Chapter 14
A Classification of Hyper-Heuristic Approaches: Revisited

Edmund K. Burke, Matthew R. Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward

Abstract Hyper-heuristics comprise a set of approaches that aim to automate the development of computational search methodologies. This chapter overviews previous categorisations of hyper-heuristics and provides a unified classification and definition. We distinguish between two main hyper-heuristic categories: heuristic *selection* and heuristic *generation*. Some representative examples of each category are discussed in detail and recent research trends are highlighted.

E. K. Burke (✉)
University of Leicester, Leicester, UK
e-mail: edmund.burke@le.ac.uk

M. R. Hyde · E. Özcan
Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, Nottingham, UK
e-mail: matthewrobertyhyde@gmail.com; Ender.Ozcan@nottingham.ac.uk

G. Kendall
University of Nottingham Malaysia Campus, Semenyih, Malaysia

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, Nottingham, UK
e-mail: Graham.Kendall@nottingham.ac.uk

G. Ochoa
Computing Science and Mathematics, University of Stirling, Stirling, UK
e-mail: goc@cs.stir.ac.uk

J. R. Woodward
Queen Mary University of London, London, UK
e-mail: j.woodward@qmul.ac.uk

© Springer International Publishing AG, part of Springer Nature 2019
M. Gendreau, J.-Y. Potvin (eds.), *Handbook of Metaheuristics*,
International Series in Operations Research & Management Science 272,
https://doi.org/10.1007/978-3-319-91086-4_14

14.1 Introduction

The current state-of-the art in hyper-heuristic research represents a set of approaches that share the common goal of automating the design and adaptation of heuristic methods in order to address computational search problems. The motivation behind these approaches is to raise the level of generality at which search methodologies can operate [13]. The term hyper-heuristic was first used in 1997 [32] to describe a protocol that combines several artificial intelligence methods for automated theorem proving. The term was independently used in 2000 [28] to describe 'heuristics to choose heuristics' in the context of combinatorial optimisation. In this context, a hyper-heuristic is a high-level approach that, given a particular problem instance and a number of low-level heuristics, can select and apply an appropriate low-level heuristic at each decision point [13, 79]. The idea of automating the heuristic design process, however, is not new. Indeed, it can be traced back to the early 1960s [30, 38, 39], and was independently developed by a number of authors during the 1990s [36, 45, 49, 64, 95, 98]. Some historical notes, and a brief overview of early approaches can be found in [13] and [79], respectively. A more recent research trend in hyper-heuristics attempts to automatically *generate* new heuristics that are suited to a given problem or class of problems. This is typically done by combining, through the use of genetic programming for example, *components* or *building-blocks* of human designed heuristics [20].

We differentiate between the terms heuristic, metaheuristic and hyper-heuristic. A heuristic is a "rule of thumb" offering guidance for finding good solutions according to domain knowledge. Two main types of search heuristics can be distinguished, *perturbative* or local search heuristics, which operate on fully instantiated candidate solutions, and *constructive* heuristics which iteratively expand partial candidate solutions. Metaheuristics are search methodologies that coordinate local search and higher-level strategies to perform a robust search on a solution space and which aim to escape local optima. Hyper-heuristics are high-level strategies that operate on a search space of heuristics rather than directly on a search space of solutions.

A variety of hyper-heuristic approaches have been proposed in the literature. An introduction to the area appeared in the 2003 edition of the Handbook of Metaheuristics [13]. The present chapter updates our previous version [21], which suggested a unified classification and definition of hyper-heuristics. The proposed classification inspired a thorough survey article that appeared in 2013 [25], and has been widely adopted by the research community.

The next section outlines previous classifications of hyper-heuristics. Section 14.3 proposes our unified classification and definition. Sections 14.4 and 14.5 describe the main categories of the proposed classification. They discuss some representative examples and highlight current research trends. Finally, Sect. 14.6 summarises our categorisation.

14.2 Previous Classifications

The first attempt to classify hyper-heuristics was reported in [94], where two categories are proposed: (1) with learning, and (2) without learning. Hyper-heuristics without learning include approaches that use several heuristics (neighbourhood structures), but select the heuristics to call according to a predetermined sequence. Therefore, this category contains approaches such as variable neighbourhood search [47]. The hyper-heuristics with learning include methods that dynamically change the preference of each heuristic based on their historical performance, guided by some learning mechanism. As discussed in [94], hyper-heuristics can be further classified with respect to the learning mechanism employed, and a distinction is made between approaches which use a genetic algorithm, from those which use other mechanisms. This is because several hyper-heuristics have been based on genetic algorithms. In these genetic algorithm-based hyper-heuristics the idea is to evolve the solution methods, not the solutions themselves.

In [79], hyper-heuristics are classified into those which are *constructive* and those which are *local search* methods. Constructive hyper-heuristics build a solution incrementally by adaptively selecting heuristics, from a pool of constructive heuristics, at different stages of the construction process. Local search hyper-heuristics, on the other hand, start from a complete initial solution and iteratively select, from a set of neighbourhood structures, appropriate heuristics to lead the search in a promising direction.

When genetic programming started being used for hyper-heuristic research in the late 2000s (see [20] for an overview), a new class of hyper-heuristics emerged. This class was explicitly and independently mentioned in [5] and [22]. In the first class of heuristics, or 'heuristics to choose heuristics', the framework is provided with a set of pre-existing, generally widely known heuristics for solving the target problem. In contrast, in the second class, the aim is to generate new heuristics from a set of *building-blocks*, *components* or search trail of known heuristics, which are given to the framework. Therefore, the process requires, as in the first class of hyper-heuristics, the selection of a suitable set of heuristics known to be useful in solving the target problem. However, instead of supplying these directly to the framework, the heuristics are first decomposed into their basic components. Genetic programming hyper-heuristic researchers [5, 20, 22] have also made the distinction between 'disposable' and 'reusable' heuristics. A disposable heuristic is created just for one problem, and is not intended for use on unseen problems. Alternatively, the heuristic may be created for the purpose of re-using it on new unseen problems of a certain class.

In [27], hyper-heuristics are classified into four categories: (1) hyper-heuristics based on the random choice of low-level heuristics, (2) greedy and peckish hyper-heuristics, which require preliminary evaluation of all or a subset of the heuristics in order to select the best performing one, (3) metaheuristics based hyper-heuristics, and (4) hyper-heuristics employing learning mechanisms to manage low level heuristics.

14.3 The Proposed Classification and Definition

Building upon some of the previous classifications discussed above, and realising that hyper-heuristics lie at the interface of optimisation and machine learning research, we propose a general classification of hyper-heuristics according to two considerations: (1) the nature of the heuristic search space, and (2) the source of feedback during learning. These considerations are orthogonal in that different heuristic search spaces can be combined with different sources of feedback, and thus different machine learning techniques.

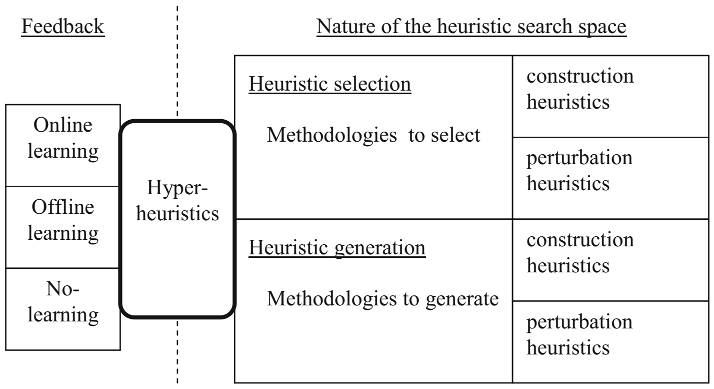

Fig. 14.1 A classification of hyper-heuristic approaches, according to two considerations: (1) the nature of the heuristic search space, and (2) the source of feedback during learning

The most fundamental hyper-heuristic categories from the previous classifications, are those represented by the processes of:

- *Heuristic selection*: Methodologies for choosing or selecting existing heuristics
- *Heuristic generation*: Methodologies for generating new heuristics (from primitive components or observed search trails of existing heuristics)

There is no reason why the higher level strategy (for selecting or generating heuristics) should be a heuristic. Indeed, sophisticated knowledge-based techniques such as case-based reasoning have been employed in this way for university timetabling [15]. This leads us to propose the following more general definition of the term ‘hyper-heuristic’ which is intended to capture the idea of a method for automating the heuristic design and selection process:

A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve computational search problems.

