CHOICE FUNCTION AND RANDOM HYPERHEURISTICS

Graham Kendall, Eric Soubeiga[*]

School of Computer Science & IT
University of Nottingham, Nottingham
NG8 1BB, UK; gxk/exs@cs.nott.ac.uk

Peter Cowling

Department of Computing, University of Bradford
Bradford BD7 1DP, UK;
Peter.Cowling@scm.brad.ac.uk

ABSTRACT

A *hyperheuristic* is a high-level heuristic which adaptively controls the combination of several low-level knowledge-poor heuristics so that while using only cheap and easy-to-implement low-level heuristics, we may achieve solution quality approaching that of an expensive knowledge-rich approach. Hyperheuristics have been successfully applied by the authors to three real-world problems of personnel scheduling. In this paper, the low-level behaviour of the choice-function based hyperheuristic is investigated and compared with a range of other heuristics and hyperheuristics. We show that the choice-function hyperheuristic makes an effective and realistic combination of the low-level heuristics at hand. Furthermore the combination of the low-level heuristics is intelligently adapted to both the problem being solved and the region of the search space currently being explored.

Key words: hyperheuristic, hybrid heuristic, choice function, personnel scheduling.

1. INTRODUCTION

Personnel scheduling involves allocating timeslots and possibly locations and other resources to people. This subject has been the topic of much research since the 1970's [2, 12, 3]. Like other combinatorial optimisation problems, the resulting NP-hard problem is usually solved using heuristics which often requires the use of sophisticated metaheuristic methods and problem-specific information to obtain a good solution. For example Dowsland [8] used tabu search combined with strategic oscillation to schedule nurses. Using a variety of sophisticated local search moves, the search is allowed to cross the infeasible regions in the hope of finding a good solution beyond. The result was a robust and effective method capable of producing solutions of similar quality to those of a human expert. The same problem was successfully solved in [1] using a co-evolutionary strategy based on co-operating subpopulations built using knowledge of the problem structure. While domain-knowledge can help boost the search as in [8, 1] it makes the resulting tailor-made metaheuristic less re-usable for other problems. Heuristic and metaheuristic approaches tend to be knowledge-rich, requiring substantial expertise in both the problem domain and appropriate heuristic methods. It is in this context that we proposed a *hyperheuristic* approach in [4] as a heuristic that operates at a higher level of abstraction than current metaheuristic approaches. The hyperheuristic adaptively controls the combination of several low-level knowledge-poor heuristics (e.g. add, delete, swap moves). At each decision point the hyperheuristic must choose which low-level heuristic to apply, without recourse to domain-knowledge. Hence hyperheuristics may be used in cases where little domain-knowledge is available (e.g. when dealing with a new, poorly understood or unusual problem) or when a solution must be produced quickly (e.g. prototyping). A hyperheuristic is a generic and fast-to-implement method (compared to a bespoke metaheuristic), which should produce solutions of acceptable quality, based on a set of easy-to-implement low-level heuristics. In order to apply a hyperheuristic to a given problem, all we need is a set of low-level heuristics and a formal means of evaluating solution quality. Similar approaches of managing several low-level heuristics are proposed in [11] in which to each low-level heuristic is associated a utility function and a preference weight. Different weight-adaptation schemes are empirically compared using two problems (Orc Quest and Logistic Domains). In this paper we focus on a class of hyperheuristics, the choice-function based hyperheuristics, which has been successfully applied to three real-world scheduling problems [4, 5, 6, 7]. We investigate the low-level behaviour of the choice-function based hyperheuristic (which is presented in section 2). Our problem application is that of scheduling project presentations (section 3) [6]. We shall demonstrate that the choice-function based hyperheuristic makes an effective and realistic combination of the low-level heuristics (section 4).

