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

Computer Science Technical Report No. NOTTCS-TR-SUB-0906241359-0664

A Classification of Hyper-heuristic Approaches

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

First released: February 2009

Technical Report

A Classification of Hyper-heuristic Approaches

Edmund K. Burke · Matthew Hyde · Graham Kendall · Gabriela Ochoa · Ender Ozcan · John Woodward

Abstract Hyper-heuristics comprise a set of approaches that share 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 heuristic design process 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'. In this paper we overview previous categorisations of hyper-heuristics and propose a unified classification. Our goal is to both clarify the main features of existing approaches and to suggest new directions for hyper-heuristic research.

1 Introduction

Hyper-heuristics comprise a set of approaches that share the common goal of automating the design and tuning of heuristic methods to solve hard computational search problems. The motivation behind these approaches is to raise the level of generality at which search methodologies can operate. The term hyper-heuristic was introduced in the early 2000's [18] to describe 'heuristic to choose heuristics' that is, high-level approaches that given a particular problem instance and a number of low-level heuristics, select and apply an appropriate low-level heuristic at each decision point [6]. The idea of automating the heuristic design process, however, is not new. Indeed it can be traced back to the early 1960s [20,27,28], and was independently developed by a number of authors during the 1990s [25,33,34,43,53,57]. Some historical notes, and a brief overview of early approaches can be found in [6] and [48], respectively. 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 [7].

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, UK E-mail: {ekb,mvh,gxk,gxo,exo,jrw}@cs.nott.ac.uk

A variety of hyper-heuristic approaches using different high level techniques, low-level heuristics, and applied to different combinatorial problems, have been proposed in the literature. In this paper, we build upon previous categorisations of hyper-heuristics and provide a unified classification. Our goal is to both clarify the main features of existing approaches and to suggest new directions for hyper-heuristic research.

The following section surveys previous classification of hyper-heuristic approaches. Thereafter, section 3 proposes a unified classification. Sections 4 and 5, describe in more detail the main categories of the proposed classification, giving references to a number of representative examples in the literature. Finally, section 6 summarises our categorisation of approaches and points toward future research directions in hyper-heuristics.

2 Previous Classifications

In [52] hyper-heuristics are categorised into two types: (i) with learning, and (ii) without learning. Hyper-heuristics without learning included approaches that use several heuristics (neighborhood structures), but select the heuristics to call according to a predetermined sequence. Therefore, this category contains approaches such as variable neighbourhood search [42]. The hyper-heuristics with learning include methods that dynamically change the preference of each heuristic based on their historical performance guided by some learning mechanisms. As discussed in [52] hyper-heuristics can be further classified with respect to the learning mechanism employed, and a distinction is made between approaches which use a genetic algorithm during the learning process, from those which use other mechanisms. This is because many hyper-heuristics so far have been based on genetic algorithms. In these genetic-algorithm based hyper-heuristics the idea is to evolve the solution methods, not the solutions themselves.

In [4] hyper-heuristics are classified into those which are *constructive* and those which represent *local search* methods. This distinction is also mentioned by Ross [48]. Constructive hyper-heuristics build a solution incrementally by adaptively selecting heuristics, from a pool of constructive heuristics, at different stages of the construction process. Local search hyper-heuristics, on the other hand, start from a complete initial solution and iteratively select, from a set of neighborhood structures, appropriate heuristics to lead the search in a promising direction.

With the incorporation of genetic programming into hyper-heuristics research in the mid and late 2000's (see [7] for an overview), a new class of hyper-heuristics emerged. This new class of approaches was explicitly and independently mentioned in [1] and [10]. In the first class of heuristics, or 'heuristics to choose heuristics', the framework is provided with a set of pre-existing, generally widely known heuristics for solving the target problem. In contrast, in the second class or 'heuristics to generate heuristics' the aim is to generate new heuristics from a set of *building-blocks* or *components* of known heuristics, which are given to the framework. Therefore, the process requires, as in the first class of hyper-heuristics, the selection of a suitable set of heuristics known to be useful in solving the target problem. But, instead of supplying these directly to the framework, the heuristics are first decomposed into their basic components. Genetic programming hyper-heuristic researchers [1, 7, 10] have also made the distinction between 'disposable' and 'reusable' heuristics. A disposable heuristic is created just for one problem, and is not intended for use on unseen problems. Alternatively the

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.

