Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation

Peter Cowling, Graham Kendall, Eric Soubeiga*

Automated Scheduling, optimisAtion and Planning (ASAP) Research Group, School of Computer Science and Information Technology (CSIT), The University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, England, United Kingdom

Abstract. The term *hyperheuristic* was introduced by the authors as a high-level heuristic that adaptively controls several low-level knowledge-poor heuristics so that while using only cheap, easy-to-implement low-level heuristics, we may achieve solution quality approaching that of an expensive knowledge-rich approach. For certain classes of problems, this allows us to rapidly produce effective solutions, in a fraction of the time needed for other approaches, and using a level of expertise common among non-academic IT professionals. Hyperheuristics have been successfully applied by the authors to a real-world problem of personnel scheduling. In this paper, the authors report another successful application of hyperheuristics to a rather different real-world problem of personnel scheduling occuring at a UK academic institution. Not only did the hyperheuristics produce results of a quality much superior to that of a manual solution but also these results were produced within a period of only three weeks due to the savings resulting from using the existing hyperheuristic software framework.
Key words: Hyperheuristic, Heuristic, Rapid prototyping, Personnel Scheduling.

1 Introduction

Personnel scheduling involves the allocation of timeslots and possible locations to people. This subject has been the focus of research since the 1970's [2, 11, 4]. As for other combinatorial optimisation problems, the resulting NP-hard problem is often solved using heuristic techniques [10, 1, 8, 7, 3], many of which use sophisticated metaheuristic methods and problem-specific information to arrive at a good solution. For instance Levine [9] used a hybrid genetic algorithm to solve an airline crew scheduling problem. To obtain better results, the genetic algorithm is hybridised with a local search heuristic which tries to repair infeasibilities present in the rows of the constraint matrix. Experiments also compared the hybrid genetic algorithm with branch-and-cut and branch-and-bound algorithms and these latter algorithms both outperformed the hybrid genetic algorithm. Dowsland [8] used tabu search combined with strategic oscillation to schedule

* Corresponding author

nurses. Using a variety of sophisticated local search neighbourhoods, the search is allowed to make some moves into infeasible regions in the hope to quickly reach a good solution beyond. The result is a robust and effective method capable of producing solutions of similar quality to those of a human expert. The same problem was solved in [1] using a co-evolutionary strategy based on co-operating subpopulations and problem-specific considerations.

However, as noted in [5, 6] heuristic solution methods are often tailored specifically for the problem they are being applied to, so that it is unlikely that they may be successfully applied to a different problem. Heuristic and metaheuristic approaches tend to be knowledge-rich, requiring substantial expertise in both the problem domain and appropriate heuristic techniques [1]. It is in this context that we proposed a *hyperheuristic* approach [5] as a heuristic that operates at a higher level of abstraction than current metaheuristic approaches. The hyperheuristic manages a set of simple, knowledge-poor, low-level heuristics (for example swap, add and drop moves). At any given decision point the hyperheuristic must choose which low-level heuristic to apply, without access to domain-knowledge. Hence we may use hyperheuristics in cases where little domain-knowledge is available (for instance when dealing with a new, poorly understood or unusual problem) or when a solution must be produced quickly (for example for prototyping). A hyperheuristic could be regarded as an "off-the-peg" method as opposed to a "made-to-measure" metaheuristic. A hyperheuristic is therefore a generic and fast method, which should produce solutions of acceptable quality, based on a set of easy-to-implement low-level heuristics. In order for a hyperheuristic to be applicable to a given problem, all that is needed is a set of low-level heuristics and one or more measures for evaluating solution quality. In [6] we described various ways of choosing the low-level heuristic to apply at each decision point and reported successful applications to a real-world problem of scheduling a sales summit. In this paper, we use our hyperheuristic techniques to solve another real-world problem of personnel scheduling occuring at a UK university. We aim to demonstrate that hyperheuristics are not only readily applicable to a wide range of problems of scheduling and other combinatorial optimisation problems, but also are capable of generating good quality solutions given very little development time.

