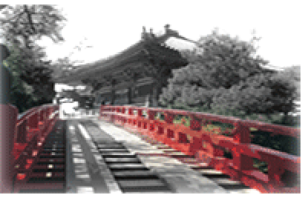
2015 IEEE Congress on Evolutionary Computation (CEC2015)
Sendai International Center, Sendai, Japan 25th-28th May 2015



Tutorial: An Overview of Evolutionary Algorithms and Hyper-Heuristics

Nelishia Pillay

Table of Contents

Introduction

Hyper-heuristics is a rapidly developing domain which has proven to be effective at providing generalized solutions to problems and across problem domains. Evolutionary algorithms have played a pivotal role in the advancement of hyper-heuristics. Evolutionary algorithm hyper-heuristics have been successful applied to solving problems in various domains including packing problems, educational timetabling, vehicle routing, permutation flowshop and financial forecasting amongst others. The aim of the tutorial is to provide an introduction to evolutionary algorithm hyper-heuristics for researchers interested in working in this domain. Researchers will be provided with a foundation to start their own research in this area. The tutorial will firstly provide an overview of hyper-heuristics and evolutionary algorithm hyper-heuristics. The tutorial will examine the different categories of hyper-heuristics, showing how evolutionary algorithms can be used for each type of hyper-heuristic. Challenges in the implementation of evolutionary algorithm hyper-heuristics will be highlighted. The tutorial will also look at recent and emerging research directions in evolutionary algorithm hyper-heuristics. Two emerging directions are hyper-heuristics for the design of evolutionary algorithms and evolutionary algorithm hyper-heuristics for algorithm design. The tutorial will end with a discussion session on future research directions in evolutionary algorithm hyper-heuristics.

1. An Overview of Hyper-Heuristics

This section provides an overview of hyper-heuristics in terms of low-level heuristics, classification of hyper-heuristics and hyper-heuristic frameworks.

1.1 Low-level Heuristics

Hyper-heuristics explore a heuristic space of low-level heuristics. These heuristics are divided into two categories, namely, constructive and perturbative. Constructive heuristics are used to create a solution to a problem while perturbative heuristics are used to improve an initial solution. The initial solution is created randomly or by using a low-level construction heuristic.

Case Study: Low-Level Heuristics

<u>The Examination Timetabling Problem</u>

Description: This problem involves the allocation of examinations to a set of specified timetable periods.

Constraints: The problem includes hard and soft constraints. Hard constraints must be met for the timetable to be feasible. The soft constraint cost is a measure of the quality of the timetable. Not all soft constraints can be satisfied as these are sometimes contradictory.

Objective function: Is a measure of the hard and/or soft constraints violated. A hard constraint cost of zero must be obtained and the soft constraint cost is minimized.

Constructive low-level heuristics: These heuristics are used to estimate the difficulty of scheduling an examination. The examinations are sorted in order of difficulty and scheduled accordingly. Heuristics commonly used for this purpose include:

- Largest degree (l) - The examination with the most clashes is scheduled first.
- Largest enrollment (e) - The examination with the largest number of students is scheduled first.
- Largest weighted degree (w) - The examination with the largest number of students involved in clashes is scheduled first.
- Saturation degree (s) - The examination with the least number of feasible periods on the timetable is scheduled first.

Exercises: Low-Level Construction Heuristics

Consider a simplified version of the examination timetabling problem that requires scheduling of four examinations. The hard constraint for the problem is there must be no clashes, i.e. a student must not be scheduled to sit two examinations simultaneously. The soft constraint for the problem is that the examinations must be well spread. The clash matrix for the examinations to be scheduled is listed below:

	E1	**E2**	**E3**	**E4**
E1	0	20	10	50
E2	20	0	0	0
E3	10	0	0	30
E4	50	0	30	0

Each cell indicates the number of students in common for the two exams. For example, 20 students are registered for both E1 and E2.

1. Complete the table below by calculating the heuristics:

Examination	**Largest Degree**	**Largest Weighted Degree**
E1		
E2		
E3		
E4		

2. Construct a timetable for the four examinations using the saturation degree heuristic.

Perturbative low-level heuristics: Perturbative or move low-level heuristics are applied to an examination timetable created by randomly scheduling exams or using one of the constructive heuristics. Perturbative heuristics for examination timetabling include:

1. Swapping two randomly selected exams
2. Swapping subsets of exams
3. Deallocating exams
4. Rescheduling exams
5. Swapping timeslots of two randomly selected examinations

In some instances these low-level heuristics are applied with hill-climbing, i.e. only if a move improves the feasibility and/or equality of the timetable will its effect be retained.

Exercises: Low-Level Perturbative Heuristics

Consider the one-dimensional bin-packing problem:

A number of items of different sizes have to be placed in bins. Each bin has the same capacity. Solving the problem begins with one bin, once this bin is full a second bin is opened and so on.

All the items must be stored using the minimum number of bins.

1. Define low-level construction heuristics for this problem.
2. Define low-level perturbative heuristics for this problem.
3. Can any of the heuristics used for the examination timetabling problem above be used for this problem?

