A Review of Hyper-Heuristic Frameworks

Patricia Ryser-Welch [1] and **Julian F. Miller** [2]

Abstract. Hyper-heuristic frameworks have emerged out of the shadows of meta-heuristic techniques. In this very active field, new frameworks are developed all the time. Shared common features that help to classify them in different types of hyper-heuristic. Similarly to an iceberg, this large subfield of artificial intelligence hide a substantial amount of bio-inspired solvers and many research communities. In this paper, the tip of the iceberg is reviewed; recent hyper-heuristic frameworks are surveyed and the overall context of the field is presented. We believe its content complements recent reviews and offers another perspective of this important and developing field to the research community. Some hyper-heuristic frameworks tend to be largely constrained and prevent the state-of-the-art algorithms being obtained. We suggest in addition to relaxing constraints together with analysis of the evolved algorithms may lead to human-competitive results.

1 Introduction

In recent years, hyper-heuristic frameworks have emerged out of the shadows of meta-heuristic techniques. Those share common features that help to classify them in different types of hyper-heuristics. An analysis of shared common features allows them to be classified into different types of hyper-heuristics Similarly to an iceberg, this large subfield of artificial intelligence hides a substantial amount of bio-inspired solvers and many research communities.

Instead of exploring a search space of problem solutions, hyper-heuristics automatically produce an algorithm that solves a problem more efficiently. A global optima is not guaranteed to be found with heuristics, however it provides at least one solution whenever the algorithm stops. In the worst case, the algorithm iterates over a large number of candidates solutions before finding the best one. In the best case scenario, the best solution is found rapidly. The *"No Free lunch theorem"* (NFL) makes us aware that if a good performance is demonstrated by an algorithm on a certain class of problems it will have a trade-off; the algorithm performance will be degraded on others classes. Hyper-heuristics offers a general technique for optimising algorithms. Learning mechanisms can customize algorithms to the unique needs of a restricted class of problems; this should reliably find a more suitable solution faster for a well-defined problem class [46].

Our motivation is to review a variety of hyper-heuristic models and frameworks. identify their main purpose and the problems they have solved successfully. The next section compares two computing models of hyper-heuristics, before discussing the advantages and disadvantages of this search methodology. The following sections review algorithm-portfolio-based solvers, cross-domain hyper-heuristic and evolutionary frameworks. To conclude we discuss opportunities for further development and the wider applicability of such techniques. In this paper, only the tip of the iceberg can be reviewed; space restrictions prevent us to cover the entire field. We believe its content complements other reviews and offers another perspective to this important and rapidly developing research area.

[1] York University,England, email: patricia.ryser-welch@york.ac.uk
[2] York University,England, email: julian.miller@york.ac.uk

2 Hyper-heuristic models

2.1 Heuristic, metaheuristic and hyper-heuristic

Heuristic techniques are often referred to as "search algorithms". They solve problems by discovering a solution in the set of all possible solutions for a given problem, which is regarded as the ''search space". Non-deterministic search methods such as ''evolutionary algorithms", ''local search methods", ''Simulated annealing", and others search algorithms offer an alternative approach to exhaustive search to solve difficult computational problems in a reasonable amount of time. These methods guarantee finding a solution at any time, but it may not be optimum [11, 23, 34].

Within a metaheuristic context, the focus shifts from solving a specific problem to producing a heuristic that solves the problem. The purpose of such approaches is to find, generate, or select a method or algorithm to solve a problem; their search space is now the collection of all possible heuristics and the outcome can be formulae or algorithms together with a solution of the problem it solves. One example is Genetic Programming (GP) that solves *''automatically problems without the users to know or specify the the form or structure of the solution in advance. At the most abstract level GP is a systematic, domain-independent method for getting computers to solve problems automatically starting from a high-level statement of what needs to be done"* [32].

In the literature, the terms heuristic and metaheuristic are often used interchangeably and often effectively solve NP-hard problems. Nonetheless, these search strategies can be resource-intensive to implement and develop. Many of these algorithms rely on a stochastic population; runs can produce very different outcomes. Additionally, the algorithms and problem solutions can be unusual as well as challenging to understand; new domains of a heuristic search space may be explored that lead to problem solutions that cannot always be easily explained logically [34].

Hyper-heuristics aim to address some of these issues; this search methodology discovers some algorithms that are capable of solving a whole range of problems, with little or nor direct human control. It has been described as *''heuristics to choose heuristics"* [6] or *''hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational problems"* [5]. These techniques search the space of algorithms for any given problem in two ways. The first method assesses whether some combinations of pre-existing heuristics can improve the performance of the algorithm.

The second option generates some new heuristics with a metaheuristic search mechanism [34, 35].

