School of Computer Science and Information Technology
University of Nottingham
Jubilee Campus
NOTTINGHAM NG8 1BB, UK

Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747

A Survey of Hyper-heuristics

Edmund K. Burke, Matthew Hyde, Graham Kendall
Gabriela Ochoa, Ender Ozcan and Rong Qu

First released: March 2009

Noname manuscript No.
(will be inserted by the editor)

A Survey of Hyper-heuristics

Edmund K. Burke · Matthew Hyde · Graham Kendall · Gabriela Ochoa · Ender Ozcan · Rong Qu

Received: date / Accepted: date

Abstract Hyper-heuristics comprise a set of approaches with the common goal of automating the design and tuning of heuristic methods to solve hard computational search problems. The main goal is to produce more generally applicable search methodologies. The term hyper-heuristic was coined in the early 2000's to refer to the idea of 'heuristics to choose heuristics'. However, the idea of automating the design of combined heuristics can be traced back to the early 1960's. With the incorporation of Genetic Programming into hyper-heuristic research, a new type of hyper-heuristics has emerged that we have termed 'heuristics to generate heuristics'. The distinguishing feature of hyper-heuristics is that they operate on a search space of heuristics (or heuristic components) rather than directly on the search space of solutions to the underlying problem, as is the case with most meta-heuristic approaches. This paper presents a literature survey of hyper-heuristics including their origin and intellectual roots, a detailed account of the main types of approaches, and an overview of some related areas. Current research trends and directions for future research are also discussed.

Keywords hyper-heuristics · combinatorial optimisation · search methodologies

1 Introduction

Despite the success of heuristic search methods in solving real-world computational search problems, there are still some difficulties for easily applying them to newly encountered problems, or even new instances of known problems. These difficulties arise mainly from the significant range of parameter or algorithm choices involved when using this type of approaches, and the lack of guidance as to how to proceed for select them. Another drawback of these techniques is that state-of-the-art approaches for real-world problems tend to represent bespoke problem-specific methods which are expensive to develop and maintain. Hyper-heuristics encompass a set of approaches with the goal

F. Author
first address
Tel.: +123-45-678910
Fax: +123-45-678910
E-mail: fauthor@example.com

of automating the design and tuning of heuristic methods to solve hard computational search problems (Burke et al, 2003a; Ross, 2005; Burke et al, 2009b). The main motivation behind hyper-heuristics is to develop algorithms that are more generally applicable than many of the current implementations of search algorithms. The main feature of hyper-heuristics is that they operate on a search space of heuristics (or components of such heuristics) rather than directly on the search space of solutions to a given problem. In this sense they differ from most applications of metaheuristics. In consequence, when using hyper-heuristics, we are attempting to find the right method or sequence of heuristics in a given situation rather than trying to solve the problem directly. Hyper-heuristics could be regarded as "off-the-peg" methods as opposed to "made-to-measure" meta-heuristics. Therefore, the goal is to produce generic methods, which should produce solutions of acceptable quality, based on a set of easy-to-implement low-level heuristics. A hyper-heuristic can be seen as a (high-level) methodology which, when given a particular problem instance or class of instances, and a number of low-level heuristics (or its components), automatically produces an adequate combination of the provided components to effectively solve the given problem(s).

We can identify two fundamental ideas behind the notion of hyper-heuristics. Firstly, the recognition that the process of selecting or designing efficient hybrid and/or cooperative heuristics can be regarded as a computational search problem in itself. Secondly, there is significant potential to improve search methodologies by the incorporation of learning mechanisms that can adaptively guide the search. These two fundamental ideas have inspired different types of hyper-heuristics (Burke et al, 2009a). The term hyper-heuristics is relatively new, it was introduced in the early 2000s (Cowling et al, 2000) to describe the idea of 'heuristics to choose heuristics' in the context of combinatorial optimisation . The first journal paper to use the term was (Burke et al, 2003b), whilst the first widely available refereed conference paper was (Cowling et al, 2000). The idea, however, can be traced back to the early 1960s (Fisher and Thompson, 1961, 1963), as we will discuss further in section 2. A more recent research trend in hyper-heuristics attempts to automatically *generate* new heuristics suited to a given problem or class of problems, by combining, mainly through the use of genetic programming, *components* or *building-blocks* of human designed heuristics (Burke et al, 2009b).

The reminder of this article is presented as follows. The next section discusses the intellectual roots and early hyper-heuristic approaches. We consider as early approaches, those that appeared before early 2000, when the term hyper-heuristic was first used. Section 3 overviews the existing related introductory and review chapters, presents a classification of hyper-heuristic approaches, and surveys the literature of the different types of hyper-heuristics. Section 4 briefly overviews some related approaches that also seek to automate the design and tuning of search algorithms. Finally, section 5 highlights the main research trends in hyper-heuristics and suggests some potentially interesting future research directions.

2 Origins and early approaches

The ideas behind of hyper-heuristics are not new. They can be traced back to the early 1960s when Fisher and Thompson (1961, 1963); Crowston et al (1963) hypothesised that combining scheduling rules (also known as priority or dispatching rules) in production scheduling would be superior than any of the rules taken separately. This pioneering work, well ahead its time, proposed a method of combining scheduling rules using 'probabilistic learning'. Notice that in the early 1960s, meta-heuristic and local search techniques were still not mature. However, the proposed learning approach resembles a stochastic local search algorithm operating in the space of scheduling rules' sequences. The main conclusions from this study are the following: "(1) an unbiased random combination of scheduling rules is better than any of them taken separately; (2) learning is possible" (Fisher and Thompson, 1963). The ideas by Fisher and Thompson were independently rediscovered and enhanced several times, 30 years later (Dorndorf and Pesch, 1995; Fang et al, 1993; Storer et al, 1992, 1995), when modern meta-heuristics were already widely known and used. Therefore, the authors had the appropriate conceptual and algorithmic tools (also the computer power) to propose and conduct search within a heuristic search space.

Storer et al (1992, 1995) clearly stated the problem of designing a good combination of problem-specific (fast) heuristics as a search problem, and defined neighbourhoods within the heuristic space. The authors also proposed neighbourhoods on the problem data. Therefore, by perturbing the heuristic combination, the problem data, or both, a subset of solutions in these new search spaces were produced. These neighborhoods formed the basis for local search. The authors stressed that, due to their generic nature, the basic principle can be applied to develop heuristics for a broad spectrum of problems. Since the emphasis is placed on developing search spaces, the authors used a simple variant of hill-climbing. They stressed, however, that various search strategies such as simulated annealing, genetic algorithms or tabu search could be applied. The ideas were successfully tested by developing heuristics for job-shop scheduling. The resulting heuristics produced encouraging solutions when compared to the state-of-art shifting bottleneck procedure.

Fang et al (1993, 1994), used a genetic algorithm that searched the space of sequences of heuristic choices in the context of open-shop scheduling. A heuristic choice was represented as a pair (j, h) indicating an uncompleted job j, and a heuristic h (from a set of scheduling rules) to select a task from that job (j) to insert next into the schedule. The sequence of pairs is interpreted from left to right to construct a complete schedule. The approach produced very good results on benchmark problems including some best-known solutions at that time. Hart and Ross (1998) successfully applied a variant of this idea to dynamic job-shop scheduling problems. The approach is based upon evolving combined iterations of previously proposed heuristic algorithms for choosing tasks to schedule. Very good results were obtained on benchmark problems when tested on a variety of criteria. Hart et al (1998) also used a genetic algorithm based approach to solve a real-world scheduling problem. The problem was to schedule the collection and delivery of live chickens from farms all over Scotland and Northern England to one of two processing factories, in order to satisfy a set of orders. The approach combined two genetic algorithms that evolved heuristic choices, one to manage the assignment of orders, and the second to schedule the arrival of deliveries. The proposed strategy successfully met the project's requirements.

Norenkov and Goodman (1997) also used evolutionary algorithms for searching over a space of heuristic sequences. They considered the multistage flow-shop scheduling problem, and included two types of scheduling rules: (i) for determining jobs orderings, and (ii) for determining job assignments to machines. A set of experiments was conducted and the quality of the solutions obtained was found to strongly depend on the subset of heuristics applied.

Notice that the early approaches discussed above were all online. That is, directed to find good sequences of heuristics to solve a given instance of a problem. Also the problem studied was in all cases related to job-shop scheduling. Another root inspiring the concept of hyper-heuristics comes from the Artificial Intelligence (AI) community. Specifically, AI work on automated planning systems, and its eventual focus towards the problem of learning control knowledge. In (Gratch and Chien, 1996; Gratch et al, 1993), the so-called COMPOSER system was used for controlling satellite communication schedules involving a number of earth-orbiting satellites and three ground stations. The system can be characterised as a hill-climbing search in the space of possible control strategies. The learning system alternately proposes changes to the current control strategy and statistically evaluates them to determine whether they enhance the expected utility. The approach is off-line in that a large supply of representative training problems are required in order to have an adequate estimate of the expected utility for various control strategies.

3 Survey and representative research

3.1 Related articles

This is the first comprehensive literature review journal article on hyper-heuristic approaches. However, a number of introductory, tutorial and review book chapters have been published. The first introductory and overview article appeared in 2003 (Burke et al, 2003a), where the authors introduce the idea of hyper-heuristics, and stress their main objective; namely, to raise the level of generality at which optimisation systems can operate. The chapter also gives a brief history of the area and discusses in detail some of the latest work published at the time. A tutorial article is later published by Ross (2005), which not only gives useful guidelines for implementing a hyper-heuristic approach; but also discusses a number of relevant research issues and identifies promising application domains. The chapter by Chakhlevitch and Cowling (2008) provides a review of recent the developments in hyper-heuristics. The authors classify and review hyper-heuristic approaches into the following four categories: based on the random choice of low level heuristics, greedy and peckish, metaheuristic-based, and those employing learning mechanisms to manage low level heuristics. Their chapter does not review the literature on genetic programming approaches, nor includes a detailed account of the origins and early hyper-heuristic approaches. Finally, a very recent overview and tutorial chapter (Burke et al, 2009b) discusses the more recent approach that aims to *generate* new heuristics from a set of potential heuristic components, in which Genetic Programming is the most widely used methodology. The chapter includes a detailed description of the steps needed to apply this approach, some representative case studies, a brief literature review of related work, and a discussion of relevant issues of this class of hyper-heuristic.

3.2 A classification of hyper-heuristic approaches

Burke et al (2009a) 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 (see figure 1). These dimensions are orthogonal in that different

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.

3.3 Heuristics to choose heuristics (based on constructive heuristics)

These approaches build a solution incrementally. Starting with an empty solution, they intelligently select and use constructive heuristics to gradually build a complete solution. The hyper-heuristic framework is provided with a set of pre-existing (generally problem specific) constructive 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) has been reached. Notice that there is a natural ending to the construction process 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 in this scenario, the interesting possibility of learning associations between partial solution stages and adequate heuristics for those stages.

Application Domain	**Reference(s)**
Production Scheduling	(Fisher and Thompson, 1961, 1963) (Storer et al, 1992, 1995) (Dorndorf and Pesch, 1995) (Fang et al, 1993, 1994) (Norenkov and Goodman, 1997) (Hart and Ross, 1998; Hart et al, 1998) (Váquez-Rodríguez et al, 2007a,b; Ochoa et al, 2009b)
Educational Timetabling	(Terashima-Marín et al, 1999) (Ahmadi et al, 2003; Asmuni et al, 2005) (Ross et al, 2004; Ross and Marín-Blázquez, 2005) (Burke et al, 2005a, 2006b) (Burke et al, 2007c; Qu and Burke, 2008; Ochoa et al, 2009a) (Pillay and Banzhaf, 2007; Pillay, 2008)
1D Packing	(Ross et al, 2002, 2003; Ross and Marín-Blázquez, 2005) (Marín-Blázquez and Schulenburg, 2007)
2D Cutting and Packing	(Terashima-Marín et al, 2006, 2007, 2008a) (Garrido and Riff, 2007a,b)
Constraint Satisfaction	(Terashima-Marín et al, 2008b)

Table 1 Application domains of heuristic to choose heuristics based on constructive low-level heuristics.

Several approaches have been recently proposed to generate efficient hybridisations of existing constructive heuristics in domains such as timetabling, cutting and packing, production scheduling, and constraint satisfaction problems (see table 1). Both online and offline approaches, and different high-level strategies have been investigated (see table 2). The following subsections survey the approaches so far according to the type of low-level heuristics employed, which is in turn related to the application domain.

3.3.1 Graph-colouring heuristics in timetabling

There is a well known analogy between a basic version of the timetabling problem and the graph colouring problem. Nodes can represent events and edges represent

High-level strategy	**Reference(s)**
Hill-climbing	(Storer et al, 1992, 1995) (Gratch and Chien, 1996; Gratch et al, 1993)
Genetic Algorithms	(Dorndorf and Pesch, 1995) (Fang et al, 1993, 1994) (Norenkov and Goodman, 1997) (Hart and Ross, 1998; Hart et al, 1998) (Terashima-Marín et al, 1999) (Ahmadi et al, 2003) (Váquez-Rodríguez et al, 2007a,b; Ochoa et al, 2009b) (Garrido and Riff, 2007a,b) (Pillay and Banzhaf, 2007; Pillay, 2008)
Meta-heuristics (TS, ILS, VNS)	(Ahmadi et al, 2003) (Burke et al, 2007c; Qu and Burke, 2008)
Fuzzy Systems	(Asmuni et al, 2005)
Case-based Reasoning (Offline)	(Burke et al, 2005a, 2006b)
Classifier Systems (Offline)	(Ross et al, 2002; Marín-Blázquez and Schulenburg, 2007) (Terashima-Marín et al, 2007)
Messy Genetic Algorithms (Offline)	(Ross et al, 2003, 2004; Ross and Marín-Blázquez, 2005) (Terashima-Marín et al, 2006, 2007, 2008a,b)

Table 2 High-level strategies in heuristics to choose heuristics based on constructive low-level heuristics. The strategies that incorporate a training phase (offline learning) are indicated with the word 'Offline' between brackets. TS, ILS, and VNS refer to tabu search, iterated local search and variable neighbourhood search, respectively.

conflicts between events. Thus, some of the best timetabling algorithms are based upon graph colouring heuristics. The first attempt at using evolutionary algorithms to evolve instructions for constructing an examination timetable rather than inducing the actual timetable was proposed in (Terashima-Marín et al, 1999). The approach use a non-direct chromosome representation based on evolving the configuration of constraint satisfaction methods for examination timetabling problems.

