Guided Operators for a Hyper-Heuristic Genetic Algorithm

Limin Han, Graham Kendall
Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science and IT University of Nottingham NG8 1BB, UK
Email: lxh/gxk@cs.nott.ac.uk

Abstract. We have recently introduced a hyper-heuristic genetic algorithm (hyper-GA) with an adaptive length chromosome which aims to evolve an ordering of low-level heuristics so as to find good quality solutions to given problems. The guided mutation and crossover hyper-GA, the focus of this paper, extends that work. The aim of a guided hyper-GA is to make the dynamic removal and insertion of heuristics more efficient, and evolve sequences of heuristics in order to produce promising solutions more effectively. We apply the algorithm to a geographically distributed training staff and course scheduling problem to compare the computational result with the application of other hyper-GAs. In order to show the robustness of hyper-GAs, we apply our methods to a student project presentation scheduling problem in a UK university and compare results with the application of another hyper-heuristic method.

1. Introduction

Since their introduction by Bremermann [2] and Fraser [12] and the seminal work of Holland [17] genetic algorithms have been used to solve a whole range of problems including the travelling salesman problem [25], bin packing problems [11] and scheduling problems [24].

Personnel scheduling problems have also successfully been solved using GAs. Aickelin and Dowsland [1] used a GA for a nurse rostering problem in a large UK hospital. The result of their approach was a fast, robust implementation and the approach proved flexible and able to solve the large rostering problem in the hospital with a range of objectives and constraints. Easton and Mansour [10] also used a GA for deterministic and stochastic labour scheduling problems. Their approach is a distributed genetic algorithm, which runs in parallel, on a network of workstations. Their procedure uses a combination of feasibility and penalty methods to help exploit favourable adaptations in infeasible offspring so as to maintain the search near the feasible region. They applied this approach to three different sets of published test suites of labour scheduling problem. They compared the results to those problems solved by other meta-heuristics and conventional heuristics and found, on average, the genetic algorithm outperformed other methods.

In addition to GAs with a direct chromosome representation, indirect genetic algorithms have been studied widely. For example, Terashima-Marin, Ross and Valenzuela-Rendon [23] designed an indirect GA to solve an examination timetabling problem. They encode strategies for the problem as parameters into a 10-position array, thus the chromosome represents how to construct a timetable rather than representing the timetable itself. The use of an indirect chromosome representation avoids the limitation of a direct chromosome, which is known as the coordination failure between different parts of a solution when solving examination timetabling problem. Corne and Ogden [4] compared their indirect and direct GA for a Methodist preaching timetabling problem and found the former was more efficient.

Although genetic algorithms have been shown to be effective in solving a range of problems, they are often time intensive and require domain knowledge. Normally the chromosome of a genetic algorithm is either the solution of the target problem or a structure of the solution. This means problem specific knowledge is essential in the design of chromosome. The heavy dependence of domain knowledge makes it difficult to reuse on different problems. In order address this problem, and have a reusable, robust and fast-to-implement approach, applicable to a wide range of problems and instances, we have designed a genetic algorithm using an indirect chromosome representation based on evolving a sequence of heuristics which are applied to the problem. In this case the problem is a personnel scheduling problem that can be regarded as the allocation of staff to timeslots and possibly locations [26]. The motivation behind this approach is a hyper-heuristic, which we introduce in the next section.

2. Hyper-Heuristics

Hyper-heuristics [3] are an approach that operate at a higher level of abstraction than a meta-heuristic. It is described in [6] thus: "*The hyper-heuristics manage the choice of which lower-level heuristic method should be applied at any given time, depending upon the characteristics of the heuristics and the region of the solution space currently under exploration.*" Their hyper-heuristic consists of a set of low level heuristics and a high level heuristic selector. They defined a general framework for the hyper-heuristic which selects which low-level heuristics to apply at each decision point. A state is maintained within the hyper-heuristic which records the amount of CPU time taken by each low-level heuristic and the change in the evaluation function. A hyper-heuristic only knows whether the objective function is to be maximised or minimised and has no information as to what the objective function represents. There is no domain knowledge in the hyper-heuristic, and each low level heuristic communicates with the hyper-heuristic using a common, problem independent, interface architecture [6]. In addition to the general framework, they also have a *choice function* which decides which heuristic to call next. This is calculated based on information derived from recently called low-level heuristics. The three levels of data they maintain are the recent improvement of each individual heuristic, the recent improvement of each pair of heuristics, and the CPU time used by each heuristic. Their hyper-heuristic approach was applied to a sales summit scheduling problem [7], a project presentation scheduling problem [8], and a nurse scheduling problem [9], and they found that it solved these problems effectively. The choice function was further improved in [8].

