An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem

Peter Cowling* Graham Kendall+ and Limin Han+
+ Automated Scheduling, Optimisation and Planning Research Group, School of Computer Science and IT, Jubilee Campus
University of Nottingham, Nottingham, NG8 1BB, UK, Email: gxk/lxh@cs.nott.ac.uk
*Department of Computing, University of Bradford, Bradford BD7 1DP, UK, Email: Peter.Cowling@scm.brad.ac.uk

Abstract-This paper investigates a genetic algorithm based hyperheuristic (hyper-GA) for scheduling geographically distributed training staff and courses. The aim of the hyper-GA is to evolve a good-quality heuristic for each given instance of the problem and use this to find a solution by applying a suitable ordering from a set of low-level heuristics. Since the user only supplies a number of low-level problem-specific heuristics and an evaluation function, the hyperheuristic can easily be reimplemented for a different type of problem, and we would expect it to be robust across a wide range of problem instances. We show that the problem can be solved successfully by a hyper-GA, presenting results for four versions of the hyper-GA as well as a range of simpler heuristics and applying them to five test data set

1. INTRODUCTION

Personnel scheduling problems involve the allocation of staff to timeslots and possibly locations (Wren, 1995a). Personnel scheduling covers many areas, such as the nurse rostering problem (Burke et al, 1998; Burke et al, 2001 and Dowsland, 1998), transportation staff scheduling (Wren, 1995b), educational institute staff scheduling (Schaerf, 1999) and airline crew scheduling (Emden-Weinert and Proksch, 1999).

Metaheuristic approaches have been applied successfully to a range of personnel scheduling problems. For example, Aickelin and Dowsland (2000) applied genetic algorithms to a nurse rostering problem in a large UK hospital. Easton and Mansour (1999) developed a distributed GA for deterministic and stochastic labour scheduling problems, which is proved to be effective for three labour scheduling problems. Burke et al (1998) and Dowsland (1998) used tabu search to solve nurse rostering problems. Emden-Weinert and Proksch (1999) applied simulated annealing for an airline crew scheduling problem, and Thompson (1995) applied simulated annealing for shift scheduling using non-continuously available employees. Burke et al (2001) tackled a nurse scheduling problem in Belgian hospitals using a memetic algorithm.

These metaheuristic approaches can effectively solve problems, but they often require plenty of implementation time, domain knowledge and expertise in heuristics. Moreover, the reusability of these approaches may be poor because they often depend heavily on problem specific knowledge. In order to have a reusable, robust and fast-to-implement approach applicable to a wide range of problems and instances, we designed a hyperheuristic approach which will be presented later.

The work presented in this paper describes a hyper-heuristic genetic algorithm and applies it to a trainer scheduling problem. In this problem, there are a number of training events to be scheduled using a limited number of staff, locations and timeslots. The delivery of these events is highly constrained by the working ability of staff and resource limits upon time and locations. We investigate a GA-based hyper-heuristic for the problem using an indirect genetic algorithm, a hyper-GA, which may be regarded as a hyper-heuristic that uses a GA to select low-level heuristics to solve the problem. We 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 problems of scheduling and optimisation.

The term "hyper-heuristic" was introduced by Cowling, Kendall and Soubeiga (Cowling et al, 2000) as an approach that operates at a higher level of abstraction than a meta-heuristic. They described it 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 region of the solution space currently under exploration." The approach successfully solved a sales summit scheduling problem (Cowling et al, 2000), (Cowling et al, 2001) and a project presentation scheduling problem (Cowling et al, 2002). In their approach, they first design a general framework for a hyper-heuristic to select which low-level heuristic to apply, and further improve their approach using a choice function. Their choice function is calculated based on information regarding recent improvement of each low-level heuristic, recent improvement of each consecutive pair of heuristics and the amount of time elapsed since each heuristic was last called. By using the choice function, they can select which low-level heuristic to call next effectively. Hart, Ross and Nelson (Hart et al, 1998) develop an evolving heuristically driven schedule builder for a real-life chicken catching and transportation problem. In the application they divide the problem into two sub-problems and solve each using a separate genetic algorithm. The result of the two genetic algorithms is to evolve a strategy for producing schedules, rather than a schedule itself. They express the information they collect from the company into a set of rules, and combined these rules into a schedule builder by exploiting the searching capabilities of the genetic algorithm. A sequence of heuristics is evolved to dictate which heuristic to use to place a task into the schedule. Gratch and Chien (1996), develop an adaptive problem solving system, which selects proper heuristic methods from a space of heuristics after a period of adaptation, and applied it to a network scheduling problem.