Ahmadi et al (2003) use a variable neighborhood search algorithm to find good combinations of parameterised heuristics in examination timetabling. Several constructive heuristics are proposed based on a weighted decision function and several basic graph colouring heuristics. Low-level heuristics are used for exam selection (6 heuristics), period selection (2 heuristics), and room selection (3 heuristics). These sets of heuristics include in the three cases a randomised selection. The approach is tested on a real data set from the University of Nottingham due to its rich set of constraints. No comparisons with other approaches, or further applications to other data sets, is provided.

Asmuni et al (2005) investigate the potential of implementing fuzzy system in solving course timetabling problems. The events (courses) to be scheduled are ordered by combining the following three heuristics: *largest-degree, saturation-degree, and largest-enrollment*. The fuzzy weight of an event is used to represent how difficult it is to schedule. A restricted form of exhaustive search was used to find the most appropriate shape for the fuzzy membership functions. The events are decreasingly ordered according to their weights, and then they are sequentially chosen to be assigned to the last slot with least penalty cost value. The proposed algorithm was tested on 11 benchmark data sets of course timetabling problems and the results show that this approach can produce good quality solutions with low requirements for rescheduling.

Ross et al (2004); Ross and Marín-Blázquez (2005) apply a messy genetic algorithm (Goldberg et al, 1990) hyper-heuristic based on graph colouring heuristics to both class and exam timetabling problems. The idea is to learn associations between problem states and adequate heuristics for timetabling. Specifically, the system tries to discover a set of labelled points in the space of the problem states. Each label refers to a heuristic, and the algorithm works by repeatedly finding the nearest labelled point to the current condition and applies its label until a complete solution had been built. Low-level heuristics are used to select both events and slots. A heuristic chooses an event to place, and then a slot to place it in. The approach considerers 16 ways of choosing an event and 28 ways of choosing a slot. Some points are labelled with an event-picking heuristic and others are labelled with a slot-picking heuristic. The event-picking heuristics specify an ordering, from which the first available event that does not violate a hard constraint is chosen. Various different forms of problem-state description and methods of measuring the fitness were studied. The approach was able to generate fast and simple problem-solving algorithms that offer good performance over a range of exam and class timetabling problems.

Burke et al (2005a, 2006b) use a knowledge discovery technique, case-based reasoning (Leake, 1996), as a heuristic selector for solving both course and exam timetabling problems. A set of three graph colouring heuristics: *largest-degree, colour-degree, saturation-degree*) and a hill-climbing procedure were selected as low-level heuristics. In (Burke et al, 2006b), tabu search is employed to discover the most relevant features used in evaluating the similarity between problem solving situations in case-based reasoning. The objective was to choose the best heuristics from the most similar previous problem solving situation to construct good solutions for the problem in hand. In (Burke et al, 2005a), different ways of hybridising the low level graph heuristics (with and without case-based reasoning) were compared for solving the Toronto datasets. It was shown that employing knowledge based techniques rather than randomly/systematically hybridising heuristics in a hyper-heuristic framework presented better results.

Burke et al (2007c) developed a hyper-heuristic framework that implements five commonly used graph colouring heuristics: *largest-degree, largest-weighted-degree, color-degree, largest-enrolment* and *saturation-degree.* A random ordering heuristic was also considered. Tabu search was employed as the high-level search method for producing good sequences of the low-level heuristics. Each heuristic list produced by the tabu search algorithm, is evaluated by sequentially using the individual heuristics to order the unscheduled events, and thus construct a complete timetable. 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 on both course and exam timetabling benchmark instances with competitive results. This graph-based hyper-heuristic was later extended in (Qu and Burke, 2008) 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 neighborhood search are implemented and compared to the previously implemented tabu search. The results suggests that the choice of a particular neighborhood structure on the heuristic space is not crucial to the performance. Moreover, iterative techniques such as iterated local search and variable neighborhood search, were found more effective for traversing the heuristic search space than more elaborate meta-heuristics such as tabu search. The authors suggest that the heuristic search space is likely to be smooth and contain large plateaus (i.e. areas where different heuristic sequences can produce similar quality solutions). Qu and Burke (2008) also study hybridisations of their hyper-heuristic framework with local search operating on the solution space. They found that this strategy greatly improves the performance of the overall system, making it competitive with state-of-the-art approaches on the studied benchmark instances. A further study (Ochoa et al, 2009a), use the notion of fitness landscapes to study the search space of graph colouring heuristics. The study confirms some the observations about the structure of the heuristic search space discussed in (Qu and Burke, 2008). 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 the early decisions (heuristic choices) in a construction process have a higher impact on the overall quality of the solution than the late decisions.

Pillay and Banzhaf (2007); Pillay (2008) study the performance of evolutionary algorithms on a similar search space as that discussed above, namely the space of combinations of graph colouring heuristics for examination timetabling. In the initial study (Pillay and Banzhaf, 2007), each element of the population is a string of variable length composed of characters each representing one of the following 5 low-level heuristics: *largest-degree, largest-enrollment, largest-weighted degree, saturation-degree*, and *highest-cost*. The authors report that the system produced feasible examination timetables with soft constraints within the range of other search methods employed for this purpose. They also report that the evolutionary system outperformed previous hyper-heuristics on number of the tested instances. A follow up study (Pillay, 2008) explores the effect of different representations on the performance of the evolutionary hyper-heuristic system. Specifically, three alternative encodings, fixed length, variable length, and $n - times$ heuristic combinations were tested. The results suggest that the fixed length encoding does not perform well. The performance of the other two encodings was found to be problem dependant.

3.3.2 Dispatching rules in production scheduling

Dispatching rules are among the most frequently applied heuristics in production scheduling, due to their ease of implementation and low time complexity. Whenever a machine is available, a dispatching rule inspects the waiting jobs and selects the job with the highest priority to be processed next. Dispatching rules differ from each other in the way they calculate priorities. As discussed in section 2, many early hyper-heuristic approaches were based on dispatching rules. More recently Váquez-Rodríguez et al (2007a) consider combinations of different despatching rules to solve a multi-machine cardboard box shop scheduling problems. A standard genetic algorithm was employed as the high level search of sequences of 13 despatching rules: *minimum release time, shortest processing, time, longest processing time, less work remaining, more work remaining, earliest due date, latest due date, weighed shortest processing time, weighted longest processing time, lowest weighted work remaining, highest weighted work remaining, lowest weighted due date and highest weighted due date.* The hyper-heuristic was shown to be capable of learning effective hybridisations upon despatching rules during scheduling, and thus was superior to employing single rules for the whole scheduling process. A hybrid hyper-heuristic was also developed where a genetic algorithm was employed to solve the first stage of the problem while the rest of the problem is handled by the hyper-heuristic. Experiments using different objective functions demonstrated both the effectiveness and generality of this hybrid hyper-heuristic. A study of the search space composed by sequences of dispatching rules is presented in (Váquez-Rodríguez et al, 2007b), where a formal definition and some properties of these spaces are discussed. The notion of a *decision block* is also introduced to refer to a set of decisions that are treated as a single unit (i.e. processed by a single heuristic). A further study (Ochoa et al, 2009b), conducts a landscape analysis of the dispatching rules search space. Two different objective functions, and several hyper-heuristic representation sizes (with different block sizes) are considered. The study confirms the suitability of these heuristic search spaces for evolving solutions to production scheduling problems. Moreover, some characteristics of the scheduling-rules landscapes were found to resemble those of the graph colouring hyper-heuristic landscapes (Ochoa et al, 2009a), which suggest that constructive heuristic search space share some features that may be exploited by high-level search strategies.

3.3.3 Packing heuristics in 1D packing problems

Ross et al (2002) use a modern classifier system (accuracy-based classifier system (Wilson, 1995)), 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 constructive heuristics. A simplified description of the current state of the problem is proposed, which considers the number of items remaining to be packed and their size ranges. 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 learning process a total of 890 benchmark instances from the literature were used. The authors evaluate a variety of previously proposed bin-packing heuristics, and select the four that produce the best results on the studied benchmark set: *largest-fit-decreasing, next-fit-decreasing, Djang and Finch's (DJD)* heuristic, and a variation of DJD. The approach use single-step environments, meaning that rewards were available only 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 a similar setting Ross et al (2003) use the messy genetic algorithm hyper-heuristic (described above in the context of timetabling) for learning associations between problem states and adequate heuristics for bin-packing. The approach is applied to the 1D bin-packing problem, and overall the results were found a little better than those obtained with the classifier system, and on a larger set of benchmark problems.

Marín-Blázquez and Schulenburg (2007), extend the classifier system approach 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 benchmark 1D bin-packing problems. The classifier system was able to generalise well and create solution processes that performed competitively well on a large set of NP-hard benchmark instances. The authors report that multi-step environments can obtain better results than the single-step could achieve at the expense of a higher number of training cycles.

3.3.4 Packing heuristics in 2D packing and cutting stock problems

Terashima-Marín et al (2006) apply the messy genetic algorithm hyper-heuristic to solve 2D-regular cutting stock problems. The authors also compare their classifier system hyper-heuristic against the messy genetic algorithms approach when solving 2D bin-packing problems, reporting competitive results for both approaches (Terashima-Marín et al, 2007). For a two dimensional problem, additional difficulty is introduced because it is necessary to specify where a particular figure should be placed inside the object. Therefore, two types of heuristics are required: for selecting the figures and objects, and for placing the figures into the objects. The proposed framework used a set of 8 heuristics for selecting the items: *next-fit, first-fit, best-fit, worst-fit, almost worst fit, first-fit decreasing, next-fit-decreasing*, and *Djang and Fitch* heuristics. The placement heuristics used belong to the class of bottom-left heuristics, that is, they keep the bottom-left stability in the layout. Two heuristics are considered: bottom-left, and improved-bottom-left. The state of the problem is described by the percentage of pieces that remain to be packed according to the following categories: huge items, large items, medium items, and small items. The size of the items is judged in proportion to the object area. A large empirical study was conducted. The approach produced general hyper-heuristics after going through a learning process with training and testing phases. When applied to unseen examples, those hyper-heuristics solved most of the problems very efficiently, better than the best single heuristic for each instance. A more extensive investigation of the messy genetic algorithm approach on 2D-regular instances is presented in Terashima-Marín et al (2008a). The study also extends the hyper-heuristic system to handle 2D irregular (convex polygonal) bin packing problems. Very encouraging results are reported for both types of problems.

Garrido and Riff (2007a,b) also propose a genetic algorithm based hyper-heuristic for solving the 2-D strip packing problems. In this case the system is online, that is solution methods are evolved for solving a single problem instance. The hyper-heuristic is based on a set of four low-level heuristics: *best-fit, bottom-left-fill*, a recursive heuristic that locates the objects on the bottom left corner (called HR), and a rotational heuristic called $BFDH$. The approach proposes a variable length representation that considers a categorisation of the low-level heuristics according to their functionality: greedy, ordering and rotational. The chromosome also includes the number of objects to be positioned using each low-level heuristic. The authors compare their approach against state-of-the-art algorithms on the selected instances. Very good results are reported that even outperform some of the specialised algorithms for the problem and benchmark instances.

3.3.5 Variable ordering heuristics in constraint satisfaction

A Constraint Satisfaction problem is defined by a set of variables and a set of constraints. When solving this type of problem, an important issue is that of determining which variable is the next to be instantiated. This ordering has been found to impact the cost of the solution search. Terashima-Marín et al (2008b) use the messy genetic algorithms for producing hyper-heuristics for the dynamic variable ordering within Constraint Satisfaction Problems. Combinations of condition-action rules are produced, after going through a learning process which includes training and testing phases. The evolved hyper-heuristics produced encouraging results when tested with large set of randomly generated benchmark problems.

3.3.6 Summary and discussion

We have covered the literature on 'heuristics to chose heuristics' that are based on constructive heuristics. The intellectual roots and origins of hyper-heuristics can be found in this type of approach, specially in the idea of combining priority or dispatching rules for solving production scheduling rules. From these early studies it was known that combine several rules or heuristics is advantageous over using a single one. This fact has been confirmed several times within different domains in recent years. Apart from production scheduling, which was the first domain studied, the application that has received more attention is educational timetabling. Specifically, several frameworks have been proposed that combine graph colouring heuristics in educational timetabling. Another application domain that has been widely studied is that of cutting and packing. With respect to foundational aspects, some studies are starting to look at the features of heuristic search spaces. Both online and offline approaches has been equally explored, and by far evolutionary algorithms, including genetic classifier systems and messy genetic algorithms, are the methodologies most frequently used.

3.4 Heuristics to choose heuristics based on perturbative heuristics

A perturbative (improvement) hyper-heuristic operates over a set of perturbative low-level heuristics. The search is conducted iteratively, selecting and applying a low-level heuristic (or a subset of low level heuristics) to the current solution(s) until a set of stopping conditions has been met. Most of the recently proposed perturbative hyper-heuristics perform a *single point search*, processing a single candidate solution at each iteration. However, there are others that use a population of candidate solutions within the search process. Perturbative hyper-heuristics have been applied to a wide variety of combinatorial optimisation problems as summarised in Table 3.

t

Application domain	**Reference(s)**
Channel assignment	Kendall and Mohamad (2004a,b)
Component placement	Ayob and Kendall (2003)
Personnel scheduling	Cowling et al (2000, 2002a); Cowling and Chakhlevitch (2003); Han and Kendall (2003); Burke et al (2003b); Bai et al (2007a)
Packing	Dowsland et al (2007); Bai et al (2007a)
Planning	Nareyek (2003)
Shelf space allocation	Bai and Kendall (2005); Bai et al (2008)
Timetabling	Burke et al (2003b, 2005b); Bilgin et al (2006); Chen et al (2007); Bai et al (2007a)
Vehicle routing problems	Pisinger and Ropke (2007)

Table 3 Application domains of heuristic to choose heuristics based on perturbative low-level heuristics.

