

A Hyperheuristic Approach to Scheduling a Sales Summit

Peter Cowling, Graham Kendall, and Eric Soubeiga

Automated Scheduling, Optimisation and Planning (ASAP) Research Group,
School of Computer Science and Information Technology, The University of Nottingham,
Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK
{pic|gxx|exs}@cs.nott.ac.uk

Abstract. The concept of a *hyperheuristic* is introduced as an approach that operates at a higher level of abstraction than current metaheuristic approaches. The hyperheuristic manages the choice of **which lower-level heuristic method should be applied at any given time**, depending upon the characteristics of the region of the solution space currently under exploration. We analyse the behaviour of several different hyperheuristic approaches for a real-world **personnel scheduling problem**. Results obtained show the effectiveness of our approach for this problem and suggest wider applicability of hyperheuristic approaches to other problems of scheduling and combinatorial optimisation.

Keywords: hyperheuristics, metaheuristics, heuristics, personnel scheduling, local search, choice function

1 Introduction

Personnel scheduling involves the allocation of personnel to timeslots and possibly locations. The literature uses a variety of terms to describe the same or similar problems. For example, Meissels and Lusternik [11] used the term *employee timetabling* when utilising a constraint satisfaction problem (CSP) model to schedule employees. The term *rostering* can be found in Burke et al. [2], [3] where they employ a hybrid tabu search algorithm to schedule nurses in a Belgian hospital. Dodin et al. [5] use the term *(audit) scheduling* and employ tabu search to schedule audit staff. *Labour scheduling* is used by Easton et al. [7] where they utilise a distributed genetic algorithm technique to determine the number of employees and their work schedules based on predetermined work patterns.

Mason et al. [10] presented an integrated approach using heuristic descent, simulation, and integer programming techniques to schedule staff at the Auckland International Airport, New Zealand. They obtained results which triggered major changes in the attitude of the airport staff who became enthusiastic about the contribution of computer-based decision. Burke et al. [2] used a hybrid tabu search

algorithm to schedule nurses. Tabu search is hybridised with a memetic approach which combines a steepest descent heuristic within a genetic algorithm framework. The resultant search produces a solution which is better than either the memetic algorithm or the tabu search when run in isolation. The hybridised method was run using data supplied by a Belgian hospital and the results were much better than the manual techniques currently being used. Dowsland [6] uses tabu search combined with strategic oscillation to schedule nurses. Dowsland defined chain neighbourhoods as a combination of basic and simple neighbourhoods. Using these neighbourhoods, the search is allowed to make some moves into infeasible regions in the hope that it quickly reaches a good solution beyond the infeasible regions. The result is a robust and effective method which is capable of producing solutions which are of similar quality to those of a human expert.

However, the heuristic and metaheuristic approaches developed for particular personnel scheduling problems are not generally applicable to other problem domains (or even instances of the same or similar problems). Furthermore, heuristic and metaheuristic approaches tend to be knowledge rich, requiring substantial expertise in both the problem domain and appropriate heuristic techniques [1], and are thus expensive to implement. In this paper we propose a **hyperheuristic** approach, which operates at a level of abstraction above that of a metaheuristic. The hyperheuristic will have no domain knowledge, other than that embedded in a range of simple knowledge-poor heuristics. The resulting approach should be cheap and fast to implement, requiring far less expertise in either the problem domain or heuristic methods, and robust enough to effectively handle a wide range of problems and problem instances from a variety of domains.

Other researchers have investigated general-purpose heuristic-based methods for scheduling and optimisation problems. Hart et al. [9] used a genetic algorithm-based approach to select which of several simple heuristics to apply at each step of a real-world problem of chicken catching and transportation. Tsang and Voudouris [12] introduced the idea of having a fast local search (FLS) combined with a guided local search GLS) and applied it to a workforce scheduling problem. FLS is a fast *hill climbing* method which heuristically ignores moves used in the past without any improvement and GLS is a method which diversifies the search to other regions each time a local optimum is reached. Mladenovic and Hansen [8] introduced the idea of variable neighbourhood search (VNS) and applied it to many combinatorial optimisation problems including the travelling salesman problem and the p -median problem. VNS uses a range of higher-level neighbourhood operators for diversification. When a lower-level neighbourhood search operator reaches a local optimum, the search jumps to a random neighbour in the current high-level neighbourhood. When this diversification move proves ineffective, the next higher level neighbourhood is used.

Our hyperheuristic method does not use problem-specific information other than that provided by a range of simple, and hence easy and cheap to implement, knowledge-poor heuristics. A **hyperheuristic** is able to **choose** between **low-level heuristics** without the need to use domain knowledge, using performance indicators which are not specific to the problem each time a low-level heuristic is called, in order to decide **which heuristic to use** when **at a particular point** in the search space.

