ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/226740468

Hyper-Heuristics: An Emerging Direction in Modern Search Technology

Chapter · January 2003

DOI: 10.1007/0-306-48056-5_16

CITATIONS
694

READS
1,924

6 authors, including:

Graham Kendall
University of Nottingham, Malaysia Campus
355 PUBLICATIONS **13,370** CITATIONS
SEE PROFILE

Emma Hart
Edinburgh Napier University
199 PUBLICATIONS **3,565** CITATIONS
SEE PROFILE

Peter Ross
132 PUBLICATIONS **4,542** CITATIONS
SEE PROFILE

Sonia Schulenburg
Level E Research Limited
24 PUBLICATIONS **1,142** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

examination timetabling problems View project

Diversity-based multi-objective evolutionary algorithms applied to VRPs View project

All content following this page was uploaded by Emma Hart on 16 May 2014.

The user has requested enhancement of the downloaded file.

Chapter #

HYPER-HEURISTICS: AN EMERGING DIRECTION IN MODERN SEARCH TECHNOLOGY

Authors : Edmund Burke[1], Emma Hart[2], Graham Kendall[1], Jim Newall[1], Peter Ross[2] and Sonia Schulenburg[2]
Affiliations : [1]The University of Nottingham, UK; [2]Napier University, UK

Abstract: This chapter introduces and overviews an emerging methodology in search and optimisation. One of the key aims of these new approaches, which have been termed *hyper-heuristics*, is to raise the level of generality at which optimisation systems can operate. An objective is that hyper-heuristics will lead to more general systems that are able to handle a wide range of problem domains rather than current meta-heuristic technology which tends to be customised to a particular problem or a narrow class of problems. Hyper-heuristics are broadly concerned with intelligently choosing the right heuristic or algorithm in a given situation. Of course, a hyper-heuristic can be (often is) a (meta-)heuristic and it can operate on (meta-)heuristics. In a certain sense, a hyper-heuristic works at a *higher* level when compared with the typical application of meta-heuristics to optimisation problems i.e. a hyper-heuristic could be thought of as a (meta)-heuristic which operates on *lower* level (meta-)heuristics. In this chapter we will introduce the idea and give a brief history of this emerging area. In addition, we will review some of the latest work to be published in the field.

Key words: Hyper-heuristic, meta-heuristic, heuristic, optimisation, search

1. INTRODUCTION

Meta-heuristics have played a key role at the interface of Artificial Intelligence and Operational Research over the last 10-15 years or so. The investigation of meta-heuristics for a wide and diverse range of application

areas has strongly influenced the development of modern search technology. Indeed, applications of meta-heuristic development can be found in such diverse areas as scheduling, data mining, stock cutting, medical imaging and bio-informatics, and many others. However, while such developments have deepened our scientific understanding of the search process and the automatic solution of large complex problems, it is true to say that the practical impact in commercial and industrial organisations has not been as great as might have been expected some years ago. Many state-of-the-art meta-heuristic developments are too *problem-specific* or too *knowledge-intensive* to be implemented in cheap, easy-to-use computer systems. Of course, there are technology provider companies that have brought such developments to market but such products tend to be expensive and their development tends to be very resource intensive. Often, users employ simple heuristics which are easy to implement but whose performance is often rather poor. There is a spectrum which ranges from cheap but fragile heuristics at one extreme and knowledge-intensive methods that can perform very well but are hard to implement and maintain at the other extreme. Many small companies are not interested in solving their optimisation problems to optimality or even close to optimality. They are more often interested in "*good enough – soon enough – cheap enough*" solutions to their problems. There is a current school of thought in meta-heuristic and search technology that contends that one of the main goals of the discipline over the next few years is to raise the level of generality at which meta-heuristic and optimisation systems can operate. This would facilitate the development of easy to implement (and cheap to implement) systems that can operate on a range of related problems rather than on one narrow class of problems. Of course, many papers in the literature discuss meta-heuristic and heuristic development on just one problem instance. Such papers provide a valuable insight into meta-heuristic development but they offer little assistance to a small company that simply cannot afford the significant amount of resources that are required to *tailor make* special purpose meta-heuristics for the company's own version of whatever optimisation problem it has.

This chapter is concerned with ***hyper-heuristics*** which is an emerging search technology that is motivated, to a large extent, by the goal of raising the level of generality at which optimisation systems can operate. The term has been defined to broadly describe the process of using (meta-)heuristics to *choose* (meta-)heuristics to solve the problem in hand. The majority of papers in the (meta-)heuristics area investigate the use of such approaches to operate directly on the problem. For example most of the papers on Evolutionary Computation in timetabling consider populations of timetables and the basic idea is that the population will evolve over a number of generations with the aim of generating a strong population. However, a

hyper-heuristic Evolutionary approach to timetabling [18, 1] would deal with a population of (meta-)heuristics for the timetabling problem and, over a number of generations, it is these (meta-)heuristics that would evolve. Another example of the use of hyper-heuristics is presented in [2] where a genetic algorithm evolves the choice of heuristic (in open shop scheduling) whenever a task is to be added to the schedule under construction. Of course, in 1994, the term hyper-heuristic did not exist and the process was called "evolving heuristic choice." These examples are discussed in more detail later in the chapter. There are many more examples of such approaches and indeed one of the main purposes of this chapter is to give an overview of them.

One of the main motivations for studying hyper-heuristic approaches is that they should be cheaper to implement and easier to use than problem specific *special purpose* methods and the goal is to produce good quality solutions in this more general framework. Of course, the overall aim of the hyper-heuristic goal goes beyond meta-heuristic technology. Indeed, there is current work which is investigating the development of machine learning approaches (such as case-based reasoning) to intelligently select heuristics according to the situation in hand [3, 4]. Such an approach is very much concerned with the same goal that motivates hyper-heuristic development. Actually, there is considerable scope for hybridising meta-heuristics with such machine learning approaches to intelligent heuristic selection [4].

This chapter is not intended to be an intensive survey of all the scientific papers which are related to the basic hyper-heuristic definition and that seek to satisfy the same objective. Rather, it is intended to give a brief overview of the idea and to set it within the context of the current state-of-the-art in meta-heuristic technology.