Fig. 2 A hyper-heuristic framework performing single point search

3.4.1 Learning in heuristic selection

The heuristic selection process that does not use any type of learning mechanism can be *random*, *exhaustive* or their hybrid, referred to as *peckish* (Cowling and Chakhlevitch, 2003). A *learning* mechanism can be introduced into the heuristic selection to improve the decision making process over a set of possible neighbourhoods. The majority of the heuristic selection methods used within the single point perturbative hyper-heuristic framework generate *online* score(s) for each heuristic based on their performances. Then these values are processed and/or combined in a systematic manner to select the heuristic to be applied to the candidate solution at each step. All score based heuristic selection techniques require five main components to be implemented: (i) initial scoring, (ii) memory length adjustment, (iii) strategy for heuristic selection based on the scores,(iv and v) score update rules in case of improvement and worsening, respectively. All low level heuristic are assigned an initial score. Depending on the mechanism used, these scores might affect the performance of a hyper-heuristic. In general, initial scores are set to the same value, typically zero. Memory length determines the affect of previous performance of a heuristic while making the heuristic selection at a decision point. Given a set of scores, heuristic selection can be performed in many different ways. For example, *max* strategy selects the heuristic with the maximal score. On the other hand, *Roulette-wheel* (*score proportionate*) strategy associates a probability with each heuristic which is computed by dividing each individual score by the total score. Then, a heuristic is selected randomly based on these probabilities. A high score generates a higher probability of being selected.

One of the commonly used method in hyper-heuristics is *reinforcement learning*, see Kaelbling et al (1996); Sutton and Barto (1998) for more details. A reinforcement learning system interacts with the environment (or a *model* of the environment) and given a state, takes an action based on a *policy*. By trial and error, the system attempts to learn which actions to perform by evaluating state and action pairs through accumulated rewards. In the context of hyper-heuristics, rewarding and punishing each heuristic depending on their individual performance during the search is a scoring mechanism. If a low-level heuristic improves a solution, then it is rewarded and its score gets updated positively, while a worsening move causes punishment of a heuristic by decreasing its score. Different combination of operators can be designed for reward and punishment.

The acceptance strategy is an important component of any local search heuristic (operating on any search space). Two different types of acceptance strategies can be identified in the literature: *deterministic* or *non-deterministic*. Deterministic methods make the same decision for acceptance regardless of the decision point during the search using given current and new candidate solutions(s). On the other hand, a non-deterministic approach might generate a different decision at a different decision point for the same input. The decision process in almost all non-deterministic move acceptance methods requires additional parameters, such as the time (or current iteration). The heuristic selection mechanism Table 4 summarises well known heuristic selection and move acceptance methods.

Single point search based perturbative hyper-heuristics will be discussed in four distinct subsections considering the nature of their components as follows:

– Hyper-heuristics using deterministic move acceptance
– Hyper-heuristics using heuristic selection methods with no learning and non-deterministic move acceptance
– Hyper-heuristics using heuristic selection methods with learning and non-deterministic move acceptance
– Comparison studies

3.4.2 Hyper-heuristics using deterministic move acceptance

In Cowling et al (2000, 2002c), the authors proposed and compared a variety of the hyper-heuristic components on two real-world scheduling problems: a sales summit and a project presentation problem, respectively. *Simple Random* heuristic selection method chooses a low level heuristic at random at each step. *Random Gradient* is a variant of Simple Random, a randomly selected heuristic is repeatedly applied until no improvement is achieved. The same affect of Random Gradient can be achieved by modifying the operation of each heuristic as discussed and employing Simple Random. *Random* t

Component name	Reference(s)
Heuristic selection with no learning	
Simple Random	Cowling et al (2000, 2002c)
Random Permutation	Cowling et al (2000, 2002c)
Greedy	Cowling et al (2000, 2002c); Cowling and Chakhlevitch (2003)
Peckish	Cowling and Chakhlevitch (2003)
Heuristic selection with learning	
Random Gradient	Cowling et al (2000, 2002c)
Random Permutation Gradient	Cowling et al (2000, 2002c)
Choice Function	Cowling et al (2000, 2002c)
Reinforcement Learning	Nareyek (2003)
Reinforcement Learning with Tabu Search	Burke et al (2003b); Dowsland et al (2007)
Deterministic move acceptance	
All Moves	Cowling et al (2000, 2002c)
Only Improvements	Cowling et al (2000, 2002c)
Improving and Equal	Cowling et al (2000, 2002c)
Non-deterministic move acceptance	
Monte Carlo	Ayob and Kendall (2003)
Great Deluge	Kendall and Mohamad (2004a); Bilgin et al (2006)
Record to Record Travel	Kendall and Mohamad (2004b)
Tabu Search	Chakhlevitch and Cowling (2005)
Simulated Annealing	Bai and Kendall (2005); Bilgin et al (2006)
Simulated Annealing with Reheating	Dowsland et al (2007); Bai et al (2007a)
Late Acceptance	Ozcan et al (2009)

Table 4 Heuristic selection and move acceptance strategies (separated by double lines) in single-point search based hyper-heuristics

Permutation generates a random ordering of the low level heuristics and at each step applies a low level heuristic in the provided order successively. *Random Permutation Gradient* is a variant of Random Permutation that proceeds in the same manner as Random Gradient without changing the order of heuristics until no improvement is achieved. *Greedy* exhaustively applies all low level heuristics to a candidate solution and selects the one that generates the best *improved* solution. Although Random Gradient and Random Permutation Gradient heuristic selection methods make use of a random component, they can still be considered as *intelligent* heuristic selection mechanisms that embed a reinforcement learning mechanism. Initial scores of all heuristics are set to 0, which is also the lower bound for the scores. The upper bound for them is set to 1. As a score update rule, score of an improving heuristic is increased by one (additive), otherwise it is punished by decreasing its score by 1 (subtractive). The scores are kept within the bounds, hence; the memory length is set to the shortest possible value for such a reinforcement learning scheme. This type of strategy can be useful if the search landscape is highly rugged and there are not many plateaus. Except Greedy, the rest of the heuristic selections methods execute fast.

Choice Function heuristic selection methods introduced in Cowling et al (2000) are score based techniques that guide the selection of the low-level heuristics during the search process. The function adaptively ranks the low-level heuristics with respect to a combined score based on; how well a given heuristic has performed individually (Equation 1), and as a successor of previously invoked heuristic (Equation 2), and the elapsed time since the heuristic was last called (Equation 3). The first two components intensify recent performance, while the third provides an element of diversification. The parameters α, β ($\in (0,1]$) and δ reflects the relative weight of each component in the overall score computation for each heuristic. They also arrange the memory length. For example, if a local optimum is achieved and if recently used heuristics do not help, a heuristic that has not been used for a while is likely to be chosen for invocation. In this way, the search process is diversified. Score computations for the low level heuristics are repeated at each step. For implementing the heuristic selection, max and roulette wheel strategies were tested, with the former approach producing better performance. $I_n(y)$ and $T_n(y)$ ($I_n(x,y)$ and $T_n(x,y)$) denote the change in the evaluation function and the amount of time taken, respectively, when the n^{th} last time the heuristic y was selected and employed (immediately after the heuristic x).

$$\forall i,\ g_1(h_i) = \sum_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \tag{1}$$

$$\forall i,\ g_2(h_{\text{ID}}, h_i) = \sum_n \beta^{n-1} \frac{I_n(h_j, h_{\text{ID}})}{T_n(h_j, h_{\text{ID}})} \tag{2}$$

$$\forall i,\ g_3(h_i) = \mathit{elapsedTime}(h_i) \tag{3}$$

$$\forall i,\ \mathit{score}(h_i) = \alpha g_1(h_i) + \beta g_2(h_{\text{ID}}, h_i) + \delta g_3(h_i) \tag{4}$$

As the acceptance criteria, two deterministic approaches were considered: *All Moves* (AM) and *Only Improvements* (OI). The experimental results in Cowling et al (2000) show that the Choice Function–All Moves hyper-heuristic is promising. The best parameter set is obtained through a manual tuning process during the experiments. Cowling et al (2001) proposes a parameter-less Choice Function that updates all parameters ((α, β and δ)) automatically at each step based on a reinforcement learning scheme. In Cowling et al (2002c), this parameterless Choice Function–All Moves was found to outperform the simple ones over the problems studied, and produced improved results as compared to both a previous manual and a constructive approach. The design of this hyper-heuristic is further extended in Rattadilok et al (2005), by both proposing a model for general-purpose low-level-heuristics, and exploiting parallel computing frameworks for the hyper-heuristics.

Nareyek (2003) uses *Reinforcement Learning* (RL) as a heuristic selection method attempting to learn how to select the promising heuristic at each decision point. The learning process is based on scores (weights) as described previously. Each heuristic starts with the same score and they are updated by a predetermined scheme during the move acceptance process. The approach is evaluated on two problems: the Orc Quest problem Nareyek (2001), and modification of the Logistics Domain, a classic benchmark in the action-planning community. The proposed score update mechanism considers and compares alternative positive and negative reinforcement update schemes: additive/subtractive, multiplicative/divisional, and power/root adaptation.

The heuristic selection process makes a decision based on the scores of the available heuristics. Once a heuristic is selected it is directly applied, that is All Moves is used as an acceptance criterion. The results of this study suggest that combining a low rate of adaptation (additive update) for rewarding an improvement with a strong (root update) rate of adaptation for punishing a deterioration is a good choice. Moreover, choosing a heuristic with max strategy at each step generates often better results as compared to choosing a heuristic with a roulette wheel scheme.

Burke et al (2003b) presents *Reinforcement Learning with Tabu Search* (RLTS) heuristic selection method. Similar to the previous study of Nareyek Nareyek (2003), the low-level heuristics are selected according to learnt scores (ranks). The proposed hyper-heuristic also incorporates a dynamic tabu list of low-level heuristics that are temporarily excluded from the available heuristics. The algorithm deterministically selects the low-level heuristic with the highest rank (max strategy) that is not included in the tabu list. The chosen heuristic is employed regardless of whether the selected move causes an improvement or not (All Moves acceptance). If there is an improvement, the heuristic rank is increased, otherwise not only the rank is decreased, but also the heuristic is enqueued into the tabu list. The rank update scheme is additive, and the tabu list is emptied each time a non-improvement move is accepted. This hyper-heuristic is evaluated on various instances of two distinct timetabling problems:university course timetabling and nurse rostering. The results were competitive with those obtained using the state-of-the-art problem-specific techniques.

In Cowling and Chakhlevitch (2003), a range of hyper-heuristics are studied based on Simple Random and Greedy heuristic selection methods. According to the description of the Greedy method in Cowling et al (2000), it is actually the Greedy–Improving and Equal hyper-heuristics and worsening moves are never considered. On the other hand, it is possible that all heuristics might worsen the quality of the candidate solution at hand when Greedy is used. In Cowling and Chakhlevitch (2003), such situations are allowed. This is a more general approach that allows the move acceptance component to deal with the worsening moves, enriching generation of different hyper-heuristics embedding different acceptance mechanisms. In this study, Peckish heuristic selection strategies are also investigated along with four different Tabu Search based move acceptance strategies. These strategies accept an improving move and the related heuristic is removed from the tabu list if it is there. A non-improving move is accepted only if the employed heuristic is not in the tabu list. The hyper-heuristics utilise Only Improving, All Moves and a variant of All Moves that discards moves generating the same objective value with the current solution at hand are utilised as move acceptance criterion. The approaches were evaluated on a real-world personnel scheduling problem with ninety five low level heuristics yielding promising results. However, a disadvantage is that the process for selecting a low-level heuristic to apply at each decision point is slow since it involves examining all heuristics from a large set. Therefore, in Chakhlevitch and Cowling (2005), two learning strategies were experimented for choosing the subset of the fittest low-level heuristics. At each step, the changes in the quality of a solution is compiled to reflect the total improvement due to a heuristic. Greedy–Tabu Search (event based tabu list) that linearly reduces the number of the fittest low level heuristics turned out to be the most promising hyper-heuristic.

3.4.3 Hyper-heuristics using heuristic selection with no learning and non-deterministic move acceptance

In Ayob and Kendall (2003), a Monte Carlo move acceptance strategy is proposed. This approach accepts all improving moves, while the non-improving moves might be accepted with a certain probability that decreases with respect to the amount of the objective function worsening $\delta = f(s_{new}) - f(s_{old})$, assuming a minimisation problem. The authors explored a *Linear* (LMC), an *Exponential* probability function (EMC), and included their most sophisticated formulation based on the computation time and a counter of consecutive non-improvement iterations (EMCQ):

$$e^{-\frac{\Delta f m}{Q}}, \tag{5}$$

where m is the unit time in minutes and Q is a counter for successive worsening moves. EMCQ formulation is similar to that of a simulated annealing approach (Kirkpatrick et al, 1983; Cerny, 1985). The difference, is that it does not include a temperature parameter and thus a cooling schedule. Hyper-heuristics combining Simple Random and {Linear, Exponential, EMCQ} are applied to optimise the scheduling of electronic component placement on a printed circuit board. Their performances are compared to the combination of {Simple Random, Choice Function} and {All Move, Only Improving} hyper-heuristics. Simple Random–EMCQ having a non-deterministic acceptance strategy delivered a superior performance as compared to the hyper-heuristics using deterministic acceptance with and without learning for the given problem instances. Although, it seems no parameter tuning is necessary, more instructions will be executed in a unit time on a faster machine as compared to a slower machine. Consequently, different results might be produced for the same number of iterations.

