A Classification of Hyper-heuristic Approaches

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

Abstract The current state of the art in hyper-heuristic research comprises a set of approaches that share the common goal of automating the design and adaptation of heuristic methods to solve hard computational search problems. The main goal is to produce more generally applicable search methodologies. In this chapter we present an overview of previous categorisations of hyper-heuristics and provide a unified classification and definition, which capture the work that is being undertaken in this field. We distinguish between two main hyper-heuristic categories: heuristic *selection* and heuristic *generation*. Some representative examples of each category are discussed in detail. Our goals are to clarify the main features of existing techniques and to suggest new directions for hyper-heuristic research.

1 Introduction

The current state of the art in hyper-heuristic research comprises a set of approaches that share the common goal of automating the design and adaptation of heuristic methods in order to solve hard computational search problems. The motivation behind these approaches is to raise the level of generality at which search methodologies can operate [6]. The term hyper-heuristic was first used in 1997 [21] to describe a protocol that combines several artificial intelligence methods in the context of automated theorem proving. The term was independently used in 2000 [18] 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 [6, 52]. The idea of automating the heuris-

Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan and John R. Woodward
Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, UK. e-mail: {ekb,mvh,gxk,gxo,exo,jrw}@cs.nott.ac.uk

tic design process, however, is not new. Indeed, it can be traced back to the early 1960s [20, 26, 27], and was independently developed by a number of authors during the 1990s [24, 32, 33, 43, 58, 62]. Some historical notes, and a brief overview of early approaches can be found in [6] and [52], respectively. A more recent research trend in hyper-heuristics attempts to automatically *generate* new heuristics 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 [7].

A variety of hyper-heuristic approaches using high-level methodologies, together with a set of low-level heuristics, and applied to different combinatorial problems, have been proposed in the literature. The aim of this chapter is to provide an updated version to the hyper-heuristic chapter [6] in the 2003 edition of the Handbook of Metaheuristics. We present an overview of previous categorisations of hyper-heuristics and provide a unified classification and definition which captures all the work that is being undertaken in this field. Our goals are to clarify the main features of existing techniques and to suggest new directions for hyper-heuristic research.

The next section outlines previous classifications of hyper-heuristics. Section 3 proposes both a unified classification and a new definition of the term. Sections 4 and 5, describe the main categories of the proposed classification, giving references to work in the literature and discussing some representative examples. Finally, section 6 summarises our categorisation and suggests future research directions in the area.

2 Previous classifications

In [57], hyper-heuristics are categorised into two types: (i) with learning, and (ii) 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 [42]. 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 [57], 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 many hyper-heuristics to date 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 [2], hyper-heuristics are classified into those which are *constructive* and those which are *local search* methods. This distinction is also mentioned by Ross [52]. 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 2000's (see [7] for an overview), a new class of hyper-heuristics emerged. This class was explicitly and independently mentioned in [1] and [10]. 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* or *components* 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. But, instead of supplying these directly to the framework, the heuristics are first decomposed into their basic components. Genetic programming hyper-heuristic researchers [1, 7, 10] 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 [16], hyper-heuristics are classified into four categories: (i) hyper-heuristics based on the random choice of low-level heuristics, (ii) greedy and peckish hyper-heuristics, which requires preliminary evaluation of all or a subset of the heuristics in order to select the best performing one, (iii) metaheuristics based hyper-heuristics, and (iv) hyper-heuristics employing learning mechanisms to manage low level heuristics.

3 The proposed classification and new 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 dimensions: (i) the nature of the heuristic search space, and (ii) the source of feedback during learning. These dimensions are orthogonal in that different heuristic search spaces can be combined with different sources of feedback, and thus different machine learning techniques.

We consider that 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 components of existing heuristics

There is no reason why the higher level strategy (for selecting or generating heuristics) should be restricted to be a heuristic. Indeed, sophisticated knowledge-based techniques such as case-based reasoning has been employed in this way with good results for university timetabling [15]. This leads us to propose the following

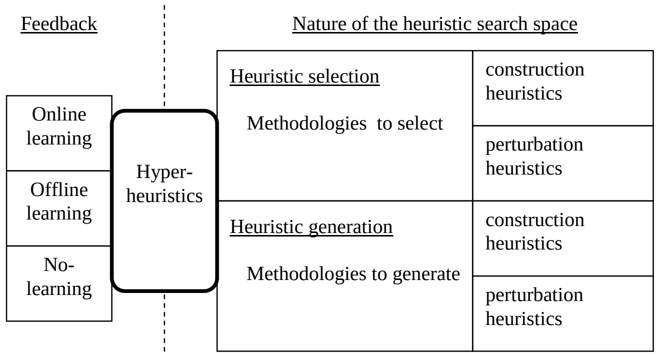

Fig. 1 A classification of hyper-heuristic approaches, according to two dimensions (i) the nature of the heuristic search space, and (ii) the source of feedback during learning.