2.2 A two-level model

A modular model separates the functionalities of a given problem from the functionalities of the algorithm optimization process. Described by Cowling, this easy-to-implement architecture has been widely adopted by the hyper-heuristic research community. In this paper, we refer to the top level as ''the Hyper level" and the lower level as "the Base level" (see fig 1) [6, 28, 34, 37, 41].

The Base level encapsulates a set of predefined heuristics for the given problem, a fitness evaluation function and a specific search space (see table 1). Its input parameters include a chosen heuristic with its the location memory and a chosen problem instance. Its only output is an performance evalution of the algorithm. [6, 41].

The Hyper level decides which Base-level heuristic to solve a chosen problem. This can be achieved with a learning mechanism that evaluates the quality of of the algorithm solutions, so that they can become general enough to solve unseen instances of a given problem. Grefenstette suggested that hyper-heuristics can be viewed as a form of reinforcement learning. Both methods employ online and offline learning. Online learning learns directly from its experience from its operational environment; self-modifying operators and autoconstructive evolution are examples of this type of hyper-heuristic (see sections 6.3 and 6.4). Offline learning gathers information in the form of programs from a set of training problems; automated design, meta-genetic programming, and full evolutionary algorithms are some examples [3, 14, 28].

The Hyper level encapsulates a workspace that acts as repository of its metaheuristic states and the states of the search and should offer more freedom to the learning mechanism. The only input is the performance indicator of the chosen heuristic and its output includes a chosen heuristic with its location memory and a chosen problem instance (see table 2 and fig. 1).

Quite naturally, the domain barrier interface between the Hyper and Base level. Once the Hyper level has selected randomly some heuristics, these are passed to the Base level. The lower level can then pass the performance indicators to the higher level.

Table 1: Encapsulation of the problem domain at the Base level as suggested by [41]

Variable	Description
S	Solution-state Space
$O = \{o_1, ..., o_n\}$	A set of predefined heuristics
e	Fitness function for instance of a problem.

2.3 The Algorithm Selection Problem

The Algorithm Selection Problem simply describes the iterative mechanism that is likely to take place during the learning process. *''For a particular problem instance* $p \epsilon P$ *with feature vector* $f(p) \epsilon F$, *find the selection mapping* $S(f(p))$ *into the algorithm space* A , *such the selected algorithm* $a \epsilon A$ *maximises the performance measure* $\| y \|$ *for* $y(a, p) \epsilon Y$" [38]. In fact, this architecture illustrates how the Problem space is embedded in the Feature space and subsquently affect the state the Algorithm space. The selection

Table 2: Operational environment of the *Hyper* level as suggested by [41]

Variable	Description
$H : [Q]xW$	The hyper-heuristic function
$Q : (i, j, k, e(s_k))$	metaheuristic states.
W	Repository for the states of the search and the states of the algorithm
(i, j, k)	A tuple made of variables i,j,k variables where - i represents the chosen heuristic - j the location memory of this heuristic - k the chosen problem instance.

Figure 1: The mathematical model suggested by [41] to represent a hyper-heuristic system.

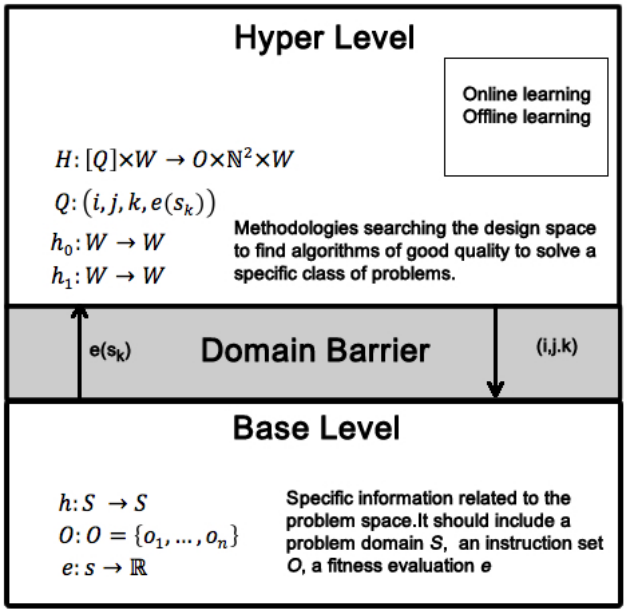
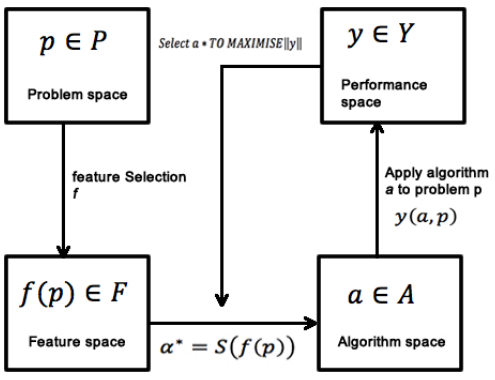


Figure 2: An illustration of the Algorithm Selection Problem, as originally proposed by [33]



process of a new heuristic can employ a variety of search methodologies, including online and offline learning. So we believe this general model cannot only model hyper-heuristic frameworks, but also meta-heuristics and others search methodologies that are outside the scope of this paper [7, 33, 38].

