Hyperheuristics for Managing a Large Collection of Low Level Heuristics to Schedule Personnel

Peter Cowling and Konstantin Chakhlevitch
Modelling Optimisation Scheduling And Intelligent Computing (MOSAIC) Research Centre
Department of Computing, University of Bradford
Bradford BD7 1DP, UK
P.I.Cowling@Bradford.ac.uk
K.Chakhlevitch@Bradford.ac.uk

Abstract- This paper investigates the performance of several hyperheuristics applied to a real-world personnel scheduling problem. A hyperheuristic is a high-level search method which manages the choice of low level heuristics, making it a robust and easy to implement approach for complex real-world problems. We need only to develop new low level heuristics and objective functions to apply a hyperheuristic to an entirely new problem. Although hyperheuristic methods require limited problem-specific information, their performance for a particular problem is determined to a great extent by the quality of low level heuristics used. This paper addresses the question of designing the set of low level heuristics for the problem under consideration. We construct a large set of low level heuristics by using a technique which allows us to "multiply" partial low level heuristics. We apply hyperheuristic methods to a trainer scheduling problem using commercial data from a large financial institution. The results of the experiments show that simple hyperheuristic approaches can successfully tackle a complex real-world problem provided that low level heuristics are carefully selected to treat various constraints. We also examine how the choice of different sets of low level heuristics affects solution quality.

1 Introduction

Given its economic importance, there is continuing research interest in solving personnel scheduling problems in order to optimise measures such as worker quality of life, workforce utilisation and service quality. The purpose of personnel scheduling is to allocate the available workforce to timeslots and locations and to assign particular tasks to each member of staff.

However, real-world scheduling problems require increasingly complex models and computing optimal solutions may require prohibitive amounts of computer time. Heuristic methods are often used in practice, which produce solutions of acceptable quality in reasonable time.

Various metaheuristic approaches have been developed and successfully applied for different personnel scheduling problems. Recent examples include fast local search and guided local search algorithms applied to British Telecom's workforce scheduling problem (Tsang and Voudouris, 1997); a simulated annealing approach for shift scheduling problems presented in (Thompson, 1996); tabu search applied to audit staff scheduling (Dodin, Elimam and Rolland, 1998); and different approaches to tackle a nurse rostering problem, specifically tabu search with strategic oscillation (Dowsland, 1998), genetic algorithm (Aickelin and Dowsland, 2000), and memetic algorithms and their hybrids with tabu search (Burke et al., 2001).

Although metaheuristics and especially their hybrids proved to be quite efficient for solving some real-world scheduling problems, their application is usually dependent on problem domain. Specific metaheuristic approaches designed to effectively solve a particular problem may not be applicable or may produce very poor solutions for other problems or even for the other instances of the same problem. Metaheuristics incorporate information specific for the problem and require expertise both in the problem domain and in heuristic methods. Therefore, metaheuristics are often quite expensive to implement (Cowling, Kendall and Soubeiga, 2000). For that reason, development of general domain-independent heuristic search techniques has received increased attention among researchers. These new approaches have recently become known as *hyperheuristics* and their development is described by (Burke et al., 2003) as "an emerging direction in modern search technology".

The term "hyperheuristic" was introduced in (Cowling, Kendall and Soubeiga, 2000), as an approach that manages the choice of which low level heuristic method should be applied at any given time, depending upon the characteristics of the region of the solution space currently under exploration, and the history of each low level heuristic. This means that a hyperheuristic does not search directly for a better solution of the problem but instead it looks for a method by which a solution can be obtained. A hyperheuristic requires limited domain-specific information which is concentrated in the set of low level heuristics and the objective function(s). Low level heuristics usually represent simple local search neighbourhoods or the rules used by a human expert for constructing solutions and are usually easy to implement for real-world problems.

Hyperheuristics and evolutionary algorithms (EAs) can be considered as closely related approaches. Indeed, in traditional EA the search for the best solution is performed

in a population of feasible solutions and each member of the population has its fitness value. The fittest individuals are selected for reproduction, while recombination and mutation modify those individuals in order to produce better ones. In hyperheuristic approach, the solution is modified iteratively by applying each time some basic low level heuristic from a given set (population of low level heuristics). In terms of EA, the algorithm selects the next "fittest" (most promising) candidate in the space of low level heuristics. The fittness of the individual is measured as a performance (improvement) of low level heuristic.

This paper investigates the performance of simple hyperheuristics applied to a real-world personnel scheduling problem. We also address the question of designing the set of low level heuristics for the problem under consideration. Although hyperheuristic methods use limited problem-specific information, their performance for a particular problem is determined to a significant extent by the low level heuristics used. We split simple local search neighbourhoods into "event selection" and "resource selection" rules, then use software engineering techniques to "multiply" these approaches. Hence by writing only 27 pieces of code (where only 5 are substantially different) we are able to quickly produce 95 low level heuristics. Such an approach allows to effectively treat various constraints of the problem by having several low level heuristics which can deal with each constraint. We also examine how the choice of different subsets of low level heuristics affects the outcome of applying a hyperheuristic.