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 hard computational search problems.

From this definition, there are two clear categories of hyper-heuristics: heuristic selection and heuristic generation, which form the first level in our first dimension (the nature of the search space). The second level in this dimension corresponds to the distinction between constructive and local search hyper-heuristics, also discussed in section 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 4 and 5 describe these categories in more detail, discussing some concrete examples of recent approaches that can be found in the literature.

We consider a hyper-heuristic to be a learning algorithm when it uses some feedback from the search process. Therefore, non-learning hyper-heuristics are those that do not use any feedback. According to the source of the feedback during learning, 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 section 2), the notions of *disposable* and *reusable* have been used to refer to analogous ideas 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. Examples of on-line learning approaches within hyper-heuristics are: the use of reinforcement learning for heuristic selection and, generally, the use of metaheuristics as high-level search strategies across a search space of heuristics.

Offline learning hyper-heuristics: The idea is to gather knowledge in the form of rules or programs, from a set of training instances, that would hopefully generalise to the process of solving unseen instances. Examples of offline learning approaches within hyper-heuristics are: learning classifier systems, case-based reasoning and genetic programming.

The proposed classification of hyper-heuristic approaches can be summarised as follows (also see figure 1):

- With respect to the nature of the heuristic search space
  - **Heuristic selection methodologies**: Produce combinations of pre-existing:
    - Construction heuristics
    - Perturbation heuristics
  - **Heuristic generation methodologies**: Generate new heuristic methods using basic components (building-blocks) of:
    - Construction heuristics
    - 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 feedback from the search process.

Note that these categories describe current research trends. There is, however, nothing to stop the exploration of hybrid methodologies that combine for example construction with perturbation heuristics, or heuristic selection with heuristic generation methodologies. These hybrid approaches are already starting to emerge.

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.

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 pre-existing (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 recently been proposed to generate efficient hybridisations of existing construction heuristics in domains such as bin packing [41, 55], timetabling [13, 15, 53, 54], production scheduling [63], and stock cutting [60, 61]. 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. For example, genetic algorithms [25, 33, 62, 63], tabu search [13] and other single-point based search strategies [51]. For this type of hyper-heuristic, recent research is starting to explore the structure of the heuristic search space or hyper-heuristic landscape, in both timetabling [45] and production scheduling [46]. Examples of offline techniques are the use of learning classifier systems [41, 55], messy genetic algorithms [53, 54, 61] and case-based reasoning [15].

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 [13], 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 others 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 i^{th} iteration, the i^{th} graph-colouring heuristic in the list is used to order the events not yet scheduled at that step, and the first e events in the ordered list are assigned to the first e least-cost timeslots in the timetable (see figure 2).

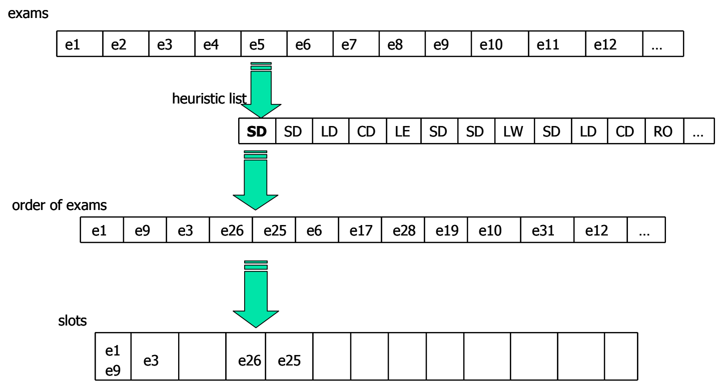
exams
e1 e2 e3 e4 e5 e6 e7 e8 e9 e10 e11 e12 ...
heuristic list
SD SD LD CD LE SD SD LW SD LD CD RO ...
order of exams
e1 e9 e3 e26 e25 e6 e17 e28 e19 e10 e31 e12 ...
slots
e1 e9 e3 e26 e25

Fig. 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.

