Towards the ‘Decathlon Challenge’ of Search Heuristics

Edmund K. Burke, Tim Curtois, Graham Kendall, Matthew Hyde,
Gabriela Ochoa, Jose A. Vazquez-Rodriguez and Sanja Petrovic
Automated Scheduling, Optimisation and Planning (ASAP) Group
School of Computer Science
University of Nottingham, UK
{ekb,tec,gxk,mvh,gxo,jav,sxp}@cs.nott.ac.uk

ABSTRACT

We present an object oriented framework for designing and evaluating heuristic search algorithms that achieve a high level of generality and work well on a wide range of combinatorial optimization problems. Our framework, named HyFlex, differs from most software tools for meta-heuristics and evolutionary computation in that it provides the algorithm components that are problem-specific instead of those which are problem-independent. In this way, we simultaneously liberate algorithm designers from needing to know the details of the problem domains; and prevent them from incorporating additional problem specific information in their algorithms. The efforts need instead to be focused on designing high-level strategies to intelligently combine the provided problem specific algorithmic components. We plan to 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. Using an Olympic metaphor, we are not solely focussed on the 100 meters race, but instead on the decathlon of modern search methodologies. Categories

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*

General Terms

Algorithms, Design.

Keywords

Combinatorial optimization, hyper-heuristics, meta-heuristics, parameter tuning, software frameworks

1. INTRODUCTION

Meta-heuristics or general purpose heuristics have dominated the combinatorial optimization arena for the last two decades. Researchers in this area, however, are often constrained on the number of applications on which they can test their general purpose methods. One of the reasons is that a large proportion of the required effort must be invested in the construction of the problem-specific heuristics and data structures. Recent developments in search methodologies towards more generally applicable techniques has been termed hyper-heuristics [5]. The majority of current hyper-heuristic approaches attempt to intelligently combine or select between previously proposed simpler heuristics, where it is not clear which one will be most effective for the problem instance at hand. The hyper-heuristic framework is provided with a repository of problem specific heuristics (often referred to as ‘low-level’ heuristics). Several hyper-heuristics have been proposed, which use meta-heuristics or machine learning techniques as the high-level strategy and combine the strengths of the low-level heuristics implemented. However, results are only ever reported on a few problem domains. Again, this is due the time and effort it takes to implement each problem domain and its associated low-level heuristics. Many of the low-level heuristics are complex and sophisticated, and it requires a reasonable level of familiarity and experience if one is to implement a different/new problem domain.

To overcome such limitations and to support research into generally applicable search algorithms, we developed HyFlex, a modular and flexible Java class library for designing and testing search methodologies. The framework was designed having in mind hyper-heuristics, but can be used also for testing other search methodologies such as meta-heuristics. Our framework differs from other meta-heuristics software tools in that it provides the algorithm components that are problem-specific (as opposed to those which are problem-independent). for The idea is to provide problem domain modules that encapsulate the solution representation, fitness evaluation, and a repository of associated low-level heuristics for a number of hard combinatorial optimization problems. Importantly, only the high-level strategy needs to be implemented by the user, as HyFlex provides an easy to use interface with which the problem domains can be accessed. Each HyFlex domain module includes a library of problem instances. It also provides the means to manipulate a population of solutions, so a high-level strategy is not limited to using just one incumbent solution. Our aim is to make it easy for researchers to test the generality and effectiveness of their search algorithms, by taking away the difficult task of implementing many problem domains. Valuable research effort can thus be devoted to developing the high-level strategies that manage the (provided) problem specific heuristics.

The idea for a search heuristics challenge or competition was inspired by the the metaphor of a "many-walls" game as opposed to the more traditional "up-the-wall" game in search methodologies [6]. In the up-the-wall game, the idea is to develop and apply newly proposed search methodologies to existing benchmark problems, and compare with other "players". The goal is to "get further up the wall" than the other players. On the other hand, the goal of the many-walls game will be to operate on as many different walls as possible, while still getting acceptably high up each wall. In other words, the goal is to rise the level of generality of search methodologies

Our framework is also inspired by the hyper-heuristic approach discussed in [5, 9], which operates at a high level of abstraction and often has no knowledge of the domain. It only has access to a set of perturbative low-level heuristics (neighborhood structures) that it can call upon, and has no details as to 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. The idea of having a software interface for hyper-heuristics was suggested in [18]. Another relevant antecedent to our framework is PISA [2], a text-based software interface for multi-objective evolutionary algorithms. PISA proposes to divide the implementation of an evolutionary algorithm into an application-specific part and an algorithm-specific part. The latter contains the selection procedure, whilst the former encapsulates the solution representation, the generation of new solutions, and the calculation of the fitness values. The two parts are implemented as distinct programs that communicate via a text-based interface, which provides maximum independence of programming languages and computing platforms.