In Kendall and Mohamad (2004a), a variant of the *Great Deluge* (GD) acceptance criteria Dueck (1993) is incorporated within a hyper-heuristic framework. In this acceptance strategy, at each iteration, any configuration is accepted which is not much worse than an expected objective value referred to as *level* that changes at a linear rate every step. If $f(s_{new}) < level = f(s_0) - (t\Delta F)/T$, then move is accepted at step t, where $f(s_0)$ is the objective value of the initial solution, ΔF is the difference between the objective values of $f(s_0)$ and the expected final objective value and T is the maximum number of iterations. Heuristic selection is performed randomly. Simple Random–Great Deluge generated competitive results as compared to a constructive heuristic and a genetic algorithm for solving channel assignment benchmark problems, a real world-problem from the mobile communications industry.

In another study, Kendall and Mohamad (2004b) extends *Record-to-Record Travel* (RRT) acceptance criteria Dueck (1993) to be used within hyper-heuristics. Any new candidate solution is accepted which is not much worse than the current one within a given fixed limit, referred to as *deviation.* If $\delta = f(s_{new}) - f(s_{old}) < deviation$ (assuming a minimisation problem), the algorithm will accept the new solution. If an accepted move is an improving one, then the quality of the solution is updated as well. Otherwise, it is not. Simple Random–Record-to-Record Travel hyper-heuristic is also applied to benchmark instances of a channel assignment problem. The empirical results suggest that this hyper-heuristic performs superior to the ones using All Move, Only Imroving and EMCQ move acceptance strategies. The approach is also compared to two problem specific state-of-the-art algorithms for the channel assignment problem: a constructive heuristic and a genetic algorithm. The performance of Simple Random–Record-to-Record Travel turns out to be comparable to the previous studies.

A *Simulated Annealing* (SA) acceptance method in hyper-heuristics is experimented in Bai and Kendall (2005). The approach is investigated on a shelf space allocation problem. In Simulated Annealing, the improving solutions are always accepted, and worsening moves are accepted according to the Metropolis criterion, defined by $exp(-\delta/\tau)$ where $\delta = f(s_{new}) - f(s_{old})$ is the change in the objective function, and τ is the temperature parameter. The temperature is decreased during the algorithm run using a *cooling schedule*. There are too many parameters that should be decided for Simulated Annealing, but the authors discuss the ways of computing them automatically from the beginning. Different approaches are allowed to improve an initial candidate solution that is generated by a greedy heuristic. The results show that the Simple Random–Simulated Annealing hyper-heuristics outperform Simple Random–Only Improving, Simple Random–All Moves, Greedy–Only Improving, Choice Function–All Moves, two conventional simulated annealing approaches each using a different single neighbourhood operator in all problem instances tested. Two strategies to decide the initial temperature are compared. One of them computes the initial temperature as a factor of the initial objective value, while the other one samples a set of random solutions and makes the computation based on the largest objective difference. The former scheme performs slightly better than the latter one.

3.4.4 Hyper-heuristics using heuristic selection with on-line learning and non-deterministic move acceptance

In Dowsland et al (2007), a variant of Reinforcement Learning with Tabu Search is integrated with a *Simulated Annealing with Reheating* (SAR) move acceptance strategy. In particular, RLTS is modified in such a way that 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 performance of heuristic h is based on the best cost obtained during those k iterations. Additionally, an undulating cooling schedule is proposed as a mean to deal with the effects of having different neighbourhood sizes (given by the pool of low-level heuristics used). The temperature is reduced based on the schedule in Lundy and Mees (1986):

$$t = \frac{t}{1 + \beta t}, \tag{6}$$

A reheating scheme is employed after a rejected move and the required acceptance rate r is reduced periodically as discussed in Thompson and Dowsland (1996) at every k iterations. In a way, the updates of scores for low level heuristics in Reinforcement Learning with Tabu Search and reductions of the acceptance rate in Simulated Annealing with Reheating are performed together. The proposed hyper-heuristic is applied to a packing problem of determining shipper sizes for storage and transportation. Real-world data from a cosmetics company is used as a base for generating experimental data. The Reinforcement Learning with Tabu Search–Simulated Annealing with Reheating performs better than a simpler local search strategy (random descent).

Bai et al (2007a) presents a different hyper-heuristic scheme that is possibly inspired from the studies provided in Burke et al (2003b); Bai and Kendall (2005); Dowsland et al (2007). The proposed hyper-heuristic uses a reinforcement learning mechanism with a short term memory as a heuristic selection component. Each heuristic receives a weight (score) that is updated periodically. A heuristic is selected based on the roulette wheel strategy after these weights are converted into probabilities as described before.

In this study, a different Simulated Annealing with Reheating scheme is used as a move acceptance method which executes switching between annealing and reheating phases during the search. During the annealing phase, $e^{-\frac{\Delta f}{t}}$ is used as function and the temperature is reduced using the Equation , where $\beta = \frac{(t_0 - t_{final}) i_{temp}}{maxIterations \cdot t_0 \cdot t_{final}}$ and i_{temp} is the number of iterations at a temperature. During the reheating phase, $t = \frac{t}{1 - \beta t}$ is used to increase the temperature to a value when the last improvement has been observed. Then the system goes into the annealing phase again. During the annealing phases, the percentage of accepted calls is used as the weight of a given low level heuristic, while the percentage of heuristic invocations yielding new solutions is used as its weight during the reheating phases. The proposed hyper-heuristic is tested over nurse rostering, course timetabling and bin packing problems and comparisons to previously proposed approaches show that it is competitive.

In Pisinger and Ropke (2007), a competent unified methodology is presented for solving different vehicle routing problems. The proposed approach extends the large neighbourhood search framework presented in Shaw (1998) with an adaptive layer. This layer adaptively chooses among a number of insertion and removal heuristics to intensify and diversify the search, according to learnt scores for each heuristic accumulated during the iterations. The hyper-heuristic combines the adaptive heuristic selection mechanism with a standard Simulated Annealing acceptance strategy based on a linear cooling rate. A heuristic causing a rejected move gets punished with no increase in its recent score, while an accepted move is rewarded. The rate of reward for an accepted move is higher if the resulting solution is a new one. A large number of tests were performed on standard benchmarks from the literature on five variants of the vehicle routing problem. The results proved highly promising, as the methodology was able to improve on the best known solutions of over some of the tested instances.

Ozcan et al (2009) investigates different heuristic selection methods that will perform the best in combination with a recently proposed acceptance criteria, namely; *Late Acceptance* strategy (Burke and Bykov, 2008). This method is a memory based technique that maintains the history of objective values from the previous solutions in a list of size L. The new solution is compared to a previous solution obtained at the L^{th} step and the acceptance decision is made accordingly. The results show that Simple Random performs the best when combined with Late Acceptance as compared to Greedy, Choice Function, Reinforcement Learning and Reinforcement Learning with Tabu Search. The delay within the acceptance strategy seems to deceive the learning mechanisms. As for Greedy, overall hyper-heuristic acts as a gradient hill climber and get stuck at a local optimum, hence it turns out to be the worst heuristic selection choice to be combined with Late Acceptance.

3.4.5 Comparison studies

In Bilgin et al (2006), an empirical study is carried out and thirty five of hyper-heuristics resulting from the combinations of previously proposed heuristic selection and acceptance strategies are compared: {Simple Random, Random Gradient, Random Permutation, Random Permutation Gradient, Greedy, Choice Function, Reinforcement Learning with Tabu Search} versus {All Moves, Only Improving, Improving and Equal, Great Deluge, EMCQ}. The hyper-heuristics are evaluated on a set of function optimisation, and timetabling benchmark instances. On function optimisation benchmarks, none of the combinations of heuristic selection and acceptance strategy is found to dominate over the others. However, Improving and Equal produces better average performance. It is observed that the performance differences between heuristic selection methods are not as significant as it is between the acceptance criteria. The Choice Function heuristic selection scheme produced a slightly better average performance. On the exam timetabling instances, the Choice Function–EMCQ and Simple Random–Great Deluge produce better performances than the rest of the hyper-heuristic combinations in the overall.

Ozcan et al (2006) investigates the performance of a hyper-heuristic based on the choice of low level heuristics and different hyper-heuristic frameworks. The low level heuristics are classified as either a *hill climber* (*meme*) that aims to generate an improved solution or a *mutational* heuristic that perturbs a candidate solution without considering whether the resulting solution will be improved or not. Three alternative hyper-heuristic frameworks to the standard framework (Type F_A) are proposed that allows inclusion and/or separation of the hill-climbing process. In F_A, a hyper-heuristic manages a single set of mutational and hill climbing heuristics simultaneously. F_B framework extends F_A with an additional step. A predefined single hill-climber is applied to the solution if the chosen and employed low level heuristic is a mutational one. In F_C framework, the low level heuristic set contains only mutational heuristics and a predefined hill climber is applied after each invocation of the chosen heuristic. F_D framework is the most general framework that uses two hyper-heuristics successively for managing mutational heuristics and hill climbers, respectively. The experimental results over benchmark functions suggest that using a different hyper-heuristic framework in case of single or multiple hill climbers might have a positive impact on the hyper-heuristic performance. Considering the average performance of the frameworks, F_C turns out to be the most promising one.

Ozcan et al (2008) extend the previous studies in Bilgin et al (2006); Ozcan et al (2006). Based on the observation that some low level heuristics are rarely chosen if a learning heuristic selection scheme is used, the benchmark function optimisation experiments are repeated using a reduced set of mutational heuristics and hill climbers. The results show that the choice of single hill climber affects the performances of F_B and F_C frameworks. With the best hill climber, both frameworks perform significantly better than the others. F_C with the best hill climber delivers a much better performance as compared to the genetic algorithm. Moreover, its performance turns out to be comparable to the memetic algorithms. The memetic algorithm embedding an Simple Random–Improving and Equal hyper-heuristic which is categorised as a static external-level adaptation mechanism in Ong et al (2006) also performs well. More investigations on the memetic algorithms embedding hyper-heuristics to choose hill climbers can be found in Ersoy et al (2007).

Bai et al (2008) investigates the performance of a set of *fast* approaches for solving a fresh produce inventory and shelf space allocation problem as compared to the multi-start generalised reduced gradient algorithm proposed in Bai and Kendall (2008). Four different greedy heuristics, GRASP (Feo and Resende, 1995), traditional simulated annealing using a single neighbourhood operator, Reinforcement Learning with Tabu Search–All Moves (Burke et al, 2003b), Simple Random–Simulated Annealing (Bai and Kendall, 2005) and Reinforcement Learning with Tabu Search–Simulated Annealing that combines components from the former two hyper-heuristics are implemented. A comparison between hyper-heuristics show that Simulated Annealing based hyper-heuristics perform better than Reinforcement Learning with Tabu Search–All Moves. Moreover, they deliver a matching performance to the traditional simulated annealing approach and GRASP. As a heuristic selection component, Reinforcement Learning with Tabu Search generates a slightly better performance than Simple Random within the hyper-heuristic framework.

Burke et al (2008) compares the performances of hyper-heuristics using Monte Carlo move acceptance criteria. In their study, Bai et al (2007a) synchronise parameter updates for heuristic selection and move acceptance methods, as if both hyper-heuristic components were required to operate in joint. In fact, they are separable. In this study, combinations of {Simple Random, Choice Function} versus {Simulated Annealing, Simulated Annealing with Reheating, EMCQ}, Greedy–Simulated Annealing with Reheating and the proposed hyper-heuristic in Bai et al (2007a) are tested on examination timetabling problem yielding the success of Choice Function–Simulated Annealing with Reheating. Nareyek (2003) shows that using max strategy for heuristic selection performs better. The Choice Function makes use of this strategy, while the learning mechanism in Bai et al (2007a) does not. Moreover, this mechanism rewards a heuristic based on its acceptance ignoring whether an improvement is achieved or not. This strategy combined with a short term memory might be misleading the heuristic selection process.

3.5 Population-based perturbative hyper-heuristics

In Cowling et al (2002b), an indirect genetic algorithm for solving a personnel scheduling problem is proposed. This approach can be regarded as a hyper-heuristic that uses a GA as the heuristic selection mechanism. The genetic representation is given by a sequence of integers each of which represents a single low-level heuristic. Each individual in the population gives a sequence of heuristic choices that indicates which level heuristic to apply and in what order. Extensions to the original approach are proposed that consider adaptive length representation Cowling et al (2002a), and guided operators Han and Kendall (2003). The proposed approach produces better results on the studied trainer scheduling problem when compared to both a direct encoding genetic algorithm and a memetic algorithm. It also outperforms its component heuristics.

The ant colony algorithm is used as a hyper-heuristic in Burke et al (2005b) and Chen et al (2007), to solve a personnel scheduling and a sport timetabling problem, respectively, with promising results. Each vertex in the ant algorithm represents a low-level heuristic. A number of ants (or hyper-heuristic agents) are located uniformly among the vertices and carry initial solutions. In their path through the network, the ants apply the low-level heuristic at each encountered node. The application to the sport timetabling problem Chen et al (2007), is able to produce good quality solutions for the studied instances, even outperforming best-known results on some of the instances.