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. Each heuristic in the list is used to schedule a number e of events. Therefore, the length of the heuristic list is n/e where n is the number of events to be scheduled. Values in the range of $e = 1,\ldots,5$ were tested (details can be found in [13]). This work also highlights the existence of two search spaces in constructive hyper-heuristics (the heuristic space and the problem solution). The approach was tested on both course and exam timetabling benchmark instances with competitive results. This graph-based hyper-heuristic was later extended in [51] 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 steepest descent method, iterated local search and variable neighbourhood search are implemented and compared to the previously implemented tabu search. The results suggests 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 can produce 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 [45], 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 [51]. 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 comprising 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 [34] 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 subcomponent or partial solution, in decomposable problems being solved collectively by a set of partial solutions.

In [55], a modern classifier system (accuracy-based classifier system [64]) 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 put in the first bin where they fit (a new bin is opened if necessary, and effectively all bins stay open).
- *Next-Fit-Decreasing*: An item is placed in the current bin if possible, or else a new bin is opened into which the piece 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 [41], the classifier system approach is extended to multi-step environments. The authors test 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.

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 has been 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 [12, 18], timetabling [5, 12], shelf space allocation [3, 4], packing [23] and vehicle routing problems [50].

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 [17].

As suggested in [48, 49] perturbation hyper-heuristics can be separated into two processes: (i) (low-level) heuristic selection, and (ii) move acceptance strategy. 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 [18, 19]. 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 [12, 44, 50], or some type of performance statistics [18, 19]. 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) used for 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 commonly used as non-deterministic acceptance methods, such as those of great deluge [38] and simulated annealing [4, 23, 50].

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 [23] the tabu search hyper-heuristic, originally presented in [12], is integrated within a simulated annealing framework. That is, a simulated annealing acceptance strategy is combined with the previously proposed heuristic selection mechanism. Figure 3 outlines the simulated annealing-based hyper-heuristic.

The tabu search hyper-heuristic [12], 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 [23]. 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.

A general heuristic for vehicle routing problems: In [50], 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 (ALNS), which extends a previous framework presented in [56]. ALNS can be based on any local search framework, e.g. simulated annealing, tabu search or guided local search. The general framework is outlined in Fig.4, where the repeat loop corresponds to the local search framework at the master level. Implementing a simulated annealing algorithm is straightforward as one solution is sampled in each iteration of the ALNS. 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 a neighbourhood N_i has contributed to

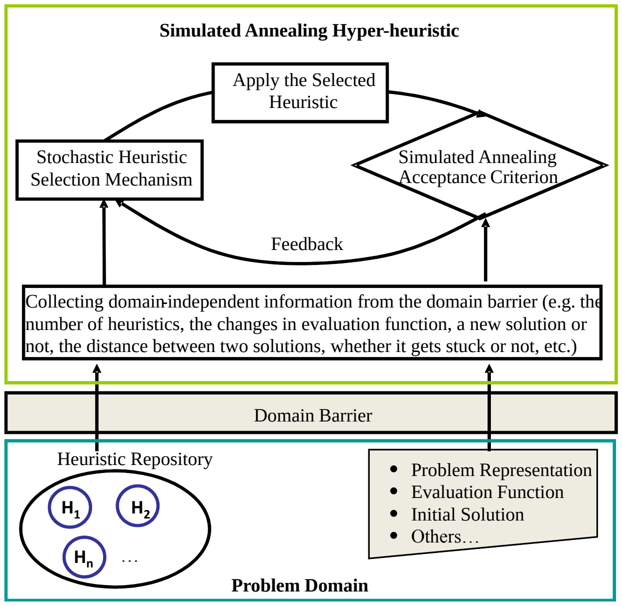
Simulated Annealing Hyper-heuristic
Apply the Selected Heuristic
Stochastic Heuristic Selection Mechanism
Simulated Annealing Acceptance Criterion
Feedback
Collecting domain-independent information from the domain barrier (e.g. the number of heuristics, the changes in evaluation function, a new solution 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. 3 A simulated annealing hyper-heuristic framework.

