Searching the Hyper-heuristic Design Space

Jerry Swan[1], John Woodward[2],
Ender Özcan[2], Graham Kendall[2], Edmund Burke[1]

1. Computing Science and Mathematics, School of Natural Sciences,
University of Stirling, Stirling FK9 4LA, SCOTLAND.
2. School of Computer Science, University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK.
jerry.swan@cs.stir.ac.uk,{jrw,exo,gxk}@cs.nott.ac.uk,e.k.burke@stir.ac.uk

Abstract. We extend a previous mathematical formulation of hyper-heuristics to reflect the emerging generalization of the concept. We show that this leads naturally to a recursive definition of hyper-heuristic and to a division of responsibility that is suggestive of a blackboard architecture, in which individual heuristics annotate a workspace with information that may also be of interest to other heuristics. Such a framework invites consideration of the kind of relaxations of the domain barrier that can be achieved without loss of generality.

1 Introduction

The term *hyperheuristic* first appeared in [1] in the context of automated theorem proving. As introduced in [2], *hyper-heuristics* can be considered as the study of 'heuristics to *choose* heuristics'. Recent work by Burke et al. [3] classified diverse approaches under the more general concept of 'heuristics for *searching the space of* heuristics'. We proceed to reflect this generalization by extending the mathematical framework given by Woodward et al. in [4]. In formulating a generalization, we also address some issues arising from the initial framework definition. We proceed from there to discuss some of the architectural and engineering implications of this generalization. We draw parallels with the operation of a cognitively-inspired architecture and discuss the notion of incorporating 'heuristic-autonomy' as an additional dimension of the design space.

In order to achieve the side-effect free requirements of a mathematical formulation within the context of computer science, we have found it most convenient to adhere (where possible) to the notation of the *Haskell* programming language [5] — for those unfamilar with the Haskell language, knowlege of the elementary notion of function signatures is likely to suffice. In [4], the following definitions are provided: for solution-state space S, a *heuristic* is a function $h : S \to S$ and $O = \{o_1, o_2, \ldots, o_n\}$ is a set of predefined domain-specific heuristics. (i, j, k) is a 3-tuple intepreted in the invoking 'problem layer' as follows: apply heuristic o_i to the solution stored in list position j and store the resulting solution s_k in position k. Let $e : S \to \mathbb{R}$ be the objective function and Q be the 4-tuple

$(i, j, k, e(s_k))$. A *hyper-heuristic* is then a function:

$$H : [Q] \rightarrow O \times \mathbb{N}^2 \tag{1}$$

(where square brackets denote a list) i.e. a hyper-heuristic can generate the information required for a new list entry and ask for it to be inserted into any chosen position in the list (the idea being that some heuristics may make elect use of this position ordering). Fig. 1 depicts the enforcement of the flow of information between the hyper-heuristic and problem layers.

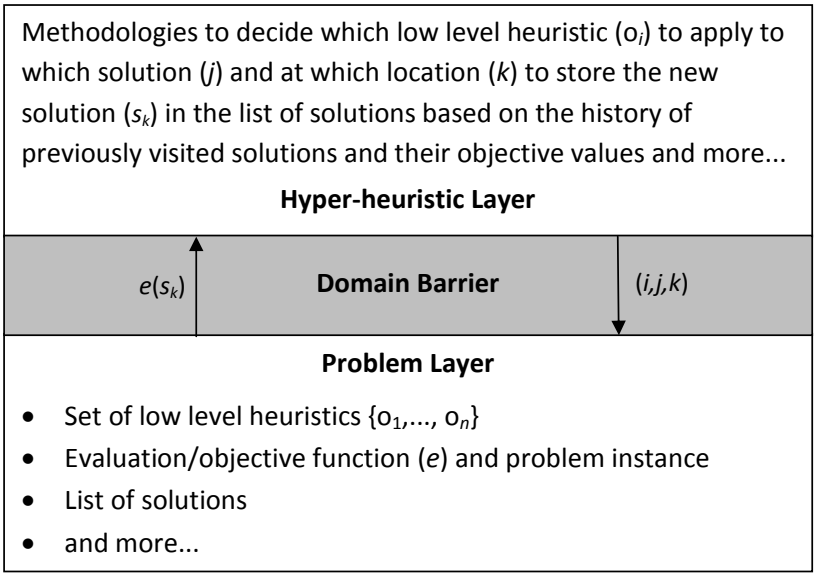

Fig. 1. Decomposition of a Hyper-heuristic into Hyper- and Problem- layers.