Indirect genetic algorithms have been studied by a number of researchers, such as Terashima-Marin, Ross and Valenzuela-Rendon (Terashima-Marin et al, 1999), who designed an indirect GA to solve an examination timetabling problem. They have three strategies for the timetabling problem. Their

representation is a 10-position array which encodes the three strategies and parameters for guiding the search algorithm that builds the timetable. Therefore the chromosome represents the way a timetable is constructed rather than the timetable itself. Corne and Ogden (Corne and Ogden, 1997) developed an indirect GA for a Methodist preaching timetabling problem and found it is more effective than a direct genetic algorithm when applied to the same problem.

This paper is structured as follows. Section 2 will present the problem and our model. Section 3 will first discuss the genetic and memetic algorithms which we have developed, then we will present low-level problem-specific heuristics and provide a general framework for the hyper-GA. Section 4 gives experimental result and section 5 presents conclusion and possible directions for further work.

2. PROBLEM DESCRIPTION

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 limited number of events in each timeslot due to the limited number of rooms 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 event 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*

Variable y_{istl} equals to 1 when event e_i is delivered by staff s at location l commencing at timeslot t, or 0 otherwise. Variable x_{istl} equals to 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 address that if one event is delivered, its duration must be consecutive.

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 to:

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

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

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

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

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

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

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

3. IMPLEMENTATION

3.1.Genetic and Memetic Algorithms

We first develop a direct GA for the problem. The chromosome is a 25-position array with each gene representing one member of staff. Each gene holds details of the training delivered by that staff member.

Our crossover operator randomly selects two genes and after crossover, the staff members will have swapped the events they are scheduled to deliver. The crossover is combined with a repair operator that can repair a chromosome by removing conflicting events. The mutation operator randomly selects one gene to do mutation. When one gene in the chromosome is seleted to be mutated, the algorithm will try to find one event in the unscheduled event list which can be added to the schedule for a staff member.

Empirical testing has shown that a crossover rate of 0.6 (from a range of 0..1), a mutation rate of 0.03 (from a range of 0..0.2), population size 30 (from a range of 5..50), and 100 generations (from a range of 10..100) were the parameters that yielded the best results. We use elitism, which maintains the best 10 individuals from the previous generation, and the other 20 individuals are selected randomly from other members in the previous generation. The initial population is derived by randomly scheduling 30 timetables using a simple greedy hill-climbing heuristic. Table 1 lists some result of the parameter selection. From table 1 we find that the result of algorithm with bigger or smaller crossover and mutation rate cannot beat the algorithm with the rate we have used. Moreover, when the population size is small, the result is not as promising as the

algorithm using large population size. Although using population size 100 produces slightly better result than using population size 30, it consumes much more CPU time.

