Cogn Comput
DOI 10.1007/s12559-013-9201-8

Searching the Hyper-heuristic Design Space

Jerry Swan · John Woodward · Ender Özcan · Graham Kendall · Edmund Burke

Received: 25 June 2012 / Accepted: 7 January 2013

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-heuristics and to a division of responsibility that is suggestive of a blackboard architecture, in which individual heuristics annotate a shared workspace with information that may also be exploited by other heuristics. Such a framework invites consideration of the kind of relaxations of the domain barrier that can be achieved without loss of generality. We give a concrete example of this architecture with an application to the 3-SAT domain that significantly improves on a related token-ring hyper-heuristic.

Keywords Hyper-heuristics · Metaheuristics · Optimization · Machine-learning · Blackboard architecture

J. Swan (✉) · J. Woodward · E. Burke
Computing Science and Mathematics, School of Natural Sciences, University of Stirling, Stirling FK9 4LA, Scotland, UK
e-mail: jsw@cs.stir.ac.uk

J. Woodward
e-mail: jrw@cs.stir.ac.uk

E. Burke
e-mail: e.k.burke@stir.ac.uk

E. Özcan · G. Kendall
School of Computer Science, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
e-mail: exo@cs.nott.ac.uk

G. Kendall
e-mail: gxk@cs.nott.ac.uk

Introduction

It has been postulated that increasing the autonomy of artificial systems is a necessary step in tackling intractible, complex or real-world problems [1, 2, 33]. Such an increase in autonomy would help to address criticism of recent advances in Artificial Intelligence, in which it is argued that the real power of the technology lies in the hands of the programmers who adapt their design to suit the context of end use [3]. Such criticisms argue that the requirement for extensive programmer (or end user) intervention invalidates much of the benefit by demanding modification when the problem domain changes.

A greater measure of domain-independence is therefore a clear requirement. Such genericity is unarguably one of the defining characteristics of cognition, and this article gives an application of a cognitive architecture to the domain of *hyper-heuristics*—a methodology that is, to a large degree, motivated by the goal of establishing domain-independence in the context of search and optimization problems.

The structure of this paper can be outlined as follows: in "Hyper-heuristics", we describe a previous mathematical formulation of hyper-heuristics, and in "A Revised Formulation", we extend this formulation to allow autonomous exploration of the design space described in [4]. In Implications of this Formulation, we discuss a number of possible architectures for achieving such autonomy, giving a concrete example in "Blackboard Architectures" in terms of a *blackboard architecture* [5] together with some experiments demonstrating the effectiveness of this approach in "Experiments with a Blackboard Architecture".

Hyper-heuristics

The term *hyperheuristic* first appeared in [6] in the context of automated theorem proving. However, around 10–12 years ago the term became widely used to describe study of 'heuristics to *choose* heuristics' [7–9]. More recently, Burke et al. [4] classified diverse approaches under a more general framework that also includes the generation of new heuristics. We proceed to reflect this continuing generalization by extending the mathematical framework given by Woodward et al. [10]. In formulating a generalization, we also address some issues arising from the initial framework definition [10]. We then 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 [11]. For those unfamiliar with the Haskell language, knowledge of the elementary notion of function signatures is likely to suffice.

In [10], 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 interpreted 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:

$$\begin{aligned} H &: [Q] \to O \times \mathbb{N}^2 \\ q &\mapsto (o_i, (j, k)) \end{aligned} \tag{1}$$

(where square brackets denote a list), that is, 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 elect to make use of this position ordering). Figure 1 depicts the enforcement of the flow of information between the hyper-heuristic and problem layers.

We note that this formulation does not allow for O to vary during the course of the run, that is, 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. Mathematics and theoretical computer science are typically concerned with functions that are 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 [12]. As a minimum, it is therefore necessary to extend the signature of H to include a *workspace* parameter W:

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

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 of Q.

