10-12 August 2009, Dublin, Ireland

MISTA 2009

HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics

Edmund K. Burke · Tim Curtois · Matthew Hyde · Graham Kendall · Gabriela Ochoa · Sanja Petrovic · José Antonio Vázquez-Rodríguez

1 Introduction

Despite the success of heuristic search methods in solving real-world computational search problems, it is often still difficult to easily apply them to new problems, or even new instances of similar problems. These difficulties arise mainly from the significant number of parameter or algorithm choices involved when using these type of approaches, and the lack of guidance as to how to proceed when selecting them. Hyper-heuristics are an emergent search methodology, the goal of which is to automate the process of either (i) selecting and combining simpler heuristics [5], or (ii) generating new heuristics from components of existing heuristics [6]; in order to solve hard computational search problems. The main motivation behind hyper-heuristics is to raise the level of generality in which search methodologies can operate. They can be distinguished from other heuristic search algorithms, in 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.

The hyper-heuristic framework presented in [5,10], operates at a high level of abstraction and often has no knowledge of the domain. It only has access to a set of low-level heuristics (neighbourhood structures) that it can call upon, and has no knowledge of the functioning of those low-level heuristics. The motivation behind this approach is that once a hyper-heuristic algorithm has been developed, it can be applied to a new problem by replacing the set of low level heuristics and the evaluation function. Figure 1 illustrates that there is a barrier between the low-level heuristics and the hyper-heuristic. Domain knowledge is not allowed to cross this barrier. Therefore, the hyper-heuristic has no knowledge of the domain under which it is operating. It only knows that it has n low-level heuristics on which to call, and it knows it will be passed the results of a given solution once it has been evaluated by the evaluation function. A well defined interface between the hyper-heuristic layer and the problem domain layer needs to be provided, which will allow both the communication between the high-level strategy and the low-level heuristics, and the interchange of relevant non-domain information between the two layers. Furthermore, such an interface would permit the rapid incorporation of new problem domains. In other words, once a new domain is

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

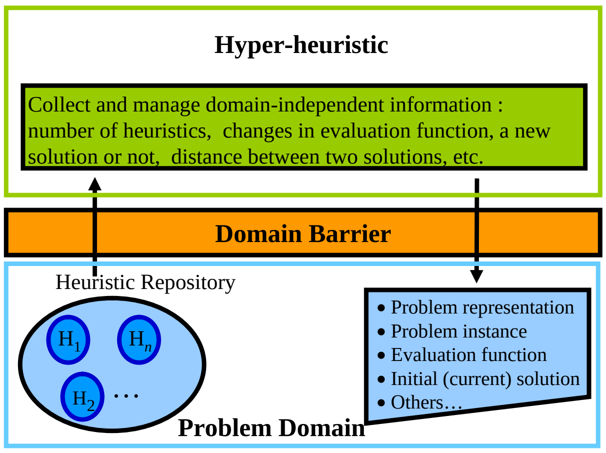
Hyper-heuristic
Collect and manage domain-independent information : number of heuristics, changes in evaluation function, a new solution or not, distance between two solutions, etc.
Domain Barrier
Heuristic Repository
H_1
H_n
H_2
…
Problem Domain
• Problem representation
• Problem instance
• Evaluation function
• Initial (current) solution
• Others…

Fig. 1 Hyper-heuristic framework performing single point perturbative search

identified, it would be relatively easy for an expert in the domain to produce a module according to the specifications of the proposed interface.

In this paper we propose a software framework inspired by the hyper-heuristic approach described above. Our goal is to provide software tool for the implementation and comparison of different hyper-heuristics. In doing so, we provide a software interface between the hyper-heuristic and the problem domain layers. The idea of having a software interface for hyper-heuristics was mentioned in [20]. An important feature of our framework, is that we provide the implementation (currently in Java) of a number of diverse combinatorial problem domains, including their solution representation, evaluation function and a set of useful and varied low-level heuristics. These domain modules encapsulate the problem specific algorithm components, and thus liberate the hyper-heuristic designer from knowing the details of the underlying applications. The creative and implementation efforts will instead be focused on the higher-level hyper-heuristic.