Table 3: Description of the main components of the Algorithm Selection Problem. These components are similar to the variables of the two-level model and simply describes all the elements of a heuristic.

Elements	Description
$p \epsilon P$	The problem search space, referred as Problem Space
$a \epsilon A$	The algorithm search space, called Algorithm Space
$y \epsilon Y$	The performance measures
$y(a, p) \epsilon Y$	The fitness evaluation of an algorithm for specific problem instance
$f(p) \epsilon F$	A set of existing heuristics for a given problem
$S(f(p)) \epsilon A$	Selection process of a new heuristic for a given problem

3 Discussion of hyper-heuristic methodologies

Using these four important distinctive components of section 2, the concept offers many advantages:

1. Hyper-heuristics should influence positively the selection of heuristics. The optimized heuristics for a given problem should compute high quality solutions. The learning phase should refine the algorithms, so that the algorithm solutions meet the needs of the training set and subsequently problems of a certain class can be solved more efficiently. Both models complement each other and comply with the ''No Free lunch" theorem. Their response mechanism should move towards optimum algorithm solutions in the workspace, as it guides the selection of heuristic. The Algorithm Selection Problem represents in a three-dimensional coordinate system the relationship between a problem instance, an algorithm solution and its performance. Comparatively, the two-level model offers a clear separation between the optimization of an algorithm and the optimization process of a specific problem. This provides a visualisation of the NFL [20, 34, 46].
2. The existence of the two models not only raises questions about the level of generality, but also introduces the concept of plug-and-play of heuristics. Both models at least separates the problem domain from the algorithm search space. Like Lego bricks the models offer elements a degree of freedom to be changed. With very little data being passed between each component, each element can be changed as long as they respect the interfaces in place. For example, the Hyper level search methodologies have no knowledge of the problem-domain concealed in the Base level. In turn, the Base level is not aware of the learning mechanism used to choose its heuristic, in the Hyper level. In comparison, every space of the Algorithm Selection Problem can also change each of its spaces, without affecting of the others [7, 28, 41].
3. Both models explore a greater design space. The stochastic process explores more candidate algorithms in the design space. We can imagine that hyper-heuristics can either produce algorithms that are close to the state-of-the-art methodologies or algorithms that have not yet been thought of by humans. They offer a viable and powerful tool that is able to respond to some performance indicators and probabilistically move the search forward to new areas in a reasonable amount of time. As suggested by [47], the development cost of writing heuristic could be potentially lowered. *"In addition Moore's law states that processor speed is increasing exponentially, while the cost of human labour increases in-line with inflation"* [4, 12].

Nonetheless the following issues needs to be considered too.

1. Experienced-based methodologies provide algorithms that may not be guaranteed to be optimal. These algorithms may vary after each run and be challenging to understand intuitively. The chosen heuristic can produce solutions of a lower quality than expected. It may also not be trusted by its users; the algorithm search may have generated an unknown order of instructions. The chosen problem area must then be able to cope with the speculative and randomness of hyper-heuristics. It could be disastrous if the maximum strain of a steel cable is solved with an algorithm of poor quality. Lives could be lost, if the cable is used inappropriately, with a lift with a load that is too heavy [12, 34, 35].
2. The simplicity and modularity of the two models offers the opportunity to represent simple or very complex hyper-heuristics. This varying complexity can be implemented in either one element, several elements or all of them. Adding too much technical knowledge and the programmers' expertise can result in reducing the reusability and the applicability of a framework. These systems require a lot of effort to understand them. Additionally, the embedded conceptual elements in the application programming interface could become challenging to use again; some logic may not be suitable in another context. In others areas of evolutionary computations (EC), researchers have shown that EC can produce designs that surpass the state-of-the-art. Overly complex frameworks may prevent this creative feature occuring [24, 43].
3. Similarly to the full evolution of an evolutionary algorithm, the training phase could be quite power-hungry with a long training time. Although the performance of computers is improving all the time, this important factor cannot be ignored. The search in the algorithm space could be affected; the domain knowledge may be gained with fewer generations than expected and affect the quality of the learning. Also the produced algorithm may find good quality solutions, but their execution time and number of generations may be too large. To overcome this issue, some hyper-heuristics extend the fitness measure at the Hyper level by including higher level variables such as the execution time [8, 28].