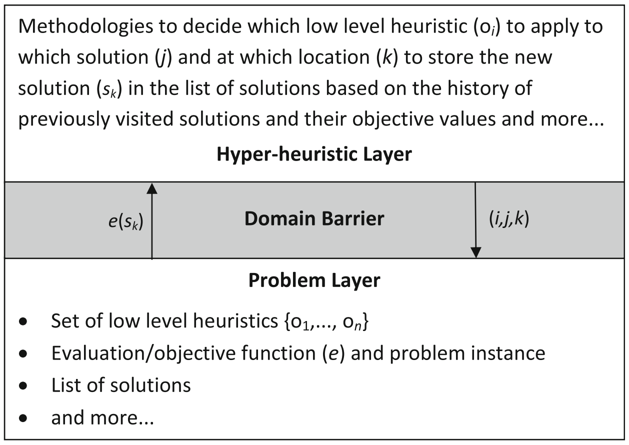
Methodologies to decide which low level heuristic (o_i) to apply to which solution (j) and at which location (k) to store the new solution (s_k) in the list of solutions based on the history of previously visited solutions and their objective values and more...
Hyper-heuristic Layer
e(s_k)
Domain Barrier
(i,j,k)
Problem Layer
Set of low level heuristics {o_1,..., o_n}
Evaluation/objective function (e) and problem instance
List of solutions
and more...

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

A Revised Formulation

In Burke et al. [4], the authors classify hyper-heuristics in two orthogonal dimensions. The first dimension denotes selection versus generation and the second dimension expresses the source of feedback during learning (online, offline or none). Further categorization is possible by considering whether the search space is constructive or perturbative. 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. [4] note the emerging trend for hybrid methodologies that seek to address 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 achieving significant further hyper-heuristic autonomy 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 that are unlikely unless the discovered solution strategy is dictated *a priori* by the end user. For example, the output of a constructive heuristic might serve as the starting point for some subsequent perturbative heuristic. Whilst this is an extremely elementary example, in much of current search methodology, such hybridizations are traditionally obtained manually and generally require explicit software development effort. We proceed to describe a unifying formulation intended to facilitate such dynamic exploration. 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

$$\textit{perturb} : S \rightarrow S,$$

that is, a mapping from state to state for some generic state type S, with constructive heuristics as

$$\textit{construct} : P \rightarrow \textit{Maybe}\ P$$

where P is some generic notion of partial state. The *Maybe* syntax for *construct* expresses the notion that constructive heuristics are in general partial (as opposed to total) functions (e.g. a stochastic or greedy construction may fail to extend a partial solution).

The above classification means that the primitive heuristics present a number of distinct function signatures to the hyper-heuristic level (taken from the Cartesian product of selection\generation and constructive\perturbative). In fact, multiple concrete representations of solution-state are clearly possible (e.g. list versus graph for the Travelling Salesman Problem). 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 Wheeler's famous maxim (often wrongly attributed to Dijkstra), we resolve this issue via an additional level of indirection [13]. Specifically, we mandate a signature for both types of heuristic that takes a *single* workspace argument. Hence, we now have $\textit{perturb} : W \rightarrow W$ and $\textit{construct} : W \rightarrow W$.

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

$$h'_0 : W \rightarrow W,\ h'_1 : W \rightarrow W,$$

where the parameter W is a workspace that is now taken to be the repository for 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 relevant 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. The implementor of a heuristic may therefore elect to read and write the representation of their choice to and from the workspace.

Implications of this Formulation

Since $h_0{}'$ and $h_1{}'$ share a common signature, our definition of hyper-heuristic is now *recursive*—the list of heuristics $\mathcal{H}$ that $h_1{}'$ accesses from the workspace may themselves be hyper-heuristics. Hyper-heuristic invocations may thus be recursively nested to an arbitrary depth. Such an aggregation mechanism facilitates the generation of new hyper-heuristics, with aggregation being possible offline (i.e. in a fixed, user-specified manner) or online (i.e. adaptively). An equivalent formulation for recursion (with attendant aggregation facilities) has been implemented in the Hyperion hyper-heuristic solution-domain framework [14].