the solution process, the larger score P_i it obtains, and hence it has a larger 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. The proposed framework adaptively chooses among a number of insertion and removal heuristics to intensify and diversify the search. These competing sub-heuristics are selected with a frequency corresponding to their historic performance (stored as learned weights for each heuristic). The approach uses a simulated annealing acceptance strategy with a standard exponential cooling rate. 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 criteria is met
return x*
```

Fig. 4 Outline of the Adaptive Large Neighbourhood framework. $N-$ and $N+$ correspond to destroy and repair neighbourhoods respectively, whilst P_i stands for the score associated to the heuristic i.

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 [7], a branch of evolutionary computation concerned with the automatic generation of computer programs [40]. Besides the particular representation (using trees as chromosomes[1]), it 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. Genetic programming has been successfully applied to the automated generation of heuristics that solve hard combinatorial optimisation problems, such as boolean satisfiability, [1, 28, 29, 30, 39], bin packing [9, 8, 11], traveling salesman problem [36, 37] and production scheduling [22, 31, 59].

Some genetic programming-based hyper-heuristics have evolved local search heuristics [1, 29, 30, 37, 36] or even evolutionary algorithms [47]. 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 [9, 8, 11, 22, 31, 59]. 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 [1, 36, 37]. 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 section 4, except that a new heuristic is generated for each instance, instead of choosing a sequence of heuristics from a predefined set.

[1] According to the genetic programming literature, programs can be represented in ways other than trees. Research has already established the efficacy of both linear and graph based genetic programming systems.

The adaptation of heuristic orderings can also be considered as a methodology for heuristic generation. The adaptive approach proposed in [14], 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.

5.1 Representative examples

We discuss two representative examples of heuristic generation using genetic programming. The first evolves packing heuristics that operate on a constructive framework, whilst 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: As mentioned earlier, the one-dimensional bin packing problem involves a set of integer pieces L, 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 pieces 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 pieces one at a time, in the order that they are presented. The heuristic iterates through the open bins, and the current piece is placed in the first bin into which it fits.

In [9, 8, 11], construction heuristics are evolved for the online bin packing problem. The evolved heuristics, represented as trees (see Fig. 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 pieces and bin capacities, that represent the state of the problem, in order to evolve functions (expressions) that would direct the process of packing. Each evolved function (GP tree) is applied in turn to the available bins, returning a value. If the value is zero or less then the system moves on to the next bin, but if the value is positive the piece is packed into the bin. In this way, it is the expression which decides when to stop the search for a suitable bin and place the piece. The algorithm (depicted in Fig. 6) then repeats the process for each of the other pieces until all the pieces have been packed.

In a genetic programming framework, the set of terminals and functions need to be specified. The hyper-heuristic framework for online bin packing uses some attributes that describe the state of the problem to define the terminals. In [9, 8], the authors use the following terminals:

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

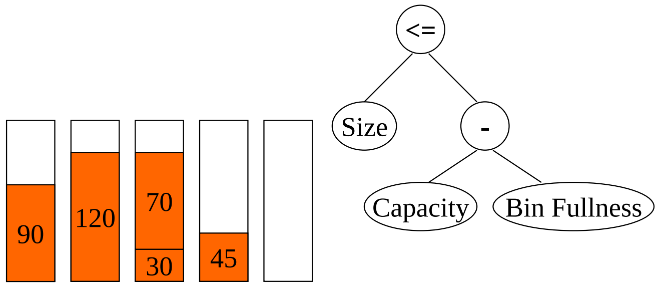

Fig. 5 Evolving one-dimensional packing heuristics with genetic programming.

Later [11], these three attributes were replaced by two: S,the size of the current item and, E $(= C - F)$, the emptiness of a bin (i.e. how much space is remaining in the bin, or how much more space can be allocated to it before it exceeds its capacity). The function set used in [9, 8] consisted of $\leq, +, -, \times, \%$, where $\%$ is the 'protected divide function'[40]. The results in [8] show that a simple genetic programming system can discover human designed heuristics such as first-fit, whilst in [9, 11], heuristics that outperformed first-fit were evolved. In [9], it was also shown empirically that the choice of the training instances (categorised according to the piece size distribution), impacts on the trade-off between the performance and generality of the heuristics generated and their applicability to new problems.

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

Fig. 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 [28, 29, 30] a genetic programming system, named CLASS (Composite Learned Algorithms for SAT Search), is proposed which automatically discovers new SAT local search heuristics. Figure 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, local search heuristics have improved dramatically since the original method. The CLASS system was developed in order to automatically discover variable selection heuristics for SAT local search. It was noted in [28] that many of the best-known SAT heuristics (such as GSAT, HSAT, Walksat, and Novelty [30]) 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 [30], 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
For j:= 1 to termination condition
  If A satisfies formula then return A
    v:= Choose variable using
         "variable selection heuristic"
    A:= A with value of v inverted
  End If
End For
return FAILURE (no assignment found)
```

Fig. 7 A generic SAT local search algorithm. The “variable selection heuristic” is replaced by the evolved function.