We note that this formulation does not appear to allow for O to vary during the course of the run, i.e. it is restricted to selective rather than generative activity. The restriction to an objective value of $\mathbb{R}$ rather than $\mathbb{R}^n$ also precludes multiobjective optimization, though this is trivially addressed (at least on a conceptual level). A more significant issue arises with the 'mathematical' nature of this formulation. The definition implies specific activity on behalf of the invoking problem layer: the idea is that the invoker iteratively applies H in order to obtain a solution. In order to be genuinely 'mathematical', each invocation should be stateless, i.e. free of side-effects. The above definition is not sufficient in this respect for many of the metaheuristics we might wish to implement in practice. For example, if H is a metaheuristic such as A^*, then it is necessary to annotate each encountered state with the current 'least path cost' - this information cannot be determined merely from the history list Q. The same issue applies when annotating states with recency and frequency information in tabu-search.

As a minimum, it is therefore necessary to extend the signature of H to include a *workspace* parameter W:

$$H : [Q] \times W \rightarrow O \times \mathbb{N}^2 \times W \tag{2}$$

In the most elementary realization, a workspace can be considered to be a set of $(key, value)$ pairs that can be used to index elements of metaheuristic state (such as the path-cost described above) that cannot be derived from the list Q.

2 A Revised Formulation

In [3], Burke et al. classify hyper-heuristics in two orthogonal dimensions. The first dimension represents selection versus generation and the second the source of feedback during learning (online,offline or none). Both dimensions are further partitioned by the nature of the search space (constructive or pertubative). Deciding which elements of this design space to explore and when to do so is of course a search problem in its own right and Burke et al. note the emerging trend for hybrid methodologies that seek to achieve this.

It is nonetheless the case that the vast majority of current research work in hyper-heuristics requires the designer to make an *a priori* decision regarding which (singular) element of the hyper-heuristic design space their system is to explore. It is our contention that significant further progress in hyper-heuristics requires a system that is capable of allocating resources to the exploration of *any* element of the design space. Such a system would ideally marshall co-operation between these activities to produce solutions otherwise unobtainable.

We proceed to describe a unifying formulation intended to facilitate dynamic exploration of any of the dimensions of the design space. We start by defining the signature of a low-level heuristic $h_0 \in \mathcal{H}_T$ to be:

$$h_0 : T \rightarrow T$$

where T is a *type parameter* denoting the type of the solution state (e.g. bitstring, permutation, real-valued vector etc), as described below. We then define a hyper-heuristic $h_1 \in \mathcal{H}_T$ to have signature:

$$h_1 : T \times [\mathcal{H}_T] \rightarrow T$$

A specific instantiation of the type parameter T determines the role of $\mathcal{H}_T$. For perturbative heuristics, T is some complete solution-state S, whereas for constructive heuristics T is some partial solution-state P. Selective hyper-heuristics may then be instantiated over complete or partial solution-states as appropriate. For some solution-state of type E, let G be the space of functions : $E \rightarrow E$. A generative hyper-heuristic is then $\mathcal{H}_{E \rightarrow E}$.

In order for a framework to range over the entire design space, distinct elements of the design space must be able to interoperate where meaningful. Since we are defining our framework in functional terms, the first place we might

seek a mechanism for (hyper)heuristic interoperability is via a unified function signature. Consider perturbative heuristics as having signature

$$perturb : S \to S$$

i.e. a mapping from state to state for some generic state type S, with constructive heuristics as

$$construct : P \to Maybe\ P$$

where P is some generic notion of partial state. This syntax for *construct* expresses the notion constructive heuristics are in general partial (as opposed to total) functions (e.g. in the case of a fallible stochastic or greedy construction).

The above classification means that the primitive heuristics present a number of distinct function signatures to the hyperheuristic level (taken from the Cartesian product of selection\generation and constructive\pertubative). In fact, multiple concrete representations of solution state are clearly possible (e.g. list versus graph for the TSP). In addition to this proliferation of function signatures, an issue arises from the differing atomic granularity of partial construction versus perturbation. Unless we stipulate that partial construction is internally iterated to yield a complete state, then perturbative and constructive heuristics cannot share a simple common signature.