C/M/G	*P: 5*	*P: 30*	*P: 50*
0.6/0/50	136/1721	905/1781	1372/1765
0.6/ 0.03/50	130/1735	873/1785	1353/1759
0.6/ 1/50	132/1728	869/1774	1359/1759
1/ 0/50	154/1715	1004/1770	1460/1752
1/0.03/ 50	163/1723	1045/1778	1487/1760
0/1/ 50	125/1724	763/1745	1286/1750
0.6/ 0/100	257/1721	1736/1789	2547/1792
0.6/0.03/100	243/1735	**1628/1796**	2471/1797
0.6/ 1/ 100	246/1732	1721/1782	2501/1790
1/0/100	261/1715	1905/1779	2864/1788
1/0.03/ 100	285/1723	2053/1784	2896/1791
0/1/ 100	194/1729	1448/1750	2071/1779

TABLE 1. COMPARISON OF PARAMETERS FOR GA
C: CROSSOVER, M: MUTATION
P: POPULATION, G: GENERATION
TIME/OBJECTIVE

We also designed a memetic algorithm (Moscato, 1989) based on the direct genetic algorithm introduced above. Each member of the population is improved using one of a set of low-level heuristics that we will describe in section 3.2.1. The local search operator is applied after mutation and crossover in each generation. The mechanism of the local search is as follows. Randomly select one event from the unscheduled event list, try to add the event by using a randomly chosen *add* heuristic (see section 3.2.1). If the event cannot be added, apply a randomly chosen *add-swap* move. If none of add and add-swap works, apply a randomly chosen *add-delete* move for the event. Repeat until all the unscheduled events have been tried. The results of the memetic algorithm are listed in table 3 as comparison to the result of hyper-GA.

3.2. Hyper-GA

3.2.1 Low-level Heuristics

We have designed twelve problem-specific low-level heuristics, which accept a current solution, and modify it locally in an attempt to return an improved solution. At each generation the hyper-GA can call upon the set of low-level heuristics and apply them in any sequence. All these low-level heuristics may be considered in three groups: *add, add-swap,* and *add-delete.* 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. The add heuristics can be described as follows:

- *Add-first* tries the available staff members and locations in order until a staff member who can deliver the course at a location is found.
- *Add-random* considers the staff members and locations in a random order until a staff member who can deliver the event at a location is found.
- *Add-best* considers all possible staff and locations and selects those yielding the lowest travel penalty.
- *Add-first improvement* tries staff and locations in order as above until the first one which yields an overall improvement in the objective function is found.
- *Add-best improvement* tries all staff members and locations until the best improving combination is found.

There are four add-swap heuristics, and they are also sub-divided into two groups according the order of the unscheduled event list.

- *Swap-first* and *swap-randomly* are analogous to add-first and add-random, except that 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.
- *Swap-first improvement* and *swap-best improvement* are similarly analogous to add-first improvement and add-best improvement with the addition of this swapping step to resolve conflicts.

The mechanism of the third group (add-delete heuristics) 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 is able to work for it during its 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. This group of heuristics consists of:

- *Add-delete-first,* which tries the available staff members and locations in order for the unscheduled event until a staff member who can deliver the course at a location is found.
- *Add-delete-random,* considers the staff members and locations in a random order for the unscheduled event until a staff member who can deliver the event at a location is found.
- *Add-delete-worst,* considers all possible staff and locations for the unscheduled event and selects those yielding the lowest travel penalty.

3.2.2 Representation

The representation is a sequence of integers each of which represents one low-level heuristic. Each individual in a hyper-GA population give us a sequence of heuristic choices which tell us which low-level heuristics to use and in what order to apply them

We use one-point crossover and a mutation operator which randomly selects some positions in one chromosome and mutates integers at these positions to other values ranging from 0 to 11, (Davis, 1991). After empirical testing over a range of parameter rates, we use 0.6 for crossover rate, 0.1 for mutation rate, a population size of 30, 200 generations (100 generations gives equally good results, but we use 200 to see the further change of low-level heuristics' distribution) and retain the 30 fittest chromosomes in each generation. Table 2 lists some results of the selection of parameters, and shows that the hyper-GA approach is robust across a wide range of parameters.

