HyFlex: A Benchmark Framework for Cross-domain Heuristic Search

Gabriela Ochoa[1], Matthew Hyde[1], Tim Curtois[1], Jose A. Vazquez-Rodriguez[1], James Walker[1], Michel Gendreau[2], Graham Kendall[1], Barry McCollum[3], Andrew J. Parkes[1], ' Sanja Petrovic[1], and Edmund K. Burke[4]

[1] School of Computer Science, University of Nottingham, UK
[2] CIRRELT, University of Montreal, Canada
[3] School of Electronics and Computer Science, Queen's University, UK,
[4] Department of Computing Science and Mathematics, University of Stirling, UK

Abstract. This paper presents *HyFlex*, a software framework for the development of cross-domain search methodologies. The framework features a common software interface for dealing with different combinatorial optimisation problems and provides the algorithm components that are problem specific. In this way, the algorithm designer does not require a detailed knowledge of the problem domains and thus can concentrate his/her efforts on designing adaptive general-purpose optimisation algorithms. Six hard combinatorial problems are fully implemented: maximum satisfiability, one dimensional bin packing, permutation flow shop, personnel scheduling, traveling salesman and vehicle routing. Each domain contains a varied set of instances, including real-world industrial data and an extensive set of state-of-the-art problem specific heuristics and search operators. HyFlex represents a valuable new benchmark of heuristic search generality, with which adaptive cross-domain algorithms are being easily developed and reliably compared.This article serves both as a tutorial and a as survey of the research achievements and publications so far using HyFlex.

1 Introduction

There is a renewed and growing research interest in techniques for automating the design of heuristic search methods. The goal is to reduce the need for a human expert in the process of designing an effective algorithm to solve a search problem and consequently raise the level of generality at which search methodologies can operate. Evolutionary algorithms and metaheuristics have been successfully applied to solve a variety of real-world complex optimisation problems. Their design, however, has become increasingly complex. In order to make these methodologies widely applicable, it is important to provide self-managed systems that can configure themselves 'on the fly'; adapting to the changing problem (or search space) conditions, based on general high-level guidelines provided by their users.

Researchers pursuing these goals within combinatorial optimisation, are often limited by the number of problem domains available to them for testing their

adaptive methodologies. This can be explained by the difficulty and effort required to implement state-of-the-art software components, such as the problem model, solution representation, objective function evaluation and search operators; for many different combinatorial optimisation problems. Although several benchmark problems in combinatorial optimisation are available [24, 1, 3] (to name just a few); they contain mainly the data of a set of instances and their best known solutions. They generally do not incorporate the software necessary to encode the solutions and calculate the objective function, let alone existing search operators for the given problem. It is the researcher who needs to provide these in order to later test their high-level adaptive search method. To overcome such limitations, we propose *HyFlex*, a modular and flexible Java class library for designing and testing iterative heuristic search algorithms. It provides a number of problem domain modules, each of which encapsulates the problem-specific algorithm components. Importantly, only the high-level control strategy needs to be implemented by the user, as HyFlex provides an easy-to-use interface with which the problem domain modules can be linked.

A number of research themes within operational research, computer science and artificial intelligence would benefit (and are already benefiting) from the proposed framework. Among them: hyper-heuristics [8, 22], adaptive memetic algorithms [19, 23], adaptive operator selection [13], reactive search [2], variable neighborhood search and its adaptive variants [21]; and generally the development of adaptive parameter control strategies in evolutionary algorithms [12]. HyFlex can be seen as a unifying and extended benchmark for combinatorial optimisation, with which the performance of different cross-domain adaptive techniques can be reliably assessed and compared. Practitioners can also gain even if they are only interested in one specific domain, because they could have available a large range of hyper-heuristics. A simple test process could determine the hyper-heuristic that best exploits the underlying domain and so allows practitioners to quickly and easily obtain their results without having to implement a complex search control process themselves.