3.6 Summary and discussion

The majority of the existing perturbative hyper-heuristics are designed based on a single point search framework. Initial studies concentrate on investigations of different mechanisms to manage a given set of low level heuristics. As hyper-heuristics are initially defined as “heuristics to choose heuristics”, almost no emphasise is given to the acceptance mechanisms during these initial studies. Later, it has been observed that by using a better move acceptance method, the performance can be improved substantially. After the initial studies, there is a rapid growth in the usage of well known acceptance methods in different perturbative hyper-heuristic frameworks. Most frequently used and successful heuristic selection methods are Choice Function (Cowling et al, 2000, 2002c) and reinforcement learning variants (Nareyek, 2003; Burke et al, 2003b; Dowsland et al, 2007). Simulated annealing (Bai and Kendall, 2005; Bilgin et al, 2006; Dowsland et al, 2007; Bai et al, 2007a) and great deluge variants (Kendall and Mohamad, 2004a; Bilgin et al, 2006) turns out to be the best choices as move acceptance components to be used within perturbative hyper-heuristics. It appears that the choice of move acceptance component is slightly more important than the choice of heuristic selection. One of the reasons for this might be the total number of low level heuristics used within hyper-heuristics apart from their nature. In most of the applications of hyper-heuristics to different problem domains, the number of low level heuristics does not exceed tens. In fact, Cowling and Chakhlevitch (2003); Chakhlevitch and Cowling (2005) are the only studies that employs a large set of low level heuristics within a perturbative hyper-heuristic framework. In order to observe how well the proposed hyper-heuristics generalise, they need to be applied to other problem domains. We can expect more comparison studies in the future.

In order to generate better learning schemes that will improve the decision making process during heuristic selection, one should always *remember* that a learning process involves memory. Memory length can be handled in a different way. For example, the scores for low level heuristics are updated based on the entire historical information in Burke et al (2003b); Dowsland et al (2007). On the other hand, the minimum and maximum scores are bounded in Nareyek (2003). This approach serves as some type of a forgetting mechanism for successive improving or non-improving moves. The empirical study carried out in Bai et al (2007b) on university course timetabling, shows that hyper-heuristics using a short-term memory produces better results than both an algorithm without memory and an algorithm with infinite memory. The memory length is found to be sensitive to different problem instances.

Combining simple, yet effective problem dependent low level heuristics with intelligent heuristic selection and move acceptance methods in a single framework provides a powerful search and optimisation tool. Perturbative hyper-heuristics can be used for solving any type of complex real world problems. They are easy to implement. Since, the main components of a hyper-heuristic process only problem domain independent information to perform search, they require no modification whatsoever to be used in a different problem domain. The empirical investigations up to now shows that they are fast techniques that produce solutions with *reasonable* quality in a *reasonable* time. Moreover, their performance is comparable to bespoke systems.

3.7 Heuristics to generate heuristics

Hyper-heuristics to choose heuristics have been discussed in previous sections. In contrast, this section will review the distinct class of hyper-heuristics to *generate* heuristics. The defining feature of this class is that the hyper-heuristic searches a space of heuristics constructed from components, rather than a space of complete, pre-defined, heuristics. While both classes output a solution at the end of a run, a heuristic generator also outputs the new heuristic that produced the solution.

Genetic programming Koza (1992); Koza and Poli (2005) is an evolutionary computation technique that evolves a population of executable computer programs, and is the most common methodology used in the literature to automatically generate heuristics. In the case that the evolved programs are heuristics, genetic programming can be viewed as a hyper-heuristic to generate heuristics. However, genetic programming is not inherently a hyper-heuristic, as the evolved programs can also directly represent problem solutions. For example, in symbolic regression, the solution is a formula, and the 'programs' in the population are candidate formulas, not heuristics which can be reused to find formulas for future problems.

Automatically generated heuristics may be 'disposable' in the sense that they are created for just one problem, and are not intended for use on unseen problems. Alternatively, the heuristic may be created for the purpose of reusing it on new unseen problems of a certain class. It is generally the case that all heuristics generated by a hyper-heuristic are reusable, as they can be applied to a new instance to produce a legal solution. However, they may not perform well on new instances if the hyper-heuristic has not been designed with reusability in mind. For a generated heuristic to be successful when reused, the hyper-heuristic would usually train it offline, on a set of representative problem instances.

There are a number of potential advantages of automatically generating heuristics. The characteristics of problem instances vary, and obtaining the best possible result for an instance would ideally require a new heuristic specialised to that instance, or a specialised variation of a previously created heuristic. It is inefficient for a human analyst to specialise heuristics on a per-instance basis. As such, human created heuristics are rarely successful on only one problem instance, they are usually designed to be effective on a given class of problems. In contrast, an automated heuristic design process makes it potentially feasible, and cost effective, to design a heuristic for each problem instance. As the process is automated, it is less demanding on human resources and time. As it is more specialised, a generated heuristic could even produce a better solution than that which can be obtained by any current human created heuristic, and many such examples are discussed in this section.

For example, 'best-fit' is a human created heuristic for one dimensional bin packing, and performs well on a wide range of bin packing instances. It was created as a general heuristic for all bin packing problems, and no heuristic is superior in both the average and worst case (Kenyon, 1996). However, over a narrower set of bin packing problems with piece sizes defined over a certain distribution, best-fit can be outperformed by automatically generated heuristics which are 'tailored' to the distribution of piece sizes (Burke et al, 2007a).

Table 5 presents a summary of the references in this section. The rest of the section is organised by application area, as follows; production scheduling (section 3.7.1), cutting and packing (section 3.7.2), SAT (section 3.7.3), the travelling salesman problem (section 3.7.4), function optimisation (section 3.7.5), binary decision diagrams (section 3.7.6), constraint satisfaction (section 3.7.7), and compiler optimisation (section 3.7.8). Conclusions are given in section 5.

3.7.1 Production scheduling

Genetic programming has rarely been used to solve production scheduling instances directly, because it is unsuitable for encoding solutions. However, it is highly suitable for encoding scheduling *heuristics* (Jakobovic et al, 2007). The evolution of dispatching rules is the most common application of hyper-heuristics in this domain.

Application Domain	References
Production Scheduling	Jakobovic et al (2007) Ho and Tay (2005) Tay and Ho (2008) Dimopoulos and Zalzala (2001) Geiger et al (2006)
Cutting and Packing	Burke et al (2006a, 2007a,b) Poli et al (2007) Kumar et al (2008)
Satisfiability	Fukunaga (2002, 2004, 2008) Bader-El-Din and Poli (2007, 2008)
Travelling Salesman Problem	Keller and Poli (2007b,a, 2008b,c,a) Oltean and Dumitrescu (2004)
Function Optimisation	Oltean (2005) Oltean and Grosan (2003) Tavares et al (2004)
Minimisation of Binary Decision Diagrams	Schmiedle et al (2002) Drechsler and Becker (1995) Drechsler et al (1996)
Constraint Satisfaction	Minton (1996)
Compiler Priority Functions	Stephenson et al (2003)
Quadratic Assignment	Oltean (2005)

Table 5 Application domains of heuristics to generate heuristics.

Ho and Tay (2005) and Tay and Ho (2008) employ genetic programming to evolve composite dispatching rules for the flexible job shop scheduling problem. Jakobovic et al (2007) employ the same technique for the parallel machine scheduling problem. The evolved dispatching rules are functions, which assign a score to a job based on the state of the problem. When a machine becomes idle, the dispatching rule is evaluated once for each job in the machine's queue, and each result is assigned to the job as its score. The job in the queue with the highest score is the next job to be assigned to the machine.

The terminals of the genetic programming algorithm are components of previously published dispatching rules, and the functions are the arithmetic operators and an automatically defined function (see Koza, 1994; Koza and Poli, 2005). The results obtained by the best evolved dispatching rule are better on over 85% of the instances. This shows that genetic programming can combine and rearrange heuristic components, to create heuristics superior to those which have been created by humans. Importantly, the heuristics are shown to be reusable on new problem instances because an appropriate training set is used to train them during their evolution.

Dimopoulos and Zalzala (2001) evolve priority dispatching rules for the single machine scheduling problem, to minimise the total tardiness of jobs. The terminal set is based on the human designed 'Montagne' dispatching rule, and contains five elements, representing both global and local job information. The function set consists of the four basic arithmetic operators. While the function and terminal sets are relatively simple, the system evolves heuristics superior to the Montagne, ADD, and SPT heuristics.

Geiger et al (2006) also employ genetic programming to evolve dispatching rules for single machine problems. The function and terminal sets are expanded from that presented by Ho and Tay (2005) and Dimopoulos and Zalzala (2001). Human competitive heuristics are produced under a variety of scheduling conditions, often replicating the human created heuristics for the problems. The system also obtains human-competitive results on a two machine flowshop problem, where a unique dispatching rule is evolved for each machine simultaneously.

3.7.2 Cutting and packing

Burke et al (2006a, 2007a,b) have produced a series of papers which present genetic programming as a hyper-heuristic, to generate heuristics for one dimensional bin packing. The heuristics generated by this system are functions consisting of arithmetic operators and properties of the pieces and bins. The heuristics operate within a fixed framework that packs the pieces of an instance one at a time. For each piece in turn, the framework iterates through the bins, applying the heuristic once to each. The heuristic returns a value for each bin. In initial research, the piece is placed into the first bin which receives a value greater than zero (Burke et al, 2006a). A modification of the system is later presented, where the bin which receieves the highest value is the one into which the piece is placed (Burke et al, 2007a,b).

Burke et al find that better results can be obtained by putting the piece into the bin with the highest score. All of the bins are considered by the heuristic with this method, and so the heuristic has a greater choice of bins. Burke et al (2006a) presents evolved heuristics which are equal to the first-fit heuristic. However, with the improved bin selection framework (Burke et al, 2007a,b), heuristics are found which exceed the performance of the 'best-fit' heuristic. The generated heuristics cannot be superior to best-fit in general (Kenyon, 1996), but they are shown to be better than best-fit on new instances of the same type as those they are trained on.

These heuristics maintain their performance on new instances much larger than the training set in Burke et al (2007b). This work shows that there is a trade-off between the time taken to evolve a heuristic on larger instances, and the heuristic's scalability.

Poli et al (2007) also employ genetic programming to evolve heuristics for one-dimensional bin packing. The structure within which their heuristics operate is based on matching the piece size histogram to the bin gap histogram, and is motivated by the observation that space is wasted if, when placing a piece into a bin, the remaining space is smaller than the size of the smallest piece still to be packed. For example, if the smallest piece still to be packed is of size 20, and there is a gap of size 19 available, this gap will never be filled and the space is wasted. In addition to the histogram approach, the work further differs from that of Burke et al., as linear genetic programming is employed to evolve the heuristics, and the problem is offline bin packing, rather than online.

Recent work by Kumar et al (2008) presents a genetic programming system which evolves heuristics for the biobjective knapsack problem. This is the first paper in which heuristics for a multiobjective problem have been automatically generated with a hyper-heuristic. To pack a knapsack instance, an evolved heuristic iterates through the list of pieces still to be packed, and is evaluated on each, using the profit and weight of the piece as inputs. When an evaluation returns a value of greater than or equal to one, then the iteration stops and that piece is packed. This is similar to the bin packing methodology of Burke et al (2006a), as it uses a threshold to make a decision before all of the options have been evaluated.

3.7.3 Boolean satisfiability (SAT)

Fukunaga presents 'CLASS' (Composite Learned Algorithms for SAT Search), an automated heuristic discovery system for the SAT problem. Earlier papers by Fukunaga (2002, 2004) represent the initial work, while much more analysis is given in (Fukunaga, 2008). SAT is a domain where the most successful heuristics from the literature have a similar structure. Indeed, better heuristics have been created simply by adding a 'random walk' element to an existing heuristic. Fukunaga has broken this structure down into component parts, and the CLASS system is a genetic programming methodology used to evolve human competitive heuristics consisting of these components. Amongst others, there are some components which supply a set of variables, some of which select a variable from such a set, and some which make use of conditions to decide which subset of components to execute. Fukunaga (2008) shows that certain human created heuristics from the literature, such as GWSAT and WalkSAT, can be represented with this component set. Fukunaga states that, because of the number of possibilities involved, the task of combining the components to create effective new heuristics is difficult for humans, but well suited for an automated system.

Fukunaga (2002, 2004, 2008) does not employ the genetic programming operators of crossover and mutation in their standard form. Instead of standard crossover, individuals are combined with a conditional operator, which keeps the original individuals intact and 'blends' their behaviour. If the condition is met, one individual is executed, else the other is executed. A form of mutation occurs when the combined trees violate a given maximum depth constraint. Bader-El-Din and Poli (2007) observe that this results in heuristics consisting of other nested heuristics. This results in heuristics which are composites of those in early generations, and which are therefore relatively slow to execute. Bader-El-Din and Poli present a different heuristic generation methodology for SAT, which makes use of traditional crossover and mutation operators to produce heuristics which are more parsimonious, and faster to execute. A grammar is defined, which can express four existing human created heuristics, and allows significant flexibility to create completely new heuristics.

SAT is also the test domain for a proof of concept of the inc* system, presented by Bader-El-Din and Poli (2008). The inc* methodology incrementally solves a sequence of progressively more complex instances with a fixed human created heuristic. An instance is decomposed into a much simpler instance to solve, before using the result to solve one slightly more complex. The difficulty of the instance is incrementally increased in this way until the original instance is reached and solved. The example given in the paper is to progressively add clauses to SAT instances. The inc* algorithm involves backtracking in the situation where an instance is judged too difficult, and the problem is made simpler again. Genetic programming is used to evolve the heuristic which decides how many clauses to remove from the instance when backtracking, and how many to add to the instance when making it more complex. The inc* methodology as a concept is potentially general over many problem domains, but the heuristic to simplify the problem is necessarily problem specific. The genetic programming operates as a hyper-heuristic in this case because it searches the space of these problem specific heuristics.

3.7.4 Travelling Salesman Problem

Keller and Poli (2007b) present a linear genetic programming hyper-heuristic for the travelling salesman problem (TSP). The hyper-heuristic evolves programs which represent the repeated application of a number of simple local search operations. The programs are sentences of a language defined by a given grammar, which is progressively made more complex.

The system is first shown to evolve sequences of swap heuristics which perform better than random search. The two swap heuristics involve rearranging two or three pairs of edges. Conditional and loop components are then added to the grammar, to increase the complexity of the evolved heuristics. The results obtained by the evolved heuristics are within 1% of a state of the art hybrid genetic algorithm, which obtains the best known results on those two instances. Further analysis of the algorithm is provided in Keller and Poli (2007a), with regards to the computational resource utilisation.