4 Algorithm-Portfolio-based frameworks

Hyper-heuristic frameworks known as Algorithm-Portfolio-based frameworks aim at predicting the running time of algorithms, so that the time they take to solve a problem can be reduced. This idea identified that consideration of the running time of algorithms had been neglected from the Algorithm Selection Problem. The "Algorithm Portfolio" technique provides a means to overcome this problem. This method applies the Algorithm Selection Problem to construct models of algorithms runtimes using statistical regression. Then for a given instance of a problem, each algorithms time is predicted and the fastest predicted algorithm is used to solved the problem until the allotated time is used up or a suitable solution is found [25].

4.1 The SATzilla framework

SATzilla applies the Algorithm portfolio to Boolean satisfiability (SAT) problems; SAT solvers have been built for nearly a decade.

An offline learning process develops first a portfolio of algorithms, before applying each of them against an instance of the problem to select the fastest algorithm and predicts its runtime. More recently, new benchmarks instances and a variety of new base solvers were added to the framework [29, 49–51].

4.2 The Snappy framework

The "Simple Neighborhood-based Algorithm Portfolio in PYthon" (Snappy) is a more recent framework. Although this framework also adopts the Algorithm portfolio, its aim is to provide a tool that can improve its own performances through online learning. Instead of using the traditional offline training step, a neighbourhood search predicts the performance of the algorithms. Snappy outperformed state-of-the-art benchmarks problems previously solved by SATzilla [36].

5 Cross-domain hyper-heuristic frameworks

In this section we review some cross-domain frameworks that have been recently mentioned in the literature. All these frameworks are implemented with Java, to provide a library that helps the programmers to write hyper-heuristic algorithms more easily in the Hyper level. All these frameworks offer a range of tools abstracted from iterated local search methodologies, that can be used to quickly create some hyper-heuristics.

5.1 Hyflex and parHyFlex

The motivation of Hyflex was inspired by the two-level hyper-heuristic model (see figure 1). *''The emphasis of our HyFlex framework lies in providing the algorithm components that are problem specific, thus liberating the algorithm designers needing to know the problem's domain's specific details"* [2]. An interface between the Hyper and the Base level is provided, with the main purpose of comparing a variety of hyper-heuristics. In fact, the algorithm designers can only devise new Hyper level algorithms; the Base level contains a library of well-known combinatorial problem domains with their benchmarks. In this context, the low-level heuristic supplies a set of operators that either apply small or large changes in the problem solutions. These perturbations should expand the search to a larger neighborhood and then guarantees better solutions are found [1, 2].

The flexibility offered by object oriented programming gives a simple and convenient method to easily create some hyper-heuristics. The framework structure hides strictly within the domain barrier the problem domain, in order to implement a domain-independent form of hyper-heuristic. *"Using the framework, one can implement a hyper-heuristic without any knowledge about the algorithm running on parallel systems"* [44]. The "Problem-domain, Hyper-heuristic and Heuristic type" classes decompose the system in explicit templates; a diagram can be found in [1] and [30]. New hyper-heuristics are then derived from those components and only the code that specifically differs from the original problem domains or hyper-heuristics is then written. For example, [45] developed a specific subclass of the Problem Domain for the vehicle routing problem and from the Hyper-heuristic another three subclasses that implement three different adaptive iterated local search. This new class encoded a representation of this NP-hard problem, an evaluation function with some benchmark problems and the current state-of-the-art operations. On the other hand, [27] used Hyflex to implement a more complex Hyper level. The research used again the problem domain library with an Adaptive Dynamic Heuristic Set strategy enhanced with a learning automaton.

This strict use of templates could limit the ability of Hyflex of solving large real-world problems; such problem-domain preferably require less domain information [35]. Also the algorithm designers are required to structure their code with the explicit definitions of the three components. Finally, the framework seems to only support local search meta-heuristic in the Hyper level, making it very challenging to use Genetic Programming.

5.2 Hyperion

Hyperion applies a general reusable hyper-heuristic solution, to offer the tools to rapidly create a prototype. Its main aim helps identifying the components that contribute to an algorithm's good performance. A transition function uses the problem domain variables (see table 1) to transform a problem solution into another one; $Transition : S \rightarrow S$. In this case the transition has been defined as Eqn. 1. These transitions result from a variety of search methodologies that are built in a library. Hyperion also provides the four learning mechanisms described by [31]; the most complex framework recursively aggregates the hyper-heuristic to implement a hierarchy of hyper-heuristics.

$$\begin{aligned} Transition \quad = \quad & \{(from, fromValue, Operator, to, toValue)\} \\ & from \in S, \\ & fromValue \in R, \\ & operator \in O, \\ & to \in S, \\ & toValue \in R \end{aligned} \tag{1}$$

Experiments using the Boolean satisfiability compared the performance of several neighbourhood techniques [42].

5.3 hMod

Inspired by the previous frameworks, hMod abstracts all the elements of flow charts in a new object-oriented architecture. This model encodes the core of the Hyper level in several modules, referred as algorithm containers. hMod directs the programmer to define the Hyper level heuristic using two separate XML files; one for the heuristic selection process and another one for the acceptance move. These XML files are then read and interpreted with the code [43].

