job shop with a large terminal and function sets. They developed three new GP methods based on island models and SASEGASA [2] in which rules are evolved in multiple subpopulations. The results show that the SASEGASA method can cope better with the states of exception in the simulation than island-based methods. In a very recent study, Karunakaran et al. [63] investigated GP with different topologies of the island model to deal with multi-objective job shop scheduling. Their experimental results showed that the proposed techniques outperform some general-purpose multi-objective optimization methods, including NSGA-II and SPEA-2.

Park et al. [116] proposed two GP techniques to evolve ensembles of dispatching rules based on Multilevel Genetic Programming (MLGP) [140] and cooperative coevolution [121]. While MLGP aims at automatically finding a group of individuals that work together effectively, the cooperative coevolution uses decomposition approaches to coevolve multiple subpopulations. The experimental results showed that MLGP outperformed the simple GP technique with no significant increase in computational times. Meanwhile, the cooperative coevolution technique are better than MLGP in terms of performance of evolved rules but significantly slower than MLGP.

Other search mechanisms

Nguyen et al. [97] developed a new technique called automatic programming via iterated local search (APRILS) to design dispatching rules for dynamic job shop scheduling. APRILS used tree-based representation similar to GP but it employed iterated local search to search for the best rule instead of population-based search in most studies. In the proposed algorithm, the neighbour heuristics are created by applying subtree mutation (but only small random subtree is generated). To help the algorithm escape from the local optimum, subtree mutation and subtree extraction operators are used. Given a fixed number of fitness evaluations, the experimental results showed that APRILS is significantly better than simple GP with the tree-based representation in terms of performance of evolved heuristics, program lengths, and the running times.

Hart and Sim [42] developed a hyper-heuristic based on an ensemble method called NELLI [131]. The proposed NELLI-GP extends NELLI by evolving a set of dispatching rules represented as an expression tree. NELLI uses a method inspired by Artificial Immune Systems [25] to evolve a set of heuristics, which are behaviourally diverse in the sense that each solves different subsets of a large instance set. The three main components of NELLI-GP are: (1) a heuristic generator to generate new scheduling heuristics, (2) sets of problem instances, and (3) a network inspired by the idiotypic network theory of the immune system. The key idea is to evolve ensembles of heuristics that interact to cover a problem space. The experiments showed that NELLI-GP can produce promising results.

Pickardt et al. [118] proposed a two-stage approach to evolve 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) is used to select the most suitable dispatching rule in the set of candidate rules for each work centre in the shop. The experiments compared 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.

Post-processing

The evolved scheduling heuristics are usually large in size and it is not straightforward to understand how and why scheduling decisions are made. Post-processing steps are usually included to analyse the obtained heuristics to understand how they handle scheduling problems. Following are some post-processing techniques to analyse scheduling heuristics commonly used in the recent literature:

– Simplification of obtained heuristics.
– Visualisation.
– Analyse feature usage within obtained heuristics.
– Analyse code fragment.
– Relearn obtained heuristics.

Simplification is the most popular technique to remove redundant parts of evolved heuristics, which make the heuristics smaller and easier to understand [89]. Manual simplification is usually applied to the best evolved scheduling heuristics [30,135]. Often the length of evolved scheduling heuristics can be significantly reduced via manual simplification since there are some parts that make no contributions to the outputs (e.g. some conditions are always true). Symbolic simplification function available in some mathematical softwares can also be used. Nguyen et al. [99] applied a numerical simplification routine that transverses the evolved tree and check if the performance of the heuristic will be deteriorated as the considered subtree is reduced to some constant.

Visualisation is also an attractive alternative to interpret the evolved scheduling heuristics. Branke et al. [18] used contour plot to visualise priorities as the functions of attributes. Nguyen et al. [99] used parallel coordinate plot which is able to show how priorities change with different combinations of attributes. These visualisation techniques are helpful to show the general characteristics of the evolved heuristics and the differences between different heuristics. However, it is still hard to fully understand the complex behaviours of evolved scheduling heuristics.

Analysing the usages of attributes included in the evolved scheduling heuristics has also been done in the literature to understand the contributions of these attributes. For example, Nguyen et al. [89] analyses the frequency usage of attributes in the final dispatching rules evolved by GP to show what are the most useful attributes. Branke et al. [18] analysed the importance of each attribute by measuring the performance of the best rules when certain attributes are not available. Their analyses show that some attributes are more important for specific representations. Instead of independently investigating each attribute, Hunt et al. [55] performed fragment analyses to show the most common fragments (with the depth of two) in the evolved scheduling heuristics. The analyses show that some fragments representing the differences between due date, machine ready time, and current time appear most often in the evolved heuristics. The analyses also showed that most frequent fragments from different GP techniques can be different.

Since the evolved heuristics are usually complicated, Nguyen et al. [99] attempted to apply supervised machine learning techniques to relearn the scheduling heuristics obtained by GP. Random sampling are applied to randomly pick pairs of jobs and decide which one has the higher priority based on the heuristics to be relearned. From the collected data, a binary classification problem is created. In this problem, the attributes are the relative attributes of the two jobs and the label is whether or not the first job has a higher priority than the second one. Decision tree has been applied as the obtained decision tree is easy to understand and more important attributes can be easily detected (usually in the top of the decision tree).

Evaluating GP methods

The performance of a GP method is measured by the performance of the evolved scheduling heuristics. Similar to traditional studies in the scheduling literature, scheduling heuristics are evaluated based on the quality of obtained scheduling solutions (usually the average objective value from a set of test problem instances), the robustness (i.e. the test performance in unseen scenarios), and the computational times. Well-known benchmark instances in the scheduling literature [4,29,71,132] are commonly used for evaluation purposes. Some random instance generators [42,59,135] are also applied to generate training and test instances for GP. However, these are mainly used for static production scheduling problems. For dynamic stochastic scheduling problems, DES (as discussed in“Evaluation models”) is typically applied. Most DES simulators are developed by researchers to cope with their GP systems and specific research objectives. [20] suggested that the publication of entire simulators (e.g. Jasima from [44]) would greatly help replicability and facilitate fair comparisons.