C/M/G	*P: 5*	*P: 30*	*P: 50*
0.6/0/50	67/1943	390/1950	652/1952
0.6/ 0.1/50	59/1945	384/1951	620/1951
0.6/ 1/50	63/1943	392/1949	663/1949
1/ 0/50	74/1940	420/1948	752/1946
1/0.1/ 50	75/1942	434/1950	796/1948
0/1/ 50	58/1941	321/1947	530/1942
0.6/ 0/100	125/1943	790/1953	1350/1952
0.6/0.1/100	118/1945	**804/1958**	1318/1957
0.6/ 1/ 100	127/1947	872/1957	1485/1952
1/0/100	152/1940	1064/1953	1527/1950
1/0.1/ 100	152/1942	1045/1954	1513/1951
0/1/ 100	110/1948	643/1951	970/1949
0.6/0/50	193/1945	**1448/1958**	2136/1957

TABLE 2. COMPARISON OF PARAMETERS FOR HYPER-GA
C: CROSSOVER, M: MUTATION, P: POPULATION
G: GENERATION, TIME /OBJECTIVE

3.2.3 Implementation

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 formula of new mutation rate = (old mutation rate + 1)/2 and the crossover rate will be decreased using the formula of new crossover rate =old crossover rate/2. If the average fitness has improved over 3 generations, the mutation rate will be decreased and the crossover rate will be increased in a converse fashion. There are two types of fitness function in the 4 versions. One uses total priority minus total travelling penalty for the solution resulting from applying the heuristics given by the chromosome to the best solution found so far, and the other uses total priority minus total travelling penalty divided by the CPU time of the application of that chromosome, so that improvement per unit time is the fitness. The consideration of CPU time is 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.

Thirty individuals are generated for the initial population by randomly selecting numbers ranged from 0 to 11 for each gene of the chromosome.

4. RESULTS

All algorithms were implemented in C++ and the experiments were conducted on a AMD 800MHZ with 128MB RAM running under Windows 2000. We use five sets of data in this work to test the suitability of the algorithm, which describe realistic problem instances having differing degrees of difficulty. The difficulty is generally given by the average number of competent staff members and possible locations for each event. In the least difficult data set, each event can be delivered by all staff members at all locations, while in the most difficult data set, each event can be delivered by at most 5 staff members at a specified location. 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.

We compare each of our heuristics over the five problem instances. The number of trials on each problem instance is 1. The four versions of hyper-GA, according to the fitness function and the context of parameters for mutation and crossover rate, are as follows:

- **PPPN** uses total event priority minus total travel penalty as the fitness.
- **PPPA** uses same fitness function as PPPN, and the crossover and mutation rate are adapted as discussed in section 3.3 during the evolution of the algorithm.
- **FTPN**, whose fitness function is the fitness above divided by the CPU time of the application of each chromosome so that we consider the improvement per unit time.
- **FTPA**, whose fitness function is the same as FTPN, and where the mutation and crossover rate are adapted during the evolution of algorithm.

Since the result of evolution is a heuristic, we also present, in table 3, the results of applying heuristics H1, H2, …. , H5 given by five different runs of the PPPN hyper-GA on a relatively difficult problem instance (the Basic data set), to each other data set. An upper bound (in table 3) is calculated by solving a relaxed knapsack problem (Martello and Toth, 1990) where we ignore travel penalties.

Finally, in order to see the efficiency and the robustness of the hyper-GA, we compare our genetic and memetic algorithms, each low-level heuristic considered alone and our hyper-GA implementation. The stopping criterion for each low-level heuristic is when all unscheduled events have been tried for the schedule. We note that the fast, greedy heuristics H1, H2,…, H5 are all better than the best of the low-level heuristics, so that hyper-GA has found ways of combining the low-level heuristics which are effective across all instances.