1.2 Classification of Hyper-Heuristics

Hyper-heuristics aim to provide a more generalized solution to a particular problem instead of producing the best results for some of data sets. This is achieved by exploring a heuristic space rather than a solution space as in the case of metaheuristics. The heuristic space is comprised of low-level constructive or perturbative heuristics. Hyper-heuristics select the low-level heuristic to construct or improve a solution at each stage of the construction or improvement process. Alternatively, hyper-heuristics are used to generate new low-level constructive or perturbative heuristics. Thus hyper-heuristics can be classified into four main categories:

- Selection constructive
- Selection perturbative
- Generation constructive
- Generation perturbative

If a hyper-heuristic uses some feedback obtained during the search process, it is considered to be a learning algorithm. This learning process can be online or offline. In online learning the hyper-heuristic learns while solving the problem. In offline learning the hyper-heuristic learns on training instances and is applied to unseen instances. The following sections examine each category of hyper-heuristic.

1.2.1 Selection Constructive Hyper-Heuristics

Selection constructive hyper-heuristics select the constructive heuristic to use at each stage in constructing a solution to a particular problem. A problem specific objective function and low-level construction heuristics provide input to the hyper-heuristic. Examples of application domains which evolutionary algorithm hyper-heuristics have been successfully applied to include educational timetabling, production scheduling, bin packing and cutting stock problems. Selection hyper-heuristic have generally used cased-based reasoning or metaheuristics to choose a low-level heuristic at each point of solution construction. In the case of the latter the metaheuristics explore the heuristic space instead of a solution space. Metaheuristics used for this purpose include tabu search, variable neighbourhood search, genetic algorithms and simulated annealing.

1.2.2 Selection Perturbative Hyper-Heuristics

Selection perturbative hyper-heuristics choose a low-level perturbative heuristic at each stage in the improvement of an initially created solution. Selection perturbative hyper-heuristics can perform a multi-point or single-point search. Single-point selection perturbative hyper-heuristics are comprised of two components, namely, a heuristic selection method and a move acceptance method. Heuristic selection methods include random selection, max selection which chooses the heuristic with the highest score, roulette wheel selection and reinforcement learning. Move acceptance can be deterministic or non-deterministic. Deterministic methods are not stochastic in nature and will always produce the same decision for the same input, e.g. accept all moves or accept improvements only. Non-deterministic acceptance methods are generally stochastic in nature such as great deluge and simulated annealing. Multi-point selection perturbative hyper-heuristics are population-based and do not have distinct heuristic selection and move acceptance components. The hyper-heuristic employs a metaheuristic to explore the space of low-level perturbation heuristics. In the context of genetic algorithms these have also been referred to as GAs using an indirect representation. Other metaheuristics employed for this purpose include ant colonization and particle swarm optimization.

Applications domains that evolutionary algorithm selection perturbative hyper-heuristics have been successfully applied to include educational timetabling, sports scheduling, personal scheduling and vehicle routing.

Exercises: Selection Hyper-Heuristics

Consider the examination timetabling problem in the exercises in part 1.

1. Design a selection construction hyper-heuristic to solve the problem.
2. Design a single-point selection perturbative hyper-heuristic for solving this problem using simple random for heuristic selection and accept all moves for move acceptance.
3. Apply the hyper-heuristic in 1. to solve the problem. What is the effect of changing the method of move acceptance to accept improving moves only?
4. Design a multi-point genetic algorithm selection perturbative hyper-heuristic for solving this problem.

1.2.3 Generation Hyper-Heuristics

Generation hyper-heuristics create new low-level heuristics. Genetic programming has chiefly been employed for this process. Each program combines existing low-level heuristics or problem domain characteristics to create new low-level heuristics. Variations of genetic programming, namely, grammar-based genetic programming and grammatical evolution have also been used to generate new low-level heuristics. More recently gene expression programming has been used for this purpose. Generation hyper-heuristics have produced good results for the domains of educational timetabling and packing problems.

The heuristics evolved by the hyper-heuristic can be disposable or reusable. Disposable heuristics are evolved for the particular problem instance. A resuable heuristic is evolved using one or more problem instances and applied to unseen data.

1.3. Hyper-Heuristics Frameworks

Hyper-heuristic frameworks have been introduced to reduce the workload involved in the development of hyper-heuristics. In [11] a review of the available hyper-heuristic frameworks is provided. Hyflex [8] was developed for the cross-domain hyper-heuristic challenge(CheSC 2011). This framework provides the objective function, low-level perturbative heuristics and a method for creating an initial solution for various combinatorial optimization problems. The researcher focuses on creating a selection perturbative hyper-heuristic for solving the problem. HYPERON^2 is a Java framework for tracing and analyzing the trace performance of hyper-heuristics [1].

References

1. Brownlee, A.E.I., Swan, J. (2014) HYPERION 2A ToolKit for {meta-, hyper-} heuristic research. In proceedings of GECCO '14, 12-16 July 2014, Vancouver, British Columbia, pp. 1133-1139, ACM.
2. Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R. (2013) Hyper-Heuristics: A Survey of the State of Art. Journal of the Operational Research Society, 64, 1695–1724.
3. Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S. (2003) Hyper-Heuristics: An Emerging Direction in Modern Research. In the Handbook of Metaheuristics, Chapter 16, 457– 474.
4. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodard, J. (2009) Exploring Hyper-Heuristic Methodologies with Genetic Programming. Computational Intelligence, 6, 177-201.