2 Literature Review

Several hyperheuristic approaches have been presented over recent years. (Gratch and Chien, 1996) develops a general adaptive problem-solving approach which automatically acquires domain-specific information and selects well-suited heuristic method from a given set. (Randall and Abramson, 2001) develops a general metaheuristic based solver for combinatorial optimisation problems. Their solver uses a linked list representation of the problem which enables rapid prototyping of solution heuristics for different problems. (Nareyek, 2003) presents a method which learns how to select promising heuristics during the search process using different weight adaptation mechanisms. The method is tested on two real-world optimisation problems.

Genetic algorithm (GA) based hyperheuristics use indirect chromosome coding, i.e. the chromosome represents a sequence of heuristics for solving the problem rather than a solution. The indirect GA evolves the heuristic choice in order to find combinations that lead to improved solutions of the problem. Fang, Ross and Corne (Fang et al., 1994) use an indirect GA for solving open shop scheduling problems. They use a set of simple dispatching rules for schedule construction. The GA developed in (Terashima-Marín, Ross and Valenzuela-Rendón, 1999) evolves constraint satisfaction strategies for examination timetabling problems. The method of evolving heuristic choice is successfully implemented in (Hart, Ross and Nelson, 1998) for a complex real-world problem of scheduling chicken catching and transportation. (Cowling, Kendall and Han, 2002) introduces a robust hyper-GA approach which can easily be reimplemented for different problems. Each individual in the hyper-GA population encodes a sequence of low level heuristics and indicates which heuristics to apply and in what order. The hyper-GA is applied for a trainer scheduling problem and produces strong results. The approach is further advanced in (Han, Kendall and Cowling, 2002) to allow the length of the chromosome to change adaptively during the search.

A novel hyperheuristic approach based on statistical ranking of low level heuristics is proposed in (Cowling, Kendall and Soubeiga, 2000, 2001, 2002a). They introduce a choice function for low level heuristics which accumulates the information about their recent performance. The choice function represents the weighted sum of three components which contain the information regarding recent performance of each low level heuristic, information about recent effectiveness of consecutive pairs of low level heuristics, and the amount of time since each heuristic was last called. The weights of the components are selected empirically (Cowling et al., 2000) or automatically adjusted as the search progresses (Cowling, et al., 2001, 2002a). The authors present hyperheuristics which employ different techniques for selection of low level heuristics based on the values of the choice function. The approaches are successfully tested on different real-world personnel scheduling problems.

The contribution of this paper is twofold. Firstly, it further investigates the application of hyperheuristic methodology for solving real-world optimisation problems. The paper presents a group of hyperheuristics where simple greedy and random approaches are effectively combined in order to achieve a good balance between intensification and diversification of the search in the space of low level heuristics. We also study the use of metaheuristic approaches at a higher level for managing the choice of low level heuristics. The previous work in this area has mostly employed GAs (Cowling, Kendall and Han, 2002 and Han, Kendall and Cowling, 2002). Here we develop hyperheuristics based on the concepts of the variable neighbourhood search (Hansen and Mladenović, 2001) and tabu search (Glover and Laguna, 1997) metaheuristics.

Secondly, we introduce an original scheme for designing a set of low level heuristics for the problem. The approach is based on the following simple idea. In order to improve (or to modify) the current solution at any time, we have to answer two questions: which part or component of the solution should be a subject of changes (WHAT question) and what changes should be applied to it (HOW question). The idea is implemented by creating two separate sets of simple rules for event selection (WHAT) and resource selection (HOW) respectively. The choice of rules allows to treat various problem constraints. The rules from two sets are combined ("multiplied") resulting in a large collection of low level heuristics. This approach is easy to implement and can be used for different real-world scheduling problems with some problem-specific modifications.

The rest of the paper is organised as follows. In section 3 we formulate the trainer scheduling problem. Section 4 describes the set of low level heuristics, in section 5 we introduce hyperheuristic approaches, and section 6 contains the discussion of the results. Section 7 concludes the paper.

3 The trainer scheduling problem

The trainer scheduling problem arises in a large commercial company which regularly organises training for its personnel. All results presented are for real-world data instances from the company. The problem involves assigning a number of training courses (events) to a limited number of training staff, locations, and timeslots. Each event has a numerical priority and the travel of each trainer is penalised depending on the distance from the home location of the trainer to the destination location where the event is conducted. The objective is to maximise the total priority for scheduled events while minimising the total travel penalty for the training staff. We use a weighted sum of these two elements in this paper, with weights given by the user.