[*]Coresponding author

2. HYPERHEURISTIC TECHNIQUES

We use two types of hyperheuristic. The first one is a random hyperheuristic, RD, which repeatedly chooses one low-level heuristic uniformly at random and applies it until some stopping criterion is met. The chosen low-level heuristic is applied in a descent fashion (i.e until no further improvement is possible). Note that RD is similar to Variable Neighbourhood Search (VNS) [9]. The second type of hyperheuristic, HH, is based on a *Choice-Function* which adaptively ranks the low-level heuristics. In its original version presented in [4], the choice function reflects recent improvements of each low-level heuristic (f_1), recent improvements for consecutive pairs of low-level heuristics (f_2), and the number of CPU seconds elapsed since the heuristic was last called (f_3). Thus we have $f_1(h_j) = \sum_n \alpha^{n-1}(\frac{I_n(h_j)}{T_n(h_j)})$ and $f_2(h_k, h_j) = \sum_n \beta^{n-1}(\frac{I_n(h_k,h_j)}{T_n(h_k,h_j)})$ where $I_n(h_j)$/$I_n(h_k, h_j)$ (respectively $T_n(h_j)$/$T_n(h_k, h_j)$) is the change in the evaluation function (respectively the number of CPU seconds used) the n^{th} last time heuristic h_j was called/ immediately after heuristic h_k. Both α and β are parameters in interval $[0, 1]$, which reflects the greater importance attached to recent performance. f_1 and f_2 are there for the purpose of intensification[1]. f_3 provides an element of diversification, by favouring those low-level heuristics that have not been called recently. Then we have $f_3(h_j) = \tau(h_j)$ where $\tau(h_j)$ is the number of CPU seconds elapsed since h_j was last called. If the low-level heuristic just called is h_k then for any low-level heuristic h_j, the choice function f of h_j is defined as $f(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j)$. Parameter values for α, β and δ are changed adaptively using a procedure described in [5], similar to the idea of reinforcement learning [10]. Given a problem P to be solved, an evaluation function E for solution quality and a set $H = \{h_1, h_2, ..., h_n\}$ of low-level heuristics, the choice-function hyperheuristic works as follows:

Do

- *Select the low-level heuristic that maximises choice-function f and apply it.*
- *Update choice function f's parameters using the adaptive procedure*

Until Stopping condition is met.

We next present our case study problem.

3. SCHEDULING OF PROJECT PRESENTATIONS

The problem occurs every year at the School of Computer Science and Information Technology of the University of Nottingham which has to schedule final year undergraduate students' project presentations during a period of up to 4 weeks. As part of their degree requirements, the students must give a 15-minute presentation of their project. A project is defined by its topic, the student who works on it, and the lecturer who supervises the student's work. Each project must be presented by the corresponding student to a jury composed of a Chair or First Marker, a Second Marker and an Observer. Ideally, the project's supervisor should be a member of the jury (as Chair or Observer) but this is often not the case in practice. Individual presentations are grouped into hourly sessions[2] to which a room is allocated. The problem is to determine all (student, 1st marker, 2nd marker, observer, room, timeslot) tuples. Let $\mathbf{I}, \mathbf{S}, \mathbf{Q}$ and $\mathbf{R}$ be the sets of students, staff members, sessions and rooms respectively. Our main decision variables are denoted by x_{ijklqr} ($i \in \mathbf{I}, j, k, l \in \mathbf{S}, j \neq k, j \neq l, k \neq l, q \in \mathbf{Q}, r \in \mathbf{R}$), where x_{ijklqr} is 1 if presentation of student i is assigned to 1st marker j, 2nd marker k, observer l and allocated to session q in room r; otherwise x_{ijklqr} is 0. The problem constraints are (1) each presentation must be scheduled once; (2) at most four presentations for each session and room; (3) no lecturer can be scheduled to 2 different rooms within the same session. The problem formulation is given in [6] as a minimisation problem with E, the overall objective function made up of 4 weighted goals[3]. We use the same low-level heuristics as in [6]. They are all simple and based around moving, replacing or swapping an object. (h_1): Replace a random lecturer j_1, with another