A hyper-heuristic method was also developed by Hart, Ross and Nelson [16]. They developed an evolving heuristically driven schedule builder for a real-life chicken catching and transportation problem. The problem was divided into two sub-problems and each was solved using a separate genetic algorithm. The two genetic algorithms evolved a strategy for producing schedules, rather than a schedule itself. All the information collected from the company is put into a set of rules, which were combined into a schedule builder by exploiting the searching capabilities of the genetic algorithm. A sequence of heuristics was evolved to dictate which heuristic to use to place a task into the schedule. The approach successfully solved a highly constrained real-life scheduling problem of a Scottish company that must produce daily schedules for the catching and transportation of a huge number of live chickens. They also compared the performance of the approach to other methods such as hill-climbing and simulated annealing and found their approach to be superior.

Randall and Abramson [22] designed a general purpose meta-heuristic based solver for combinatorial optimisation problems, where they used linked list modelling to represent the problem. The problem was specified in a textual format and solved directly using meta-heuristic search engines. The solver worked efficiently and returned good quality results when being applied to several traditional combinatorial optimisation problems. Nareyek [21] proposed an approach that was able to learn how to select promising heuristics during the search process. The learning was based on weight adaptation. The configuration of heuristics was constantly updated during the search according to the performance of each heuristic under different phases of the search. The results showed that the adaptive approach could improve upon static strategies when applied to the same problems. Gratch and Chien [13] developed an adaptive problem solving system to select heuristic methods from a space of heuristics after a period of adaptation and applied it successfully to a network scheduling problem.

In our previous work we have designed an indirect genetic algorithm hyper-heuristic approach, hyper-GA, which may be regarded as a hyper-heuristic that uses a GA to select low-level heuristics [5] and further improved this approach to an adaptive length chromosome hyper-GA (ALChyper-GA), which is more parameter-adaptive [14] than our previous work. We investigated the behaviour of the algorithms for a trainer scheduling problem and believe that given an appropriate set of low-level heuristics and an evaluation function the hyper-GA approach may be applied to a wide range of scheduling and optimisation problems.

In this paper, we investigate a guided mutation and crossover hyper-GA. The motivation for this work is to find good heuristic combinations that we can use in a crossover operator in order to guide the search. We also want to identify badly performing sequences of heuristics and use mutation to remove them from a chromosome. In previous work [14] we explicitly controlled the length of the chromosome. We hope this new approach will maintain the chromosome length to reasonable limits by a method of self-adaptation. This is similar to the idea of "*select programming components based on the evolution of the program*" of genetic programming [19].

3. Geographically Distributed Course Scheduling Problem

The problem is to create a timetable of geographically distributed courses over a period of several weeks using geographically distributed trainers. We wish to maximise the total priority of courses which are delivered in the period, while minimising the amount of travel for each trainer. To schedule the events, we have 25 staff, 10 training centres (or locations) and 60 timeslots. Each event is to be delivered by one member of staff from the limited number who are competent to deliver that event. Each staff member can only work up to 60% of his/her working time (i.e. 36 timeslots). Each event is to be scheduled at one location from a limited list of possible locations. Each location, however, can only be used by a limited

number of events in each timeslot because of room availability at each site. The start time of each event must occur within a given time window. The duration of each event varies from 1 to 5 time slots. Each course has a numerical priority value. Each member of staff has a home location and a penalty is associated with a staff member who must travel to an event. The objective function is to maximise total priority for scheduled courses minus total travel penalty for trainers. A mathematical model for the problem is shown in figure 1, where we have
***E**: the set of events;* ***S**: the set of staff members;* ***T**: the set of timeslots;*
***L**: the set of locations;* ***dur**$_i$: the duration of event e_i;*
***d**$_{sl}$: the distance penalty for staff member s delivering a course at location l;*
***w**$_i$: the priority of event e_i;* ***c**$_l$: the number of room at location l*

Objective

$$MaxW = \sum_{i \in E} (w_i * \sum_{s \in S} \sum_{t \in T} \sum_{l \in L} y_{istl}) - \sum_{s \in S} \sum_{l \in L} d_{sl} \sum_{i \in E} \sum_{t \in T} y_{istl}$$