The problem has a number of constraints:

- Each event can be delivered only by trainers from a limited pool who are competent in the particular topic of the event.
- Each event can be delivered at only one location from the limited list of possible locations for the event.
- Each location has a limited number of rooms and the rooms differ in types and capacities. Each event requires a specified number of rooms which satisfy the capacity and type requirements.
- Each event can start only within a given time window.
- Trainers are not available on pre-booked holidays. Part-time trainers work only on certain days of the week.
- Each trainer can deliver courses at most a specified proportion of the available timeslots.
- The events are compound, i.e. each event consists of one or more parts (elements). There are complex space/time/resource relationships between the elements of the event. The first element of each event should start within the time window for the event; start times of subsequent elements may be shifted relative to the start of the first element and may have their own time windows. Each element has its own requirements for rooms and trainers. Different elements may require the same trainer. The duration of each element is given and preemptions are not allowed.

We use real data from two datasets provided by a commercial company, each representing the training provided by about 50 training staff over a period of 3 months in 16 different locations, with around 200 events to be scheduled. Prior to the use of a commercial decision support system, it required about 9-person days of regional manager time to produce an acceptable quarterly schedule.

4 Low level heuristics

The design of the set of problem-specific low level heuristics for the problem is important for the quality of the outcome of applying a hyperheuristic. We use the following approaches to choosing our low level heuristics.

1) Since our primary objective is to maximise the total priority of scheduled events, we need low level heuristics which insert events which are currently not scheduled.

2) We have to include into the set heuristics concerned with decreasing travel penalties.

3) We need a range of low level heuristics to resolve conflicts in trainer, room or timeslot requirements.

4) Our low level heuristics need to recognise and deal with difficult, tightly constrained events, ideally scheduling them early in the process.

Taking into account the above considerations we propose the following scheme for low level heuristics. Each low level heuristic represents the combination of an event selection rule and a resource selection rule. Both categories of selection rules are divided into two subsets: rules for events which are not yet scheduled and rules for events which are already in the current schedule.

4.1 Event selection rules

We use 5 rules for selection from the list of not scheduled events:

ESn1. Select event at random.

ESn2. Select event with the highest priority.

ESn3. Select event with the smallest number of possible trainers.

ESn4. Select event with the smallest number of possible centres.

ESn5. Select event with the smallest number of feasible trainer/location/timeslot combinations.

For already scheduled events we consider the following 7 rules:

ESs1. Select event at random.

ESs2. Select event with the highest priority.

ESs3. Select event with the widest time window.

ESs4. Select event with the highest travel penalty.

ESs5. Select event with the largest number of possible trainers.

ESs6. Select event with the largest number of possible locations.

ESs7. Select event with the largest number of feasible trainer/location/timeslot combinations.

Note that in rules *ESn3 – ESn5* and *ESs3 – ESs7* we randomly select one event from the top 30% of events in the whole list of already scheduled events sorted according to the corresponding criterion. This allows diversity in the solutoin process without introducing too much randomness. In rules *ESn2* and *ESs2* we simply randomly select one of the events with the highest priority, since the number of events with the same priority is sufficiently large in this case.

4.2 Resource selection rules

For not scheduled events we suggest 5 resource selection rules:

- *RSn1*. Select the first found combination of available resources (if any exists) or (otherwise) resources available after unscheduling the first found conflicting event.
- *RSn2*. Select the best combination of available resources in terms of travel penalty (if any exists) or (otherwise) resources available after unscheduling a conflicting event which leads to the lowest penalty for the inserted event.
- *RSn3*. As *RSn2*, but the conflicting event with the smallest objective value is considered.
- *RSn4*. As *RSn2*, but we look for the conflicting event with the lowest priority.
- *RSn5*. As *RSn2*, but the conflicting event with the highest travel penalty is chosen.

Note that after scheduling the selected event instead of some conflicting event, we immediately attempt to schedule the latter. If there are no possible resources, we include the event into the list of not scheduled events.

In order to reschedule already scheduled events, we should find available resources which are different to the currently scheduled ones. We introduce 10 rules of resource selection:

- *RSs1*. Select the first found combination of available resources.
- *RSs2*. Select the best combination of available resources in terms of penalty.
- *RSs3*. Select a possible location with the lowest travel penalty for unchanged trainers/timeslots.
- *RSs4*. Select a possible location randomly for unchanged trainers/timeslots.
- *RSs5*. Select the next available timeslot for unchanged location/trainers.
- *RSs6*. Select possible trainers randomly for unchanged location/timeslots.
- *RSs7*. Select possible trainers with the lowest travel penalty for unchanged location/timeslots.
- *RSs8*. Select possible trainers with the maximum number of available timeslots for unchanged location/timeslots.
- *RSs9*. Select possible trainers with the minimum number of scheduled events for unchanged location/timeslots.
- *RSs10*. Select possible trainers with the maximum number of timeslots remaining under their workload limits for unchanged location/timeslots.

