

A Mathematical Framework for Hyper-Heuristics

XXX¹ and YYY¹

¹ University

abc@xyz,

WWW home page: <http://homepage.html>

² University, XYZ,

Country

1 Introduction

Hyper-heuristics are informally defined as “heuristics to choose heuristics” [1]; given a set of heuristics, produce a sequence in which to apply them. The aim of this paper is to formalize this statement, and ultimately give a model for implementation, thus supporting the central motivation of “raising the level of generality” [1] of search methodologies. In order to abstract away from a problem domain, an interface (or problem domain barrier), must be defined in order for the two layers to communicate coherently (i.e. the problem layer and the hyper-heuristic layer). It is the specification of this interface which is one of the concerns of this paper. We must also define components of the two layers.

A primary contribution of this paper is that in the problem layer we include a list of solutions; this list supports ‘probing’ of the problem instance by the heuristics, where newly created solutions can be rejected. This would be *impossible* to achieve if only a single solution was maintained by the problem layer, and is therefore a central component. The use of a list also potentially allows population-based methods to be incorporated within the hyper-heuristic framework.

Heuristics and Hyper-Heuristics. A heuristic is domain specific with an objective function in mind. Meta-heuristics, on the other hand, are more general in the sense that they can be applied to any problem domain, provided there is some method to represent possible solutions. Examples are biologically inspired search algorithms, greedy search and simulated annealing. For intractable problems with a definite need for a heuristic approach, a promising avenue to finding practical solutions is to “hyper-heuristically” choose heuristics from a set. One intention is that the resulting combination will give better performance than any single heuristic in the set. The sequence of heuristics can be chosen by an algorithm (e.g. reinforcement learning or machine learning). This idea has been used by other researchers, for example [2] where two solutions are generated via a pair of operators and the better is accepted. This sort of approach, can readily be expressed in this framework (see example below).

Components of Framework. The abstraction consists of two layers; a *hyper-heuristic layer* and a *problem layer* (see fig 1). The components of the

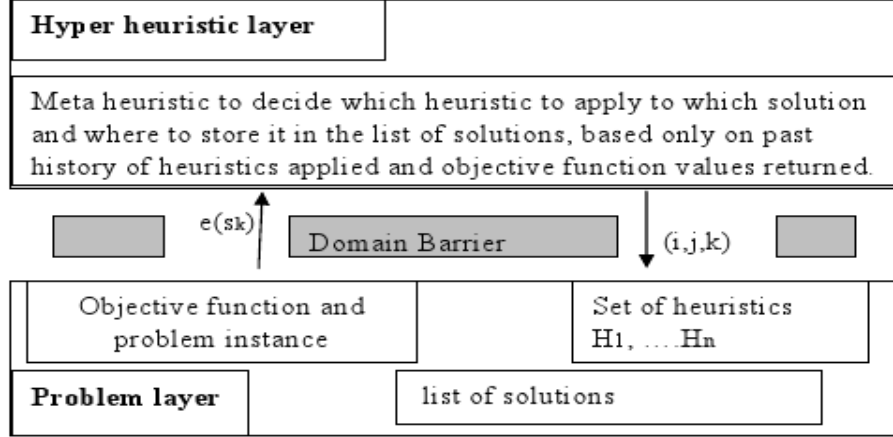


Fig. 1. The problem layer consists of; the problem instance, objective function, a set of heuristics and a list of solutions. The list allows the hyper-heuristic to store new solutions, but not necessarily accept them. In the hyper-heuristic layer, the choice of which heuristic to apply is made, based solely on the feedback from the problem domain (i.e. the objective function values) and the choices it has made in the past (i.e. all previous triples).

problem layer are; a problem instance, an objective function, a set of applicable heuristics and a list data structure to store solutions. In the problem layer a heuristic is applied to a specified solution in the list and the value of the objective function is returned to the hyper-heuristic layer. In the *hyper-heuristic layer*, the choice of which heuristic to apply is made by an algorithm.