Subject

$$\sum_{s \in S} \sum_{t \in T} \sum_{l \in L} y_{istl} \leq 1 \quad (i \in E) \qquad (1)$$

$$\sum_{i \in E} \sum_{l \in L} \sum_{t \in T} x_{istl} \leq 1 \quad (s \in S) \qquad (2)$$

$$\sum_{i \in E} \sum_{s \in S} \sum_{t \in T} x_{istl} \leq c_l \quad (l \in L) \qquad (3)$$

$$x_{istl} <= \sum_{j=1}^{t} y_{istl} \quad (i \in E)(s \in S)(t \in T)(l \in L) \qquad (4)$$

$$\sum_{s \in S} \sum_{t \in T} \sum_{l \in L} x_{istl} = dur_i * \sum_{s \in S} \sum_{t \in T} \sum_{l \in L} y_{istl} \quad (i \in E) \qquad (5)$$

$$x_{istl} <= \sum_{\substack{j \in T \\ 0 <= t - j < dur_i}} y_{istl} \quad (i \in E)(s \in S)(t \in T)(l \in L) \qquad (6)$$

Figure 1. Mathematical model for the geographically distributed trainer scheduling problem

Variable y_{istl} = 1 when event e_i is delivered by staff *s* at location *l* commencing at timeslot *t*, or 0 otherwise. Variable x_{istl} = 1 when event e_i is delivered by staff *s* at location *l*, or 0 otherwise. Constraint (1) ensures that one event can happen at most once. Constraint (2) ensures that each staff member is only required to deliver at most one event in each timeslot. Constraint (3) ensures that each location has sufficient room capacity for the event scheduled. Constraints (4), (5), and (6) link the x_{istl} and y_{istl} variables, which ensures that if one event is delivered, its duration must be consecutive.

4. Low-Level Heuristics

A hyper-heuristic consists of a set of low-level heuristics and a high level selector. We have developed fourteen problem-specific low-level heuristics, which accepted a current solution, and modify it in an attempt to return an improved solution. At each generation the hyper-GA could call the set of low-level heuristics and apply them in any sequence. These low-level heuristics may be considered in four groups: (1) *add,* (2) *add-swap,* (3) *add-remove,* and (4) *remove.* The add heuristics comprise five methods which can be sub-divided into two groups. *Add-first, add-random* and *add-best* try to add unscheduled events by descending priority, and a*dd-first-improvement* and *add-best-improvement* consider the unscheduled list in a random order. There are four add-swap heuristics: *add-swap-first* and *add-swap-randomly* try to schedule one event but if there is a conflicting event when considering a particular timeslot, staff member and location, we will consider all swaps between that conflicting event and other scheduled events to see if the conflict can be resolved; *add-swap-first-improvement* and *add-swap-best-improvement* do the same but consider the unscheduled list in a random order. The mechanism of the third group (*add-remove-first, add-remove-random,* and *add-remove-worst*) is: select one event from the unscheduled event list by descending priority, if the event is in conflict with event(s) in the timetable (none of the event's possible staff members are able to work in the possible timeslots), and the event's fitness is higher than the fitness(es) of the conflicting event(s), delete the conflicting event(s) and add the unscheduled event. The last group has two heuristics: *remove-first* and *remove-random.* These two heuristics try to remove events from the schedule. They are likely to return a worse solution but will hopefully lead to an improvement later on, after other heuristics have been applied. We list all our 14 low-level problem specific heuristics as follows:

0.	*Add-first*	1.	*Add-random*
2.	*Add-best*	*3.*	*Add-swap-first*
4.	*Add-swap-randomly*	*5.*	*Add-remove-first*
6.	*Add-remove-random*	*7.*	*Add-remove-worst*
8.	*Add-first-improvement*	*9.*	*Add-best-improvement*
10.	*Add-swap-first-improvement*	*11.*	*Add-swap-best- improvement*
12.	*Remove-first*	*13.*	*Remove-random*

The integer in front of each heuristic is the integer used in the chromosome.

5. Hyper-GA and Guided Operators

5.1 Hyper-GA

Hyper-GA [5] and the adaptive length chromosome hyper-GA (ALChyper-GA) [14] are hyper-heuristics that use a GA to select low-level heuristics to produce high quality solutions for the given problem. The GA is an indirect GA with the representation being a sequence of integers each of which represents a single low-level heuristic. Each individual in a GA population gives a sequence of heuristic choices which tell us which low-level heuristics to apply and in which order to apply them. Figure 2 is an example of hyper-GA. Integer 2 represents low-level heuristic *add-best,* 3 refers *add-swap-first* and so on.