2. THE EMERGENCE OF HYPER-HEURISTICS

2.1 The concept and its origins

This section builds on the concept of hyper-heuristics and describes some early examples.

For many real-world problems, an exhaustive search for solutions is not a practical proposition. The search space may be far too big, or there may not even be a convenient way to enumerate the search space. For example, there may be elaborate constraints that give the space of feasible solutions a

very complex shape. It is common then to resort to some kind of heuristic approach, sacrificing a guarantee of finding an optimal solution for the sake of speed and perhaps also a guarantee of obtaining at least a certain level of solution quality. Enormous numbers of heuristics have been developed over the years, each typically justified either by experimental results or by an argument based on the specific problem class for which the heuristic in question had been tailored.

The term 'heuristic' is sometimes used to refer to a whole search algorithm and is sometimes used to refer to a particular decision process sitting within some repetitive control structure. Viewing heuristics as search algorithms, some authors have occasionally tried to argue for the absolute superiority of one heuristic over another. This practice started to die out when in 1995 Wolpert and MacReady [5] published their "No Free Lunch Theorem" which showed that, when averaged over all problems defined on a given finite search space, all search algorithms had the same average performance. This is an intuitively natural result since the vast majority of possible problems have no exploitable structure whatsoever, such as some form of global or local continuity, differentiability or regularity. They can only be defined by a complete lookup table. The "No Free Lunch Theorem" helped to focus attention on the question of what sorts of problems any given algorithm might be particularly useful for.

Long before the theorem was published it was clear that individual heuristics, however valuable, could have interesting quirks and limitations. Consider, for example, the topic of one-dimensional bin-packing. In its simplest incarnation there is an unlimited supply of identical bins and there is a set of objects to be packed into as few bins as possible. Each object has an associated scalar (think of it as the object's weight) and a bin cannot carry more than a certain total weight. The general task of finding an optimal assignment of objects to bins is NP-hard. A commonly used heuristic is 'largest first, first fit': sort the objects into decreasing order of weight, then, taking them in this order, put each object into the first bin into which it will fit (the bins in use are ordered too, according to when they first came into use). This heuristic has the benefits of simplicity and cheapness; it may not produce a solution that uses the minimal number M of bins, but it is known that it will not use more than $11M/9+4$ bins [6]. See [7] for a good survey of such results. A worst-case performance guarantee of this sort can be very reassuring if, for example, money is at stake. However, the following example due to Ron Graham of AT&T Labs shows that even this simple heuristic contains surprises. In this problem, the bins have capacity 524 and the 33 objects have the weights shown in table #-1.

Table #-1. A bin-packing problem

442	252	127	106	37	10	10
252	252	127	106	37	10	9
252	252	127	85	12	10	9
252	127	106	84	12	10	
252	127	106	46	12	10	

You can easily verify that the 'largest first, first fit' heuristic will produce a solution that uses seven bins, exactly filling all of them. However, if the object of weight 46 is removed from the problem, leaving only 32 objects to pack, the algorithm now uses eight bins. It is counter-intuitive that the heuristic should produce a worse result on a sub-problem than it does on the strictly larger problem.

As an example of heuristics that are tailored to specific types of problem, consider another one-dimensional bin-packing heuristic due to Djang and Finch [8], called by them 'Exact Fit'. The heuristic fills one bin at a time, as follows. Objects are taken largest first, and placed in the bin until the bin is at least one-third full. It then sets an allowable wastage w, initially w=0. The heuristic searches for one object that fills the bin to within w of its capacity. If this fails it searches for any two objects that fill the bin to within w. If this fails it searches for any three objects that fill the bin to within w. If this also fails it sets $w \Leftarrow w+1$ and repeats. As reported by the authors, this outperforms a variety of earlier heuristics on a large set of benchmark problems. However, these benchmark problems are of a particular sort: all the objects have a weight that is a significantly large fraction of a bin's capacity. Benchmark problems tend not to involve many objects with very small weights, because those objects can be treated as a form of 'sand' that can be used to fill up otherwise wasted space. It should be clear that the 'Exact Fit' heuristic can perform very badly on problems in which the objects are all very small. For example, consider a problem in which the bins have capacity 100 and there are 1000 objects each of weight 1. In the optimal solution ten bins are needed; the 'Exact Fit' heuristic will put at most 37 objects in any one bin and so will use 28 bins.

It would be plausible to argue that any self-respecting bin-packing heuristic should not start a new bin if there were existing partially-filled bins still capable of holding further items. Such heuristics would never use more than 2M bins because, if they did, there would be at least two bins whose combined contents fitted into one bin and so the heuristic should have at least been able to combine their contents or otherwise ensure that at least one of them was better filled. So there is a large class of heuristics whose worst-

case performance on a large class of problems is better than that of 'Exact Fit', even though 'Exact Fit' is very successful on benchmark problems that are generally acknowledged to be hard (but see [9] for a dissenting view).

2.2 The concept of hyper-heuristics

Since different heuristics have different strengths and weaknesses, it makes sense to see whether they can be combined in some way so that each makes up for the weaknesses of another. A simplistic way of doing this would be as shown in figure #-1.

```
If ( problemType(P) == p1 )
    apply(heuristic1, P);
else if (problemType(P) == p2)
    apply(heuristic2, P);
else ...
```

Figure #-1. A naive way to combine heuristics

One logical extreme of such an approach would be an algorithm containing an infinite switch statement enumerating all finite problems and applying the best known heuristic for each. There are many more practical problem-solving frameworks than this, such as the greedy randomised adaptive search procedure GRASP [10] which repeatedly fabricates a candidate solution *C* from parts on a so-called 'restricted candidate list', conducts a local search starting from *C* to find a locally optimal answer, and uses that to update information about desirability of parts and thus revise the restricted candidate list.