From this definition, there are two clear categories of hyper-heuristics aimed at either (1) heuristic selection or (2) heuristic generation. This is the first point when considering the nature of the search space. The second point is the distinction between constructive and local search hyper-heuristics, also discussed in Sect. 14.2. Notice that this categorisation is concerned with the nature of the low-level heuristics used in the hyper-heuristic framework. Our classification uses the terms *construction* and *perturbation* to refer to these classes of low-level heuristics. Sections 14.4 and 14.5 describe these categories in more detail, discussing some concrete examples of recent approaches reported in the literature.

There is an underlying theme to these two clear categories of hyper-heuristics. In both cases a high-level hyper-heuristic operates on a set of heuristics which in turn operate on the solution space. In the case of selection, we are choosing from a set of atomic predefined heuristics. In the case of generation, we are operating on a space of heuristic components.

We consider a hyper-heuristic to be a learning algorithm when it uses feedback information on the performance of the low-level heuristics from the search process. A non-learning hyper-heuristic selects a heuristic to apply uniformly at random or in a prefixed order from the existing pool, without keeping a record of their previous performance. According to the source of the feedback during learning about the low-level heuristics performance, we propose a distinction between *online* and *offline* learning. Notice that in the context of heuristic generation methodologies, an example of which is genetic programming-based hyper-heuristics (discussed in Sect. 14.2), the notions of *disposable* and *reusable* are analogous to those of online and offline learning, as described below:

Online learning hyper-heuristics: The learning takes place while the algorithm is solving an instance of a problem. Therefore, task-dependent local properties can be used by the high-level strategy to determine the appropriate low-level heuristic to apply.

Offline learning hyper-heuristics: The idea is to gather knowledge in the form of rules or programs, from a set of representative training instances, that we would expect to generalise to the process of solving unseen instances.

The proposed classification of hyper-heuristic approaches can be summarised as follows (see also Fig. 14.1):

- With respect to the nature of the heuristic search space
  - **Heuristic selection methodologies**: Produce combinations of pre-existing construction or perturbation heuristics.
  - **Heuristic generation methodologies**: Generate new heuristic methods using basic components/building-blocks or search trail of pre-existing construction or perturbation heuristics.
- With respect to the source of feedback during learning
  - **Online learning hyper-heuristics**: Learn while solving a given instance of a problem.

– **Offline learning hyper-heuristics**: Learn, from a set of training instances, a method that would generalise to unseen instances.
– **No-learning hyper-heuristics**: Do not use previous information from the search process on the low-level heuristics performance.

These categories reflect the most common research trends. However, there are methodologies that can cut across categories. For example, we can see hybrid methodologies that combine constructive with perturbation heuristics [43], or heuristic selection with heuristic generation [56, 63, 77, 88].

14.4 Heuristic Selection Methodologies

This section is not intended to be an exhaustive survey. The intention is to present a few examples to give the reader a flavour of the research that has been undertaken in this area. Some research trends are also highlighted.

14.4.1 Approaches Based on Construction Low-Level Heuristics

These approaches build a solution incrementally. Starting with an empty solution, the goal is to intelligently select and use construction heuristics to gradually build a complete solution. The hyper-heuristic framework is provided with a set of preexisting (generally problem specific) construction heuristics, and the challenge is to select the heuristic that is somehow the most suitable for the current problem state. This process continues until the final state (a complete solution) is obtained. Notice that there is a natural end to the construction process, that is, when a complete solution is reached. Therefore, the sequence of heuristic choices is finite and determined by the size of the underlying combinatorial problem. Furthermore, there is the interesting possibility of learning associations between partial solution stages and adequate heuristics for those stages.

Several approaches have been proposed to generate efficient hybridisations of existing construction heuristics in domains such as bin packing [62, 81], timetabling [15, 19, 74, 80, 82], production scheduling [101], and stock cutting [99, 100]. Both online and offline machine learning approaches have been investigated. Examples of online approaches are the use of metaheuristics in a search space of construction heuristics like genetic algorithms [37, 49, 98, 101], tabu search [19] and other single-point based search strategies [76]. For this type of hyper-heuristic, the structure of the heuristic search space or hyper-heuristic landscape has been studied [67]. Examples of offline techniques are the use of learning classifier systems [62, 81], messy genetic algorithms [80, 82, 100] and case-based reasoning [15].

14.4.1.1 Representative Examples

Two hyper-heuristics based on construction heuristics are described here in more detail. The first approach is online and is based on graph-colouring heuristics for timetabling problems, whilst the second is offline and is based on bin packing heuristics.

Graph-Colouring Hyper-Heuristic for Timetabling In educational timetabling, a number of courses or exams need to be assigned to a number of timeslots, subject to a set of both hard and soft constraints. Timetabling problems can be modelled as graph colouring problems, where nodes in the graph represent events (e.g. exams), and edges represent conflicts between events. Graph heuristics in timetabling use the information in the graph to order the events by their characteristics (e.g. number of conflicts with other events or the degree of conflict), and assign them one by one into the timeslots. These characteristics suggest how difficult it is to schedule the events. Therefore, the most difficult event, according to the corresponding ordering strategy, will be assigned first. The graph-based hyper-heuristic developed in [19], implements the following five graph colouring-based heuristics, plus a random ordering heuristic:

- *Largest Degree (LD)*: Orders the events in decreasing order based on the number of conflicts the event has with the other events.
- *Largest Weighted Degree (LW)*: The same as *LD*, but the events are weighted by the number of students involved.
- *Colour Degree (CD)*: Orders the events in decreasing order in terms of the number of conflicts (events with common students involved) each event has with those already scheduled.
- *Largest Enrolment (RO)*: Orders the events in decreasing order based on the number of enrolments.
- *Saturation Degree (SD)*: Orders the events in increasing order based on the number of timeslots available for each event in the timetable at that time.

A candidate solution in the heuristic search space corresponds to a sequence (list) of these heuristics. The solution (timetable) construction is an iterative process where, at the ith iteration, the ith graph-colouring heuristic in the list is used to order the events not yet scheduled at that step. The first e events in the ordered list are then assigned to the first e least-cost timeslots in the timetable (see Fig. 14.2).

Tabu Search is employed as the high-level search strategy for producing good sequences of the low-level heuristics. Each heuristic list produced by tabu search is evaluated by sequentially using the individual heuristics to order the unscheduled events, and thus construct a complete timetable. Since each heuristic in the list is used to schedule a number e of events, the length of the heuristic list is n/e where n is the number of events to be scheduled. Tests were performed for $e = 1,\ldots,5$ (details can be found in [19]). This work also highlights the existence of two search spaces in constructive hyper-heuristics (the heuristic space and the problem solution space). The approach was tested on both course and exam timetabling benchmark