In order for our hyperheuristic approach to be applicable, we assume that implementing simple local search neighbourhoods and other heuristics (such as

greedy constructive heuristics) for the problem in question is relatively easy. Our experience in real-world personnel and production scheduling problems suggests that this is often the case. Indeed, on first presenting a problem which is solved using manual or simple computer techniques, it is often easier for the manual scheduler to express the problem by discussing the *ways* in which the problem is solved currently, rather than the *constraints* of the problem. Usually these ways of manually solving a scheduling or optimisation problem correspond to simple, easy-to-implement heuristics. We may also implement very easily **simple local search heuristics** based upon **swapping, adding and dropping** events in the schedule. In addition we also require some method of numerically comparing solutions, i.e. **one or more quantitative objective functions**.

Each low-level heuristic communicates with the hyperheuristic using a common problem-independent interface architecture. The hyperheuristic can either choose to call a low-level heuristic in order to see **what would happen if the low-level heuristic were used**, or to **allow the low-level heuristic to change the current solution**. The hyperheuristic may also provide additional information to the low-level heuristic such as the amount of time which is to be allowed. When called, **a low-level heuristic returns a range of parameters** related to **solution quality** or other features (in the case we describe in this paper, **a single objective function** value is returned) and details of the **CPU time** used by the neighbourhood function, which allows us to monitor the **expected improvement** per time unit of **each low-level heuristic**. It is important to note that the **hyperheuristic only knows** whether each objective function is to be **maximised or minimised** (or kept within some range etc.) and has no direct information as to what the objective function represents. We illustrate this idea in Figure 1. All communication between the problem domain and the hyperheuristic is made through a barrier, through which domain knowledge is not allowed to cross.

The rest of the paper is organised as follows. In Section 2 we define a real-world sales summit scheduling problem that we use as a case study to test the effectiveness of our methods. In Section 3 we introduce our hyperheuristic approaches and in Section 4 we present the choice function which many of the approaches require. We then give the results of our experimentation in Section 5. Finally Section 6 presents conclusions and discusses the wider potential for application of hyperheuristic approaches.

2 The Sales Summit Scheduling Problem

The problem we are studying is encountered by a commercial company that organises regular sales summits which bring together two groups of company representatives. The first group, *suppliers*, represent companies who wish to sell some product or service and the second group, *delegates*, represent companies that are potentially interested in purchasing the products and services. Suppliers pay a fee to have a stand at the sales summit and provide a list of the delegates that they would like to meet, where each meeting requested by a supplier is classified as either a *priority* meeting which the supplier feels strongly may yield a sale, or a *nonpriority* meeting about which the supplier feels less strongly. Delegates do not pay a fee and have their travelling and hotel expenses paid by the organiser of the sales summit. In addition to

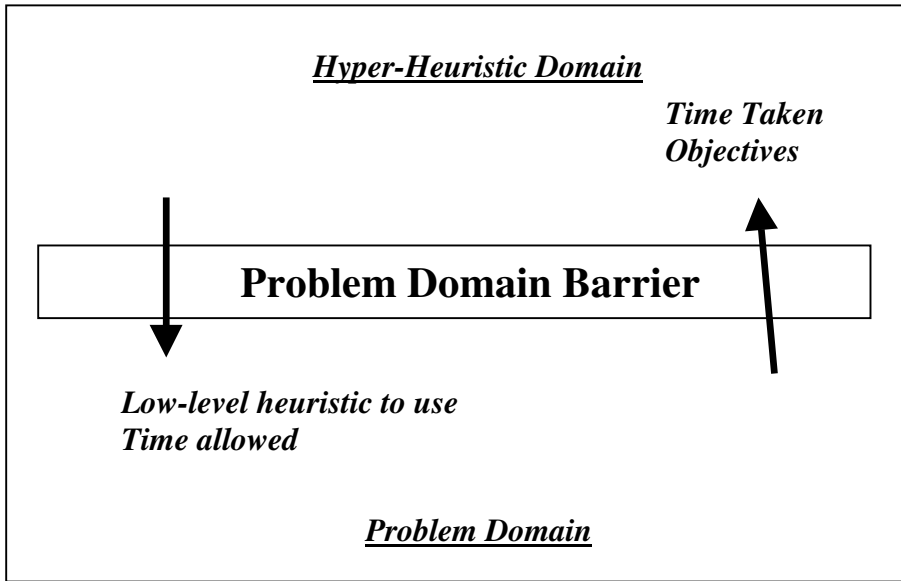


Fig. 1. The hyperheuristic approach and the problem domain barrier

meetings with suppliers, seminars are organised where delegates may meet other delegates. Each delegate supplies a list of the seminars which he will attend in advance of the sales summit, and is guaranteed attendance at all of the seminars requested. There are 24 meeting timeslots available for both seminars and meetings, where each seminar lasts as long as three supplier/delegate meetings. There are 43 suppliers, 99 potential delegates and 12 seminars. The problem is to schedule meetings consisting of (supplier, delegate, timeslot) triples subject to:

1. Each delegate must attend all seminars which they have requested;
2. Each delegate must have at most 12 meetings;
3. No delegate can be scheduled for more than one activity (meeting or seminar) within the same timeslot;
4. No supplier can be scheduled for more than one meeting within the same timeslot;
5. Each supplier should have at least 17 priority meetings;
6. Each supplier should have at least 20 priority and nonpriority meetings in total.