5. Burke, E.K., Hyde, M.R., Kendall, K. (2006) Evolving Bin Packing Heuristics with Genetic Programming. Lecture Notes in Computer Science: Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), 4193, Springer, 860-869.
6. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Woodard, J. (2010) A Classification of Hyper-Heuristic Approaches. In the Handbook of Metaheuristics, International Series in Operations Research and Management Science, Volume 146, 449-468.
7. Chakhlevitch, K., Cowling, P. (2008) Hyperheursitics: Recent Developments. Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence, Vol. 136, 3-29.
8. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A. J., Petrovic, S., Burke, E. K. (2012) HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. European Conference on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012), LNCS series, Vol. 7245, Springer.
9. Ross, P. (2005) Hyper-Heuristics. In Burke, E.K. & Kendall, G., Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques, Chapter 17, 529-556, Springer.
10. Ross, P. (2014) Hyper-Heuristics. In Burke, E.K. & Kendall, G., Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, 611-638.
11. Ryser-Welch, P., Miller, J.F. (2014) A Review of Hyper-Heuristic Frameworks. In proceedings of the 50th Anniversary Convention of the AISB(Artificial Intelligence and the Simulation of Behaviour), http://doc.gold.ac.uk/aisb50/#s22.

2. Evolutionary Algorithm Hyper-Heuristics

This section will provide an overview of evolutionary algorithm hyper-heuristics and applications for each type of hyper-heuristic. Theoretical aspects will also be examined, e.g. using an evolutionary algorithm to explore the solution space directly versus searching the heuristic space. The section concludes by looking at the challenges associated with the implementation of evolutionary algorithm hyper-heuristics and potential solutions, e.g. parameter tuning, high runtimes and computational effort.

2.1 An Overview of Evolutionary Algorithm Hyper-Heuristics

Evolutionary algorithms have been used for both the selection and generation of low-level construction and perturbative hyper-heuristics. Genetic algorithms have essentially been used for selection and genetic programming for generation. More recently, variations of genetic programming have proven to be effective for generation.

2.2 Selection Hyper-Heuristics

Genetic algorithms explore the heuristic space by searching for a combination of low-level heuristics that will produce a solution (in the case of construction heuristics) or improve a solution (in the case of perturbative heuristics). Each chromosome is composed of genes representing the low-level heuristics. A case study is provided below to illustrate this.

Case Study: Selection Constructive Hyper-Heuristics

We will now design a selection constructive hyper-heuristic for solving the examination timetabling problem presented as an exercise in part 1. A genetic algorithm will be employed to explore the space of low-level construction heuristics. The construction heuristics are those listed in part 1:

- Largest degree (*l*)
- Largest enrollment (*e*)
- Largest weighted degree (*w*)
- Saturation degree (*s*)

Initial population generation: Each element of the population is a string comprised of low-level construction heuristics. Each character of the string is randomly chosen to be one of the four low-level heuristics listed above. The length of each string is equal to the number of examinations to be scheduled. Alternatively, the length of each string can be randomly chosen to be within a specified range which would improve the diversity of the initial population. If the length of the string is less than the number of exams to be scheduled, the string is applied again beginning at the first character. If the string is longer than the number of examinations to be scheduled only the first *n* heuristics are applied, where *n* is the number of examinations to be scheduled. An example of an element of the population is *lwws*.

Fitness evaluation: The fitness of each element of the population is calculated by using it to create a timetable. For example, consider the element *lwws*. The first examination will be scheduled using the largest degree heuristic, the next two using the largest weighted degree and the last examination using the saturation degree. The fitness measure of the element is a function of the hard and soft constraint cost of the timetable constructed using the element. The simplest function is the hard constraint cost plus one times the soft constraint cost.

Selection of parents: Each parent is selected using the tournament selection method. This method randomly chooses *n* elements of the population and the fittest element is returned as a parent. Selection is with replacement and thus an element may be a parent more than once.

Mutation: A low-level heuristic is randomly selected in an element and is replaced with another randomly selected heuristic from the heuristic set. For example suppose that *ssde* is a parent and *d* is the randomly chosen heuristic. If *w* is the randomly selected heuristic chosen to replace *d* then the resulting offspring will be *sswe*.

Crossover: A crossover point is selected for two parents and both the parents are crossed over at this point to produce two offspring. For example suppose that *ssde* and *lwws* are the chosen parents with 2 being the crossover point. Then the resulting offspring are *ssws* and *lwde*.

Exercises: Selection Hyper-Heuristics

1. Calculate the fitness of *dssd* and *lwws* in constructing a solution to the examination timetabling problem in the exercise in Part 1. Use the sum of the hard and soft constraint violations as the fitness function. The hard constraint is that there must be no clashes. The soft constraint cost is a measure of how the examinations are spread for students and is calculated to be the sum of *w(d)* where *d* is the distance between two examinations that have students in common and *w(d)* is calculated as follows: *w(1)* = 8, *w(2)* = 6, *w(3)* = 4, *w(4)* = 2 and *w(5)* = 1, i.e. the smaller the distance between periods the higher the weight allocated. Note for *n* > 5, *w(n)* = 0.
2. What changes would you make to the genetic algorithm hyper-heuristic to convert it to a selection perturbative hyper-heuristic to solve the examination timetabling problem?

