Comparing Hyper-heuristics with Blackboard Systems

Kevin Graham
University of Stirling
kgr@cs.stir.ac.uk

Leslie Smith
University of Stirling
l.s.smith@cs.stir.ac.uk

ABSTRACT

This paper aims to draw a comparison between the traditional view of hyper-heuristics and a lesser known type of multi-agent system known as a blackboard system. Both approaches share many similarities in both implementation and philosophy but also have several important differences in terms of characteristics and approach, such as a difference in control scheme. To investigate the consequences of the perceived differences, both approaches are decomposed into their constituent parts and compared with a focus on the perceived strengths and weaknesses of adopting one methodology over the other.

CCS CONCEPTS

•Computing methodologies → Search methodologies; Multi-agent systems;

KEYWORDS

hyper-heuristics, blackboard systems

ACM Reference format:
Kevin Graham and Leslie Smith. 2017. Comparing Hyper-heuristics with Blackboard Systems. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 5 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082055

1 INTRODUCTION

Hyper-heuristic techniques have become a successful methodology for the combination of existing heuristic techniques into a single framework, allowing each to apply its various strengths towards searching the space of possible solutions to a given problem. Hyper-heuristics have gained a strong community of researchers since its introduction by Cowling et al. in [7]. A far less recent type of system, a multi-agent system initially developed for speech recognition, has slowly been regaining momentum as a general approach to problem solving after a decline in further research occurring in the 1990s. On the surface, both approaches appear to share many similarities, such as; the separation of domain-knowledge into a well-defined set of 'active entities' and the use of a single, globally accessible workspace on which solution objects and related information reside, etc. There are also several differences, some superficial, and others not so.

In [12], the authors have stated the need for new hyper-heuristic techniques that are capable of making better use of richer forms of domain knowledge - a characteristic that is exemplified in blackboard systems. It is hoped that through the comparison of the various differences drawn between these two types of system, that other possible benefits in using components and concepts from one system as part of the other can be uncovered and discussed.

2 BACKGROUND

2.1 Hyper-heuristics

As opposed to metaheuristics that search within the space of solutions to a given problem, hyper-heuristics are a high-level methodology that search within a space of lower-level heuristics - it is these heuristics that then search a problems solution-space. According to Burke et al. [3], the term hyper-heuristic was first coined in by Cowling et al. in a 2001 conference paper [7] which was then further developed for scheduling problems in operation research. These publications described hyper-heuristics as high-level techniques that when given an appropriate set of heuristics and a target problem instance, can select and execute an appropriate heuristic from the set at each decision point [3]. However, Burke et al. [4] define hyper-heuristics more generally as, "*… a search method or learning mechanism for selecting or generating heuristics to solve computational search problems*".

From this standpoint, hyper-heuristic techniques can be seen to fall into one of 2 main categories; (i) selective hyper-heuristics, and (ii) generative hyper-heuristics. Selective hyper-heuristics, simply select from a set of pre-implemented heuristics - these heuristics can be further delineated as constructive or perturbative depending on whether a heuristic constructs a solution from candidate partial-solutions or perturbs elements of a complete solution to generate a new candidate. On the other hand, generative hyper-heuristics use techniques for generating new heuristics by combining components of previously decomposed existing heuristics [4].

In terms of structure, a basic hyper-heuristic can be thought of being comprised of 3 main components: The Hyper-heuristic layer, the problem layer and the set of low-level heuristics to apply within the problem layer. In more complex hyper-heuristics, this structure can also include a feedback layer that encapsulates various forms of online or offline learning. This structure can be seen in Fig. 1 - as derived from [13] - where $\{i, j, k\}$ represents a 3-tuple vector, interpreted in the problem layer as: apply heuristic i to the solution j from the list of solutions and store the resulting solution S_k in the list at index k. The label f(s_k) represents the fitness of the new solution s_k [13].