The objective is to minimise the number of delegates who actually attend the sales summit out of the 99 possible delegate attendees, and hence the variable cost of the sales summit, whilst ensuring that suppliers have sufficient delegate meetings. Several other commercial considerations are of secondary importance and are not considered in this paper.

Once delegates have been put into seminar groups, reducing the number of delegate timeslots available, a set of (supplier, delegate, timeslot) meeting triples must be found which minimises the number of attending delegates, whilst keeping all

suppliers and the attending delegates happy. Analysis of the solutions produced using the method currently used by the company, a greedy heuristic still simpler than that which we use below to find an initial solution, suggests that in practice we may relax constraints 5 and 6, so long as no individual supplier has substantially fewer than 17 priority meetings, or 20 meetings in total. We have relaxed these constraints in the model given below.

We denote by S the set of suppliers, D the set of delegates and T the set of timeslots. Let P_{ij} be 1 if (supplier i , delegate j) is a Priority meeting and 0 otherwise ($i \in S, j \in D$). Our decision variables are denoted x_{ijk} ($i \in S, j \in D, k \in T$), where x_{ijk} is 1 if supplier i is to meet delegate j in timeslot k , otherwise x_{ijk} is 0. We can now formulate the problem as follows:

$$\begin{aligned} \text{minimise } E(x) = & \sum_{i \in S} \left(\max \left\{ 0, 17 - \sum_{j \in D} \sum_{k \in T} P_{ij} x_{ijk} \right\} \right)^2 \\ & + 0.05 \sum_{i \in S} \left(\max \left\{ 0, 20 - \sum_{j \in D} \sum_{k \in T} x_{ijk} \right\} \right)^2 + 8 \left(\sum_{j \in D} \min \left\{ 1, \sum_{i \in S} \sum_{k \in T} x_{ijk} \right\} - 72 \right) \end{aligned}$$

subject to

$$\sum_{i \in S} \sum_{k \in T} x_{ijk} \leq 12, \quad j \in D \quad (1)$$

$$\sum_{i \in S} x_{ijk} \leq 1, \quad j \in D, k \in T \quad (2)$$

$$\sum_{j \in D} x_{ijk} \leq 1, \quad i \in S, k \in T \quad (3)$$

$$x_{ijk} \in \{0, 1\}, \quad i \in S, j \in D, k \in T. \quad (4)$$

The evaluation function $E(x) = B(x) + 0.05 C(x) + 8 H(x)$, where

$$B(x) = \sum_{i \in S} \left(\max \left\{ 0, 17 - \sum_{j \in D} \sum_{k \in T} P_{ij} x_{ijk} \right\} \right)^2$$

$$C(x) = \sum_{i \in S} \left(\max \left\{ 0, 20 - \sum_{j \in D} \sum_{k \in T} x_{ijk} \right\} \right)^2$$

$$H(x) = d(x) - 72 \quad \text{with} \quad d(x) = \sum_{j \in D} \min \left\{ 1, \sum_{i \in S} \sum_{k \in T} x_{ijk} \right\}$$

$B(x)$ represents the penalty associated with suppliers who have less than 17 priority meetings, where the quadratic nature of the penalty ensures that any suppliers with substantially less than 17 priority meetings result in a large penalty. $C(x)$ represents the penalty associated with suppliers who have less than 20 meetings in

total, where again the quadratic nature of the penalty ensures that any suppliers with substantially less than 20 meetings are heavily penalised. However, these meetings are much less significant overall than priority meetings (and, for example, we would only want to include delegates with a large number of priority meetings) and $C(x)$ is multiplied by a factor of 0.05 to reflect this. $d(x)$ is the number of delegates who attend the sales summit in the meeting schedule. $H(x)$ represents the penalty associated with the cost of each delegate, and the factor of 8 reflects the fact that a delegate should be included only to satisfy a supplier who would otherwise have significantly less than 17 priority meetings, or eight suppliers who are missing a single meeting. Note that in a solution where each supplier had the required 20 meetings, there would be $43 \times 20 = 860$ meetings. Each delegate can attend at most 12 supplier meetings, so that $\lceil 860/12 \rceil = 72$ delegates are required in this case. We penalise only delegates over 72, to avoid a large constant term in $H(x)$ dominating $B(x)$ and $C(x)$. This will be of particular importance for the roulette wheel approach which we will discuss later. Later, when the vector x to which we are referring is clear, we will simply refer to these quantities as E , B , C , d and H .