The key idea in hyper-heuristics is to use members of a set of known and reasonably understood heuristics to transform the state of a problem. The key observation is a simple one: the strength of a heuristic often lies in its ability to make some good decisions on the route to fabricating an excellent solution. Why not, therefore, try to associate each heuristic with the problem conditions under which it flourishes and hence apply different heuristics to different parts or phases of the solution process? For example, it should be clear from the preceding discussion that in bin-packing, some combination of the 'Exact Fit' procedure and the 'largest first, first fit' procedure should be capable of outperforming either of them alone.

The alert reader will immediately notice an objection to this whole idea. Good decisions are not necessarily easily recognizable in isolation. It is a sequence of decisions that builds a solution, and so there can be considerable

epistatis involved - that is, a non-linear interdependence between the parts. However, many general search procedures such as evolutionary algorithms can cope with a considerable degree of epistasis, so the objection is not necessarily fatal. And, on the positive side, there are some real potential benefits as far as real-world use is concerned. For example, in attempting to find a way to combine heuristic ingredients it can be possible to start from a situation in which a single, pure and unalloyed heuristic is used throughout the solution process. If it fails to survive the search process that is attempting to combine ingredients from different heuristics, it is because it wasn't good enough; the process has discovered something better.

Here, therefore, is one possible framework for a hyper-heuristic algorithm:

1. start with a set H of heuristic ingredients, each of which is applicable to a problem state and transforms it to a new problem state. Examples of such ingredients in bin-packing are a single top-level iteration of 'Exact Fit' or a single top-level iteration of 'largest first, first fit';
2. let the initial problem state be S_0
3. if the problem state is S_i then find the ingredient that is in some sense most suitable for transforming that state. Apply it, to get a new state of the problem S_{i+1};
4. if the problem is solved, stop. Otherwise go to 3.

There could be many variants of this, for example in which the set H varies as the algorithm runs or in which suitability estimates are updated across the iterations or in which the size of a single state transformation varies because the heuristic ingredients are dynamically parameterised. There is very considerable scope for research here.

2.3 Some historical notes

Intellectually, the concept of hyper-heuristics owes a debt to work within the field of Artificial Intelligence on automated planning systems. The earliest of such systems tried to devise a series of actions to achieve a given goal, usually a goal composed of a conjunct of required state features, by finding actions which would reduce the difference between the current state of the world and the desired state. This hill-climbing approach suffered all

the familiar problems of such a method. Later systems such as the DART logistical planning system [11] were much more sophisticated; DART was used in the Gulf War and was later judged by the US Chamber of Commerce to have saved more money than the US Government had spent on funding all forms of AI research over the previous 30 years. The focus eventually turned towards the problem of learning control knowledge; perhaps the best-known example is Minton's PRODIGY system [12] which used explanation-based learning to learn what action would be best to apply at each decision point.

This thread of AI research on planning and scheduling led to one of the earliest examples of a hyper-heuristic approach, the LR-26 scheduler within the COMPOSER system [13] was used for planning satellite communication schedules involving a number of earth-orbiting satellites and three ground stations. The problems involved are far from trivial. For example, certain satellites must communicate at some length with a ground station several times per day, with a predetermined maximum interval between communications, and yet the communication windows and choice of ground stations are constrained by the satellites' orbits. LR-26 treats the problem as a 0-1 integer programming problem involving hundreds of variables and thousands of linear constraints. The system handles many of the constraints by Langrangian relaxation, that is, by converting them to weighted components of the objective function that is to be maximised, so that if a constraint is violated the consequence is that the objective function value will be reduced by some amount that depends on the associated weight. The scheduler works by finding a partial schedule that may not satisfy all constraints, finding uncommitted variables within unsatisfied constraints and proposing values for them, and searching through a stack of such proposals to find good extensions to the partial schedule.

There are various possible heuristics used for each of several decision steps in this process. In particular, there are four different weight-adjustment heuristics, 9 primary and 9 secondary heuristic methods of ordering the set of unsatisfied constraints, 2 heuristic methods for proposing possible solutions to unsatisfied constraints and 4 heuristic methods for stacking these proposals for consideration. There are thus $4\times9\times9\times2\times4=2592$ possible strategies. To evaluate any one strategy properly meant testing it on 50 different problems which, the authors calculate, would have meant spending around 450 CPU days to evaluate all the strategies. Instead, COMPOSER applied a simple hill-climbing strategy thus restricting the search to $4+2+(9\times4)+9=51$ strategies, at a tolerable cost of 8.85 CPU days. The outcome was "a significant improvement in performance", in terms of solution speed and quality and in the number of problems that could be solved at all, compared to the originally-used strategy. A potential

disadvantage of the approach lies in the assumption that the training set is in some way representative of future problems. In the area of satellite communication scheduling this is unlikely to be true – new satellites can have new orbits and different and potentially less demanding communication window requirements, for example.

A second example of the use of a hyper-heuristic approach concerns open-shop scheduling problems. In such problems there are, say, *j* jobs each consisting of a certain number of tasks. A task consists of visiting a certain machine for a certain task-specific length of time. The tasks associated with a job can be done in any order – if it was a fixed job-specific order then it would be a job-shop problem instead. Fang et al [2] used a genetic algorithm which built solutions as follows. A chromosome was a series of pairs of integers $[t_0,h_0,t_1,h_1,...]$ interpreted from left to right and meaning, for each *i*, 'consider the t_i-th uncompleted job (regarding the list of uncompleted jobs as circular, so that this is always meaningful) and use heuristic h_i to select a task to insert into the schedule in the earliest place where it will fit'. Examples of heuristics used included:

- choose the task with largest processing time;
- choose the task with shortest processing time;
- of those tasks which can be started as early as possible, choose the one with largest processing time;
- among those operations which can be inserted into any gap in the schedule, pick the one that best fills a gap (that is, leaves as little idle time as possible);
- and so on.

This approach, termed *evolving heuristic choice*, provided some excellent results on benchmark problems including some new best results at that time. Nevertheless, there are some caveats. First, what is the nature of the space being searched? It is likely that in many cases, several heuristics might lead to the same choice of task, so it may be that the search space is not nearly as large as it might first seem. Second, is the genetic algorithm necessary or might some simpler, non-population-based search algorithm do as well? Third, if there are *n* tasks then there are effectively *n*-1 pairs of genes (there is no choice when it comes to inserting the very last task), so if *n* is large the chromosome will be very long and there is a real risk that genetic drift will have a major impact on the result. Fourth, the process evolves solutions to

individual problems rather than creating a more generally applicable algorithm.