1. Each flowchart has a start and an end. An initial step is encapsulated in an "algorithm" class and the "flow control" in a "step" class. This variable points to the next operation, except for the last operation, which points to nothing.
2. A generic processing step holds a set of instructions that describe a specific behaviour.
3. The "decision" is treated as special step with two flow controls; one if the condition is met and another one if the condition is not met. The decision is useful with iterations and conditional execution.
4. "Input/output" has its own set of data classes with the traditional get and set methods.

At the time of writing, this new framework was only at the proposal stage. No result of its performance was available to allow comment.

6 Evolutionary hyper-heuristics

This second branch of hyper-heuristics optimises algorithms with Genetic Programming (GP) at the *hyper* level. Unlike the previous frameworks, the top level remains mostly unchanged, and most of the effort is required to encode the problem domain at the *Base* level.

6.1 Automated design and meta-genetic programming

These methods specialise in the automatic design of components of Genetic Algorithms. Similarly to meta-genetic programming (see [13]), automated design considers components of an evolutionary algorithm with the purpose of improving their performance without losing excessively the generality of this well-known algorithm. Both techniques can be applied to the hyper-heuristic models aforementioned. At the Base level the performance of the operators is assessed by executing several times a Genetic Algorithm with the newly generated operators. While, meta-genetic programming encodes their algorithms in a tree, automated design uses register machines at the Hyper level. Only automated design strongly aims at producing selection or mutation operators with improved performance on a class of problems. The domain knowledge is acquired with a set of instances chosen from a given problem class, during a training phase. The operators' performance is assessed during the validation phase and the real-world phase can apply fully-developed algorithms on problems of the same given class.

Automated design has successfully improved the performance of selection operators for a GA that solved problem classes of the one-max problem. It was also used to explore mutation operators for a GA that solves mathematical functions with arbitrary chosen bit-strings. In these experiments, the parameters of each problem were defined by a Gaussian distribution. Finally, automated design discovered and generated new statistical distribution for the mutation operators [21, 47, 48].

6.2 Full evolution of evolutionary algorithms

Evolving evolutionary algorithms (EEA) fully adapts an evolutionary algorithm to the given needs of a problem; it is a very specialised hyper-heuristic method. It lets an EA discover the rules and knowledge, so that it can find the best EA to optimise the solutions of a problem. Several subfields of Genetic Programming have achieved this purpose with some success; Linear Genetic Programming (LGP), Multiple Expression Programming (MEP) and Grammatical Evolution (GE) have produced unknown evolutionary algorithms [9, 10, 22, 26].

To the best of our knowledge, only a few researchers have yet focused on optimising the sequence of an EA to the specific need of a problem. Two approaches demonstrated the feasibility automatically evolving evolutionary algorithms. The first method used linear genetic programming and multi-expression genetic programming, to optimise the EA solving unimodal mathematical functions. An evolutionary algorithm manipulates a sequence of genetic and reproductive operators with their parameters. A solution consists of a new evolutionary algorithm that is capable of outperforming GA when solving a specific class of unimodal test functions. The result was an initial time-consuming and power-hungry learning process, such systems seems to be better suited to test hypothesis rather than actively solving a problem [9, 10].

A Grammatical Evolutionary framework also evolved EAs, to instead solve the Royal Road problems. Genetic operators are performed on binary string that encodes the EEAs. A mapping process, inspired by the protein synthesis, then transform this simple code into an EEA. The binary string are transcribed into an integer string, which is in turn derived into a tree structure using a set of predefined grammatical rules. Then the EEA is executed to solve classes of Royal Road Problems. Results demonstrates GE can optimise EAs, despite the grammatical rules imposed during evolution seeming to hinder the production of innovative EAs [26].

6.3 Self-modifying operators

Instead of repetitively using GP to select or generate the lower heuristic, before assessing its performance, self-modifying operators integrate the variation operators within the algorithm itself. These added online-learning features in the Hyper level empowered with the capability of adapting operators at the higher-level operate at the same time it is optimising low-level heuristics.

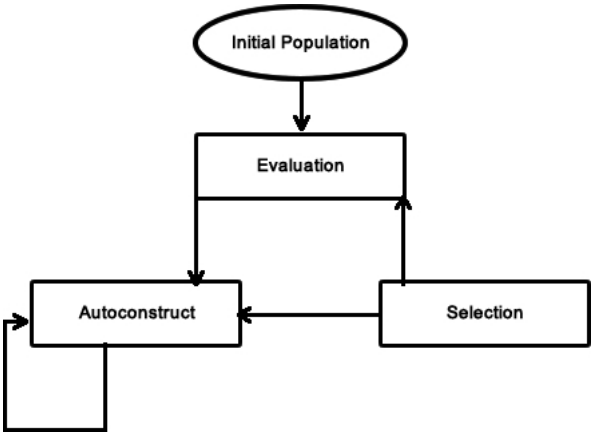
Self-modification Cartesian Genetic Programming (SMCGP) encodes in a graph the chosen low-level heuristic, alongside self-modifying operators. During the evaluation phase, a copy of the CGP program is made. This graph is then executed, and if they are any alterations to be made, then they modify the new graph. The encoded graph is then executed the same manner as a CGP program. SMCGP has been used to solve a number or problems. It has found general solutions to arbitrary parity problems, binary addition and Fibonacci series. It has discovered algorithms for computing e and pi to arbitrary position [15–17].