Currently, meetings are scheduled using a very simple heuristic which cycles through all suppliers and allocates the first (supplier, delegate, timeslot) triple available from an ordered list of delegates, where the order is simply the order in which the delegates were entered onto the database. The resulting solution has $B = 226$, $C = 48.65$, $d = 99$, $H = 216$, giving a total penalty of 444.43.

We find an initial schedule using a greedy approach INITIALGREEDY as follows:

INITIALGREEDY:

Do

1. Let S_0 be a list of suppliers ordered by increasing number of scheduled priority meetings (and increasing number of total meetings where two suppliers have the same number of priority meetings in the current schedule).
2. Let D_0 be a list of delegates who currently have less than 12 meetings scheduled, ordered by decreasing number of meetings scheduled.
3. Find the first supplier $s \in S_0$ such that there is a delegate $d \in D_0$ where s and d both have a common free timeslot t , and (s, d, t) is a priority meeting.
4. If no meeting triple was found in 3, then find the first supplier $s \in S_0$ such that there is a delegate $d \in D_0$ where s and d both have a common free timeslot t , and (s, d, t) is a non-priority meeting.

Until no meeting is found in either step 3 or step 4.

By considering the most priority-meeting dissatisfied supplier first at each iteration, we attempt to treat suppliers equitably. By attempting to choose the busiest possible delegate at each iteration, we try to minimise the number of delegates in the solution. The solution produced by the constructive heuristic is used as starting solution for all hyperheuristics that we consider below. It yields a solution with $B = 52$, $C = 111$, $d = 93$, $H = 168$, giving a total penalty of 225.55.

3 Hyperheuristic Approaches to the Sales Summit Scheduling Problem

Having introduced the general nature of hyperheuristic approaches in the introduction, we will now consider the specifics of our approach for the sales summit scheduling problem given above.

The low-level heuristics which we used may all be regarded as local search neighbourhoods which accept a current solution, perform a single local search move, and return a perturbed solution. We denote these neighbourhoods $N_1, N_2, \dots, N_{\eta}$. The neighbourhoods that we used are given in the appendix. It should be noted that all of our hyperheuristic approaches are independent of the nature or number of low-level heuristics $N_1, N_2, \dots, N_{\eta}$. Each neighbourhood can be requested to actually perform the best perturbation on the current solution, or investigate the effect upon the single objective function if the neighbourhood perturbation were performed. Each neighbourhood also returns the amount of CPU time which a call used.

We have considered three different categories of hyperheuristic approaches: random approaches, greedy approaches and choice-function-based approaches. Further, for each of the approaches, we investigate two varieties. In the first variety, denoted by the suffix OI (only improving), we will only accept moves which improve the current solution. In the second variety, denoted by the suffix AM (all moves), all moves are accepted. Each hyperheuristic will continue until a stopping criterion is met, which is a time limit in all cases.

We consider three random approaches. The first, SIMPLERANDOM, randomly chooses a low-level heuristic to apply at each iteration until the stopping criterion is met. The second, RANDOMDESCENT, again chooses a low-level heuristic at random, but this time, once a low-level heuristic has been chosen, it is applied repeatedly until a local optimum is reached where it does not result in any improvement in the objective value of the solution. The third, RANDOMPERMDESCENT, is similar to RANDOMDESCENT except that first we choose a random permutation of the low-level heuristics $N_1, N_2, \dots, N_{\eta}$, and when application of a low-level heuristic does not result in any improvement, we cycle round to the next heuristic in this permutation. Note that for the AM versions of RANDOMDESCENT and RANDOMPERMDESCENT, we will carry out one move which makes the current solution worse, before moving on to a new neighbourhood.

The GREEDY approach which we consider will evaluate, at each iteration, the change in objective function value caused by each low-level heuristic upon the current solution and apply the best low-level heuristic so long as this yields an improvement. The AM and OI versions of the GREEDY approach are then identical to each other.