HyFlex was used to support an international research competition: the first Cross-Domain Heuristic Search Challenge [18]. The challenge is analogous to the athletics Decathlon event, where the goal is not to excel in one event at the expense of others, but to have a good general performance on each. Competitors submitted one Java class file using HyFlex representing their hyper-heuristic or high-level search strategy. This ensures that the competition is fair, because all of the competitors must use the same problem representation and search operators. Moreover, due to the common interface, the competition considered not only hidden instances, but also two hidden domains.

The purpose of this article is to describe the HyFlex framework as a benchmark tool for research in hyper-heuristics and adaptive/autonomous heuristic search. A detailed analysis of the 2011 CHeSC competition is beyond the scope of this article and will be discussed elsewhere. The next section describes the antecedents and architecture of the HyFlex framework. It also includes a detailed account of how to implement and run hyper-heuristics within the framework

and a brief summary of the problem domains currently implemented. Section 3 presents a survey of published research and achievements made possible by the framework so far. Finally, section 4 summarises our contribution and suggests directions for future research.

2 The HyFlex Framework

HyFlex (Hyper-heuristics Flexible framework) is a software framework designed to enable the development, testing and comparison of iterative general-purpose heuristic search algorithms (such as hyper-heuristics). To achieve these goals it uses modularity and the concept of decomposing a heuristic search algorithm into two main parts:

1. A general-purpose part: the algorithm or hyper-heuristic.
2. The problem-specific part: provided by the HyFlex framework.

In the hyper-heuristics literature, this idea is also referred to as the domain barrier between the problem-specific heuristics and the hyper-heuristic [7, 10]. HyFlex extends the conceptual domain-barrier framework by maintaining a population (instead of a single incumbent solution) in the problem domain layer. Moreover, it provides a richer variety of problem specific heuristics and search operators (i.e. it includes crossover and 'ruin-recreate' heuristics). Another relevant antecedent to HyFlex is PISA [4], a text-based software interface for multi-objective evolutionary algorithms. PISA provides a division between the application-specific and the algorithm-specific parts of a multi-objective evolutionary algorithm. In HyFlex, the interface is given by an abstract Java class. This allows a more tight coupling between the modules and overcomes some of the speed limitations encountered in PISA. Moreover, HyFlex provides a richer variety of fully implemented combinatorial optimisation problems including real-world instance data.

The framework is written in Java which is familiar to and commonly used by many researchers. It also benefits from object orientation, platform independence and automatic memory management. At the highest level, the framework consists of just two abstract classes: ProblemDomain and HyperHeuristic. The structure of these classes is shown in the class diagram of figure 1. In the diagram, the signatures adjacent to circles are public methods and fields and the signatures adjacent to diamonds are protected. Abstract methods are denoted by italics and the implementations of these methods are necessarily different for each instantiation of problem domain or hyper-heuristic.

2.1 The `ProblemDomain` Class

As shown in figure 1, an implementation of the `ProblemDomain` class provides the following elements, each of which is easily accessed and managed with one or more methods.

1. A user-configurable memory (a population) of solutions, which can be managed by the hyper-heuristic through methods such as `setMemorySize` and `copySolution`.
2. A routine to randomly initialise solutions, `initialiseSolution(`i`)`, where i is the index of the solution array in the memory.
3. A set of problem specific heuristics, which are used to modify solutions. These are called with the `applyHeuristic(`i, j, k`)` method, where i is the index of the heuristic to call, j is the index of the solution in memory to modify and k is the index in memory where the resulting solution should be placed. Solution j is not changed by this operation. Each problem-specific heuristic in each problem domain is classified into one of four groups, shown below. The heuristics belonging to a specific group can be accessed by calling `getHeuristicsOfType(`*type*`)`.
   - Mutational or perturbation heuristics: perform a small change on the solution, by swapping, changing, removing, adding or deleting solution components.
   - Ruin-recreate (destruction-construction) heuristics: partly destroy the solution and rebuild or recreate it afterwards. These heuristics can be considered as large neighbourhood structures. They are, however, different from the mutational heuristics in that they can incorporate problem specific construction heuristics to rebuild the solutions
   - Hill-climbing or local search heuristics: iteratively make small changes to the solution, only accepting non-deteriorating solutions, until a local optimum is found or a stopping condition is met. These heuristics differ from mutational heuristics in that they incorporate an iterative improvement process and they guarantee that a non-deteriorating solution will be produced.
   - Crossover heuristics: take two solutions, combine them and return a new solution.
4. A varied set of instances that can be easily loaded using the method: `loadInstance(`a`)`, where a is the index of the instance to be loaded.
5. A fitness function, which can be called with the `getFunctionValue(`i`)` method, where i is the index of the required solution in the memory. HyFlex problem domains are always implemented as minimisation problems, so a lower fitness is always better. The fitness of the best solution found so far in the run can be obtained with the `getBestSolutionValue()` method.
6. Two parameters: α and $\beta, (0 <= [\alpha, \beta] <= 1)$, which are the 'intensity' of mutation and 'depth of search', respectively, that control the behaviour of some search operators.