We consider that the most fundamental hyper-heuristic categories from previous classifications, are those captured by the phrases: (i) **'heuristics to choose heuristics'** and (ii) **'heuristics to generate heuristics'**. Therefore, they form the first branch in our first dimension (the nature of the search space). The second ramification level in this dimension corresponds to the distinction between constructive and local search hyper-heuristics, also discussed in section 2. Notice that this categorisation is concerned with the nature of the low-level heuristics used in the hyper-heuristic framework. Our classification uses the terms *constructive* and *perturbative* to refer to these classes of low-level heuristics. Sections 4 and 5 describe in more detail these categories, listing some concrete examples of recent approaches in the literature.

We consider a hyper-heuristic a learning algorithm when it uses some feedback from the search process. Therefore, non-learning hyper-heuristics are those that do not

use any feedback. According to the source of feedback during learning, we propose a distinction between *online* and *offline* learning. Notice that in the context of heuristics to generate heuristics, or genetic programming hyper-heuristics (as discussed in section 2), the notions of *disposable* and *reusable* have been used to refer to analogous ideas to those of online and offline learning, as described below:

Online learning hyper-heuristics: The learning takes place while the algorithm is solving an instance of a problem. Therefore, task-dependent local properties can be used by the high-level strategy to determine the appropriate low-level heuristic to apply. Examples of on-line learning approaches within hyper-heuristics are: the use of reinforcement for heuristic selection, and generally the use of meta-heuristics as high-level search strategies in a search space of heuristics.

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

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

- With respect to the nature of the heuristic search space
  - **Heuristic to choose heuristics**: produce combinations of pre-existing:
    - Constructive heuristics
    - Perturbative heuristics
  - **Heuristics to generate heuristics**: Generate new heuristic methods using basic components (building-blocks) of:
    - Constructive heuristics
    - Perturbative heuristics
- With respect to source of feedback during learning
  - **Online learning hyper-heuristics**: learn while solving a given instance of a problem.
  - **Offline learning hyper-heuristics**: learn from a set of training instances a heuristic method that would generalise to unseen instances.
  - **Hyper-heuristics without learning**: do not use feedback from the search process.

4 Heuristics to Choose Heuristics

4.1 Approaches based on constructive low-level 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.

Several approaches have been recently proposed to generate efficient hybridisations of existing constructive heuristics in domains such as bin-packing [41,51], timetabling [14,15,49,50], production scheduling [58], and cutting stock [55,56]. Both online and offline machine learning approaches have been investigated. Examples of online approaches are the use meta-heuristics in a search space of constructive heuristics, specifically, genetic algorithms [26,34,57,58] and tabu search [15]. Examples of offline techniques are the use of learning classifier systems [40,41,51], messy genetic algorithms [49,50,56] and case based reasoning [14].

4.2 Approaches based on perturbative low-level heuristics

These approaches start with a complete solution, generated either randomly or using simple constructive heuristics, and thereafter try to iteratively improve on the current solution. The hyper-heuristic framework is provided with a set of neighborhood structures and/or simple local searchers, and the goal is to iteratively select and apply them to the current complete solution. This process will continue until a stopping condition has been met. Notice that these approaches differ from those based on constructive heuristics, in that they do not have a natural termination condition. The sequence of heuristic choices can, in principle, be arbitrarily extended. This class of hyper-heuristics has the potential to be applied successfully to different combinatorial optimisation problems, since general neighbourhood structures or simple local searchers can be made available. Perturbative or improvement hyper-heuristics have been applied to personnel scheduling [12,18], timetabling [5,12], shelf space allocation [2,3], packing [23] and vehicle routing problems [47].