In the third category of hyperheuristic approaches we introduce a choice function F , that the hyperheuristic will use to decide on the choice of low-level heuristic to be called next. For each low-level heuristic the choice function F aims to measure how likely that low-level heuristic is to be effective, based upon the current state of knowledge of the region of the solution space currently under exploration. We have implemented four different methods for using the choice function. The first three methods are independent of both the low-level heuristics used and the exact details of

how the choice function is arrived at. The fourth method is also independent of the low-level heuristics used but decomposes the choice function into its component parts. We shall describe the fourth method later, once the definition of F is given. In the first (STRAIGHTCHOICE) method, we simply choose, at each iteration, the low-level heuristic which yields the best value of F . In the second (RANKEDCHOICE) method we rank the low-level heuristics according to F and evaluate the changes in objective function value caused by a fixed proportion of the highest ranking heuristics, applying the heuristic which yields the best solution. The third (ROULETTECHOICE) method assumes that for all low-level heuristics, F is always greater than zero. At each iteration a low-level heuristic N_i is chosen with probability which is proportional to $F(N_i)/\sum F(N_i)$. RANKEDCHOICE and ROULETTECHOICE are analogous to the rank-based selection and the roulette wheel selection from the genetic algorithms literature [4].

4 The Choice Function

The choice function is the key to capturing the nature of the region of the solution space currently under exploration and deciding which neighbourhood to call next, based on the historical performance of each neighbourhood. In our implementation we record, for each low-level heuristic, information concerning the recent effectiveness of the heuristic (f_1), information concerning the recent effectiveness of consecutive pairs of heuristics (f_2) and information concerning the amount of time since the heuristic was last called (f_3).

So for f_1 we have

$$f_1(N_j) = \sum_n \alpha^{n-1} \frac{I_n(N_j)}{T_n(N_j)}$$

where $I_n(N_j)$, respectively $T_n(N_j)$, is the change in the evaluation function (respectively the amount of time taken) the n th last time heuristic j was called, and α is a parameter between 0 and 1, which reflects the greater importance attached to recent performance. Then after calling heuristic N_j , the new value of $f_1(N_j)$ can be calculated from the old value using the formula

$$f_1(N_j) \leftarrow I_1(N_j)/T_1(N_j) + \alpha f_1(N_j).$$

f_1 expresses the idea that if a low-level heuristic recently improved well on the quality of the solution, this heuristic is likely to continue to be effective. Note that $I_n(N_j)$ is negative if there was an improvement and positive otherwise.

We consider that f_1 alone fails to capture much information concerning the synergy between low-level heuristics. Part of that synergy is measured by f_2 which may be expressed as

$$f_2(N_j, N_k) = \sum_n \beta^{n-1} \frac{I_n(N_j, N_k)}{T_n(N_j, N_k)}$$

where $I_n(N_j, N_k)$, resp. $T_n(N_j, N_k)$, is the change in the evaluation function (resp. amount of time taken) the n th last time heuristic k was called immediately after heuristic j and β is a parameter between 0 and 1, which again reflects the greater importance attached to recent performance. Then if we call heuristic N_k immediately after N_j , the new value of $f_2(N_j, N_k)$ can be calculated from the old value using the formula

$$f_2(N_j, N_k) \leftarrow I_1(N_j, N_k) / T_1(N_j, N_k) + \beta f_2(N_j, N_k).$$

f_2 expresses the idea that, if heuristic N_j immediately followed by heuristic N_k was recently effective and we have just used heuristic N_j , then N_k may well be effective again. Note that $I_n(N_j, N_k)$ is negative if there was an improvement and positive otherwise.

Both f_1 and f_2 are there for the purpose of intensifying the search. f_3 provides an element of diversification, by favouring those low-level heuristics that have not recently been used. Then we have

$$f_3(N_j) = \tau(N_j)$$

where $\tau(N_j)$ is the number of seconds of CPU time which have elapsed since heuristic N_j was last called.

For STRAIGHTCHOICE and RANKEDCHOICE hyperheuristics we will use the choice function F only to provide a ranking, and we will be indifferent as to the sign of F . However, for the ROULETTECHOICE hyperheuristic approach we want F to take only positive values, even for low-level heuristics which result in the objective function becoming much worse. Assume that the solution was perturbed most recently by low-level heuristic N_j . Recall that for our minimisation problem, large negative values of f_1 and f_2 are desirable. We define F as follows:

$$F(N_k) = \max \{ -\alpha f_1(N_k) - \beta f_2(N_j, N_k) + \delta f_3(N_k), \\ Q\rho^{\alpha f_1(N_k) + \beta f_2(N_j, N_k) - \delta f_3(N_k)} \}$$

Here δ is a parameter set at a value which leads to sufficient diversification,

$$Q = \frac{\sum_k \max \{ 0, -\alpha f_1(N_k) - \beta f_2(N_j, N_k) + \delta f_3(N_k) + \varepsilon \}}{10\eta}$$

where we have used $\varepsilon = 1$ and $\rho = 1.5$ to ensure that low-level heuristics which worsen the objective function value of the solution have a small, but non-zero probability of being chosen in the ROULETTECHOICE hyperheuristic, and that this probability falls rapidly for low-level heuristics which have exhibited very bad performance. The small term $\varepsilon/10\eta$ should enable every neighbourhood, no matter how bad, to be able to come around and diversify the solution after every other neighbourhood has been visited about 10 times.