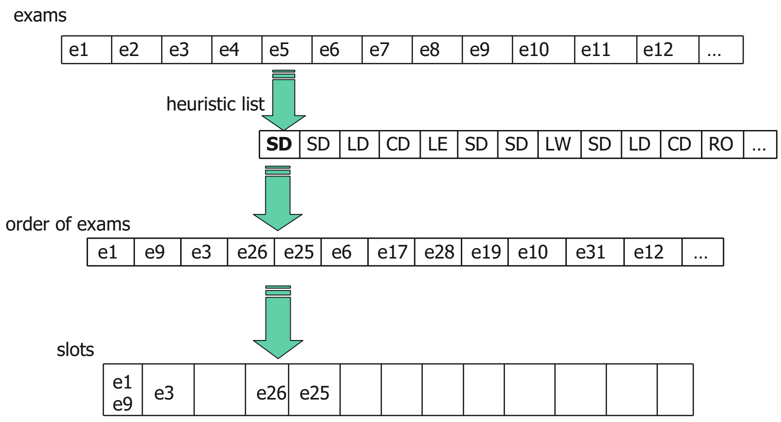

Fig. 14.2 A solution (timetable) is constructed by iteratively considering each heuristic in the list, and using it to order the events not yet scheduled. The first *e* events (in the figure $e = 5$) in the resulting ordering are assigned to the first *e* least-cost timeslots in the timetable

instances with competitive results. This graph-based hyper-heuristic was later extended in [76] where a formal definition of the framework is presented. The authors also compare the performance of several high-level heuristics that operate on the search space of heuristics. Specifically, a best-improvement hill-climber, iterated local search and variable neighbourhood search are implemented and compared to the previously implemented tabu search. The results suggest that the choice of a particular neighbourhood structure on the heuristic space is not crucial to the performance. Moreover, iterative techniques such as iterated local search and variable neighbourhood search, were found to be more effective for traversing the heuristic search space than more elaborate metaheuristics such as tabu search. The authors suggest that the heuristic search space is likely to be smooth and to contain large plateaus (i.e. areas where different heuristic sequences produce solutions of similar quality). The work also considers hybridisations of the hyper-heuristic framework with local search operating on the solution space. This strategy greatly improves the performance of the overall system, making it competitive with state-of-the-art approaches on the studied benchmark instances.

In a further study [67], the notion of fitness landscapes is used to analyse the search space of graph colouring heuristics. The study confirms some observations about the structure of the heuristic search space discussed in [76]. Specifically, these landscapes have a high level of neutrality (i.e. the presence of plateaus). Furthermore, although rugged, they have the encouraging feature of a globally convex or *big valley* structure, which indicates that an optimal solution would not be isolated but surrounded by many local minima. The study also revealed a *positional bias* in the search space made of sequences of heuristics. Specifically, changes in the earlier positions of a heuristic sequence have a larger impact on the solution quality than changes in the later positions. This is because early decisions (heuristic choices) in a construction process have a higher impact on the overall quality of the solution than later decisions.

Classifier System Hyper-Heuristic for Bin Packing Classifier systems are rule-based learning systems that evolve fixed length stimulus-response rules. The rules are encoded as ternary strings, made of the symbols $\{0, 1, \#\}$, and have an associated strength. The system operates in two phases. First, the population of classification rules is applied to some task; and secondly, a genetic algorithm generates a new population of rules by selection based on strength, and by the application of standard genetic operators. Calculating the strength of each rule is a *credit assignment problem*, which refers to determining the contribution made by each sub-component or partial solution, in decomposable problems being solved collectively by a set of partial solutions.

In [81], a classifier system was used in the domain of one-dimensional bin packing, to learn a set of rules that associate characteristics of the current state of a problem with different low-level construction heuristics. In the one-dimensional bin packing problem, there is an unlimited supply of bins, each with capacity C. A set of n items is to be packed into the bins, the size of each item is given, and items must not overfill any bin. The task is to minimise the total number of bins required.

The set of rules evolved by the classifier system is used as follows: given the initial problem characteristics P, a heuristic H is chosen to pack an item, thus gradually altering the characteristics of the problem that remains to be solved. At each step a rule appropriate to the current problem state P' is selected, and the process continues until all items have been packed. For the training phase a total of 890 benchmark instances from the literature were used. The authors chose four bin packing heuristics from the literature, the selection being based on those that produced the best results on the studied benchmark set. These heuristics were as follows:

- *Largest-Fit-Decreasing*: Items are taken in order of size, largest first, and are put in the first bin where they fit (a new bin is opened if necessary).
- *Next-Fit-Decreasing*: An item is packed into the current bin if possible, or else a new bin is opened into which the item is placed. This new bin becomes the current bin.
- *Djang and Finch's (DJD)*: A heuristic that considers combinations of up to three items to completely fill partially full bins.
- *A Variation of DJD*: A variation of the previous heuristic that considers combinations of up to five items to completely fill partially full bins.

A simplified description of the current state of the problem is proposed. This description considers the number of items remaining to be packed, and calculates the percentage of items in each of four size ranges (huge, large, medium, and small), where the size of the items is judged in proportion to the bin size. The approach used single-step environments, meaning that rewards were available after each action had taken place. The classifier system was trained on a set of example problems and showed good generalisation to unseen problems. In [62], the classifier system approach is extended to multi-step environments. The authors tested several reward schemes in combination with alternate exploration/exploitation ratios, and several sizes and types of multi-step environments. Again, the approach was tested using a large set of one-dimensional benchmark bin packing problems. The classifier system was able to generalise well and create solution processes that performed well on a large set of NP-hard benchmark instances. The authors report that multi-step environments can obtain better results than single-step environments at the expense of a higher number of training cycles.

14.4.2 Approaches Based on Perturbation Low-Level Heuristics

These approaches start with a complete solution, generated either randomly or using simple construction heuristics, and thereafter try to iteratively improve the current solution. The hyper-heuristic framework is provided with a set of neighbourhood structures and/or simple local searchers, and the goal is to iteratively select and apply them to the current complete solution. This process continues until a stopping condition is met. Notice that these approaches differ from those based on construction heuristics, in that they do not have a natural termination condition. The sequence of heuristic choices can, in principle, be arbitrarily extended. This class of hyper-heuristics has the potential to be applied successfully to different combinatorial optimisation problems, since general neighbourhood structures or simple local searchers can be made available. Hyper-heuristics based on perturbation have been applied to personnel scheduling [14, 28], timetabling [7, 9, 23, 74, 84, 90, 91], shelf space allocation [6, 8], packing [7, 34] and vehicle routing problems [43, 75, 88].

So far, the approaches that combine perturbation low-level heuristics in a hyper-heuristic framework use online learning, in that they attempt to adaptively solve a single instance of the problem under consideration. Furthermore, the majority of the proposed approaches are single-point algorithms, in that they maintain a single incumbent solution in the solution space. Some approaches that maintain a population of points in the heuristic space have been attempted [29, 103].

As suggested in [71, 72] perturbation hyper-heuristics can be separated into two processes: (1) (low-level) heuristic selection, and (2) move acceptance strategy. Thus, the authors classify hyper-heuristics with respect to the nature of the heuristic selection and move acceptance components. The heuristic selection can be done in a *non-adaptive* (simple) way: either randomly or along a cycle, based on a prefixed heuristic ordering [28]. No learning is involved in these approaches. Alternatively, the heuristic selection may incorporate an *adaptive* (or on-line learning) mechanism based on the probabilistic weighting of the low-level heuristics [14, 65, 75], or some type of performance statistics [28]. Both non-adaptive and adaptive heuristic selection schemes are generally embedded within a single-point local search high-level heuristic.

The acceptance strategy is an important component of any local search heuristic. Many acceptance strategies have been explored within hyper-heuristics. Move acceptance strategies can be divided into two categories: *deterministic* and *non-*

deterministic. In general, a move is accepted or rejected, based on the quality of the move and the current solution during a single point search. At any point in the search, deterministic move acceptance methods generate the same result for the same candidate solution(s) involved in the acceptance test. However, a different outcome is possible if a non-deterministic approach is used. If the move acceptance test involves other parameters, such as the current time, then these strategies are referred to as non-deterministic strategies. Well known meta-heuristic components are used as non-deterministic acceptance methods such as simulated annealing [34, 75].

14.4.2.1 Representative Examples

Two hyper-heuristics based on perturbation heuristics are described here. The first is applied to a real-world packing problem, whilst the second uses large neighbourhood heuristics and is applied to five variants of the well known vehicle routing problem.

A Simulated Annealing Hyper-Heuristic for Determining Shipper Sizes In [34] the tabu search hyper-heuristic, originally presented in [14], is integrated within a simulated annealing framework. That is, a simulated annealing acceptance strategy is combined with the previously proposed heuristic selection mechanism. Figure 14.3 outlines the simulated annealing-based hyper-heuristic, illustrating the *domain barrier* that separates the high-level search strategy from the underlying problem domain. The framework requires a repository of low-level heuristic from which the high-level strategy selects and applies to the incumbent solution considering only domain-independent information on the performance of each heuristic.