Schaffer [14] carried out an early investigation into the use of a genetic algorithms which select heuristics. The paper describes the Philips FCM SMD robot and the heuristic selection genetic algorithm.

In [15] Hart and Ross considered job-shop scheduling problems. The approach there relies on the fact that there is an optimal schedule which is *active* - an active schedule is one in which, to get any task completed sooner you would be forced to alter the sequence in which tasks get processed on some machine, and to do that you would force some other task to be delayed. The optimal schedule might even be *non-delay*, that is, not only active but also such that no machine is ever idle when there is some task that could be started on it. There is a widely-used heuristic algorithm due to Giffler and Thompson [16] that generates active schedules:

1. let C = the set of all tasks that can be scheduled next
2. let t = the minimum completion time of tasks in C, and let m= machine on which it would be achieved
3. let G = the set of tasks in C that are to run on m whose *start* time is $< t$
4. choose a member of G, insert it in the schedule
5. go to 1.

Note that step 4 involves making a choice. This algorithm can be simplified so as to generate non-delay schedules by only looking at the earliest-starting tasks:

1. let C = the set of all tasks that can be scheduled next
2. let G = the subset of C that can start at the earliest possible time
3. choose a member of G, insert it in the schedule
4. go to 1.

Note that step 3 also involves making a choice. The idea in [15] is to use a chromosome of the form $[a_1,h_1,a_2,h_2,\ldots]$, where the chromosome is again read from left to right and a_i is 0 or 1 and indicates whether to use an iteration of the Giffler and Thompson algorithm or an iteration of the non-delay algorithm to place one more task into the growing schedule, and h_i indicates which of twelve heuristics to use to make the choice involved in

either algorithm. Again, this produced very good results on benchmark problems when tested on a variety of criteria. The authors looked at the effective choice at each stage and found that fairly often, there were only one or two tasks to choose between, and never more than four. The space being searched is thus much smaller than the chosen representation would suggest; many different chromosomes represent the same final schedule. The authors also observed that, in constructing a schedule, it is the early choices that really matter. For example, if the first 50% of choices are made according to what the evolved sequence of heuristic choices suggests, but the other 50% are made at random, then the result is still a very satisfactory schedule. This suggests at least a partial answer to the worry raised above about using very long chromosomes: quite simply, do not do it. Instead, use a much shortened chromosome just to evolve the early choices and then resort to using a fixed heuristic to complete the construction.

A real-world example of using a hyper-heuristic approach is described in [17], where the problem is to schedule the collection and delivery of live chickens from farms all over Scotland and Northern England to one of two processing factories, in order to satisfy the set of orders from supermarkets and other retailers. The customer orders would change week by week and sometimes day by day. The task was to schedule the work done by a set of ‘catching squads’ who moved around the country in mini-buses, and a set of lorries who would ferry live chickens from each farm back to one of the factories. The principal aim was to keep the factories supplied with work without requiring live chickens to wait too long in the factory yard, for veterinary and legal reasons. This was complicated by many unusual constraints. For example, there were several types of catching squad, differentiated by which days they worked, when they started work, what guaranteed minimum level of work they had been offered and what maximal amount they could do. There were constraints on the order in which farms could be visited, to minimise potential risks of spreading diseases of chickens. There were constraints on the lorry drivers and on how many chickens could be put into a single 'module’ (a tray-like container) and variations in the number of such modules different lorries could carry, and so on. Overall, the target was not to produce optimal schedules in cost terms, because the work requirements could anyway change at very short notice but it was not generally practicable to make very large-scale changes to staff schedules at very short notice. Instead, the target was to create good schedules satisfying the many constraints, that were also generally similar to the kinds of work pattern that the staff were already familiar with, and to do so quickly and reliably. The eventual solution used two genetic algorithms. One used a heuristic selection approach to decompose the total set of

customer orders into individual tasks and assign those tasks to catching squads. The other started with these assignments and evolved the schedule of lorry arrivals at each factory; from such schedules it was possible to reason backwards to determine which squad and lorry had to arrive at each farm at which times, and thus construct a full schedule for all participants. In the first genetic algorithm, the chromosome specified a permutation of the customer orders and then two sequences of heuristic choices. The first sequence of choices worked through the permutation and split each order into loads using simple heuristics about how to partition the total customer order into convenient workload chunks; the second sequence of choices suggested how to assign those chunks to catching squads. The end result did meet the project's requirements but, like many other practically-inspired problems, it was not feasible to do a long-term study to determine just how crucial each of the ingredients was to success. However, the authors report that a more conventional permutation-based genetic algorithm approach to this scheduling task had not been successful.

In the above examples (apart from the hill-climbing approach used in the LR-26 satellite communication scheduler), a genetic algorithm was used to evolve a fine-grained sequence of choices of heuristics, with one choice per step in the process of constructing a complete solution. Although the end results were generally good, this is still somewhat unsatisfactory because the method evolves solutions only to specific instances of problems and may not handle very large problems well. A different kind of hyper-heuristic approach that begins to tackle such objections is described in [18]. That paper is concerned with solving large-scale university exam timetabling problems. There are a number of fixed time-slots at which exams can happen, and there are rooms of various sizes. Two exams cannot happen at the same time if there is any student who takes both; more than one exam can happen in a room at the same time if no student takes both and the room is large enough. Certain exams may be constrained to avoid certain time-slots when, for example, an appropriate invigilator is unavailable. There are also some 'soft' constraints which it would be good to respect but which can be violated if necessary. For example, it is desirable that no student should have to sit exams in consecutive time-slots on the same day and it is often desirable that very large exams should happen early so as to permit more time for marking them before the end of the whole exam period. Such problems can involve thousands of exams and tens of thousands of students.