The fourth (DECOMPCHOICE) method considers the individual components f_1 , f_2 and f_3 , of F . It tries the (up to four) low-level heuristics which yield the best values of

f_1 , f_2 , f_3 , and F and performs the **best move** yielded by one of these low-level heuristics.

As we can see the information used by the hyperheuristic approaches to choose low-level heuristics is not specific to the summit scheduling problem (**change in the evaluation function**, **time taken on the last call**, **time elapsed since last call** for each heuristic).

5 Results

We used each of our hyperheuristics to solve the sales summit scheduling problem described in Section 2. The hyperheuristics were implemented in C++ and the experiments were conducted on a Pentium II 225MHz with 128MB RAM running under Windows NT version 4.0. In all experiments the stopping condition was 300 seconds of CPU time. $\eta = 10$ low-level heuristics were used, all of which are simple (and easy to implement). They are based either on the methods currently used for generating a schedule, or on simple moves such as swaps.

At this stage of development we determined values of α , β and δ experimentally. We chose $(\alpha, \beta, \delta) = (0.9, 0.1, 1.5)$ for all the AM cases and $(\alpha, \beta, \delta) = (0.2, 0.2, 0.8)$ for the OI ones. Experiments with a wide range of different values of (α, β, δ) showed that, while these values produced slightly better solutions on average, the hyperheuristic was insensitive to the parameter's values. It is undesirable that parameters need to be tuned in order for a general hyperheuristic approach to be effective, but the tuning process can be automated to preserve the problem-independence of the approach. Future work will investigate adaptively changing heuristic parameters during the solution process itself.

In RANKEDCHOICE the top r neighbourhoods (with respect to F) are tested and the best neighbourhood is retained. In our experiments we chose $r = \lceil 0.25\eta \rceil$.

In the roulette wheel approach we make the choice of the next neighbourhood randomly based upon a weighted probability function. The hyperheuristic chooses a random number v in the range $[0, A]$ where

$$A = \sum_{j=1}^{\eta} F(j).$$

Given

$$a_0 = 0$$

$$a_k = \sum_{j=1}^k F(j), k = 1, \dots, \eta$$

we choose neighbourhood k if $a_{k-1} \leq v < a_k$.

All choice-function-based hyperheuristics were started with the choice function values f_1 , f_2 , and f_3 set to 0 for each low-level heuristic. In order for the choice-

function-based hyperheuristics to initialise the values of f_1 , f_2 , f_3 , and F for each neighbourhood, we randomly call the neighbourhoods for an initial warm-up period. This warm-up period is included in the time allowed to the choice-function-based hyperheuristics. In our case the warm-up lasts 100 seconds of CPU time out of the total 300 seconds allowed. Apart from GREEDY which is entirely deterministic, all our hyperheuristic are averaged over 5 runs and in each run we changed the seed used to generate random values.

Results for all of the hyperheuristic approaches as well as the greedy heuristic, which is currently used for the sales summit problem, and the INITIALGREEDY heuristic which is used to generate an initial solution for each of our hyperheuristic approaches, are given in Table 1. For each algorithm, in addition to E , we give the values of B , C , H , d , defined in Section 2. We also give m and c where m is the number of meetings scheduled and c the total number of neighbourhood calls made.

We see that our INITIALGREEDY heuristic produced a much better solution than the algorithm currently used to schedule the sales summit (original greedy heuristic). All the hyperheuristics except SIMPLERANDOM and RANDOMDESCENT produced a better solution in the AM case than in the OI case. It appears that the OI version, which does not accept neighbour moves which yield a worse solution, has a greater tendency than the AM version to get stuck early in a local optimum, from which it never escapes. The SIMPLERANDOM and RANDOMDESCENT approaches use the different low-level heuristics in an erratic and unselective manner, and in their case accepting only improving moves limits the damage done by poor random choice of low-level heuristics. Note that the GREEDY approach will always produce identical results in AM and OI cases (since the only non-improving move ever accepted in the GREEDY-AM case is the final move).

We see that the choice-function-based approaches which accept non-improving moves are all significantly better than the other approaches. The large difference between AM and OI versions of these hyperheuristics is probably due to the diversification component of the choice function being rendered ineffective in the OI case, so that we become stuck in a local optimum too early. Encouragingly, all of the choice-function-based hyperheuristics produce good results. This would lend some support to the idea that each of these approaches is a general approach which could be used for a wide range of problem instances and a wide range of problems (so long as appropriate low-level heuristics were available). Overall DECOMPCHOICE hyperheuristic performed better than all the others. It also appears that the controlled randomness of the ROULETTECHOICE yields improvement over the STRAIGHTCHOICE and RANKEDCHOICE hyperheuristics. All of these simple choice-function-based approaches appear worthy of further investigation.