The tabu search hyper-heuristic [14], selects the low-level heuristics according to learned utilities or ranks. The framework also incorporates a dynamic tabu list of low-level heuristics that are temporarily excluded from the selection pool. The algorithm deterministically selects the low-level heuristic with the highest rank (not included in the tabu list), and applies it once regardless of whether the selected move causes an improvement or not (all moves acceptance). If there is an improvement, the rank is increased. If the new solution is worse, the rank of the low-level heuristic is decreased and it is made tabu. The rank update scheme is additive, and the tabu list is emptied each time a non-improvement move is accepted. This general tabu search approach was evaluated on various instances of two distinct timetabling and rostering (personal scheduling) problems, and the obtained results were competitive with those obtained using state-of-the-art problem-specific techniques. Apart from the simulated annealing acceptance criteria, some modifications are also introduced in [34]. In particular, a single application of a low-level heuristic h, is defined to be k iterations of h. Therefore, the decision points are set every k iterations, and the feedback for updating the quality of heuristic h is based on the best cost obtained during those k iterations. Additionally, a non monotonic cooling schedule is proposed to deal with the effects of having different neighbourhood sizes (given by the pool of low-level heuristics used). The methodology was applied to a packing problem in the cosmetics industry, where the shipper sizes for storage and transportation had to be determined. Real data was used for generating the instances, and the approach was compared with a simpler local search strategy (random descent), with favourable results.

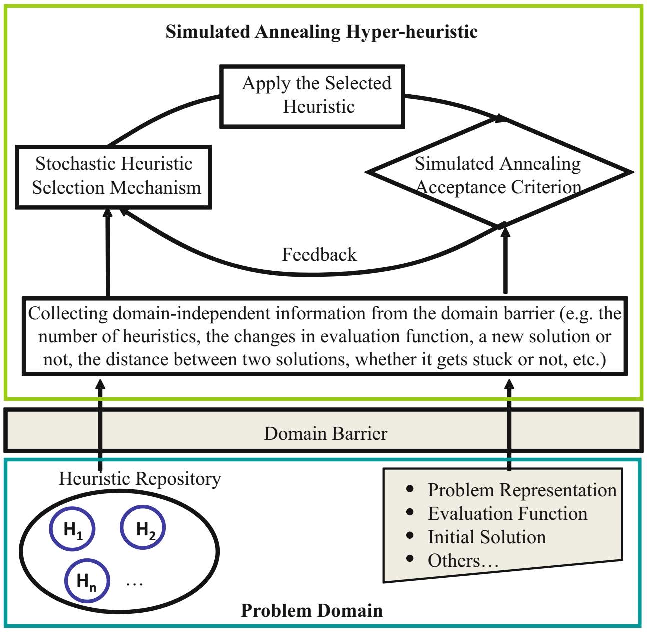
Simulated Annealing Hyper-heuristic
Apply the Selected Heuristic
Stochastic Heuristic Selection Mechanism
Simulated Annealing Acceptance Criterion
Feedback
Collecting domain-independent information from the domain barrier (e.g. the number of heuristics, the changes in evaluation function, a new solution or not, the distance between two solutions, whether it gets stuck or not, etc.)
Domain Barrier
Heuristic Repository
H1
H2
Hn
…
Problem Representation
Evaluation Function
Initial Solution
Others…
Problem Domain

Fig. 14.3 A simulated annealing hyper-heuristic framework

A General Heuristic for Vehicle Routing Problems In [75], a unified methodology is presented, which is able to solve five variants of the vehicle routing problem: the vehicle routing problem with time windows, the capacitated vehicle routing problem, the multi-depot vehicle routing problem, the site-dependent vehicle routing problem and the open vehicle routing problem. All problem variants are transformed into a rich pickup and delivery model and solved using an adaptive large neighbourhood search methodology [78]. The general framework is outlined in Fig. 14.4, where the repeat loop corresponds to the high-level local search framework. Implementing a simulated annealing algorithm is straightforward as one solution is sampled in each iteration of the algorithm. The acceptance strategy considers a standard exponential cooling rate. In each iteration of the main loop, the algorithm chooses one destroy ($N-$) and one repair neighbourhood ($N+$). An adaptive layer stochastically controls which neighbourhoods to choose according to their past performance (score, P_i). The more the neighbourhood N_i has contributed to the solution process, the larger the score P_i it obtains, and hence the larger the probability of being chosen. The adaptive layer uses roulette wheel selection for choosing a destroy and a repair neighbourhood.

The pickup and delivery model is concerned with serving a number of transportation requests using a limited number of vehicles. Each request involves moving a number of goods from a pickup location to a delivery location. The task is to construct routes that visit all locations such that the corresponding pickups and deliveries are placed on the same route and such that a pickup is performed before the corresponding delivery. Different constraints are added to model the different problem variants. A large number of tests were performed on standard benchmarks from the literature on the five variants of the vehicle routing problem. The results proved to be highly promising, as the methodology was able to improve on the best known solutions of over one third of the tested instances.

```
Construct a feasible solution x; set x*:=x
Repeat
  Choose a destroy and a repair neighbourhood: N- and N+
     based on previously obtained scores (Pi)
  Generate a new solution x′ from x using the heuristics
    corresponding to the chosen destroy and repair neighbourhoods
  If x′ can be accepted then set x:=x′
  Update scores Pi of N- and N+
  If f(x) < f(x*) set x*:=x
Until a stopping criterion is met
return x*
```

Fig. 14.4 Outline of the Adaptive Large Neighbourhood framework. $N-$ and $N+$ correspond to destroy and repair neighbourhoods, respectively. P_i stands for the score associated to heuristic i

14.4.3 Recent Research Trends

The most prominent approaches for heuristic selection have focused on a high-level search strategy resembling an existing metaheuristic such as simulated annealing, variable neighbourhood or genetic algorithms, which searches in a space of simple low-level heuristics. In recent years, new hyper-heuristic methodologies, such as the use of Monte Carlo Search [23, 83] have been explored. In addition, new domains such as software engineering [51, 103, 104], game playing [58], competitive travelling salesman problem [54], and DNA sequencing [10] have been studied. Attention has also been devoted to developing software frameworks, considering multi-objective problems and studying the theoretical foundations of these methods, as discussed below.

14.4.3.1 Software Frameworks

The *HyFlex* (Hyper-heuristic Flexible) framework [68], features a common software interface for dealing with different combinatorial optimisation problems, and provides the algorithm components that are problem specific. The algorithm designer does not require a detailed knowledge of the problem domains, and thus can concentrate his/her efforts on designing adaptive general-purpose optimisation algorithms. HyFlex provides six combinatorial problems implemented in Java, namely: Boolean satisfiability, one dimensional bin packing, permutation flow shop, personnel scheduling, travelling salesman and vehicle routing. These domains supported an international research competition: the first Cross-Domain Heuristic Search Challenge (CHeSC) [66]. The framework has been widely used by the research community [25], an extension to the framework was proposed in [69] and three new domains have been recently added: the 0-1 knapsack, quadratic assignment and max-cut problem [1].

In [96] a unified object-oriented formulation of hyper-heuristics is introduced, which includes perturbative and constructive heuristics, as well as selection and generation heuristics. The goal is to extend the software design space of hyper-heuristic algorithms.

14.4.3.2 Multi-Objective

There are emerging studies on multiobjective optimisation approaches mixing a set of existing multiobjective metaheuristics. For example, [102] describes a multialgorithm, genetically adaptive multiobjective method applied to benchmark function optimisation using NSGA-II [31], particle swarm optimisation, adaptive Metropolis search, and differential evolution. This study demonstrates that combining standard metaheuristic algorithms can be better than using each one is isolation, as well as being competitive with other state-of-the-art methodologies on a set of benchmark functions. However, it has been once again observed in [59] that the choice of low level (meta)heuristics influences the overall performance of a hyper-heuristic. This latter study considers a variety of selection hyper-heuristics for multi-objective optimisation. The results showed that combining simple random choice and great deluge move acceptance from [61] is the best performing approach using three modern multi-objective evolutionary algorithms as low level metaheuristics. This approach is additionally tested on a real world wind farm layout optimisation problem.