In the remainder of the paper, sections 2, 3, 4 and 5 are devoted respectively to the application problem (Project Presentation Scheduling), the hyperheuristic approaches that we have used, an experimental study and conclusions.

2 The Project Presentation Scheduling Problem

Every academic year the School of Computer Science and Information Technology of the University of Nottingham is faced with the problem of scheduling final year BSc students' project presentations during a period of up to 4 weeks. As part of their course requirements, final year BSc students have to give a 15-minute presentation of their project. Each student works on a chosen project topic and is assigned a member of academic staff to supervise the

project. Project presentations are then organised and each student must present his/her project before a panel of three members of academic staff who will mark the student's presentation: The Chair or First Marker, the Second Marker and the Observer. Ideally, the project's supervisor should be involved in the presentation (as Chair or Observer) but this is rarely the case in practice. Once every student has been assigned to a supervisor for his/her project, the problem is to schedule all individual presentations, that is, determine a first marker, a second marker and an observer for each individual presentation, and allocate both a room and a timeslot to the resulting quadruple (student, 1st marker, 2nd marker, observer). The presentations are organised in sessions, each containing up to six presentations. Typically the same markers and observers will see all of the presentations in a particular session. So the problem can be seen as that of determining (student, 1st marker, 2nd marker, observer, room, timeslot) tuples, that respect the following constraints: (1) Each presentation must be scheduled exactly once; (2) No more than six presentations for each room and for each session; (3) No member of staff (whether as 1st marker or as 2nd marker or as observer) can be scheduled to 2 different rooms within the same session. In addition presentations can only be scheduled in a given session when both the academic members of staff and the room assigned to those presentations are available during that session. There are four objectives to be achieved: (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 "bad" sessions per staff member, i.e. sessions at bad times (before 10:00 am, after 4:00 pm); (D) Optimise the match between staff research interest and project themes, and try to ensure that a supervisor attends presentations for projects which they supervise. To formulate the problem, we denote by $\mathbf{I}$ the set of students, $\mathbf{S}$ the set of academic staff members, $\mathbf{Q}$ the set of sessions and $\mathbf{R}$ the set of seminar rooms. Our 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 seminar room r, otherwise x_{ijklqr} is 0; and y_{jqr} $(j \in \mathbf{S}, q \in \mathbf{Q}, r \in \mathbf{R})$ where y_{jqr} is 1 if staff j is in room r during session q, otherwise y_{jqr} is 0. We may then formulate the problem as follows:

Minimise $E(x) = 0.5A + B + 0.3C - D$

s.t.

$$\sum_{j,k,l \in \mathbf{S}} \sum_{q \in \mathbf{Q}} \sum_{r \in \mathbf{R}} x_{ijklqr} = 1, \quad (i \in \mathbf{I}) \tag{1}$$

$$\sum_{i \in \mathbf{I}} \sum_{j,k,l \in \mathbf{S}} x_{ijklqr} \leq 6, \quad (q \in \mathbf{Q}, r \in \mathbf{R}) \tag{2}$$

$$\sum_{r \in \mathbf{R}} y_{jqr} \leq 1, \quad (j \in \mathbf{S}, q \in \mathbf{Q}) \tag{3}$$

$$\sum_{i \in \mathbf{I}} \sum_{k,l \in \mathbf{S}} (x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) \leq M y_{jqr}, \quad (j \in \mathbf{S}, q \in \mathbf{Q}, r \in \mathbf{R}) \tag{4}$$

$$x_{ijklqr}, y_{jqr} \in \{0,1\}, \quad i \in \mathbf{I}, j,k,l \in \mathbf{S}, j \neq k \neq l, q \in \mathbf{Q}, r \in \mathbf{R} \qquad (5)$$