Table 2.1 below provides examples of some of the areas that evolutionary algorithm selection hyper-heuristics have been successfully applied to.

Table 2.1: Selection Hyper-Heuristics

Approach	**Application**	**Hyper-Heuristic**	**Study**
Genetic algorithm	Trainer Scheduling Problem	Selection perturbative	Cowling et al. [3]
Messy GA	One and two dimensional bin-packing problems	Selection constructive	Lopez-Camacho et al. [17]
Genetic algorithm	Distributed staff trainer scheduling problem	Section perturbative	Han and Kendall [5]
Genetic algorithm	One dimensional bin-packing	Selection constuctive	Pillay[13]
Genetic algorithm	Educational timetabling	Selection constructive	Pillay [10], Pillay[11], Pillay [14], Pillay [15]

2.3 Generation Hyper-Heuristics

Genetic programming and variations of genetic programming (grammar based genetic programming and grammatical evolution) have chiefly been employed by hyper-heuristics to create new low-level construction and perturbative hyper-heuristics. Generative construction heuristics combine variables representing the characteristics of the problem or existing low-level construction heuristics to create new heuristics. Generative perturbative hyper-heuristics combine different heuristic selection and move acceptance components to create new perturbative heuristics. Evolutionary algorithm generative hyper-heuristics have been successfully applied to various domains, including educational timetabling, packing problems and vehicle routing problems.

Sidetrack

The aim of this sidetrack is to provide an overview of genetic programming if you are not familiar with this area as a knowledge of genetic programming is needed for the case studies and exercises that follow.

Genetic programming is a variation of genetic algorithms which searches a program space rather than a solution space. The aim is to evolve an optimal program which when executed will produce a solution to the problem. Each element of the population is usually represented as a parse tree. An example is illustrated in Figure 1. Each element is created by randomly choosing elements from a function and terminal set. The function set contains operators such as +, * and - in Figure 1. The terminal set contains elements that do not take arguments, e.g. variables and constants.

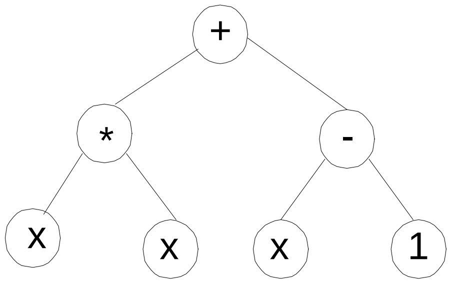


Figure 1. Example of a program

1. Suppose that you are required to implement a GP system to evolve an algorithm for the factorial of a positive number n.

    a) Define the terminal set.
    b) Define the function set.
    c) How will the fitness be calculated?
    d) What test cases would you use?
    e) What would a solution tree look like?
    f) Define mutation and crossover operators.

Case Study: Generation Constructive Hyper-Heuristics

We will now look at the design of a generation constructive hyper-heuristic for the one-dimensional bin-packing problem [5]. Genetic programming is used to evolve new low-level heuristics. The evolved heuristic takes a bin and the next item to be allocated to the bin as input and outputs a numerical value. If the value outputted is greater than zero the item is placed in the bin.

Primitives: The function set is comprised of arithmetic operators:

- Standard addition (+), subtraction (-) and multiplication (*).
- % - protected division which will return a 1 if the denominator is zero.
- < - will return a value of 1 if its first argument is less than or equal to its second and -1 otherwise.

The elements of the terminal set represent characteristics of the problem:

- F - the fullness of a bin, i.e. the sum of the sizes of the elements in the bin.
- C - the bin capacity
- S - the size of the item to place next.

Fitness evaluation: Each heuristic in the population is evaluated by using it to solve the problem instance. The fitness is calculated to be the difference of the number of bins used and the ratio of the sum of the items to the sum of the capacity of all bins used.

Example heuristic:

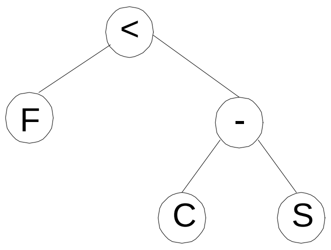


Table 2.2 lists examples of evolutionary algorithm generative hyper-heuristics:

Table 2.2: Generation Hyper-Heuristics

Approach	**Application**	**Hyper-Heuristic**	**Study**
Genetic programming	Packing problem	Generation constructive	Burke et al. [1], Burke et al. [2], Hyde [6]
Genetic programming	Multidimensional knapsack problem	Generation constructive	Drake et al. [4]
Grammatical evolution	One dimensional bin-packing problem	Generation constructive	Sotelo-Figueroa [25]
Genetic algorithm	Educational timetabling	Generation constructive	Pillay [9], Pillay[12], Pillay and Banzhaf [17], Pillay and Banzhaf [18]
Grammatical evolution	Educational timetabling	Generation perturbative	Sabar et al. [22]
Grammatical evolution	Educational timetabling	Generation perturbative	Sabar et al. [22]

Exercises: Generation Hyper-Heuristics

1. Design a generation constructive hyper-heuristic for the examination timetabling problem discussed in the previous sections.
2. Design a generation perturbative hyper-heuristic for the examination timetabling problem discussed in the previous sections.