2	*3*	*1*	*5*	*0*	*7*	*9*	*8*	*11*	*1*	*10*	*9*	*12*	*13*

Figure 2. Example of hyper-GA

5.2 Adaptive Length Chromosome Hyper-GA (ALCHyper-GA)

The adaptive length chromosome hyper-GA assumes that a fixed length chromosome is not always the optimal length and aims to evolve good combinations of low-level heuristics without explicit consideration of the chromosome length. Each heuristic may work efficiently at one moment but work poorly during other periods. A heuristic such as *add-random* helps to add a few courses to the schedule during the initial generations, but it becomes less helpful when the schedule is "full". Thus, the behaviour of each heuristic can be different in different chromosomes at different times of the evolutionary process. Because the behaviour of a given low-level heuristic, or a combination of low-level heuristics, within a chromosome, could be very promising, while another low-level heuristic or combination could perform poorly, we use removal of poor-performing heuristics from a chromosome or the injection of efficient heuristics from one chromosome to another in order to try and improve the quality of solutions. As a result, the length of the chromosomes in each generation changes as genes are inserted or removed. A new crossover operator and two new mutation operators were designed for this algorithm. The *best-best crossover*, will select the best group of genes (the call of low-level heuristics by these genes that gives the most improvement of the objective function) in either selected chromosome, and exchanges them. One new mutation operator, *removing-worst mutation*, will remove the worst group of genes (the call of low-level heuristics by these genes which gives the largest decrease in the objective function, or which is the longest group giving no improvement to the objective function) in the selected chromosome. Another mutation, *inserting-good mutation*, inserts the best group of genes from a randomly selected chromosome to a random point of the desired chromosome.

The reason why we combine good groups of genes and remove bad genes is that we hope the work of particular heuristic or combination of heuristics can be helpful with in other chromosome. See [14] for detailed description of these operators.

5.3 Guided Operators

The adaptive length chromosome, hyper-GA, produced promising results with our test data sets. However, we hypotheses the algorithm should work more efficiently if removal and injection of genes can be better guided. The ALChyper-GA has the ability to identify good groups and poor groups of genes in each chromosome, but it still needs to identify whether the chromosome needs new genes to enhance the search or redundant genes need to be removed. For example, the ALChyper-GA should have the ability to know that new genes need to be added to very short chromosomes and some genes need to be removed from a chromosome when it becomes too long (and thus increases the computation time). In order to add this ability to ALChyper-GA, we wanted to design a strategy to guide the algorithm to adapt the length of chromosome more effectively and efficiently. We wanted the strategy to be able to identify when it is appropriate to remove genes and when it should inject genes. There are two points in the strategy:

a. When the chromosome is longer than the average length of chromosomes over previous generations, remove the worst-block of genes.
b. When the chromosome is shorter than the average length of chromosomes over previous generations, inject the best-block of genes from another chromosome.

These two strategies are designed to help the dynamic injection/removal of genes, and should also maintain a reasonable length chromosome. The average length should not become too short, because this will not help the solution. On the other hand, long chromosomes will be computationally expensive to evaluate.

The pseudo-code for the guided operator hyper-GA is shown below:

1. Generate an initial solution (S) randomly.
2. Generate 30 initial chromosomes (length of 14), put them into a pool
3. For each chromosome k ($0 \leq k<30$),
    a. Apply low-level heuristics in the order given in the chromosome to S
    b. Record the solution S_k (k is the position of the chromosome in the pool)
    c. Record the change each single gene makes to the objective function
4. Compare each S_k to S: if $S_k>S$, then $S=S_k$.
5. Select parents. For each pair of parents: decide which crossover operator to use (either *best-best crossover* or *one-point crossover*, choosing them with equal probability), if *best-best crossover* is used, select best groups of genes in each parent, and exchange them.
6. Select chromosomes for mutation and for each selected chromosome: decide which mutation operator to use (*inserting-good mutation* or *removing-worst mutation*). If *a* (see previous page) can be used, using *removing-worst mutation* to remove the worst group of genes from the chromosome; if *b* can be used, use *inserting-good mutation* to inject the best group of genes from a randomly selected chromosome to the desired chromosome.
7. Add all new chromosomes and 10 best chromosomes in current pool to a new pool. If the stopping criteria is met, stop the evolution, else, go to 3.