So far, the approaches that combine perturbative low-level heuristics in a hyper-heuristic framework are online, in that they attempt to adaptively solve a single instance of the problem under consideration. Furthermore, the majority of the proposed approaches are single-point algorithms in that they keep a single incumbent solution in the solution space.[1] As suggested in [5,46] their working can be separated into two processes: (i) (low-level) heuristic selection, and (ii) move acceptance strategy. In [46], hyper-heuristics are classified with respect to the nature of the heuristic selection and move acceptance components. We will adapt a similar classification for the hyper-heuristics choosing from a set of perturbative low level heuristics and extend the classification in Figure 1. The heuristic selection can be done in a *non-adaptive* (simple) way: either randomly or cyclically following a prefixed heuristic ordering [18, 19]. In these approaches no learning is involved. Alternatively, the heuristic selection may incorporate an *adaptive* (or on-line learning) mechanism based on probabilistic weighting of the low-level heuristics [13,44], or some type of performance statistics [18,19]. Both non-adaptive and adaptive heuristic selection schemes, are generally embedded within a single-point local search high-level heuristic. The acceptance strategy is an important component of any local search heuristic, therefore, many acceptance strategies have been explored within hyper-heuristics. Move acceptance strategies can

[1] Some approaches have been attempted that maintain a population of points in the heuristic space [17].

be divided into two categories: *deterministic* and *non-deterministic*. In general, a move is accepted or rejected considering the quality of the move and the current solution during a single point search. At any point of the search, deterministic move acceptance methods generate the same result for the same candidate solutions(s) used for acceptance test, where as, a different outcome is possible if a non-deterministic approach is used. If the move acceptance test involves other parameters, such as the current time then these strategies are referred to as non-deterministic strategies. Well known meta-heuristic components are commonly used as non-determistic acceptance methods, such as great deluge [37] and simulated annealing [3, 24].

5 Heuristics to Generate Heuristics

This section describes approaches that have the potential to automatically generate heuristics for the problem at hand. As we discussed earlier (sections 2 and 3), they are heuristics to *generate* heuristics, as opposed to heuristics to *choose* heuristics. Approaches to generate heuristics use genetic programming, a branch of evolutionary computation concerned with the automatic generation of computer programs [39]. Besides the particular representation (using trees as chromosomes [2]), it differs from other evolutionary approaches in its application area. While most applications of evolutionary algorithms deal with optimisation problems, genetic programming could instead be positioned in machine learning. Genetic programming has been successfully applied to the automated generation of heuristics that solve hard combinatorial optimisation problems, such as boolean satisfiability, [1, 29–31, 38], bin packing [8, 9, 11], the traveling salesman problem [35, 36] and production scheduling [21, 22, 32, 54].

One approach to use genetic programming as a hyper-heuristic has been to evolve local search [1, 30, 31, 36, 35] heuristics or even evolutionary algorithms [45]. An alternative idea has been to use genetic programming to evolve a a program representing a function, which is part of the processing of a given problem specific constructive heuristic [8, 9, 11, 21, 22, 32, 54]. Most applications of genetic programming as a hyper-heuristic are offline in that a training set is used for generating a program that acts as a heuristic, that would thereafter be tested for solving unseen instances of the underlying problem. That is, the idea is to generate *reusable* heuristics. However, research on *disposable* heuristics has also been conducted [1, 35, 36]. In other words, heuristics are evolved for solving a single instance of a problem. This approach is analogous to the 'heuristic to choose heuristics' online approaches discussed in section 4, except that a new heuristic is generated for each instance, instead of choosing a sequence of heuristics from a predefined set.

6 Summary and Discussion

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

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

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. We have further categorised the two main classes of hyper-heuristics, according to whether they are based on *constructive* or *perturbative* low-level heuristics. We also considered an additional orthogonal criterium for classifying hyper-heuristics with respect to the source providing feedback during the learning process, which can be either one instance (online approaches) or many instances of the underlying problem studied (offline approaches). Both online and offline approaches are potentially useful and therefore worth investigating. Although having a reusable method will increase the speed of solving new instances of problems, using online (or disposable) methods can have other advantages. In particular, searching 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.