2.1.1 *Hyper-heuristic layer*

The hyper-heuristic layer is responsible for the storage and organisation of all required domain-independent knowledge, such as: the number of heuristics in the heuristic set, the use of one objective

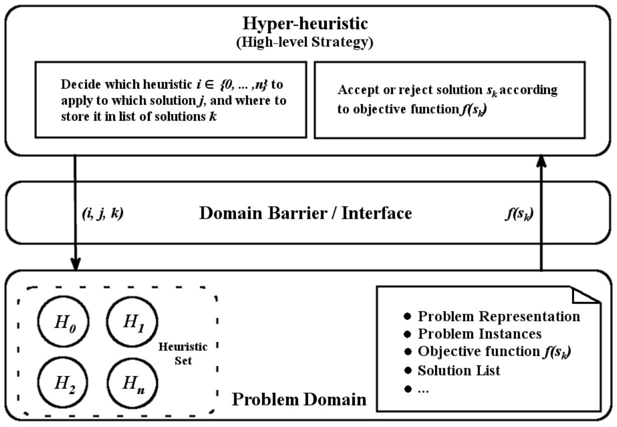
Hyper-heuristic
(High-level Strategy)
Decide which heuristic $i \in \{0, \dots, n\}$ to apply to which solution j, and where to store it in list of solutions k
Accept or reject solution s_k according to objective function $f(s_k)$
(i, j, k)
Domain Barrier / Interface
$f(s_k)$
H_0
H_1
H_2
H_n
Heuristic Set
Problem Domain
● Problem Representation
● Problem Instances
● Objective function $f(s_k)$
● Solution List
● ...

Figure 1: Basic structure of a hyper-heuristic set-up [13]

function over another, the distance between two solutions etc. [2]. The hyper-heuristic layer possesses no knowledge of the problem domain, meaning that a hyper-heuristic can tackle new problems by simply replacing the heuristic set and objective function [2].

2.1.2 *Domain Barrier*

The domain barrier defines an interface between the hyper-heuristic layer and the supplied set of heuristics. The barrier represents the fact that the hyper-heuristic layer should maintain no knowledge of the domain in question and should only be aware of the set of low-level heuristics that can be used against a given problem instance [2].

2.1.3 *Problem Domain Layer*

The problem domain in a hyper-heuristic system can be viewed as the workspace on which solutions are placed or modified collaboratively by the supplied set of heuristics in order to solve a given problem instance. This layer also represents the storage area for all knowledge pertaining to the domain, including but not limited to: problem representation, problem instances, list of solutions and the objective function or functions [13].

2.1.4 *Heuristic Set*

The set of domain specific heuristic maintained in the problem layer can be thought of as 2 distinct subsets: those that build full solutions piece-wise from partial solutions - referred to as constructive heuristics - and those that modify existing complete solutions in order to generate new complete solutions - perturbative heuristics. As such, the set of heuristics can be considered as the 'active agents' in a hyper-heuristic that create new solution objects within the problem layer.

Depending on the hyper-heuristic approach used, selective vs. generative, heuristics exhibit varying levels of mutual independence. In a purely selective scheme, heuristics have the most independence from other heuristics as they maintain their own internal state and are self contained entities that do not require the assistance of other heuristics in making changes on the problem layer. On the other hand, in a generative hyper-heuristic, decomposed heuristic components are no longer self contained in the sense that they can no longer produce any useful work without first being combined with other heuristic components. In terms of system independence, all heuristics tend to be driven entirely from the point of view of the hyper-heuristic layer - only becoming active as the result of a direct call to execute from the hyper-heuristic layer. As will be discussed in section 4.2.1, this style of controlling a systems active agents is at odds with the control philosophy of blackboard systems.

2.2 Blackboard Systems

Blackboard systems are a type of multi-agent problem solving system arising from the ideas presented as part of the Hearsay-II speech understanding system [8]. In the years since Hearsay-II, blackboard systems have been used successfully to solve a variety of complex and under-specified problems for which the solution strategy is not well defined.

The blackboard model of problem solving derived from Hearsay-II is comprised of 3 primary components, namely: a blackboard data-structure, knowledge sources (agents) and a main control shell (often referred to as the control component).