We have 4 versions of hyper-GA, two with adaptive parameters and two with non-adaptive parameters. In the adaptive versions, the mutation rate and crossover rate adapt according to the change in fitness in each generation. When there is no improvement in average fitness over 3 generations, the mutation rate will be increased using the following formula:

$$\textit{New Mutation Rate} = (\textit{Old Mutation Rate} + 1)/2 \qquad (7)$$

and the crossover rate will be decreased using :

$$\textit{New Crossover Rate} = \textit{Old Crossover Rate}/2. \qquad (8)$$

If the average fitness has improved over 3 generations, the mutation rate will be decreased using:

$$\textit{New Mutation Rate} = \textit{Old MutationRrate}/2 \qquad (9)$$

and the crossover rate will be increased using:

$$\textit{New Crossover Rate} = (\textit{Old Crossover Rate} + 1)/2 \qquad (10)$$

The aim of the modification to the crossover/mutation rates is to observe the effect these have on the performance of hyper-GA.

There are two fitness functions in our algorithm. One uses total priority minus total travelling penalty. The formula is:

$$\sum \mathrm{Pr}\,iority - \sum Travelling\ \ Penalty \qquad (11)$$

The other uses total priority minus total travelling penalty divided by the CPU time, so that improvement per unit time is the fitness. The formula for this objective function is:

$$(\sum \mathrm{Pr}\,iority - \sum TravellingPenalty)/(CPU\ \ Time\ \ in\ \ Chromosome) \qquad (12)$$

The consideration of CPU time is so as to easily compare the efficiency of each individual sequence of low-level heuristic. The comparison of these four versions can test the robustness of hyper-GA under a range of conditions.

The four versions of hyper-GA, according to the objective function and the context of parameters for mutation and crossover rate, are as follows:

- PPPN uses (11) as the objective function.

- PPPA uses same objective function as PPPN, and the crossover and mutation rate are adapted using (7)-(10).
- FTPN, uses (12) as the objective function.
- FTPA, whose objective function is the same as FTPN, and the crossover and mutation rate are adapted using (7)-(10).

PPPN, PPPA, FTPN, and FTPA are simply mnemonic names.

6. Results

All algorithms were implemented in C++ and the experiments were conducted on an AMD 800MHZ with 128MB RAM running under Windows 2000. We used five data sets, which describe realistic problem instances, each having differing degrees of difficulty [5]. Each data set contains more than 500 events. The events in each data set are generated randomly, based on the characteristics of a real staff trainer scheduling problem at a large financial institution.

In order to ascertain the correct length at which to remove and inject genes, we tried to remove genes when the chromosome was 10%, 20%, 30%, 40%, 50%, 100%, or 200% longer than the average length of chromosome in one generation, and inject genes when the chromosome was 10%, 20%, 30%, 40%, 50% shorter than the average length in our experiment. We also combined these rates so as to find the effect of them on the evolution of hyper-GA. Table 1 shows the results.

	10%	20%	30%	40%	50%
10%	**1973/1042**	1971/1021	1967/1035	1965/1024	1964/1039
20%	1972/1053	1967/1060	1969/1040	1966/1053	1968/1042
30%	1969/1031	1968/1043	1965/1031	1964/1035	1969/1052
40%	1971/1046	1969/1045	1963/1052	1965/1059	1969/1047
50%	1970/1038	1967/1049	1966/1061	1966/1049	1968/1051
100%	1970/1041	1965/1057	1967/1060	1968/1074	1967/1072
200%	1962/1074	1963/1080	1962/1079	1964/1083	1962/1076

Table 1. Results of guided ALChyper-GA applied to the basic data set [5] (percentage of shorter than average length over the generation in row, percentage of longer than average length over the generation in column. Objective (maximise)/CPU time (secs))

All combinations find relatively good results (the best result in our previous work [14] is *1961/1357*), and the CPU time is less than our previous work. However, trying to find the best combination is computationally extensive, since the sum of CPU time in table 1 is large. In order to avoid tuning parameters manually, we tried to evolve the guiding rates within the algorithm itself.