6.4 Autoconstructive evolution

Another approach lets the Hyper level programs be subjected to changes in their structure, at the same time they are responsible themselves produce their own children. *"In fact, autoconstructive evolution is a hyper-heuristic in two ways: reproductive mechanisms are evolved which are then used to vary problem solutions, and reproductive mechanisms vary the reproductive mechanisms."* [19] (see figure 3). Push is a well-known autoconstructive evolution system defined by [39]. A detailed tutorial demonstrates clearly how stacks differentiate in term of data type and store the values of variables. [40] A stack can either represent a simple data type like a boolean, or a more complex data structure like locations within a tree. The latter becomes very useful to encode instructions that adapt the tree structure of a tree dynamically. This type of hyper-heuristic method was successfully applied to solve the order and majority problems, with the following evolutionary elements:

1. An autoconstructive mutation operator was defined as $f' = f(g)$ and an auto constructive crossover operator had the mathematical function $f' = f(f, g)$
2. Each problem domain has their own fitness function defined separately.
3. The Push interpreter maintains a parent and a children population. A tournament selects the parent from the parent population before the autoconstructive operators are applied. The children population only accepts candidate solutions that meet the size requirements, have less errors than their parents and finally are different from their parents.
4. After the Push interpreter is initialised the two selected parent programs f and g are copied in the children population. Each of the

Figure 3: The Autoconstructive evolution as suggested by [18]. The hyper-heuristic gives the program the capability to evolve itself.



autoconstructive genetic operators are executed and solve either the order problem or the majority problem. The program becomes a candidate child program and the acceptance criteria is applied.

The entanglement of the low-level heuristic and its problem domain within the higher-heuristic program can lead to the problem solutions and its algorithm being inseparable. By treating the produced algorithm as a variation operator, the program may be used independently to solve problems belonging to the same class [18, 19, 39].

7 Discussion and conclusion

We have discussed types of heuristics, and compared and discussed the differences between the main hyper-heuristic models and frameworks. The hyper-heuristic research community is very active and have produced impressive results. However, they are still many questions that remain to be considered to advance the field further.

Although hyper-heuristic frameworks are well developed, there has been little discussion of the generated algorithms themselves. Can something be learned from study and analysis of generated algorithms that perform well? It seems that early discussions of these aspects in meta-heuristics have not been continued. The chosen encoding scheme is not expressive enough to allow easily the analysis of computational patterns of operations against their performance.

The operations that have been used to generate new candidate algorithms have been dominated by local search operations. Such operations would be very unlikely to create algorithms which carry out a form of cross-over. In addition, the form that generated algorithm can take appears to be very limited. For instance, any forms of GP cannot be currently produced by these cross-domain hyper-heuristics approaches. Too much human domain knowledge from software engineering has been included in the structure of the frameworks. This seems to prevent the framework from being adapted easily from their current educational and prototyping focus.

Algorithm design for solving NP-hard problems is an area of intense research. Many sophisticated state-of-the-art algorithms exist, yet currently it is unknown whether hyper-heuristic frameworks are capable of expressing such algorithms. To the best of our knowledge, structures of algorithms are seldom analysed or compared with the state-of-the-art algorithms. For instance, in the field of logic synthesis and minimisation there exist number of very effective algorithms. Can such algorithms be improved on using hyper-heuristic methods?

So far, hyper-heuristic frameworks have been restricted to a constrained set of possible algorithms. In general, it may not have enough expressiveness to represent a greater variety of algorithms, closer to the state-of-the-art or even programming languages.

The range of computer languages used to encode cross-domain hyper-heuristic frameworks has been quite limited and needs to take advantage of a greater ranges of programming platforms.

REFERENCES

[1] Edmund Burke, Tim Curtois, Matthew Hyde, Gabriela Ochoa, and Jose A Vazquez-Rodriguez, 'Hyflex: A benchmark framework for cross-domain heuristic search', *arXiv preprint arXiv:1107.5462*, (2011).

[2] Edmund K Burke, Tim Curtois, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, and JA Vazquez-Rodriguez, 'Hyflex: A flexible framework for the design and analysis of hyper-heuristics', in *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland*, pp. 790–797, (2009).

[3] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward, 'A classification of hyper-heuristic approaches', in *Handbook of Metaheuristics*, 449–468, Springer, (2010).