From table 3 we find that our direct GA and MA implementations achieve poorer solutions than hyper-GA, in more CPU time. The time for each low-level heuristic is much shorter, but the result is much worse than hyper-GA, as well as our GA and MA implementation. Comparing Hyper-GA with its component heuristics, and with direct GA and MA heuristics, supports the idea that Hyper-GA is greater than the sum of its parts. When low-level heuristics should direct the search in a promising region, the high-level GA will call those heuristics very often, but when a low-level heuristic cannot give the algorithm further help, the algorithm will use them less frequently and change the range of heuristics used. Heuristics which include randomness, will tend to be called when diversification is needed. For example, we see that hyper-GA identifies low level heuristic delete-first, delete-random for PPPN in figure 3 and swap-random, delete-first for FTPA in figure 5 are unpromising, and their frequency remain low. Also we see that add-best-improvement for FTPA in figure 5, add-best for PPPN in figure 2, add-first for PPPA in figure 3 and add-best-improvement for FTPN in figure 4 are identified as promising, and their frequency remains high.

The overall performance of heuristics in each version of hyper-GA is different. In PPPN (figure 2), the frequency of each low-level heuristics keeps on changing even if there is no further improvement to the fitness. In PPPA (figure 3), each heuristic's performance becomes flat after generation 63 as the

Heuristics	*Basic data set*	*Very few staff*	*Few staff (1)*	*Few staff (2)*	*Non-restricted staff*
Upper bound (number/priority)	345.75 /2261.57	332.25/2179.40	323.80 /2124.12	337.33/ 2244.17	332.25 /2179.53
Add first	0.16/1502.89	0.11/1351.74	0.16/1454.75	0.16/1487.94	0.17/1471.31
Add random	0.17/1471.33	0.17/1360.04	0.16/1458.24	0.16/1488.00	0.17/1472.30
Add best	0.17/1544.18	0.17/1374.67	0.17/1460.83	0.16/1490.11	0.17/1472.85
Add first improvement	0.16/1506.84	0.16/1352.63	0.16/1451.83	0.15/1490.25	0.17/1472.04
Add best improvement	0.16/1556.75	0.16/1361.34	0.16/1453.90	0.16/1492.37	0.17/1475.93
Swap first	0.33/1551.42	0.34/1378.62	0.35/1461.21	0.31/1493.86	0.32/14776.42
Swap random	0.33/1548.90	0.34/1378.62	0.35/1460.97	0.31/1494.01	0.32/1475.98
Swap first improvement	0.32/1553.06	0.33/1380.38	0.35/1462.06	0.32/1494.59	0.33/1476.77
Swap best improvement	0.34/1553.06	0.34/1380.90	0.36/1465.49	0.31/1494.59	0.33/1476.77
Delete first	0.33/1549.28	0.32/1375.37	0.33/1461.47	0.34/1493.45	0.33/1474.83
Delete random	0.33/1545.70	0.33/1375.25	0.33/1461.89	0.34/1494.55	0.33/1475.01
Delete worst	0.35/1549.76	0.34/1377.17	0.33/1461.89	0.34/1495.79	0.33/1476.74
GA (30, 100)	1628/1796.19	1629/1633.96	1641/1589.34	1721/1706.28	1699/1644.7
MA (30, 100)	2064/1832.14	2054/1678.85	2129/1617.03	2254/1769.69	2133/1698.43
Hyper-GA (30, 200) PPPN	**1456/1959.09**	**1387/1780.15**	**1404/1749.33**	**1496/1858.92**	1422/1742.13
Hyper-GA (30, 200) PPPA	1448/1939.38	1461/1754.41	1306/1712.3	1475/1854.47	**1571/1814.38**
Hyper-GA (30, 200) FTPN	1411/1943.81	1437/1770.55	1436/1673.79	1422/1803.93	1434/1774.96
Hyper-GA (30, 200) FTPA	1420/1951.52	1424/1731.85	1436/1738.84	1427/1769.69	1419/1770.52
H1	20.19/1958.96	20.78/1629.44	21.06/1619.79	20.93/1724.08	19.98/1651.77
H2	21.53/1937.56	20.97/1597.59	21.38/1602.73	21.02/1692.25	20.31/1644.7
H3	20.74/1949.26	20.42/1617.07	22.07/1622.94	21.94/1706.28	20.59/1652.02
H4	21.37/1944.25	21.00/1629.48	21.64/1578.85	21.80/1660.97	21.06/1637.36
H5	20.76/1959.09	21.35/1582.06	20.44/1597.39	20.58/1647.42	20.27/1595.3