	csit0	csit1	csit2
ch	-908.5	-2557.6	-946.6
RD_{am}	-1274.55*	-2884.49	-1668.76
RD_{oi}	-1303.38*	-2892.81	-1675.48
HH_{am}	**-1444.99***	-2960.3	-1650.67
HH_{oi}	-1316.56*	**-2963.37**	**-1720.23**
$RD1_{am}$	-1406.64*	-2892.02	**-1724.15**
$RD1_{oi}$	-1398.74*	-2887.90	-1720.29
ml	-90.1	-	-
$HH_{am}(ml)$	-644.43	-	-
s_1	71.1	4304.5	6051.5
$HH_{am}(s_1)$	-1326.37	-323.23	717.96
s_2	516.8	98.9	986.4
$HH_{am}(s_2)$	-991.96	-1342.3	-114.5

Table 1: Initial solution is ch. csit0 results marked with * are taken from [6] which used a 1Ghz PC with 128Mb RAM.

[1]The idea behind f_1 and f_2 is analogous to exponential smoothing [13].

[2]A session contains up to four 15-minute presentations.

[3](A) Fair distribution of the total number of presentations per staff member; (B) Fair distribution of the total number of sessions per staff member; (C) Fair distribution of the number of 'early' (before 10:00am) and 'late' (after 4:00pm) sessions per staff member; (D) matching of staff research interest to project topic, and where possible involvement of supervisors in corresponding presentations

random one j_2 in a random session q during which j_1 is scheduled for presentations; (h_2):Same as h_1 but j_1 has the largest number of scheduled sessions; (h_3):Same as h_2 but q is the one where j_1 has the smallest number of presentations; (h_4):Move a random presentation i from its current session-room into another random session-room q-r; (h_5): Same as h_4 but presentation i is that for which the sum of presentations involving all three lecturers (1st marker, 2nd marker, observer) is smallest of all sessions; (h_6): Same as h_5 but session q is one where at least one of the lecturers (1st marker, 2nd marker, observer) is already scheduled for presentations; (h_7): Swap 2nd marker of one presentation with observer of another; (h_8): Swap 1st marker of one presentation with 2nd marker of another[4].

4. EXPERIMENTS

The first set of experiments[5] aimed at making a direct comparison between the random hyperheuristic (RD), and the choice-function based hyperheuristic (HH). For both algorithms we distinguished the case where all moves (am) are accepted and the case where only improving moves (oi) are accepted. Results (averaged over 10 runs) are given in the upper part of Table 1 for three instances $csit0$, $csit1$ and $csit2$. The first instance is taken from [6] and has the following problem characteristics $|I| = 151$, $|Q| = 80$, $|R| = 2$, and $|S| = 26$. Instance $csit1$ have $|I| = 240$, $|Q| = 36$, $|R| = 2$, and $|S| = 24$. csit2 is the same as csit1 except in $|S| = 22$. Thus $csit2$ is more difficult (tighter constraints) than $csit1$. Note that $csit0$ is much easier (slack constraints) than both $csit1$ and $csit2$ as from the from the former to the latter instances there is a 58% increase in $|I|$ and a 45% decrease in $|Q|$, hence many more projects to schedule in fewer timeslots. In increasing order of difficulty we have $csit0, csit1, csit2$. Note that all instances have thousands of constraints and several millions of variables in our integer programming model. Both RD and HH start with a solution produced by a constructive heuristic, ch, used in [6]. The stopping condition was 600 seconds CPU. The results (averaged over 10 runs) correspond to the best value of E found during the search of each algorithm. We see that both algorithms produced results much better than ch. Also HH gave better results than RD. We note that the gap in terms of E between HH and RD is greatest with $csit0$ and smallest with $csit2$. It seems that HH outperforms RD though the difference appears to decrease as the difficulty of the instances to solve increases. Furthermore it was observed in [6] that finding a better solution becomes very challenging as the search goes on. This suggests that there is an advantage in using HH over RD. In [6] HH also achieved results superior to those of RD when both methods started from a very poor-quality solution constructed manually for $csit0$ (called ml in Table 1). The hyperheuristic results were however inferior to those of ch. We decided to run HH from two initial solutions of very poor quality. Both initial solutions (s_1 and s_2) are obtained randomly. In the lower part of Table 1 the objective is given for HH after 2 hours of CPU time in order to see if HH will get any closer to ch. The results (averaged over 3 runs) suggest that the area of the search space the initial solutions are at are so poor that it is difficult to quickly move to a good area. It should be noted however that HH made a huge improvement on the initial solutions and is able to catch up and even overtake ch on instance csit0. Therefore it is still possible for HH to reach good areas even if they are far away from the part of the space currently under exploration.