One possible criticism of this formulation might be that simplicity of (1) has been lost. Even apart from the limitations observed in "Introduction" on the metaheuristics expressible in this formulation, it is our contention that this alleged simplicity comes at the expense of an overly restricted notion of the domain barrier. In particular, the use of the workspace as a repository of shared information 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 make use of workspace information contributed by other heuristics. 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-independent fashion is the notion of the *inverse* of a perturbative heuristic—such information can profitably be used in tabu-lists, for example [15].

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 end 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’ approaches 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.

Learning

Up to this point, the ‘learning’ dimension of the design space has not been directly addressed. The success of learning is often proportionate 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 [16], tabu-search [15, 17] and adaptive penalties [18]. 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 include learning-classifier systems [19], case-based reasoning [20] and genetic programming [21]. While much work has been done in the offline analysis of landscapes and devising metrics of landscape quality [22–27], this information is not routinely used to inform online activity. As a simple but concrete example, the autocorrelation length of the landscape [28] could be used to inform parameters such as tabu-tenure or population size, but such interplay between design and experiment is not currently 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, that is, information produced online (sampling the search space) is likely to be of use offline (e.g. to inform a subsequent heuristic generation process). It is therefore our belief that a hard distinction between online and offline learning is a significant obstacle to progress. We are also of the opinion that this issue (as with many in the current state of meta-heuristic research) is much more of an architectural\software engineering issue than a conceptual one. We believe that this gives further motivation for an integrated exploration of the design space.

Options and Responsibilities for Top-Level Control

We briefly consider here some options in the specific context of hyper-heuristic control. If sufficiently abstracted from the underlying problem domain, it is possible to view the very topmost hyper-heuristic control mechanism as solving the multi-armed bandit problem [29] 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 trade-offs (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 [30]. 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. One example of such a system for meta-heuristics is MAGMA [31]. In the domain of hyper-heuristics, co-operation between low-level heuristics is explored in [32], 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 autonomy' invites the possibility of a subsumption-architecture approach in which the system has no conventional top-down control mechanism [33], but which can nonetheless be *grounded* (in the "good-enough, fast-enough, cheap-enough" sense [34]) via the choice of suitable parameters for the heuristic agents such as 'time-out' etc.

Blackboard Architectures

In concrete implementation terms, the decentralization of concerns discussed in this article is an essential characteristic of a *blackboard architecture* [5]. Informally, a blackboard architecture can be considered as a team of experts (termed 'agents' or 'knowledge sources') working together via a physical blackboard on which they are each contributing partial elements of a shared solution. In [35], Booch et al. describe the application of a blackboard architecture to a cryptanalysis problem, with agents having expertise at the different hierarchical levels of letters, words and sentences. One property enjoyed by modern blackboard architectures is concurrency, and numerous control variants are discussed in [36]. Such indirectly mediated interaction facilitates the generation of an autonomous search process that arises from the initial set of (hyper-)heuristics [3].

The overall intention is to integrate agents (i.e. heuristics in this case) to obtain solutions not necessarily readily achievable by any single agent. The addition of a shared workspace adds a social element to the search process in a manner analogous to the stigmergic behaviour seen in some insect species, prominent examples of metaheuristics inspired by social interaction being Ant Colony Optimization and Particle Swarm Optimization [37].

However, a blackboard architecture is somewhat more general than either of the above since (in addition to the same rich representation of partial solution-state possible in Ant Colony Optimization) the generic nature of the workspace also facilitates a) the incorporation of arbitrary information about the problem domain in declarative form and b) the calculation of computationally expensive metrics by 'background' agents. In particular respect of a), blackboard architectures enjoy a higher degree of domain-independence. In the section entitled "Experiments with a Blackboard Architecture", we use a blackboard architecture to adaptively set metaheuristic parameters from information typically obtained offline—incorporating such information into Particle Swarm Optimization might clearly also prove useful, but the core Particle Swarm Optimization algorithm would then have to be extended to include something like a shared workspace in any event.