14.4.3.3 Theoretical and Foundational Studies

The structure of heuristic search spaces has been studied using the notion of fitness landscapes [67]. These landscapes are found to have a high level of neutrality (i.e. the presence of plateaus). Furthermore, although rugged, they have the encouraging feature of a globally convex or *big valley* structure, which indicates that there is a gradient guiding search towards good solutions. More recently, the search space of Boolean satisfiability (MAX-SAT) heuristics was analysed [26]. Using systematic enumeration of millions of heuristics, they gave evidence that the heuristic landscape had many clusters of good local optima, and was also amenable to other search methods, such as (iterated) hill-climbing.

A series of formal run-time analysis of simplified hyper-heuristic algorithms has been conducted. In [50], the authors show the improved performance of a (1+1) evolutionary algorithms using a strategy combining multiple operators as compared to a strategy employing a single mutation operator with respect to a performance metric referred to as *asymptotic hitting time*. In [57], a rigorous run-time analysis of hyper-heuristic mixing heuristics (operators) was performed, showing that there is some value to mixing heuristics leading to exponentially faster search than individual heuristics on some problems. In [2], theoretical analyses showed that alternatives to the additive updates in reinforcement learning, which is a commonly used scheme for heuristic selection, should be considered, since this strategy behaves in an asymptotically similar way to random choice based on the assumption that the probability of improving a solution at each step is less than 0.5. *Heuristic space diversity* is the focus of [46].

A fundamental open question in selection hyper-heuristics is which low-level heuristics (and how many) to use as part of the pool. In [3], the authors use tensor analysis as an advanced machine learning approach to decide on the subset of low level heuristics operating effectively with chosen deterministic move acceptance, yielding improved results. A methodology for selecting a subset of effective heuristics from a given larger set is proposed in [92]. The approach considers nonparametric statistics and fitness landscape measurements from an available set of heuristics and benchmark instances, and it produces a compact subset of effective heuristics with improved performance for the underlying problem.

14.5 Heuristic Generation Methodologies

This section provides some examples of approaches that have the potential to automatically generate heuristics for a given problem. Many of the approaches in the literature to generate heuristics use genetic programming [20], a branch of evolutionary computation concerned with the automatic generation of computer programs. Genetic programming has been successfully applied to the automated generation of heuristics that solve hard combinatorial optimisation problems, such as Boolean satisfiability, [5, 40–42, 55], bin packing [16, 17, 22, 60, 89], the traveling salesman problem [52, 53] and production scheduling [11, 33, 44, 48, 97]. In addition to the particular representation, using trees, graphs, grammars or linear program encodings, genetic programming differs from other evolutionary approaches in its application area. While most applications of evolutionary algorithms deal with optimisation problems, genetic programming could instead be positioned in the field of machine learning.

Some genetic programming-based hyper-heuristics have evolved local search heuristics [5, 24, 41, 42, 52, 53] or even evolutionary algorithms [70]. An alternative idea is to use genetic programming to evolve a program representing a function, which is part of the processing of a given problem specific construction heuristic [16, 17, 33, 44, 97]. Most examples of using genetic programming as a hyper-heuristic are offline in that a training set is used for generating a program that acts as a heuristic, which is thereafter used on unseen instances of the same problem. That is, the idea is to generate *reusable* heuristics. However, research on *disposable* heuristics has also been conducted [5, 52, 53]. In other words, heuristics are evolved for solving a single instance of a problem. This approach is analogous to the online heuristic selection methodologies discussed in Sect. 14.4, except that a new heuristic is generated for each instance, instead of choosing a sequence of heuristics from a predefined set.

The adaptation of heuristic orderings can also be considered as a methodology for heuristic generation. The adaptive approach proposed in [12], starts with one heuristic and adapts it to suit a particular problem instance 'on the fly'. This method provides an alternative to existing forms of 'backtracking', which are often required to cope with the possible unsuitability of a heuristic. The adaptive method is more general, significantly easier to implement, and produces results that are at least comparable (if not better) than the current state-of-the-art examination timetabling algorithms.

14.5.1 Representative Examples

Two representative examples of heuristic generation using genetic programming are discussed below. The first evolves packing heuristics that operate on a constructive framework. The second evolves complete local search algorithms, using components of successful, existing local search heuristics for Boolean satisfiability.

Generation of Construction Heuristics for Bin Packing The one-dimensional bin packing problem involves a set of n items, which must be packed into bins of a certain capacity C, using the minimum number of bins possible. In the online version of the problem, the number of items and their sizes are not known in advance. This is in contrast to the offline version of the problem where the set of items to be packed is available at the start. An example of a construction heuristic used in online bin packing is first-fit, which packs a set of items one at a time, in the order that they are presented. The heuristic iterates through the open bins, and the current item is placed in the first bin into which it fits.

In [16, 17], construction heuristics are evolved for the online bin packing problem. The evolved heuristics, represented as trees (see Fig. 14.5 for an example), operate within a fixed framework that resembles the operation of the first-fit heuristic discussed above. The key idea is to use the attributes of the items and bin capacities, that represent the state of the problem, in order to evolve functions (expressions) that would direct the packing process. Each evolved function (GP tree) is applied in turn to the available bins, returning a value. If the value is less than or equal to zero then the system moves on to the next bin, but if the value is positive the item is packed into the current bin. In this way, the function decides when to stop the search for a suitable bin for the item. The algorithm (depicted in Fig. 14.6) then repeats the process for each of the other items until all the items have been packed.

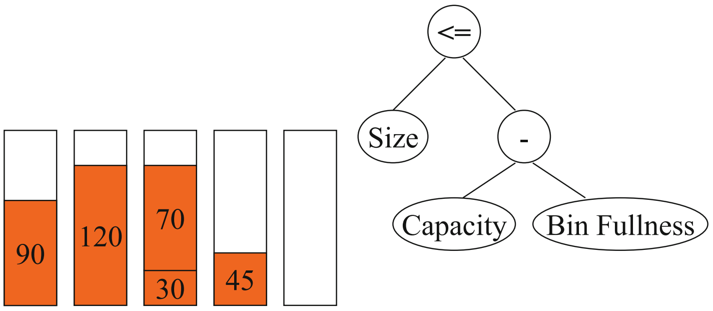

Fig. 14.5 Evolving one-dimensional packing heuristics with genetic programming. The tree illustrates an evolved heuristic, while the bins indicate an example current state of the online bin-packing solving process

In a genetic programming framework, the set of terminals and functions need to be specified. The hyper-heuristic framework for online bin packing uses attributes that describe the state of the problem to define the terminals. In [16], the authors use the following terminals:

- S, the size of the current item,
- C, the capacity of a bin (this is a constant for the problem) and,
- L, the load of a bin (i.e. the total size of all of the items occupying that bin).

Later [18], these three attributes were replaced by only two attributes: S, the size of the current item and E ($= C - L$), the residual capacity of a bin (i.e. how much space is remaining in the bin). The function set used in [16, 17] consists of $\leq, +, -, \times, \%$, where $\%$ is the 'protected divide function'. The results in [16] show that a simple genetic programming system can discover human designed heuristics such as first-fit, whilst in [18], heuristics that outperformed first-fit were evolved. In [17], it was also shown empirically that the choice of the training instances (categorised according to the item size distribution) has an impact on the trade-off between the performance and generality of the heuristics generated and their applicability to new problems.

```
For each item p
  For each bin b
    output := evaluate Heuristic
    If (output > 0)
      place item p in bin b
      break
    End If
  End For
End For
```

Fig. 14.6 Pseudo code showing the overall program structure within which an evolved packing heuristic operates

Generation of Local Search Heuristics for Satisfiability Testing The Boolean satisfiability problem consists of finding the true/false assignments of a set of Boolean variables, to decide if a given propositional formula or expression (in conjunctive normal form) can be satisfied. The problem, denoted as SAT, is a classic NP-complete problem.