The second set of experiments aimed at investigating the low-level behaviour of the choice-function based hyperheuristic. In Table 2 we give the proportion of call, by HH_{am}, of each low-level heuristic during the first 100 heuristic calls and during the last 100 heuristic calls to the best solution. We also rank the low-level heuristics according to their overall proportion of call so that the top (bottom) heuristic is the one that has been called most (least) often. Results are obtained after 30 minutes CPU of run in order to allow for a realistic sampling. From the proportion of calls during the 1st 100 calls it is clear that in the early stage of the search calls are spread fairly evenly over the low-level heuristics, as the hyperheuristic has not 'learned' which ones are best. Because HH_{am} seems to be continually improving on the search, the last 100 heuristic calls to the best solution correspond to the last 100 heuristic calls of the search. It is interesting to note that not all low-level heuristics need be called at this later stage. Thus only low-level heuristics h_2, h_3, h_5 and h_6 are needed for instances $csit1$ and $csit2$ whereas h_3 alone suffices for problem instance $csit0$. Interestingly enough this difference in the choice of the low-level heuristics reflects the difference between $csit1$ and $csit2$ on the one hand and $csit0$ on the other. It seems that the choice-function based hyperheuristic shows different behaviours for different problems. This suggests that the hyperheuristic is capable of learning about the interplay existing between the low-level heuristics dependent on both the problem being solved and the part of the search space currently being explored. From the overall proportion of calls we see that overall (across the 3 instances), heuristics h_2, h_3 and h_6 figure among the top 3 heuristics whereas heuristic h_1 is at the bottom. This can be regarded as a feature common to the 3 problem instances. As noted in [6] it seems that h_3 and h_6 which are the most sophisticated version of their category ('replace' type h_1,

[4]In both h_7 and h_8 supervisors may not be removed.

[5]Unless otherwise stated, all algorithms reported in this paper were coded in Micosoft Visual C++ version 6 and all experiments were run on a PC Pentium III 1500MHz with 228MB RAM running under Microsoft Windows 2000 version 5.

h_2, h_3 and 'move' type h_4, h_5, h_6) deserve to be called more often than the others. There was no plateau landscape during the hyperheuristic search on instances csit1 and csit2. For csit0 however a plateau of solutions evaluated at -1390.6 was identified. The 100 heuristic calls covering the plateau landscape were distributed as 0, 28, 35, 0, 5, 18, 2, 12 for heuristics h_1, h_2, .., h_7 and h_8 respectively. Comparing this to csit0 results in Table 2 we see a totally different low-level behaviour, which helped the hyperheuristic escape from the plateau by first accepting worse solutions (up to -1292.6) in order to reach out for good ones, ending up at -1414.6 (at the 100^{th} heuristic call).