To evolve rates, we add two more genes to each chromosome, one for the rate of removing genes, the other one for the rate of injection. These two genes are real numbers in the range 0 to 1, they don't exchange information with other genes during the evolution. The evolution of these parameters is carried out by adding a random number. The result of the self-evolving rate experiment is **1972/1184**. The objective function is not better than the best result (*1973/1042*) in the fixed rate group, and the CPU time is greater. However, the chromosomes are kept to a reasonable length and the guiding parameter does not need to be manually tuned. This is important as different problems, or even problem instances could require different parameters to produce good quality solutions.

In some experiments in table 1, because the length of chromosomes drops sharply during the evolution (the average length becomes 1 or 2 after about 100 generations), we designed a new mutation, (*add-remove-mutation)*. This operator adds a *best block of genes* from a list of *best blocks* to the selected chromosome and replaces the *worst-block of genes* in that chromosome. This heuristic is added to the original 14 and is used in the experiments discussed below.We compare each of our hyper-heuristic approaches over the five problem instances (each result is the average of 5 runs).

The results of the self-evolving guiding rate hyper-GA tests on the 5 data sets are shown in table 2 along with the results of our previous work: application of hyper-GA and ALChyper-GA to the same data. We also compare with the results of the application of genetic algorithm and memetic algorithm (defined by Moscato [20]). The upper bound is calculated by solving a relaxed knapsack problem [18] where we ignore travel penalties. We can see from the table that the results of our new algorithm are all better than previous results. In addition the computation al time is less. We find that the result of guided operator hyper-GA for the basic data set is *1972/1184*, which is not better than the best value in table 1, however, the self-evolving guiding rate hyper-GA avoids manually tuning the guiding rate.

Results from our hyper-TGA [15] are also included in table 2. Although the hyper-TGA consumes less processing time, the guided-operator hyper-GA produces superior results. We suspect this is due to the increased computation required to identify the best/worst genes in each chromosome by the guided-operator hyper-GA.

Heuristics	*Basic data*	*Very few staff*	*Few staff 1*	*Few staff 2*	*Non -restricted*
Upper bound (priority)	2261	2179	2124	2244	2179
GA (30, 100) [5]	1796/1628	1633/1629	1589/1641	1706/1721	1644/1699
MA (30, 100) [5]	1832/2064	1678/2054	1617/2129	1769/2254	1698/2133
Hyper-GA PPPN [5]	1959/1456	1780/1387	1749/1404	1858/1496	1742/1422
Hyper-GA PPPA [5]	1939/1448	1754/1461	1712/1306	1854/1475	1814/1571
Hyper-GA FTPN [5]	1943/1411	1770/1437	1673/1436	1803/1422	1774/1434
Hyper-GA FTPA [5]	1951/1420	1731/1424	1738/1436	1769/1427	1770/1419
ALCHyper-GA PPPN [14]	1961/1357	1788/1250	1816/1163	1831/1591	1822/1437
ALCHyper-GA PPPA [14]	1933/1638	1757/1644	1795/1325	1862/1506	1804/1638
ALCHyper-GA FTPN[14]	1949/1450	1780/1365	1781/1277	1821/1638	1813/1488
ALCHyper-GA FTPA1[14]	1954/1526	1764/1496	1766/1364	1799/1583	1799/1419
Guided Hyper-GA PPPN	**1972/1184**	**1792/1139**	**1819/1087**	**1869/1257**	**1826/1194**
Guided Hyper-GA PPPA	1960/1208	1780/1158	1796/1135	1849/1306	1814/1270
Guided Hyper-GA FTPN	1964/1223	1786/1164	1802/1158	1852/2286	1811/1248
Guided Hyper-GA FTPA	1969/1215	1785/1162	1807/1143	1865/1324	1816/1186
Hyper-TGA PPPN [15]	1966/972	1789/911	1820/834	1866/1004	1824/941
Hyper-TGA PPPA [15]	1959/958	1782/931	1804/864	1852/996	1809/970
Hyper-TGA FTPN [15]	1960/963	1784/933	1799/856	1852/1012	1814/982
Hyper-TGA FTPA [15]	1965/985	1782/942	1811/892	1857/997	1804/930

Table 2. Comparison of Guided Operator Hyper-GA with Other Algorithms (Objective (maximise)/Time)

7. Student Project Presentation Scheduling Problem

From table 2 it is apparent that the guided-operator hyper-GA outperforms other algorithms across all problem instances for the trainer scheduling problem. In order to further demonstrate the effectiveness of the guided-operator hyper-GA, and to test the robustness and generality of our hyper-GAs, we applied the hyper-heuristic to a student project presentation scheduling problem, which was solved by Cowling et al [8].