TABLE 3. COMPARISON OF LOW-LEVEL HEURISTICS, GA, MA AND HYPER-GA (TIME/OBJECTIVE)

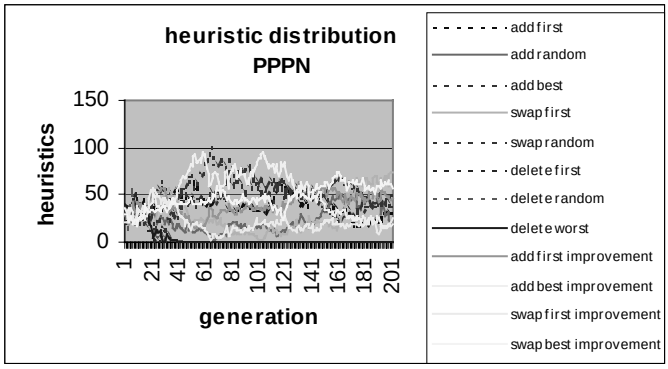

Figure 2 heuristic distribution of PPPN

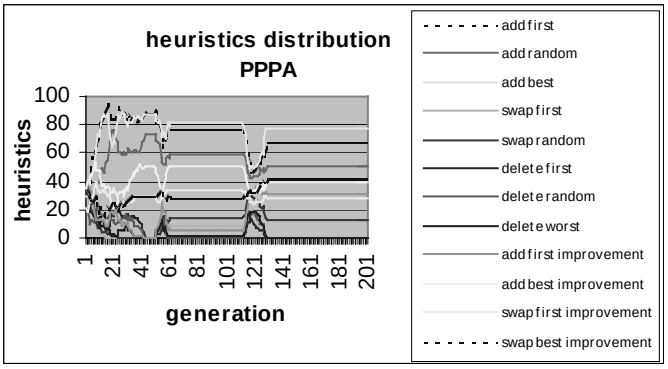

Figure 3 heuristic distribution of PPPA

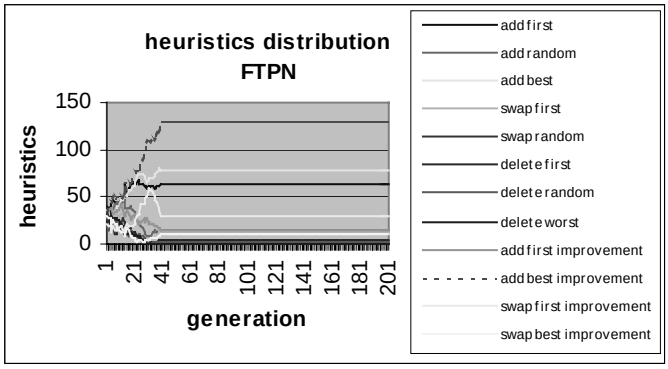

Figure 4 heuristic distribution of FTPN

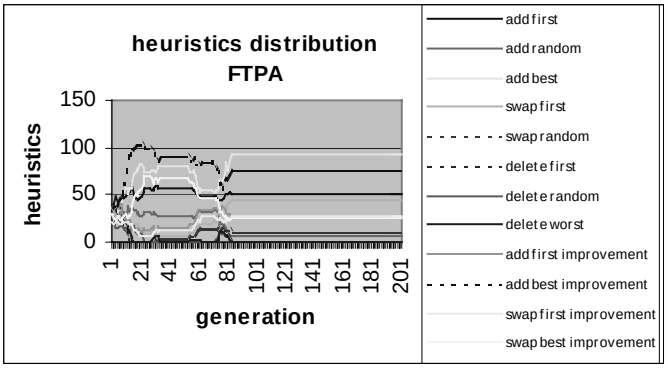

Figure 5 heuristic distribution of FTPA

population converges. There are some variances from generation 113 to generation 129 because the adaptive parameter change leads to some improvement. FTPN converges early at generation 46 (figure 5) and there are no further changes to the population. In figure 6, we see that the FTPA hyper-GA varies between being mostly crossover and mostly mutation in three large cycles, and the parameters adapt to the region of the solution space under exploration.