2.2 The `HyperHeuristic` Class

The HyperHeuristic class is designed to allow algorithms which implement this class to be compared and benchmarked across one or more of the problem domains available (for example, in a competition). Users create cross-domain

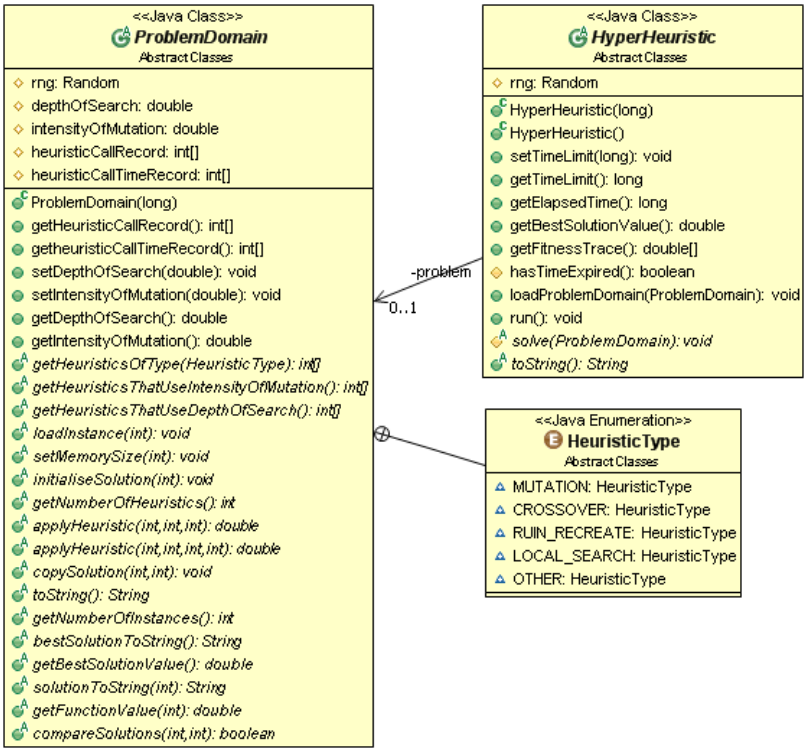

Fig. 1. Class diagram for the HyFlex framework.

heuristic algorithms by creating implementations of this abstract class. Each class must contain a `toString()` method, to give the methodology a name. It must also contain a `solve()` method, in which the functionality of the particular methodology is written.

The `solve()` method would normally contain a loop, which continues while the time limit (defined by the user) has not been exceeded. In the loop, the code should provide a mechanism for selecting between the available problem-specific heuristics and choose to which solutions in memory to apply the heuristics. This class could choose to work with a memory size of 1 for a single point search, or a large memory could be maintained for a population based approach. The memory can be easily defined and maintained through calling methods of the `ProblemDomain` class, where the memory is stored. A hyper-heuristic class automatically records the length of time for which it has been running and this can be monitored through methods such as `hasTimeExpired()` and `getElapsedTime()`.

The solve method is the only method which must be implemented. All other common functionality is provided by the HyFlex software, such as the timing function and the recording of the best solution.