6 Summary and discussion

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 [40], and other methods such as *squeaky wheel* optimisation [35], into hyper-heuristic research, 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 in the 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. M. Bader-El-Den and R. Poli. Generating SAT local-search heuristics using a GP hyper-heuristic framework. In *Evolution Artificielle, 8th International Conference*, volume 4926 of *Lecture Notes in Computer Science*, pages 37–49, Tours, France, 2007. Springer, Berlin.
2. R. Bai. *An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, September 2005.
3. R. Bai, E. K. Burke, and G. Kendall. Heuristic,meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society*, 59:1387 – 1397, 2008.
4. R. Bai and G. Kendall. An investigation of automated planograms using a simulated annealing based hyper-heuristics. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solver - (Operations Research/Computer Science Interface Serices, Vol.32)*, pages 87–108. Springer, Berlin, 2005.
5. B. Bilgin, E. Özcan, and E. E. Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of the 6th Practice and Theory of Automated Timetabling (PATAT 2006)*, volume 3867 of *Lecture Notes in Computer Science*, pages 394–412, Brno, Czech Republic, 2007. Springer, Berlin.

6. E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.
7. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In C. Mumford and L. Jain, editors, *Collaborative Computational Intelligence*, pages 177–201. Springer, Berlin, 2009.
8. E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, volume 4193 of *Lecture Notes in Computer Science*, pages 860–869, Reykjavik, Iceland, September 2006. Springer, Berlin.
9. E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Proceedings of the 9th ACM Genetic and Evolutionary Computation Conference (GECCO'07)*, pages 1559–1565, London, UK., July 2007. ACM, NY, USA.
10. E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. *IEEE Transactions on Evolutionary Computation (accepted, to appear)*, 2010.
11. E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward. The scalability of evolved on line bin packing heuristics. In *2007 IEEE Congress on Evolutionary Computation*, pages 2530–2537, Singapore, 2007. IEEE Computational Intelligence Society, IEEE Press.
12. E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.
13. E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176:177–192, 2007.
14. E. K. Burke and J. Newall. Solving examination timetabling problems through adaptation of heuristic orderings. *Annals of operations Research*, 129:107–134, 2004.
15. E. K. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.
16. K. Chakhlevitch and P. I. Cowling. Hyperheuristics: Recent developments. In Carlos Cotta, Marc Sevaux, and Kenneth Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer, Berlin, 2008.
17. P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Congress on Evolutionary Computation (CEC2002)*, pages 1185–1190, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, USA, May 12-17 2002.
18. P. Cowling, G. Kendall, and 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, pages 176–190, Konstanz, Germany, August 2000. Springer, Berlin.
19. P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In S. Cagoni, J. Gottlieb, Emma Hart, M. Middendorf, and R. Goenther, editors, *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10, Kinsale, Ireland, April 3-4 2002. Springer-Verlag, Berlin.
20. W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR research memorandum, Carnegie-Mellon University,Pittsburgh, 1963.
21. J. Denzinger, Matthias Fuchs, and Marc Fuchs. High performance ATP systems by combining several AI methods. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 102–107, CA, USA, 1997. Morgan Kaufmann.
22. C. Dimopoulos and A. M. S. Zalzala. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, 32(6):489–498, 2001.