Powerful object oriented frameworks for designing and implementing local search heuristics and evolutionary algorithms have been proposed [2, 14, 9]. These frameworks provide a set of modules for implementing the components of search heuristics, leaving the implementation of the problem specific algorithm components to their clients or users. Our HyFlex framework takes the opposite direction! We provide a set of domain modules that encapsulate the problem specific algorithm components, namely, solution representation, evaluation function and a set of low-level heuristics for several hard (real-world) combinatorial problems. What is left to the user is to design high-level strategies (hyper-heuristics) that intelligently combine the set of heuristics provided. We argue that our framework provides a valuable tool for researchers that seek to test their algorithmic ideas on a wide set of problems.

The following section describes in more detail the proposed framework, including the application domains implemented and their associated heuristics.

2 The Proposed Framework

We extended the original perturbative hyper-heuristic framework (Figure 1) in two important ways. First, a memory or list of solutions is maintained in the domain layer, instead of a single incumbent solution. This extension enriches the possibilities for the hyper-heuristic designer, allowing for example the implementation of population based approaches. The idea of providing a list of solutions in a hyper-heuristic framework was first proposed in [25]. Second, a large set of low-level heuristics of different types is provided. Specifically, we consider four types of low-level heuristics, which to the best of our knowledge have not been incorporated simultaneously in a hyper-heuristic framework:

– *Mutational or perturbative heuristics*: perform a (generally) small change in the solution, by swapping, changing, removing, adding or deleting solution components. Note that different mutational heuristics can be easily defined by increasing the extent of the change in the solution.
– *Hill-Climbing heuristics*: iteratively make small changes (mutations or perturbations) to the solution, accepting improving or 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, therefore they guarantee that a non-deteriorating solution will be produced. Different hill-climbing heuristics can be defined by both modifying the extent of the perturbation, or the number of iterations (depth) of the search.
– *Ruin & recreate 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 constructive heuristics to rebuild the solutions. Different ruin and recreate heuristics can be defined by modifying the extent of the destruction. The construction process can be handled by different constructive heuristics each of which can be used to define a different ruin and recreate heuristic.
– *Crossover heuristics*: combine solution components from two input solutions (parents) to produce a new solution or solutions (offspring). Different variants of a crossover operator can be defined by modifying the proportion of solution components interchanged between parents.

As mentioned above, a number of heuristics of each type can be easily defined by altering the magnitudes controlling their operation. The framework provides two general parameters for defining heuristics: *intensity of change* and *depth of search*. These parameters can be varied in the range $[0, 1]$, and their effects are problem and heuristic dependent. It is the responsibility of the domain module designer to give an adequate meaning to these parameters (together with their default behavior) and properly document his/her choices.

In addition to the low-level heuristics described above, each HyFlex problem domain incorporates:

1. A routine to initialise solutions in the population.
2. A set of interesting instances that can be easily loaded using the method `loadInstance(a)`, where a is the index of the instance to be loaded.

3. A population of one or more solutions that has to be administered.

Currently 4 domain modules are implemented. Namely, the permutation flowshop problem, 1D bin packing, boolean satisfiability and personnel scheduling. Below we overview the main design choices for each domain. Technical reports are available describing the details of each module [11, 17, 16, 24].

2.1 The permutation flow shop problem

The permutation flow shop problem requires finding the order in which n jobs are to be processed in m consecutive machines. The jobs are processed in the order machine 1, machine 2, . . . , machine m. Machines can only process one job at a time and jobs can be processed by only one machine at a time. No job can jump over any other job, meaning that the order in which jobs are processed in machine 1 is maintained throughout the system. Moreover, no machine is allowed to remain idle when a job is ready for processing. All jobs and machines are available at time 0. Each job i requires a processing time on machine j denoted by p_{ij}.

Initialization : Solutions are initialized using the well established NEH procedure [19]. This heuristic has been used as an important component of many effective metaheuristics for the permutation flow shop problem. It has been used as both the initialization procedure of solutions, to be later improved, and also as the improving mechanism within the main iteration of more elaborate algorithms.