For a new application of GP, it is important to compare evolved scheduling heuristics with the state-of-the-art heuristics in the literature to demonstrate its effectiveness [46,59, 89,135]. As discussed in “Evolutionary multi-objective optimisation”, these comparisons can reveal interesting insights about evolved heuristics [90]. When comparing different GP methods, the average (relative) objective values of obtained scheduling heuristics are the primary performance measures. The complexity of evolved heuristics, i.e. often measured in terms of the lengths of GP individuals, is also used to compare GP methods [97,99]. Interpretability has been recently investigated when comparing different hyper-heuristics methods [18,55]. For multi-objective scheduling problems, common EMO metrics such as hyper-volume and inverted generational distance can be used to measure the quality of the obtained trade-off heuristics [75,95].

In this section, we have discussed key components for automated design of scheduling heuristics with GP and showed how these components are connected under a unified framework. For each component, the basic setting as well as the more advanced techniques developed for complex situations have been presented. The new techniques have been developed based on the needs of discovering more effective and interpretable scheduling heuristics while reducing the computational times. Although there have been many fruitful studies in the last five years, many issues still remain unsolved and require more studies in the future.

Connections with other artificial intelligence (AI) and operations research (OR) techniques

Automated heuristic design is a relatively new area of research and has attracted much attention of many researchers in AI and OR. In previous studies, both machine learning and operations research techniques have been applied within automated heuristic design. In the rest of this section, we discuss different ways that machine learning and operations research can be used to enhance the way GP evolves production scheduling heuristics.

Machine learning

In most previous studies, GP is used as a *unsupervised learning* technique to learn the most effective heuristics for scheduling problems. GP has to discover both the heuristic structures as well as the corresponding parameters. From this viewpoint, automated heuristic design can be simply treated as an optimisation problem where the objective function is the fitness function to evaluate the quality of heuristics. As the search space of GP is very large, searching for (near) optimal heuristics is very challenging and it is even more difficult if there are many attributes or features to be considered (in the terminal set). To deal with this issue, feature selection are needed to remove redundant attributes which may influence the performance of GP. Mei et al. [79] has shown that selecting a good feature subset can significantly improve the performance of GP, i.e. finding better scheduling heuristics. Feature construction [52] and feature extraction will be also an interesting aspects that need to be considered in the future studies.

Supervised learning techniques such as decision tree [107,127], logistic regression [57], support vector machines [129], and artificial neural networks [32,138] have also been investigated in the literature for automated design of production scheduling heuristics. For supervised learning, optimal decisions from solving small instances with exact optimisation methods or from the historical data are needed to build the training set. However, there are a number of challenges with supervised learning. As scheduling decisions are highly interdependent (i.e. the decision for an operation may influence decision of other operations), learning the optimal decisions for the whole schedule will not be easy. If the goal is to determine which dispatching rules to apply given a set of jobs and system status, there is also no guarantee that the learned heuristics will actually provide the (near) optimal solution as the available dispatching rules may not be effective (similar to the cases when historical data are used). Nguyen et al. [93] proposed a sequential GP to learn a set of rules that can learn optimal scheduling decisions for order acceptance and scheduling problem. The training set is a number of decision situations which the optimal decisions obtained by exact methods. The obtained heuristics are very efficient and are competitive as compared to customised meta-heuristics developed in the literature. Combining the power of advanced supervised machine learning techniques with GP would be an interesting research direction in the future.

The scheduling literature has covered a wide range of scheduling problems. In production scheduling, many popular problems have been investigated intensively such as single machine scheduling, parallel machine scheduling, (flexible) flow shop scheduling, (flexible) job shop scheduling, and open shops. There is shared knowledge between these problems which can be used to develop different heuristics (e.g. ATC can be extend to ATCS to deal with setup dependent scheduling problems). Clearly, *transfer learning* [7,33] and *multi-task learning* [8,40,108] will be very useful in automated design of production scheduling heuristics. The knowledge to solve a simple scheduling problem can be reused to solve hard problems and scheduling jobs at different machines can be based on some common pieces of knowledge.

Operations research

Discrete event simulation (DES) has been used intensively in automated design of production scheduling heuristics and it is proven to be an effective method to evaluate the performance of scheduling heuristics, especial in dynamic environments. DES is also very flexible which allows it to model a wide range of complex real-world problems and to be embedded into GP. In terms of simulation, many aspects should be considered to improve the evaluation accuracy and efficiency:

- *Continuous simulation* In some cases, production systems need continuous simulation method [70] as the states of the system change continuously [48] like the movement of liquids (e.g. oil, chemical) or the steel making process. Scheduling with the continuous production process is an interesting topic and continuous simulation or hybrid simulation combining both DES and continuous simulation will be useful (e.g. in food industry).

– *Multi-agent simulation* To gain better understandings of the system and investigate how individual behaviours may influence the overall performance, multi-agent simulation, a powerful technique in OR and AI, will be a more suitable method. Miyashita [81] investigated different GP-based agent models and showed interesting preliminary results. In the future studies, different real-world aspects (e.g. human factors) should be considered to see how GP-based agents will behave.
– *Simulation optimisation* Automated design of production scheduling heuristics can be treated as simulation optimisation problems as the fitness of heuristics is stochastic. Then, many advanced techniques developed in simulation optimisation can be applied in this case to improve the accuracy of fitness evaluations and improve the efficiency of GP.

Queueing theories and stochastic models of production systems have been studied intensively in the last few decades and it would be useful to incorporate the knowledge from these research fields into the automated design process. For an example, useful scheduling policies developed for the stochastic environments or policies to cope with the machine breakdowns can be considered when developing the meta-algorithms of scheduling heuristics (see "Meta-algorithm of scheduling heuristics"). Similarly, to build a more competitive scheduling heuristics, the advances in the scheduling literature and combinatorial optimisation need to be taken into account.