where $A = \sum_{j \in \mathbf{S}} \left(\sum_{q \in \mathbf{Q}} \sum_{r \in \mathbf{R}} \sum_{i \in \mathbf{I}} \sum_{k,l \in \mathbf{S}} (x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) - K \right)^2$, $B = \sum_{j \in \mathbf{S}} \left(\sum_{q \in \mathbf{Q}} \sum_{r \in \mathbf{R}} y_{jqr} - K_1 \right)^2$, $C = \sum_{j \in \mathbf{S}} \left(\sum_{q \in \mathbf{Q_{bad}}} \sum_{r \in \mathbf{R}} y_{jqr} - K_2 \right)^2$, $D = \sum_{j \in \mathbf{S}} \left(\sum_{q \in \mathbf{Q}} \sum_{r \in \mathbf{R}} \sum_{i \in \mathbf{I}} \sum_{k,l \in \mathbf{S}} (p_{ij} + 10Sup_{ij})(x_{ijklqr} + x_{ikjlqr} + x_{ikljqr}) \right)$.

Equations (1), (2), (3) express constraints (1), (2), (3) respectively. Equation (4) links variables x_{ijklqr} with y_{jqr}, where M is a large number. $K = \frac{3|I|}{|S|}$, $K_1 = \frac{6P_1}{|S|}$ and $K_2 = \frac{6P_2}{|S|}$ where K, (K_1/K_2) is the average number of presentations (sessions/"bad" sessions) per member of staff, with P_1 (P_2) the total number of (bad) sessions used in the solution and Q_{bad} a subset of Q containing early sessions (before 10:00 am) and late sessions (after 4:00pm). In objective D, p_{ij} is an integer value associated with the level of matching between the topic of presentation i and the research interest of staff member j if he/she is involved in presentation i. The higher p_{ij}, the better the matching. Sup_{ij} is an indicator of whether staff member j is the supervisor of presentation i ($Sup_{ij} = 1$) or not ($Sup_{ij} = 0$). The different coefficients in the objective function were set so as to reflect the relative importance of each objective. The problem admits a feasible solution if there is enough room-time to allocate each presentation to, hence if $6|R||Q| > |I|$. In this instance of the problem, $|R| = 2$, $|Q| = 80$, $|I| = 151$, $|S| = 26$ and therefore the problem admits a feasible solution. At present the problem is solved manually. We developed a constructive heuristic that produces an initial solution better than the manual one. The constructive heuristic iteratively chooses a triple of staff members and assigns them to as many as 6 presentations in the first available session and room. Priority is given to non-bad sessions (to optimise objective C), to presentations whose supervisor is among the three staff and whose project topic is most related to the concerned staff research interest (to optimise objective D) , and the staff members are chosen on a cyclic basis (to optimise objectives A and B). The solution of the constructive heuristic is used as starting solution for all our algorithms presented in this paper. In the next section we present the different hyperheuristics used for this work.

3 Hyperheuristic Solution Techniques

We used two types of hyperheuristics, simple ones and choice function-based ones as described in [6]. The first type of hyperheuristic comprises simple multiple-neighbourhood search techniques which choose the low-level heuristics cyclically or at random. We used the following simple hyperheuristic algorithms. *SimpleRandom*: This algorithm repeatedly chooses one low-level heuristic at random and applies it once, until some stopping criterion is met. *RandomDescent*: This algorithm repeatedly chooses one low-level heuristic at random and applies it until no further improvement is possible, until some stopping criterion is met.

RandomPerm: This algorithm repeatedly chooses a random sequence of all the low-level heuristics and applies each low-level heuristic once in the sequence order until some stopping criterion is met. It cycles from the last low-level heuristic in the sequence order back to the first one. *RandomPermDescent*: This algorithm does the same thing as *RandomPerm* but each low-level heuristic is applied in a steepest descent fashion.