Acting together, these design issues indicate that a direct unification of these function signatures (i.e. in which the parameters are the union of those required by the heuristics from each element of the design space) is neither natural nor desirable. As per Dijksta's famous maxim, we resolve this issue via an additional level of indirection. Specifically, we mandate a signature for both types of heuristic that takes a *single* workspace argument. Hence, we now have $perturb : W \to W$ and $construct : W \to W$.

We therefore have a unified signature for our heuristics as follows:

$$h'_0 : W \to W, \quad h'_1 : W \to W$$

where the parameter W is a workspace that is now taken to be the repository for the both the state of the heuristics (as above) *and* the state of the search. The associated semantics are that heuristics read whatever parameters they require from the workspace (e.g. search trajectory and/or other hyper-heuristic-specific state) and write the relavant output of their computation (e.g. an update to its associated search trajectory and/or heuristic state) into the result. Note that h'_0 and h'_1 are no longer explicitly bound to a type-parameter S - which representations a heuristic may choose to access from the workspace and manipulate is now an implementation issue of that heuristic.

3 Implications of this Formulation

Since h'_0 and h'_1 share a common signature, our definition of hyperheuristic is now *recursive* - the list of heuristics $\mathcal{H}$ that h_1 accesses from the workspace may

themselves be hyperheuristics. Hyperheuristic invocations may thus be recursively nested to an arbitrary depth. Such an aggregation mechanism facilitates the generation of new hyperheuristics. An equivalent formulation for recursion has been implemented in the HYPERION hyperheuristic solution-domain framework [6]. The use of the workspace as the repository for collective state has the following implications:

1. By means of the workspace, we can pass not only the parameters $[Q], O, i, j$ of the original formulation, but also any other information that might profitably be shared between heuristics. In general, heuristics can elect to make use of workspace information contributed by other heuristics of which they are aware. For example, the heuristics output by any generative hyper-heuristic might profitably be made available to selective hyper-heuristics as a palette for selection, or to other generative hyper-heuristics as elements for variation operators. It is therefore implicit in this formulation that the set O of primitive heuristics and hyper-heuristics created therefrom (or indeed any other parameter) can be varied dynamically by a suitably-informed heuristic.
2. A constructive heuristic that eventually (i.e. after successive invocations) yields a complete solution from a succession of partial ones can add this to the complete solution trajectory.
3. Perhaps most significantly, the domain barrier need now be no more opaque than is genuinely useful in practice. As a concrete example of additional workspace information that can be formulated in a domain-independant fashion is the notion of the *inverse* of a perturbative heuristic — such information can profitably be used in tabu-lists, for example.

A consequence of 2. is that we need no longer require that any single invocation of a heuristic succeeds in adding a new (complete or partial) solution-state to the head of the search trajectory. Activity can thus be considered to be amortized over a succession of invocations. This notion of 'amortized' or 'eventual' construction means that we are freed from having to choose between 'timesliced' and 'greatest common multiple' approach to the different atomic granularities of partial construction and perturbation — the indirection via the workspace means that it is not necessary to enforce any specific iteration policies.

In fact, if we view the differing granularities of partial construction and perturbation as a special case of process-granularity in general, this approach allows us to integrate heuristics that are resident in main-memory with those that are federated (e.g. accessed via remote-procedure call or web-service). The invocation overhead for the latter will generally be much higher than the former, and so we might want a single invocation to perform much more 'useful work' (e.g. exploring to a local optimum) in the latter case. The indirected approach we advocate is necessary in order to remain as agnostic as possible about granularities.

3.1 Learning

Up to this point, the 'learning' dimension of the design space has not been directly addressed. The success of learning is often in proportion to the CPU

time or storage-capacity allocated to it, so rather than being a static dimension of the system, we might profitably consider the allocation of resources to (online or offline) learning to be an aspect of some high-level control mechanism (which may of course itself be a hyper-heuristic).

Examples of online learning include reinforcement learning [7], tabu search ([8], [9]) and adaptive penalties [10]. Offline learning seeks to extract information obtained from a set of training instances in order to better inform the subsequent search process. Examples of offline learning for hyper-heuristics includes learning-classifier systems [11], case-based reasoning [12] and genetic programming [13]. While much work has been done in the offline analysis of landscapes and devising metrics of landscape quality ([14], [15], [16], [17], [18], [19]), this information is not routinely used to inform online activity. As a simple but concrete example, the autocorrelation length of the landscape [20] could be used to inform parameters such as tabu-tenure or population size, but such interplay between design and experiment is not a routine part of metaheuristic development. To draw a real-world analogy, this is akin to attempting to design a car without prior knowledge of the nature of roads.