2.4 Theoretical Aspects

Why are hyper-heuristics effective?

Hyper-heuristics have proven to be effective in solving combinatorial optimization problems and in some cases have proven to be more successful than searching the solution space directly. A hyper-heuristic results in greater exploration of the solution space indirectly than when directly searching the solution space. Changes made to potential solutions when working directly with the solution space are small resulting in not much and slow movement through the solution space, whereas small changes in the heuristic space result in large movements in the solution space resulting in quicker movement through the space and hence greater exploration [21] . A study of fitness landscapes for an evolutionary algorithm hyper-heuristic applied to timetabling has revealed that the fitness landscape has a big valley structure in which a global optimum is not isolated but surrounded by local optima which also accounts for the effectiveness of the hyper-heuristic for this domain [8].

Exercises: Hyper-Heuristics and the No Free Lunch Theorem

The no free lunch theorem for search and optimization states that:

"All algorithms that search for an extremum of a cost function perform exactly the same when averaged over all possible cost functions. So, for any search/optimization algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class".

The theorem is applicable to the search of finite spaces without re-sampling. Is there no free lunch for hyper-heuristics?

2.5 Challenges

The two major challenges with evolutionary algorithm hyper-heuristics are high runtimes and the need for parameter tuning. Evolutionary algorithms when applied directly to the problem usually have high runtimes which are further increased for evolutionary algorithm hyper-heuristics. For example, calculating the fitness in a selection evolutionary algorithm hyper-heuristic involves using each chromosome of a population to construct a potential solution which is then

evaluated. Distributed computing can be used to reduce the runtimes [23, 24]. There have already also been initiatives aimed at parallelizing hyper-heuristic frameworks, namely, *hMod* [26] and HyFlex [27]. Evolutionary algorithms have parameters, e.g. population size, probabilities of genetic operators, which have to be tuned which can also be time consuming. Parameter tuning tools such as iRace and ParamILS can be used. More recently hyper-heuristics have been used for the design of evolutionary algorithms and for determining parameter values. The parameter values are co-evolved.

References

1. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R. (2007) Scalability of Evolved Online Bin Packing Heuristics. In Proceedings of the Congress of Evolutionary Computation (CEC 2007) (Singapore). IEEE Press, 2530-2537.
2. Burke, E.K., Hyde, M., Kendall, G., Woodward, J. (2010) A Genetic Programming Hyper-Heuristic Approach for Evolving Two Dimensional Strip Packing Heuristics. IEEE Transaction on Evolutionary Computation, Vol. 16, No. 6, pp. 942-958.
3. Cowling, P., Kendall, G., Han, L. (2002) An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In proceedings of the 2002 Congress on Evolutionary Computation, Vol. 2, 12-17 May 2002, Honolulu, pp. 1185 - 1190, doi:10.1109/CEC.2002.1004411.
4. Drake, J.H., Hyde, M., Ibrahim, K., Ozcan, E. (2014) A Genetic Programming Hyper-Heuristic for the Multidimensional Knapsack Problem. Kybernetes, Vol. 43, Issue 9/10, pp.1500 - 1511.
5. Han, L., Kendall, G. (2003) Guided Operators for a Hyper-Heuristic Genetic Algorithm. AI 2003: Advances in Artificial Intelligence, Lecture Notes in Computer Science Volume 2903, 2003, pp. 807-820.
6. Hyde, M. (2010) A Genetic Programming Hyper-Heuristic Approach to Automated Packing. PhD thesis, School of Computer Science, University of Nottingham, UK.
7. Lopez-Camacho, E., Terashima-Marin, H., Ross, P., Ochoa, G. (2014) A Unified Hyper-Heuristic Framework for Solving Bin Packing Problems. Expert Systems with Applications, Vol. 41, pp. 6876-6889.
8. Ochoa, G., Qu, R., Burke, E.K. (2009) Analyzing the Landscape of Graph Based Hyper-Heuristics for Timetabling Problems. In proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09), 341-348.
9. Pillay N. (2009) Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem, in proceedings of the Multidisciplinary International Conference on Scheduling 2009 (MISTA 2009), Dublin, August 2009, 409-422.
10. Pillay, N. (2010) Evolving Hyper-Heuristics for a Highly Constrained Examination Timetabling Problem, in Proceedings of PATAT 2010, Belfast, UK, August 2010, 336-346.
11. Pillay, N. (2011) A Hyper-Heuristic Approach to Solving School Timetabling Problems, in Proceedings of MISTA (2011, Phoenix, Arizona, August 2011, 628-632.
12. Pillay, N. (2011) Evolving Heuristics for the School Timetabling Problem, in Proceedings of the 2011 IEEE Conference on Intelligent Computing and Intelligent Systems (ICIS 2011), Guangzhou, China, November 2011, Vol. 3, 281-286, IEEE Press.
13. Pillay, N. (2012) A Study of Evolutionary Algorithm Selection Hyper-Heuristics for the One-Dimensional Bin Packing Problem, South African Computer Journal, No. 48, 31-40.
14. Pillay, N. (2012) Evolving Hyper-Heuristics for the Uncapacitated Examination Timetabling Problem, Journal of the Operational Research Society, 63, 47-58.