Current issues and challenges

There are many issues that are worth considering in the future studies. Here we point out three key issues needed to be addressed if we want to apply automated heuristic design in practice.

Dynamic changes

Dynamic changes are unavoidable in the real-world applications and coping with this issue is essential. Traditional optimisation methods could not handle dynamic change well. Fortunately, scheduling heuristics evolved by GP can cope very well with the dynamic changes. Basically, these heuristics can deal easily with most dynamic changes such as dynamic job arrivals, machine breakdowns, and stochastic processing times. However, to improve the quality as well as the robustness of evolved heuristics, the problem domain knowledge is needed. Either the knowledge is provided to GP by the researchers or automatically extracted from the environment will need further investigation.

Strategies for dynamic scheduling in production systems [109] can be classified as:

– *Completely reactive scheduling* no schedule is generated in advance and decisions are made in real time. Priority dispatching rules are the main techniques for completely reactive scheduling.
– *Predictive-reactive scheduling* scheduling/rescheduling is triggered by the real-time events where both objectives of interest and stability (measured by the deviation from original schedule) are considered.
– *Robust pro-active scheduling* focus on building predictable schedules; the key idea is to improve the predictability of the schedules in a dynamic environment (e.g. by inserting additional time in the predictive schedule) with minimal effects on the schedule performance.

While GP has been applied to many studies to evolve priority rules, there is no study on predictive-reactive scheduling. Yin et al. [142] and Nguyen et al. [95] are the only two studies that considered pro-active scheduling issues. Yin et al. [142] tried to evolve a scheduling heuristics that include a priority rule to determine the job sequence and a function to estimate the idle time needed to be inserted into the schedule to buffer against stochastic machine breakdowns. Their objective was to minimise both the mean tardiness of the schedule and the deviations between initial and final schedules. In this case, Yin et al. [142] proposed a fitness function that is a weighted sum of mean tardiness and mean deviations of completion times. Nguyen et al. [95] aimed at coevolve both the dispatching rules and due date assignment rules to optimise the scheduling performance measures and minimise the deviations between the realised completion times and assigned due dates of jobs. Different from Yin et al. [142], Nguyen et al. [95] used evolutionary multi-objective optimisation to evolve the set of non-dominated scheduling policies. GP studies in this research direction are still at a very early stage and many things will need to be done such as improving stability and predictability of schedules, handling different sources of disturbances, and improving the efficiency of GP and its evolved heuristics.

Multiple decisions

Previous studies on production planning and scheduling focused mainly on scheduling or sequencing decisions and assumed that other related decisions are fixed. For example, the due date assignment rules (e.g. total work content) and the job release policy (e.g. immediate release) are fixed and we try to find the best scheduling heuristics. These assumptions reduce the computational burden of the optimisation or learning techniques, but they also restrict us from developing an effective comprehensive systems. It should be noted that past studies are limited by manual designs of scheduling heuristics and computational power, which is not a serious issue now. The automated heuristic design and the growing computing power provide us with the chance to consider a much wider scope.

However, to effectively handle multiple decisions, better search mechanisms will need to be developed. There are two common approaches to dealing with multiple decisions. The first approach creates a sophisticated representation that contains two or more decision rules and programs based on this representation are evolved and reproduced based on some customised genetic operators. The second approach is to apply cooperative coevolution technique to coevolve multiple subpopulations for multiple decision rules. In these two approaches, each rule is constructed based on an independent set of terminals and functions. Since each decision rule has its own characteristic, it is not necessary to use GP to evolve all the rules. For example, supervised learning (e.g. regression) can be used to estimate due dates of randomly arriving jobs given a fixed dispatching rule. It would be useful to investigate how GP can be combined with other machine learning techniques to deal with multiple decisions in production planning and scheduling.

Multiple conflicting objectives

Similar to multiple decisions, GP allows the researchers to cope with multiple conflicting objectives in various ways. Using pure EMO search mechanisms such as NSGA-II [28] may have troubles dealing with this designing problems because of a number of reasons. First, the search space for GP to explore can be very large because of the number of terminals, functions sets, and the number of objectives to be optimised. Therefore, it will be much harder for the search methods to find a good set of non-dominated heuristics. Second, GP usually requires a large population to maintain a large and diverse genetic materials to create effective scheduling heuristics, especially when dealing with multiple conflicting objectives. As a result, more heuristic evaluations will be needed, which significantly increases the computational costs of GP. Finally, it will be more difficult to understand how the trade-offs are achieved via the evolved heuristics (it has been already very hard to understand scheduling heuristics in the case of optimising a single objective). In addition, how to measure the robustness of scheduling heuristics (when they are applied to different/unseen scenarios) is still a open research question.

Possible approaches to handle these issues are:

– Developing more specialised genetic operators and local search heuristics to improve the search effectiveness and efficiency.
– Incorporating user's preferences to guide the search of GP to improve the efficiency of GP.
– Developing new surrogated assisted GP to reduce the computational costs of GP and improve its effectiveness.
– Developing new representations to allow GP to deal with multiple objectives effectiveness and improve the interpretability of evolved scheduling heuristics.

Other challenges

Developing an efficient, effective, and scalable GP systems for evolving scheduling heuristics will continue to be a major challenge for the research community. As the real-world production systems can be very complicated with many different types of resources and technical constraints, many attributes need to be considered to construct heuristics and the search space of scheduling heuristics can be very large. The key point for future research is to enhance the search mechanism of GP so that GP is able to evolve effective sophisticated structures of scheduling heuristics and optimise their related parameters for complex production environments. As mentioned earlier, transfer learning can be an interesting research topic for GP to reuse the knowledge obtained from handling different scheduling problems.