Experiments with a Blackboard Architecture

As a simple but concrete example of the utility of the blackboard architecture, we demonstrate the interleaving of (what are traditionally considered to be) online and offline activities with an application to the well-known boolean satisfiability problem k-SAT. We use 3 variants of the Simulated Annealing metaheuristic [38] with a geometric annealing schedule, instances of which are parameterized by a temperature range obtained by sampling the state space [39]. The problem instances are the first 30 of the Uniform Random 3-SAT UF250 SATLIB problem set[1] [40]. All instances have 250 variables and 1065 clauses and are known to be satisfiable. The fitness function employed is the number of unsatisfied clauses, with the neighbourhood of a state s being all states within a Hamming distance of 1 from s. With such elementary heuristics, there is clearly no possibility of competing with state-of-the-art SAT solvers, which often have highly sophisticated feature discrimination in their fitness function and\ or employ an algorithm portfolio approach [41]. The 3 Simulated Annealing variants have the common hyper-heuristic algorithm template given in Listing 1.1. The algorithm described by the template is based on the well-known token-ring metaheuristic [42]. The variants (token-ring, blackboard and proportional) are distinguished by their implementation of the functions initializeMetaheuristics and chooseMetaheuristic, defined as follows:

Token-Ring

initializeMetaheuristics constructs 10 Simulated Annealing metaheuristics, each having its own statically determined temperature range, that is, obtained by an offline random walk. Hence, although initializeMetaheuristics is invoked multiple times, the metaheuristics it returns are the same for each invocation. chooseMetaheuristic returns the i-th metaheuristic.

[1] http://www.cs.ubc.ca/hoos/SATLIB/benchm.html

Proportional

initializeMetaheuristics As for token-ring. chooseMetaheuristic proportionally selects a metaheuristic according to the maximum fitness value of the random walk used to generate it.

Blackboard

initializeMetaheuristics As discussed above, when a metaheuristic is invoked, it writes its search trajectory into the workspace—for current purposes, we can consider a trajectory to be a sequence of fitness values. We construct 10 Simulated Annealing metaheuristics, each having a temperature range that is dynamically determined as follows. The blackboard controller proportionally selects a trajectory according to the maximum fitness value it contains. These trajectories are then used to derive temperature range and hence the metaheuristic instances. chooseMetaheuristic returns the *i*-th metaheuristic.

```
tokenRingHH()
begin
    $currentState \leftarrow initialState()$;
    $currentValue \leftarrow numUnsatisfiedClauses(currentState)$;
    $(bestState, bestValue) \leftarrow (currentState, currentValue)$;
    $numUnimprovingTrials \leftarrow 0$;
    do
        $metas \leftarrow initializeMetaheuristics()$;
        for ( $i \leftarrow [0 \ldots \#metas)$ )
        begin
            $alg \leftarrow chooseMetaheuristic(metas, i)$;
            $currentState \leftarrow alg.search(currentState)$;
            $currentValue \leftarrow numUnsatisfiedClauses(currentState)$;
            if ( $currentValue < bestValue$ )
                $(bestState, bestValue) \leftarrow (currentState, currentValue)$;
            else
                $numUnimprovingTrials \leftarrow numUnimprovingTrials + 1$;
        end
    while ($numUnimprovingTrials \leq MAX\_UNIMPROVING\_TRIALS$) ;

    $print(bestState, numUnsatisfiedClauses(bestState))$;
end
```

Listing 1.1. Token-Ring Hyper-heuristic template.

Table 1 presents the average heuristic values obtained from 100 applications of the three Simulated Annealing variants to the 30 problem instances, with MAX_UNIMPROVING_TRIALS equal to 10. Execution times for the three variants were not significantly different, averaging in each case at around 80 seconds across all trials. The blackboard variant was determined (via Wilcoxon signed-rank followed by Friedman tests, at a significance level of 0.05) to yield significantly better results for all instances, with a consistent improvement of the order of 10%. We reiterate that these results are intended to illustrate what can readily be achieved (both conceptually and in implementation terms) by agent intercommunication, rather than as a study in contemporary Simulated Annealing practice. More ambitious interaction between online and offline activity is clearly possible, with the potential for notifying\activating online agents when some expensive offline calculation (e.g. recalculating fitness-distance correlation measures [43]) is completed and added to the workspace. The entire framework (generic blackboard architecture, Simulated Annealing metaheuristics plus SAT domain specifics) is implemented in the Java^{TM} programming language in under 1,500 lines of code. Since blackboard architectures are well-understood in software engineering terms, the implementation is relatively simple and is likely to be further simplifiable via more powerful concurrency mechanisms (e.g. actor-based message-passing) than are natively supported in Java. As demonstrated by the equivalent instance timings, the additional mechanisms required for the blackboard infrastructure do not introduce a performance penalty.