The second type of hyperheuristic is based on a *Choice-Function* which provides guidance regarding which low-level heuristic to choose. The choice function adaptively ranks the low-level heuristics. In its original version presented in [5], the choice function is determined based upon information regarding recent improvement of each low-level heuristic (first order improvement) denoted by f_1, recent improvement for consecutive pairs of heuristics (second order improvement) denoted by f_2 and the amount of time elapsed since the heuristic was last called denoted by f_3. Thus we have $f_1(N_j) = \sum_n \alpha^{n-1}(\frac{I_n(N_j)}{T_n(N_j)})$ and $f_2(N_j, N_k) = \sum_n \beta^{n-1}(\frac{I_n(N_j,N_k)}{T_n(N_j,N_k)})$ where $I_n(N_j)/I_n(N_j, N_k)$ (respectively $T_n(N_j)/T_n(N_j, N_k)$) is the change in the objective function (respectively the number of CPU seconds) the n^{th} last time heuristic N_j was called/called immediately after heuristic N_k. Both α and β are parameters between 0 and 1, which reflects the greater importance attached to recent performance. f_1 and f_2 aim at intensifying the search. The idea behind the expressions of f_1 and f_2 is analogous to the exponential smoothing forecast of their performance [12]. f_3 provides an element of diversification, by favouring those low-level heuristics that have not been called recently. Then we have $f_3(N_j) = \tau(N_j)$ where $\tau(N_j)$ is the number of CPU seconds which have elapsed since low-level heuristic N_j was last called. If the low-level heuristic just called is N_j then for any low-level heuristic N_k, the choice function f of N_k is defined as

$$f(N_k) = \alpha f_1(N_k) + \beta f_2(N_j, N_k) + \delta f_3(N_k) \tag{6}$$

In the above expression, the choice function attempts to predict the overall performance of each low-level heuristic. In [6] we presented a choice function which separately predicts the performance of each low-level heuristic with respect to each criterion of the objective function instead (i.e. A, B, C and D). The choice function f is then decomposed into

$$f(N_k) = \sum_{l \in \mathbf{L}} f_l(N_k) = \sum_{l \in \mathbf{L}} \left[\alpha_l f_{1l}(N_k) + \beta_l f_{2l}(N_j, N_k) + \frac{\delta}{|L|} f_3(N_k) \right] \tag{7}$$

where $\mathbf{L} = \{A, B, C, D\}$ is the set of the objective function criteria, and $f_{1l}(N_k)$ (respectively $f_{2l}(N_j, N_k)$) is obtained by replacing $I_n(N_k)$ (respectively $I_n(N_j, N_k)$) with $I_{ln}(N_k)$ (respectively $I_{ln}(N_j, N_k)$) in the expression of $f_1(N_j)$ (respectively $f_2(N_j, N_k)$) above. $I_{ln}(N_k)$ (respectively $I_{ln}(N_j, N_k)$) is the first (respectively second) order improvement with respect to criterion $l \in \mathbf{L}$. We will consider both variants of our choice function in the experiments. In [5],

parameters of the original choice function were tuned manually. Instead, we presented a procedure in [6] that monitors the choice function parameters α, β and δ. The procedure can be applied to both choice function expressions (6) and (7). The procedure adaptively adjusts the values of the different parameters so that, in light of the observed historical performance of each low-level heuristic, the weighting assigned to each factor is modified. More precisely it rewards/penalises the choice function intensification (f_1 and f_2) and diversification (f_3) factors by increasing/decreasing the values of the corresponding parameters α, β, δ. Overall the hyperheuristic works as follows

Do

For choice function (7) only: Choose a search criterion l

- Select the low-level heuristic that maximises f (f_l for (7)) and apply it.

- Update choice function f (f_l for (7))'s parameters using the adaptive procedure

Until Stopping condition is met.