The approach taken in [18] is to suppose that there is an underlying timetable construction algorithm of the general sort shown in figure #-2.

```
/* Do first phase of construction */
while(condition(X) == FALSE)
```

```
{
   event = apply_event_heuristic(H1);
   timeslot = apply_slot_heuristic(H2,event);
   add_to_timetable(event, timeslot);
}
/* Do second phase of construction */
while(not(completed()))
{
   event = apply_event_heuristic(H3);
   timeslot = apply_slot_heuristic(H4,event);
   add_to_timetable(event, timeslot);
}
```

Figure #-2. A two-phase timetable building algorithm

The idea is to use a genetic algorithm to evolve the choices of H1, H2, H3 and H4 and the condition X which determines when to switch from the first phase to the second. Some of the heuristics and possible conditions involved further parameters, such as whether and how much to permit backtracking in making a choice, or switching to phase 2 after placing N events. The rationale for using such an algorithm is that many timetabling problems necessitate solving a certain sort of problem initially (for example, a bin-packing problem to get the large exams well packed together if room space is in short supply) but a different sort of problem in the later stage of construction. Soft constraints are handled within the heuristics, for example, in order to cut down the chances of a student having exams in adjacent time-slots an heuristic might consider time-slots in some order that gave adjacent slots a very low priority.

A chromosome was evaluated by constructing the timetable and assessing the quality of the result. Interestingly, this method solved even very large timetabling problems very satisfactorily in under 650 evaluations. The authors also conducted a brute-force search of the space of chromosomes in order to check whether the genetic algorithm was delivering very good-quality results (at least as far as the chosen representation would permit) whilst visiting only a tiny proportion of the search space, and were able to confirm the truth of this. However, they did not also examine whether the discovered instances of the framework described in figure #-2 could be applied successfully to other problems originating from the same university; this is a topic for further research.

3. HYPER-HEURISTIC FRAMEWORK

This section describes a particular hyper-heuristic framework that has been presented in [19,20,25,26,27,28]. As has been discussed earlier, a meta-heuristic typically works on the problem directly, often with domain knowledge incorporated into it. However, this hyper-heuristic framework operates at a higher level of abstraction and often has no knowledge of the domain. It only has access to a set of low level heuristics that it can call upon, but with no knowledge as to the purpose or function of a given low level heuristic. The motivation behind this suggested approach is that once a hyper-heuristic algorithm has been developed then new problem domains can be tackled by only having to replace the set of low level heuristics and the evaluation function, which indicates the quality of a given solution.

A diagram of a general hyper-heuristic framework is shown in figure #-3.

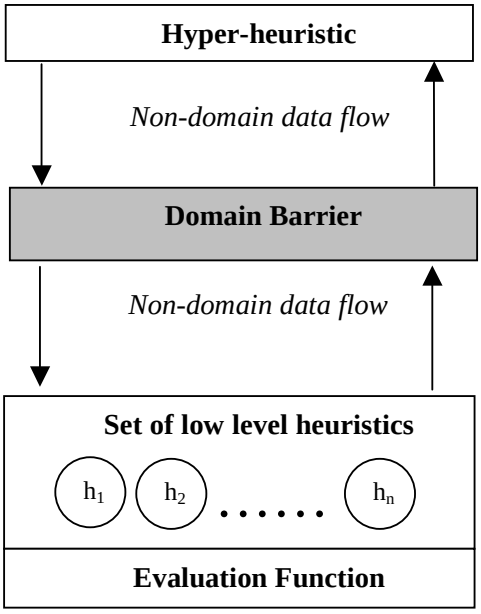

Figure #-3. Hyper-heuristic Framework

The figure shows that there is a barrier between the low level heuristics and the hyper-heuristic. Domain knowledge is not allowed to cross this barrier. Therefore, the hyper-heuristic has no knowledge of the domain under which it is operating. It only knows it has *n* low level heuristics on which to call and it knows it will be passed the results of a given solution once it has been evaluated by the evaluation function.

There is, of course, a well defined interface between the hyper-heuristic and the low level heuristics. The reasons for this are two-fold

1. It allows the hyper-heuristic to communicate with the low level heuristics using a standard interface, otherwise the hyper-heuristic would need a separate interface for each low level heuristic which is obviously nonsensical. In addition, it facilitates the passing of non-domain data between the low level heuristics and the hyper-heuristic (and vice versa). For example, the interface developed in [25] includes components such as the result of the evaluation function and the CPU time taken by the low level heuristic (which, equally, could be calculated by the hyper-heuristic). In addition, we have also included a component that allows us to "tell" a low level heuristic how long it has to run. The motivation behind this idea is that we call each heuristic in turn giving it a specified amount of time and the heuristic that performs the best, in the time allowed, is the one that is applied to the current solution. In conjunction with this component, the interface also defines if the low level heuristic should apply its changes to the current solution or if it should just report what effect it would have *if* it did apply those changes. The idea is that the hyper-heuristic can ask each low level heuristic how well it will do against a given solution. The hyper-heuristic can then decide which heuristic (or set of heuristics) should be allowed to update the solution. We have not fully explored these ideas yet but the interface will allow us to investigate them at an appropriate time.
2. It allows rapid development for other domains. When implementing a new problem, the user has to supply a set of low level heuristics and a suitable evaluation function. If the low level heuristics follow a standard interface the hyper-heuristic does not have to be altered in any way. It is able to simply start solving the new problem as soon as the user has supplied the low level heuristics and the evaluation function. That is, as stated above, we aim for a hyper-heuristic to operate at a higher level of abstraction than a meta-heuristic approach.

An example of a hyper-heuristic operating at a higher level of abstraction can be seen in the work of Cowling, Kendall and Soubeiga [19,25,26,27]. In [25] a hyper-heuristic approach was developed and applied to a sales summit problem (a problem of matching suppliers to potential customers at a sales seminar). In [25] the hyper-heuristic had access to 10 low level heuristics which included *removing a delegate from a meeting with a particular supplier, adding a delegate to supplier to allow them to meet* and *remove*

meetings from a supplier who has more than their allocation. In [26] the hyper-heuristic was modified so that it automatically adapted some of its parameters but, again, the sales summit problem was used as a test bench. In [27], the hyper-heuristic remained the same but a new problem was introduced. This time, the problem was scheduling third year undergraduate project presentations for a UK university. Eight low level heuristics were developed, which included *replace a staff member in a given session*, *move one presentation from one session to another* and *swap the 2nd marker for a given session*. The changes were such that only the low level heuristics and the evaluation function was changed. The hyper-heuristic remained the same in the way that it chose which low level heuristic to call next. Good quality solutions were produced for this different problem domain.