It is also worth noting that online and offline activity actually aid one-another in *both* directions, i.e. information produced online (sampling the search space) is likely to be of use offline (e.g. to inform a subsequent heuristic generation process). Overall, it is therefore our belief that a hard distinction between online and offline learning is a significant obstacle to progress. We are further of the opinion that this issue (as with many in the current state of metaheuristic research) is much more of an architectural\software engineering issue than a conceptual one. We believe that this is further motivation for an integrated exploration of the design space.

3.2 Options and Responsibilities for Top-Level Control

In concrete implementation terms, the decentralization of concerns discussed in this article is an essential characteristic of a *blackboard architecture* [21]. Blackboard architectures are opportunistic, data-driven mechanisms for mediating between a collection of indirectly-collaborative knowledge sources. They are of greatest utility when the control sequence or solution structure for a problem is not best delineated in advance, which are certainly properties of hyper-heuristic search. One property enjoyed by modern blackboard architectures is concurrency. Numerous control variants are discussed in [22]. We briefly consider here some options in the specific context of hyper-heuristic control:

A somewhat polar school of thought contends that (if sufficiently abstracted from the underlying problem domain) the very topmost hyper-heuristic control mechanism must necessarily be solving the multi-armed bandit problem [23] for the agents (resources) that it is coordinating. Such a resource-management abstraction has a number of theoretical benefits, not least that it allows us to ground the system in utilitarian terms (CPU or cloud-computing costs) and to entertain space-time tradeoffs (e.g. the length of the history list Q) within a unified framework.

One might also consider a top-level that embarks on a form of 'generate and test' for hypotheses in the manner of scientific discovery systems such as BACON,QNM or LaGrange [24]. As a concrete example of the utility of such a system, in order to avoid overfitting it is imperative that the system be exposed to a set of representative training examples. The ability to generate new test cases would thus be an interesting avenue of exploration.

Alternatively, if one were to take devolution of concerns to its logical conclusion, this effectively yields a multi-agent system. Co-operation between low-level heuristics is explored in [25], but to the knowledge of the authors no complete integration of design space elements has been proposed. From this perspective, 'heuristic autonomy' can be seen as another dimension of the design space. This notion of 'maximal automomy' invites the possibility of a subsumption-architecture approach in which the system has no conventional top-down control mechanism [26], but which can nonetheless be *grounded* (in the 'good-enough, fast-enough, cheap-enough' sense) via the choice of suitable 'timeout' etc. parameters for the heuristic agents.

4 Conclusion

Metaheuristic researchers are enthusiastic in adopting of metaphors of computation from the natural world. The human mind is the the most stellar known example of natural parallel computation, so it perhaps suprising that cognitive architectures have done little to inform hyper-heuristic activity.

Thus inspired, we have described a unifying 'mathematical' formulation for hyper-heuristics and proceeded to show that a number of design forces motivate the adoption of a shared repository (workspace) for heuristic activity, updated indirectly via heuristics that can enjoy an arbitrary degree of autonomy.

The formulation as described facilitates the removal of the hard distinction between online and offline activity, relegating them instead to a higher-order form of "exploration versus explotation". Thus, clients of hyper-heuristics (researchers and end-users) are not required to work within a single statically-determined compartment of the design-space — in this sense, all activities are online and compete to be allocated resources by the top-level controller. We hope in future work to demonstrate that this characterizes two key elements of domain-agnostic optimization: a) an abstractly utilitarian controller and b) a resource-allocation strategy that is 'grounded' in metrics such as CPU time and rate of improvement.

In architectural terms, this formulation also invites the relaxation of the domain barrier. While it has always been implicit that any concrete realisation of the domain barrier represents a particular point on the 'generality versus leverage' continuum, the default concrete example (as exemplified by [4]) is perhaps too-often perceived as mandatory. It is our hope that the revised formulation presented here will help to change the prevailing conceptual model of hyper-heuristics in this respect.