We would like to emphasize the fact that the implementation of the hyperheuristic techniques was quite fast. In effect all hyperheuristics presented here are "standard" approaches which worked well for another real-world problem [5, 6]. Indeed all that was needed was a set of low-level heuristics to be input to the hyperheuristic black box. The way the hyperheuristic works is independent of both the nature of the low-level heuristics and the problem to be solved except for the objective function's value and CPU time which are passed from the low-level heuristics to the hyperheuristic. Whilst producing the hyperheuristic framework has taken over 18 months, using this framework took us only the equivalent of 101 hours of work (or two and a half weeks at 40 hours work per week) from understanding the problem to obtaining good hyperheuristic solutions. In the next section we report experiments carried out on all hyperheuristics when applied to the CSIT third year problem of scheduling project presentations.

4 Experiments

All algorithms were coded in Micosoft Visual C++ version 6 and all experiments were run on a PC Pentium III 1000MHz with 128MB RAM running under Microsoft Windows 2000 version 5. In all experiments the stopping condition was 600 seconds of CPU time. All experimental results were averaged over 10 runs. For each algorithm we distinguished the case where all moves (AM) are accepted and the case where only improving moves (OI) are accepted. We used three types of low-level heuristics based on "Replacing" one staff member in a session with a diferent one, "Moving" a presentation from one session to another, and "Swapping" two staff members, one from each presentation. The "Replace" and "Move" type have three variants, and the "Swap" type two variants. Overall we used the following $n = 8$ low-level heuristics.

1. *Replace one staff member in a session (N_1)*: This heuristic chooses a random staff member, say j_1, chooses a random session, say q during which staff j_1

is scheduled for presentations and replaces j_1 with another random staff member, say j_2, in all presentations involving staff j_1 during session q. Staff j_2 must not be involved in any presentations during session q prior to the substitution.

2. *Replace one staff member in a session (N_2) Version 2*: Same as previous heuristic but staff j_1 has the largest number of scheduled sessions.
3. *Replace one staff member in a session (N_3) Version 3*: Same as N_2 but session q is the one where staff j_1 has the smallest number of presentations. Also staff j_2 may be involved in presentations during session q prior to the substitution.
4. *Move a presentation from one session to another (N_4)*: This heuristic chooses a random presentation, removes it from its current session and reschedules it in another random session and a random room.
5. *Move a presentation from one session to another (N_5)Version 2*: Same as previous heuristic but the chosen presentation is that for which the sum of presentations involving all three staff (i.e. 1st marker, 2nd marker, observer) is smallest of all sessions.
6. *Move a presentation from one session to another (N_6)*: Same as N_5 but the new session is one where at least one of the staff members (i.e. 1st marker, 2nd marker, observer) is already scheduled for presentations.
7. *Swap 2nd marker of one presentation with observer of another (N_7)*: This heuristic chooses two random presentations and swaps the 2nd marker of the first presentation with the observer of the second presentation. The swap cannot involve the removal of a supervisor.
8. *Swap 1st marker of one presenation with 2nd marker of another (N_8)*: This heuristic chooses two random presentations and swaps the 1st marker of the first presentation with the 2nd marker of the second presentation. The swap cannot involved the removal of a supervisor.

For each of "Replace" and "Move" types of low-level heuristic the third version generally yields solutions of better quality than the two others. We shall see later on that the choice function hyperheuristic is capable of detecting this behaviour.

In Table 1, we present results for the simple hyperheuristics, the original choice function hyperheuristic (*OriginalHyperheuristic*), which needs manual tuning of its choice function parameters and both choice function (6) and (7) hyperheuristics (*Hyperheuristic6* and *Hyperheuristic7*). The choice of the search criterion l for *Hyperheuristic7* is based on a probability distribution which assigns the probability with which a criterion is chosen depending on the relative weight of that criterion in the objective function [6]. In this case we choose A with probability $p_a = \frac{0.5}{0.5+1+0.3+1}$, B with probability $p_b = \frac{1}{0.5+1+0.3+1}$, C with probability $p_c = \frac{0.3}{0.5+1+0.3+1}$, and D with probability $p_d = \frac{1}{0.5+1+0.3+1}$. Table 1 presents the results for all our algorithms. We show the objective value of both the manual and constructive heuristic solutions.