This idea was further developed in [19] when the same hyper-heuristic approach was applied to nurse rostering.

The only information that the hyper-heuristic has access to in this framework is data which is common to all problem types and which it decides to record as part of its internal state. For example, the hyper-heuristic might store the following

- How much CPU time a given heuristic used when it was last called?
- The change in the evaluation function when the given heuristic was called?
- How long (possibly in CPU time) has elapsed since a given heuristic has not been called?

The important point is that the hyper-heuristic has no knowledge as to the function of each heuristic. For example, it will not know that one of the heuristics performs 2-opt for the Traveling Salesman Problem. Indeed, it does not even know that the problem being optimised is the Traveling Salesman Problem.

Of course, the hyper-heuristic designer is allowed to be as imaginative as he/she wishes within the constraints outlined above. For example, the internal state of the hyper-heuristic could store how well pairs of heuristics operate together. If the hyper-heuristic calls one heuristic followed by another, does this lead to a worsening of the evaluation function after the first call but a dramatic (or even slight) improvement after the second heuristic has been called? Therefore, one hyper-heuristic idea might be to store data about pairs (triples etc.) of heuristics that appear to work well when called consecutively but, if called in isolation, each heuristic, in general, performs badly.

Using its internal state, the hyper-heuristic has to decide which low level heuristic(s) it should call next. Should it call the heuristic that has led to the largest improvement in the evaluation function? Should it call the heuristic that runs the fastest? Should it call the heuristic that has not been called for the longest amount of time? Or, and more likely, should it attempt to balance all these factors (and more) to make an informed decision as to which low-level heuristic (or pairs, triples etc.) it should call next.

The internal state of the hyper-heuristic is a matter for the designer, as is the process to decide on which heuristic to call next, but once a suitable hyper-heuristic is in place then the hope is that it will perform reasonably well across a whole range of problems and not just the one for which it was originally implemented.

The hyper-heuristic framework described above is the framework adopted in [25,26,27], where the hyper-heuristic maintains an internal state which is accessed by a *choice* function to decide which low level heuristic to call next. The choice function is a combination of terms which considers recent performance of each low level heuristic (denoted by f_1), recent performance of pairs of heuristics (denoted by f_2) and the amount of time since a given heuristic has been called (denoted by f_3). Thus we have

$$f(H_k) = \alpha f_1(H_k) + \beta f_2(H_j, H_k) + \delta f_3(H_k)$$

where

H_k is the kth heuristic
α, β and δ are weights which reflect the importance of each term. It is these weights that are adaptively changed in [26].
$f_1(H_k)$ is the recent performance of heuristic H_k
$f_2(H_j,H_k)$ is the recent performance of heuristic pair H_j,H_k
$f_3(H_k)$ is a measure of the amount of time since heuristic H_k was called.

f_1 and f_2 are aiming to intensify the search while f_3 attempts to add a degree of diversification. The maximal value, $f(H_k)$, dictates the heuristic which is called next.

A different type of hyper-heuristic is described in [28] where a genetic algorithm is used to evolve a sequence of calls to the low level heuristics. In effect, the genetic algorithm replaces the choice function described above and each member of the population (that is, each chromosome) is evaluated by the solution it returns when applying the heuristics in the order denoted

by the chromosome. In later work by Han, Kendall and Cowling [20] the chromosome length is allowed to adapt and longer chromosomes are penalised on the basis that they take longer to evaluate.

However, in [20] and [28] the concept of the domain barrier remains and the genetic algorithm has no knowledge of the problem. It simply tries to evolve a good sequence of heuristic calls.

4. MODERN HYPER-HEURISTIC APPROACHES

There has been much recent research directed to the various aspects of Hyperheuristics. An example that follows the framework described in Section 2.2 can be found in [21]. The focus here is on learning solution processes applicable to many problem instances rather than learning individual solutions. Such process would be able to choose one of various simple, well-understood heuristics to apply to each state of a problem, gradually transforming the problem from its initial state to a solved state. The first use of such model has been applied to the one-dimensional bin-packing problem described in Section 2.1. In this work, an accuracy-based Learning Classifier System (XCS) [22] has been used to learn a set of rules that associates characteristics of the current state of a problem with, in this case, eight different heuristics, two of which have been explained in Section 2.1 (largest-first-first-fit and exact-fit). The set of rules is used as follows: given the initial problem characteristics *P*, a heuristic *H* is chosen to pack a bin, gradually altering the characteristics of the problem that remains to be solved. At each step, a rule appropriate to the current problem state *P'* is chosen, and the process repeats until all items have been packed.

The approach is tested using 890 benchmark bin-packing problems, of which 667 were used to train the XCS and 223 for testing. The combined set provides a good test of whether the system can learn from a very varied collection of problems. The method (HH) achieved optimal results on 78.1% of the training problems, and 74.6% of the remaining test solutions. This compares well with the best single heuristic (the author's improved version of exact-fit) which achieved optimality 73% of the time. Largest-first-first-fit, for instance, achieves optimality in 62.2%, while another one of the heuristics used named 'next-fit' reaches optimality in 0% of the problems. Even though the results of the best heuristic might seem close to HH, it is also noteworthy that when HH is trained purely on some of the harder problems (which none of the component heuristics could solve to optimality alone), it manages to solve seven out of ten of these problems to optimality.