Our proposal extends the perturbative hyper-heuristic framework [5, 9] in two important ways. First, a memory or list of solutions is maintained in the domain layer, instead of a single incumbent solution (this follows a suggestion presented in [23]). This extension enriches the possibilities for the hyper-heuristic designer allowing, for example, the implementation of population based approaches. Second, a large set of low-level heuristics of different types is provided, consisting of four types: mutational, ruin-recreate, hill-climbers and crossover. Also, HyFlex differs from PISA [2] in that its interface is not text-based but instead given by an abstract Java class. Our proposal is not tied to evolutionary algorithms, but allows the implementation of any single-point meta-heuristics and hyper-heuristics. Moreover, whilst PISA provides mainly classic benchmarks and abstract domains, it is our intention to provide a rich and varied collection of real-world hard combinatorial problems.

The following section describes the components of the proposed domain modules and overviews the four application domains we have implemented so far.

2. THE PROBLEM DOMAIN MODULES

Each HyFlex problem domain module consists of:

1. A routine to initialise solutions in the population.
2. A set of heuristics to modify solutions classified into five groups:

   mutational : makes a (typically random) modification to the current solution.

   ruin-recreate : destroy part of the solution and rebuild it using a constructive procedure (typically a greedy procedure).

   local search : searches in the *neighbourhood* of the current solution for an improved solution.

   crossover : takes two initial solutions, combines them and returns a new solution.

   others : heuristics that do not fit into any of the above categories.
3. 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.
4. A population of one or more solutions that has to be administered.
5. Two parameters α and β, $0 \leq \alpha, \beta \leq 1$, which are the "intensity of mutation" and "depth of search", respectively, that may control the behaviour of some low level heuristics.

Note that items 1, 2 and 3 are problem dependent. 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 [10, 15, 14, 22].

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 $2, \ldots,$ 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 [17]. This heuristic has been used as an important component of many effective meta-heuristics 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 [19]), 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 [22].

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	[11]
bp2	Triples [25,50]	100	501	[11]
bp3	Uniform [150,200]	1000	100	[20]

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

Instance data : A total of 120 instances from the widely known Taillard set [21], 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 [16]. 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 [15]

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

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.

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

Table 3: Boolean satisfiability module instances.

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 [12], and comprise state of the art local search heuristics for this problem. For more details see [14]

Instance data : The problem instances are taken from the "Uniform Random-3-SAT" category on the 'SATLIB' website [13]. 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. 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 [10]), 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 [10], 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 AND FUTURE WORK

We are proposing a novel framework for supporting research into modern search methodologies. The HyFlex framework provides the algorithmic components that are problem specific, thus liberating the users (algorithm designers) from needing to know the problem domain's specific details. At the same time, we prevent the users from incorporating additional problem domain knowledge. The design efforts will instead be focused on designing high-level strategies to intelligently combine the the low-level heuristics provided. Four domain modules have been implemented: the permutation flowshop problem, 1D bin-packing, Boolean satisfiability and personnel scheduling. Preliminary tests on the reusability of the modules have successfully been conducted. Moreover, several meta-heuristics and hyper-heuristics have been implemented and initial testing has been carried out. This is work in progress, we are not yet announcing the challenge. Our current efforts are directed to extend the number of problem domains. Specifically, we plan to include modules on educational timetabling, and vehicle routing. Our goal is to promote research into generally applicable search methodologies and eventually conduct the Decathlon challenge of search heuristics. In order to decide the winner of our competition, we plan to follow a similar point system than the one currently used in the Olympic event. Further work is needed to adequately adapt this system.

4. 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, G. Kendall, and R. Qu. Hyper-heuristics and theier emploument on search problems. The 25th Workshop of the UK Planning AND Scheduling, PlanSIG 2006, December 2006. Keynote talk.

[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] 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.

[10] T. Curtois. A hyflex module for the personnel scheduling problem. Technical report, School of Computer Science, University of Nottingham, 2009.

[11] E Falkenauer. A hybrid grouping genetic algorithm for bin packing. *Journal of Heuristics*, 2:5–30, 1996.

[12] A. S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation (MIT Press)*, 16(1):31–1, 2008.

[13] 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.

[14] M. Hyde. A hyflex module for the boolean satisfiability problem. Technical report, School of Computer Science, University of Nottingham, 2009.

[15] M. Hyde. A hyflex module for the one dimensional bin-packing problem. Technical report, School of Computer Science, University of Nottingham, 2009.

[16] 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.

[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] A. J. Parkes. A proposal for a hyper-heuristics software interface. Oral presentation, May 2007. Automated Scheduling, Optimisation and Planning Research Group. Internal Seminar.

[19] 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.

[20] 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.

[21] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[22] J. A. Vazquez-Rodriguez. A hyflex module for the permutation flow shop problem. Technical report, School of Computer Science, University of Nottingham, 2009.

[23] 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.

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