The loop component in the grammar is controlled by one parameter, which defines how many times the loop will be executed. Keller and Poli (2008b) show the improvement in efficiency when this parameter is evolved along with the heuristic itself. The quality of the results does not improve, but the number of low-level heuristic calls decreases. It is not shown if the evolved heuristics can be reused successfully on new problem instance. However, they are successful as disposable heuristics on the instance they were evolved for, and it is shown that a good heuristic is produced consistently when the genetic programming hyper-heuristic is run numerous times.

Keller and Poli (2008c,a) further extend their grammar by adding a ‘subheuristic’ component. This is a component which is a constituent part of a heuristic. Specifically, they add one component to the grammar which swaps the order of two nodes in the tour. This increases the efficiency of the search, with less heuristic calls needed to produce the same quality of solution.

Also for the travelling salesman problem, Oltean and Dumitrescu (2004) evolve constructive heuristics, as opposed to the local search heuristics evolved by Keller and Poli (2007b). They use multi expression programming to evolve functions that decide which node of the graph to add to the tour. This is another example of the common technique of evolving a scoring function which operates within a fixed iterative framework. The decision-maker is evolved, but the context within which the decision is made remains fixed. In general, at each decision point, the function is evaluated on all available options to obtain a score for each. The option with the highest score is the one that is actually performed on the partial solution. In this case, Oltean and Dumitrescu (2004) apply the candidate function to all of the cities that are not yet included in the partial tour, and the one which receives the highest score from the function is added to the tour.

3.7.5 Function Optimisation

Oltean (2005) present a linear genetic programming hyper-heuristic, which generates evolutionary algorithms. The problem domains employed to validate the technique are function optimisation, the travelling salesman problem, and the quadratic assignment problem.

A standard individual in a linear genetic program represents a series of instructions that manipulate values in a memory array. The memory positions are referred to as ‘registers’ A typical instruction could be similar to that shown in equation 7.

$$reg[1] = reg[0] + reg[7] \tag{7}$$

Oltean (2005) represent an evolutionary algorithm population as this memory array, with each member of the population stored in one register. The genetic operators are the instructions that operate on the memory array. For example, equation 8 and 9 show two possible instructions which operate on the population array. Each individual in the genetic programming population therefore represents a method of evolving a population.

$$reg[1] = crossover(reg[7], reg[3]) \tag{8}$$

$$reg[3] = mutate(reg[5]) \tag{9}$$

This work is based on earlier work by Oltean and Grosan (2003), which evolves evolutionary algorithms with multi expression programming. This is similar to linear genetic programming, but each chromosome encodes multiple evolutionary algorithms as the instructions are not necessarily executed sequentially.

Tavares et al (2004) also present a methodology for evolving an evolutionary algorithm. They specify the main components of a generic evolutionary algorithm, including initialisation, selection, and the use of genetic operators. Tavares et al explain how each of these steps, can be evolved individually by a genetic programming hyper-heuristic. They demonstrate the approach through an example of evolving an effective mapping function of genotype to phenotype, for a function optimisation problem.

3.7.6 Binary Decision Diagrams

Schmiedle et al (2002) evolve heuristics for the minimisation of binary decision diagrams. A heuristic in this domain can be represented as an ordering of predefined optmisation modules. Their genetic programming hyper-heuristic evolves an ordering by assigning the optimisation modules to be terminals. The genetic programming functions are iterative and conditional nodes, which therefore apply the terminal nodes in a given order. This work is based on work by Drechsler and Becker (1995), who evolve heuristics for computer aided design of ICs, using a genetic algorithm. Subsequent work by Drechsler et al (1996) employs an evolutionary algorithm to evolve heuristics for minimising ordered binary decision diagrams.

3.7.7 Constraint Satisfaction

In the domain of constraint satisfaction problems, Minton (1996) presents a system for generating reusable heuristics for the minimum maximal matching problem. The system modifies given elements of algorithm ‘schema’, which are templates of generic algorithms. This study is unique in the literature, as the hyper-heuristic is compared against three NASA programmers, to assess whether the automatically generated heuristics are human competitive. Both the programmers and the hyper-heuristic were given a maximum of eight hours to develop their heuristics. The results were very competitive with the human generated heuristics, and better in one out of the three experiments.

3.7.8 Compiler Optimisation

Stephenson et al (2003) employ genetic programming as a hyper-heuristic to evolve improved compiler priority functions for three aspects of compiler optimisation. While they do not use the same terminology, Stephenson et al (2003) perform experiments evolving disposable and reusable heuristics, and find that extremely good heuristics can be automatically generated on a per-instance basis. This shows the flexibility of a hyper-heuristic approach to generating heuristics. Because the output is both a heuristic and a solution to the training instances, the methodology can simply be an effective method to generate a solution to just one instance. The evolved heuristics consistently outperform the benchmark human created heuristic in each of the three compiler optimisation areas.

3.7.9 Summary and Discussion

We have presented a summary of the hyper-heuristic literature on 'heuristics to generate heuristics'. Very promising results have been obtained on a wide variety of optimisation problems, which have relied on human-generated heuristics thus far. The literature shows that, typically, evolutionary computation methods are employed to automatically generate heuristics, which are reusable on new problem instances.

The evolutionary heuristic generation process is often computationally expensive when compared with a methodology that operates directly on the solution space. However, this is only a disadvantage in the short term, when results will not be required for future problems. Consider the application of an evolutionary algorithm directly to the problem space. The output is just the solution to the instance, and the entire evolutionary algorithm must be run again if a solution is required for future problems. If the evolutionary process is employed instead as a hyper-heuristic, to generate a quick reusable heuristic, then only one run is required. The heuristic can then obtain a comparable result on the future problems, much more quickly than the application of an evolutionary algorithm. This is one of the main benefits of searching for a solution *method* rather than just searching for a solution.

While the evolution process is long compared with a direct search methodology, it is often quicker than manual heuristic generation. For example, Geiger et al (2006) state that production scheduling heuristics from the literature are the result of years of scheduling research, and the identical evolved heuristic rules are generated within a fraction of this time. This illustrates the power of automatically generating heuristics with genetic programming.

However, the potential components of the evolved heuristics must still be defined by humans, and it can be argued that successful sets of components are only inspired by the literature on human created heuristics. The research in this new area of automatic heuristic generation, shows that it is not yet able to completely replace human ingenuity. As Fukunaga (2008) states, humans are able to invent good building blocks. However, the literature *does* show that hyper-heuristic methodologies have been able to successfully combine these human-defined building blocks in superior ways.

4 Related Areas

Heuristic search is widely used nowadays in Operational Research and Artificial Intelligence. In its different varieties, tree-based search and local search, it provides the core engine for real-world applications as diverse as timetabling, planning, personal and production scheduling, cutting and packing, space allocation, and bioinformatics. A promising direction for developing improved search techniques is to integrate learning components that can adaptively guide the search. Many techniques have independently arisen in recent years that exploit either some form of learning, or search on a configuration space, to improve problem-solving and decision making. These techniques can be offline or online, and applied to tree-structured or local search. We briefly survey some of these approaches below.

4.1 Offline approaches

Algorithm configuration, that is, the determination of appropriate values for free algorithm parameters, is commonly (either implicitly or explicitly) treated as an optimisation problem, where the objective function captures performance on a fixed set of benchmark instances Hutter et al (2007). Depending on the number and type of parameters, the methods used to solve this optimisation problem include exhaustive enumeration, beam search (Minton, 1996), experimental design (Coy et al, 2001; Ridge and Kudenko, 2007), the application of racing algorithms (Birattari et al, 2002; Birattari, 2004), combinations of fractional experimental design and local search (Adenso-Diaz and Laguna, 2006), and iterated local search (Hutter et al, 2007).

An interesting related approach at the interphase between machine learning and engineering was termed “Teacher” (Wah et al, 1995; Wah and Ieumwananonthachai, 1999) (an acronym for TEchniques for the Automated Creation of HEuRistics), which was designed as a system for learning and generalising heuristics used in problem solving. The objective was to find, under resource constraints, improved heuristic methods (HMs) as compared to existing ones, in applications with few or non-existent domain knowledge. The Teacher system employed a genetic-based machine learning approach, and was successfully applied to several domains such as: process mapping, load balancing on a network of workstations, circuit placement, routing and testing, among others.

4.2 Online approaches

A different approach to algorithm configuration, also called parameter control, is the idea of tuning algorithm parameters on-line at execution time. This has since long being the approach in evolution strategies (Rechenberg, 1973), where feedback from the search process is used to control the mutation step size. In (Eiben et al, 1999), the authors proposed an useful classification into adaptive and self-adaptive approaches, and surveyed previous work on parameter control in evolutionary algorithms.

Reactive search (Battiti and Brunato, 2007; Battiti and Protasi, 1997; Battiti, 1996) is an on-line methodology that advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimisation problems. The machine learning component acts on top of the search heuristic, in order to let the algorithm self-tune its operating parameters during the search operation. The learning component is implemented as a reactive feedback scheme that uses the past history of the search to increase its efficiency and efficacy. These ideas have been mainly applied to the Tabu search meta-heuristic. In (Battiti and Brunato, 2007), the authors briefly mention other techniques related to reactive search such as model based search (Baluja and Davies, 1997; Pelikan et al, 2002), guided local search (Voudouris and Tsang, 1999), ant colony optimisation (Gambardella et al, 1999; Dorigo et al, 1996), and dynamic local search (Hutter et al, 2002)

Although generally not including an adaptive mechanism, Variable Neighbourhood search (VNS) (Mladenovic and Hansen, 1997), is related to improvement hyper-heuristics in that they exploit the search power of multiple neighbourhoods. VNS systematically switches neighbourhoods in a predefined sequence so that the search can explore increasingly distant neighbourhoods of the current solution. Another approach closely related to improvement hyper-heuristics is that of adaptive memetic algorithms (MAs), a recent breed of hybrid evolutionary algorithms, in which several memes (or local searchers) are available and adaptively selected (or evolved altogether) during the search (Jakob, 2006, 2002; Krasnogor and Smith, 2000, 2001; Krasnogor and Gustafson, 2004; Ong and Keane, 2004; Ong et al, 2006; Smith, 2003, 2007). An important distinction between hyper-heuristics and adaptive MAs is that the former concentrates on searching in the heuristic space (generally using single point heuristic search), while the latest search simultaneously on both spaces by maintaining two populations: one of memes and one of genes.

Finally, an alternative way of automating the design of search techniques is the *algorithm portfolio* method, first proposed in (Huberman et al, 1997), which follows the standard practice in economics to obtain different return-risk profiles in the stock market by combining different stocks. An algorithm portfolio would run different algorithms concurrently in a time sharing manner, by allocating a fraction of the total CPU cycles to each of them. The first algorithm to finish reports the solution and determines the completion time of the portfolio, while the other algorithms are immediately stopped. Dynamic portfolios, that include online learning, have been considered in (Gagliolo and Schmidhuber, 2006)

5 Discussion and future work

The term hyper-heuristics was introduced in the early 2000s to describe the idea of 'heuristics to choose heuristics' in the context of combinatorial optimisation. The origin of the idea can, however, be traced back to the early 1960s and was independently explored several times during the 1990s. The defining feature of hyper-heuristics is that they operate on a search space of heuristics rather than directly on the search space of solutions to the problem at hand. This feature provides the potential for increasing the generality of search methodologies. Several hyper-heuristic approaches have been proposed that incorporate different search and machine learning paradigms.

With the incorporation of genetic programming into hyper-heuristic research, a new class of approaches can be identified that we have termed 'heuristics to generate heuristics'. 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, as compared to the more classic "heuristic to choose heuristics", since they require the decomposition of the available existing heuristics, and the design of an appropriate framework. There is much more to be done in this class of approaches.

Both online and offline approaches are valuable directions of research in hyper-heuristics. In offline approaches, the significant search effort is applied to find a search algorithm, but once the algorithm has been found, it is extremely cheap and fast to apply it to any new instance with no additional search effort. In other words, we have a reusable method. On the other hand, online approaches are ready available to be applied to newly encountered instances or problems without a training phase, they adapt while solving a given instance (although they include the overhead of online learning). Searching on a space of heuristics may be more effective than searching directly on the underlying problem space, as heuristics my provide an advantageous structure to the search space. Moreover, in newly encountered problems there may not be a set of related instances in which to train off-line hyper-heuristic methods.

In the light of the progress and incorporation of new techniques into hyper-heuristic research, we prose the following updated definition of hyper-heuristics:

> *Hyper-heuristics comprise a set of approaches with the goal of automating, often by the incorporation of machine learning techniques, the process of either (i) selecting and combining simpler heuristics, or (ii) generating new heuristics from components of existing heuristics; in order to solve hard computational search problems.*

While most of the related approaches discussed above deal with either automatically tuning the free parameters in a fixed search heuristic, or selecting a fixed algorithm for the problem at hand; hyper-heuristics represent a more general framework that aims at intelligently combine or hybridise basic perturbative or constructive heuristics. Furthermore, when compared with other hybridisation techniques, such as adaptive memetic algorithms, hyper-heuristics stand again as a more general search methodology, not restricted to the evolutionary metaphor. Therefore, hyper-heuristic research has the potential of bringing together promising ideas in the fields of Stochastic Local Search and Machine Learning, with knowledge (in the form of heuristics) accumulated over the years in the field of Operational Research. The overall aim is to solve complex real-life combinatorial optimisation problems in a more general fashion.

Hyper-heuristics research lies in the interphase of machine learning and optimisation. Machine learning is an old discipline in Artificial Intelligence, with a set of proven methodologies and tools. The exploration of these techniques for automating the design of heuristics has only started. Therefore, some of the promising areas for further research in hyper-heuristics encompass the integration of additional machine learning techniques, the study of heuristic search spaces, and the integration of co-operative and fuzzy techniques. Additional studies are also required to seize the generality of these approaches in a controlled and statistically sounded manner. The long-term goal is to produce reusable technologies to underpin future research into automated heuristic design to facilitate systems which can work with users to home in on high quality solutions to problems.