4.3 Heuristics

Combining event selection rules with resource selection rules for each category of events, we construct the set of 95 low level heuristics (5*5=25 heuristics for not scheduled events and 7*10=70 heuristics for scheduled events).

Although the actual number of low level heuristics is large, such an approach is quite easy to implement. Since different heuristics can use as their components the same event or resource selection rules, we create only 27 pieces of code representing different event/resource selection mechanisms, and only 5 of these pieces of code are substantially different to each other.

5 Hyperheuristics

In this section we present 3 groups of hyperheuristic approaches which can be easily applied with only minor modifications for various problems given a set of problem-specific low level heuristics and an objective function.

The first group includes simple random and greedy methods. Hyperheuristic ***Random*** selects a low level heuristic randomly from the whole set at each iteration and applies it, repeating this process until some stopping condition is true.

We consider two versions of the greedy approach. Hyperheuristic ***Greedy*** selects and applies at each iteration the best low level heuristic from the set. The "best" heuristic means either the heuristic providing the greatest inprovement to the objective function or the heuristic leading to the smallest deterioration (or yelding zero improvement) if there are no improving heuristics. Here and later through the text we consider improvements upon the *current* objective value, not upon the best value found so far. The ties are broken randomly. Hyperheuristic ***Greedy-Improvement*** uses only improving low level heuristics.

To combine random and greedy methods, a group of so-called peckish hyperheuristics is introduced. The term "peckish" is used in (Corne and Ross, 1995) which considers population initialisation algorithms for evolutionary timetabling containing features of both greedy and random methods. In our peckish hyperheuristics we experiment with various degrees of greediness and randomness aiming to achieve better balance between intensification and diversification components of the search process than in pure greedy or random approaches. Hyperheuristic ***Peckish1*** randomly selects a low level heuristic at each iteration from the candidate list of improving low level heuristics. If the candidate list is empty, a low level heuristic is selected randomly from the whole set of heuristics. Hyperheuristic ***Peckish2*** randomly selects a low level heuristic from the candidate list which contains the *n* best (not necessarily improving) heuristics. The candidate list size *n* plays an important role here: it determines how "greedy" and how "random" the hyperheuristic is – increasing the candidate list size adds randomness and decreasing it makes the hyperheuristic more greedy. Hyperheuristic ***Peckish3*** combines the features of the previous two methods. At each iteration it attempts to form the candidate list of only improving heuristics and if such a list is not empty, selects a low level heuristic randomly from it. Otherwise, random selection from the candidate list of *n* best non-improving heuristics is applied. Hyperheuristic ***Peckish4*** employs the idea similar to Variable Neighbourhood Search (Hansen and Mladenović, 2001). The candidate list size *n* is initially set to 1. If at some iteration there is an improving low level heuristic, the hyperheuristic selects the best one. Otherwise, a low level

heuristic is randomly selected from *n* best non-improving heuristics and candidate list size *n* is then incremented. The candidate list size is reset back to 1 at the iteration where the next improvement occurs. The dynamic changes of the candidate list size provide intensification/diversification of the search when necessary.

Hyperheuristics from the third group are based on the tabu search metaheuristic (Glover and Laguna, 1997). Hyperheuristic ***TabuHeuristic*** employs a tabu list of recently called heuristics. The size of the tabu list is fixed and set to some specified value *m*. The algorithm greedily selects the best low level heuristic at each iteration. If such a heuristic leads to an improved objective function value it is always selected and released from the tabu list if there; a non-improving heuristic is chosen only if it is not in the tabu list and immediately becomes tabu after its application. Hyperheuristic ***TabuEvent*** is similar to ***TabuHeuristic*** except that it is more conventional in that the tabu list holds recently selected events.

We also consider hyperheuristics ***TabuHeuristicAdaptive*** and ***TabuEventAdaptive*** which are analogous to previous two but allow the tabu list size to change adaptively as the search progresses. We decrease the tabu list size when the current solution keeps improving and increase it when we are having trouble finding better solution. After empirical evaluations of different adaptation schemes we have adopted the following one: increase the tabu list size by one in case of non-improvement and decrease it by $\lfloor current\ size/10 \rfloor *4$ when an improvement occurs, where $\lfloor x \rfloor$ denotes the largest integer which is less than or equal to *x*. The maximum value *m* for the tabu list size is also specified to prevent unlimited growth of the tabu list.

For all our hyperheuristics except ***Random*** we do not consider low level heuristics producing no changes to the current schedule (although we do consider low level heuristics which leave the objective value unchanged). This prevents "dummy" iterations and ensures that the solution is always perturbed at each iteration.