In [40–42] a genetic programming system, named CLASS (Composite Learned Algorithms for SAT Search), is proposed which automatically discovers new SAT local search heuristics. Figure 14.7 illustrates a generic SAT local search algorithm, where the 'key detail' is the choice of a *variable selection heuristic* in the inner loop. Much research in the past decade has focused on designing a better variable selection heuristic, and as a result, the performance of local search heuristics have improved dramatically. The CLASS system was developed in order to automatically discover variable selection heuristics for SAT local search. It was noted in [40] that many of the best-known SAT heuristics (such as GSAT, HSAT, Walksat, and Novelty [42]) could be expressed as decision tree-like combinations of a set of primitives. Thus, it should be possible for a machine learning system to automatically discover new, efficient variable selection heuristics by exploring the space of combinations of these primitives. Examples of the primitives used in human designed SAT heuristics are the gain obtained by flipping a variable (i.e. the increase in the number of satisfied clauses in the formula) or the age of a variable (i.e. how long since it was last flipped).

The results using CLASS [42] show that a simple genetic programming system is able to generate local search heuristics that are competitive with efficient implementations of state-of-the-art heuristics (e.g., Walksat and Novelty variants), as well as previous evolutionary approaches. The evolved heuristics scale and generalise fairly well on random instances as well as more structured problem classes.

```
A:= randomly generated truth assignment
While termination condition is not met
  If A satisfies formula then return A
    v:= Choose variable using
        "variable selection heuristic"
    A:= A with value of v inverted
  End If
End While
return FAILURE (no assignment found)
```

Fig. 14.7 A generic SAT local search algorithm. The "variable selection heuristic" is replaced by the evolved function

14.5.2 Some Recent Examples

The most common approach so-far for generating heuristics has been tree-based genetic programming. Recently other genetic programming approaches have been used such as grammar-based [93], gene expression programming [86, 87], and grammatical evolution [24, 85].

Other machine learning techniques and representations have also been employed. In [4, 73], the authors applied a generic genetic algorithm which creates and searches heuristics in the form of policy matrices, communicating with an online bin packing simulator for evaluation. The empirical results show that the generated heuristics are specialised to the distribution of item sizes and outperform the existing human designed heuristics.

A lifelong learning approach is proposed in [89], where an artificial immune system is combined with genetic programming in a system that continuously generates new heuristics and samples problems from its environment. The system is successfully tested on a large set of dynamically changing one-dimensional bin packing instances. A system that evolves an ensemble of heuristics for the job-shop scheduling problem is presented in [48]. The ensemble adopts a divide-and-conquer approach in which each heuristic solves a unique subset of the instance set considered. The system incorporates a heuristic generator that evolves heuristics composed of linear sequences of dispatching rules. Following a training period, the ensemble is shown to outperform both existing dispatching rules and a standard genetic programming algorithm on a large set of new test instances.

A new application domain includes generating strategies for games, such as the work in [35] where heuristics are evolved to guide staged deepening search for the hard game of FreeCell, obtaining solvers that outperform human players in this challenging puzzle.

14.6 Conclusions

The defining feature of hyper-heuristic research is that it investigates methodologies that operate on a search space of heuristics rather than directly on a search space of problem solutions. This feature provides the potential for increasing the level of generality of search methodologies. Several hyper-heuristic approaches have been proposed that incorporate different search and machine learning paradigms. We have suggested an updated definition of the term ‘hyper-heuristic’ to reflect recent work in the area.

With the incorporation of genetic programming and other machine learning methods, a new class of approaches can be identified; that is, heuristic generation methodologies. These approaches provide richer heuristic search spaces, and thus the freedom to create new methodologies for solving the underlying combinatorial problems. However, they are more difficult to implement than their counterpart, heuristic selection methodologies, since they require the decomposition of existing heuristics, and the design of an appropriate framework.

We have further categorised the two main classes of hyper-heuristics (heuristic *selection* and heuristic *generation*), according to whether they use *construction* or *perturbation* low-level heuristics. These categories describe current research trends. However, the possibilities are open for the exploration of hybrid approaches. We also considered an additional orthogonal criterion for classifying hyper-heuristics with respect to the source of the feedback during the learning process, which can be either one instance (online approaches) or many instances of the underlying problem (offline approaches). Both online and offline approaches are potentially useful and therefore worth investigating. Although having a reusable method will increase the speed of solving new instances of problems, using online (or disposable) methods can have other advantages. In particular, searching over a space of heuristics may be more effective than directly searching the underlying problem space, as heuristics may provide an advantageous search space structure. Moreover, in newly encountered problems there may not be a set of related instances on which to train off-line hyper-heuristic methods.

Hyper-heuristic research lies at the interface between search methodologies and machine learning methods. Machine learning is a well established artificial intelligence sub-field with a wealth of proven tools. The exploration of these techniques for automating the design of heuristics is only in its infancy. We foresee increasing interest in these methodologies in the coming years.

References

1. S. Adriaensen, G. Ochoa, A. Nowé, A benchmark set extension and comparative study for the hyflex framework, in *IEEE Congress on Evolutionary Computation, CEC* (2015), pp. 784–791

2. F. Alanazi, P.K. Lehre, Limits to learning in reinforcement learning hyper-heuristics, in *European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP*. Lecture Notes in Computer Science, vol. 9595 (2016), pp. 170–185
3. S. Asta, E. Özcan, A tensor-based selection hyper-heuristic for cross-domain heuristic search. Inf. Sci. **299**, 412–432 (2015)
4. S. Asta, E. Özcan, A.J. Parkes, CHAMP: creating heuristics via many parameters for online bin packing. Exp. Syst. Appl. **63**, 208–221 (2016)
5. M. Bader-El-Den, R. Poli, Generating SAT local-search heuristics using a GP hyper-heuristic framework, in *Artificial Evolution Conference, EA*. Lecture Notes in Computer Science, vol. 4926 (2007), pp. 37–49
6. R. Bai, E.K. Burke, G. Kendall, Heuristic, meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. J. Oper. Res. Soc. **59**(10), 1387–1397 (2008)
7. R. Bai, J. Blazewicz, E.K. Burke, G. Kendall, B. McCollum, A simulated annealing hyper-heuristic methodology for flexible decision support. 4OR Quart. J. Oper. Res. **10**(1), 43–66 (2012)
8. R. Bai, E.K. Burke, G. Kendall, T. van Woensel, A new model and a hyper-heuristic approach for two-dimensional shelf space allocation. 4OR Quart. J. Oper. Res. **11**(1), 31–55 (2013)
9. B. Bilgin, E. Özcan, E.E. Korkmaz, An experimental study on hyper-heuristics and exam timetabling, in *Practice and Theory of Automated Timetabling, PATAT*. Lecture Notes in Computer Science, vol. 3867 (2007), pp. 394–412
10. J. Blazewicz, E.K. Burke, G. Kendall, W. Mruczkiewicz, C. Öguz, A. Swiercz, A hyper-heuristic approach to sequencing by hybridization of DNA sequences. Ann. Oper. Res. **207**(1), 27–41 (2013)
11. J. Branke, S. Nguyen, C. Pickardt, M. Zhang, Automated design of production scheduling heuristics: a review. IEEE Trans. Evol. Comput. **20**(1), 110–124 (2016)
12. E.K. Burke, J. Newall, Solving examination timetabling problems through adaptation of heuristic orderings. Ann. Oper. Res. **129**(1–4), 107–134 (2004)
13. E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, S. Schulenburg, Hyper-heuristics: an emerging direction in modern search technology, in *Handbook of Metaheuristics*, ed. by F. Glover, G. Kochenberger (Kluwer, Dordecht, 2003), pp. 457–474
14. E.K. Burke, G. Kendall, E. Soubeiga, A tabu-search hyperheuristic for timetabling and rostering. J. Heuristics **9**(6), 451–470 (2003)
15. E.K. Burke, S. Petrovic, R. Qu, Case based heuristic selection for timetabling problems. J. Sched. **9**(2), 115–132 (2006)
16. E.K. Burke, M.R. Hyde, G. Kendall, Evolving bin packing heuristics with genetic programming, in *Parallel Problem Solving from Nature, PPS*. Lecture Notes in Computer Science, vol. 4193 (2006), pp. 860–869
17. E.K. Burke, M.R. Hyde, G. Kendall, J. Woodward, Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one, in *Genetic and Evolutionary Computation Conference, GECCO* (2007), pp. 1559–1565
18. E.K. Burke, M.R. Hyde, G. Kendall, J.R. Woodward, The scalability of evolved on line bin packing heuristics, in *IEEE Congress on Evolutionary Computation, CEC* (2007), pp. 2530–2537
19. E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyper-heuristic for educational timetabling problems. Eur. J. Oper. Res. **176**(1), 177–192 (2007)
20. E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. Woodward, Exploring hyper-heuristic methodologies with genetic programming, in *Collaborative Computational Intelligence*, ed. by C. Mumford, L. Jain (Springer, Berlin, 2009), pp. 177–201
21. E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, J.R. Woodward, *A Classification of Hyper-Heuristic Approaches* (Springer, Boston, 2010), pp. 449–468
22. E.K. Burke, M.R. Hyde, G. Kendall, J.R. Woodward, A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics. IEEE Trans. Evol. Comput. **14**(6), 942–958 (2010)
23. E.K. Burke, G. Kendall, M. Misir, E. Özcan, Monte Carlo hyper-heuristics for examination timetabling. Ann. Oper. Res. **196**(1), 73–90 (2012)