2.3 Running a Hyper-Heuristic

Algorithm 1 shows the ease with which a hyper-heuristic can be run on a problem domain. An object is created for the problem domain (in this example MAX-SAT) and for the hyper-heuristic, each with a random seed. Then a problem instance is loaded from the selection available in the problem domain object. In this example we choose the instance with index 0. The problem domain is now set up for the hyper-heuristic.

We set the time for which the hyper-heuristic will run, in milliseconds. Then the hyper-heuristic object is given a reference to the problem domain object. Now that the setup is complete, the `run()` method of the hyper-heuristic is called to start the search process. The hyper-heuristic will run for 60 seconds in this example and the best solution found during that time is retrievable with the `getBestSolutionValue()` method, as shown in Algorithm 1. This (or indeed any) hyper-heuristic can be run on the 5 other problem domains by changing just one line of code.

Algorithm 1 Java code for running a hyper-heuristic on a problem domain

```
ProblemDomain problem = new SAT(seed1);
HyperHeuristic HHObject = new ExampleHyperHeuristic1(seed2);
problem.loadInstance(0);
HHObject.setTimeLimit(60000);
HHObject.loadProblemDomain(problem);
HHObject.run();
System.out.println(HHObject.getBestSolutionValue());
```

2.4 An Example Hyper-Heuristic

This section provides an example hyper-heuristic, to illustrate the ease with which a hyper-heuristic can be created. This is done by extending the Hyper-Heuristic abstract class and implementing only one method. All of the common functionality is provided by the HyFlex software, such as the timing function and the recording of the best solution. This example demonstrates exactly how to use certain elements of HyFlex functionality, including the solution memory.

After the `run()` method of the hyper-heuristic is called (see section 2.3), the hyper-heuristic abstract class performs some housekeeping tasks, such as initialising the timer and then calls the `solve` method of the chosen hyper-heuristic. In Algorithm 1, this is an object of the class `ExampleHyperHeuristic1`. Algorithm 2 shows the code for the `solve()` method in `ExampleHyperHeuristic1`. It shows that very few lines of code are necessary in order to implement a hyper-heuristic method with the HyFlex framework. Algorithm 2 is written in pseudocode, but each line corresponds to no more than one line of actual Java code. The `solve()` method is the only substantial method which needs to be implemented. Indeed

the only other necessary method is `toString()`, which requires one line to give the hyper-heuristic a name.

From Algorithm 2, we can see that the `solve()` method takes the problem domain object as an argument and checks for the number of search operators available within it. We also initialise a value to store the current objective function value. It is also necessary to initialise at least one solution in the memory. The default memory size is 2 and we initialise the solution at index 0, which means we build an initial solution with the method specified in the problem domain (generally a fast randomised constructive heuristic). The solution at index 1 remains uninitialised and therefore has a value of `null`.

An implemented hyper-heuristic must always contain a while loop which checks if the time limit has expired. The code within the loop specifies the main functionality of the hyper-heuristic. In this example, we choose a random operator and then apply it to the solution at index 0. The modified solution is put in the memory at index 1 (previously not initialised). Note that a random number generator `rng` is provided by the HyperHeuristic abstract class. This is created when the hyper-heuristic object's constructor is called and is the reason why that constructor requires a random seed.

If the new solution is superior to the old solution, it is accepted and the new solution overwrites the old one in memory. The `copySolution` method of the problem domain class is employed to manage this. If the new solution is not superior, then the new solution is accepted with 0.5 probability.