A blackboard system maintains a collection of knowledge sources and a globally accessible user-defined 'blackboard' data structure which contains all information pertaining to the state of the search process as well as other types of information that may help to guide the search process. Knowledge sources communicate solely through the addition or modification of 'blackboard objects' on the blackboard - in doing so, other knowledge sources can be guided to contribute to the solution in progress. Fig.2 shows the basic structure of a blackboard system as derived from the Hearsay-II system.

2.2.1 *The Blackboard*

The blackboard is a user-defined data structure that holds the state of problem solving as well as other information useful in guiding the search process. As shown in Fig.2, the blackboard is often delineated into a number of levels known as the abstraction hierarchy, where each level contains the objects and information pertaining to different levels of problem analysis. Given that the solution to a large problem may involve many blackboard objects, the blackboard structure also provides an efficient means of locating and retrieving information on the blackboard as related objects can be grouped together on the same level.

2.2.2 *Knowledge Sources*

Knowledge sources (or KSs) represent the domain knowledge of a system and are the entities which create or make changes to blackboard objects. Each KS consists of a condition-action pairing where the KS is considered 'triggered' based on an event on the blackboard satisfying its condition part allowing it then to execute its action part involving the creation or modification of one or more objects on the blackboard. KSs are independent entities in the sense that each KS does not require the assistance from any other KS in making its contribution, further, the internal representation, state and implementation details are unavailable to other KSs [6]. This independence of knowledge and representation highlights a

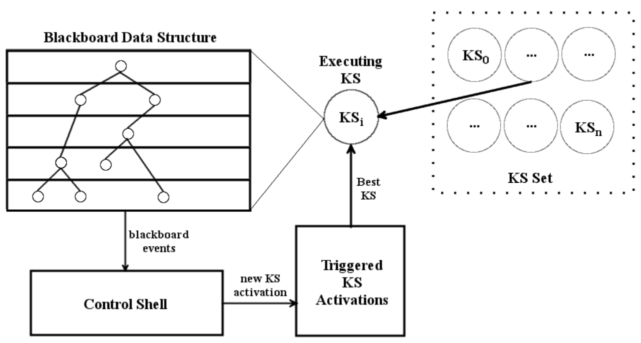

Figure 2: Basic Components of a Hearsay-II Style Blackboard System - derived from Corkill in [6]

strength of the blackboard model - that heterogeneous operators are able to co-exist and collaborate together on a problem within a blackboard system.

2.2.3 The Control Shell

The control shell, or control component as it's sometimes known, is responsible for organising the flow of problem solving processes and computational resources by allowing KSs to respond opportunistically to blackboard events but controlling when each KS can execute its action part on the basis of the state of the blackboard and/or immediate focus-of-attention [6]. The control shell must be able to carry out this responsibility without possessing any of the diverse knowledge contained within the set of KSs.

As a common control scheme, the control shell can collect and prioritise the events from a currently executing KS which it then uses to trigger the condition parts of other KSs. These triggered KSs are then ranked and the most appropriate KS based on the triggering events priority is allowed to execute [6].

3 PROPERTIES OF BLACKBOARD SYSTEMS

The following paragraphs describe several of the more desirable defining properties of blackboard systems, that combine to produce powerful and flexible problem solving systems.

Modularity. The loose-coupling between the knowledge sources and the control in a blackboard system means that new control strategies and knowledge sources can be added to the system, even during runtime, without the need to modify the representation of domain knowledge or the internal implementation of existing components [9].

Interoperability of Operators (KSs) and Solution Representations. Blackboard systems are built on the principle of syntactic interoperability, which is defined as the ability of multiple systems or sub-systems to communicate and exchange information through the use of a pre-determined format defined at design time. This syntactic interoperability is facilitated through the use of a standardised format for the objects on the blackboard. Without KSs being required to understand the internal mechanics of every other KS in the system, they are able to collaborate with others in building a shared solution representation [9].

Persistence of Blackboard Objects. In contrast to traditional hyper-heuristic approaches, objects and partial solutions placed on the blackboard at any point during runtime remain there until the system terminates. Even when objects are modified or new solutions or partial solutions are built from existing ones, a copy of the original object(s) remain. This is facilitated by the way knowledge sources typically build solutions on the blackboard - where new solutions are constructed by forming links between other solution objects on the blackboard, often from different abstraction levels, leaving the linked objects intact and still allowing them to be further linked to other new objects at any point during the systems execution. The consequence of this is that solution objects found to be of little interest during the beginning stages of the search process can become useful and subsequently exploited as the search progresses.