24. E.K. Burke, M.R. Hyde, G. Kendall, Grammatical evolution of local search heuristics. IEEE Trans. Evol. Comput. **16**(3), 406–417 (2012)
25. E.K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, R. Qu, Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 695–1724 (2013)
26. A.W. Burnett, A.J. Parkes, Exploring the landscape of the space of heuristics for local search in SAT, in *IEEE Congress on Evolutionary Computation CEC* (2017), pp. 2518–2525
27. K. Chakhlevitch, P.I. Cowling, Hyperheuristics: recent developments, in *Adaptive and Multilevel Metaheuristics*. Studies in Computational Intelligence, vol. 136 (Springer, Berlin, 2008), pp. 3–29
28. P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach for scheduling a sales summit, in *Selected Papers of the Third International Conference on the Practice and Theory of Automated Timetabling, PATAT 2000*. Lecture Notes in Computer Science (2001)
29. P. Cowling, G. Kendall, L. Han, An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem, in *IEEE Congress on Evolutionary Computation, CEC* (2002), pp. 1185–1190
30. W.B. Crowston, F. Glover, G.L. Thompson, J.D. Trawick, Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR research memorandum, Carnegie-Mellon University, Pittsburgh (1963)
31. K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
32. J. Denzinger, M. Fuchs, M. Fuchs, High performance ATP systems by combining several AI methods, in *International Joint Conference on Artificial Intelligence IJCAI* (1997), pp. 102–107
33. C. Dimopoulos, A.M.S. Zalzala, Investigating the use of genetic programming for a classic one-machine scheduling problem. Adv. Eng. Softw. **32**(6), 489–498 (2001)
34. K.A. Dowsland, E. Soubeiga, E.K. Burke, A simulated annealing hyper-heuristic for determining shipper sizes. Eur. J. Oper. Res. **179**(3), 759–774 (2007)
35. A. Elyasaf, A. Hauptman, M. Sipper, Evolutionary design of freecell solvers. IEEE Trans. Comput. Intell. AI Games **4**(4), 270–281 (2012)
36. H.L. Fang, P. Ross, D. Corne, A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems, in *International Conference on Genetic Algorithms, ICGA* (1993), pp. 375–382
37. H.L. Fang, P. Ross, D. Corne, A promising hybrid GA/heuristic approach for open-shop scheduling problems, in *European Conference on Artificial Intelligence, ECAI* (1994)
38. H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in *Factory Scheduling Conference*. Carnegie Institue of Technology (1961)
39. H. Fisher, G.L. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in *Industrial Scheduling*, ed. by J.F. Muth, G.L. Thompson (Cengage Learning, Boston, 1963)
40. A.S. Fukunaga, Automated discovery of composite SAT variable selection heuristics, in *AAAI Conference on Artificial Intelligence* (2002), pp. 641–648
41. A.S. Fukunaga, Evolving local search heuristics for SAT using genetic programming, in *Genetic and Evolutionary Computation, GECCO* (2004), pp. 483–494
42. A.S. Fukunaga, Automated discovery of local search heuristics for satisfiability testing. Evol. Comput. J. **16**(1), 31–61 (2008)
43. P. Garrido, M.C. Riff, DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. J. Heuristics **16**(6), 795–834 (2010)
44. C.D. Geiger, R. Uzsoy, H. Aytŭg, Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. J. Sched. **9**(1), 7–34 (2006)
45. J. Gratch, S. Chien, Adaptive problem-solving for large-scale scheduling problems: a case study. J. Artif. Intell. Res. **4**(1), 365–396 (1996)
46. J. Grobler, A.P. Engelbrecht, G. Kendall, V.S.S. Yadavalli, Heuristic space diversity control for improved meta-hyper-heuristic performance. Inf. Sci. **300**, 49–62 (2015)
47. P. Hansen, N. Mladenovic, Variable neighborhood search: principles and applications. Eur. J. Oper. Res. **130**(3), 449–467 (2001)

48. E. Hart, K. Sim, A hyper-heuristic ensemble method for static job-shop scheduling. Evol. Comput. J. **24**(4), 609–635 (2016)
49. E. Hart, P. Ross, J.A.D. Nelson, Solving a real-world problem using an evolving heuristically driven schedule builder. Evol. Comput. J. **6**(1), 61–80 (1998)
50. J. He, F. He, H. Dong, Pure strategy or mixed strategy?, in *European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP*. Lecture Notes in Computer Science, vol. 7245 (2012), pp. 218–229
51. Y. Jia, M.B. Cohen, M. Harman, J. Petke, Learning combinatorial interaction test generation strategies using hyperheuristic search, in *IEEE/ACM International Conference on Software Engineering, ICSE* (2015), pp. 540–550
52. R.E. Keller, R. Poli, Cost-benefit investigation of a genetic-programming hyperheuristic, in *Artificial Evolution Conference, EA*. Lecture Notes in Computer Science, vol. 4926 (2007), pp. 13–24
53. R.E. Keller, R. Poli, Linear genetic programming of parsimonious metaheuristics, in *IEEE Congress on Evolutionary Computation, CEC* (2007), pp. 4508–4515
54. G. Kendall, J. Li, Competitive travelling salesmen problem: a hyper-heuristic approach. J. Oper. Res. Soc. **64**(2), 208–216 (2013)
55. R.H. Kibria, Y. Li, Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming, in *European Conference on Genetic Programming, EuroGP*. Lecture Notes in Computer Science, vol. 3905 (2006), pp. 331–340
56. N. Krasnogor, S. Maturana Gustafson, A study on the use of "self-generation" in memetic algorithms. Nat. Comput. **3**(1), 53–76 (2004)
57. P.K. Lehre, E. Özcan, A runtime analysis of simple hyper-heuristics: to mix or not to mix operators, in *Workshop on Foundations of Genetic Algorithms, FOGA XII* (2013), pp. 97–104
58. J. Li, G. Kendall, A hyper-heuristic methodology to generate adaptive strategies for games. IEEE Trans. Comput. Intell. AI Games **9**(1), 1–10 (2017)
59. W. Li, E. Özcan, R. John, Multi-objective evolutionary algorithms and hyper-heuristics for wind farm layout optimisation. Renew. Energy **105**, 473–482 (2017)
60. E. Lopez-Camacho, H. Terashima-Marin, P. Ross, G. Ochoa, A unified hyper-heuristic framework for solving bin packing problems. Exp. Syst. Appl. **41**(15), 6876–6889 (2014)
61. M. Maashi, G. Kendall, E. Özcan, Choice function based hyper-heuristics for multi-objective optimization. Appl. Soft Comput. **28**, 312–326 (2015)
62. J.G. Marin-Blazquez, S. Schulenburg, A hyper-heuristic framework with XCS: learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients, in *Learning Classifier Systems*. Lecture Notes in Computer Science, vol. 4399 (2007), pp. 193–218
63. J. Maturana, F. Lardeux, F. Saubion, Autonomous operator management for evolutionary algorithms. J. Heuristics **16**(6), 881–909 (2010)
64. J. Mockus, L. Mockus, Bayesian approach to global optimization and applications to multi-objective constrained problems. J. Optim. Theory Appl. **70**(1), 155–171 (1991)
65. A. Nareyek, Choosing search heuristics by non-stationary reinforcement learning, in *Metaheuristics: Computer Decision-Making*, ed. by M.G.C. Resende, J.P. de Sousa, Chap. 9 (Kluwer, Dordecht, 2003), pp. 523–544
66. G. Ochoa, M. Hyde, The cross-domain heuristic search challenge (CHeSC 2011) (2011). http://www.asap.cs.nott.ac.uk/chesc2011/
67. G. Ochoa, R. Qu, E.K. Burke, Analyzing the landscape of a graph based hyper-heuristic for timetabling problems, in *Genetic and Evolutionary Computation Conference, GECCO* (2009), pp. 341–348
68. G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, A.J. Parkes, S. Petrovic, E.K. Burke, Hyflex: a benchmark framework for cross-domain heuristic search, in *European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP*. Lecture Notes in Computer Science, vol. 7245 (2012), pp. 136–147