When we consider the approach where crossover and mutation rate are adapted, we see wide variation in the frequency of each low-level heuristic during early generations. The variation disappears later on for some generations, and appears again when diversification is needed, with the frequency of each heuristic finally converging. We can clearly see the phase change caused by diversification in the PPPA and FTPA graphs in figure 3 and 5. Note however that the adaptive approaches perform less well than the PPPN approach where parameters were carefully tuned, for all but the least constrained problem instance.

In figure 5 and 6 when the mutation rate is nearly 1 in later generations, there is no variation in the number of calls of each heuristic. This is because the selection function keeps the best chromosomes from previous generations in the current generation. In these generations, the hyper-GA converged and all chromosomes in each population are identical. New chromosomes produced by the evolution cannot enter the population, once these high quality solutions have been found.

5. CONCLUSIONS AND FUTURE WORK

Hyper-GA is a promising and generic approach to personnel scheduling and other optimisation problems. It uses a genetic algorithm as the high-level selector, with low-level heuristics as the genes in the chromosome. The evolution results in sequences of these low-level heuristics, and these low-level heuristics are applied to the problem according to the sequence specified. The nature of the low-level heuristics applied varies from one generation to the next and the hyper-GA appears to have some ability to determine when to call each heuristic, across a number of different problem instances. For the problem which we consider, hyper-GA significantly outperformed both a GA and MA, and very greatly outperformed its component heuristics.

In future, we will consider variable-length chromosomes, as well as considering different methods of parameter adaptation and other techniques to maintain population diversity. We will apply our technique to a range of real-world problems.

REFERENCES:

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.

Burke, E.K., De Causmaecker, P., Vanden Berghe, G., A Hybrid Tabu Search Algorithm For The Nurse Rostering Problem, 1998, *Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning*, vol. 1, Applications IV. pp. 187-194

Burke, E.K., Cowling, P.I., De Causmaecker, P. and Vanden Berghe, G., A Memetic Approach to the Nurse Rostering Problem, 2001, *Applied Intelligence*, vol 15, pp. 199-214.

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 (PATAT 2000)*, Springer LNCS vol 2079, pp. 176-190.

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

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

Corne, D., Ogden, J., Evolutionary Optimisation of Methodist Preaching Timetables, Lecture Notes in Computer Science: Selected papers of *the Second International Conference of Practice And Theory of Automated Timetabling (PATAT 1997)*, LNCS: 1408, pp. 142-155.

Davis, L., *Handbook of Genetic Algorithms*, 1991, Van Nostrand Reinhold, New York.

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

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

Emden-Weinert, T., Proksch, M., Best Practice Simulated Annealing For The Airline Crew Scheduling Problem, 1999, *Journal of Heuristics*, 5, pp. 419-436.

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.

Hart, E., Ross, P., Nelson, J., Solving a Real-World Problem Using an Evolving Heuristically Driven Schedule Builder, 1998, *Evolutionary Computation* vol 6, Number 1, P61-80.

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

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.

Schaerf, A., Local Search Techniques for Large High School Timetabling Problems, 1999, *IEEE Transactions on Systems, Man and Cybernetics Part A: systems and human*, vol. 29, number 1, pp. 368-377.

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

Thompson, G., A Simulated Annealing Heuristic For Shift Scheduling using Non-Continuously Available Employees, 1996, *Computers and Operations Research*, vol. 23, pp. 275-288.

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

Wren, A., Wren, D.O., A Genetic Algorithm for Public Transport Driver Scheduling, 1995b, Computers and Operations Research, vol. 22, pp. 101-110.