Algorithm 2 Pseudocode for the solve method of `ExampleHyperHeuristic1`. This is called when the `run()` method of the hyper-heuristic is called (see Algorithm 1)

```
Require: A ProblemDomain object, problem
  int numberOfHeuristics = problem.getNumberOfHeuristics
  double currentObjValue = Double.POSITIVE-INFINITY
  problem.initialiseSolution(0)
  while hasTimeExpired = FALSE do
    int h = rng.nextInt(numberOfHeuristics)
    double newObjValue = problem.applyHeuristic(h, 0, 1)
    double delta = currentObjValue - newObjValue
    if delta > 0 then
      problem.copySolution(1, 0)
      currentObjValue = newObjValue;
    else
      if rng.nextBoolean = TRUE then
        problem.copySolution(1, 0)
        currentObjValue = newObjValue;
      end if
    end if
  end while
```

2.5 HyFlex Problem Domains

Currently, six problem domain modules are implemented. From these, 4 were given as test domains for the CHeSC competition: maximum satisfiability (MAX-SAT), one-dimensional bin packing, permutation flow shop and personnel scheduling. Two additional domains, namely, the traveling salesman and the capacitated vehicle routing problem [25], were later implemented and used as hidden domains in the competition. Each domain includes 10 training instances from different sources and a number of problem-specific heuristics of the types discussed in section 2.1. The six domains together with technical reports describing them in detail, including the problem formulation, solution initialisation, instance date and low-level heuristics, can be found on the competition site [18]. Due to space constraints we only present a summary describing the solution initialisation, the total number of low-level heuristics and the number of heuristics of each type (Table 1).

Table 1. HyFlex problem domains, indicating initialisation, the total number of low-level heuristics and the number of heuristics per type.

Domain	*Initialisation*	*Total*	*Mut.*	*R&R*	*Xover.*	*LS.*
Max-SAT	Random bit-string	9	4	1	2	2
Bin Packing	Randomised first-fit heuristic [15]	8	3	2	1	2
Flow Shop	Randomised NEH procedure [17]	15	5	2	3	4
Pers. Sched.	Randomised hill climbing heuristic	12	1	3	3	4
TSP	Random permutation	15	5	1	3	6
VRP	Randomised constructive heuristic	12	4	2	2	4

3 HyFlex Achievements

HyFlex was made available in August 2010 when the CHeSC competition was launched at the International Conference on the Practice and Theory of Automated Timetabling (PATAT 2010)[1]. In May 2011, a web statistics counter was added to the website and since then, up to January 30^{th} 2012, it has recorded 4,721 visits and 9,451 page views. This section briefly surveys the research achievements and publications made possible with HyFlex so far.

The first article implementing hyper-heuristics using HyFlex was published in 2010 [5], where several hyper-heuristics combining two heuristic selection mechanisms and three acceptance criteria were compared. A multiple neighbourhood iterated local search was also implemented and found to outperform the other approaches as a general optimiser. This iterated local search hyper-heuristic contains a perturbation stage, during which a neighborhood move is selected

[1] `http://www.cs.qub.ac.uk/~B.McCollum/patat10/`

uniformly at random (from the available pool of mutation and ruin-recreate heuristics) and applied to the incumbent solution; followed by a greedy improvement stage (using all the local search heuristics). The approach is extended in [6] by substituting the uniform random selection of neighbourhoods in the perturbation stage, by online learning strategies. Two strategies were implemented: *choice function* [10] (from the hyper-heuristics literature) and *extreme value based adaptive operator selection* [13] (from the evolutionary computation literature), with the latter producing better overall results. This last implementation was the best performing hyper-heuristic before the competition started.

In [14], the authors used reinforcement learning for heuristic selection and explored several variants for the rewards, policy and learning functions. Different ways of modeling the agents' states and actions were also explored. The results reported are preliminary and do not compare well with those generated by other HyFlex hyper-heuristics.

In [20], the authors implement a multi-stage hyper-heuristic, combining a greedy stage with a random descent stage, followed by a simple solution acceptance mechanism. During the greedy stage, all the available low-level heuristics are applied during a number of steps and a subset of the best performing heuristics (active set) is constructed using a dominance-based strategy. The subsequent random descent stage, randomly selects a heuristic from the active set and applies it repeatedly until no improvement is achieved. The transition between stages is controlled by a probability parameter. This relatively simple approach produces very good results when compared with previous HyFlex hyper-heuristics.