[4] Edmund K Burke, Matthew R Hyde, Graham Kendall, and John Woodward, 'Automating the packing heuristic design process with genetic programming', *Evolutionary computation*, **20**(1), 63–89, (2012).

[5] Konstantin Chakhlevitch and Peter Cowling, 'Hyperheuristics: recent developments', in *Adaptive and multilevel metaheuristics*, 3–29, Springer, (2008).

[6] Peter Cowling, Graham Kendall, and Eric Soubeiga, 'A hyperheuristic approach to scheduling a sales summit', in *Practice and Theory of Automated Timetabling III*, 176–190, Springer, (2001).

[7] Laura Cruz-Reyes, Claudia Gómez-Santillán, Joaquín Pérez-Ortega, Vanesa Landero, Marcela Quiroz, and Alberto Ochoa, 'Algorithm selection: From meta-learning to hyper-heuristics', *Intelligent Systems*, (2012).

[8] Laura Dioşan and Mihai Oltean, 'Evolutionary design of evolutionary algorithms', *Genetic Programming and Evolvable Machines*, **10**(3), 263–306, (2009).

[9] Laura Diosan and Mihai Oltean, 'Evolutionary design of evolutionary algorithms', *Genetic Programming and Evolvable Machines*, **10**(3), (2009).

[10] Laura Silvia Diosan and Mihai Oltean, 'Evolving evolutionary algorithms using evolutionary algorithms', in *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, GECCO'07, pp. 2442–2449. ACM, (2007).

[11] Stephane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, and Vincent Padois, 'Evolutionary robotics: Exploring new horizons', in *New Horizons in Evolutionary Robotics*, 3–25, Springer, (2011).

[12] Stéphane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, and Vincent Padois, 'Evolutionary robotics: Exploring new horizons', in *New Horizons in Evolutionary Robotics*, 3–25, Springer, (2011).

[13] Bruce Edmonds, 'Meta-genetic programming: Co-evolving the operators of variation', *Elektrik*, **9**(1), 13–30, (2001).

[14] John J Grefenstette, David E Moriarty, and Alan C Schultz, 'Evolutionary algorithms for reinforcement learning', *arXiv preprint arXiv:1106.0221*, (2011).

[15] Simon Harding, Julian F Miller, and Wolfgang Banzhaf, 'Smcgp2: self modifying cartesian genetic programming in two dimensions.', in *GECCO*, pp. 1491–1498, (2011).

[16] Simon Harding, Julian Francis Miller, and Wolfgang Banzhaf, 'Self modifying cartesian genetic programming: Parity', in *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pp. 285–292. IEEE, (2009).

[17] Simon L Harding, Julian F Miller, and Wolfgang Banzhaf, 'Self-modifying cartesian genetic programming', in *Cartesian Genetic Programming*, 101–124, Springer, (2011).

[18] K Harrington, E Tosch, L Spector, and J Pollack, 'Compositional autoconstructive dynamics', in *Proc. of the 8th Intl. Conf. on Complex Systems*, (2011).

[19] Kyle I. Harrington, Lee Spector, Jordan B. Pollack, and Una-May O'Reilly, 'Autoconstructive evolution for structural problems', in *Proceedings of the fourteenth international conference on Genetic and*

evolutionary computation conference companion, GECCO Companion '12, pp. 75–82. ACM, (2012).

[20] Yu-Chi Ho and David L Pepyne, 'Simple explanation of the no-free-lunch theorem and its implications', *Journal of Optimization Theory and Applications*, **115**(3), 549–570, (2002).

[21] Libin Hong, John Woodward, Jingpeng Li, and Ender Özcan, 'Automated design of probability distributions as mutation operators for evolutionary programming using genetic programming', in *Genetic Programming*, 85–96, Springer, (2013).

[22] W. Kantschik, P. Dittrich, M. Brameier, and W. Banzhaf, 'Empirical analysis of different levels of meta-evolution', in *Proceedings of the Congress on Evolutionary Computation, (CEC 99)*, volume 3, pp. 2086–2093, (1999).

[23] Natallia Kokash, 'An introduction to heuristic algorithms', *Department of Informatics and Telecommunications*, (2005).

[24] John R Koza, 'Human-competitive results produced by genetic programming', *Genetic Programming and Evolvable Machines*, **11**(3-4), 251–284, (2010).

[25] Kevin Leyton-Brown, Eugene Nudelman, Galen Andrew, Jim McFadden, and Yoav Shoham, 'A portfolio approach to algorithm selection', in *IJCAI*, volume 1543, p. 2003, (2003).

[26] Nuno Lourenço, Francisco Pereira, and Ernesto Costa, 'Evolving evolutionary algorithms', in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion, pp. 51–58. ACM, (2012).

