Automated Design of Production Scheduling Heuristics: A Review

Jürgen Branke, *Member, IEEE*, Su Nguyen, *Member, IEEE*, Christoph W. Pickardt, and Mengjie Zhang, *Senior Member, IEEE*

***Abstract*—Hyper-heuristics have recently emerged as a powerful approach to automate the design of heuristics for a number of different problems. Production scheduling is a particularly popular application area for which a number of different hyper-heuristics have been developed and are shown to be effective, efficient, easy to implement, and reusable in different shop conditions. In particular, they seem to be a promising way to tackle highly dynamic and stochastic scheduling problems, an aspect that is specifically emphasized in this survey. Despite their success and the substantial number of papers in this area, there is currently no systematic discussion of the design choices and critical issues involved in the process of developing such approaches. This paper strives to fill this gap by summarizing the state-of-the-art approaches, suggesting a taxonomy, and providing the interested researchers and practitioners with guidelines for the design of hyper-heuristics in production scheduling. This paper also identifies challenges and open questions and highlights various directions for future work.**

***Index Terms*—Evolutionary design, genetic programming (GP), hyper-heuristic, scheduling.**

I. Introduction

SCHEDULING is concerned with the allocation of limited resources to tasks over time, with the basic aim to ensure an effective and efficient use of the available resources. A classic problem area is the scheduling of manufacturing systems, in which machines (the resources) have to be allocated to jobs (the tasks) in the best possible way (minimizing or maximizing some objective function). Some of the costs that are typically affected by a production schedule are the holding costs of in-process inventory, contractual penalties for late deliveries, setup costs, and the costs of scrap and rework, which illustrate the importance of production scheduling to manufacturers in their endeavor to become and remain competitive.

A number of exact solution methods that solve deterministic scheduling problems optimally have been proposed in [1]. However, due to the high complexity of most scheduling problems of interest, exact methods are usually unable to solve large instances within a reasonable computational time. Moreover, many problems are stochastic and dynamic, i.e., they are subject to change over time due to random, stochastic events such as new job arrivals, stochastic processing times, or machine breakdowns. Consequently, many researchers and practitioners have turned to heuristics, which deliver acceptable, but not necessarily optimal, solutions in a short computational time.

In general, heuristics are problem-specific solution methods and have to be designed for the problem at hand. Unfortunately, the design of sophisticated heuristics is usually a tedious trial-and-error process, with candidate heuristics tested on some instances of the considered problem, modified, and retested until they meet the demands for actual implementation, which requires a significant amount of expertise, time, and coding effort. To handle this issue, various methods to (partially) automate the design of heuristics have been proposed in the literature, also known as hyper-heuristics.

In [2], hyper-heuristics are defined as "an automated methodology for selecting or generating heuristics to solve hard computational search problems." In other words, hyper-heuristics explore a search space of heuristics to discover those that work effectively. In this survey, we use fitness to denote the effectiveness of heuristics (discussed in Section III-E), whereas the objective value or function denotes the quality of a schedule.

Burke *et al.* [2] classified hyper-heuristics with respect to the nature of their process, i.e., whether they select or generate heuristics. Moreover, they distinguish hyper-heuristics that learn online, i.e., while solving a problem instance, from those that learn offline, i.e., that gather reusable knowledge from a set of training instances. Burke *et al.* [3] provided a general overview of the state-of-the-art of hyper-heuristic design, covering all categories of hyper-heuristics. The scope of this survey is on (offline) hyper-heuristics for the generation of a reusable heuristic, which can be applied to quickly solve new problem instances once it has been generated. We deliberately do not cover hyper-heuristics that select a heuristic for every decision point of a particular problem instance (see [4]–[6]), as the generated sequence of heuristics can generally not be reused, nor do we cover hyper-heuristics that learn to select heuristics for a given problem instance (see [7]–[9]), as this is problem classification rather than heuristic generation. Hyper-heuristics for the generation of heuristics have also been developed for other problem areas, including bin packing [10]–[12], vehicle routing [13]–[15], timetabling [16], [17], air traffic control [18], [19], and project scheduling [20].

Manuscript received July 21, 2014; revised December 8, 2014 and March 23, 2015; accepted April 22, 2015. This work was supported by the Marsden Fund of New Zealand Government under Contract VUW1209.

J. Branke and C. W. Pickardt are with Warwick Business School, University of Warwick, Coventry CV4 7AL, U.K.

S. Nguyen and M. Zhang are with the Evolutionary Computation Research Group, Victoria University of Wellington, Wellington 6140, New Zealand (e-mail: su.nguyen@ecs.vuw.ac.nz).

Digital Object Identifier 10.1109/TEVC.2015.2429314

TABLE I
SCHEDULING PROBLEMS ADDRESSED BY HEURISTIC GENERATION HYPER-HEURISTICS

Problem class		References
Shop configuration	Single machine	[21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33]
	Parallel machines	[34]
	Job shop	[26], [31], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56]
	Flexible job shop	[57], [58], [59], [60], [61], [62], [63], [64], [65]
	Flow shop	[25], [66], [67]
Special processing characteristic	Sequence-dependent setups	[28], [31], [32], [33], [34], [59], [61], [63], [65]
	Job precedence constraints	[31]
	Batch processing	[27], [65]
	Machine eligibility restrictions	[60], [61], [62], [64]
	Dynamic/stochastic environment	[23], [35], [36], [38], [40], [41], [44], [47], [51], [52], [53], [54], [55], [56], [57], [58], [65]
Objective	Completion time-based	[23], [24], [25], [27], [28], [29], [31], [34], [36], [37], [39], [42], [44], [45], [46], [48], [49], [50], [52], [53], [55], [57], [61], [64], [66]
	Due date-based	[22], [23], [25], [26], [27], [29], [30], [31], [33], [34], [35], [36], [38], [40], [41], [43], [47], [48], [49], [50], [54], [55], [58], [59], [60], [62], [63], [66], [64], [65], [67]
	Multi-objective	[32], [51], [56]

This paper presents a comprehensive review of the literature on hyper-heuristics for the design of construction heuristics in production scheduling with the aim to guide the interested researcher and practitioner through the key issues in developing such hyper-heuristics. The remainder of this paper is organized as follows. Section II describes the scheduling problems addressed and heuristics generated by means of hyper-heuristics. Section III reviews the design of hyper-heuristics, with a particular focus on the representation and evaluation of candidate heuristics. Some potential issues and challenges with specific suggestions for future work are discussed in Section IV, followed by a short summary of this paper and a more general outlook on future research.

II. SCHEDULING ENVIRONMENTS AND HEURISTICS

In general, heuristics are designed to be effective for a specific problem or class of problems. Production scheduling problems can be categorized by various properties, two important ones being the shop configuration and the objective. The simplest "shop" configuration is a single machine that is responsible for processing all jobs. If there is more than one machine available to process a job, this is called a parallel machine environment. Multistage problems, which are generally NP-hard [68], are characterized by jobs that consist of a number of processing steps, or operations, that need to be performed on distinct machines in a specified order. Depending on whether all jobs share the same processing order or not, the configuration is called a flow shop or job shop. Flow shops and job shops are further called flexible, if they contain at least one work center that consists of parallel machines [1, Ch. 2]. Objectives can be broadly classified as completion time based, with a focus on the efficiency of the manufacturing system, and due date based, with a focus on adherence to promised delivery dates. Table I provides a summary of the scheduling problems that have been addressed in the literature by means of a hyper-heuristic.

In most cases, the heuristics generated by the respective hyper-heuristic belong to the class of dispatching rules. Dispatching rules are a particularly simple type of scheduling heuristic, which progressively construct solutions by scheduling one operation at a time. Whenever a machine is available and there are jobs waiting to be processed on that machine, dispatching rules compute a priority index for each eligible job as a function of some job attributes (e.g., its processing time or due date), and shop attributes (e.g., the average processing time in the queue of the considered work center), and schedule only the imminent operation of the job with the highest priority. Due to their locally restricted horizon, dispatching rules have very low computational and information requirements, irrespective of the complexity of the overall problem. Moreover, because each scheduling decision is made at the latest possible moment, i.e., immediately before its implementation, dispatching rules naturally possess the ability to quickly react to unexpected changes, which makes them particularly suited for stochastic and dynamic scheduling problems (for a list of papers explicitly addressing stochastic dynamic environments, see Table I). These properties, together with their simple and intuitive nature, their ease of implementation and their flexibility to incorporate domain knowledge and expertise [69] explain the wide usage of dispatching rules in practice [70] and the ongoing research on the development of new, more effective dispatching rules (see [71]–[73]).

While dispatching rules that have been trained on a set of static, deterministic problem instances could, in principle, be applied to dynamic, stochastic problems, Hildebrandt *et al.* [44] and Nguyen *et al.* [49] showed that this does not necessarily lead to good results, and that it is better to use dynamic, stochastic problems also during training. In terms of hyper-heuristic design, there are some minor differences between using deterministic or stochastic problems for training, which will be discussed in the corresponding sections. In particular, other attributes may be needed (Section III-B), and the fitness function becomes stochastic (Section III-E), which in turn raises issues such as the determination of an appropriate run length of the simulation (Section III-E2). Moreover, the definition of stochastic benchmark problems is also more difficult (Section IV-D).

TABLE II
CLASSIFICATION OF HYPER-HEURISTICS FOR THE GENERATION OF PRODUCTION SCHEDULING HEURISTICS BY LEARNING METHOD AND REPRESENTATION

	Supervised learning	Unsupervised learning
Parametric representation	[21], [38], [42], [46]	[35], [36], [41], [45], [47], [52], [57], [58], [59], [67]
Grammar-based representation	[24], [30], [39]	[22], [23], [25], [26], [27], [28], [29], [31], [32], [33], [34], [37], [40], [43], [44], [48], [49], [50], [51], [52], [53], [54], [55], [56], [60], [61], [62], [63], [64], [65], [66]

In the literature, dispatching rules are typically designed for and tested on a (flexible) job shop problem, which is reflected in Table I by the relatively large number of studies dealing with this shop configuration. In general, hyper-heuristics have been used to evolve dispatching rules for a variety of scheduling problems with various objective functions and processing characteristics. Also, some recent work has focussed on the development of hyper-heuristics that can evolve a set of Pareto-optimal dispatching rules for multiobjective problems. A general conclusion of these studies is that hyper-heuristics are able to generate dispatching rules that outperform manually designed benchmark rules.