Table 1 $(\bar{x}, \sigma)$ of average fitness over 50 trials for 3 Simulated Annealing variants on 30 3-SAT instances

Instance	Proportional	Blackboard	Token-ring	Instance	Proportional	Blackboard	Token-ring
UF250-01	(64.90, 5.89)	**(56.46, 6.64)**	(65.18, 6.25)	UF250-016	(66.72, 7.44)	**(60.96, 7.37)**	(67.68, 7.13)
UF250-02	(68.62, 6.71)	**(63.92, 5.93)**	(67.8, 6.51)	UF250-017	(67.54, 7.2)	**(61.18, 5.93)**	(68.34, 5.97)
UF250-03	(65.72, 6.62)	**(59.56, 8.29)**	(66.92, 5.94)	UF250-018	(66.82, 7.73)	**(57.36, 8.71)**	(66.76, 5.96)
UF250-04	(65.94, 7.92)	**(56.66, 8.70)**	(65.52, 7.48)	UF250-019	(66.98, 6.63)	**(58.4, 7.07)**	(65.06, 7.43)
UF250-05	(66.12, 6.98)	**(60.18, 7.26)**	(67.94, 6.72)	UF250-020	(66.20, 7.14)	**(60.86, 7.77)**	(65.36, 7.11)
UF250-06	(65.38, 6.90)	**(60.16, 7.17)**	(66.86, 5.72)	UF250-021	(65.26, 6.65)	**(57.56, 7.74)**	(65.22, 7.51)
UF250-07	(65.68, 6.78)	**(60.98, 8.15)**	(64.54, 5.78)	UF250-022	(66.98, 6.62)	**(61.64, 6.44)**	(66.62, 6.68)
UF250-08	(65.86, 7.42)	**(56.46, 6.80)**	(64.64, 7.20)	UF250-023	(65.88, 5.93)	**(59.8, 7.68)**	(66.34, 6.64)
UF250-09	(67.24, 6.89)	**(61.22, 7.04)**	(65.66, 6.12)	UF250-024	(68.52, 8.48)	**(59.92, 7.39)**	(68.10, 6.01)
UF250-010	(63.46, 6.42)	**(58.96, 7.55)**	(66.18, 8.30)	UF250-025	(64.42, 8.34)	**(59.36, 6.24)**	(64.80, 6.58)
UF250-011	(68.22, 6.38)	**(59.54, 7.91)**	(69.34, 6.64)	UF250-026	(66.36, 7.57)	**(60.24, 6.95)**	(67.42, 6.24)
UF250-012	(65.52, 6.25)	**(59.24, 6.44)**	(64.90, 7.42)	UF250-027	(67.12, 6.77)	**(61.3, 7.29)**	(68.54, 6.14)
UF250-013	(65.98, 6.08)	**(60.36, 7.09)**	(64.22, 6.74)	UF250-028	(64.46, 6.95)	**(59.20, 6.42)**	(66.06, 7.48)
UF250-014	(68.42, 7.98)	**(62.74, 7.49)**	(67.62, 6.92)	UF250-029	(66.78, 6.82)	**(59.04, 6.96)**	(66.66, 7.48)
UF250-015	(65.22, 6.80)	**(59.54, 7.52)**	(66.12, 6.28)	UF250-030	(67.06, 6.16)	**(61.62, 6.38)**	(68.84, 7.87)

Bold values indicate that they were superior to other results

Conclusion

Metaheuristic researchers are enthusiastic in adopting metaphors of computation from the natural world. The human mind is perhaps the most effective (known) application of natural parallel computation to problem solving, so it is somewhat surprising that cognitive architectures have done little to inform hyper-heuristic activity.