6 Experiments and results

The problem model, all low level heuristics and hyperheuristics were implemented in Microsoft Visual C++ and the experiments were conducted on a Pentium 4 1600MHz PC with 640MB RAM running under Windows 2000. We use real data from two datasets which represent two problem instances of different dimensions and with different degrees of difficulty. ***Dataset1*** contains 224 events, 53 training staff, 16 locations and 37 rooms. In ***Dataset2*** there are 147 events, 54 trainers, 16 locations and 39 rooms. The scheduling period is 3 months in both cases. In spite of significantly smaller number of events, the second instance of the problem is more difficult than the first one. The majority of the events from ***Dataset2*** have very tight time windows and very restricted lists of possible trainers and locations whereas the constraints in ***Dataset1*** are not so strict.

6.1 Initial schedules

Initial schedules are constructed using simple random and greedy heuristics. In the random approach we take events one by one in random order and select the first available combination of possible resources for each event (if any exists) until all the events have been tried. The possible trainers and locations are considered in order in which they appear in the dataset.

The greedy heuristic first sorts the events in descending order of their priority. Then each group of events with the same priority is random permuted and the final ordered list of events is created. Events are taken from the list one by one; all available combinations of possible trainers and locations are considered for each event and the combination yelding the lowest travel penalty is selected for scheduling.

We use two initial solutions of different quality for our experiments. The "poor" initial solution is the worst one among 10 solutions obtained by applying the random method and the "good" initial solution is the best one out of 10 runs of the greedy heuristic. The good initial solution is far superior to the poor one in terms of the total priority of scheduled events, the total penalty and the number of not scheduled events.

6.2 Upper bounds

Before calculating upper bounds, we analysed the results of multiple runs of hyperheuristic ***Random*** applied to both problem instances in order to check whether all the events could be scheduled. Since the random approach is very fast, we could easily allow it to perform a large number of iterations. Comparing the results of multiple runs, we found that there were always a few events which a hyperheuristic failed to schedule. Moreover, similar events (however, not always the same) repeatedly appeared in the lists of not scheduled events after each run of the hyperheuristic. Further analysing the input data, we identified the groups of events which shared conflicting trainers, locations, and timeslots. We proved that such conflicts could not be resolved and at least one event from each group of conflicting events was impossible to schedule. Therefore, the maximum possible number of scheduled events was found.

We calculate the upper bound of all possible schedules as follows. We assume that all the events are scheduled except those which are impossible to schedule due to unresolved conflicts. We select from each group of conflicting events the event with the lowest priority and form the list of not scheduled events which are excluded from consideration. The total priority for the rest of the events is calculated giving the optimal value for the total priority. Then we relax the constraints on availability of trainers and rooms, rooms' types and capacities, and on starting times for the events. For each event we identify the location/trainer(s) combination yielding the minimum travel penalty. Subtracting the sum of the lowest travel penalties for all the events from the total priority found earlier, we obtain an upper bound on the objective value.

Note that the upper bounds calculated in such a way, while useful in our experiments, are not particularly tight since a large number of constraints is ignored.

Hyperheuristic	Dataset1		Dataset2	
	good init. solution	poor init. solution	good init. solution	poor init. solution
Initial solution	2.35	10.52	8.53	17.48
Random	0.69	1.63	4.74	5.96
Greedy-Improvement	0.71	1.18	6.47	5.38
Greedy	0.30	0.40	3.01	2.33
Peckish1	0.27	0.63	3.03	2.96
Peckish2(25)	0.57	0.59	3.60	2.99
Peckish3(25)	0.30	0.56	3.54	2.56
Peckish4	0.50	0.47	3.27	2.82
TabuHeuristic(30)	0.40	0.35	2.52	2.46
TabuEvent(N/2)	0.29	0.34	2.49	2.20
TabuHeuristicAdaptive(45)	0.40	0.30	2.90	2.10
TabuEventAdaptive(45)	0.29	0.29	2.76	2.56

Table 1: Average performance of hyperheuristics for the set of 95 low level heuristics (deviation in % from upper bound)

6.3 Experiments

A single experiment includes 10 runs for each hyperheuristic described in section 4, starting from the same initial solution. Both poor and good initial solutions are tried in separate experiments for both problem instances. We test different values of parameters *n* and *m* for hyperheuristics ***Peckish2*** and ***Peckish3*** and for tabu list based hyperheuristics respectively. The stopping condition for each run is the number of iterations which is set to 500.

In addition to experiments with the whole set of 95 low level heuristics, we conduct separate experiments with subsets of 3, 5, and 10 low level heuristics. The heuristics for such the small sets are selected randomly from the whole set. We ensure that in each subset approximately the quarter of heuristics deals with not scheduled events and the remaining heuristics contain the rules for already scheduled events.