A few researchers have used hyper-heuristics for the generation of other types of scheduling heuristics. Yin *et al.* [23] evolved the so-called predictive heuristics, which aim to construct schedules that are robust to unpredictable breakdowns of machines and are shown to outperform a benchmark heuristic from the literature. Vázquez-Rodríguez and Ochoa [66] evolved variants of the iterative Nawaz, Enscore, and Ham (NEH) heuristic [74] for a number of permutation flow shop problems, which are significantly better than the original NEH heuristic and a randomized version. Mascia *et al.* [67], also generated iterative heuristics for a permutation flow shop problem. Park *et al.* [33] and Nguyen *et al.* [50] employed a hyper-heuristic for the generation of iterative dispatching rules and variants of a size limited beam search heuristic. These iterative scheduling heuristics evaluate (partial) candidate solutions, and are thus restricted to static and deterministic problems. As in the case of dispatching rules, a key component of the above scheduling heuristics is their priority function (or index), which is generally the part that is evolved by the hyper-heuristic. Hence, the subsequent discussion will focus on the evolution of dispatching rules, and priority functions in particular.

III. Hyper-Heuristic Design Choices

Fig. 1 shows a simplified outline of the procedure of a hyper-heuristic for the generation of heuristics. The main components in the design of such a hyper-heuristic concern the encoding or representation of candidate heuristics, which defines the search space, the optimization algorithm to explore this search space, and the fitness function to determine the quality of candidate heuristics. In this survey, we classify the existing hyper-heuristics according to the learning method they

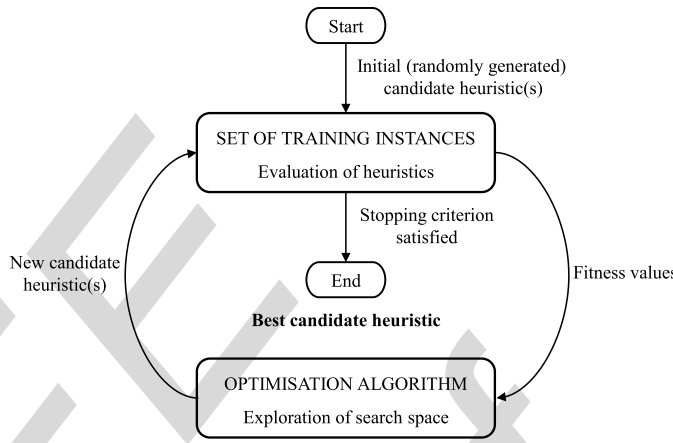

Fig. 1. Basic procedure of a hyper-heuristic for the generation of heuristics.

adopt (supervised or unsupervised) and their representation of candidate heuristics (parametric or grammar based), as shown in Table II.

Section III-A discusses the two types of learning methods used within hyper-heuristics, followed by a discussion of the selection of attributes to be provided to the hyper-heuristic in Section III-B. The different representations of priority functions are presented in Section III-C together with suitable optimization algorithms as they are closely tied to the chosen representation. Section III-D discusses the definition of the eligible job set, and Section III-E discusses appropriate fitness functions for the evaluation of candidate heuristics.

A. Learning Method

All hyper-heuristics generate new heuristics by learning from a set of training instances. This learning can be supervised or unsupervised. The basic idea of hyper-heuristics using supervised learning in scheduling is to supply the hyper-heuristic with a number of very good (preferably optimal) schedules that it uses to derive a priority function that reproduces these schedules as closely as possible. These priority functions can then be used as part of a heuristic, e.g., a dispatching rule, to solve other problem instances.

A variety of such supervised hyper-heuristics have been proposed in the literature. El-Bouri *et al.* [21] and Eguchi *et al.* [38] developed hyper-heuristics that operate on a neural network representation and use a back-propagation optimization algorithm to learn from optimal schedules. Ingimundardottir and Runarsson [46] followed the same approach but use logistic regression. Weckman *et al.* [42], also proposed a neural network hyper-heuristic, based on a variant of the back-propagation algorithm, but learn from solutions generated by an evolutionary algorithm (EA) instead of optimal solutions. Similarly, Koonce and Tsai [39] employed attribute-oriented induction to derive decision rules that reproduce the sequences generated by an EA. Li and Olafsson [24] and Olafsson and Li [30] developed a hyper-heuristic that generates priority functions in the form of decision trees. However, they only learn from solutions obtained by some simple dispatching rules, which often generate solutions far from optimal.

While some of the above studies report promising results, a major drawback of the supervised learning approach is that good global schedules can only be obtained for static problems of relatively small size and low complexity, which limits the applicability of these hyper-heuristics. In contrast, hyper-heuristics using unsupervised learning generate effective scheduling heuristics by simply applying candidate heuristics to a set of problem instances (the training instances), measuring their performance, and using this feedback to guide the search toward promising areas of the search space. Hence, unsupervised hyper-heuristics can be applied with relative ease to any scheduling problem that can be simulated, which makes them more practical in general. This is also reflected by the fact that most studies in the area of production scheduling develop hyper-heuristics that are based on unsupervised learning (see Table II), and the subsequent discussion hence concentrates on those.

B. *Attributes*

Irrespective of the representation used, an important design decision concerns the selection of adequate job and shop attributes that form the components of the priority functions that can be evolved. To distinguish jobs from each other and be able to prioritize one over another, it is obviously necessary to include some job attributes, whereas the inclusion of shop attributes allows for the generation of rules that can adapt to changing shop conditions. Moreover, in the special case of evolving iterative scheduling heuristics that make use of the characteristics of candidate solutions in solving a problem, as proposed by Park *et al.* [33] and Nguyen *et al.* [49], the hyper-heuristic has to be provided with some attributes related to the current candidate solution, e.g., the realized completion time of a job. In general, the challenge is to select all the relevant attributes while keeping the search space as small as possible. Table III lists a number of promising attributes that have been commonly used in the development of effective dispatching rules in the literature, where shop attributes are divided into attributes that concern the work center for which a dispatching decision is being made and global attributes. Attributes should be carefully chosen in consideration of the given scheduling problem, e.g., there is no benefit in providing the hyper-heuristic with due date attributes when the objective function is not due date based (see [44]), and certain attributes do not make sense in a dynamic scheduling environment with new jobs arriving all the time (e.g., sum of all processing times).

An important question regarding the selection of attributes is whether to include attributes in their most basic or in some aggregate form. To illustrate, a number of researchers provide their hyper-heuristics with the job due date d_j and the current time t as separate attributes [23], [25], [31], [32], [40], [62], [64]. However, it could be argued that due dates are more meaningful if they are expressed relative to the current time, i.e., $d_j - t$, and that integrating the absolute due date with other job attributes directly will often lead to rules that change their behavior over time, which is generally questionable and particularly unsuitable for long dynamic scheduling problems. In fact, the term $d_j - t$ appears in many effective manually developed dispatching rules [75]–[77] and several studies on hyper-heuristics have resorted to including due dates (and also release dates and arrival times) in the set of attributes in their relative form [35], [36], [44], [65]. It may make sense to aggregate attributes even further, e.g., to define the non-negative slack $\max(d_j - t - p_j^{\mathrm{r}}, 0)$ [29], [31], [41] or the non-negative time to arrival $\max(r_j^{\mathrm{i}} - t, 0)$ [29], [31] as one attribute to distinguish jobs on schedule from late jobs and jobs arriving in the future from waiting jobs, respectively, where r_j^{i} denotes the arrival time of job j to the work center required for its imminent operation. Kuczapski *et al.* [45] and Baek and Yoon [58] selected a number of composite priority indices of dispatching rules from the literature as attributes for their hyper-heuristics. However, these priority indices may integrate attributes in a suboptimal manner and restrict the hyper-heuristic in its search for a better priority function. Overall, it appears that it is best to provide a hyper-heuristic with attributes in their most basic form and let the hyper-heuristic search for good combinations unless there is a good theoretical foundation for an aggregate attribute, as in the cases above.

Another question related to the selection of attributes is whether or not to normalize them to a similar scale. In some cases, this may be necessary to fit a certain representation, e.g., the grammar-based representation by Nguyen *et al.* [49] or the neural network representation by Branke *et al.* [52]. Hershauer and Ebert [35], Eguchi *et al.* [41], and Baek and Yoon [58] also scaled the attributes to a similar range. In a recent study, Branke *et al.* [52] tested two hyper-heuristics for the generation of dispatching rules (one operating on a parametric, the other on a grammar-based representation of priority functions) with and without normalized attributes. They find that normalizing the attributes improves the performance of both hyper-heuristics, especially when the original attributes differ largely in scale. The following sections describe different representations of priority functions used within hyper-heuristics to combine the individual attributes.

C. *Representations of Priority Functions*

The choice of representation is very important as it determines the range and complexity of the priority functions (or indices) that can be generated by the hyper-heuristic.