We have described a unifying 'mathematical' formulation for hyper-heuristics and proceeded to show that 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 "intensification versus diversification". 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 for resources allocated by the top-level controller. This 'loose coupling' mechanism facilitates both heuristic interoperability and 'racing' to determine better heuristic configurations [44].

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 specific realization of the domain barrier represents a particular point on the 'generality versus leverage' continuum, the default concrete example (as exemplified by [8]) 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.

Reference

1. Bishop JM, Erden YJ. Computational creativity, intelligence and autonomy. Cognit Comput. 2012;4(3):209–1.
2. Kendall G, Su Y. Imperfect evolutionary systems, Evolutionary Computation. IEEE Trans. 2007;11(3):294–7 doi:10.1109/TEVC.2006.887348.
3. d'Inverno M, Luck M. Creativity through autonomy and interaction. Cognit Comput. 2012;4(3):332–46.
4. Burke EK, Hyde M, Kendall G, Ochoa G, Ozcan E, Woodward JR. A classification of hyper-heuristics approaches. In: Gendreau M, Potvin J-Y, editors. Handbook of metaheuristics, 2nd Edition, vol 57 of international series in operations research & management science. Berlin:Springer; 2010. Ch. 15, p. 449–68.
5. Hayes-Roth B. A blackboard architecture for control. Artif Intell. 1985;26(3):251–21.
6. Denzinger J, Fuchs M, Fuchs M. High performance ATP systems by combining several AI methods, Tech. rep., University of Kaiserslautern 1997.
7. Burke EK, Kendall G, Soubeiga E . A tabu-search hyperheuristic for timetabling and rostering. J Heuristics. 2003;9(6):451–70.
8. Rattadilok P, Gaw A, Kwan RK. Distributed choice function hyper-heuristics for timetabling and scheduling. In: Burke E, Trick M, editors. Practice and theory of automated timetabling, vol. 3616 of Lecture Notes in Computer Science. Springer: Berlin; 2005. p. 51–67.