HyFlex was used to support the CHeSC 2011 competition. The event successfully attracted the interest and participation of universities and academic institutions across the six continents; 43 registrations and 20 submissions were received. We received several positive and encouraging comments regarding both HyFlex and the competition, from the registered participants. The competition results and brief technical reports describing the participant entries can be found in the website [18]. Here, we briefly summarise the top 4 hyper-heuristics:

1. *AdapHH*: Mustafa Misir, University KaHo Sint-Lieven, Belgium.
   A 'traditional' selective hyper-heuristic that includes two stages: heuristic selection and solution acceptance. Heuristic selection is done by learning dynamic heuristic sets and effective pairs of heuristics. The algorithm also incorporates adaptation of the heuristic parameters and an adaptive threshold acceptance. This approach was presented in [16].
2. *VNS-TW*: Ping-Che Hsiao, National Taiwan University, Taiwan.
   A variable neighborhood search algorithm that orders perturbation heuristics according to strength. It includes two stages: diversification and intensification and incorporates an adaptive technique to adjust the strength of the local search heuristics.
3. *ML*: Mathieu Larose, Montreal University,Canada.
   An adaptive iterated local search algorithm with three stages: diversification, intensification and a simple adaptive move acceptance. Reinforcement learning is used for selecting heuristics.

4. *PHUNTER*: Fan Xue, Hong Kong Polytechnic University, Hong Kong.
   A hyper-heuristic that can assemble different iterated local search algorithms. The authors use the metaphor of pearl hunting; there is a diversification stage (surface and change target area) and an intensification stage (dive and find pearl oysters). The algorithm also uses offline learning to identify search modes. This approach was presented in [9].

Several new best-known solutions have been found for personnel scheduling instances [11] using HyFlex (see Table 2). This is an interesting result, considering that HyFlex was designed to implement general-purpose search heuristics.

Table 2. Personnel scheduling best-known solutions obtained by the PHUNTER HyFlex hyper-heuristic.

Instance name	HyFlex best-known	Previous best-known	staff	Shift types	days
CHILD-A2	1095	1111	41	5	42
ERRVH-A	2142	2197	51	8	42
ERRVH-B	3121	6859	51	8	42
MER-A	9017	9915	54	12	42

A special session on Cross-domain Heuristic Search was held as part of the Learning and Intelligent OptimizatioN conference (LION 2012)[2]. Seven papers were accepted and presented using HyFlex for implementing cross-domain hyper-heuristics. HyFlex is also being used as a tool for teaching modules in meta-heuristics and evolutionary algorithms. Finally, HyFlex is potentially useful from the point of view of practitioners. If they provide their problem-specific software components following the interface, they will have at their disposal a growing number of hyper-heuristics and adaptive search controllers ready to use.

4 Discussion and Future Work

HyFlex is a software framework which enables cross-domain algorithms to be easily developed and reliably compared. It currently provides 6 problem domains, each containing a set of problem instances and search operators to apply. Therefore, it represents a novel extension of the notion of benchmark for combinatorial optimisation. Researchers from different communities and themes within computer science, artificial intelligence and operational research, can benefit from HyFlex, as it provides a common benchmark in which to test the performance and behavior of single-point and population-based self-configuring search heuristics. When using HyFlex, researchers can concentrate their efforts on designing their adaptive methodologies, rather than implementing the required set of problem domains. There is currently ample evidence that HyFlex is being used by the research community for both research and teaching.

[2] `http://intelligent-optimization.org/LION6`

Different algorithm design ideas have been implemented and tested using HyFlex. Some successful design principles start to emerge such as the use of diversification and intensification phases, the use of reinforcement learning for heuristic selection, adaptation of the heuristic parameters and the use of adaptive acceptance criteria. Interesting emerging ideas are the use of co-evolution and evolution of heuristic sequences. The use of a population is starting to be valued within selective hyper-heuristic research, which has traditionally focused on single-point search algorithms. It is our vision that the HyFlex framework will continue to facilitate and increase international interest in developing hyper-heuristics and adaptive heuristic search methodologies that can find wider application in practice.