15. Pillay, N. (2013) A Comparative Study of Hyper-Heuristics for Solving the School Timetabling Problem, in the proceedings of SAICSIT 2013, October 2013, South Africa, 278-285, ACM Press.
16. Pillay, N. (2014) A Review of Hyper-Heuristics for Educational Timetabling. Annals of Operations Research, August 2014, doi: :10.1007/s10479-014-1688-1.
17. Pillay N., Banzhaf W. (2007) A Genetic Programming Approach to the Generation of Hyper Heuristics for the Uncapacitated Examination Timetabling Problem, in Neves et al. (eds.), Progress in Artificial Intelligence, Lecture Notes in Artificial Intelligence, Vol. 4874, 223 234, Springer, 2007.
18. Pillay N., Banzhaf W. (2009) A Study of Heuristic Combinations for Hyper Heuristic Systems for the Uncapacitated Examination Timetabling Problem, European Journal of Operational Research, Vol. 197, September 2009, 482-491.
19. Poli, R. (2008) Some Ideas About No-Free Lunch for Hyper-Heuristics. http://citeseerx.ist .psu.edu/viewdoc/summary?doi=10.1.1.167.1971
20. Poli, R., Graff, M. (2009) There Is a Free Lunch for Hyper-Heuristics, Genetic Programming and Computer Scientists. Proceedings of the 12th European Conference on Genetic Programming (EuroGP '09), 195-207.
21. Qu, R., Burke, E.K. (2005) Analysing the High Level Heuristics within a Graph Based Hyper-Heuristic (Computer Science Technical Report No. NOTTCS-TR-2005-3). School of Computer Science and Information, University of Nottingham.
22. Sabar, N.R., Ayob, M., Kendall, G., Qu, R. (2013) Grammatical Evolution Hyper-heuristic for Combinatorial Optimization problems. IEEE Transactions of Evolutionary Computation, Vol. 17, No. 6, pp. 840-861.
23. Segredo, E., Segura, C., Leon, C. (2012) Analysing the Adaptation Level of Parallel Hyperheuristics Applied to Mono-objective Optimisation Problems. Nature Inspired Cooperative Strategies for Optimization (NICSO 2011), Studies in Computational Intelligence Volume 387, 169-182.
24. Segura, C., Miranda, G., Leon, C. (2011) Parallel Hyperheuristics for the Frequency Assignment Problems. Memetic Computing, 3(1), 33-49.
25. Sotelo-Figueroa, M.A., Soberanes, H.J. P., Carpio, J.M. (2013) Evolving and Reusing Bin Packing Heuristic through Grammatical Differential Evolution. In proceedings of the 2013 World Congress on Nature and Biologically Inspired Computing, 12-14 August 2014, Fargo, pp.92-98.
26. Urra, E., Cabrera-Paniagua, D., Cubillos, C. (2014) Towards a Distributed Hyperheuristic Deploy Architecture. Proceedings of Proceedings of the 7th Euro American Conference on Telematics and Information Systems (EAIS '14), Article no. 31, April 02 - 04 2014, Valparaiso, Chile.
27. Van Onsem, W., Demoen, B. (2013) ParHyFlex:A Framework for Parallel Hyper-Heuristics. BNAIC 2013: Proceedings of the 25th Benelux Conference on Artificial Intelligence, Delft, The Netherlands, November 7-8.

3. Recent and Emerging Directions

This section will look at recent and emerging directions in hyper-heuristics and evolutionary algorithm selection hyper-heuristics. There has been a large scale initiative by the hyper-heuristic community to promote the generalization of hyper-heuristics across domains. Two cross-domain challenges were organized for this purpose (CHeSC 2011 and CHeSC 2014).

Some interesting initial results have been obtained from hybridizing heuristic and solution space search and warrants further investigation. An overview of the emerging directions of using hyper-heuristics for evolutionary algorithm design, evolutionary algorithm hyper-heuristics for algorithm design and evolutionary algorithm hyper hyper-heuristics, e.g. combining selection constructive and selection perturbative hyper-heuristics, will be presented.

3.1 Cross Domain Selection Hyper-Heuristics

In order to promote the use of hyper-heuristics across domains the Cross Domain Heuristic Search Challenge (CHeSC 2011) was held in 2011. The HyFlex selection perturbative hyper-heuristic framework was developed for this purpose. The framework is coded in Java and can be downloaded by researchers to test different techniques to drive the hyper-heuristic. The results obtained can also be compared to those of the competitors. HyFlex provides an objective-function and low-level perturbative heuristics for the following problems:

- Flow-shop
- Personnel scheduling
- One-dimensional bin-packing
- Boolean satisfiability

Table 3.1 lists the low-level heuristics and method used to create an initial solution for each of the problems.