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

References

1. M. Bader-El-Den and R. Poli. Generating sat local-search heuristics using a gp hyper-heuristic framework. In *Proceedings of Artificial Evolution (EA'07)*, 2007.
2. R. Bai, E. K. Burke, and G. Kendall. Heuristic,meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *Journal of the Operational Research Society*, 59:1387 – 1397, 2008.
3. R. Bai and G. Kendall. An investigation of automated planograms using a simulated annealing based hyper-heuristics. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solver - (Operations Research/Computer Science Interface Serices, Vol.32)*, pages 87–108. Springer, 2005.
4. Ruibin Bai. *An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, September 2005.
5. B. Bilgin, E. Ozcan, and E. E. Korkmaz. 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)*, pages 123–140, 2006.
6. E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 457–474. Kluwer, 2003.
7. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In C. Mumford and L. Jain, editors, *Collaborative Computational Intelligence*. Springer, 2009.

8. E. K. Burke, M. Hyde, G. Kendall, and J. Woodward. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2007)*, London, UK, 2007. to appear.
9. E. K. Burke, M. R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, volume 4193 of *Lecture Notes in Computer Science*, pages 860–869, Reykjavik, Iceland, September 2006.
10. E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. Technical report, University of Nottingham, Dept of Computer Science, 2008.
11. E. K. Burke, M. R. Hyde, G. Kendall, and J. R. Woodward. The scalability of evolved on line bin packing heuristics. In *2007 IEEE Congress on Evolutionary Computation*, pages 2530–2537, Singapore, 2007. IEEE Computational Intelligence Society, IEEE Press.
12. E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.
13. E. K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics*, 9(6):451–470, 2003.
14. E. K. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.
15. E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research*, 176:177–192, 2007.
16. Konstantin Chakhlevitch and Peter I. Cowling. Hyperheuristics: Recent developments. In Carlos Cotta, Marc Sevaux, and Kenneth Sörensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer, 2008.
17. P. Cowling, G. Kendall, and L. Han. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In *Proceedings of the Congress on Evolutionary Computation (CEC2002)*, pages 1185–1190, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, USA, May 12-17 2002.
18. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach for scheduling a sales summit. In *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, Lecture Notes in Computer Science, pages 176–190, Konstanz, Germany, August 2000. Springer.
19. P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In S. Cagoni, J. Gottlieb, Emma Hart, M. Middendorf, and R. Goenther, editors, *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10, Kinsale, Ireland, April 3-4 2002. Springer-Verlag.
20. W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh*, -(117), 1963.
21. C. Dimopoulos and A. M. S. Zalzala. Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, 32(6):489–498, 2001.
22. C. Dimopoulos and A. MS Zalzala. A genetic programming heuristic for the one-machine total tardiness problem. In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC 99)*, pages 2207–2214, 1999.
23. K. A. Dowsland, E. Soubeiga, and E. K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.
24. K. A. Dowsland, E. Soubeiga, and E. K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.
25. H.L Fang, P. Ross, and D. Corne. A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. In S. Forrest, editor, *Fifth International Conference on Genetic Algorithms*, pages 375–382, San Mateo, 1993. Morgan Kaufmann.
26. H.L Fang, P. Ross, and D. Corne. A promising hybrid ga/ heuristic approach for open-shop scheduling problems. In A. Cohn, editor, *Eleventh European Conference on Artificial Intelligence*. John Wiley & Sons, 1994.