TABLE III
PROMISING ATTRIBUTES FOR THE GENERATION OF PRIORITY FUNCTIONS

Type	Attribute	References
Job	(Imminent) operation processing time	[21], [22], [23], [24], [25], [26], [27], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [58], [59], [60], [62], [64], [65]
	Processing time of subsequent operation(s)	[25], [36], [37], [44], [49], [51], [52], [53], [54]
	Sum of processing times of remaining operations	[25], [26], [31], [35], [37], [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58], [60], [61], [62], [63], [64], [65], [66], [67]
	Weighted (linearly decreasing) sum of processing times of remaining operations	[66]
	Number of remaining operations	[26], [31], [35], [38], [39], [40], [41], [42], [43], [44], [45], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [59], [63], [64], [65]
	Release date	[23], [24], [25], [26], [27], [29], [30], [31], [32], [33], [34], [44], [48], [51], [52], [53], [56], [60], [61], [62], [64], [66]
	Arrival time at considered work centre	[26], [31], [40], [44], [45], [46], [47], [48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [61], [64]
	Due date of job	[21], [22], [23], [24], [25], [26], [27], [29], [30], [31], [32], [33], [34], [35], [36], [38], [40], [41], [43], [44], [45], [47], [48], [49], [50], [51], [54], [55], [56], [58], [59], [60], [61], [62], [63], [64], [65], [66], [67]
	Due date of imminent operation	[44], [65]
	Weight	[21], [24], [26], [30], [31], [32], [33], [34], [38], [40], [41], [45], [49], [50], [54], [55], [56], [65], [66], [67]
	Setup time (given the current setup)	[31], [32], [33], [34], [63], [65]
	Number of machines that can process (imminent) operation	[59], [61], [63]
	Number of successor operations in precedence graph	[31]
	Level of (imminent) operation in precedence graph	[31]
Work centre	Sum of (imminent) operation processing times of all waiting jobs	[25], [27], [38], [41], [49], [56]
	Average (imminent) operation processing time of all waiting jobs	[25], [27], [51], [58], [65]
	Minimum processing time of (imminent) operations of waiting jobs	[25], [27], [49], [56]
	Maximum processing time of (imminent) operations of waiting jobs	[25], [27], [38], [41], [49], [56]
	Maximum sum of processing times of remaining operations of waiting jobs	[38], [41]
	Number of waiting jobs	[25], [26], [27], [38], [41], [47], [54], [55], [63]
	Minimum due date of waiting jobs	[25], [27]
	Maximum due date of waiting jobs	[25], [27], [38], [41]
	Maximum weight of waiting jobs	[38], [41]
	Average setup time of waiting jobs (given the current setup)	[65]
	Maximum saving in setup time if (imminent) operation is processed on parallel machine	[65]
	Number of waiting jobs of same family	[27], [65]
	Fullness of batch (using currently waiting jobs)	[65]
	Speed of considered machine	[34]
	Sum of speeds of all parallel machines	[34]
Global	Sum of processing times of remaining operations of all jobs requiring considered work centre	[23], [26], [31], [34], [49], [56]
	Number of remaining operations of all jobs requiring considered work centre	[23], [26], [31], [34]
	WINQ = Sum of (imminent) operation processing times (adjusted for number of parallel machines) of jobs waiting at next work centre (required for subsequent operation of a job)	[36], [38], [41], [44], [48], [51], [52], [53], [57], [58]
	NINQ = Number of jobs waiting at next work centre	[36], [37], [38], [41], [54]
	Maximum WINQ of waiting jobs	[38], [41]
	Maximum NINQ of waiting jobs	[38], [41]
	Average waiting time of jobs last processed across all work centre in the shop	[54]
	Sum of processing times of remaining operations of all jobs requiring next work centre	[37]
	Sum of (imminent) operation processing times of jobs waiting at critical work centre (the one with the greatest sum of processing times of all remaining operations)	[49], [56]
	Sum of processing times of remaining operations of all jobs requiring critical work centre	[49], [56]
	Sum of (imminent) operation processing times of jobs waiting at considered work centre that still have to visit critical work centre	[49], [56]
	Sum of (imminent) operation processing times of jobs waiting at work centre that still have to visit the work centre with currently largest queue (measured in processing time)	[49], [56]

For ease of presentation, priority indices in this paper are defined so that a higher index I_j corresponds to a higher priority of a job j. To illustrate, the priority index of the well-known minimum slack (MS) rule is given by

$$I_j^{\mathrm{MS}} = -\left(d_j - t - p_j^{\mathrm{r}}\right) \quad (1)$$

where d_j denotes the due date of job j, p_j^{r} the processing time of the remaining operations of job j, and t refers to the current time. Sections III-C1 and III-C2 discuss parametric and grammar-based representations of priority functions, respectively. Examples of these two representations are compared empirically in [52].

1) Fixed-Length Parametric Representations: One approach to encoding priority functions is to predefine their basic format and parameterize it. Then, a priority function can be represented by a vector of (real valued) parameter values. A simple and commonly used format is that of the weighted sum [35], [36], [45], [57], that is

$$I_j = \sum_{y=1}^{a} w_y x_{y,j} \tag{2}$$

where $x_{y,j}$ denotes one of the a attributes of job j provided to the hyper-heuristic, and w_y refers to the corresponding weight. To illustrate, if the weighted sum were used as the predefined format of priority functions and $x_{1,j} = d_j$, $x_{2,j} = t$, and $x_{3,j} = p_j^{\mathrm{r}}$, the priority function of the MS rule would be encoded by the parameter vector $\mathbf{w} = (-1, +1, +1)$. Other simple formats based on if-then-else rules have also been proposed in [47] and [59].

Clearly, a representation based on a simple format such as the weighted sum is often too restrictive to allow for the discovery of the most effective priority functions (see [52]), which motivates the use of more complex representations, e.g., based on artificial neural networks [41], [52]. On the other hand, such representations lead to a significantly larger search space, and also to priority functions that are so complex that they defy interpretation. The challenge is to choose a format that is as simple as possible without compromising the ability of the hyper-heuristic to generate effective priority functions, which is difficult as the complexity required for a given problem is normally unknown in advance. One study that examines the impact of the flexibility of the representation on the results is [52].

One advantage of a parametric representation is that search spaces of real-valued vectors are relatively common, implying the availability of a number of suitable optimization algorithms. In fact, Hooke–Jeeves pattern search [35], [36], simulated annealing [41], and EAs [45], [47], [52], [58], [59] have all been successfully used for the generation of scheduling heuristics based on parametric representations.

2) Variable-Length Grammar-Based Representations: An alternative way of defining the search space of priority functions is by means of a grammar that specifies how the individual components can be assembled to yield a valid priority function. Fig. 2 gives an example of a grammar for the generation of priority functions, where the expressions on the left-hand side can be replaced by any of the expressions on the right-hand side (alternative options are separated by the "|" symbol). A specific priority function can then be represented by an expression tree, which is a popular representation in the literature (see [27], [29], [31], [37]). Expression trees are composed of leaf nodes, representing terminals such as the attributes in this case, and internal nodes, representing the functions to combine the terminals with each other. Fig. 3 shows an expression tree that could be generated with the grammar from Fig. 2, which encodes the MS priority function, where expression trees are decoded recursively, starting from the root node, and from left to right.

<start> ::= <node>
<node> ::= <function> <node> <node> | <terminal>
<function> ::= $+$ | $-$ | $\times$ | $/$
<terminal> ::= d_j | p_j^{r} | t

Fig. 2. Simple grammar for the construction of priority functions.

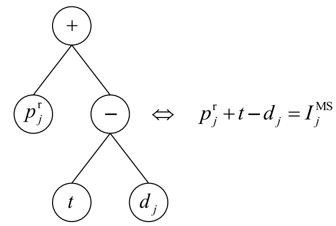

Fig. 3. Expression tree representing the priority function of the MS rule.

The main reason for the popularity of grammar-based representations is that they allow for the generation of priority functions of variable format and length without the requirement to define a basic format in advance. Apart from the attribute (terminal) set, the only input that has to be provided to the hyper-heuristic is a set of suitable functions that it can use to combine the attributes.

Table IV lists the functions that have been used within hyper-heuristics. It shows that the four basic arithmetic operators are included in the function set in every of the reviewed papers, with the division either implemented as protected (returns 1 if divisor is 0) or unprotected (returns a very large number if divisor is 0). These operators allow for the reconstruction of many priority functions of the most effective manually designed rules, which justifies their selection. The function set is further often supplemented with a ternary or quaternary if-then-else (ifte) operator, defined as

$$\mathrm{ifte}(x_1, x_2, x_3) = \begin{cases} x_2 & \text{if } x_1 \geq 0 \\ x_3 & \text{otherwise} \end{cases} \tag{3}$$

and

$$\mathrm{ifte}(x_1, x_2, x_3, x_4) = \begin{cases} x_3 & \text{if } x_1 \geq x_2 \\ x_4 & \text{otherwise} \end{cases} \tag{4}$$

and/or some common mathematical functions such as max or min. As in the case of the attribute set, a larger function set increases the size of the search space and the aim should thus be to select only the most relevant functions. Against this background, the value of including more complex functions such as cos, sin, exp, log, pow, or sqrt that generally do not occur in priority functions of effective rules from the literature, and for which there is no theoretical justification, is questionable. Moreover, some of the above functions can be expressed

TABLE IV
FUNCTIONS USED WITHIN HYPER-HEURISTICS OPERATING ON A GRAMMAR-BASED REPRESENTATION OF PRIORITY FUNCTIONS

Type	Functions	References
Arithmetic	$+, -, \times, /$	[22], [23], [25], [26], [27], [29], [31], [32], [33], [34], [37], [40], [44], [43], [48], [49], [50], [51], [52], [53], [54], [55], [56], [60], [61], [62], [63], [64], [65], [66]
Logical	ifte (ternary)	[32], [33], [44], [43], [49], [50], [51], [52], [53], [54], [55], [56], [61], [63], [65]
	ifte (quaternary)	[25], [26], [27], [40]
	$>$	[23], [43], [61], [63]
	$\geq$	[61]
	$\equiv$	[43], [61]
	$\wedge, \vee, \neg$	[43], [63]
Mathematical	max	[23], [25], [27], [33], [44], [50], [51], [52], [53], [54], [56], [65]
	min	[23], [25], [27], [33], [50], [51], [54], [56], [65]
	avg	[63]
	sgn	[61]
	pos	[26], [31], [34]
	abs	[50], [51], [56]
	neg	[25], [27], [49]
	sin, cos	[43], [61]
	exp	[25], [27], [43], [61], [63]
	log	[43], [61], [63]
	pow	[25], [27], [43], [61]
	sqrt	[26], [43], [61]

by means of other functions, e.g., $\max(x_1, x_2) = \text{ifte}(x_1 - x_2, x_1, x_2)$, $\min(x_1, x_2) = \text{ifte}(x_2 - x_1, x_1, x_2)$, $\text{neg}(x_1) = 0 - x_1$, $\text{abs}(x_1) = \max(x_1, \text{neg}(x_1))$, raising the question of whether those functions should be directly provided or whether it should be left to the hyper-heuristic to reconstruct them in case they are beneficial.