Conclusions

Automated design of production scheduling heuristics is an interesting and challenging research area which has a lot of potential applications. GP has been the most popular technique for the automated design tasks in the last several years. Different from existing survey papers that focused on general ideas and taxonomies, the goal of this paper is to emphasise on the technical issues when using GP to evolve production scheduling heuristics. In this paper, we discussed the key issues related to automated design of scheduling heuristics with GP including meta-algorithms of scheduling heuristics, selection of component(s) to be evolved, representations, evaluation models, fitness functions, search mechanism and post-processing. A unified framework was developed to provide beginning researchers with an overall picture of all essential steps, components, and their connections when developing a GP system for automated design of production scheduling heuristics. Through analyses of each component, we also pointed out the strength and weakness of each technique proposed in the literature and provided hints for future studies. Representations, evaluation models, and post-processing are still three main research directions to be explored as they can directly influence the applicability of these techniques in practice. Researchers from GP communities can develop better representations and genetic operators to help GP discover more powerful and more interpretable scheduling heuristics. Advanced knowledge from the fields of simulation and optimisation of expensive functions can be very useful when systematically applied to GP. For post-processing, it is a space for creativeness, in which the goal is to explain how the evolve how discovered heuristics work, its sensitivity, and the reliability of decisions made by the heuristic.

Automated design of production scheduling heuristics is a multi-disciplinary and inter-disciplinary research area where the knowledge from operations research and artificial intelligence is required. Scheduling has its root from operation research and many clever techniques have been proposed in the literature. It would be interesting to see how GP can assist to make scheduling research more productive. For AI, automated heuristics design will greatly enlarge the scope of machine learning applications from traditional prediction tasks to making optimal decisions based on historical operational data. In addition, many aspects from supervised learning, unsupervised learning, and transfer learning will need to be investigated in the context of automated heuristic design.

Production environments can be complex and it is critical for GP to handle key issues that commonly occur in real-world situations such as dynamic changes, multiple decisions, and multiple conflicting objectives. Although many issues can be handled directly by GP (e.g. reactively dealing with dynamic changes), some have not been investigated or have not had a satisfactory solutions. In addition, many aspects discussed here will be true for other scheduling and combinatorial optimisation problems and we expect that more applications will appear in the near future.

References

1. Abednego L, Hendratmo D (2011) Genetic programming hyper-heuristic for solving dynamic production scheduling problem. In: IEEE 2011 international conference on electrical engineering and informatics (ICEEI)
2. Affenzeller M, Wagner S (2004) SASEGASA: a new generic parallel evolutionary algorithm for achieving highest quality results. J Heuristics 10(3):243–267
3. Alsina EF, Capodieci N, Cabri G, Regattieri A, Gamberi M, Pilati F, Faccio M (2015) The influence of the picking times of the components in time and space assembly line balancing problems: an approach with evolutionary algorithms. In: 2015 IEEE symposium series on computational intelligence, pp 1021–1028
4. Applegate D, Cook W (1991a) A computational study of the job-shop scheduling instance. ORSA J Comput 3(2):149–156
5. Applegate D, Cook W (1991b) A computational study of the job-shop scheduling problem. ORSA J Comput 3(2):149–156
6. Banzhaf W, Nordin P, Keller R, Francone F (1998) Genetic programming: an introduction. Morgan Kaufmann, San Francisco
7. Baxter J (1998) Theoretical models of learning to learn. In: Thrun S, Pratt L (eds) Learning to learn. Springer US, Boston, pp 71–94. doi:10.1007/978-1-4615-5529-2_4
8. Baxter J (2000) A model of inductive bias learning. J Artif Int Res 12(1):149–198
9. Baykasoglu A (2008) Gene expression programming based meta-modelling approach to production line design. Int J Comput Integr Manuf 21:657–665
10. Baykasoglu A, Ozbakr L (2015) Discovering task assignment rules for assembly line balancing via genetic programming. Int J Adv Manuf Technol 76:417–434
11. Baykasoglu A, Gocken M, Ozbakir L (2010) Genetic programming based data mining approach to dispatching rule selection in a simulated job shop. SIMULATION Trans Soc Model Simul 86:715–728
12. Baykasolu A, Gken M (2009) Gene expression programming based due date assignment in a simulated job shop. Expert Syst Appl 36(10):12143–12150
13. Beasley JE (1990) Or-library: distributing test problems by electronic mail. J Oper Res Soc 41(11):1069–1072
14. Beham A, Winkler S, Wagner S, Affenzeller M (2008) A genetic programming approach to solve scheduling problems with parallel simulation. In: Wu J, Robert Y (eds) Proceedings of the 2008 IEEE International Parallel & Distributed Processing Symposium. IEEE Computer Society Press, Los Alamitos, CA, pp 1–5
15. Belisrio LS, Pierreval H (2015) Using genetic programming and simulation to learn how to dynamically adapt the number of cards in reactive pull systems. Expert Syst Appl 42(6):3129–3141
16. Bierwirth C, Mattfeld DC (1999) Production scheduling and rescheduling with genetic algorithms. Evol Comput 7(1):1–17
17. Brameier MF, Banzhaf W (2010) Linear genetic programming, 1st edn. Springer Publishing Company, Incorporated, Berlin
18. Branke J, Hildebrandt T, Scholz-Reiter B (2015) Hyper-heuristic evolution of dispatching rules: a comparison of rule representations. Evol Comput 23(2):249–277
19. Branke J, Groves MJ, Hildebrandt T (2016a) Evolving control rules for a dual-constrained job scheduling scenario. In: Proceedings of the 2016 Winter Simulation Conference, Winter Simulation Conference
20. Branke J, Nguyen S, Pickardt CW, Zhang M (2016b) Automated design of production scheduling heuristics: a review. IEEE Trans Evol Comput 20(1):110–124
21. Burke EK, Hyde M, Kendall G, Woodward J (2007) Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. In: GECCO '07: Proceedings of the 9th annual conference on genetic and evolutionary computation. ACM Press, New York, pp 1559–1565
22. Burke EK, Hyde MR, Kendall G, Ochoa G, Ozcan E, Woodward JR (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C, Jain L (eds) Computational intelligence, intelligent systems reference library, vol 1. Springer, Berlin, pp 177–201
23. Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Handbook of metaheuristics, international series in operations research & management science, vol 146. Springer, New York, pp 449–468
24. Burke EK, Gendreau M, Hyde M, Kendall G, Ochoa G, Ozcan E, Qu R (2013) Hyper-heuristics: a survey of the state of the art. J Oper Res Soc 64(12):1695–1724
25. Castro LRd, Timmis J (2002) Artificial immune systems: a new computational intelligence paradigm. Springer-Verlag New York Inc, secaucus