Table 3.1. HyFlex Method for Creating the Initial Solution and Low-Level Heuristics

Problem Domain	**Method to Create an Initial Solution**	**Low-Level Perturbative Heuristics**
Boolean satisfiability	Randomly assigning true and false values to each variable.	9 low-level heuristics: • local search – domain specific • mutational –domain specific, • ruin-and create - domain independent • crossover – domain independent
One-dimensional bin packing	Initial solution created using the first-fit heuristic.	7 low-level heuristics: • local search – domain specific • mutational –domain specific • ruin-and create - domain specific • crossover – domain independent
Flow-shop	Standard NEH procedure used to create an initial solution.	9 low-level heuristics: • local search – domain specific • mutational –domain specific • ruin-and create - domain independent • crossover – domain independent

Personnel scheduling	A local search perturbation heuristic to create an initial solution.	12 low-level heuristics: • local search – domain specific, • mutational –domain independent • ruin-and create - domain specific • crossover – domain specific

The performance of each hyper-heuristic was assessed over all four combinatorial optimization problems as well as two "hidden" domains, namely, the travelling salesman and vehicle routing problem. Subsequent to the competition an adaptive evolutionary algorithm to select perturbative heuristics which produces good results for the vehicle routing problem was developed [9]. CHeSC 2014 extends the cross-domain challenge by introducing "batching" and allowing multithreading strategies.

Resource

The source code for HyFlex together with publications on CHeSC 2011 and HyFlex can be accessed at http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html and the same for CHeSC2014 can be accessed at http://www.hyflex.org/chesc2014/.

3.2 Hybridization of Solution and Heuristic Space Search

The research conducted into hyper-heuristics for timetabling has shown the effectiveness of simultaneously exploring the heuristic space and the solution space [11]. The use of evolutionary algorithms for this purpose has not been investigated. The co-evolution of a heuristic and solution space could be effective for this.

3.3 Hyper-heuristics and Design

Both selection and generation evolutionary algorithm hyper-heuristics have been used to design other artificial intelligence methods such as decision trees for classification, evolutionary algorithms for the knapsack problem and ant colonization methods for the travelling salesman and vehicle routing problems. Methods used for the design include genetic algorithms and grammatical evolution. These selection perturbative hyper-heuristics have also been used to design algorithms for software clustering and scheduling and inspection planning in software development. Design has ranged from choosing parameter values, selection of selection methods and genetic operators, generation of operators, to the hybridization of different methods, which are considered as low-level heuristics to solve artificial intelligence and optimization problems. Table 3.2 below provides examples of evolutionary algorithm hyper-heuristics used for design:

Table 3.2: Evolutionary Algorithm Hyper-Heuristics for Design

Approach	**Design Aspect**	**Study**
Grammar based GP	Decision tree generation and design for data classification.	Barros et al. [1]

Evolutionary algorithms	Decision tree generation for data classification..	Barros et al. [2], Barros et al. [3]
NSGA-II	Combine neural networks into a stacked neural network.	Furtuna et al. [5]
Grammatical evolution	Selects operators and parameter values to solve a vehicle routing problem	Marshall et al. [11]
Genetic programming	Generates black box search algorithms to solve the deceptive trap problem.	Martin and Tauritz [12]
Grammatical evolution	Selects operators and parameter values for ant colonization to solve the travelling salesman problem.	Tavares and Pereira [20]
Genetic programming	Generates mutation operators for evolutionary programming	Hong et al. [6]
Grammatical evolution	Selects operators and parameter values for an evolutionary algorithm to solve the knapsack problem.	Lourenco et al. [9]

Hyper-heuristics have also proven to be effective for the design of evolutionary algorithms and has been used to decide on the control flow in an evolutionary algorithm by choosing when during the evolution process to use which operators, selection of parameter values and hybridization of evolutionary algorithms. Table 3.3 lists examples of the use of hyper-heuristics for evolutionary algorithm design:

Table 3.3: Hyper-Heuristics for Evolutionary Algorithm Design

Design Aspect	**Study**
Selection of recombination operator and selection method in differential evolution.	Tinoco and Coelllo [21]
Selects one of three multi-objective evolutionary algorithms to solve at each point of solving the problem	Maashi et al. [10]
Selection of crossover operator, mutation operator and selection method in an evolutionary algorithm.	Kumari and Srinivas [7], Kumaria and Srinivas [8]
Parameter tuning and determining the stopping condition in and implementation of MOEA.	Segredo et al. [19]
Generation of mutation operators in a genetic algorithm	Woodward and Swan [21]

3.4 Hyper-Hyper-Heuristics

Hyper-hyper-heuristics is the extension of the idea of hyper-heuristics to select or combine hyper-heuristics and generate new hyper-heuristics. Evolutionary algorithms can be used for the design of hyper-heuristics as well. A first study in this area uses gene expression programming to evolve a selection perturbative hyper-heuristic which is used in the HyFlex framework to generalize over 6 problem domains [18]. Gene expression programming evolves a heuristic selection and move acceptance component.