Although one advantage of the grammar-based representation is that the complexity of the resulting priority indices is potentially unbounded, in practice many researchers have bounded the complexity and search space by limiting the maximum tree depth. Unfortunately, there are no theoretical guidelines on the determination of an adequate maximum tree depth for the evolution of priority functions. If it is chosen too small, some high quality heuristics might not be representable and thus the quality of the solutions the algorithm can find is limited. On the other hand, if it is chosen too large, the hyper-heuristic may get lost in the vast search space. The depth used in previous studies varies between 6 and 17 [22], [34], [40], [48], [49], [62], [65]. Jakobović and Marasović [31] tested their hyper-heuristic with values ranging from 9 to 17 and find that a maximum tree depth of 14 leads to the best results. However, the best value is likely to depend on the given problem as well as the employed optimization algorithm in general.

In addition to job and shop attributes, many researchers have included some (random) constants in the terminal set of their hyper-heuristics [23], [25], [33], [37], [44], [56], [65], [66]. This enables the hyper-heuristic to weigh attributes differently, especially since the latter have different units and can be of different magnitude.

Search spaces of expression trees are typically explored by means of genetic programming (GP), which is also the predominant optimization algorithm employed in the literature for the evolution of scheduling heuristics (see [25], [49]). An exception by Nie *et al.* [29], [48], [64], who used gene expression programming (GEP) instead. In [29], they compare their GEP hyper-heuristic to a GP hyper-heuristic and report that the former generates slightly better priority functions in most cases, and in much less time. Moreover, the priority functions evolved by GEP are shown to be relatively simple and easy to understand, whereas GP has the tendency to evolve unnecessarily large expression trees (see [22], [44], [51], [61]). This phenomenon, also referred to as bloating, is generally undesired as it increases the runtime of GP and leads to priority functions that are more complex but not necessarily more effective.

D. Set of Eligible Jobs

In general, a dispatching rule does not only specify a priority function but also the eligible job set, i.e., the jobs from which the next job to be scheduled can be selected. Most dispatching rules only consider jobs eligible for scheduling that are already waiting at the given work center. This implies that a machine is never left idle if there are jobs waiting to be scheduled, which is also referred to as nondelay scheduling. While nondelay scheduling is generally effective, as it minimizes delays due to idle times, it is not necessarily optimal. In fact, it can be beneficial to leave a machine idle in some situations, e.g., in anticipation of a high priority job arriving in the near future that should be processed without delay. In order for dispatching rules to be able to take such a decision, the eligible job set has to be extended to include also jobs arriving in the future.

In most of the hyper-heuristics designed to evolve dispatching rules the eligible job set is restricted to waiting jobs [25], [35], [36], [40], [41], [51], [62], [65]. Another common setting is to also include future jobs that are expected to arrive before the shortest operation of waiting jobs can be completed [29], [31], [37], [45], which is in the spirit of the Giffler and Thompson algorithm [78]. Hildebrandt *et al.* [44] tested their hyper-heuristic with and without inclusion of future jobs in the set of eligible jobs and find that the best rule evolved with the extended job set outperforms the best evolved nondelay rule. On the other hand, the performance of the hyper-heuristic is shown to vary a lot from run-to-run with the extended job set, which indicates that it is more difficult to generate effective dispatching rules that take future jobs into account and allow for decisions to leave a machine idle.

Instead of providing a hyper-heuristic with a fixed definition of the eligible job set, the latter can also be optimized by the hyper-heuristic itself. Nguyen *et al.* [49] let their hyper-heuristic optimize a parameter called the nondelay factor that controls the extent to which future jobs are included in the eligible job set. In another paper, Nguyen *et al.* [50] designed a hyper-heuristic that evolves a separate function (of some shop attributes) for the nondelay factor, which can then adapt to changing shop conditions.

E. Evaluation of Candidate Heuristics

In order to know whether a candidate heuristic is effective or not, unsupervised hyper-heuristics need to obtain an estimate of the performance of that heuristic by applying it to some training instances. The quality of the solutions generated by the heuristic for these instances then determines its fitness, which in turn governs the search behavior of the hyper-heuristic. Hence, in evaluating candidate heuristics, two important decisions to be made concern the selection of training instances and the definition of the fitness function, which are discussed in Sections III-E1 and III-E2, respectively.

1) Training Instances: For reasons of simplicity, a training instance is defined in this paper as an instance that provides a measure of performance for a given heuristic. This includes static problem instances as well as runs of a stochastic simulation. Clearly, whether to use static instances or stochastic simulation for the evaluation of candidate heuristics depends on which problems the heuristic is supposed to solve once it has been generated. This is illustrated by Hildebrandt *et al.* [44], who tested the dispatching rules from Tay and Ho [62], which have been trained on and shown to be effective for static instances, in a long-term simulation with dynamic job arrivals and find that they perform poorly. Nguyen *et al.* [49], also examined the effectiveness of dispatching rules that have been evolved using static instances in a long-term simulation. They report that the evolved rules perform well if the shop utilization is equal or less than 80% but become worse than some benchmark rules as utilization increases beyond that value. They attribute this to the fact that static instances reflect conditions of low utilization, in which few new jobs arrive over time. Furthermore, the relative performance of evolved scheduling heuristics has been shown to deteriorate with an increasing deviation between the test and training instances in terms of job processing orders [37], number of jobs [22], and due date setting [32], [33], [65]. These results emphasize the importance of using a set of training instances that reflect the problems the heuristics are likely to encounter in their future use.

Another important factor with regard to the training set is its size, i.e., the number of training instances. If a small training set is chosen, the evolved heuristics are likely to overfit the training instances and not perform well on the unseen test instances, which implies that their reusability is very limited. On the other hand, a larger training set increases the runtime of the hyper-heuristic, without necessarily leading to better heuristics. Geiger and Uzsoy [27] reported that the performance of their hyper-heuristic improves as the number of training instances approaches 10 but does not improve further with a larger training set. In contrast, Jakobović and Marasović [31] found that their hyper-heuristic performs best with the largest of the tested settings (100 training instances), which indicates that the best training set size is highly problem-specific and has to be determined through pilot experiments.

Some researchers have argued that a hyper-heuristic can either be used to generate a heuristic that performs reasonably well for a number of related problems or one that is very effective for the specific problem it has been tailored to, and ineffective otherwise [66], [79]. This argument cannot be supported from a theoretical perspective, as there is no reason why two specialized heuristics could not be combined by a hyper-heuristic into one heuristic that analyzes the characteristics of a given problem and applies the (specialized) heuristic that is most suitable for it. On the other hand, the generation of more sophisticated heuristics certainly poses a challenge to hyper-heuristics up to an extent where the underlying relations are merely too complex to be discovered by the hyper-heuristic. In any case, the generation of heuristics that is supposed to perform well on a wider range of problems certainly requires a larger and more heterogeneous training set that covers this problem range, which in turn increases the runtime of the hyper-heuristic.

Note that, if the set of problem instances is very large or randomly generated (as is usually the case if a stochastic simulation is used for evaluating a dynamic problem) then computational limitations make it necessary to restrict evaluation to a subset (sample) of all possible training instances. Effectively, due to the sampling, the fitness function then becomes stochastic. In such cases, to reduce the sampling variance, it has been recommended to evaluate all solutions competing for survival within a generation by the same subset of problem instances, while changing the subset from iteration to iteration to make sure individuals that survive several generations are tested on a large variety of problem instances [23], [44], [65]. Furthermore, if the problem used for evaluation consists of a sequence of random jobs dynamically generated over time, at least in principle there are two ways to improve the accuracy of evaluation: by increasing the number of problem instances tested, or the number of jobs considered in each problem instance. In such cases, to get a proper estimate of the steady-state behavior of a solution, it is also necessary to discard the first jobs as warm-up period.

2) Fitness Function: The application of a scheduling heuristic $\mathcal{H}$ to a number of training instances $T = \{1, 2, \ldots, |T|\}$ 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 $f(.)$ to determine the overall fitness of the heuristic.

The following fitness functions have been proposed in the literature.

1) Sum [or average] of objective values [22], [23], [31], [32], [41], [48], [52]

$$f(\mathcal{H}) = \left[\frac{1}{|T|}\right]\sum_{i=1}^{|T|} z_i(\mathcal{H}).$$

2) Average relative objective value [44]

$$f(\mathcal{H}) = \frac{1}{|T|}\sum_{i=1}^{|T|} \frac{z_i(\mathcal{H})}{z_i(\text{ref})}.$$

3) Sum [or average] of relative deviations [33], [49], [66]

$$f(\mathcal{H}) = \left[\frac{1}{|T|}\right]\sum_{i=1}^{|T|} \frac{z_i(\mathcal{H}) - z_i(\text{ref})}{z_i(\text{ref})}$$

where $z_i(\text{ref})$ denotes a reference objective value for instance i, obtained by some other solution method. Which fitness aggregation is most desirable depends very much on the intentions of the designer of the algorithm.

The sum (or average) of objective values concentrates on performing well on problem instances with a large potential for improvement. To illustrate this, consider a problem where the objective is to minimize the mean tardiness of all jobs so that the fitness is computed as the sum of mean tardiness values obtained from all training instances. If there is one training instance in the set with a very tight due date setting, the tardiness of this instance will be much higher than that of other training instances and therefore strongly correlate with the fitness value. In consequence, the hyper-heuristic will focus its search on heuristics that can solve well this particular instance while largely ignoring their performance on other instances.

Alternatively, one can use the average relative objective value or the sum (or average) of relative deviations as the fitness function, which are equivalent for hyper-heuristics that operate on the ranks of fitness values rather than the values itself. These fitness functions reduce the weight of difficult training instances by relating the objective value of each instance to a reference value before combining them. Their disadvantage is that they require good reference values, which are generally only available for well-studied benchmark instances from the literature for which (near)-optimal solutions are known. In all other cases, reference values have to be obtained, typically by applying some benchmark heuristic(s) to the problem [44], [49], [66], which may or may not yield good results.

Another issue that arises only if candidate heuristics are evaluated with long simulation runs is that some heuristics, which may be present particularly at the start of the run of the hyper-heuristic, can lead to an unstable system, i.e., the number of jobs in the shop grows steadily. The fitness of these inferior heuristics may then be never obtained and excessive time wasted in the attempt. To prevent this from happening, Hildebrandt *et al.* [44] proposed to monitor the number of jobs in the shop during a simulation run and abort the run if a preset threshold value for the number of jobs is exceeded. The fitness of these heuristics is then largely reduced by a penalty, which ensures that they are quickly discarded. In a follow-up paper, Branke *et al.* [52] showed that this measure reliably detects the inferior heuristics without prematurely stopping the evaluation of good heuristics.