26. Chen L, Zheng H, Zheng D, Li D (2015) An ant colony optimization-based hyper-heuristic with genetic programming approach for a hybrid flow shop scheduling problem. In: CEC'15: IEEE congress on evolutionary computation
27. Coello Coello CA (1999) A comprehensive survey of evolutionary-based multiobjective optimization techniques. Knowl Inf Syst 1(3):269–308
28. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evol Comput 6(2):182–197
29. Demirkol E, Mehta S, Uzsoy R (1998) Benchmarks for shop scheduling problems. Eur J Oper Res 109(1):137–141. doi:10.1016/S0377-2217(97)00019-2
30. Dimopoulos C, Zalzala AMS (2001) Investigating the use of genetic programming for a classic one-machine scheduling problem. Adv Eng Softw 32(6):489–498
31. Durasevic M, Jakobovi D, Kneevi K (2016) Adaptive scheduling on unrelated machines with genetic programming. Appl Soft Comput 48:419–430
32. Eguchi T, Oba F, Toyooka S (2008) A robust scheduling rule using a neural network in dynamically changing job-shop environments. Int J Manuf Technol Manag 14(34):266–288
33. Feng L, Ong YS, Lim MH, Tsang IW (2015) Memetic search with interdomain learning: a realization between CVRP and CARP. IEEE Trans Evol Comput 19(5):644–658
34. Ferreira C (2006) Gene expression programming: mathematical modeling by an artificial intelligence, 2nd edn. Springer, Berlin
35. Freitag M, Hildebrandt T (2016) Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization. CIRP Ann Manuf Technol 65(1):433–436
36. Furuholmen M, Glette K, Hovin M, Torresen J (2009) Coevolving heuristics for the distributor's pallet packing problem. In: CEC'09: IEEE congress on evolutionary computation, pp 2810–2817
37. Geiger CD, Uzsoy R (2008) Learning effective dispatching rules for batch processor scheduling. Int J Prod Res 46(6):1431–1454
38. Geiger CD, Uzsoy R, Aytu H (2006) Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. J Sched 9(1):7–34
39. Giffler B, Thompson GL (1960) Algorithms for solving production-scheduling problems. Oper Res 8(4):487–503
40. Gupta A, Ong Y, Feng L (2016) Multifactorial evolution: toward evolutionary multitasking. IEEE Trans Evol Comput 20(3):343–357
41. Han S, Seo J, Park J (2012) Designing an effective scheduling scheme considering multi-level BOM in hybrid job shop. In: Proceedings of the 2012 international conference on industrial engineering and operations management, pp 1302–1310
42. Hart E, Sim K (2016) A hyper-heuristic ensemble method for static job-shop scheduling. Evol Comput. doi:10.1162/EVCO_a_00183
43. Hart E, Ross P, Corne D (2009) Evolutionary scheduling: a review. Genetic Program Evol Mach 6(2):191–220
44. Hildebrandt T (2014) Jasima an efficient java simulator for manufacturing and logistics. http://code.google.com/p/jasima/
45. Hildebrandt T, Branke J (2014) On using surrogates with genetic programming. Evol Comput 23(3):343–367
46. Hildebrandt T, Heger J, Scholz-Reiter B (2010) 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. ACM Press, Portland, Oregon, USA, pp 257–264
47. Hildebrandt T, Goswami D, Freitag M (2014) Large-scale simulation-based optimization of semiconductor dispatching rules. In: Proceedings of the 2014 winter simulation conference, pp 2580–2590
48. Hmida JB, Lee J, Wang X, Boukadi F (2014) Production scheduling for continuous manufacturing systems with quality constraints. Prod Manuf Res 2(1):95–111
49. Ho NB, Tay JC (2004) GENACE: An efficient cultural algorithm for solving the flexible job-shop problem. In: Evolutionary computation, 2004. CEC2004. Congress on, IEEE, vol 2, pp 1759–1766
50. Ho NB, Tay JC (2005) Evolving dispatching rules for solving the flexible job-shop problem. In: IEEE congress on evolutionary computation, vol 3, pp 2848–2855. doi:10.1109/CEC.2005.1555052
51. Holthaus O, Rajendran C (2000) Efficient jobshop dispatching rules: further developments. Prod Plan Control 11(2):171–178
52. Hunt R (2016) Genetic programming hyper-heuristics for job shop scheduling. PhD thesis, Victoria University of Wellington, http://researcharchive.vuw.ac.nz/handle/10063/5219
53. Hunt R, Johnston M, Zhang M (2014) Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming. In: Igel C, Arnold DV (eds) GECCO '14: Proceedings of the 2014 conference on genetic and evolutionary computation. ACM Press, New York, pp 927–934
54. Hunt R, Johnston M, Zhang M (2014) Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming. In: Liu D, Hussain A, Zeng Z, Zhang N (eds) 2014 IEEE congress on evolutionary computation (CEC). IEEE Press, Piscataway, NJ, pp 618–625
55. Hunt R, Johnston M, Zhang M (2015a) Evolving dispatching rules with greater understandability for dynamic job shop. Tech. Rep. ECSTR15-06, Victoria University of Wellington
56. Hunt R, Johnston M, Zhang M (2015b) Using Local Search to Evaluate dispatching rules in dynamic job shop scheduling. In: Ochoa G, Chicano F (eds) Evolutionary computation in combinatorial optimization. Springer International Publishing, Lecture notes in computer science, pp 222–233
57. Ingimundardottir H, Runarsson TP (2011) Supervised learning linear priority dispatch rules for job-shop scheduling. In: Coello Coello CA (ed) Learning and intelligent optimization, Springer, Berlin and Heidelberg, LNCS, vol 6683, pp 263–277
58. Jakobovi D, Marasovi K (2012) Evolving priority scheduling heuristics with genetic programming. Appl Soft Comput 12(9):2781–2789
59. Jakobovic D, Budin L (2006) Dynamic scheduling with genetic programming. In: Collet P, Tomassini M, Ebner M, Gustafson S, Ekrt A (eds) Genetic programming, Springer, Berlin, LNCS, vol 3905, pp 73–84
60. Jakobovic D, Jelenkovic L, Budin L (2007) Genetic programming heuristics for multiple machine scheduling. In: Ebner M, O'Neill M, Ekrt A, Vanneschi L, Esparcia-Alczar AI (eds) Genetic programming, Springer, Berlin, LNCS, vol 4445, pp 321–330
61. Jin Y (2006) Multi-objective machine learning (studies in computational intelligence) (studies in computational intelligence). Springer-Verlag New York Inc, Secaucus
62. Jin Y (2011) Surrogate-assisted evolutionary computation: recent advances and future challenges. Swarm Evol Comput 1(2):61–70
63. Karunakaran D, Chen G, Zhang M (2016a) Parallel multi-objective job shop scheduling using genetic programming. In: Ray T, Sarker R, Li X (eds) Artificial life and computational intelligence. Springer International Publishing, Lecture notes in computer science, pp 234–245
64. Karunakaran D, Mei Y, Chen G, Zhang M (2016b) Dynamic job shop scheduling under uncertainty using genetic programming. In: Asia-Pacific symposium on intelligent and evolutionary systems (IES) (to appear)