Improvements over this initial approach are reported in [23], where a new heuristic (*R*) that uses a random choice of heuristics has been introduced to

compare results of HH. *R* solves only 56.3% of the problems optimally, while HH reaches 80%. This work also looks more closely at the individual processes evolved arising from two different reward schemes presented during the learning process of HH. The behaviour of HH is further analysed and compared to the performance of the single heuristics. For instance, in HARD9, one of the very difficult problems used in the set, HH learned that the combination of only two of the eight single heuristics (here called *h1* to *h8*) reaches optimality when none of the heuristics used individually achieved it. The solution found by HH used only 56 bins, while the best of the individual heuristics used alone needs 58 bins. The best reported result, using a method not included in any of the heuristics used by HH, used 56 bins (see [24], Data Set 3). Improvements of this type were found in a large number of other problems, including all the HARD problems. The approach looks promising and further research is being carried out to evaluate it in other problem domains.

Cowling, Han, Kendall and Soubeiga (previous section) propose a hyper-heuristic framework existing at a higher level of abstraction than meta-heuristic local search methods, where different neighbourhoods are selected according to some “choice function”. This choice function is charged with the task of determining which of these different neighbourhoods is most appropriate for the current problem. Traditionally the correct neighbourhoods are selected by a combination of expert knowledge and/or time consuming trial and error experimentation, and as such the automation of this task offers substantial potential benefits to the rapid development of algorithmic solutions. It naturally follows that an effective choice function is critical to the success of this method, and this is where most research has been directed [19,25,26,27]. See the previous section for a more detailed discussion.

The approach of Cowling, Kendall and Han to a trainer scheduling problem [28] also fits into this class of controlling the application of low level heuristics to construct solutions to problems. In this approach a “Hyper-genetic algorithm” is used to evolve an ordering of 25 low-level heuristics which are then applied to construct a solution. The fitness of each member of the genetic algorithm population is determined by the quality of the solution it constructs. Experimental results show that the method outperformed the tested conventional genetic and memetic algorithm methods. It also greatly outperformed any of the component heuristic methods, albeit in a greatly multiplied amount of CPU time.

Another approach proposed by Burke and Newall to examination timetabling problems [29] uses an adaptive heuristic to try and improve on an initial heuristic ordering. The adaptive heuristic functions by first trying

to construct a solution by initially scheduling exams in an order dictated by the original heuristic. If using this ordering means that an exam cannot be acceptably scheduled, it is promoted up the order in a subsequent construction. This process continues either until the ordering remains static (all exams can be scheduled acceptably), or until a pre-defined time limit expires. The experiments showed that the method can substantially improve quality over that of the original heuristic. The authors also show that even when given a poor initial heuristic acceptable results can still be found relatively quickly.

Other approaches that attempt to harness run-time experience include the concept of Squeaky wheel optimisation proposed by Joslin and Clements [30]. Here a greedy constructor is applied to a problem, followed by an analysis phase that identifies problematic elements in the produced solution. A prioritiser then ensures that the greedy constructor concentrates more on these problematic elements next time, or as the authors phrase it: "The squeaky wheel gets the grease". This cycle is iterated until some stopping criteria are met. Selman and Kautz propose a similar modification to their GSAT procedure [31]. The GSAT procedure is a randomised local search procedure for solving propositional satisfiability problems. It functions by iteratively generating truth assignments and then successively "flips" the variable that leads the greatest increase in clauses satisfied, in a steepest descent style. The proposed modification increases the "weight" of clauses if they are still unsatisfied at the end of the local search procedure. This has the effect that on subsequent attempts the local search element will concentrate more on satisfying these clauses and hopefully in time lead to full satisfaction of all clauses.

Burke et al [3] investigate the use of the case based reasoning paradigm in a hyper-heuristic setting for selecting course timetabling heuristics. In this paper, the system maintains a case base of information about which heuristics worked well on previous course timetabling instances. The training of the system employs knowledge discovery techniques. This work is further enhanced by Petrovic and Qu [4] who integrate the use of Tabu Search and Hill Climbing into the Case Based Reasoning system.

5. CONCLUSIONS

It is clear that hyper-heuristic development is going to play a major role in search technology over the next few years. The potential for scientific progress in the development of more general optimisation systems, for a wide variety of application areas, is significant. For example, Burke and Petrovic [32] discuss the scope for hyper-heuristics for timetabling problems

and this is currently an application area that is seeing significant research effort [1,3,4,18,25,29,32]. Indeed, in 1997, Ross, Hart and Corne [1] said, "However, all this naturally suggests a possibly worthwhile direction for timetabling research involving Genetic Algorithms. We suggest that a Genetic Algorithm might be better employed in searching for a good algorithm rather than searching for a specific solution to a specific problem." Ross, Hart and Corne's suggestion has led to some important research directions in timetabling. We think that this suggestion can be generalised further and we contend that a potentially significant direction for meta-heuristic research is to investigate the use of hyper-heuristics for a wide range of problems. As this chapter clearly shows, this work is already well underway.

ACKNOWLEDGEMENTS

The authors are grateful for support from the UK Engineering and Physical Sciences Research Council (EPSRC) under grants GR/N/36660, GR/N/36837 and GR/M/95516.

REFERENCES

[1] P.Ross, E.Hart and D.Corne. Some Observations about GA-based Exam Timetabling. In LNCS 1408, *Practice and Theory of Automated Timetabling II : Second International Conference, PATAT 1997*, Toronto, Canada, August 1997, selected papers (eds Burke E.K. and Carter M), Springer-Verlag, pp 115-129

[2] H-L Fang, P.M.Ross and D.Corne. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems", in *Proceedings of ECAI 94: 11th European Conference on Artificial Intelligence*, A. Cohn (ed), pp 590-594, John Wiley and Sons Ltd, 1994

[3] E.K.Burke, B.L.MacCarthy, S.Petrovic and R.Qu. Knowledge Discovery in a Hyper-heuristic for Course Timetabling using Case Based Reasoning. To appear in the Proceedings of the *Fourth International Conference on the Practice and Theory of Automated Timetabling (PATAT'02)*, Ghent, Belgium, August 2002

[4] S.Petrovic and R.Qu. Case-Based Reasoning as a Heuristic Selector in a Hyper-Heuristic for Course Timetabling. To appear in Proceedings of the *Sixth International Conference on Knowledge-Based Intelligent Information & Engineering Systems (KES'2002)*, Crema, Italy, Sep 2002