IV. Issues and Challenges

The research on hyper-heuristics for the automated design of scheduling heuristics is still in an early stage and there remain a number of open questions and challenges. The following sections discuss some of the main issues that future work should focus on.

A. Evolving Sets of Heuristics

A common theme of the existing studies on hyper-heuristics is that they predominantly address simple scheduling environments and/or employ a hyper-heuristic to evolve a single scheduling heuristic. However, in more complex scheduling environments, there may be a number of interrelated decision problems that have to be resolved, e.g., the formation and scheduling of batches in the presence of batch processing machines, the assignment of operations to machines and scheduling of these machines in parallel machine environments, or the coordination of resources in environments with multiple resource constraints. This raises the question of how to best deal with such more complex scenarios.

The most straightforward approach seems to be to design a set of heuristics, one for each decision, and to simply encode the set of heuristics as one individual. This approach is followed by Nie *et al.* [64], who developed a hyper-heuristic for flexible job shop problems that operates on a search space in which each individual consists of two functions, one for routing, i.e., assigning operations to machines, the other for sequencing, i.e., scheduling the machines. Their results show that this hyper-heuristic can evolve sets of heuristics that outperform the single sequencing heuristics evolved by a conventional hyper-heuristic (and combined with some benchmark routing heuristic). On the other hand, the drawback of encoding multiple heuristics as one individual is that the search space of the hyper-heuristic grows exponentially in the number of heuristics, which limits the approach to the generation of small sets of heuristics.

One possible solution to overcome the issue of search space size may be the use of coevolution, which implies a division of the search space into several sub-spaces handled separately by different subpopulations. Nguyen *et al.* [56] designed a coevolutionary hyper-heuristic, in which a subpopulation of priority functions used for dispatching is coevolved with a separate subpopulation of functions for due date assignment. The hyper-heuristic is shown to be very competitive to some other hyper-heuristics, in which the two functions are represented by a single individual. Another option is to generate heuristics sequentially, in a similar way as proposed in [57]. However, such hyper-heuristics implicitly assume that the relative effectiveness of candidate heuristics at one stage is more or less independent of the heuristics to be evolved in subsequent stages, and thus cannot be expected to perform very well if there are strong interdependencies between the subproblems. On the other hand, it may be sufficient to resolve several subproblems using the same heuristic in some situations. To illustrate, Park *et al.* [33] addressed an order acceptance and scheduling problem with a hyper-heuristic that evolves two separate functions for the acceptance and dispatching of jobs. They compare its performance to a hyper-heuristic evolving a single function to handle both decisions and find that the latter is more effective, indicating that there is no need for a separate heuristic to deal with the acceptance of jobs, possibly due to the strong correlation underlying the two decisions. Hence, it is important to carefully assess the interrelations between the subproblems to be solved prior to designing a hyper-heuristic.

In multimachine problems, it has been found beneficial to use different dispatching rules at different machines if the latter vary with respect to their positions in the shop [80]–[83] and/or workload [84]–[86]. Consequently, some researchers have developed hyper-heuristics that generate sets of machine-specific rules by selecting a (potentially) different rule for each work center from a number of given rules [65], [87], [88] or by evolving several composite rules, where each rule is tailored to a certain work center [25], [55], [57], [58]. The search space then grows exponentially in the number of work centers or machines, which can be very large in shops of realistic size, requiring some specific measures in addition to the above techniques to deal with this problem. Miyashita [40] proposed a hyper-heuristic that is based on a predetermined classification of machines into bottlenecks and nonbottlenecks and evolves one rule for each class of machines. Similarly, Jakobović and Budin [26] designed a hyper-heuristic that optimizes the classification of machines while searching for good dispatching rules for each class. More specifically, each individual consists of three functions, where one of them is a discriminating function of attributes relating to the workload of a machine that determines which of the two dispatching rules, encoded by the other two (priority) functions, to apply. The best rule sets evolved by these hyper-heuristics are generally shown to outperform single benchmark rules. However, the results of the study by Nguyen *et al.* [50], who examined the performance of the three-function hyper-heuristic by Jakobović and Budin [26] more closely, show that the effectiveness among the evolved rule sets varies a lot more than that among the single rules evolved by a conventional hyper-heuristic. This indicates that the former struggles with the more complex search space. In summary, there appears to be some potential for future work on intelligent designs of hyper-heuristics for more complex scheduling environments.

B. Attribute Selection and Construction

As discussed in Section III-B, a main challenge in designing an effective hyper-heuristic is to provide it with all the relevant problem attributes while excluding any redundant or irrelevant attributes. Otherwise, the search space could be either too restrictive or unnecessarily large, which both hinder the hyper-heuristic in its ability to generate effective heuristics.

A few studies have performed some analysis of the best evolved heuristics in order to identify important attributes. Branke *et al.* [52] leaved out each of the attributes present in the priority functions of their evolved dispatching rules one-by-one and examine the performance of these rules without the respective attribute. Their analysis shows that some attributes, specifically, those that also appear in the most effective rules from the literature, are substantially more important for the performance of the evolved rules than others. Eguchi *et al.* [41] examined the first-order correlation between attribute values and priority values (applying the best evolved dispatching rule) to determine the relevance of attributes, and eliminate ineffective attributes in this way. Nguyen *et al.* [49] conducted a high-level analysis of the occurrence frequency of attributes in the priority functions of the best dispatching rules evolved. They find that the relevance of attributes is problem-specific to some extent though some seem to be generally more important than others. This highlights a major drawback of any post-generation analysis, which is that the gained insights may only be applicable to the given problem (class), for which an effective heuristic has already been generated, and therefore, be of limited value. Instead, future hyper-heuristics should be designed to perform the tasks of selecting and constructing suitable attributes automatically and simultaneously to the optimization.

C. Interpretability and Trust

The interpretability of evolved heuristics is a crucial aspect to gain the trust of users, i.e., operators or managers, particularly since hyper-heuristics are black box optimizers. Unfortunately, there is some evidence that more complex scheduling environments (for which the use of hyper-heuristics is most promising) often require heuristics of a certain complexity so that simply choosing an easy-to-interpret representation will result in heuristics of comparatively low quality [52]. On the other hand, it may be possible to allow for open-ended evolution and still search for the simplest representation of a well-performing heuristic, or to generate good tradeoffs between quality and interpretability, by means of multiobjective methods, with heuristic complexity being one objective to be minimized. Online rule simplification techniques [89]–[91] can also be applied to improve the readability of evolved heuristics.

In many cases, it may be possible to simplify heuristics after they have been generated without significantly compromising their performance. This is particularly true for heuristics evolved on the basis of a grammar-based representation, which typically contain redundant components, e.g., if-then-else operators where the condition is always true or false. The following simplication, these heuristics may then be analyzed manually and linked to some human-designed heuristics to facilitate interpretation [25], [27], [49], [56], [66]. However, fully understanding evolved heuristics is still a challenging task, especially when dealing with complex environments, which stresses the need for some methodological support.

In the literature, a few tools and methods to understand the behavior of scheduling heuristics, specifically dispatching rules, have been developed and used. Branke *et al.* [52] analyzed dispatching rules by visualizing their priority indices as functions of the attributes they incorporate. Clearly, such a visualization is only possible for a very limited number of attributes. Branke and Pickardt [92] proposed a method that identifies weaknesses in the decision logic of a given dispatching rule. All in all, future research on hyper-heuristics should place more emphasis on the issue of interpretability of heuristics and controlling or reducing their complexity.

D. Comparison of Algorithms

In order to give recommendations on when it is beneficial to use a hyper-heuristic and how to design it, extensive and meaningful performance comparisons of evolved heuristics with more sophisticated (global) solution algorithms as well as between different hyper-heuristics are needed. So far, such comparisons have been rather limited (see [28], [32], [33], [45], [49], [56] and [29], [43], [45], [49], [52], respectively). Intuitively, hyper-heuristic approaches have strengths compared to global optimization approaches in particular in dynamic and stochastic environments where a quick reaction is important. But as observed in [93], they also become more competitive as the problem size (and thus the search space for the global optimizer) increases.

One reason for the limited number of comparisons may be that hyper-heuristics possess several properties that make a fair comparison particularly difficult. For example, not only are the hyper-heuristics stochastic algorithms with many parameters to tune, but also is the evaluation function often a stochastic simulation, resulting in stochastic fitness values. Also, the running time for the simulations can be quite substantial, and, to make things worse, the running time to evaluate a particular dispatching rule strongly depends on the rule itself, as the time to calculate the priority value and the number of jobs in the system depends on the rule itself. This implies that a comparison of hyper-heuristics based on the same number of function evaluations has limited validity.

Irrespective of the challenges faced, an important prerequisite for systematic algorithm comparisons is the availability of suitable benchmark problems and algorithms. For reusable heuristics, it is further important to clearly distinguish between training and test problem instances—the hyper-heuristic may use the training instances during optimization, while the generated heuristics have to be tested on a separate, previously unseen set of test instances. While libraries exist for static, deterministic scheduling problem instances, e.g., the OR-Library [94] (which has also been used to test hyper-heuristics [33], [50]), the most promising applications for hyper-heuristics seem to be in the area of dynamic, stochastic problems, which are much more difficult to define. A possible benchmark are the dynamic job and flow shop problems designed by Rajendran and Holthaus [95] and Holthaus and Rajendran [96] for the purpose of comparing dispatching rules from the literature. We have used these problems in several hyper-heuristic studies [44], [51]–[53], and have published some results online [97]. Still, especially for more complex dynamic scheduling problems, the publication of entire simulators (see jasima [97]) would greatly help replicability and facilitate fair comparisons. Also, the generated heuristics should be published in addition to the obtained results, ideally in a format that can be directly plugged into a simulator.

E. Computational Time

A major drawback of hyper-heuristics based on unsupervised learning is their high computational requirements. Even though the obtained heuristics typically can be executed very fast, a run of the hyper-heuristic itself can last many hours, especially if the evaluation of the many candidate heuristics, evolved during the search, involves extensive simulation runs. Measures to reduce the computational time of unsupervised hyper-heuristics should consequently focus on the fitness evaluations, which usually take up the most time by far.