65. Keijzer M, Babovic V (1999) Dimensionally aware genetic programming. In: Banzhaf, W (ed) Proceedings of the first genetic and evolutionary conference (GECCO 99), Morgan, pp 1069–1076
66. Kofler M, Wagner S, Beham A, Kronberger G, Affenzeller M (2009) Priority rule generation with a genetic algorithm to minimize sequence dependent setup costs. In: Moreno-Daz R, Pichler F, Quesada-Arencibia A (eds) Computer aided systems theory EUROCAST 2009, Springer, Berlin, LNCS, vol 5717, pp 817–824
67. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
68. Kuczapski AM, Micea MV, Maniu LA, Cretu VI (2010) Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling. Inf Technol Control 39(1):32–37
69. Langdon WB, Banzhaf W (2005) Repeated sequences in linear genetic programming genomes. Complex Syst 15(4):285–306
70. Law AM, Kelton DM (1999) Simulation modeling and analysis. McGraw-Hill Higher Education, Boston
71. Lawrence S (1984) Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques. PhD thesis, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania
72. Li D, Zhan R, Zheng D, Li M, Kaku I (2016) A hybrid evolutionary hyper-heuristic approach for intercell scheduling considering transportation capacity. IEEE Trans Autom Sci Eng 13(2):1072–1089
73. Li XY, Shao XY, Gao L (2008) Optimization of flexible process planning by genetic programming. Int J Adv Manuf Technol 38(1–2):143–153
74. Mascia F, Lopez-Ibanez M, Dubois-Lacoste J, Stutzle T (2013) From grammars to parameters: automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In: Nicosia G, Pardalos P (eds) Learning and intelligent optimization, Springer, Berlin, LNCS, vol 7997, pp 321–334
75. Masood A, Mei Y, Chen G, Zhang M (2016a) Many-Objective genetic programming for job-shop scheduling. In: CEC'16: IEEE congress on evolutionary computation, pp 209–216
76. Masood A, Mei Y, Chen G, Zhang M (2016b) A PSO-based reference point adaption method for genetic programming hyper-heuristic in many-objective job shop scheduling. In: Australasian conference on artificial life and computational intelligence (ACALCI), (to appear)
77. Mckay RI, Hoai NX, Whigham PA, Shan Y, O'Neill M (2010) Grammar-based genetic programming: a survey. Genetic Program Evolv Mach 11(3–4):365–396
78. Mei Y, Zhang M (2016) A comprehensive analysis on reusability of GP-evolved job shop dispatching rules. In: WCCI-CEC'16: IEEE congress on evolutionary computation
79. Mei Y, Zhang M, Nyugen 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, pp 365–372
80. Miller JF, Thomson P (2000) Cartesian genetic programming. European Conference on Genetic Programming. Springer, Berlin, pp 121–132
81. Miyashita K (2000) Job-shop scheduling with genetic programming. In: Whitley D, Goldberg D, Cantu-Paz E, Spector L, Parmee I, Beyer HG (eds) GECCO 2000: Proceedings of the genetic and evolutionary computation conference. Morgan Kaufmann, San Francisco, pp 505–512
82. Montana DJ (1995) Strongly typed genetic programming. Evol Comput 3(2):199–230
83. Mucientes M, Vidal JC, Bugarin A, Lama M (2008) Processing times estimation in a manufacturing industry through genetic programming. In: IEEE 2008 3rd international workshop on genetic and evolving fuzzy systems (GEFS)
84. Nguyen S (2016) A learning and optimizing system for order acceptance and scheduling. Int J Adv Manuf Technol. doi:10.1007/s00170-015-8321-6
85. Nguyen S, Zhang M, Johnston M (2011) A genetic programming based hyper-heuristic approach for combinatorial optimisation. In: GECCO'11: Proceedings of the 13th annual conference on genetic and evolutionary computation, ACM, pp 1299–1306
86. Nguyen S, Zhang M, Johnston M, Tan KC (2012a) A coevolution genetic programming method to evolve scheduling policies for dynamic multi-objective job shop scheduling problems. In: CEC'12: IEEE congress on evolutionary computation (CEC), pp 1–8
87. Nguyen S, Zhang M, Johnston M, Tan KC (2012b) Evolving reusable operation-based due-date assignment models for job shop scheduling with genetic programming. In: EuroGP'12: Genetic Programming, no. 7244 in Lecture notes in computer science, pp 121–133
88. Nguyen S, Zhang M, Johnston M, Tan K (2013a) Learning iterative dispatching rules for job shop scheduling with genetic programming. Int J Adv Manuf Technol 67(14):85–100
89. Nguyen S, Zhang M, Johnston M, Tan KC (2013b) A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem. IEEE Trans Evol Comput 17(5):621–639
90. Nguyen S, Zhang M, Johnston M, Tan KC (2013c) Dynamic Multi-objective job shop scheduling: a genetic programming approach. In: Uyar AS, Ozcan E (eds) Urquhart N (eds) Automated scheduling and planning, no. 505 in studies in computational intelligence, Springer, Berlin, pp 251–282
91. Nguyen S, Zhang M, Johnston M, Tan KC (2013d) 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, Hu B (eds) Genetic programming, Springer, Berlin, LNCS, vol 7831, pp 157–168
92. Nguyen S, Zhang M, Johnston M (2014a) Enhancing branch-and-bound algorithms for order acceptance and scheduling with genetic programming. In: EuroGP'14: Genetic programming, no. 8599 in Lecture notes in computer science, Springer, Berlin, pp 124–136
93. Nguyen S, Zhang M, Johnston M (2014b) A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling. In: Liu D, Hussain A, Zeng Z, Zhang N (eds) CEC'14: IEEE congress on evolutionary computation (CEC). IEEE Press, Piscataway, NJ, pp 1824–1831
94. Nguyen S, Zhang M, Johnston M, Tan K (2014c) Genetic programming for evolving due-date assignment models in job shop environments. Evol Comput 22(1):105–138
95. Nguyen S, Zhang M, Johnston M, Tan KC (2014d) Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming. IEEE Trans Evol Comput 18(2):193–208
96. Nguyen S, Zhang M, Johnston M, Tan KC (2014e) Selection Schemes in surrogate-assisted genetic programming for job shop scheduling. In: SEAL'14: simulated evolution and learning, Springer International Publishing, pp 656–667
97. Nguyen S, Zhang M, Johnston M, Tan K (2015a) Automatic programming via iterated local search for dynamic job shop scheduling. IEEE Trans Cybern 45(1):1–14
98. Nguyen S, Zhang M, Tan KC (2015b) Enhancing genetic programming based hyper-heuristics for dynamic multi-objective job shop scheduling problems. In: CEC'15: IEEE congress on evolutionary computation (CEC), pp 2781–2788
99. Nguyen S, Zhang M, Tan KC (2016) Surrogate-assisted genetic programming with simplified models for automated design of