7.1 Problem Description

Every final year BSc student in the School of Computer Science and IT at the University of Nottingham has to give a 15-minute presentation to describe his/her project. A 4-week time slot is allocated for all presentations. Each student chooses a project topic and works under the supervision of an assigned staff member. Project presentations are then organised and each student must present his work in front of a panel of three staff members (a first marker, a second marker and an observer), who will mark the presentation. Ideally, the project's supervisor should be involved in the presentation (as the first marker or the observer) but this is rarely the case in practice. Once every student has been assigned a project supervisor, the problem is to schedule all individual presentations, i.e. determine a panel of three staff members for each presentation, and allocate both a room and a timeslot to the presentation. The presentations are organised in sessions, each of which contains up to six presentations. Typically the same markers and observers will see all presentations in a particular session. So the problem can be seen as the search of (student, 1st marker, 2nd marker, observer, room, timeslot) tuples, and has the to following constraints:
(1) Each presentation must be scheduled exactly once;
(2) No more than 6 presentations in each session;
(3) Only one session in one room at one time;
(4) No marker can be scheduled to be in 2 different rooms within the same session.

Moreover, 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 inconvenient sessions (before 10 am and after 4 pm) per staff member.
(D) Optimise the match between staff research interest and project themes, and try to ensure that supervisors attend presentations for projects they supervise.

There are 151 students, 26 staff members, 60 sessions and 2 rooms involved in this problem. For further details of the problem and its formulation please refer to [8].

7.2 Application of Hyper-GAs to The Problem

Cowling et al [8] designed a hyper-heuristic choice function to select 8 low-level heuristics to solve the problem. They ran their algorithm for 600 CPU seconds. In our application, we use the same objective function and same low-level heuristics. The choice function, however, is replaced by the hyper-GAs. The initial length of each chromosome is 8. The same parameter rates and population size as for the trainer scheduling problem were used. We ran our program over 200 generations. All experimental results were averaged over 10 runs. Table 3 presents the result.

	A	B	C	D	Obj (E)	Time (CPU sec)
Choice Function	344.40	14.60	17.70	-1637.00	-1444.99	600
Hyper-GA	360.50	19.10	10.90	-1622.00	-1419.38	984
ALCHyper-GA	347.20	15.10	9.40	-1631.80	-1440.28	975
Guided Operator Hyper-GA	**325.00**	**14.80**	**5.60**	**-1632.30**	**-1453.32**	**924**

Table 3. Application of hyper-GAs to the student project presentation scheduling problem

The A, B, C, D in table 3 represent the evaluation results of A, B, C and D in 7.1. The objective function is try to minimise $E(x) = 0.5A + B + 0.3C - D$.

From table 3 we find that the guided operator hyper-GA gives the best result, though it takes longer time than Cowling et al' hyper-heuristic choice function. The result also demonstrates that the guided operator hyper-GA is superior to our other versions of hyper-GA.

8. Conclusion

The guided adaptive length chromosome hyper-GA is a further improvement of our previous work. It is a promising approach for solving personnel scheduling problems and other optimisation problems. The removal of poorly performing heuristics and the injection of promising heuristics as a guiding strategy appears to help the search. The strategy presented in this paper also helps to reduce the CPU time. We have tried many combinations of guiding rates. However, trying to find suitable parameters is computationally expensive. Therefore, we designed a new strategy that is able to evolve the removal/injection rates. The use of this strategy can evolve good quality solutions on all our testing data sets, and it improves upon our previous work. The comparison of hyper-GAs to the hyper-heuristic choice function further shows that the guided operator hyper-GA can achieve better results than other versions of hyper-GA, and hyper-GAs are robust across a range of problem instances.

References

[1] Aickelin, U, Dowsland, K, Exploiting Problem structure In A Genetic Algorithm Approach To A Nurse Rostering Problem, 2000, *Journal Of Scheduling*, vol 3, pp.139-153.

[2] Bremermann, H.J. 1958. The Evolution of Intelligence. The Nervous System as a Model of its Environment. Technical Report No. 1, Contract No. 477(17), Dept. of Mathematics, Univ. of Washington, Seattle.

[3] Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., and Schulenburg, S., *Handbook of metaheuristics*, chapter 16, Hyper-heuristics: an emerging direction in modern search technology, pp. 457–474. Kluwer Academic Publishers, 2003.