Support for the Dynamic Introduction of Knowledge. When new and relevant sources of knowledge become available during runtime, such as machine learning processes on the state of the search trajectory that may take time and resources to compute - these can be 'hot-plugged' into the system via the introduction of new blackboard objects subsequently triggering KSs that can make use of this new information [9].

Distributed Blackboard Systems. As KSs in a blackboard system are not limited in terms of their heterogeneity, each can consist of an entire system or sub-system that may even be running of separate hardware, including both local and remote.

Support for Concurrency. The traditional blackboard system supports a form of pseudo-concurrency, in the sense that KSs actions are interleaved as the control shell prioritises their potential contribution against the current search focus. Although it is indeed true that the blackboard model does not disallow the notion of implementing true concurrency, other researchers have been studying the use of parallel blackboard systems [5] [1] - by exploiting the distributed nature of the model.

Multi-level Search. The blackboard model facilitates support for multi-level search through the delineation of the blackboard into a number of levels. Many problems exist that can be broken up into sub-problems that may be solved individually to arrive at a complete solution - the Travelling Salesman Problem (TSP) being one obvious example. Booch et al. in [1] describe a blackboard system developed to solve a relatively trivial cryptanalysis problem, where a collection of KSs have expertise at different abstraction levels representative of the sub-problems to be solved - specifically KSs with expertise at the letter substitution level, the word substitution level and at the level of sentence structure [9].

Supports Injection of Rich Domain Knowledge. As stated by Özcan et al. in [12], there is currently a need for new hyper-heuristic approaches that can operate on wider, more enriched sources of domain knowledge. This idea is reiterated in [11] which also provides two examples of hyper-heuristic systems which can make use of arbitrary domain knowledge, namely the blackboard system of Swan et al [13] and the multi-agent system by Martin et al [10]. In

[1]Even since the publication of this paper, the parallelisation of blackboard systems still remains an under-researched topic

conjunction with the support for the dynamic introduction of new knowledge and the fact that blackboard systems are well suited to the application of this knowledge, blackboard systems also exemplify representational flexibility - in that the blackboard model places no restriction to the type and scale of the information that can be represented in the blackboard data structure. [2]

4 BLACKBOARD SYSTEMS VS. HYPER-HEURISTICS

Despite being developed decades apart, blackboard systems and hyper-heuristics share many similarities but also have several differences. Presented here is a discussion of several of these similarities, beginning with some of the various similarities between approaches.

4.1 Similarities

Similarities between the two types of system include: the use of a high-level control strategy, domain knowledge separated into a set of heuristics/KSs, use of a single, globally accessible workspace and the ability to inject online and offline feedback into the search process.

4.2 Differences

Many of the differences between these two systems are cosmetic, such as the use of different terminology for what can essentially be considered as the same thing e.g, heuristics vs. KSs, the problem level vs. the blackboard data structure - however a couple of differences exist that not only show a difference of approach but may also highlight potential benefits of one system over the other.

4.2.1 Difference of Control

Where hyper-heuristics maintain direct control of its active entities, including when they are called, where they should be applied and where the result of any computation should be placed. Blackboard systems, on the other hand, allow their active entities (KSs) to react to different situations on the blackboard data structure at any time and simply prioritise their potential contributions based on the immediate search strategy and focus of attention employed.

Metaphorically, one can imagine a situation where in the one hand a group of experts are told what to do, where to do it and when to do it from a manager who *dictates* his or her own ideas in building a solution to a problem - with good to nearly incomplete knowledge of the abilities of the experts and when is the most opportune time to apply them to the problem. Conversely, a different manager with the same group of experts decides instead to *chair* the problem solving process - allowing experts to indicate their interest in contributing to the problem at any point but allowing only the most appropriate contribution at any given time to be posted. The experts who lost out on the chance to contribute in one given problem solving iteration can have the opportunity at a later time to apply their contribution should it still be valid.