Some approaches have been proposed to reduce the time spent on evaluations. References [44], [52], and [65] equipped their hyper-heuristic with a mechanism that monitors the number of jobs in the shop to detect heuristics that cause an unstable system (see Section III-E2) and terminates the evaluation of heuristics once a preset threshold value is exceeded. This idea could be extended to save time on the evaluation of other inferior candidate heuristics by monitoring similar values, e.g., the objective value after a predefined number of completed jobs. Branke *et al.* [52] suggested a duplicate detection technique to avoid evaluating the same candidate heuristic twice. Thereby, two priority functions are considered equivalent if they provide the same ranking on a set of randomly generated "dummy" operations. Hildebrandt and Branke [53] investigated the use of surrogate models to approximate the fitness of candidate heuristics, i.e., dispatching rules, in a GP hyper-heuristic. By employing a distance measure based on the behaviors of rules, the proposed surrogate-supported GP hyper-heuristic can reduce the computational cost and improve the convergence speed, which indicates that the development of surrogate models is a promising direction for future work.

As previously discussed, another aspect that influences the computational time is the complexity of candidate heuristics. It is well-known that optimization algorithms that operate on a variable-length grammar-based representation, such as GP, are liable to bloating [98], i.e., they gradually evolve larger and more complex individuals that are not necessarily better, but require more time to be evaluated. Thus, controlling or reducing the complexity of the heuristics that are evolved is also important for efficiency reasons, and the development of effective bloating control [99], [100] and online program simplication techniques [89]–[91] should also be of concern to hyper-heuristic research in the future.

F. Overfitting and Robustness

Given the high computational requirements of unsupervised hyper-heuristics, it is highly desirable that the resulting heuristics are reusable. However, like other forms of machine learning, hyper-heuristics carry the risk of generating heuristics that overfit the problem instances used in the training stage and perform poorly on all other instances, limiting their reusability. In fact, overfitting has been observed in connection with hyper-heuristics by several researchers, in particular when more complex representations [40], [49] and/or small sets of training instances [33] are used. Hence, this issue should be taken into account when designing a hyper-heuristic.

A concept closely related to overfitting is that of the robustness of heuristics, i.e., their ability to cope with unforeseen changes in the scheduling environment. Clearly, if the performance of the heuristics evolved was to strongly deteriorate in the event of a minor change, e.g., in the job arrival pattern, this would question the practicality of the approach. So far, a few studies have examined the robustness of dispatching rules evolved by various hyper-heuristics. Their results can be summarized in that evolved rules show to be reasonably robust, including to changes in the number of machines [44], processing time distribution [44], [51], job arrival pattern [44], [65], shop utilization [51], [52], [65], and due date setting [51], [65]. These studies can be further extended, e.g., by examining the limits of the robustness of evolved heuristics, i.e., when they become worse than benchmark heuristics. On the other hand, if the changes to the scheduling environment are more pronounced, there is always the option to simply rerun the hyper-heuristic to generate a new heuristic for the altered problem. How to determine the point at which rerunning the hyper-heuristic becomes beneficial is another challenging question to be investigated.

V. Conclusion

In recent years, hyper-heuristics have demonstrated their ability to automatically generate very competitive heuristics for a wide range of problems. Because hyper-heuristics generate heuristics automatically, it becomes feasible to tailor heuristics to the specific production environment, and to change them quickly whenever the environment changes. In this sense, hyper-heuristics have the potential to revolutionize production scheduling as they allow problem-specific heuristics to be applied successfully in settings where the traditional way of a human expert designing heuristics would be too expensive, or simply too time consuming.

This paper constitutes the first comprehensive review of hyper-heuristics for the automated design of production scheduling heuristics, providing a simple taxonomy and focussing on key design choices such as the learning method, attributes, representation, and fitness evaluation. Moreover, a number of the issues and challenges that should be addressed in the future have been discussed, including the generation of rule sets, algorithm comparison, interpretability of the resulting heuristics, computational time, and overfitting and robustness.

The review is aimed for researchers as well as practitioners. Researchers who aim to further advance the technique of hyper-heuristic scheduling are provided with a comprehensive review of the state-of-the-art and a discussion of the open issues suitable for future work. Also, we have established a website that may serve as a starting point for future algorithm comparisons on dynamic, stochastic benchmark problems. Practitioners in scheduling, on the other hand, can use this paper to compose a suitable hyper-heuristic and make the appropriate design choices for their particular application. This paper contains guidelines, for example, on how to select attributes, what fitness function to choose, and what representation might be the most appropriate.

Currently, the vast majority of papers fall into the category of unsupervised learning with open-ended grammar-based evolution. Clearly, some of the less explored areas may deserve more attention, and the work reviewed here may benefit from cross-fertilization also with other hyper-heuristic concepts, such as hyper-heuristics for heuristic selection [3]. Whereas the approaches to select dispatching rules mostly use machine learning algorithms such as artificial neural networks, decision trees or support vector machines, heuristic generation approaches mostly apply heuristic search methods such as EAs or tabu search, and so far there is very little overlap between the two areas. A first example that combines heuristic generation and heuristic selection (but both based on EAs) can be found in [65]. Another promising direction may be to automatically gear metaheuristics to a particular problem domain, such as in [101].

Finally, our review has focussed on the generation of reusable heuristics for production scheduling. The research in this area may benefit from ideas developed for hyper-heuristics in related problem domains like timetabling [16], [17] or project scheduling [20]. Given their potential and the various open problems, research on hyper-heuristics for the design of production scheduling heuristics is likely to continue yet for some time.

References

[1] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 3rd ed. New York, NY, USA: Springer, 2008.

[2] E. K. Burke *et al.*, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics* (International Series in Operations Research & Management Science), vol. 146, 2nd ed., M. Gendreau and J.-Y. Potvin, Eds. New York, NY, USA: Springer, 2010, pp. 449–468.

[3] E. K. Burke *et al.*, "Hyper-heuristics: A survey of the state of the art," *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, 2013.

[4] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Industrial Scheduling*, J. F. Muth and G. L. Thompson, Eds. Englewood Cliffs, NJ, USA: Prentice-Hall, 1963, pp. 225–251.

[5] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Manage. Sci.*, vol. 38, no. 10, pp. 1495–1509, 1992.

[6] U. Dorndorf and E. Pesch, "Evolution based learning in a job shop scheduling environment," *Comput. Oper. Res.*, vol. 22, no. 1, pp. 25–40, 1995.

[7] M. J. Shaw, "A pattern-directed approach to flexible manufacturing: A framework for intelligent scheduling, learning, and control," *Int. J. Flex. Manuf. Syst.*, vol. 2, no. 2, pp. 121–144, 1989.

[8] S. Nakasuka and T. Yoshida, "Dynamic scheduling system utilizing machine learning as a knowledge acquisition tool," *Int. J. Prod. Res.*, vol. 30, no. 2, pp. 411–431, 1992.

[9] W. Mouelhi-Chibani and H. Pierreval, "Training a neural network to select dispatching rules in real time," *Comput. Ind. Eng.*, vol. 58, no. 2, pp. 249–256, 2010.

[10] R. Kumar, A. H. Joshi, K. K. Banka, and P. I. Rockett, "Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming," in *Proc. 10th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Atlanta, GA, USA, 2008, pp. 1227–1234.

[11] E. Özcan and A. J. Parkes, "Policy matrix evolution for generation of heuristics," in *Proc. 13th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Dublin, Ireland, 2011, pp. 2011–2018.

[12] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automating the packing heuristic design process with genetic programming," *Evol. Comput.*, vol. 20, no. 1, pp. 63–89, 2012.

[13] M. Oltean and D. Dumitrescu, "Evolving TSP heuristics using multi expression programming," in *Computational Science—ICCS* (LNCS 3037), M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Germany: Springer, 2004, pp. 670–673.

[14] A. Beham, M. Kofler, S. Wagner, and M. Affenzeller, "Agent-based simulation of dispatching rules in dynamic pickup and delivery problems," in *Proc. 2nd Int. Symp. Logist. Ind. Informat.*, Linz, Austria, 2009, pp. 1–6.

[15] S. Vonolfen, A. Beham, M. Kommenda, and M. Affenzeller, "Structural synthesis of dispatching rules for dynamic dial-a-ride problems," in *Computer Aided Systems Theory—EUROCAST* (LNCS 8111), R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, Eds. Berlin, Germany: Springer, 2013, pp. 276–283.

[16] M. Bader-El-Den, R. Poli, and S. Fatima, "Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework," *Memet. Comput.*, vol. 1, no. 3, pp. 205–219, 2009.

[17] N. Pillay, "Evolving hyper-heuristics for the uncapacitated examination timetabling problem," *J. Oper. Res. Soc.*, vol. 63, no. 1, pp. 47–58, 2012.

[18] V. H. L. Cheng, L. S. Crawford, and P. K. Menon, "Air traffic control using genetic search techniques," in *Proc. IEEE Int. Conf. Control Appl.*, vol. 1. Kohala Coast, HI, USA, 1999, pp. 249–254.

[19] J. V. Hansen, "Genetic search methods in air traffic control," *Comput. Oper. Res.*, vol. 31, no. 3, pp. 445–459, 2004.

[20] T. Frankola, M. Golub, and D. Jakobovic, "Evolutionary algorithms for the resource constrained scheduling problem," in *Proc. 30th Int. Conf. Inf. Technol. Interf. (ITI)*, Dubrovnik, Croatia, 2008, pp. 715–722.

[21] A. El-Bouri, S. Balakrishnan, and N. Popplewell, "Sequencing jobs on a single machine: A neural network approach," *Eur. J. Oper. Res.*, vol. 126, no. 3, pp. 474–490, 2000.

[22] C. Dimopoulos and A. M. S. Zalzala, "Investigating the use of genetic programming for a classic one-machine scheduling problem," *Adv. Eng. Softw.*, vol. 32, no. 6, pp. 489–498, 2001.

[23] W.-J. Yin, M. Liu, and C. Wu, "Learning single-machine scheduling heuristics subject to machine breakdowns with genetic programming," in *Proc. Congr. Evol. Comput. (CEC)*, vol. 2. Canberra, ACT, Australia, 2003, pp. 1050–1055.