References

Adenso-Diaz B, Laguna M (2006) Fine-tuning of algorithms using fractional experimental design and local search. Operations Research 54(1):99–114

Ahmadi S, Barrone P, Cheng P, Burke EK, Cowling P, McCollum B (2003) Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. In: Proceedings of Multidisciplinary International Scheduling: Theory and Applications (MISTA 2003), Nottingham, August 13-16, pp 155–171

Asmuni H, Burke EK, Garibaldi JM (2005) Fuzzy multiple heuristic ordering for course timetabling. In: Proceedings of the 5th United Kingdom Workshop on Computational Intelligence (UKCI 2005), London, UK, pp 302–309

Ayob M, Kendall G (2003) A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In: Proceedings of the International Conference on Intelligent Technologies (InTech'03), Chiang Mai, Thailand, pp 132–141

Bader-El-Din MB, Poli R (2007) Generating sat local-search heuristics using a gp hyper-heuristic framework. In: LNCS 4926. Proceedings of the 8th International Conference on Artifcial Evolution, pp 37–49

Bader-El-Din MB, Poli R (2008) An incremental approach for improving local search heuristics. In: LNCS 4972. Proceedings of the 8th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP'08), Naples, Italy, pp 194–205

Bai R, Kendall G (2005) An investigation of automated planograms using a simulated annealing based hyper-heuristics. In: Ibaraki T, Nonobe K, Yagiura M (eds) Metaheuristics: Progress as Real Problem Solver - (Operations Research/Computer Science Interface Serices, Vol.32), Springer, pp 87–108

Bai R, Kendall G (2008) A model for fresh produce shelf-space allocation and inventory management with freshness-condition-dependent demand. INFORMS Journal on Computing 20(1):78–85

Bai R, Blazewicz J, Burke EK, Kendall G, McCollum B (2007a) A simulated annealing hyper-heuristic methodology for flexible decision support. Tech. Rep. NOTTCS-TR-2007-8, School of CSiT, University of Nottingham

Bai R, Burke EK, Kendall G, McCollum B (2007b) Memory length in hyper-heuristics: An empirical study. In: Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling (CISched2007), Hilton Hawaiian Village, Honolulu, Hawaii, USA

Bai R, Burke EK, Kendall G (2008) 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

Baluja S, Davies S (1997) Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In: Proc. 14th International Conference on Machine Learning, Morgan Kaufmann, pp 30–38

Battiti R (1996) Reactive search: Toward self–tuning heuristics. In: Rayward-Smith VJ, Osman IH, Reeves CR, Smith GD (eds) Modern Heuristic Search Methods, John Wiley & Sons Ltd., Chichester, pp 61–83

Battiti R, Brunato M (2007) Reactive search: Machine learning for memory-based heuristics. In: Gonzalez TF (ed) Approximation Algorithms and Metaheuristics, Taylor and Francis Books (CRC Press), Washington, DC, chap 21, pp 1–17

Battiti R, Protasi M (1997) Reactive search, a history-sensitive heuristic for MAX-SAT. ACM Journal of Experimental Algorithms 2:2

Bilgin B, Ozcan E, Korkmaz EE (2006) An experimental study on hyper-heuristics and final exam scheduling. In: Proc. of the International Conference on the Practice and Theory of Automated Timetabling (PATAT'06), pp 123–140

Birattari M (2004) The problem of tuning metaheuristics as seen from a machine learning perspective. PhD thesis, Universite Libre de Bruxelles

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002, Morgan Kaufmann, pp 11–18

Burke EK, Bykov Y (2008) A late acceptance strategy in hill-climbing for exam timetabling problems. In: Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)

Burke EK, Hart E, Kendall G, Newall J, Ross P, Schulenburg S (2003a) Hyper-heuristics: An emerging direction in modern search technology. In: Glover F, Kochenberger G (eds) Handbook of Metaheuristics, Kluwer, pp 457–474

Burke EK, Kendall G, Soubeiga E (2003b) A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics 9(6):451–470

Burke EK, Dror M, Petrovic S, Qu R (2005a) The Next Wave in Computing, Optimization, and Decision Technologies, Springer, chap Hybrid Graph Heuristics within a Hyper-heuristic Approach to Exam Timetabling Problems, pp 79–91. URL http://www.cs.nott.ac.uk/ rxq/files/INFORMS.pdf

Burke EK, Kendall G, Landa-Silva JD, O'Brien R, Soubeiga E (2005b) An ant algorithm hyperheuristic for the project presentation scheduling problem. In: Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, Scotland, vol 3, pp 2263–2270

Burke EK, Hyde MR, Kendall G (2006a) Evolving bin packing heuristics with genetic programming. In: Runarsson T, Beyer HG, Burke E, JMerelo-Guervos J, Whitley D, Yao X (eds) LNCS 4193, Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN'06), Reykjavik, Iceland, pp 860–869

Burke EK, Petrovic S, Qu R (2006b) Case based heuristic selection for timetabling problems. Journal of Scheduling 9(2):115–132

Burke EK, Hyde MR, Kendall G, Woodward J (2007a) 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), London, UK., pp 1559–1565

Burke EK, Hyde MR, Kendall G, Woodward J (2007b) The scalability of evolved on line bin packing heuristics. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07), Singapore, pp 2530–2537

Burke EK, McCollum B, Meisels A, Petrovic S, Qu R (2007c) A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research 176:177–192

Burke EK, Kendall G, sır MM, Özcan E (2008) Monte carlo hyper-heuristics for examination timetabling. In: Conference on the Practice and Theory of Automated Timetabling (PATAT 2008)

Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward J (2009a) A classification of hyper-heuristic approaches. In: MISTA

Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward J (2009b) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C, Jain L (eds) Collaborative Computational Intelligence, Springer

Cerny V (1985) Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. Journal of Optimization Theory and Applications 45(1):41–51

Chakhlevitch K, Cowling P (2005) Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In: Proceedings of 5th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP2005), Springer, Lecture Notes in Computer Science, vol 3448, pp 25–33

Chakhlevitch K, Cowling PI (2008) Hyperheuristics: Recent developments. In: Cotta C, Sevaux M, Sörensen K (eds) Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, vol 136, Springer, pp 3–29

Chen PC, Kendall G, Vanden-Berghe G (2007) An ant based hyper-heuristic for the travelling tournament problem. In: Proceedings of IEEE Symposium of Computational Intelligence in Scheduling (CISched 2007), Hawaii, pp 19–26

Cowling P, Chakhlevitch K (2003) Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC2003), IEEE Computer Society Press, Canberra, Australia, pp 1214–1221

Cowling P, Kendall G, Soubeiga E (2000) 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, Springer, Konstanz, Germany, Lecture Notes in Computer Science, pp 176–190

Cowling P, Kendall G, Soubeiga E (2001) A parameter-free hyperheuristic for scheduling a sales summit. In: Proceedings of the 4th Metaheuristic International Conference, pp 127–131

Cowling P, Kendall G, Han L (2002a) An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem. In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02), Orchid Country Club, Singapore, pp 267–271

Cowling P, Kendall G, Han L (2002b) An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In: Proceedings of the Congress on Evolutionary Computation (CEC2002), Hilton Hawaiian Village Hotel, Honolulu, Hawaii, USA, pp 1185–1190

Cowling P, Kendall G, Soubeiga E (2002c) Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In: Cagoni S, Gottlieb J, Hart E, Middendorf M, Goenther R (eds) Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002, Springer-Verlag, Kinsale, Ireland, Lecture Notes in Computer Science, vol 2279, pp 1–10

Coy S, Golden BL, Runger GC, Wasil EA (2001) Using experimental design to find effective parameter settings for heuristics. J Heuristics 7(1):77–97

Crowston WB, Glover F, Thompson GL, Trawick JD (1963) Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh -(117)

Dimopoulos C, Zalzala AMS (2001) Investigating the use of genetic programming for a classic one-machine scheduling problem. Advances in Engineering Software 32(6):489–498

Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man and Cybernetics, Part B 26(1):29–41

Dorndorf U, Pesch E (1995) Evolution based learning in a job shop scheduling environment. Computers and Operations Research 22(1):25–40

Dowsland KA, Soubeiga E, Burke EK (2007) A simulated annealing hyper-heuristic for determining shipper sizes. European Journal of Operational Research 179(3):759–774

Drechsler R, Becker B (1995) Learning heuristics by genetic algorithms. In: Proceedings of the ASP Design Automation Conference, Mukuhari, Japan, pp 349–352

Drechsler R, Göckel N, Becker B (1996) Learning heuristics for obdd minimization by evolutionary algorithms. In: LNCS 1141. Proceedings of the 4th International Conference on Parallel Problem Solving from Nature (PPSN'96), Berlin, Germany, pp 730–739

Dueck G (1993) New optimization hueristics: The great deluge algorithm and the record-to record travel. Journal of Computational Physics 104:86–92

Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in Evolutionary Algorithms. IEEE Transactions on Evolutionry Computation 3(2):124

Ersoy E, Ozcan E, Etaner-Uyar AS (2007) Memetic algorithms and hyperhill-climbers. In: Baptiste P, Kendall G, Kordon AM, Sourd F (eds) MISTA 2007: 3rd Multidisciplinary International Scheduling Conference: Theory and Applications, pp 159–166

Fang H, Ross P, Corne D (1993) A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. In: Forrest S (ed) Fifth International Conference on Genetic Algorithms, Morgan Kaufmann, San Mateo, pp 375–382

Fang H, Ross P, Corne D (1994) A promising hybrid ga/ heuristic approach for open-shop scheduling problems. In: Cohn A (ed) Eleventh European Conference on Artificial Intelligence, John Wiley & Sons

Feo TA, Resende MGC (1995) Greedy randomized adaptive search procedures. Journal of Global Optimization 6:109–133

Fisher H, Thompson GL (1961) Probabilistic learning combinations of local job-shop scheduling rules. In: Factory Scheduling Conference, Carnegie Institue of Technology

Fisher H, Thompson GL (1963) Probabilistic learning combinations of local job-shop scheduling rules. In: Muth JF, Thompson GL (eds) Industrial Scheduling, Prentice-Hall, Inc, New Jersey, pp 225–251

Fukunaga AS (2002) Automated discovery of composite sat variable-selection heuristics. In: Eighteenth national conference on Artificial intelligence, American Association for Artificial Intelligence, Menlo Park, CA, USA, pp 641–648

Fukunaga AS (2004) Evolving local search heuristics for SAT using genetic programming. In: Deb K, Poli R, Banzhaf W, Beyer HG, Burke E, Darwen P, Dasgupta D, Floreano D, Foster J, Harman M, Holland O, Lanzi PL, Spector L, Tettamanzi A, Thierens D, Tyrrell A (eds) LNCS 3103. Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO '04), Springer-Verlag, Seattle, WA, USA, pp 483–494

Fukunaga AS (2008) Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation (MIT Press) 16(1):31–1

Gagliolo M, Schmidhuber J (2006) Dynamic algorithm portfolios. In: Proceedings AI and MATH 06, Ninth International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida

Gambardella LM, Taillard ED, , Dorigo M (1999) Ant colonies for the quadratic assignment problem,. Journal of the Operational Research Society 50(2):167–176

Garrido P, Riff MC (2007a) Collaboration between hyperheuristics to solve strip-packing problems. In: 12th International Fuzzy Systems Association World Congress, Proceedings, Springer, Lecture Notes in Computer Science, vol 4529, pp 698–707

Garrido P, Riff MC (2007b) An evolutionary hyperheuristic to solve strip-packing problems. In: Intelligent Data Engineering and Automated Learning - IDEAL 2007 Proceedings, Springer, Lecture Notes in Computer Science, vol 4881, pp 406–415

Geiger CD, Uzsoy R, Aytug H (2006) Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. Journal of Scheduling 9(1):7–34

Goldberg DE, Korb B, Deb K (1990) Messy genetic algorithms: Motivation, analysis, and first results. Complex Systems 3(5):493–530

Gratch J, Chien S (1996) Adaptive problem-solving for large-scale scheduling problems: a case study. Journal of Artificial Intelligence Research 4:365–396

Gratch J, Chien S, DeJong G (1993) Learning search control knowledge for deep space network scheduling. In: Proceedings of the Tenth International Conference on Machine Learning, Amherst, MA, pp 135–142

Han L, Kendall G (2003) Guided operators for a hyper-heuristic genetic algorithm. In: Proceedings of AI-2003: Advances in Artificial Intelligence. The 16th Australian Conference on Artificial Intelligence (AI 03), Perth, Australia, pp 807–820

Hart E, Ross P (1998) A heuristic combination method for solving job-shop scheduling problems. In: Eiben AE, Back T, Schoenauer M, Schwefel HP (eds) Parallel Problem Solving from Nature V, Springer-Verlag, Lecture Notes in Computer Science, vol 1498, pp 845–854

Hart E, Ross P, Nelson JAD (1998) Solving a real-world problem using an evolving heuristically driven schedule builder. Evolutionary Computing 6(1):61–80

Ho NB, Tay JC (2005) Evolving dispatching rules for solving the flexible job-shop problem. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC'05), Edinburgh, UK, pp 2848–2855

Huberman BA, Lukose RM, , Hogg T (1997) An economics approach to hard computational problems. Science 275:51–54

Hutter F, Tompkins DA, , Hoos HH (2002) Scaling and probabilistic smoothing: Efficient dynamic local search for sat. In: Proceedings Constraint Programming 2002, Springer, Lecture Notes in Computer Science, vol 2470, pp 233–248

Hutter F, Hoos HH, Stützle T (2007) Automatic algorithm configuration based on local search. In: AAAI, AAAI Press, pp 1152–1157

Jakob W (2002) HyGLEAM – an approach to generally applicable hybridization of evolutionary algorithms. In: Parallel Problem Solving from Nature - PPSN VII, Springer, Berlin, pp 527–536