23. K. A. Dowsland, E. Soubeiga, and E. K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.
24. H. L. Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pages 375–382, San Mateo, 1993. Morgan Kaufmann, CA, USA.
25. H. L. Fang, P. Ross, and D. Corne. A promising hybrid ga/ heuristic approach for open-shop scheduling problems. In A. Cohn, editor, *Eleventh European Conference on Artificial Intelligence*. John Wiley & Sons, NJ, USA, 1994.
26. H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institue of Technology, May 10-12 1961.
27. H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251, New Jersey, 1963. Prentice-Hall, Inc.
28. A. S. Fukunaga. Automated discovery of composite SAT variable selection heuristics. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 641–648, Edmonton, Canada, 2002.
29. A. S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, Lecture Notes in Computer Science, pages 483–494. Springer, Berlin, 2004.
30. A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evol. Comput.*, 16(1):31–61, 2008.
31. C. D. Geiger, R. Uzsoy, and H.Aytŭg. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9:7–34, 2006.
32. J. Gratch and S. Chien. Adaptive problem-solving for large-scale scheduling problems: a case study. *Journal of Artificial Intelligence Research*, 4:365–396, 1996.
33. E. Hart, P. Ross, and J. A. D. Nelson. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computing*, 6(1):61–80, 1998.
34. John H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In *Pattern-directed inference systems*. Academic Press, New York, 1978.
35. D. Joslin and D. P. Clements. "squeaky wheel" optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.
36. R. E. Keller and R. Poli. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Proceedings of Artificial Evolution(EA'07)*, pages 13–24, Tours, France, 2007.
37. R. E. Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In *Proceedings of IEEE Congress on Evolutionary Computation (CEC 2007)*, Singapore, 2007.
38. G. Kendall and M. Mohamad. Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proceedings of the 2004 IEEE International Conference on Network (ICON2004)*, pages 769–773, Singapore, 16-19 November 2004.
39. R. H. Kibria and Y. Li. Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 331–340, Budapest, Hungary, 2006. Springer, Berlin.
40. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, MA, USA, 1992.
41. J. G. Marin-Blazquez and S. Schulenburg. A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *Learning Classifier Systems*, volume 4399 of *Lecture Notes in Computer Science*, pages 193–218, Berlin Heidelberg, 2007. Springer.
42. N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.
43. J. Mockus and L. Mockus. Bayesian approach to global optimization and applications to multi-objective constrained problems. *Journal of Optimization Theory and Applications*, 70(1):155–171, July 1991.

44. A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende and J. P. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, chapter 9, pages 523–544. Kluwer, 2003.
45. G. Ochoa, R. Qu, and E. K. Burke. Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO 2009)*, pages 341–348, Montreal, Canada, 2009.
46. G. Ochoa, J. A. Váquez-Rodríguez, S. Petrovic, and E. K. Burke. Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 1873–1880, Trondheim, Norway, 2009.
47. M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005.
48. E. Özcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in hyper-heuristics. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211, Reykjavik, Iceland, September 2006. Springer, Berlin.
49. E. Özcan, B. Bilgin, and E. E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.
50. D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403– 2435, 2007.
51. R. Qu and E. K. Burke. Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *Journal of the Operational Research Society*, 60:1273–1285, 2009.
52. P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. Springer, Berlin, 2005.
53. P. Ross and J. G. Marín-Blázquez. Constructive hyper-heuristics in class timetabling. In *IEEE Congress on Evolutionary Computation*, pages 1493–1500, Edinburgh, UK, 2005. IEEE, NJ, USA.
54. P. Ross, J. G. Marin-Blazquez, and E. Hart. Hyper-heuristics applied to class and exam timetabling problems. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1691–1698, Portland, Oregon, 2004. IEEE Press.
55. P. Ross, S. Schulenburg, J. G. Marin-Blazquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In *Proceedings of the Genetic and Evolutionary Computation COnference, GECCO'02*, New York, USA, 2002. Morgan-Kauffman, CA, USA.
56. P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proc. of International Conference on Principles and Practice of Constraint Programming (CP'98)*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, Berlin, 1998.
57. E. Soubeiga. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, June 2003.
58. R. H. Storer, S. D. Wu, and R. Vaccari. Problem and heuristic space search strategies for job shop scheduling. *ORSA Journal of Computing*, 7(4):453–467, 1995.
59. J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54:453–473, 2008.
60. H. Terashima-Marin, E. J. Flores-Alvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems. In Hans-Georg Beyer and Una-May O'Reilly, editors, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2005*, pages 637–643, Washington DC, USA, 2005. ACM, NY, USA.
61. H. Terashima-Marin, A. Moran-Saavedra, and P. Ross. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1104–1110, Edinburgh, Scotland, UK, 2005. IEEE Press.

62. H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendon. Evolution of constraint satisfaction strategies in examination timetabling. In *Genetic and Evolutionary Computation COnference, GECCO'99*, pages 635–642, 1999.
63. J. A. Vazquez-Rodriguez, S. Petrovic, and 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 Philippe Baptiste, Graham Kendall, Alix Munier, and Francis Sourd, editors, *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007)*, 2007.
64. S.W. Wilson. Classifier systems based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.