Using the results in Table 2 we implemented an 'intelligent' random hyperheuristic, $RD1$, based on RD. Instead of selecting each low-level heuristic uniformly at random (i.e. equal probability of choice) $RD1$ chooses each low-level heuristic with a certain probability which corresponds to its overall proportion of call by the choice function hyperheuristic HH[6]. The aim of the experiment was to see if the choice-function based hyperheuristic power is in choosing only the best proportion of calls, or whether the choice function gives additional power by providing a better-than-random ordering. $RD1$ 10-run average results can be found in the upper part of Table 1. Both RD and $RD1$ give similar results on instance $csit1$. On instances $csit2$ and $csit0$ however, $RD1$, which uses the choice function's combination of the low-level heuristics outperforms RD, which simply chooses the low-level heuristics with equal probability. It appears that the choice function hyperheuristic makes an effective and realistic combination of the low-level heuristics at hand and as such is better than a simple random heuristic combination. While HH and $RD1$ gave comparable results on instance $csit2$, HH outperformed $RD1$ on $csit1$ and $csit0$. It seems that an approach which maintains an adaptive combination of the low-level heuristics (HH) may appear to be more robust than one which keeps the same combination of the low-level heuristics ($RD1$), due to the ability of the former to adapt to changes in the search landscape (valleys, plateaux, ...). Therefore the similarity of results between HH and $RD1$ on instance $csit2$ suggests that the area of the landscape we are at is somewhat smooth and so the current heuristic combination used in $RD1$ is good enough to cope with that. To further confirm this, and to see if HH is nothing more than just a good random hyperheuristic (like $RD1$) we ran both $RD1$ and HH using initial solutions s_1 and s_2. This has the effect of starting the search from a rather different area (different to that of ch). $RD1$ still uses the same probabilities, which were obtained by HH with ch as initial solution. Results (averaged over 10 runs) are given in Table 3. When starting from s_1, HH beats $RD1$ by 150 (in difference) on csit1 and by 934.12 on csit2. Both algorithms have comparable results on csit0 (small difference of only 18.51). When starting from s_2, $RD1$ and HH have similar results (small difference of only 12.86) on csit1. $RD1$ beats HH by 66.94 on csit2 and by 116.82 on csit0. This suggests that adaptively changing the probability of choice of the low-level heuristic during the search allows us to deal robustly with different problem instances and starting solutions. In other words in some cases, just having a 'magic' combination of the low-level heuristic is not enough ($RD1$). We must maintain an adaptive control on the way we combine the low-level heuristics in order to carry out an effective search. The choice function hyperheuristic appears capable of achieving this intelligently. This also means that the way the hyperheuristic works is quite different from a random search, however effective that random search is. It is interesting to note that the superiority of HH over $RD1$ is greater on initial solution s_1 which is worse than s_2. This suggests a certain strong robustness of HH which is capable of finding good solutions much more quickly than $RD1$. We would like to emphasize the fact that the choice-function hyperheuristic HH presented here is a 'standard' approach which worked well for two other real-world problem of scheduling [4, 5, 7]. Indeed given a problem P to be solved, all that is needed is a solution evaluation function E and a set H of low-level heuristics to be plugged into the hyperheuristic black box. The way the hyperheuristic works is independent of the nature of the low-level heuristics and hence of the problem to be solved. The objective function value and CPU time are the only things passed from the low-level heuristics to the hyperheuristic. The sort of solutions produced by HH appeared to be practical. As a result, the choice-function hyperheuristic solution has been implemented by the school for this academic year 2001-2002. The school's timetabling officer described the results as 'excellent'.

	csit0	csit1	csit2
h_1	2/0, 7/0.006	4/0, 8/0.009	2/0, 8/0.005
h_2	25/0, 2/0.134	16/4, 3/0.118	31/6, 2/0.129
h_3	43/100, 1/0.691	16/76, 1/0.552	32/88, 1/0.672
h_4	5/0, 6/0.001	5/0, 7/0.013	10/0, 5/0.016
h_5	8/0, 5/0.041	7/2, 4/0.078	6/1, 4/0.038
h_6	10/0, 3/0.077	9/18, 2/0.126	10/5, 3/0.121
h_7	3/0, 6/0.001	28/0, 6/0.046	3/0, 7/0.009
h_8	4/0, 4/0.049	15/0, 5/0.058	6/0, 6/0.010
E	-1462.6	-2946.6	-1730