Table 1. Experiment results

Algorithm	B	C	d	H	E	m	c
Original Greedy Heuristic	226.00	48.65	99.00	216.00	444.43	823.00	-
INITIALGREEDY	52.00	111.00	93.00	168.00	225.55	811.00	-
SIMPLERANDOM-AM	27.00	83.20	89.80	142.40	173.56	828.00	1102.60
SIMPLERANDOM-OI	57.80	47.20	80.80	70.40	130.56	838.00	786.60
RANDOMDESCENT-AM	53.80	32.60	86.80	118.40	173.83	847.80	825.60
RANDOMDESCENT-OI	56.40	35.40	85.00	104.00	162.17	844.60	789.40
RANDOMPERMDESCENT-AM	57.60	28.20	85.20	105.60	164.61	849.20	850.40
RANDOMPERMDESCENT-OI	52.40	23.80	87.80	126.40	179.99	852.60	866.00
GREEDY-AM	56.00	27.00	86.00	112.00	169.35	851.00	847.00
GREEDY-OI	56.00	27.00	86.00	112.00	169.35	851.00	836.00
STRAIGHTCHOICE-AM	60.00	118.00	76.20	33.60	99.50	811.80	774.40
STRAIGHTCHOICE-OI	47.80	53.20	83.20	89.60	140.06	841.40	908.00
RANKEDCHOICE-AM	44.40	84.40	78.80	54.40	103.02	824.80	880.80
RANKEDCHOICE-OI	49.40	56.60	83.20	89.60	141.83	838.40	1007.00
ROULETTECHOICE-AM	59.20	132.60	76.00	32.00	97.83	809.40	765.20
ROULETTECHOICE-OI	53.80	43.60	83.60	92.80	148.78	842.60	937.20
DECOMPCHOICE-AM	38.80	74.80	78.40	51.20	93.74	826.00	782.40
DECOMPCHOICE-OI	47.20	61.80	83.00	88.00	138.29	837.80	1014.00

6 Conclusion

We have presented the idea of a hyperheuristic, that allows us to use knowledge-poor low-level heuristics, which generally lead to poor local optima when considered in isolation, in a framework which yields results which may, in some cases, be as good or better than those provided by knowledge-rich metaheuristic approaches. We have applied a range of hyperheuristics to a real-world sales summit scheduling problem. The results obtained are far superior to those provided by the system currently used to generate schedules. We believe that this approach is promising for a wide range of scheduling and optimisation problems.

Hyperheuristics have three important advantages over knowledge-rich approaches for practical scheduling and optimisation problems. The first is that, for many practical problems, modelling the problem using simple heuristics which describe the way that the system is currently solved (often by hand) is an easy way for problem owners to consider their problem. The second advantage is that simple heuristics based upon current user practice, simple local search neighbourhoods and greedy methods are quick to implement on a computer. Since this is all that is required in order to apply a hyperheuristic method, this should yield a method for fast prototyping of decision support systems for practical scheduling and optimisation

problems. Indeed, we might simply keep adding low-level heuristics until we are satisfied (following experimentation) that we have a sufficient set of heuristics to provide good results. We then use cheap computer time to find out how best to manage the low-level heuristics, rather than expensive expertise. The third advantage is that the approach should generalise readily to small changes in the model (and indeed to large changes in the problem through the addition and modification of low-level heuristics, if necessary) yielding an approach which is robust enough to effectively handle a very wide range of problems and problem instances.

We do not regard hyperheuristics as a panacea to solve all problems in scheduling and optimisation (and so long as no fast algorithm is found for an NP-hard problem this is unlikely even to be possible). It is simply that for a very wide range of real-world problems where a reasonable solution is required in an acceptable amount of time, hyperheuristics should prove to be a useful tool.

In this paper we have introduced a range of simple **choice-function-based** hyperheuristic approaches, which are effective in spite of their simplicity, for the real-world problem which we have considered. While the details of the choice function are relatively complex, the user is shielded from this, simply supplying the objective function and the low-level heuristics. At this stage of development our hyperheuristic has several parameters, which may be tuned automatically to preserve domain-knowledge independence of the approach.

Several issues will be dealt with in future work. Parameter values will be set adaptively by the hyperheuristic in order for it to be a genuine problem-independent method applicable to a wide range of problems and instances, using different sets and types of low-level heuristics. We shall apply our approaches not only to other instances (with different but realistic objective functions) of this real-world problem but also to other personnel scheduling problems. We shall also consider how we may embed a range of more sophisticated methods into our hyperheuristic. In particular, we will consider the development of hyperheuristics which use metaheuristic techniques to decide which low-level heuristic to use, including population-based choice functions, tabu search and simulated annealing. We also intend to consider a genetic-programming approach to choice function evolution.