6.4 Results

The average performance of each hyperheuristic using the whole set of 95 low level heuristics is presented in Table 1. The figures represent the deviation in percent from the upper bound averaged over 10 runs of hyperheuristics, so smaller numbers represent better solutions. The numbers in parentheses after the names of some hyperheuristics are the values of either candidate list size or tabu list size. Three different values of the parameters for each hyperheuristic have been tested and the ones which provide the best results for corresponding hyperheuristics are shown in Table 1. Note that hyperheuristic ***TabuEvent*** produces its best outcomes when the tabu list may contain up to approximately half of all the events in the dataset denoted by *N*/2 in the table. Note that very small differences between values in the table represent practically very significant differences in trainer inconvenience due to additional travel, or additional low priority scheduled events.

6.5 Discussion

From Table 1 we can observe that our hyperheuristics perform well for both starting solutions. As expected, hyperheuristics ***Random*** and ***Greedy-Improvement*** produce the worst results. ***Random*** frequently selects low level heuristics which lead to deterioration of the objective function and all the solutions are quite poor in terms of travel penalty. ***Greedy-Improvement*** stops too early when there are no any improving heuristics at some iteration. We will not consider these hyperheuristics in our further discussion.

The rest of the hyperheuristics produce similar high-quality results. For "easy" ***Dataset1*** the difference from the upper bound is well under 1% and for "difficult" ***Dataset2*** it does not exceed 3.6%. There is no clear champion among hyperheuristics but it seems that tabu list based versions perform uniformly better than their peckish and greedy counterparts. The consistency of outcomes is achieved perhaps due to the wide range of low level heuristics. There are several low level heuristics in the set to deal with each objective and constraint and can resolve various conflicts. The large number of low level heuristics also allows us to consider many different events at each iteration which increases the chances of selection the "right" event whose scheduling or rescheduling leads to greater improvements of the solution and may guide the search in a promising direction.

Although the performance of hyperheuristics is quite similar for good and poor initial solutions, in terms of objective value, the structure of schedules obtained is different, as it the behaviour of the hyperheuristics. The typical behaviour of a hyperheuristic starting from a good initial solution is as follows. There are frequent improvements in the first 20-30 iterations when the hyperheuristic schedules the "easiest" not scheduled events and further improves penalties for some events. Then a sequence of iterations yielding zero improvements in the objective function follows until the next low level heuristic call which results in an improved objective, and so on. The current objective value deteriorates only rarely and then only during the runs of parametrised hyperheuristics when the candidate list size or tabu list size is large enough. The improvements become more and more rare towards the end of the run and usually represent the reductions in travel penalties. It is not easy to schedule the remaining "difficult" not scheduled events because in order to put such events into a schedule we need to change the resources for some

Hyperheuristic	Dataset1		Dataset2	
	good init. solution	poor init. solution	good init. solution	poor init. solution
Initial solution	2.35	10.52	8.53	17.48
Random	0.50	1.61	4.69	7.06
Greedy-Improvement	1.81	4.53	7.81	13.71
Greedy	0.75	0.94	4.35	4.70
Peckish1	0.43	1.31	4.37	4.80
Peckish2(3)	0.63	1.49	4.28	5.44
Peckish3(6)	0.57	1.24	3.76	4.61
Peckish4	0.43	1.11	3.75	4.79
TabuHeuristic(3)	0.66	1.12	4.11	5.08
TabuEvent(60)	0.64	1.19	4.18	4.87
TabuHeuristicAdaptive(4)	0.57	1.14	4.57	4.54
TabuEventAdaptive(30)	0.71	1.17	4.36	4.87

Table 2: Average performance of hyperheuristics for the set of 10 low level heuristics (deviation in % from upper bound)

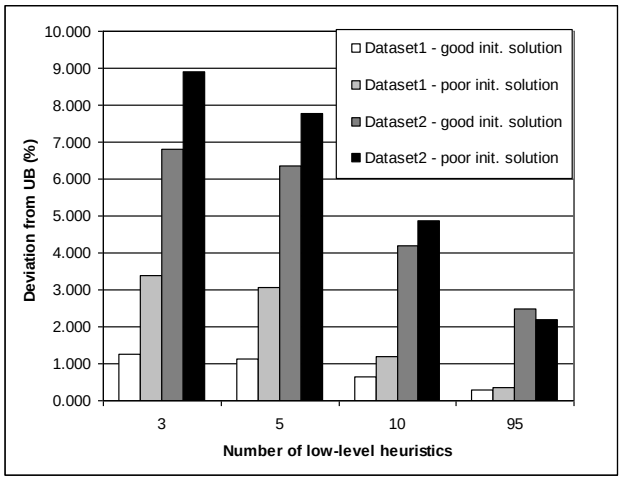

Figure 1: Average performance of hyperheuristic ***TabuEvent*** on different sets of low level heuristics