These metaphorical situations highlight several benefits or disadvantages for the use of these different control philosophies.

Since our first manager, representing the hyper-heuristic level, is able to effectively direct the experts to solve a problem in this way - it is clear that the manager is either making use of a general strategy that he or she is applying or that the manager is maintaining a concise body knowledge about the abilities of each expert. [3] In the latter situation - the manager is maintaining concise knowledge, our manager with all of his/her expert knowledge is useful in only a handful of situations where the use of this particular set of experts is to be expected - if a new expert is added, the manager will need time to adapt to and learn this new source of knowledge. On the other hand, if the former is adopted - as is the case with most hyper-heuristics - the chosen strategy, being derived solely from the managers perspective, will likely not be able to exploit all the particular benefits of applying the experts to the problem at crucial points in the process - resulting in final solutions, which although may be considered as 'good' on average, could have been make better with the injection of more domain knowledge. There are certainly strengths and drawbacks from adopting any one position, but there may be benefits in combining the best from both worlds.

A possible starting point to this end may be found by considering our second metaphorical manager - representing the control methodology of the control shell in blackboard systems. By allowing the experts to guide the process of solving a given problem, each with more perspective on how and when it can be applied effectively, the manager can remain separated from the domain knowledge - as with our hyper-heuristic manager - but still have the benefits of increased domain perspective based on control data about what parts of the problem the experts are showing an interest in. In practice, this would mean that the heuristic KSs within a blackboard system would be largely in control of when they are applied and to what task or focus in the search they are to be employed e.g. exploration and exploitation. Although this type of separation between a hyper-heuristic layer and its lower-level heuristics is present in some hyper-heuristic systems, e.g., adaptive hyper-heuristics where heuristics are able to adjust their own internalised model of the search process, a blackboard systems knowledge sources, in addition to supporting this kind of adaption, do not require a pre-existing and detailed high-level control scheme e.g., *'apply heuristic x for 10 iterations, then, apply heuristic y for 20 iterations...etc.'*, but instead each knowledge source simply makes itself known to the control shell which then prioritises them based on what contributions appear to be relevant given the current search focus e.g., exploration vs. exploitation or a focus on one particular partial solution that appears interesting. Without a pre-existing high-level control scheme, except from a simple scheduling scheme[4], some generality in the control shell can be gained between application domains.

From a practical standpoint, separation between knowledge and control is often a desirable trait in any system - a trait both types of system share to varying degrees - not only in terms of reusability but also in reducing the complexity of the resulting system. However, like the blackboard control shell, having related items of knowledge encapsulated together and applied to the problem

[2]While this is true, it is also important in a blackboard system to ensure that all information placed on the blackboard can be mutually understood by all KS by enforcing a shared representation. This requirement does not necessarily have to limit the richness of domain knowledge - only the format in which it is posted to the blackboard

[3]It is arguable whether or not a human manager can manage all the relevant knowledge about the expertise of everyone else, although in this paper the latter is assumed.

[4]Which can actually be a simple as adopting a simple scheme like FIFO, shortest-job-first or oldest first

reactively in the presence of changes to the search trajectory relieves the burden on the developer of designing a robust search strategy that incorporates the abilities of each heuristic - likely with incomplete knowledge of how best to search the space of solutions. To clarify, heuristics in a hyper-heuristic system often encapsulate some domain knowledge about the problem being solved, such as what a solution to the problem would look like, whether there are constraints on solutions and some, like tabu search, have some awareness of the search trajectory, however, many other hyper-heuristics make some of this domain knowledge available in the hyper layer meaning that this layer cannot always be transferred from one problem to another. This is less of a problem for the blackboard control shell, that contains almost no domain knowledge - only knowledge of the representations to expect from knowledge sources as part of registering to contribute to a solution. It is worth noting however that it is not yet clear whether the use of one control scheme over the other is useful in terms of performance and quality of solutions - only that there are differences in control philosophy that can affect the manageability and reusability of a system - although, this can not be ruled out.