27. H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In *Factory Scheduling Conference*, Carnegie Institue of Technology, May 10-12 1961.
28. H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251, New Jersey, 1963. Prentice-Hall, Inc.
29. A. Fukunaga. Automated discovery of composite SAT variable selection heuristics. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 641–648, 2002.
30. A. S. Fukunaga. Evolving local search heuristics for SAT using genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, Lecture Notes in Computer Science, pages 483–494. Springer-Verlag, 2004.
31. A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evol. Comput.*, 16(1):31–61, 2008.
32. C. D. Geiger, R. Uzsoy, and H.Aytŭg. Rapid modeling and discovery of priority dispatching rules: An autonomous learning approach. *Journal of Scheduling*, 9:7–34, 2006.
33. J. Gratch and S. Chien. Adaptive problem-solving for large-scale scheduling problems: a case study. *Journal of Artificial Intelligence Research*, 4:365–396, 1996.
34. E. Hart, P. Ross, and J. A. D. Nelson. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computing*, 6(1):61–80, 1998.
35. R. E. Keller and R. Poli. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Proceedings of Artificial Evolution (EA'07)*, pages 13–24, 2007.
36. R. E. Keller and R. Poli. Linear genetic programming of parsimonious metaheuristics. In *Proceedings of Congress on Evolutionary Computation (CEC 2007)*, 2007.
37. G. Kendall and M. Mohamad. Channel assignment in cellular communication using a great deluge hyper-heuristic. In *Proceedings of the 2004 IEEE International Conference on Network (ICON2004)*, pages 769–773, Singapore, 16-19 November 2004.
38. R. H. Kibria and Y. Li. Optimizing the initialization of dynamic decision heuristics in DPLL SAT solvers using genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pages 331–340, 2006.
39. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
40. J. G. Marín-Blázquez and S. Schulenburg. A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *IWLCS*, volume 4399 of *Lecture Notes in Computer Science*, pages 193–218. Springer, 2005.
41. Javier G. Marín-Blázquez and Sonia Schulenburg. A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *IWLCS*, volume 4399 of *Lecture Notes in Computer Science*, pages 193–218. Springer, 2005.
42. N. Mladenovic and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997.
43. J. Mockus and L. Mockus. Bayesian approach to global optimization and applications to multi-objective constrained problems. *Journal of Optimization Theory and Applications*, 70(1):155–171, July 1991.
44. A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende and J. P. de Sousa, editors, *Metaheuristics: Computer Decision-Making*, chapter 9, pages 523–544. Kluwer, 2003.
45. M. Oltean. Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, 13(3):387–410, 2005.
46. E. Ozcan, B. Bilgin, and E. E. Korkmaz. Hill climbers and mutational heuristics in hyperheuristics. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, volume 4193 of *Lecture Notes in Computer Science*, pages 202–211, Reykjavik, Iceland, September 2006.
47. D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403– 2435, 2007.
48. P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. Springer, 2005.

49. P. Ross and J. G. Marín-Blázquez. Constructive hyper-heuristics in class timetabling. In *IEEE Congress on Evolutionary Computation*, pages 1493–1500. IEEE, 2005.
50. P. Ross, J. G. Marin-Blazquez, and E. Hart. Hyper-heuristics applied to class and exam timetabling problems. In *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pages 1691–1698, Portland, Oregon, 2004. IEEE Press.
51. P. Ross, S. Schulenburg, J. G. Marin-Blázquez, and E. Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In *Proceedings of the Genetic and Evolutionary Computation COnference, GECCO'02*. Morgan-Kauffman, 2002.
52. Eric Soubeiga. *Development and Application of Hyperheuristics to Personnel Scheduling*. PhD thesis, School of Computer Science and Information Technology, University of Nottingham, June 2003.
53. R. H. Storer, S. D. Wu, and R. Vaccari. Problem and heuristic space search strategies for job shop scheduling. *ORSA Journal of Computing*, 7(4):453–467, 1995.
54. J. C. Tay and N. B. Ho. Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Computers & Industrial Engineering*, 54:453–473, 2008.
55. H. Terashima-Marín, E. J. Flores-Álvarez, and P. Ross. Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems. In Hans-Georg Beyer and Una-May O'Reilly, editors, *Genetic and Evolutionary Computation Conference, GECCO 2005, Proceedings, Washington DC, USA, June 25-29, 2005*, pages 637–643. ACM, 2005.
56. H. Terashima-Marin, A. Moran-Saavedra, and P. Ross. Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, volume 2, pages 1104–1110, Edinburgh, Scotland, UK, 2005. IEEE Press.
57. H. Terashima-Marin, P. Ross, and M. Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In *Genetic and Evolutionary Computation COnference, GECCO'99*, pages 635–642, 1999.
58. J. A. Vzquez-Rodrguez, S. Petrovic, and A. Salhi. A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In Philippe Baptiste, Graham Kendall, Alix Munier, and Francis Sourd, editors, *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007)*, 2007.