[24] X. Li and S. Olafsson, "Discovering dispatching rules using data mining," *J. Schedul.*, vol. 8, no. 6, pp. 515–527, 2005.

[25] C. D. Geiger, R. Uzsoy, and H. Aytuğ, "Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach," *J. Schedul.*, vol. 9, no. 1, pp. 7–34, 2006.

[26] D. Jakobović and L. Budin, "Dynamic scheduling with genetic programming," in *Genetic Programming* (LNCS 3905), P. Collet, M. Tomassini, M. Ebner, S. Gustafson, and A. Ekárt, Eds. Berlin, Germany: Springer, 2006, pp. 73–84.

[27] C. D. Geiger and R. Uzsoy, "Learning effective dispatching rules for batch processor scheduling," *Int. J. Prod. Res.*, vol. 46, no. 6, pp. 1431–1454, 2008.

[28] M. Kofler, S. Wagner, A. Beham, G. Kronberger, and M. Affenzeller, "Priority rule generation with a genetic algorithm to minimize sequence dependent setup costs," in *Computer Aided Systems Theory—EUROCAST* (LNCS 5717), R. Moreno-Díaz, F. Pichler, and A. Quesada-Arencibia, Eds. Berlin, Germany: Springer, 2009, pp. 817–824.

[29] L. Nie, X. Shao, L. Gao, and W. Li, "Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems," *Int. J. Adv. Manuf. Technol.*, vol. 50, nos. 5–8, pp. 729–747, 2010.

[30] S. Olafsson and X. Li, "Learning effective new single machine dispatching rules from optimal scheduling data," *Int. J. Prod. Econ.*, vol. 128, no. 1, pp. 118–126, 2010.

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

[32] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Learning reusable initial solutions for multi-objective order acceptance and scheduling problems with genetic programming," in *Genetic Programming* (LNCS 7831), K. Krawiec, A. Moraglio, T. Hu, A. Ş. Etaner-Uyar, and B. Hu, Eds. Berlin, Germany: Springer, 2013, pp. 157–168.

[33] J. Park, S. Nguyen, M. Zhang, and M. Johnston, "Genetic programming for order acceptance and scheduling," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Cancún, Mexico, 2013, pp. 1005–1012.

[34] D. Jakobović, L. Jelenković, and L. Budin, "Genetic programming heuristics for multiple machine scheduling," in *Genetic Programming* (LNCS 4445), M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, and A. I. Esparcia-Alcázar, Eds. Berlin, Germany: Springer, 2007, pp. 321–330.

[35] J. C. Hershauer and R. J. Ebert, "Search and simulation selection of a job-shop sequencing rule," *Manage. Sci.*, vol. 21, no. 7, pp. 833–843, 1975.

[36] P. J. O'Grady and C. Harrison, "Search-based job shop scheduling and sequencing: Extensions to the search sequencing rule," *Math. Comput. Model.*, vol. 13, no. 3, pp. 13–27, 1990.

[37] L. Atlan, J. Bonnet, and M. Naillon, "Learning distributed reactive strategies by genetic programming for the general job shop problem," in *Proc. 7th Florida Artif. Intell. Res. Symp.*, Pensacola, FL, USA, 1994, pp. 1–11.

[38] T. Eguchi, F. Oba, and T. Hirai, "A neural network approach to dynamic job shop scheduling," in *Global Production Management* (IFIP—The International Federation for Information Processing), vol. 24, K. Mertins, O. Krause, and B. Schallock, Eds. New York, NY, USA: Springer, 1999, pp. 152–159.

[39] D. A. Koonce and S.-C. Tsai, "Using data mining to find patterns in genetic algorithm solutions to a job shop schedule," *Comput. Ind. Eng.*, vol. 38, no. 3, pp. 361–374, 2000.

[40] K. Miyashita, "Job-shop scheduling with genetic programming," in *Proc. Genet. Evol. Comput. Conf. (GECCO)*, Las Vegas, NV, USA, 2000, pp. 505–512.

[41] T. Eguchi, F. Oba, and S. Toyooka, "A robust scheduling rule using a neural network in dynamically changing job-shop environments," *Int. J. Manuf. Technol. Manage.*, vol. 14, nos. 3–4, pp. 266–288, 2008.

[42] G. R. Weckman, C. V. Ganduri, and D. A. Koonce, "A neural network job-shop scheduler," *J. Intell. Manuf.*, vol. 19, no. 2, pp. 191–201, 2008.

[43] M. Kofler, A. Beham, S. Wagner, and M. Affenzeller, "Evaluation of dispatching strategies for the optimization of a real-world production plant," in *Proc. 2nd Int. Symp. Logist. Ind. Informat.*, Linz, Austria, 2009, pp. 25–30.

[44] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios—A genetic programming approach," in *Proc. 12th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Portland, OR, USA, 2010, pp. 257–264.

[45] A. M. Kuczapski, M. V. Micea, L. A. Maniu, and V. I. Cretu, "Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling," *Inf. Technol. Control*, vol. 39, no. 1, pp. 32–37, 2010.

[46] H. Ingimundardottir and T. P. Runarsson, "Supervised learning linear priority dispatch rules for job-shop scheduling," in *Learning and Intelligent Optimization* (LNCS 6683), C. A. C. Coello, Ed. Berlin, Germany: Springer, 2011, pp. 263–277.

[47] M. Kapanoglu and M. Alikalfa, "Learning IF–THEN priority rules for dynamic job shops using genetic algorithms," *Robot. Comput.-Integr. Manuf.*, vol. 27, no. 1, pp. 47–55, 2011.

[48] L. Nie, L. Gao, P. Li, and L. Zhang, "Application of gene expression programming on dynamic job shop scheduling problem," in *Proc. 15th Int. Conf. Comput. Support. Cooper. Work Design*, Lausanne, Switzerland, 2011, pp. 291–295.

[49] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Trans. Evol. Comput.*, vol. 17, no. 5, pp. 621–639, Oct. 2013.

[50] S. Nguyen, M. Zhang, M. Johnston, and K. Tan, "Learning iterative dispatching rules for job shop scheduling with genetic programming," *Int. J. Adv. Manuf. Technol.*, vol. 67, nos. 1–4, pp. 85–100, 2013.

[51] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Dynamic multi-objective job shop scheduling: A genetic programming approach," in *Automated Scheduling and Planning* (Studies in Computational Intelligence), vol. 505, A. Ş. Etaner-Uyar, E. Özcan, and N. Urquhart, Eds. Berlin, Germany: Springer, 2013, pp. 251–282.

[52] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, "Hyper-heuristic evolution of dispatching rules: A comparison of rule representations," *Evol. Comput.*, to be published. AQ1

[53] T. Hildebrandt and J. Branke, "On using surrogates with genetic programming," *Evol. Comput.*, to be published.

[54] R. Hunt, M. Johnston, and M. Zhang, "Evolving 'less-myopic' scheduling rules for dynamic job shop scheduling with genetic programming," in *Proc. Conf. Genet. Evol. Comput. (GECCO)*, Vancouver, BC, Canada, 2014, pp. 927–934.

[55] R. Hunt, M. Johnston, and M. Zhang, "Evolving machine-specific dispatching rules for a two-machine job shop using genetic programming," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Beijing, China, 2014, pp. 618–625.

[56] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic design of scheduling policies for dynamic multi-objective job shop scheduling via cooperative coevolution genetic programming," *IEEE Trans. Evol. Comput.*, vol. 18, no. 2, pp. 193–208, Apr. 2014.

[57] D. H. Baek, W. C. Yoon, and S. C. Park, "A spatial rule adaptation procedure for reliable production control in a wafer fabrication system," *Int. J. Prod. Res.*, vol. 36, no. 6, pp. 1475–1491, 1998.

[58] D. H. Baek and W. C. Yoon, "Co-evolutionary genetic algorithm for multi-machine scheduling: Coping with high performance variability," *Int. J. Prod. Res.*, vol. 40, no. 1, pp. 239–254, 2002.

[59] H. Tamaki, K. Sakakibara, H. Murao, and S. Kitamura, "Rule acquisition for production scheduling: A genetics-based machine learning approach to flexible shop scheduling," in *Proc. Annu. Conf. SICE*, vol. 3. Fukui, Japan, 2003, pp. 2762–2767.

[60] J. C. Tay and N. B. Ho, "Designing dispatching rules to minimize total tardiness," in *Evolutionary Scheduling* (Studies in Computational Intelligence), vol. 49, K. P. Dahal, K. C. Tan, and P. I. Cowling, Eds. Berlin, Germany: Springer, 2007, pp. 101–124.

[61] A. Beham, S. Winkler, S. Wagner, and M. Affenzeller, "A genetic programming approach to solve scheduling problems with parallel simulation," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, Miami, FL, USA, 2008, pp. 1–5.

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

[63] E. Pitzer, A. Beham, M. Affenzeller, H. Heiss, and M. Vorderwinkler, "Production fine planning using a solution archive of priority rules," in *Proc. 3rd IEEE Int. Symp. Logist. Ind. Informat.*, Budapest, Hungary, 2011, pp. 111–116.

[64] L. Nie, L. Gao, P. Li, and X. Li, "A GEP-based reactive scheduling policies constructing approach for dynamic flexible job shop scheduling problem with job release dates," *J. Intell. Manuf.*, vol. 24, no. 4, pp. 763–774, 2013.

[65] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, and B. Scholz-Reiter, "Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems," *Int. J. Prod. Econ.*, vol. 145, no. 1, pp. 67–77, 2013.

[66] J. A. Vázquez-Rodríguez and G. Ochoa, "On the automatic discovery of variants of the NEH procedure for flow shop scheduling using genetic programming," *J. Oper. Res. Soc.*, vol. 62, no. 2, pp. 381–396, 2011.

[67] F. Mascia, M. López-Ibáñez, J. Dubois-Lacoste, and T. Stützle, "From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness," in *Learning and Intelligent Optimization* (LNCS 7997), G. Nicosia and P. Pardalos, Eds. Berlin, Germany: Springer, 2013, pp. 321–334.

[68] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, 1976.