[4] Corne, D, Ogden, J, Evolutionary Optimisation of Methodist Preaching Timetables, Lecture Notes in Computer Science, PATAT1995: 142-155.

[5] Cowling, P.I., Kendall, G., and Han, L.. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. Proceedings of the Congress on Evolutionary Computation 2002, CEC 2002. Morgan Kaufman, pp. 1185-1190, 2002.

[6] Cowling, P.I., Kendall, G., Soubeiga, E., Hyperheuristic Approach to Scheduling a Sales Summit, Selected papers of *Proceedings of the Third International Conference of Practice And Theory of Automated Timetabling,* Springer LNCS vol 2079, pp. 176-190.

[7] Cowling, P.I., Kendall, G., Soubeiga, E., A Parameter-free Hyperheuristic for Scheduling a Sales Summit, 2001,*Proceedings of the Third Metaheuristic International Conference (MIC 2001)*, pp. 127-131

[8] Cowling, P.I., Kendall, G., Soubeiga, E., Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation, 2002, European Conference on Evolutionary Computation (EvoCop 2002), Springer LNCS.

[9] Cowling, P.I., Kendall, G., and Soubeiga, E., Hyperheuristics: A robust optimisation method applied to nurse scheduling, 2002, Seventh International Conference on Parallel Problem Solving from Nature, PPSN2002, Springer LNCS, pp. 851-860.

[10] Easton, F, Mansour, N, A Distributed Genetic Algorithm For Deterministic And Stochastic Labor Scheduling Problems, 1999,*European Journal of Operational Research*, pp. 505-523.

[11] Falkenauer, E., A Hybrid Grouping Genetic Algorithm for Bin Packing, 1996, Journal of Heuristics, vol 2, No. 1, pp. 5-30.

[12] Fraser, A.S., Simulation of genetic systems by automatic digital computers. II, 1957, Effects of linkage on rates under selection. Australian J. of Biol Sci, vol 10, pp 492-499

[13] Gratch, J., Chien, S., Adaptive Problem-Solving for Large-Scale Scheduling Problems: A Case Study, 1996, *Journal of Artificial Intelligence Research*, vol. 4, pp. 365-396.

[14] Han, L., Kendall, G., and Cowling, P., An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem, SEAL2002: 267-271.

[15] Han, L., Kendall, G., Investigation of a Tabu Assisted Hyper-Heuristic Genetic Algorithm, 2003, accepted by CEC’03, Perth, Australia.

[16] Hart, E, Ross, P, Nelson, J, Solving a Real-World Problem Using an Evolving Heuristically Driven Schedule Builder, 1998, *Evolutionary Computation* vol. 6, No.1, pp. 61-80.

[17] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems, Ann Arbor, MI: University of Michigan Press.

[18] Martello, S., Toth, P., *Knapsack Problems Algorithms and Computer Implementations*, 1990, John Wiley & Son Ltd, Chichester, England.

[19] Mitchell, M.. *An introduction to genetic algorithms*, 1996, MIT Press, Cambridge.

[20] Moscato, P., 1989, On Evolution, Search, Optimisation, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, report 826, Caltech Concurrent Computation Program, California Institute of Technology, Pasadena, California, USA.

[21] Nareyek, A., Choosing Search Heuristics by Non-Stationary Reinforcement Learning, 2001, in Resende, M.G.C., and de Sousa, J.P. (eds.), *Metaheuristics: Computer Decision-Making*, Kluwer Academic Publishers, pp.523-544.

[22] Randall, M, Abramson, D, A General Meta-Heuristic Based Solver for Combinatorial Optimisation Problems, 2001,*Computational Optimisation and Applications*, vol. 20, pp.185-210.

[23] Syswerda, G., Schedule Optimisation Using Genetic Algorithm, 1991, Handbook of Genetic Algorithms, Edited by Davis, L., International Thomson Computer Press.

[24] Terashima-Marin, H., Ross, P., Valenzuela-Rendon, M., Evolution of Constraint Satisfaction Strategies in Examination Timetabling, 1999, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO99)*. pp. 635-642.

[25] Whitley, D., Starkweather, T., and Shaner, D., The Travelling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, 1991,, Handbook of Genetic Algorithm, Edited by Davis, L., International Thomson Computer Press.

[26] Wren, A. Scheduling, Timetabling and Rostering - a Special Relationship? 1995, in: *ICPTAT'95- Proceedings of the International Conference on the Practice and Theory of Automate Timetabling*, pp. 475-495 Napier University