Definitions. A heuristic H_i is a function mapping one solution to another solution. $H = \{H_0, H_1, H_2, \dots, H_n\}$ is a set of predefined domain specific heuristics, where H_0 is the identity heuristic. $\langle i, j, k \rangle$ is a triple (i.e. an 3-tuple, $H \times \mathbb{N} \times \mathbb{N}$) which is returned by the hyper-heuristic ($0 \leq i \leq n$). It is interpreted in the problem layer as follows; apply heuristic H_i to the solution s_j stored in list position j and store the resulting solution s_k in position k . The problem layer takes the triple $\langle i, j, k \rangle$ as input and returns the value of the objective function, e , on s_k (i.e. $e(s_k)$ where e is a function mapping a solution to a real value). A step t constitutes the production of a triple $\langle i, j, k \rangle_t$ by the hyper-heuristic layer, and a response $e(s_k)_t$ by the problem layer. Let us define a quadruplet Q_t (i.e. a 4-tuple) as $Q_t = \langle i, j, k, e(s_k) \rangle_t$. At step t , the hyper-heuristic will have at its disposal a list of all previous quadruplets (the concatenation of all previous quadruplets), i.e. $\langle i, j, k, e(s_k) \rangle_1, \langle i, j, k, e(s_k) \rangle_2, \dots, \langle i, j, k, e(s_k) \rangle_t$, on which to base the decision at the $t + 1$ th step i.e. the triple $\langle i, j, k \rangle_{t+1}$. A hyper-heuristic can *only* make choices based on this information. Hence a hyper-heuristic is a function mapping a list of quadruplets to a triple, i.e. $HH : (H \times \mathbb{N}^2 \times \mathbb{R})^* \rightarrow H \times \mathbb{N}^2$.

Example. Consider the hyper-heuristic (HH) of [2] in which a pair of heuristics are applied to a current solution and the better solution of the pair of results is accepted, regardless of if its quality is better or worse than the current solution (assume minimizing). Initially, HH has no history (i.e. a null list, $\langle \rangle$) so applies H_1 to solution at position 1, storing the result in position 2, and then applies H_2 to solution at position 1, storing the result in position 3;

$HH(\langle \rangle) \rightarrow \langle 1, 1, 2 \rangle; e(s_2) \rightarrow 8;$

$HH(\langle \rangle, \langle 1, 1, 2, 8 \rangle) \rightarrow \langle 2, 1, 3 \rangle; e(s_3) \rightarrow 9;$

$HH(\langle \rangle, \langle 1, 1, 2, 8 \rangle, \langle 2, 1, 3, 9 \rangle) \rightarrow \langle 1, 2, 1 \rangle; e(s_1) \rightarrow 7;$

$HH(\langle \rangle, \langle 1, 1, 2, 8 \rangle, \langle 2, 1, 3, 9 \rangle, \langle 1, 2, 1, 7 \rangle) \rightarrow \langle 2, 2, 3 \rangle; e(s_3) \rightarrow 6;$

$HH(\langle \rangle, \langle 1, 1, 2, 8 \rangle, \langle 2, 1, 3, 9 \rangle, \langle 1, 2, 1, 7 \rangle, \langle 2, 2, 3, 6 \rangle) \rightarrow \langle 1, 3, 1 \rangle;$

Notice in this example, it is essential to be able to access more than a single solution

Discussion. A key point is that a list is used in the problem layer to store solutions effectively giving “solution memory” to the hyper-heuristic. As already seen above, if only a single solution was maintained in the problem layer it would be *impossible* to backtrack to a previous solution, as we could not ‘undo’ the heuristic, as the inverse function may not exist (i.e we cannot in general reverse the action of a heuristic to recover the previous solution).

Also, it allows the expression of population-based methods such as Genetic Algorithms to be used at the hyper-heuristic layer. A natural extension to this formulation would be to allow the “heuristic” to take more than one solution as input, and give more than one solution as output. This would allow the standard crossover operator of Genetic Algorithms to be expressed, as a “heuristic” that maps two solutions to two solutions. This has the interesting potential to unify the ideas of hyper-heuristics with the related “domain-blindness” that is also common in “off-the-shelf” Genetic Algorithm frameworks.

The problem layer returns a scalar $e(s_k)$ but could be extended to return an ordered pair (a 2-tuple), where the second component could be the amount of CPU time used by the heuristic, or a second objective value for example.

There is other trivial information which is needed to be passed across the interface which includes; the number of hyper-heuristics, termination of the hyper-heuristic, and a record (an integer) of the best solution obtained.

References

1. Burke, E.K., Kendall, G.: Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques. Springer (2005)
2. Yao, X., Liu, Y., Liu, G.: Evolutionary programming made faster. IEEE Trans. on Evolutionary Computation **3**(2) (1999) 82–102