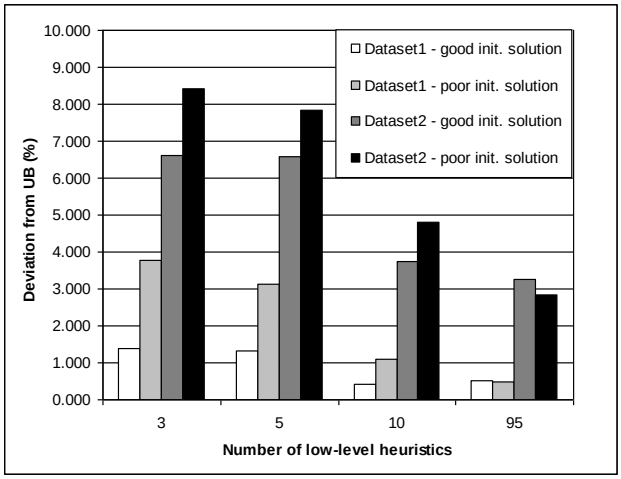

Figure 2: Average performance of hyperheuristic ***Peckish4*** on different sets of low level heuristics

conflicting events which often requires increase of travel penalties. But due to the fact that travel penalties for scheduled events in a good initial solution are already set to good values by the initial greedy heuristic and because of the certain degree of greediness in our hyperheuristics, the choice of low level heuristics at each iteration is usually limited to those yielding zero improvement. In spite of such limitations, hyperheuristics are still able to schedule the majority of the events before the stopping condition is met although each insertion of a new event requires a long sequence of iterations with zero improvements. As a result, the final schedule is very attractive in terms of travel penalty but contains some not scheduled events.

In a poor initial solution the travel penalties are often quite high. In order to achieve better values for penalties, new combinations of trainers and locations are selected which may differ significantly from the current ones. This ability to vary trainers and locations used, makes it possible to schedule new events more quickly and easily. A lot of improving iterations are observed among the first 200, then improvements become more rare and low level heuristics with zero improvements start to play their part in the search process. The schedules obtained contain more scheduled events on the average than when starting from a good initial solution and therefore higher total priority (all hyperheuristics produce the optimal value of the total priority for ***Dataset1*** except ***Greedy-Improvement***). The travel penalty for these schedules is generally worse than when we start from a good initial solution, but since total priority of scheduled events is weighted more highly than travel penalties, the schedules generated starting from poor initial solutions are generally better on the average.

The best schedules out of each group of 10 runs are usually reached starting from a good initial solution. We have also run the hyperheuristics for a much longer time (10000 iterations) and found that hyperheuristics were able to achieve solutions of similar quality starting from both good and poor initial solutions, with the good initial solution yielding slightly lower travel penalties. However, such long runs are time consuming – each run requires about 10 hours of CPU time against 25 minutes for 500 iterations.

The average performance of hyperheuristics on the reduced set of 10 randomly selected low level heuristics is presented in Table 2. We can see that the results are still quite promising although not as good as for the whole set of low level heuristics. The hyperheuristics nearly always give better performance when start from the good initial solution; the only exception is ***TabuHeuristicAdaptive*** for ***Dataset2***. The best of the combinations of 10 low level heuristics has performance similar to that of the full 95, while running in less time. The worst selection of 10 low level heuristics produces a schedule of relatively poor quality. Choosing an appropriate subset of low level heuristics from a large superset is an interesting research direction which we will pursue in the future.

The relative performance of hyperheuristics ***TabuEvent*** and ***Peckish4*** on sets of 3, 5, 10, and 95 low level heuristics is shown on Figure 1 and Figure 2. We have selected these hyperheuristics as representatives of peckish and tabu list based groups which perform uniformly well on all the sets. Figure 1 and Figure 2 show the advantages of our approach of "multiplying" event selection rules and resource selection rules for the trainer scheduling problem especially for problem instance with tight constraints. We may see from the figures that more low level heuristics is generally better.

7 Conclusions

Hyperheuristics are starting to prove themselves as fast and effective methods for solving complex real-world optimisation problems. A hyperheuristic is a robust approach which manages the choice of low level heuristics and requires limited problem-specific information. We have compared a wide range of novel hyperheuristic approaches for a real-world trainer scheduling problem.

The performance of hyperheuristics is determined to a great extent by the quality of low level heuristics used. We split simple local search neighbourhoods into "event selection" and "resource selection" rules and construct a large set of low level heuristics by using all possible

combinations of those rules. Such an approach allows us to effectively handle various constraints of the problem and requires only limited implementation time. The results of experimental study show that given such a set of low level heuristics, our hyperheuristics are able to successfully tackle two problem instances of different dimensions and degrees of complexity. Using smaller sets of low level heuristics is not so effective (unless the low level heuristics are carefully chosen), although it still leads to reasonably good schedules.