References

1. Barros, R. C., Basgulapp, M. P., de Carvalho, A. C., Freitas, A. A. (2012) A Hyper-Heuristic Evolutionary Algorithm for Automatically Designing Decision-Tree Algorithms. In proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation (GECCO '12), 1237-1244.
2. Barros, R.C., Basgalupp, M.P., de Carvalho, A.C.P.L.F., Freitas, A.A. (2013) Automatic Design of Decision-Tree Algorithms with Evolutionary Algorithms. Evolutionary Computation, Vol. 21, No. 4, 6 November, pp. 659-684.
3. Barros, R.C., Basgalupp, M.P., de Carvalho, A.C.P.L.F., Freitas, A.A. (2014) Evolutionary Design of Decision-Tree Algorithms Tailored to Microarray Gene Expression Data Sets. IEEE Transactions on Evolutionary Computation, Vol. 18, No. 6, December 2014, pp. 873-892.
4. Barros, R. C., Basgulapp, M. P., de Carvalho, A. C., Freitas, A. A. (2013) Automatic Design of Decision-Tree Algorithms with Evolutionary Algorithms. Evolutionary Computation, 21(4), 659-684.
5. Furtuna, R., Curteanu, S., Leon, F. (2012) Multi-Objective Optimization of a Stacked Neural Network Using an Evolutionary Algorithm Hyper-Heuristic. Applied Soft Computing, 2(2012), pp. 133-144.
6. Hong, L., Woodward, J., Li, J., Ozcan, E. (2013) Automated Design of Probability Distributions as Mutation Operators for Evolutionary Programming Using Genetic Programming. In proceedings of EuroGP 2013, Lecture Notes in Computer Science, 7831, pp. 85-96.
7. Kumari, A.C., Srinivas, K. (2012) Software Module Clustering Using a Fast Multi-Objective Hyper-Heuristic Evolutionary Algorithm. International Journal of Applied Information Systems, 5(6), 12-18.
8. Kumari, A. C., Srinivas, K. (2013) Scheduling and Inspection Planning in Software Development Projects Using Multi-Objective Hyper-Heuristic Evolutionary Algorithm. International Journal of Software Engineering and Application, 4(3), 45-57.
9. Lourenco, N., Pereira, F.B., Costa, E. (2013) The Importance of Learning Conditions in Hyper-Heuristics. In proceedings of GECCO '13, July 2013, Amsterdam, 1525-1532.
10. Maashi, M., Ozcan, E. & Kendall, G. (2014) A Multi-Objective Hyper-Heuristic Based on Choice Function. Expert Systems with Applications, Vol. 41, 4475-4493.
11. Marshall, R.J., Johnston, M., Zhang, M. (2014) Hyper-Heuristics, Grammatical Evolution and the Capacitated Vehicle Routing Problem. In proceedings of GECCO '14, Vancouver, Canada, pp. 71-72.
12. Martin, A.M., Tauritz, D.R. (2014) Multi-Sample Evolution of Robust Black-Box Search Algorithms. In proceedings of GECCO '14, Vancouver, Canada, pp. 195-196.
13. Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G. (2011) A New Hyper-heuristic Implementation in HyFlex: A Study on Generality. The 5th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA11), Arizona, USA.

14. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E. K. (2012) HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP 2012), LNCS series, Vol. 7245, Springer
15. Ochoa, G., Walker, J., Hyde, M., Curtois, T. (2012) Adaptive Evolutionary Algorithms and Extensions to the HyFlex Hyper-Heuristic Framework. Parallel Problem Solving from Nature - PPSN XII, Lecture Notes in Computer Science Volume 7492, 418-427.
16. Pillay, N. (2013) A Study of Hyper-Heuristics for Hybridizing Search. Advances in Artificial Intelligence - Proceedings of ALEA 2013, 9-12 September, Portugal,128-139.
17. Qu, R., Burke E.K. (2009). Hybridizations within a graph based hyper-heuristic framework for university timetabling problems. Journal of the Operational Research Society, 60(9), 1273–1285.
18. Sabar, N. R., Ayob, M., Kendall, G., Qu, R. (2014) The Automatic Design of Hyper-heuristic Framework with Gene Expression Programming for Combinatorial Optimization Problems. IEEE Transactions on Evolutionary Computation, April 2014, doi:10.1109/TEVC.2014.2319051.
19. Segredo, E., Segura, C., Leon, C. (2014) Control of Numeric and Symbolic Parameters with a Hybrid Scheme Based on Fuzzy Logic and Hyper-Heuristics. In proceedings of the 2014 IEEE Congress of Evolutionary Computation (CEC 2014), 6-11 July 2014, Beijing, China, pp. 1890-1897.
20. Tavares, J., Pereira, F. B. (2012) Automatic Design of Ant Algorithms with Grammatical Evolution. Genetic Programming, Lecture Notes in Computer Science, 7244, 206-217.
21. Tinoco, J.C.V., Coello, C.A.C. (2012) hypDE: A Hyper-Heuristic Based on Differential Evolution for Solving Constrained Optimization Problems. EVOLVE-A Bridge Between Probability, AISC 175, pp. 267-282.
22. Woodward, J., Swan, J. (2012) The Automatic Generation of Mutation Operators for Design. In proceedings of GECCO '12, Philadelphia, USA, 7-11 July 2012, pp. 67-74, ACM.