dispatching rules. IEEE Trans Cybern. doi:10.1109/TCYB.2016.2562674

100. Nie L, Shao X, Gao L, Li W (2010) Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems. Int J Adv Manuf Technol 50(58):729–747
101. Nie L, Gao L, Li P, Wang X (2011) Multi-Objective optimization for dynamic single-machine scheduling. In: Tan Y, Shi Y, Chai Y, Wang G (eds) Advances in swarm intelligence: second international conference, ICSI 2011, Chongqing, China, June 12–15, 2011. Proceedings, Part II, Springer, Berlin, pp 1–9
102. Nie L, Gao L, Li P, Zhang L (2011) Application of gene expression programming on dynamic job shop scheduling problem. In: Proceedings of the 2011 15th international conference on computer supported cooperative work in design. IEEE Press, Piscataway, NJ, pp 291–295
103. Nie L, Bai Y, Wang X, Liu K (2012) 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, Springer, Berlin, LNCS, vol 7332, pp 383–390
104. Nie L, Gao L, Li P, Li X (2013a) A GEP-based policies constructing approach for dynamic flexible job shop scheduling problem with job release dates. J Intell Manuf 24(4):763–774
105. Nie L, Gao L, Li P, Shao X (2013b) Reactive scheduling in a job shop where jobs arrive over time. Comput Ind Eng 66:389–405
106. Nowicki E, Smutnicki C (1996) A fast taboo search algorithm for the job shop problem. Manag Sci 42(6):797–813
107. Olafsson S, Li X (2010) Learning effective new single machine dispatching rules from optimal scheduling data. Int J Prod Econ 128(1):118–126
108. Ong Y, Gupta A (2016) Evolutionary multitasking: a computer science view of cognitive multitasking. Cognit Comput 8(2):125–142
109. Ouelhadj D, Petrovic S (2008) A survey of dynamic scheduling in manufacturing systems. J Sched 12(4):417
110. Park J, Nguyen S, Johnston M, Zhang M, (2013a) Evolving Stochastic dispatching rules for order acceptance and scheduling via genetic programming. In: AI, (2013) Advances in artificial intelligence. Springer International Publishing, Lecture notes in computer science
111. Park J, Nguyen S, Zhang M, Johnston M (2013) Genetic programming for order acceptance and scheduling. In: Coello Coello CA, De la Fraga LG (eds) 2013 IEEE congress on evolutionary computation (CEC). IEEE Press, Piscataway, NJ, pp 1005–1012
112. Park J, Nguyen S, Zhang M, Johnston M (2014) Enhancing heuristics for order acceptance and scheduling using genetic programming. In: SEAL'14: Simulated evolution and learning, Springer International Publishing, pp 723–734
113. Park J, Nguyen S, Zhang M, Johnston M (2015a) Evolving Ensembles of dispatching rules using genetic programming for job shop scheduling. In: EuroGP'15: Genetic programming, Springer International Publishing, pp 92–104
114. Park J, Nguyen S, Zhang M, Johnston M (2015b) A single population genetic programming based ensemble learning approach to job shop scheduling. In: GECCO'15: Proceedings of the 2015 on genetic and evolutionary computation conference companion, pp 1451–1452
115. Park J, Mei Y, Chen G, Zhang M (2016a) Niching genetic programming based hyper-heuristic approach to dynamic job shop scheduling: an investigation into distance metrics. In: GECCO'16: Proceedings of the 2016 on genetic and evolutionary computation conference companion, pp 109–110
116. Park J, Mei Y, Nguyen S, Chen G, Johnston M, Zhang M (2016b) Genetic programming based hyper-heuristics for dynamic job shop scheduling: cooperative coevolutionary approaches. In: Genetic programming, no. 9594 in Lecture notes in computer science, Springer International Publishing, pp 115–132
117. Pickardt C, Branke J, Hildebrandt T, Heger J, Scholz-Reiter B (2010) Generating dispatching rules for semiconductor manufacturing to minimize weighted tardiness. In: Johansson B, Jain S, Montoya-Torres J, Hugan J, Ycesan E (eds) Proceedings of the 2010 winter simulation conference. IEEE Press, Piscataway, NJ, pp 2504–2515
118. Pickardt CW, Hildebrandt T, Branke J, Heger J, Scholz-Reiter B (2013) Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems. Int J Prod Econ 145(1):67–77