Jakob W (2006) Towards an adaptive multimeme algorithm for parameter optimisation suiting the engineers' needs. In: Parallel Problem Solving from Nature - PPSN IX, 9th International Conference, Reykjavik, Iceland, September 9-13, 2006, Procedings, Springer, Lecture Notes in Computer Science, vol 4193, pp 132–141

Jakobovic D, Jelenkovic L, Budin L (2007) Genetic programming heuristics for multiple machine scheduling. In: LNCS 4445. Proceedings of the European Conference on Genetic Programming (EUROGP'07, Valencia, Spain, pp 321–330

Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. Journal of Artificial Intelligence Research 4:237–285

Keller RE, Poli R (2007a) Cost-benefit investigation of a genetic-programming hyperheuristic. In: Proceedings of the 8th International Conference on Artifcial Evolution, Tours, France, pp 13–24

Keller RE, Poli R (2007b) Linear genetic programming of parsimonious metaheuristics. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07), Singapore, pp 4508–4515

Keller RE, Poli R (2008a) Self-adaptive hyperheuristic and greedy search. In: Wang J (ed) 2008 IEEE World Congress on Computational Intelligence (WCCI'08), Hong Kong

Keller RE, Poli R (2008b) Subheuristic search and scalability in a hyperheuristic. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08), pp 609–610

Keller RE, Poli R (2008c) Toward subheuristic search. In: Wang J (ed) 2008 IEEE World Congress on Computational Intelligence (WCCI'08), Hong Kong

Kendall G, Mohamad M (2004a) Channel assignment in cellular communication using a great deluge hyper-heuristic. In: Proceedings of the 2004 IEEE International Conference on Network (ICON2004), Singapore, pp 769–773

Kendall G, Mohamad M (2004b) Channel assignment optimisation using a hyper-heuristic. In: Proceedings of the 2004 IEEE Conference on Cybernetic and Intelligent Systems (CIS2004), Singapore, pp 790–795

Kenyon C (1996) Best-fit bin-packing with random order. In: Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp 359–364

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:671–680

Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. The MIT Press, Boston, Massachusetts

Koza JR (1994) Genetic programming II: automatic discovery of reusable programs. The MIT Press, Cambridge, Massachusetts

Koza JR, Poli R (2005) Genetic programming. In: Burke EK, Kendall G (eds) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Kluwer, Boston, pp 127–164

Krasnogor N, Gustafson S (2004) A study on the use of 'self-generation' in memetic algorithms. Natural Computing 3(1):53 – 76

Krasnogor N, Smith JE (2000) A memetic algorithm with self-adaptive local search: TSP as a case study. In: Proceedings of the 2000 Genetic and Evolutionary Computation Conference, Morgan Kaufmann

Krasnogor N, Smith JE (2001) Emergence of profitable search strategies based on a simple inheritance mechanism. In: Proceedings of the 2001 Genetic and Evolutionary Computation Conference, Morgan Kaufmann

Kumar R, Joshi AH, Banka KK, Rockett PI (2008) Evolution of hyperheuristics for the biobjective 0/1 knapsack problem by multiobjective genetic programming. In: Proceedings of the 10th ACM conference on Genetic and evolutionary computation (GECCO'08), Atlanta, GA, USA, pp 1227–1234, DOI http://doi.acm.org/10.1145/1389095.1389335

Leake DB (1996) Case Based Reasoning: Experiences, Lessons, and Future Directions. AAI Press/MIT Press

Lundy M, Mees A (1986) Convergence of an annealing algorithm. Mathematical Programming 34:111–124

Marín-Blázquez JG, Schulenburg S (2007) A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In: IWLCS, Springer, Lecture Notes in Computer Science, vol 4399, pp 193–218

Minton S (1996) Automatically configuring constraint satisfaction problems: a case study. Constraints 1(1):7–43

Mladenovic N, Hansen P (1997) Variable neighborhood search. Computers and Operations Research 24(11):1097–1100

Nareyek A (2001) An empirical analysis of weight-adaptation strategies for neighborhoods of heuristics. In: Metaheuristic International Conference MIC'2001, Porto, Portugal, pp 211–215

Nareyek A (2003) Choosing search heuristics by non-stationary reinforcement learning. In: Resende MGC, de Sousa JP (eds) Metaheuristics: Computer Decision-Making, Kluwer, chap 9, pp 523–544

Norenkov I, Goodman E (1997) Solving scheduling problems via evolutionary methods for rule sequence optimization. 2nd World Conference on soft Computing, WSC2

Ochoa G, Qu R, Burke EK (2009a) Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2009), Montreal, Canada, submitted

Ochoa G, Váquez-Rodríguez JA, Petrovic S, Burke EK (2009b) Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009), Montreal, Norway, accepted

Oltean M (2005) Evolving evolutionary algorithms using linear genetic programming. Evolutionary Computation 13(3):387–410

Oltean M, Dumitrescu D (2004) Evolving TSP heuristics using multi expression programming. In: Bubak M, van Albada GD, Sloot PMA, Dongarra J (eds) LNCS 3037. Proceedings of the 4th International Conference on Computational Science (ICCS'04), Krakow, Poland, pp 670–673

Oltean M, Grosan C (2003) Evolving evolutionary algorithms using multi expression programming. In: Banzhaf W, Christaller T, Dittrich P, Kim JT, Ziegler J (eds) LNAI 2801. Proceedings of the 7th European Conference on Artificial Life, Dortmund, Germany, pp 651–658

Ong YS, Keane AJ (2004) Meta-lamarckian learning in memetic algorithms. IEEE Transactions on Evolutionry Computation 8:99–110

Ong YS, Lim MH, Zhu N, Wong KW (2006) Classification of adaptive memetic algorithms: a comparative study. IEEE Transactions on Systems, Man, and Cybernetics, Part B 36(1):141–152

Ozcan E, Bilgin B, Korkmaz EE (2006) Hill climbers and mutational heuristics in hyperheuristics. In: Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), Reykjavik, Iceland, Lecture Notes in Computer Science, vol 4193, pp 202–211

Ozcan E, Bilgin B, Korkmaz EE (2008) A comprehensive survey of hyperheuristics. Intelligent Data Analysis 12(1):1–21

Ozcan E, Bykov Y, Birben M, Burke EK (2009) Examination timetabling using late acceptance hyper-heuristics. In: Proceedings of Congress on Evolutionary Computation (CEC 2009), p to appear

Pelikan M, Goldberg DE, Lobo FG (2002) A survey of optimization by building and using probabilistic models. Comput Optim Appl 21(1):5–20

Pillay N (2008) An analysis of representations for hyper-heuristics for the uncapacitated examination timetabling problem in a genetic programming system. In: Proceedings of the 2008 Annual Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries, SAICSIT Conf. 2008, Wilderness, South Africa, October 6-8, 2008, ACM, ACM International Conference Proceeding Series, vol 338, pp 188–192

Pillay N, Banzhaf W (2007) A genetic programming approach to the generation of hyper-heuristics for the uncapacitated examination timetabling problem. In: Neves J, Santos MF, Machado J (eds) Progress in Artificial Intelligence, 13th Portuguese Conference on Aritficial Intelligence, EPIA 2007 Proceedings, Springer, Lecture Notes in Computer Science, vol 4874, pp 223–234

Pisinger D, Ropke S (2007) A general heuristic for vehicle routing problems. Computers and Operations Research 34:2403– 2435

Poli R, Woodward JR, Burke EK (2007) A histogram-matching approach to the evolution of bin-packing strategies. In: Proceedings of the Congress on Evolutionary Computation (CEC 2007), Singapore, pp 3500–3507

Qu R, Burke EK (2008) Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. Journal of the Operational Research Society t.a.:t.a, URL http://www.cs.nott.ac.uk/ rxq/files/JORS08.pdf, to appear, doi: 10.1057/jors.2008.102

Rattadilok P, Gaw A, Kwan RSK (2005) Distributed choice function hyper-heuristics for timetabling and scheduling. In: Burke EK, Trick M (eds) The Practice and Theory of Automated Timetabling V: Selected Papers from the 5th International Conference on the Practice and Theory of Automated Timetabling, Lecture Notes in Computer Science Series, vol 3616, Springer, pp 51–70

Rechenberg I (1973) Evolution strategy: Optimization of technical systems by means of biological evolution. Fromman-Holzboog, Stuttgart

Ridge E, Kudenko D (2007) Analyzing heuristic performance with response surface models: prediction, optimization and robustness. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, ACM Press, vol 1, pp 150–157

Ross P (2005) Hyper-heuristics. In: Burke EK, Kendall G (eds) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Springer, chap 17, pp 529–556

Ross P, Marín-Blázquez JG (2005) Constructive hyper-heuristics in class timetabling. In: IEEE Congress on Evolutionary Computation, IEEE, pp 1493–1500

Ross P, Schulenburg S, Marín-Bl ázquez JG, Hart E (2002) Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In: Proceedings of the Genetic and Evolutionary Computation COnference, GECCO'02, Morgan-Kauffman

Ross P, Marín-Blazquez JG, Schulenburg S, Hart E (2003) Learning a procedure that can solve hard bin-packing problems: A new ga-based approach to hyper-heuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2003, Springer, pp 1295–1306

Ross P, Marín-Blazquez JG, Hart E (2004) Hyper-heuristics applied to class and exam timetabling problems. In: Proceedings of the 2004 IEEE Congress on Evolutionary Computation, IEEE Press, Portland, Oregon, pp 1691–1698

Schmiedle F, Drechsler N, Große D, Drechsler R (2002) Heuristic learning based on genetic programming. Genetic Programming and Evolvable Machines 4:363–388

Shaw P (1998) 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), Springer, Lecture Notes in Computer Science, vol 1520, pp 417–431

Smith JE (2003) Co-evolving memetic algorithms: A learning approach to robust scalable optimisation. In: IEEE Congress on Evolutionary Computation, IEEE Press, pp 498–505

Smith JE (2007) Co-evolving memetic algorithms: A review and progress report. IEEE Transactions in Systems, Man and Cybernetics, part B 37(1):6–17

Stephenson M, OReilly U, Martin M, Amarasinghe S (2003) Meta optimization: Improving compiler heuristics with machine learning. In: Proceedings of the Conference on Programming Language Design and Implementation (SIGPLAN03), San Diego, CA, USA, pp 77–90

Storer RH, Wu SD, Vaccari R (1992) New search spaces for sequencing problems with application to job shop scheduling. Management Science 38(10):1495–1509

Storer RH, Wu SD, Vaccari R (1995) Problem and heuristic space search strategies for job shop scheduling. ORSA Journal of Computing 7(4):453–467

Sutton RS, Barto AG (1998) Reinforcement Learning: An Introduction. MIT Press

Tavares J, Machado P, Cardoso A, Pereira FB, Costa E (2004) On the evolution of evolutionary algorithms. In: Keijzer M, O'Reilly UM, Lucas SM, Costa E, Soule T (eds) LNCS 3003. Proceedings of the European Conference on Genetic Programming (EUROGP'04), Coimbra, Portugal, pp 389–398

Tay JC, Ho NB (2008) Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. Computers and Industrial Engineering 54(3):453–473

Terashima-Marín H, Ross P, Valenzuela-Rendón M (1999) Evolution of constraint satisfaction strategies in examination timetabling. In: Genetic and Evolutionary Computation COnference, GECCO'99, pp 635–642

Terashima-Marín H, Zarate CJF, Ross P, Valenzuela-Rendon M (2006) A ga-based method to produce generalized hyper-heuristics for the 2d-regular cutting stock problem. In: GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation, ACM Press, New York, NY, USA, pp 591–598

Terashima-Marín H, Zárate CJF, Ross P, Valenzuela-Rendón M (2007) Comparing two models to generate hyper-heuristics for the 2d-regular bin-packing problem. In: Lipson H (ed) Genetic and Evolutionary Computation Conference, GECCO 2007, ACM, pp 2182–2189, URL http://doi.acm.org/10.1145/1276958.1277377

Terashima-Marín H, Ortiz-Bayliss JC, Ross P, Valenzuela-Rendón M (2008a) Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In: Ryan C, Keijzer M (eds) Genetic and Evolutionary Computation Conference, GECCO 2008, ACM, pp 571–578, URL http://doi.acm.org/10.1145/1389095.1389206

Terashima-Marín H, Ross P, Farias-Zarate CJ, Lopez-Camacho E, Valenzuela-Rendon M (2008b) Generalized hyper-heuristics for solving regular and irregular bin packing problems. Annals of Operations Research 1(1):1–10

Thompson J, Dowsland K (1996) General cooling schedules for a simulated annealing based timetabling system. In: E K Burke PR (ed) Practice and Theory of Automated Timetabling, Springer, Lecture Notes in Computer Science, vol 1153, pp 345–363

Váquez-Rodríguez JA, Petrovic S, Salhi A (2007a) 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: Baptiste P, Kendall G, Munier A, Sourd F (eds) Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007)

Váquez-Rodríguez JA, Petrovic S, Salhi A (2007b) An investigation of hyper-heuristic search spaces. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2007), pp 3776–3783, DOI 10.1109/CEC.2007.4424962, URL http://www.asap.cs.nott.ac.uk/publications/pdf/CEC2007.pdf

Voudouris C, Tsang E (1999) Guided local search and its application to the traveling salesman problem. European Journal of Operational Research 113:469–499

Wah BW, Ieumwananonthachai A (1999) Teacher: A genetics-based system for learning and for generalizing heuristics. In: Yao X (ed) Evolutionary Computation, World

Scientific Publishing Co. Pte. Ltd., pp 124–170

Wah BW, Ieumwananonthachai A, Chu LC, Aizawa A (1995) Genetics-based learning of new heuristics: Rational scheduling of experiments and generalization. IEEE Trans on Knowledge and Data Engineering 7(5):763–785

Wilson S (1995) Classifier systems based on accuracy. Evolutionary Computation 3(2):149–175