[27] Mustafa Mısır, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe, 'A new hyper-heuristic as a general problem solver: an implementation in hyflex', *Journal of Scheduling*, 1–21, (2013).

[28] Mustafa Misir, Katja Verbeeck, Patrick De Causmaecker, and Greet Vanden Berghe, 'A new hyper-heuristic implementation in hyflex: a study on generality', in *Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory & Application*, pp. 374–393, (2011).

[29] Eugene Nudelman, Kevin Leyton-Brown, Alex Devkar, Yoav Shoham, and Holger Hoos, 'Satzilla: An algorithm portfolio for sat', *Solver description, SAT competition*, **2004**, (2004).

[30] Gabriela Ochoa, Matthew Hyde, Tim Curtois, Jose A Vazquez-Rodriguez, James Walker, Michel Gendreau, Graham Kendall, Barry McCollum, Andrew J Parkes, Sanja Petrovic, et al., 'Hyflex: a benchmark framework for cross-domain heuristic search', in *Evolutionary Computation in Combinatorial Optimization*, 136–147, Springer, (2012).

[31] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz, 'A comprehensive analysis of hyper-heuristics', *Intelligent Data Analysis*, **12**(1), 3–23, (2008).

[32] Riccardo Poli, W William B Langdon, Nicholas F McPhee, and John R Koza, *A field guide to genetic programming*, Lulu. com, 2008.

[33] John R Rice, 'The algorithm selection problem', *Advances in Computers*, **15**, 65–118, (1976).

[34] Peter Ross, 'Hyper-heuristics', in *Search methodologies*, 529–556, Springer, (2005).

[35] Peter Ross, 'Hyper-heuristics', *Search Methodologies*, 611–638, (2014).

[36] Horst Samulowitz, Chandra Reddy, Ashish Sabharwal, and Meinolf Sellmann, 'Snappy: A simple algorithm portfolio', in *Theory and Applications of Satisfiability Testing–SAT 2013*, 422–428, Springer, (2013).

[37] K Smith-Miles, R James, J Giffin, and Yiqing Tu, 'Understanding the relationship between scheduling problem structure and heuristic performance using knowledge discovery', *Learning and Intelligent Optimization, LION*, **3**, (2009).

[38] Kate Smith-Miles and Leo Lopes, 'Measuring instance difficulty for combinatorial optimization problems', *Computers & Operations Research*, **39**(5), 875–889, (2012).

[39] Lee Spector, 'Towards practical autoconstructive evolution: Self-evolution of problem-solving genetic programming systems', in *Genetic Programming Theory and Practice VIII*, 17–33, Springer, (2011).

[40] Lee Spector, 'Expressive genetic programming: tutorial: 2012 genetic and evolutionary computation conference (gecco-2012)', in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pp. 983–1012. ACM, (2012).

[41] Jerry Swan, John Drake, Ender Özcan, James Goulding, and John Woodward, 'A comparison of acceptance criteria for the daily car-pooling problem', in *Computer and Information Sciences III*, 477–483, Springer, (2013).

[42] Jerry Swan, Ender Özcan, and Graham Kendall, 'Hyperion–a recursive hyper-heuristic framework', in *Learning and intelligent optimization*, 616–630, Springer, (2011).

[43] Enrique Urra, Daniel Cabrera-Paniagua, and Claudio Cubillos, 'Towards an object-oriented pattern proposal for heuristic structures of diverse abstraction levels', in *Jornadas Chilenas de Computacin 2013, Temuco, Chile*, (2013).

[44] Willem Van Onsem, Bart Demoen, and KU Leuven, 'Parhyflex: A framework for parallel hyper-heuristics', *status: accepted*, (2013).

[45] James D Walker, Gabriela Ochoa, Michel Gendreau, and Edmund K Burke, 'Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework', in *Learning and Intelligent Optimization*, 265–276, Springer, (2012).

[46] David H Wolpert and William G Macready, 'No free lunch theorems for optimization', *Evolutionary Computation, IEEE Transactions on*, **1**(1), 67–82, (1997).

[47] John R Woodward and Jerry Swan, 'The automatic generation of mutation operators for genetic algorithms', in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pp. 67–74. ACM, (2012).

[48] John Robert Woodward and Jerry Swan, 'Automatically designing selection heuristics', in *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pp. 583–590. ACM, (2011).

[49] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown, 'Satzilla-07: The design and analysis of an algorithm portfolio for sat', in *Principles and Practice of Constraint Programming–CP 2007*, 712–727, Springer, (2007).

[50] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown, 'Satzilla2009: an automatic algorithm portfolio for sat', *SAT*, **4**, 53–55, (2009).

[51] Lin Xu, Frank Hutter, Jonathan Shen, Holger H Hoos, and Kevin Leyton-Brown, 'Satzilla2012: Improved algorithm selection based on cost-sensitive classification models', *Proc. of SAT Challenge*, 57, (2012).