HyFlex can be extended to include new domains, additional instances and operators in existing domains and multi-objective and dynamic problems. The current software interface can also be extended to incorporate additional feedback information from the domains to guide the adaptive search controllers. In particular, parameterised low-level heuristics and diversity metrics can be included. We plan to host a new edition of the competition and maintain a repository of results and updates to the HyFlex framework.

References

1. J. Argelich, C-M. Li, F. Manya, and J. Planes. Maxsat evaluation 2009 benchmark data sets. Website, 2009. `http://www.maxsat.udl.cat/`.
2. R. Battiti, M. Brunato, and F. Mascia. *Reactive Search and Intelligent Optimization*, volume 45 of *Operations Research/Computer Science Interfaces Series*. Springer, 2009.
3. J. E. Beasley. Or-library: collection of test data sets for a variety of operations research (or) problems. Website, 2010. `http://people.brunel.ac.uk/~mastjjb/jeb/info.htmll`.
4. S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 494–508, Berlin, 2003. Springer.
5. E. K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J. A. Vazquez-Rodriguez, and M. Gendreau. Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 3073–3080, Barcelona, Spain, July 2010.
6. E. K. Burke, M. Gendreau, G. Ochoa, and J. D. Walker. Adaptive iterated local search for cross-domain optimisation. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 1987–1994, New York, NY, USA, 2011. ACM.
7. 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.
8. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research*

& Management Science, chapter A Classification of Hyper-heuristic Approaches, pages 449–468. Springer, 2010. Chapter 15.
9. C.Y. Chan, Fan Xue, W.H. Ip, and C.F. Cheung. A hyper-heuristic inspired by pearl hunting. In *International Conference on Learning and Intelligent Optimization (LION 6)*, Lecture Notes in Computer Science. Springer, 2012. (to appear).
10. 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*, LNCS, pages 176–190, Konstanz, Germany, 2001. Springer.
11. T. Curtois. Staff rostering benchmark data sets. Website, 2011. `http:///www.cs.nott.ac.uk/~tec/NRP/`.
12. A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith. *Parameter Setting in Evolutionary Algorithms*, chapter Parameter Control in Evolutionary Algorithms, pages 19–46. Springer, 2007.
13. A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag. Extreme value based adaptive operator selection. In *Parallel Problem Solving from Nature PPSN X*, volume 5199 of *LNCS*, pages 175–184. Springe, 2008.
14. L. Di Gaspero and T. Urli. A reinforcement learning approach for the cross-domain heuristic search challenge. In *Proceedings of the 9th Metaheuristics International Conference (MIC 2011)*, Udine, Italy, July 25–28 2011.
15. D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. Worst-case performance bounds for simple one-dimensional packaging algorithms. *SIAM Journal on Computing*, 3(4):299–325, December 1974.
16. M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. An intelligent hyper-heuristic framework for chesc 2011. In *International Conference on Learning and Intelligent Optimization (LION 6)*, Lecture Notes in Computer Science. Springer, 2012. (to appear).
17. M. Nawaz, E. Enscore Jr., and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA-International Journal of Management Science*, 11(1):91–95, 1983.
18. G. Ochoa and M. Hyde. The Cross-domain Heuristic Search Challenge (CHeSC 2011). Website, 2011. `http://www.asap.cs.nott.ac.uk/chesc2011/`.
19. Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong. Classification of adaptive memetic algorithms: a comparative study. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):141–152, 2006.
20. E. Ozcan and A. Kheiri. A hyper-heuristic based on random gradient, greedy and dominance. In *Proceedings of the 26th International Symposium on Computer and Information Sciences ISCIS2011*, London, UK, July 25–28 2011.
21. D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403– 2435, 2007.
22. 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.
23. J. E. Smith. Co-evolving memetic algorithms: A review and progress report. *IEEE Transactions in Systems, Man and Cybernetics, part B*, 37(1):6–17, 2007.
24. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
25. J. D. Walker, G. Ochoa, M. Gendreau, and E. K. Burke. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In *International Conference on Learning and Intelligent Optimization (LION 6)*, Lecture Notes in Computer Science. Springer, 2012. (to appear).