Low-level heuristics : A total of 14 low level heuristics were implemented. Specifically, 5 mutational, 4 local search (inspired by those proposed in [21]), 3 crossover heuristics (classical recombination operators for permutation representation) and 2 ruin and recreate heuristics (which incorporate the successful NEH procedure in the construction process). For more details see [24].

Instance data : A total of 120 instances from the widely known Taillard set [23], are provided. The instance sizes are are given in Table 1, in the format $n \times m$. The job processing times, on all instances, are uniformly distributed random integers in the range $[1, 99]$.

2.2 One dimensional bin packing

The one-dimensional bin-packing problem involves a set of integer-size pieces L, which must be packed into bins of a certain capacity C, using the minimum number of bins possible. In other words, the set of integers must be divided into the smallest number of subsets so that the sum of the sizes of the pieces in a subset does not exceed C.

Initialization : Solutions are initialized by first randomizing the order of the pieces, and then applying the widely known ‘first-fit’ heuristic [18]. This is a constructive heuristic, which packs the pieces one at a time, each into the first bin that they will fit into.

Low-level heuristics : 2 mutational, 2 ruin and recreate, repacked with best-fit, and 3 local search heuristics. These heuristics are inspired by those proposed in [1]. For more details see [17]

Instance data : The problem instances are summarized in table 2. There are 60 instances in total, 20 in each of three classes.

Instance number	Size
0-9	20×5
10-19	20×10
20-29	20×20
30-39	50×5
40-49	50×10
50-59	50×20
60-69	100×5
70-79	100×10
80-89	100×20
90-99	200×10
100-109	200×20
110-119	500×20

Table 1 Permutation flowshop module instances.

Set name	Piece size distribution	Bin capacity	Number of pieces	Ref
bp1	Uniform [20,100]	150	1000	[12]
bp2	Triples [25,50]	100	501	[12]
bp3	Uniform [150,200]	1000	100	[22]

Table 2 One dimensional bin-packing instance sets. Each set contains 20 instances.

2.3 Boolean satisfiability

The boolean satisfiability or SAT problem involves determining if there is an assignment of the boolean variables of a formula, which results in the whole formula evaluating to true. If there is such an assignment then the formula is said to be satisfiable, and if not then it is unsatisfiable. The process of finding an assignment that satisfies the formula is the search problem considered in this domain module.

Initialization : Solutions are initialized by simply randomly assigning a true or false value to each variable. The problem instances included are examples of the so called 3SAT problem, where each clause contains three variables.

Low-level heuristics : 2 mutational, 4 local search, and 2 heuristics that combine mutation and local search. These heuristics are described in [13], and comprise state of the art local search heuristics for this problem. For more details see [16]

Instance data : The problem instances are taken from the “Uniform Random-3-SAT” category on the ‘SATLIB’ website [15]. There are 60 instances in total, 20 from each of three classes. The instances are summarized in table 3.

2.4 Personnel scheduling

The personnel scheduling problem involves deciding at which times and on which days (i.e. which shifts) each employee should work over a specific planning period. However, the personnel scheduling problem is actually a title for a group of very similar problems.

Instance set name	Variables	Clauses
uf200-860	200	860
uf225-960	225	960
uf250-1065	250	1065

Table 3 Boolean satisfiability module instances.

There is no general personnel scheduling problem. Instead there is a group of problems with a common structure but which differ in their constraints and objectives. This creates an additional challenge in implementing a problem domain module for personnel scheduling. To overcome this we have designed a data file format for which each instance can select a combination of a objectives and constraints from a wide choice. We then implemented a software framework containing all the functions for these constraints and objectives.

Initialization : Initial solutions are created with a hill climbing heuristic which uses a neighbourhood operator that adds new shifts to the roster.

Low-level heuristics : 3 mutational (including *vertical*, *horizontal* and *new* swaps, see [11]), 5 local search, 3 ruin and recreate, and 3 crossover heuristics. These heuristics are taken from previously proposed successful meta-heuristic approaches to nurse rostering problems [3, 4, 7, 8]

Instance data : The instances have been collected from a number of sources. Some of the instances are from industrial collaborators. These include: ORTEC an international consultancy and software company who specialise in workforce planning solutions and SINTEF, the largest independent research organisation in Scandinavia. Other instances have been provided by other researchers or taken from various publications. The collection is a very diverse data set drawn from eleven different countries. The majority of the instances are real world scenarios. An overview of the instances can be found in [11], they vary in the length of the planning horizon, the number of employees and the number of shift types. Each instance also varies in the number and priority of objectives present[1].