[69] H. Aytug, M. A. Lawley, K. McKay, S. Mohan, and R. Uzsoy, "Executing production schedules in the face of uncertainties: A review and some future directions," *Eur. J. Oper. Res.*, vol. 161, no. 1, pp. 86–110, 2005.

[70] M. E. Pfund, S. J. Mason, and J. W. Fowler, "Semiconductor manufacturing scheduling and dispatching: State of the art and survey of needs," in *Handbook of Production Scheduling* (International Series in Operations Research & Management Science), vol. 89, J. W. Herrmann, Ed. New York, NY, USA: Springer, 2006, pp. 213–241.

[71] V. Sels, N. Gheysen, and M. Vanhoucke, "A comparison of priority rules for the job shop scheduling problem under different flow time- and tardiness-related objective functions," *Int. J. Prod. Res.*, vol. 50, no. 15, pp. 4255–4270, 2011.

[72] A. El-Bouri, "A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop," *Comput. Oper. Res.*, vol. 39, no. 7, pp. 1305–1314, 2012.

[73] A. Otto and C. Otto, "How to design effective priority rules: Example of simple assembly line balancing," *Comput. Ind. Eng.*, vol. 69, pp. 43–52, Mar. 2014.

[74] M. Nawaz, E. E. Enscore, Jr., and I. Ham, "A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem," *Omega Int. J. Manage. Sci.*, vol. 11, no. 1, pp. 91–95, 1983.

[75] E. J. Anderson and J. C. Nyirenda, "Two new rules to minimize tardiness in a job shop," *Int. J. Prod. Res.*, vol. 28, no. 12, pp. 2277–2292, 1990.

[76] E. Kutanoglu and I. Sabuncuoglu, "An analysis of heuristics in a dynamic job shop with weighted tardiness objectives," *Int. J. Prod. Res.*, vol. 37, no. 1, pp. 165–187, 1999.

[77] J. J. Kanet and X. Li, "A weighted modified due date rule for sequencing to minimize weighted tardiness," *J. Schedul.*, vol. 7, no. 4, pp. 261–276, 2004.

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

[79] E. K. Burke, M. Hyde, G. Kendall, and J. Woodward, "Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one," in *Proc. 9th Annu. Conf. Genet. Evol. Comput. (GECCO)*, London, U.K., 2007, pp. 1559–1565.

[80] R. T. Barrett and S. Barman, "A SLAM II simulation study of a simplified flow shop," *Simulation*, vol. 47, no. 5, pp. 181–189, 1986.

[81] F. Mahmoodi, C. T. Mosier, and R. E. Guerin, "The effect of combining simple priority heuristics in flow-dominant shops," *Int. J. Prod. Res.*, vol. 34, no. 3, pp. 819–839, 1996.

[82] H. Sarper and M. C. Henry, "Combinatorial evaluation of six dispatching rules in a dynamic two-machine flow shop," *Omega Int. J. Manage. Sci.*, vol. 24, no. 1, pp. 73–81, 1996.

[83] S. Barman, "Simple priority rule combinations: An approach to improve both flow time and tardiness," *Int. J. Prod. Res.*, vol. 35, no. 10, pp. 2857–2870, 1997.

[84] N. Raman, F. B. Talbot, and R. V. Rachamadugu, "Due date based scheduling in a general flexible manufacturing system," *J. Oper. Manage.*, vol. 8, no. 2, pp. 115–132, 1989.

[85] R. A. Ruben and F. Mahmoodi, "Lot splitting in unbalanced production systems," *Decis. Sci.*, vol. 29, no. 4, pp. 921–949, 1998.

[86] J. A. C. Bokhorst, G. Nomden, and J. Slomp, "Performance evaluation of family-based dispatching in small manufacturing cells," *Int. J. Prod. Res.*, vol. 46, no. 22, pp. 6305–6321, 2008.

[87] N. Ishii and J. J. Talavage, "A mixed dispatching rule approach in FMS scheduling," *Int. J. Flex. Manuf. Syst.*, vol. 6, no. 1, pp. 69–87, 1994.

[88] T. Yang, Y. Kuo, and C. Cho, "A genetic algorithms simulation approach for the multi-attribute combinatorial dispatching decision problem," *Eur. J. Oper. Res.*, vol. 176, no. 3, pp. 1859–1873, 2007.

[89] P. Wong and M. Zhang, "Algebraic simplification of GP programs during evolution," in *Proc. 8th Annu. Conf. Genet. Evol. Comput. (GECCO)*, Seattle, WA, USA, 2006, pp. 927–934.

[90] D. Kinzett, M. Johnston, and M. Zhang, "Numerical simplification for bloat control and analysis of building blocks in genetic programming," *Evol. Intell.*, vol. 2, no. 4, pp. 151–168, 2009.

[91] M. Johnston, T. Liddle, and M. Zhang, "A relaxed approach to simplification in genetic programming," in *Genetic Programming* (LNCS 6021), A. I. Esparcia-Alcázar, A. Ekárt, S. Silva, S. Dignum, and A. Ş. Uyar, Eds. Berlin, Germany: Springer, 2010, pp. 110–121.

[92] J. Branke and C. W. Pickardt, "Evolutionary search for difficult problem instances to support the design of job shop dispatching rules," *Eur. J. Oper. Res.*, vol. 212, no. 1, pp. 22–32, 2011.

[93] S. Nguyen, M. Zhang, and M. Johnston, "A sequential genetic programming method to learn forward construction heuristics for order acceptance and scheduling," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Beijing, China, 2014, pp. 1824–1831.

[94] J. E. Beasley, "OR-Library: Distributing test problems by electronic mail," *J. Oper. Res. Soc.*, vol. 41, no. 11, pp. 1069–1072, 1990.

[95] C. Rajendran and O. Holthaus, "A comparative study of dispatching rules in dynamic flowshops and jobshops," *Eur. J. Oper. Res.*, vol. 116, no. 1, pp. 156–170, 1999.

[96] O. Holthaus and C. Rajendran, "Efficient jobshop dispatching rules: Further developments," *Prod. Plan. Control*, vol. 11, no. 2, pp. 171–178, 2000.

[97] T. Hildebrandt. *Jasima—An Efficient Java Simulator for Manufacturing and Logistics*. [Online]. Available: http://code.google.com/p/jasima/ AQ2

[98] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA: MIT Press, 1992.

[99] S. Luke and L. Panait, "A comparison of bloat control methods for genetic programming," *Evol. Comput.*, vol. 14, no. 3, pp. 309–344, 2006.

[100] P. A. Whigham and G. Dick, "Implicitly controlling bloat in genetic programming," *IEEE Trans. Evol. Comput.*, vol. 14, no. 2, pp. 173–190, Apr. 2010.

[101] M. Samorani and M. Laguna, "Data-mining-driven neighborhood search," *INFORMS J. Comput.*, vol. 24, no. 2, pp. 210–227, 2012.

Jürgen Branken (M'99) received the Ph.D. degree from University of Karlsruhe, Karlsruhe, Germany, in 2000.

Since 1994, he has been an Active Researcher of the area of nature-inspired optimization with University of Karlsruhe. He is a Professor of Operational Research and Systems with Warwick Business School, University of Warwick, Coventry, U.K. He has published over 150 articles in international journals and conferences on various topics such as multiobjective optimization, handling of uncertainty in optimization, dynamically changing optimization problems, and the design of complex systems.

Prof. Branke is an Associate Editor of *Evolutionary Computation Journal* and an Area Editor of *Journal of Heuristics*.

Su Nguyen (M'13) received the B.E. degree in industrial and systems engineering from Ho Chi Minh City University of Technology, Ho Chi Minh City, Vietnam, in 2006; the M.E. degree in industrial engineering and management from Asian Institute of Technology (AIT), Bangkok, Thailand, in 2008; and the Ph.D. degree in evolutionary computation from Victoria University of Wellington (VUW), Wellington, New Zealand, in 2013.

He was a Research Associate with the Department of Industrial and Manufacturing Engineering, School of Engineering and Technology, AIT. He is currently a Research Fellow with the Evolutionary Computation Research Group, VUW. His research interests include evolutionary computation, discrete-event simulation, and their applications in operations planning and scheduling.

Christoph W. Pickardt received the degree from AQ3 Warwick Business School, University of Warwick, AQ4 Coventry, U.K.; the M.Sc. degree in industrial engineering and management from University of Karlsruhe, Karlsruhe, Germany; the Master of Engineering degree from University of Auckland, Auckland, New Zealand; and the Ph.D. degree from Warwick Business School, University of Warwick, in 2014.

He is an IT/SCM Consultant with Bayer Business Services, Leverkusen, Germany. His research interests include metaheuristics, simulation optimization, and manufacturing logistics.

Mengjie Zhang (M'04–SM'10) received the B.E. and M.E. degrees from the Artificial Intelligence Research Center, Agricultural University of Hebei, Baoding, China, in 1989 and 1992, respectively, and the Ph.D. degree in computer science from RMIT AQ5 University, Melbourne, VIC, Australia, in 2000.

Since 2000, he has been with Victoria University of Wellington, Wellington, New Zealand, where he is currently a Professor of Computer Science, the Head of Evolutionary Computation Research Group, and an Associate Dean (Research and Innovation) in the Faculty of Engineering. His research interests include evolutionary computation, particularly genetic programming, particle swarm optimization, and learning classifier systems with application areas of image analysis, multiobjective optimization, classification with unbalanced data, feature selection and reduction, and job shop scheduling. He has published over 350 academic papers in refereed international journals and conferences. He has supervised over 50 postgraduate research students.

Prof. Zhang is an Associated Editor or an Editorial Board Member for five international journals, including IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION and *Evolutionary Computation Journal*, and is a Reviewer for over 20 international journals. He is a Steering Committee Member and a Program Committee Member for over 80 international conferences. He is the Chair of the IEEE Computational Intelligence Society (CIS) Evolutionary Computation Technical Committee, a member of the IEEE CIS Intelligent Systems and Applications Technical Committee and the IEEE CIS Task Force of Hyper-Heuristics, the Vice-Chair of the IEEE CIS Task Force on Evolutionary Computer Vision and Image Processing and the IEEE CIS Task Force on Evolutionary Computation for Feature Selection and Construction, and the Founding Chair for the IEEE Computational Intelligence Chapter in New Zealand.