9. Cowling P, Chekhlevitch K. Hyperheuristics for managing a large collection of low-level heuristics to schedule personnel. In: The 2003 congress on evolutionary computation (CEC '03), vol. 2; 2003. p. 1214–21.
10. Woodward J, Parkes A, Ochoa G. 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).
11. Peyton Jones S, et al. The Haskell 98 language and libraries: the revised report. J Funct Program. 2003;13(1):0–255 http://www.haskell.org/definition/
12. Battiti R, Tecchiolli G. The reactive tabu search. INFORMS J Comput 1994;6(2):126–40.
13. Spinellis D. Another level of indirection. In: Oram A, Wilson G editors. Beautiful code: leading programmers explain how they think. Sebastopol: O'Reilly and Associates; 2007. Ch. 17, p. 279–91.
14. Swan J, Özcan E, Kendall G. Hyperion - a recursive hyper-heuristic framework. In: Coello C editors. Learning and intelligent optimization, Vol. 6683 of Lecture Notes in Computer Science. Springer: Berlin; 2011. p. 616–30.
15. Glover F. Tabu search-Part I. INFORMS J Comput 1989;1(3): 190–206
16. Kaelbling LP, Littman ML, Moore AW. Reinforcement learning: a survey. J Artif Intell Res (JAIR) 1996;4:237–85.
17. Glover F. Tabu search-Part II. INFORMS J Comput 1990;2(1): 4–32.
18. Eiben AE, Ruttkay Z. Self-adaptivity for constraint satisfaction: learning penalty functions In: International conference on evolutionary computation. 1996, p. 258–61.
19. Marín-Blázquez JG, Schulenburg S. A hyper-heuristic framework with xcs: learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In: IWLCS, 2005, p. 193–218.
20. Burke EK, Petrovic S, Qu R. Case-based heuristic selection for timetabling problems. J Sched. 2006;9(2):115–32.
21. Burke EK, Hyde MR, Kendall G, Ochoa G, Ozcan E, Woodward JR. Exploring hyper-heuristic methodologies with genetic programming. In: Mumford CL, Jain LC editors. Computational intelligence, Vol. 1 of intelligent systems reference library. Berlin:Springer; 2009. Ch. 6, p. 177–201.
22. Stadler PF. Landscapes and their correlation functions. J Math Chem. 1996;20:1–45. doi:10.1007/BF01165154.
23. Stadler PF. Towards a theory of landscapes. In: Lpez-Pena R, Capovilla R, Garca-Pelayo R, Waelbroeck H, Zertuche F editors. Complex systems and binary networks, Vol. 461 of Lecture notes in physics. Berlin: Springer; 1995. p. 77–163.
24. Hordijk W. A measure of landscapes. Evol Comput 1997;4(4): 335–60.
25. Reeves CR. Landscapes, operators and heuristic search. Ann Oper Res 1999;86:473–90.
26. Reeves CR. Fitness landscapes and evolutionary algorithms. In: AE '99: selected papers from the 4th European conference on artificial evolution. London: Springer; 2000. p. 3–20.
27. Kallel L, Naudts B, Reeves CR. Properties of fitness functions and search landscapes. London: Springer; 2001. p. 175–206.
28. Weinberger E. Correlated and uncorrelated fitness landscapes and how to tell the difference. Biol Cybern 1990;63(5):325–36.
29. Berry DA, Fristedt B. Bandit problems: sequential allocation of experiments. Berlin: Springer; 1985.
30. Dzeroski S, Todorovski L. Discovering dynamics: From inductive logic programming to machine discovery. J Intell Inf Syst 1995;4:89–108 doi:10.1007/BF00962824.
31. Milano M, Roli A. Magma: a multiagent architecture for metaheuristics, systems, man, and cybernetics, part B: cybernetics. IEEE Trans. 2004;34(2):925–941 doi:10.1109/TSMCB.2003.818432.
32. Ouelhadj D, Petrovic S. A cooperative hyper-heuristic search framework. J Heuristics 2009;1–23 doi:10.1007/s10732-009-9122-6.
33. Brooks R. Intelligence without representation. Artif Intell 1991;47:139–59.
34. Burke E, Kendall G, Newall J, Hart E, Ross P, Schulenburg S. Hyper-heuristics: an emerging direction in modern search technology. In: Glover F, Kochenberger G, Hillier FS, editors. Handbook of Metaheuristics, Vol. 57 of international series in operations research and management science. New York: Springer; 2003. p. 457–74.
35. Booch G, Maksimchuk RA, Engle MW, Young BJ, Connallen J, Houston KA. Object-oriented analysis and design with applications, third edition, ACM SIGSOFT software engineering notes 33(5).
36. Carver N, Lesser V. The evolution of blackboard control architectures, Tech. rep., Amherst, USA; 1992.
37. al Rifaie MM, Bishop JM, Caines S. Creativity and autonomy in swarm intelligence systems. Cognit Comput 2012;4(3):320–31.
38. Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220:671–80.
39. White S. Concepts of scale in simulated annealing. In: Proceedings of international conference on computer design; 1984, p. 646–51.
40. Hoos HH, Stützle T. SATLIB: An online resource for research on SAT. In: Gent IP, Maaren Hv, Walsh T, editors. SAT 2000, SATLIB is available online at http://www.satlib.org 2000.
41. Xu L, Hutter F, Hoos HH, Leyton-Brown K. Satzilla: portfolio-based algorithm selection for sat. J Artif Int Res 2008;32(1):565–606.
42. Gaspero LD, Schaerf A. Easylocal++: an object-oriented framework for the flexible design of local-search algorithms. Softw Pract Exper 2003;33(8):733–765.
43. Jones T, Forrest S. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the sixth international conference on genetic algorithms. Morgan Kaufmann; 1995. p. 184–92.
44. Birattari M, Stützle T, Paquete L, Varrentrapp K. A racing algorithm for configuring metaheuristics. In: Proceedings of the genetic and evolutionary computation conference, GECCO '02. San Francisco :Morgan Kaufmann Publishers Inc.; 2002. p. 11–18.