69. G. Ochoa, J. Walker, M. Hyde, T. Curtois, Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework, in *Parallel Problem Solving from Nature, PPSN XI* (2012), pp. 418–427
70. M. Oltean, Evolving evolutionary algorithms using linear genetic programming. Evol. Comput. J. **13**(3), 387–410 (2005)
71. E. Özcan, B. Bilgin, E.E. Korkmaz, Hill climbers and mutational heuristics in hyperheuristics, in *Parallel Problem Solving from Nature, PPSN*. Lecture Notes in Computer Science, vol. 4193 (2006), pp. 202–211
72. E. Özcan, B. Bilgin, E.E. Korkmaz, A comprehensive analysis of hyper-heuristics. Intell. Data Anal. **12**(1), 3–23 (2008)
73. E. Özcan, A.J. Parkes, Policy matrix evolution for generation of heuristics, in *Genetic and Evolutionary Computation, GECCO* (2011), pp. 2011–2018
74. N. Pillay, A review of hyper-heuristics for educational timetabling. Ann. Oper. Res. **239**(1), 3–38 (2016)
75. D. Pisinger, S. Ropke, A general heuristic for vehicle routing problems. Comput. Oper. Res. **34**(8), 2403–2435 (2007)
76. R. Qu, E.K. Burke, Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. J. Oper. Res. Soc. **60**, 1273–1285 (2009)
77. S. Remde, P. Cowling, K. Dahal, N. Colledge, E. Selensky, An empirical study of hyperheuristics for managing very large sets of low level heuristics. J. Oper. Res. Soc. **63**(3), 392–345 (2012)
78. S. Ropke, D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transp. Sci. **40**(4), 455–472 (2006)
79. P. Ross, Hyper-heuristics, in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ed. by E.K. Burke, G. Kendall, Chap. 17 (Springer, Berlin, 2005), pp. 529–556
80. P. Ross, J.G. Marín-Blázquez, Constructive hyper-heuristics in class timetabling, in *IEEE Congress on Evolutionary Computation, CEC* (2005), pp. 1493–1500
81. P. Ross, S. Schulenburg, J.G. Marin-Blazquez, E. Hart, Hyper-heuristics: learning to combine simple heuristics in bin-packing problem, in *Genetic and Evolutionary Computation Conference, GECCO* (2002), pp. 942–948
82. P. Ross, J.G. Marin-Blazquez, E. Hart, Hyper-heuristics applied to class and exam timetabling problems, in *IEEE Congress on Evolutionary Computation, CEC* (2004), pp. 1691–1698
83. N.R. Sabar, G. Kendall, Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. Inf. Sci. **314**, 225–239 (2015)
84. N.R. Sabar, M. Ayob, R. Qu, G. Kendall, A graph coloring constructive hyper-heuristic for examination timetabling problems. Appl. Intell. **37**(1), 1–11 (2012)
85. N.R. Sabar, M. Ayob, G. Kendall, R. Qu, Grammatical evolution hyper-heuristic for combinatorial optimization problems. IEEE Trans. Evol. Comput. **17**(6), 840–861 (2013)
86. N.R. Sabar, M. Ayob, G. Kendall, R. Qu, Automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems. IEEE Trans. Evol. Comput. **19**(3), 309–325 (2015)
87. N.R. Sabar, M. Ayob, G. Kendall, R. Qu, A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. IEEE Trans. Cybern. **45**(2), 217–228 (2015)
88. K. Sim, E. Hart, A combined generative and selective hyper-heuristic for the vehicle routing problem, in *Genetic and Evolutionary Computation Conference, GECCO* (2016), pp. 1093–1100
89. K. Sim, E. Hart, B. Paechter, A lifelong learning hyper-heuristic method for bin packing. Evol. Comput. J. **23**(1), 37–67 (2015)
90. J.A. Soria-Alcaraz, G. Ochoa, J. Swan, M. Carpio, H. Puga, E.K. Burke, Effective learning hyper-heuristics for the course timetabling problem. Eur. J. Oper. Res. **238**(1), 7–86 (2014)

91. J.A. Soria-Alcaraza, E. Özcan, J. Swan, G. Kendall, M. Carpio, Iterated local search using an add and delete hyper-heuristic for university course timetabling. Appl. Soft Comput. **40**, 581–593 (2016)
92. J.A. Soria-Alcaraz, G. Ochoa, M.A. Sotelo-Figeroa, E.K. Burke, A methodology for determining an effective subset of heuristics in selection hyper-heuristics. Eur. J. Oper. Res. **260**(3), 972–983 (2017)
93. A. Sosa-Ascencio, G. Ochoa, H. Terashima-Marin, S.E. Conant-Pablos, Grammar-based generation of variable-selection heuristics for constraint satisfaction problems. Genet. Program Evolvable Mach. **17**(2), 119–144 (2016)
94. E. Soubeiga, Development and application of hyperheuristics to personnel scheduling. Ph.D. Thesis, School of Computer Science and Information Technology, University of Nottingham, 2003
95. R.H. Storer, S.D. Wu, R. Vaccari, Problem and heuristic space search strategies for job shop scheduling. ORSA J. Comput. **7**(4), 453–467 (1995)
96. J. Swan, J.R. Woodward, E. Özcan, G. Kendall, E.K. Burke, Searching the hyper-heuristic design space. Cogn. Comput. **6**(1), 66–73 (2014)
97. J.C. Tay, N.B. Ho, Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Comput. Ind. Eng. **54**(3), 453–473 (2008)
98. H. Terashima-Marin, P. Ross, M. Valenzuela-Rendon, Evolution of constraint satisfaction strategies in examination timetabling, in *Genetic and Evolutionary Computation Conference, GECCO* (1999), pp. 635–642
99. H. Terashima-Marin, E.J. Flores-Alvarez, P. Ross, Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems, in *Genetic and Evolutionary Computation Conference, GECCO* (2005), pp. 637–643
100. H. Terashima-Marin, A. Moran-Saavedra, P. Ross, Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems, in *IEEE Congress on Evolutionary Computation, CEC*, vol. 2 (2005), pp. 1104–1110
101. J.A. Vazquez-Rodriguez, S. Petrovic, A. Salhi, A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines, in *Multidisciplinary International Scheduling Conference: Theory and Applications, MISTA* (2007), pp. 506–513
102. J.A. Vrugt, B.A. Robinson, Improved evolutionary optimization from genetically adaptive multimethod search. Proc. Natl. Acad. Sci. **104**(3), 708–711 (2007)
103. X. Wu, P.A. Consoli, L.L. Minku, G. Ochoa, X. Yao, An evolutionary hyper-heuristic for the software project scheduling problem, in *Parallel Problem Solving from Nature, PPSN XIV* (2016), pp. 37–47
104. K.Z. Zamil, F. Din, G. Kendall, B.S. Ahmed, An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial t-way test suite generation. Inf. Sci. **399**, 121–153 (2017)