3 Discussion

We are proposing a novel framework for supporting research into modern search methodologies. The emphasis of our HyFlex framework lies in providing the algorithm components that are problem specific, thus liberating our users (algorithm designers) from needing to know the problem domain's specific details. The design efforts will instead be focused on designing high-level strategies to intelligently combine the provided tools. Preliminary tests on the reusability of the modules have successfully been conducted. Our motivation is to promote research towards the design of general search methodologies. We plan to extend the number of problem domains and propose a challenge, based on our framework, where the winners will be those algorithms with a better overall performance across all of the different domains. The competition will be 'fair' in that

[1] The instances can be downloaded from:
`http:///www.cs.nott.ac.uk/~tec/NRP/`

it will be conducted in a high-level of abstraction: the application domain implementation details are hidden to the framework users. Using an Olympic metaphor, we are no longer interested in the 100 meters race, but instead in conducting the *Decathlon* of modern search methodologies.

References

1. R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. Technical report, School of Computer Science, University of Nottingham, 2007.
2. S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, volume 2632 of *LNCS*, pages 494–508, Berlin, 2003. Springer.
3. E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A scatter search for the nurse rostering problem. Technical report, School of Computer Science, University of Nottingham, 2007.
4. E. K. Burke, T. Curtois, R. Qu, and G. Vanden Berghe. A time predefined variable depth search for nurse rostering. Technical report, School of Computer Science, University of Nottingham, 2007.
5. 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.
6. 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. (to appear).
7. E.K. Burke, P. Cowling, P. De Causmaecker, , and G. Vanden Berghe. A memetic approach to the nurse rostering problem. *Applied Intelligence*, 15(3):199–214, 2001.
8. E.K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman. A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem. *European Journal of Operational Research*, 188(2):330–341, 2008.
9. S. Cahon, N. Melab, and E-G. Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, May 2004.
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, 2000. Springer.
11. T. Curtois. A hyflex module for the personnel scheduling problem. Technical report, School of Computer Science, University of Nottingham, 2009.
12. E Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.
13. A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation (MIT Press)*, 16(1):31–1, 2008.
14. L. Di Gaspero and A. Schaerf. Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. *Softw, Pract. Exper*, 33(8):733–765, 2003.
15. H. H. Hoos and T. Stützle. Satlib: An online resource for research on sat. In I. P. Gent, H. V. Maaren, and T. Walsh, editors, *SAT 2000*, pages 283–292. IOS Press, 2000. SATLIB is available online at www.satlib.org.
16. M. Hyde. A hyflex module for the boolean satisfiability problem. Technical report, School of Computer Science, University of Nottingham, 2009.
17. M. Hyde. A hyflex module for the one dimensional bin-packing problem. Technical report, School of Computer Science, University of Nottingham, 2009.
18. 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.
19. 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.

20. A. J. Parkes. A proposal for a hyper-heuristics software interface. Oral presentation, May 2007. Automated Scheduling, Optimisation and Planning Research Group. Internal Seminar.
21. R. Ruiz and T. G. Stützle. An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *Journal of Operational Research*, 187(10):1143–1159, 2007.
22. P. Schwerin and G. Wäscher. The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp. *International Transactions in Operational Research*, 4(5):377–389, 1997.
23. E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
24. J. A. Vazquez-Rodriguez. A hyflex module for the permutation flow shop problem. Technical report, School of Computer Science, University of Nottingham, 2009.
25. J. Woodward, A. Parkes, and G. Ochoa. A mathematical framework for hyper-heuristics. Oral presentation, 2008 September. Workshop on Hyper-heuristics - Automating the Heuristic Design Process, in conjunction with the 10th International Conference on Parallel Problem Solving From Nature (PPSN X), Dortmund, Germany.