Appendix: The Low-Level Sales Summit Scheduling Heuristics

We used ten low-level heuristics:

1. *Remove one delegate*: This heuristic removes one delegate who has at least one meeting scheduled. It chooses the delegate with the least number of priority meetings, and the least number of meetings in total where there is a tie.
2. *Increase priority of one meeting*: This heuristic replaces one non-priority meeting with a priority meeting involving the same supplier, by changing the assigned delegate, if possible, without adding any new delegates.
3. *Add one delegate*: This heuristic adds one delegate (the delegate with the largest number of potential priority meetings) who currently has no meetings and greedily schedules as many meetings involving the new delegate as possible.

4. *Add meetings to dissatisfied supplier – version 1*: This heuristic adds as many meetings as possible to one dissatisfied supplier until the supplier is satisfied (if possible), without adding new delegates. This may only involve the deletion and rearrangement of meetings already arranged between delegates and other supplier, but only for “saturated” delegates who already have 12 meetings.
5. *Add meetings to dissatisfied or priority-dissatisfied supplier*: Same as the previous heuristic except that here the heuristic considers priority-dissatisfied suppliers (who may already have enough meetings, but not of sufficient priority) as well as dissatisfied ones.
6. *Add meetings to dissatisfied supplier – version 2*: Same as in heuristic 4, except that here the heuristic may move meetings of nonsaturated delegates who have less than 12 meetings as well as saturated ones.
7. *Cut surplus supplier meetings*: This heuristic takes each supplier who has more than 20 meetings scheduled and removes all the extra meetings (in increasing order of priority).
8. *Add meetings to priority-dissatisfied supplier*: This heuristic takes a supplier who has too few priority meetings and adds as many priority-meetings as possible to him, without adding delegates or violating the limitation on the maximum number of meetings per delegate .
9. *Add meetings to dissatisfied supplier*: Same as 8 but considers only suppliers who have enough priority meetings but too few meetings in total, and adds non-priority meetings, without adding delegates or violating the limitation on the maximum number of meetings per delegate.
10. *Add delegates and meetings to priority-dissatisfied supplier*: Same as 8 except we allow the addition of new delegates (those who do not currently have any meetings).

References

1. Aickelin, U., Dowsland, K.: Exploiting Problem Structure in a Genetic Algorithm Approach to a Nurse Rostering Problem. *J. Scheduling* **3** (2000) 139–153
2. Burke, E.K.: Cowling, P., De Causmaecker, P., Vanden Berghe, G.A.: Memetic Approach to the Nurse Rostering Problem. *Int. J. Appl. Intell.* to appear
3. Burke, E., De Causmaecker, P., Vanden Berghe, G.A.: Hybrid Tabu Search Algorithm for the Nurse Rostering Problem. *Selected Papers of the 2nd Asia–Pacific Conference on Simulated Evolution and Learning (SEAL '98). Lecture Notes in Artificial Intelligence, Vol. 1585*: Springer, Berlin Heidelberg New York (1998) 186–194
4. Back, T., Fogel, D.B., Michalewicz, Z. (eds.): *Handbook of Evolutionary Computation*. IOP Publishing and Oxford University Press (1997)
5. Dodin, B., Elimam, A.A., Rolland, E.: Tabu Search in Audit Scheduling. *Eur. J. Oper. Res.* **106** (1998) 373–392
6. Dowsland, K.A.: Nurse Scheduling with Tabu Search and Strategic Oscillation. *Eur. J. Oper. Res.* **106** (1998) 393–407
7. Easton, F.F., Mansour, N.A.: Distributed Genetic Algorithm for Deterministic and Stochastic Labor Scheduling Problems. *Eur. J. Oper. Res.* **118** (1999) 505–523
8. Mladenovic, N., Hansen, P.: Variable Neighborhood Search. *Comput. Oper. Res.* **24** (1997) 1097–1100

9. Hart, E., Ross, P., Nelson, J.: Solving a Real-World Problem Using an Evolving Heuristically Driven Schedule. *Evol. Comput.* 6 (1998) 61–80
10. Mason, A.J., Ryan, D.M., Panton, D.M.: Integrated Simulation, Heuristic and Optimisation Approaches to Staff Scheduling. *Oper. Res.* **46** (1998) 161–175
11. Meisels, A., Lusternik, N.: Experiments on Networks of Employee Timetabling Problems. *Practice And Theory of Automated Timetabling II: Selected papers. Lecture Notes in Computer Science*, Vol. 1408. Springer, Berlin Heidelberg New York (1997) 130–155
12. Tsang, E., Voudouris, C.: Fast Local Search and Guided Local Search and their Application to British Telecom's Workforce Scheduling Problem. *Oper. Res. Lett.* **20** (1997) 119–127