Table 2: heuristic calls by HH_{am}. Format: # calls during 1st 100 calls/last 100 calls to best solution, overall rank/overall proportion of call

[6]For example when applying $RD1$ to instance $csit1$, heuristics h_1, h_2, h_3, h_4, h_5, h_6, h_7 and h_8 are chosen with probability 0.009, 0.118, 0.552, 0.013, 0.078, 0.126, 0.046 and 0.058 respectively.

	csit0	csit1	csit2
$RD1_{am}(s_1)$	-500.93	1665.77	4164.21
$RD1_{oi}(s_1)$	-481.84	1545.55	4273.03
$HH_{am}(s_1)$	-394.77	1450.55	3230.09
$HH_{oi}(s_1)$	-482.42	1395.91	3350.64
$RD1_{am}(s_2)$	-463.01	-1059.73	27.7
$RD1_{oi}(s_2)$	-427.38	-1028.04	-3.27
$HH_{am}(s_2)$	-346.19	-976.95	63.67
$HH_{oi}(s_2)$	-345.60	-1040.09	76.06

Table 3: Comparison between HH and $RD1$ with different initial solutions

5. CONCLUSIONS

We have investigated the low-level behaviour of a choice-function hyperheuristic using an 'intelligent' tailor-made random hyperheuristic. It appears that the choice-function hyperheuristic not only makes an effective and realistic combination of the low-level heuristics at hand but is also capable of intelligently adapting this heuristic combination to both the problem being solved and the region of the search space currently under exploration. While much of the power of the hyperheuristic appears to come from selecting appropriate probabilities for calling low-level heuristics, the power of the method is that these probabilities are adaptively tailored to the solution space and low-level heuristics.

The choice-function hyperheuristic had been successfully applied to two other real-world problems of scheduling [4, 5, 7]. The way it chooses the low-level heuristic is problem-independent. This results in a method which is easily reusable for other problems. Hence substantial savings in solution development time are made possible as in [6]. In this paper we have added evidence that hyperheuristics are capable of learning about the dynamics existing between the low-level heuristics and the solution space and consequently can make the best use of these low-level heuristics. This makes hyperheuristics a useful and effective class of methods, especially when very little domain-knowledge is available. Hyperheuristics appear to be worthy of further investigation.

6. ACKNOWLEDGEMENTS

We would like to thank Dr Helen Ashman for providing us with the data.

7. REFERENCES

[1] U. Aickelin and K. A. Dowsland. Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, 3:139–153, 2000.

[2] K. Baker. Workforce allocation in cyclical scheduling problems: A survey. *Operational Research Quarterly*, 27(1):155–167, 1976.

[3] D. J. Bradley and J. B. Martin. Continuous personnel scheduling algorithms: a literature review. *Journal Of The Society For Health Systems*, 2(2):8–23, 1990.

[4] P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. Proceedings of the 3rd International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000, Springer Lecture Notes in Computer Science, 176-190, 2001.

[5] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyperheuristic for scheduling a sales summit. Proceedings of the 4th Metaheuristic International Conference, MIC 2001, 127-131.

[6] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. Proceedings of the 2nd European Conference on EVOlutionary computation for Combinatorial OPtimisation, EvoCop 2002, Springer Lecture Notes in Computer Science, 1-10, 2002.

[7] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: a robust optimisation method for nurse scheduling. 7^{th} Intl Conf on Parallel Problem Solving from Nature, 2002. To appear in Springer LNCS.

[8] K. A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.

[9] P. Hansen and N. Mladenović. Variable Neighbourhood Search: Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[10] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[11] A. Nareyek. An empirical analysis of weight-adaptation strategies for neighbourhoods of heuristics. Proceedings of the 4th Metaheuristic International Conference, MIC 2001, 211-215.

[12] J. M. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287, July 1982.

[13] S. C. Wheelwright and S. Makridakis. *Forecasting methods for management*. John Wiley & Sons Inc, 1973.