References

1. J. Denzinger, M. Fuchs, M. Fuchs, High Performance ATP Systems by Combining Several AI Methods, Tech. rep., University of Kaiserslautern (1997).
2. P. I. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III, PATAT '00, Springer-Verlag, London, UK, UK, 2001, pp. 176–190.
3. E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. R. Woodward, A classification of hyper-heuristics approaches, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, 2nd Edition, Vol. 57 of International Series in Operations Research & Management Science, Springer, 2010, Ch. 15, pp. 449–468.
4. J. Woodward, A. Parkes, G. Ochoa, A mathematical formalization of hyper-heuristics, Presented to the 'Workshop on Hyper-Heuristics' at 10th International Conference on Parallel Problem Solving From Nature (PPSN-08), Technische University Dortmund, Germany. (September 2008).
5. S. Peyton Jones, et al., The Haskell 98 language and libraries: The revised report, Journal of Functional Programming 13 (1) (2003) 0–255, http://www.haskell.org/definition/.
6. J. Swan, E. Özcan, G. Kendall, Hyperion - a recursive hyper-heuristic framework, in: C. Coello (Ed.), Learning and Intelligent OptimizatioN, Vol. 6683 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2011, pp. 616–630.
7. L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, J. Artif. Intell. Res. (JAIR) 4 (1996) 237–285.
8. F. Glover, Tabu Search - Part I, INFORMS Journal on Computing 1 (3) (1989) 190–206.
9. F. Glover, Tabu search - Part II, INFORMS Journal on Computing 2 (1) (1990) 4–32.
10. A. E. Eiben, Z. Ruttkay, Self-adaptivity for constraint satisfaction: Learning penalty functions, in: International Conference on Evolutionary Computation, 1996, pp. 258–261.
11. J. G. Marín-Blázquez, S. Schulenburg, A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients, in: IWLCS, 2005, pp. 193–218.
12. E. K. Burke, S. Petrovic, R. Qu, Case-based heuristic selection for timetabling problems, J. Scheduling 9 (2) (2006) 115–132.
13. E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J. R. Woodward, Exploring hyper-heuristic methodologies with genetic programming, in: C. L. Mumford, L. C. Jain (Eds.), Computational Intelligence, Vol. 1 of Intelligent Systems Reference Library, Springer, 2009, Ch. 6, pp. 177–201.
14. P. F. Stadler, Landscapes and their correlation functions, Journal of Mathematical Chemistry 20 (1996) 1–45, 10.1007/BF01165154.
15. P. F. Stadler, Towards a theory of landscapes, in: R. Lpez-Pena, R. Capovilla, R. Garca-Pelayo, H. Waelbroeck, F. Zertuche (Eds.), Complex Systems and Binary Networks, Vol. 461 of Lecture Notes in Physics, Springer-Verlag, Berlin, Heidelberg, New York, 1995, pp. 77–163.
16. W. Hordijk, A measure of landscapes, Evolutionary Computation 4 (4) (1997) 335–360.
17. C. R. Reeves, Landscapes, operators and heuristic search, Annals of Operations Research 86 (1999) 473–490.

18. C. R. Reeves, Fitness landscapes and evolutionary algorithms, in: AE '99: Selected Papers from the 4th European Conference on Artificial Evolution, Springer-Verlag, London, UK, 2000, pp. 3–20.
19. L. Kallel, B. Naudts, C. R. Reeves, Properties of fitness functions and search landscapes, Springer-Verlag, London, UK, 2001, pp. 175–206.
20. E. Weinberger, Correlated and uncorrelated fitness landscapes and how to tell the difference, Biological Cybernetics 63 (5) (1990) 325–336.
21. B. Hayes-Roth, A blackboard architecture for control, Artif. Intell. 26 (3) (1985) 251–321.
22. N. Carver, V. Lesser, The evolution of blackboard control architectures, Tech. rep., Amherst, MA, USA (1992).
23. D. A. Berry, B. Fristedt, Bandit Problems: Sequential Allocation of Experiments, Springer, 1985.
24. S. Dzeroski, L. Todorovski, Discovering dynamics: From inductive logic programming to machine discovery, Journal of Intelligent Information Systems 4 (1995) 89–108, 10.1007/BF00962824.
25. D. Ouelhadj, S. Petrovic, A cooperative hyper-heuristic search framework, Journal of Heuristics (2009) 1–2310.1007/s10732-009-9122-6.
26. R. Brooks, Intelligence without representation, Artificial Intelligence 47 (1991) 139–159.