4.2.2 *Differences Between the Active Entities*

In a blackboard system, the KSs are considered as independent agents capable of reacting to changes in solution state in the blackboard data structure. They each encapsulate a condition-action pairing that allows a KS to (i) determine whether it is in a position to make a contribution to the ongoing solution and (ii) how changes should be made and to what objects. Conversely, heuristics within a hyper-heuristic system are not reactive to the state of the ongoing solution - this knowledge of when and where to act resides at the hyper-heuristic level - although heuristics of course know how changes should be made. Also mentioned in Sub-section 4.2.1, this main difference is centred around the level of control the active entities have between systems. The benefit of using this event-driven approach, at least within a blackboard system, is to reduce the need for or the complexity of positioning metrics used to insert and retrieve objects on the blackboard data structure - of which, depending on the application, may contain many hundreds of different objects spread across multiple blackboard levels.

Another difference related to the varying levels of control, is that since heuristics in a hyper-heuristic system do not require detailed control knowledge such as; event handling - in the event of solution state change - and knowledge about how to calculate its perceived contribution benefit to send to a control shell: heuristics tend to be far less complex than their KS counterparts that do have this knowledge implemented. Furthermore, a heuristic written for addressing one problem instance or instance type is often more reusable against other problems whereas KSs are generally more problem specific with dependencies ranging from solution representation to the specific implementation of the blackboard data structure. In a set of KSs many of these considerations will have to be addressed on an individual basis, where other specificities can be rendered mostly non-existent through the use of inheritance and abstraction during implementation.

5 CONCLUSIONS

We have discussed a couple of differences between the traditional view of hyper-heuristic approaches and that of blackboard systems. We have found that there are differences in the way both systems treat their control responsibilities, with emphasis on the level of domain knowledge used. As others share the view that increased domain-knowledge at the hyper-heuristic level will likely be of benefit, the use of a blackboard style of control merits further investigation. A difference in the behaviour, implementation and scale of the active entities between the two systems - as a result of these control differences - has been identified.

REFERENCES

[1] Grady Booch. 2006. *Object Oriented Analysis & Design with Application.* Pearson Education India.
[2] Edmund K Burke, Tim Curtois, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, and José Antonio Vázquez-Rodríguez. 2009. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland.* 790–797.
[3] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.
[4] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. 2010. A classification of hyper-heuristic approaches. In *Handbook of metaheuristics.* Springer, 449–468.
[5] D D Corkill. 1988. Design alternatives for parallel and distributed blackboard systems. *Proceedings of the AAAI-88 Workshop on Blackboard Systems* (1988), 99–136.
[6] Daniel D Corkill. 1991. Blackboard systems. *AI expert* 6, 9 (1991), 40–47.
[7] Peter Cowling, Graham Kendall, and Eric Soubeiga. 2000. A hyperheuristic approach to scheduling a sales summit. In *International Conference on the Practice and Theory of Automated Timetabling.* Springer, 176–190.
[8] Lee D Erman, Frederick Hayes-Roth, Victor R Lesser, and D Raj Reddy. 1980. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty. *ACM Computing Surveys (CSUR)* 12, 2 (1980), 213–253.
[9] Kevin Graham, Jerry Swan, and Simon Martin. 2015. The'Blackboard Pattern'for Metaheuristics. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation.* ACM, 1265–1267.
[10] Simon Martin, Djamila Ouelhadj, Pieter Smet, Greet Vanden Berghe, and Ender ÖZcan. 2013. Cooperative search for fair nurse rosters. *Expert Systems with Applications* 40, 16 (2013), 6674–6683.
[11] Jerry Swan, Patrick De Causmaecker, Simon Martin, and others. 2016. A re-characterization of hyper-heuristics. *Recent Developments of Metaheuristics* (2016).
[12] Jerry Swan, Patrick De Causmaecker, Simon Martin, and Ender Özcan. 2016 (to appear). A re-characterization of hyper-heuristics. In *Recent Developments of Metaheuristics*, F. Yalaoui L. Amodeo, E-G. Talbi (Ed.). Springer.
[13] Jerry Swan, John Woodward, Ender Özcan, Graham Kendall, and Edmund Burke. 2014. Searching the hyper-heuristic design space. *Cognitive Computation* 6, 1 (2014), 66–73.