We see that all algorithms produced results much better than the manual solution and the constructive heuristic solution. We note that among the simple

Algorithm	A	B	C	D	**E**
Manual Solution	455	40	18	-363	-90.1
Constructive Heuristic	1313	51	0	-1616	-908.5
RandomPerm-AM	962.80	66.00	18.10	-1616.80	-1063.97
RandomPerm-OI	790.40	20.75	4.7	-1614.8	-1197.59
RandomPermDescent-AM	624.60	17.40	3.50	-1618.00	-1287.25
RandomPermDescent-OI	631.20	16.10	3.50	-1617.50	-1284.75
SmpleRandom-AM	837.40	76.50	17.30	-1621.50	-1121.11
SimpleRandom-OI	796.00	22.10	2.70	-1614.10	-1193.19
RandomDescent-AM	645.40	20.10	3.50	-1618.40	-1274.55
RandomDescent-OI	583.00	18.30	4.40	-1614.50	-1303.38
Hyperheuristic6-AM	344.40	14.60	17.70	-1637.10	**-1444.99**
Hyperheuristic6-OI	592.40	15.20	4.80	-1629.40	-1316.56
Hyperheuristic7-AM	671.80	20.60	4.90	-1628.40	-1270.43
Hyperheuristic7-OI	671.40	18.70	2.60	-1620.70	-1265.52
OriginalHyperheuristic-AM	545.60	17.00	3.20	-1617.10	-1326.34
OriginalHyperheuristic-OI	665.20	19.00	5.70	-1621.00	-1267.69

Table 1. All algorithms start from constructive heuristic solution

hyperheuristics, the best results are from *RandomPermDescent* and *RandomDescent*, which apply the low-level heuristics in a descent fashion. This was also the case in [5, 6] when these algorithms were applied to a different scheduling problem. While *Hyperheuristic7* gave results comparable to those of the simple hyperheuristics, *Hyperheuristic6* produced results much better than those of the simple hyperheuristics. It should be noted that as the search goes on and as solution quality improves, finding a better solution becomes very challenging. In light of this, it would appear that *Hyperheuristic6* performs a very effective search. We note that the original hyperheuristic whose parameters were manually set to $\alpha = 0.1, \beta = 0.1, \delta = 2.5$ for the AM case and $\alpha = 0.1, \beta = 0.1, \delta = 1.5$ for the OI case, produced good results too, as was the case in [5] when applied to another real-world problem. The fact that *Hyperheuristic6* outperforms the original hyperheuristic confirms the net advantage of having an adaptive procedure for setting parameters as noted in [6]. Beyond the obvious advantage of the reduction in intervention from automatically setting parameters the procedure in some sense learns the interplay between all factors of the choice function and adjusts values of α, β, δ accordingly.

In Table 2, we compare the frequency of call of each low-level heuristic (the number of times that each low-level heuristic has been called) in the case of *Hyperheuristic6* and *Hyperheuristic7*. We can see that these two hyperheuristics do not treat the low-level heuristics in the same way. More precisely, we see that although low-level heuristic N_3 is the most important for both hyperheuristics, low-level heuristics N_1, N_4, N_5, N_7 are given more importance by *Hyperheuristic7* than by *Hyperheuristic6*, while low-level heuristic N_6 is treated the same way

by both hyperheuristics. It seems that the three most important heuristics for *Hyperheuristic6* are N_3, N_2 and N_6 whereas heuristics N_3, N_1 and N_8 are the top three from *Hyperheuristic7*'s perspective. As mentioned earlier, a comparison of the proportion of calls of the low-level heuristics within each of "Replace" and "Move" types shows that the third version for each type is called more often than each of the other two versions of that type. Thus N_3 is called by both choice-function based hyperheuristics more often than N_1 and N_2. Similarly, N_6 is called more often by *Hyperheuristic6* than N_4 and N_5 while *Hyperheuristic7* slightly favours N_5 over N_6. Overall the choice function hyperheuristic appears capable of detecting good low-level heuristics [6].

