An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem

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

Peter Cowling
Modelling Optimisation Scheduling And Intelligent Computing (MOSAIC) research group
Department of Computing, University of Bradford, Bradford BD7 1DP, UK, Email: Peter.Cowling@scm.brad.ac.uk

ABSTRACT

Hyper-GA was introduced by the authors as a genetic algorithm based hyperheuristic which aims to evolve an ordering of low-level heuristics so as to find a good quality solution to a given problem. The adaptive length chromosome hyper-GA, let's call it ALChyper-GA, is an extension of the authors previous work, in which the chromosome was of fixed length. The aim of a variable length chromosome is two fold; 1) it allows dynamic removal and insertion of heuristics 2) it allows the GA to find a good chromosome length which could otherwise only be found by experimentation. We apply the ALChyper-GA to a geographically distributed training staff and courses scheduling problem, and report that good quality solution can be found. We also present results for four versions of the ALChyper-GA, applied to five test data sets.

1. INTRODUCTION

This paper uses a new approach to solve a trainer scheduling problem, which is in the range of personnel scheduling problems that can be regarded as the allocation of staff to timeslots and possibly locations [20].

Metaheuristic approaches have been successfully applied to a range of personnel scheduling problems. Aickelin and Dowsland [1] used GAs for a nurse rostering problem in a large UK hospital. Easton and Mansour [11] also used GAs for deterministic and stochastic labour scheduling problems, which was effective for three labour scheduling problems. Burke et al [2] and Dowsland [10] used tabu search to solve nurse rostering problems. Burke et al [3] used Memetic algorithm to tackle a nurse scheduling problem in Belgian hospitals. Simulated annealing was applied by Thompson [19] for shift scheduling using non-continuously available employees.

These metaheuristic approaches are proved effectively solving problems, but they are often time and domain knowledge intensive. The heavy dependence on problem specific knowledge also worsens the reusability of these approaches. In order to remove the disadvantage of metaheuristic approaches and have a reusable, robust and fast-to-implement approach applicable to a wide range of problems and instances, we designed a genetic algorithm based hyperheuristic approach, hyper-GA, which may be regarded as a hyper-heuristic that uses a GA to select low-level heuristics to solve the problem [4] in our previous work. We investigated the behaviour of the algorithm for a trainer scheduling problem and believed 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. In this paper, we designed a more parameter-adaptive algorithm, Adaptive Length Chromosome hyper-GA (ALChyper-GA). The motivation of ALChyper-GA is that we don't know the optimal length and want to encourage the evolution of good combinations of low-level heuristics without having to explicitly consider this length. This is related to ideas of genetic programming, which is "select programming components based on the evolution of the program" [15].

Cowling, Kendall and Soubeiga introduced the term "hyper-heuristic" [5] as an approach that operates at a higher level of abstraction than a meta-heuristic. It is described 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." The approach has been used successfully to solve a sales summit scheduling problem [5], [6] and a project presentation scheduling problem [7]. 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 of recent called low-level heuristics. The information includes the improvement of each individual heuristic or each pair of heuristics and the consuming of time of each heuristic. By using the choice function, they can effectively call next low-level heuristic.

Other authors have developed hyperheuristic method. Hart, Ross and Nelson [13] develop an evolving heuristically driven schedule builder for a real-life chicken catching and transportation problem. The problem is divided into two sub-problems and each is solved using a separate genetic algorithm. The two genetic algorithms evolve a strategy for producing schedules, rather than a schedule itself in the application. All the information collected from the company is put into a set of rules, which are combined 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 [12], develop an adaptive problem solving system to select proper heuristic methods from a space of heuristics after a period of adaptation, and apply it to a network scheduling problem. Randall and Abramson [17] designed a general meta-heuristic based solver for combinatorial optimisation problems, where they used linked list modelling to represent problem, thus the problem was specified in a textual format and solved directly using meta-heuristic search engines. Nareyek [16] proposed an approach that was able to learn how to select promising heuristics during the search process.

Indirect genetic algorithms have been studied widely. For example, Terashima-Marin, Ross and Valenzuela-Rendon [18] designed an indirect GA to solve an examination timetabling problem. They encode their strategies for the problem and parameters for guiding the search into a 10-position array, thus the chromosome represents how to construct a timetable rather than the timetable it self. Corne and Ogden [8] compared their

indirect and direct GA for a Methodist preaching timetabling problem and found the former is more efficient.

2. PROBLEM DESCRIPTION

The problem in this work is a trainer scheduling problem aiming 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. We have 25 staff, 10 training centres (or locations) and 60 timeslots to schedule events. For details of the problem, please refer to [4]. The mathematical model for the problem is cited here and 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 aaadelivering a course at location l;*
w_i*: the priority of event* e_i*;* c_l*: the number of room at aaalocation 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 to:

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

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

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

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

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

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

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

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.

In carrying out crossover and mutation, we do not allow infeasible solutions. Each event in the new chromosome will be checked to see whether it conflicts with other event(s) in the same chromosome. If this is the case, the event(s) with lower priority-penalty will be removed from the chromosome.