119. Pinedo ML (2008) Scheduling: theory, algorithms, and systems, 3rd edn. Springer, New York
120. Poli R (1998) Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. In: Artificial neural nets and genetic algorithms, pp 419–423
121. Potter MA, De Jong KA (2000) Cooperative coevolution: an architecture for evolving coadapted subcomponents. Evol Comput 8(1):1–29
122. Qin W, Zhang J, Sun Y (2013) Multiple-objective scheduling for interbay amhs by using genetic-programming-based composite dispatching rules generator. Comput Ind 64:694–707
123. Riley M, Mei Y, Zhang M (2016) Improving job shop dispatching rules through terminal weighting and adaptive mutation in genetic programming. In: IEEE congress on evolutionary computation, pp 3362–3369
124. Schmidt M, Lipson H (2009) Distilling free-form natural laws from experimental data. Science 324(5923):81–85
125. Sels V, Gheysen N, Vanhoucke M (2011) A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions. Int J Prod Res 50(15):4255–4270
126. Sha DY, Hsu CY (2006) A hybrid particle swarm optimization for job shop scheduling problem. Comput Ind Eng 51(4):791–808
127. Shahzad A, Mebarki N (2016) Learning dispatching rules for scheduling: a synergistic view comprising decision trees, tabu search and simulation. Computers 5(1):3. doi:10.3390/computers5010003. http://www.mdpi.com/2073-431X/5/1/3
128. Shi W, Song X, Sun J (2015) Automatic heuristic generation with scatter programming to solve the hybrid flow shop problem. Adv Mech Eng 7(2):1–9
129. Shiue YR (2009) Data-mining-based dynamic dispatching rule selection mechanism for shop floor control systems using a support vector machine approach. Int J Prod Res 47(13):3669–3690
130. Sim K, Hart E (2015) A novel heuristic generator for jssp using a tree-based representation of dispatching rules. In: GECCO'15: Proceedings of the companion publication of the 2015 on genetic and evolutionary computation conference
131. Sim K, Hart E, Paechter B (2015) A lifelong learning hyper-heuristic method for bin packing. Evol Comput 23(1):37–67
132. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2):278–285
133. Tan K, Lee T, Khor E (2002) Evolutionary algorithms for multi-objective optimization: performance assessments and comparisons. Artif Intell Rev 17(4):251–290
134. Tay JC, Ho NB (2007) Designing dispatching rules to minimize total tardiness. In: Dahal KP, Tan KC, Cowling PI (eds) Evolutionary scheduling, studies in computational intelligence, vol 49. Springer, Berlin, pp 101–124
135. Tay JC, Ho NB (2008) Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput Ind Eng 54(3):453–473
136. Vazquez-Rodriguez JA, Ochoa G (2011) On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming. J Oper Res Soc 62(2):381–396

137. Wang X, Nie L, Bai Y (2015) Discovering scheduling rules with a machine learning approach based on GEP and PSO for dynamic scheduling problems in shop floor. In: Computational intelligence in industrial application, pp 365–370
138. Weckman GR, Ganduri CV, Koonce DA (2008) A neural network job-shop scheduler. J Intell Manuf 19(2):191–201
139. Whigham PA (1995) Grammatically-based genetic programming. In: Rosca JP (ed) Proceedings of the workshop on genetic programming: from theory to real-world applications, pp 33–41
140. Wu SX, Banzhaf W (2011) Rethinking multilevel selection in genetic programming. In: Proceedings of the 13th annual conference on genetic and evolutionary computation, ACM, New York, NY, USA, GECCO '11, pp 1403–1410
141. Yang JW, Cheng HC, Chiang TC, Fu LC (2008) Multiobjective lot scheduling and dynamic OHT routing in a 300-mm wafer fab. In: 2008 IEEE international conference on systems, man and cybernetics, pp 1608–1613
142. Yin WJ, Liu M, Wu C (2003) Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming. In: Sarker R, Reynolds R, Abbass H, Tan KC, McKay B, Essam D, Gedeon T (eds) The 2003 congress on evolutionary computation (CEC 2003), IEEE Press, Piscataway, NJ, vol 2, pp 1050–1055