Algorithm	E	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8
Hyperheuristic12-AM	-1402.4	4	42	76	3	16	20	5	15
Proportion	-	0.02	0.23	0.42	0.02	0.09	0.11	0.03	0.08
Hyperheuristic13-AM	-1314	25	19	35	10	21	20	17	22
Proportion	-	0.15	0.11	0.21	0.06	0.12	0.12	0.10	0.13

Table 2. Comparison between *Hyperheuristic6* and *Hyperheuristic7* of the frequency of calls of the low-level heuristics

The superiority of the choice function-based hyperheuristics was also noticeable when we ran each of our algorithms starting from the manual solution (instead of that produced by the constructive heuristic). Again all algorithms produced results better than the manual solution and *Hyperheuristic6* produced the best results of all. However the results are very poor when compared to the solution obtained by the constructive heuristic. It seems that the search starts from a region of such poor quality, since the manual scheduler was unable to handle the objective of matching research interests, that it is difficult to move to a good area. *OriginalHyperheuristic* produced results of comparable quality to those of the simple hyperheuristics. Whilst *Hyperheuristic6* produces better results than *Hyperheuristic7* for the project presentation scheduling problem, the opposite happened with the sales summit scheduling problem [6]. Effectively in [6], *Hyperheuristic7*, which decomposed the objective value into three criteria, gave better results than *Hyperheuristic6*, which did not. The reason why *Hyperheuristic7* does not outperform *Hyperheuristic6* here is probably due to the fact that it would have to deal with more individual objectives than in [6]. The more individual objectives there are (i.e. the bigger $|L|$), the more parameters α_l, β_l ($l \in \mathbf{L}$) need to be managed. Thus convergence to a good solution for the hyperheuristic search could slow down when in presence of a substantial number of individual objectives in E. It is also worth noting that more time was spent "tuning" model parameters in [6] than has been undertaken for the project presentation problem.

5 Conclusions

We have applied various hyperheuristics to a real-world problem of scheduling project presentations in a UK university. Prior to our intervention the problem was solved manually. Our hyperheuristics produced solutions dramatically better than the manual one when starting from both a constructive heuristic solution and (even) the original manual solution. Comparing the performance of our hyperheuristics for two real-world problems it appears that the choice function hyperheuristic produces the best result of all hyperheuristics. This type of hyperheuristic is based on a choice function which adaptively ranks the low-level heuristics and thus provides effective guidance to the hyperheuristics. In this paper we have added evidence to our claim that hyperheuristic approaches are easy to implement which deliver solutions of acceptable quality, providing a useful tool in rapid prototyping of optimisation systems. Ongoing research will investigate other types of hyperheuristics applied to a wider range of real-world problems.

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. J. E. Beasley and B. Cao. A tree search algorithm for the crew scheduling problem. *European Journal of Operational Research*, 94:517–526, 1996.
4. 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.
5. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In E. Burke and W. Erben, editors, *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling PATAT'2000*, Springer Lecture Notes in Computer Science, 176-190, 2001.
6. 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.
7. B. Dodin, A. A. Elimam, and E. Rolland. Tabu search in audit scheduling. *European Journal of Operational Research*, 106:373–392, 1998.
8. K. A. Dowsland. Nurse scheduling with tabu search and strategic oscillation. *European Journal of Operational Research*, 106:393–407, 1998.
9. D. Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers and operations research*, 23(6):547–558, 1996.
10. G. M. Thompson. A simulated-annealing heuristic for shift scheduling using non-continuously available employees. *Computers and Operations Research*, 23(3):275–288, 1996.
11. J. M. Tien and A. Kamiyama. On manpower scheduling algorithms. *SIAM Review*, 24(3):275–287, July 1982.
12. S. C. Wheelwright and S. Makridakis. *Forecasting methods for management*. John Wiley & Sons Inc, 1973.

This article was processed using the LaTeX macro package with LLNCS style