Our next steps are to consider methodologies for finding promising low level heuristics from a large set of candidates, and for forecasting the behaviour of a low level heuristic in a given situation. Both of these aim to significantly reduce CPU times, increasing the number of iterations possible and hopefully solution quality, while still meeting the low CPU times demanded by many practical scheduling decision support systems.

We believe that our methods of quickly generating large numbers of low level heuristics and putting them together with hyperheuristic AI is likely to be applicable to a wide range of complex real-world optimisation problems.

References

U. Aickelin and K. Dowsland (2000), Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem, *Journal of Scheduling* 3, 139-153.

E. Burke, P. Cowling, P. De Causmaecker, and G. Vanden Berghe (2001), A memetic approach to the nurse rostering problem, *Applied Intelligence* 15, 199-214.

E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg (2003), Hyperheuristics: an emerging direction in modern search technology, in F. Glover and G. A. Kochenberger (eds.), *Handbook of Metaheuristics*, Kluwer Academic Publishers, 457-474.

D. Corne and P. Ross (1995), Peckish initialisation strategies for evolutionary timetabling, in E. Burke and P. Ross (eds.), *Practice and Theory of Automated Timetabling*, Springer Lecture Notes in Computer Science no. 1153, Springer-Verlag, 227-240.

P. Cowling, G. Kendall, and L. Han (2002), An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem, *Proceedings of 2002 Congress on Evolutionary Computation (CEC2002)*, IEEE Computer Society Press, Honolulu, USA, 1185-1190.

P. Cowling, G. Kendall, and E. Soubeiga (2000), A Hyperheuristic Approach to Scheduling a Sales Summit, in E. Burke and W. Erben (eds.), *Practice and Theory of Automated Timetabling III: PATAT 2000*, Springer Lecture Notes in Computer Science no. 2079, Springer-Verlag, 176-190.

P. Cowling, G. Kendall, and E. Soubeiga (2001), A parameter-free hyperheuristic for scheduling a sales summit, *Proceedings of the Third Metaheuristic International Conference (MIC'2001)*, Porto, Portugal, 127-131.

P. Cowling, G. Kendall, and E. Soubeiga (2002a), Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation, in S. Cagoni, J. Gottlieb, E. Hart, M. Middendorf and R. Günther (eds.), *Applications of Evolutionary Computing: Proceedings of Evo Workshops 2002*, Springer Lecture Notes in Computer Science no. 2279, Springer-Verlag, 1-10.

B. Dodin, A. A. Elimam, and E. Rolland (1998), Tabu search in audit scheduling, *European Journal of Operational Research* 106, 373-392.

K. Dowsland (1998), Nurse scheduling with tabu search and strategic oscillation, *European Journal of Operational Research* 106, 393-407.

H.-L. Fang, P. Ross, and D. Corne (1994), A promising hybrid GA/heuristic approach for open-shop scheduling problems, in A. Cohn (ed.), *Proceedings of ECAI 94: 11th European Conference on Artificial Intelligence*, John Wiley & Sons, 590-594.

F. Glover and M. Laguna (1997), *Tabu Search*, Kluwer Academic Publishers, Norwell, MA.

J. Gratch and S. Chien (1996), Adaptive problem-solving for large-scale scheduling problems: a case study, *Journal of Artificial Intelligence Research* 4, 365-396.

L. Han, G. Kendall, and P. Cowling (2002), An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem, *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, Singapore, 267-271

P. Hansen and N. Mladenović (2001), Variable neighbourhood search: Principles and applications, *European Journal of Operational Research* 130, 449-467.

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

A. Nareyek (2003), Choosing search heuristics by non-stationary reinforcement learning, in M. Resende, and J. de Sousa (eds.), *Metaheuristics: Computer Decision-Making*, Kluwer Academic Publishers, 523-544.

M. Randall, D. Abramson (2001), A general meta-heuristic based solver for combinatorial optimisation problems, *Computational Optimisation and Applications*, 20, 185-210.

H. Terashima-Marín, P. Ross, and M. Valenzuela-Rendón (1999), Evolution of constraint strategies in examination timetabling, in W. Banzhaf et al. (eds.), *Proceedings of the GECCO99 Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 635-642.

G. Thompson (1996), A simulated annealing heuristic for shift scheduling using non-continuously available employees, *Computers and Operations Research* 23, 275-288.

E. Tsang and C. Voudouris (1997), Fast local search and guided local search and their application to British Telecom's workforce scheduling problem, *Operations Research Letters* 20, 119-127.