[5] D.Wolpert and W.G.MacReady. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation*, 1(1) : pp 67-82, 1997

[6] D.S.Johnson. *Near-optimal bin-packing algorithms*. PhD thesis, MIT Department of Mathematics, Cambridge, Mass., 1973

[7] E.G.Coffman, M.R.Garey, and D.S. Johnson. Approximation algorithms for bin packing: a survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pp 46-93. PWS Publishing, Boston, 1996

[8] P.A.Djang and P.R.Finch. Solving one dimensional bin packing problems. Available as http: //www.zianet.com/pdjang/binpack/paper.zip

[9] I. P.Gent. Heuristic solution of open bin packing problems. *Journal of Heuristics*, 3(4): pp 299-304, 1998

[10] L.S.Pitsoulis and M.G.C.Resende. Greedy randomized adaptive search procedures. In P.M.Pardalos and M.G.C.Resende, editors, *Handbook of Applied Optimization*, pp 168–181. OUP, 2001

[11] S.E.Cross and E.Walker. Dart: applying knowledge-based planning and scheduling to crisis action planning. In M.Zweben and M.S.Fox, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994

[12] S.Minton. *Learning Search Control Knowledge: An Explanation-based Approach*. Kluwer, 1988

[13] J.Gratch, S.Chein, and G.de Jong. Learning search control knowledge for deep space network scheduling. In *Proceedings of the Tenth International Conference on Machine Learning*, pp 135-142, 1993

[14] J.D.Schaffer, Combinatorial Optimization by Genetic Algorithms: The Value of the Phenotype/Genotype Distinction. In *First International Conference on Evolutionary Computing and its Applications (EvCA'96),* E.D.Goodman, V.L.Uskov, W.F.Punch III (eds), Russian Academy of Sciences, Moscow, Russia, June 24-27 1996, pp.110-120. Publisher: Institute for High Performance Computer Systems of the Russian Academy of Sciences, Moscow, Russia

[15] E.Hart and P.M.Ross. A heuristic combination method for solving job-shop scheduling problems. In A.E.Eiben, T.Back, M.Schoenauer, and H-P.Schwefel, editors, *Parallel Problem Solving from Nature V*, LNCS 1498, pages 845-854. Springer-Verlag, 1998

[16] B.Giffler and G.L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8(4): pp 487-503, 1960

[17] E.Hart, P.M.Ross, and J. Nelson. Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, 6(1): pp 61-80, 1998

[18] H.Terashima-Marín, P.M.Ross, and M.Valenzuela-Rendón. Evolution of constraint satisfaction strategies in examination timetabling. In W. Banzhaf et al., editor, *Proceedings of the GECCO-99 Genetic and Evolutionary Computation Conference*, pp 635-642. Morgan Kaufmann, 1999

[19] P.Cowling, G.Kendall and E.Soubeiga. Hyperheuristics: a robust optimisation method applied to nurse scheduling. Technical Report NOTTCS-TR-2002-6 (submitted to PPSN 2002 Conference), University of Nottingham, UK, School of Computer Science & IT, 2002

[20] L.Han, G.Kendall and P.Cowling. An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem. Technical Report NOTTCS-TR-2002-5 (submitted to SEAL 2002 Conference), University of Nottingham, UK, School of Computer Science & IT, 2002

[21] P.Ross, S.Schulenburg, J.G.Marín-Blázquez and E.Hart. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. Accepted for *Genetic and Evolutionary Computation Conference (GECCO 20020) 2002, New York*, July 9-13 2002

[22] S.Wilson. Generalisation in the XCS classifier system. In proceedings of the *Third Genetic Programming Conference* (J.Koza ed.), pp 665-674, Morgan Kaufmann, 1998.

[23] S. Schulenburg, P. Ross, J.G. Marín-Blázquez and E. Hart. A hyper-heuristic approach to single and multiple step environments in bin-packing problems. To appear in *Proceedings of the Fifth International Workshop on Learning Classifier Systems 2002 (IWLCS-02.*

[24] http://bwl.tu-darmstadt.de/bwl3/forsch/projekte/binpp

[25] P.Cowling, G.Kendall, E.Soubeiga. A Hyperheuristic Approach to Scheduling a Sales Summit. In LNCS 2079, Practice and Theory of Automated Timetabling III : Third International Conference, PATAT 2000, Konstanz, Germany, August 2000, selected papers (eds Burke E.K. and Erben W), Springer-Verlag, pp 176-190

[26] P.Cowling, G.Kendall and E.Soubeiga. A Parameter-Free Hyperheuristic for Scheduling a Sales Summit. In *proceedings of 4th Metahuristics International Conference (MIC 2001), Porto Portugal*, 16-20 July 2001, pp 127-131

[27] P.Cowling, G.Kendall and E.Soubeiga. Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In *LNCS 2279, Applications of Evolutionary Computing : Proceedings of Evo Workshops 2002, Kinsale, Ireland*, April 3-4, 2002, (eds : Cagoni S, Gottlieb J, Hart E, Middendorf M, Günther R), pp 1-10, ISSN 0302-9743, ISBN 3-540-43432-1, Springer-Verlag

[28] P.Cowling, G.Kendal and L.Han. An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In proceedings of *Congress on Evolutionary Computation (CEC2002)*, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, May 12-17, 2002, pp 1185-1190, ISBN 0-7803-7282-4

[29] E.K.Burke and J.P.Newall. A new adaptive heuristic framework for examination timetabling problems. Technical Report NOTTCS-TR-2001-5 (submitted to Annals of Operations Research), University of Nottingham, UK, School of Computer Science & IT, 2002

[30] D.E.Joslin and D.P.Clements Squeaky Wheel Optimization, *Journal of Artificial Intelligence Research*, Volume 10, 1999, pp 353-373

[31] B.Selman and H.Kautz, Domain-independent extensions to GSAT: Solving large structured satisfiability problems, *Proc. of the 13th Int'l Joint Conf. on Artificial Intelligence*, 1993, pp 290-295

[32] E.K.Burke and S.Petrovic, Recent Research Directions in Automated Timetabling. *To appear in the European Journal of Operational Research, 2002*

View publication stats