3. HYPER-GA, ALCHYPER-GA AND LOW-LEVEL HEURISTICS

3.1. Hyper-GA

Hyper-GA, as stated in section 1, is a hyper-heuristic that uses a GA to select low-level heuristics to solve 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 hyper-GA population gives a sequence of heuristic choices which tell us which low-level heuristics to use and in what order to apply them [4].

3.2. Adaptive Length Chromosome Hyper-GA

The adaptive length chromosome hyper-GA (ALChyper-GA) is an improvement of hyper-GA. We assume the fixed length in the hyper-GA is not always the optimal length and want to encourage the evolution of good combinations of low-level heuristics without having to explicitly consider this optimal length. The behaviour of a given low-level heuristic or a combination of low-level heuristics, within a chromosome, could be very promising, while anther low-level heuristic or combination of heuristics could perform poorly. We hypothesise that if we remove the poor-performing heuristics from a chromosome or inject efficient heuristics from one chromosome to another then better quality solutions can be found as a result. Therefore, the length of chromosomes in each generation will change as genes are inserted or removed.

Within each chromosome we monitor the change of the objective function as the chromosome is evaluated. We also use the change in the objective function as we evaluate each chromosome to decide which blocks of genes are suitable for the crossover and mutation operators (see 3.4). An improvement between *gene m* and *gene n* (potentially), which means the call of low-level heuristics between gene m and gene n improves the objective function, indicates that the heuristics between *m* and *n* work well together and other chromosome might benefit by having these heuristics injected into them. Conversely, a worsening of the objective function (or no improvement) could indicate that the chromosome might perform better if these genes were removed from the chromosome.

The ALChyper-GA uses specially designed crossover and mutations to insert or remove groups of genes. We have also designed a penalising function to penalise the length of chromosome in the case where the length increases which will result in increased run times due to the additional evaluation required. The formula for the penalising function is:

(Length in chromosomes * CPU time to evaluate chromosome)/ (Improvement in objective function)

All chromosomes are sorted by descending order of their penalising function and proportional selection ensures that shorter, better improving chromosomes have a higher chance of being selected.

3.3. Low-level Heuristics

We use the same low-level heuristics as our previous work [4]. The twelve problem-specific, low-level heuristics for our hyper-GA, which accepts a current solution, modifies it locally in an attempt to return an improved solution. In addition to these twelve low-level heuristics, we designed two additional low-level heuristics for ALChyper-GA, which are likely to return a worse solution but will hopefully lead to an improvement later on, after other heuristics have been applied. The aim of this addition is to observe the adaptation of ALChyper-GA when there is a decrease

in the objective function. At each generation the hyper-GA can call upon the set of low-level heuristics and apply them in any sequence. All the low-level heuristics may be grouped into: *add, add-swap*, and *add-delete, please refer to [4] for details of these low-level heuristics.*

3.4. Operators

The hyper-GA uses 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 (0 to 13 in ALChyper-GA), [9].

We have also designed a new crossover operator and two new mutation operators for ALChyper-GA. The new crossover, called *best-best crossover*, will select the best group of genes (the call of low-level heuristics by these genes gives the most improvement of the objective function) in either selected chromosome, and exchange them. One new mutation, *removing-worst mutation*, will remove the worst group of genes (the call of low-level heuristics by these genes gives the most decrease to 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. Thee *best-best* crossover is illustrated in figure 2. Parent 1 and 2 are a chromosomes selected to crossover. The improvement in objective function of parent 1 is *57.85*, where the grey area is the group of genes (*gene 5 to 9*) that contribute most *(21.30)*. The improvement in objective function of parent 2 is *35.81*, the group of genes in the black area (*gene 7 to 13*) gives the most improvement *(16.77)*. Thus, *gene 5 to 9* in parent 1 and *gene 7 to 13* are best groups of genes in either parent. These two groups are selected and exchanged to form child 1 and child 2.

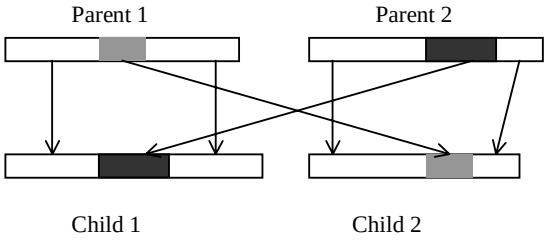

Fig 2 best-best crossover

4. IMPLEMENTATION

We have 4 versions of ALChyper-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 of each generation [4]. There are two types of fitness function in the 4 versions: $\sum \Pr iority - \sum TravellingPenalty$ and $(\sum \Pr iority - \sum TravellingPenalty)/(CPU\ Time \quad in \quad Chromosome)$

The total priority and total travelling penalty are for the solution resulting from applying the heuristics given by the chromosome to the best solution found so far. The consideration of CPU time in the second fitness function is to easily compare the efficiency of each individual sequence of low-level heuristic and use for the penalising function of the length of chromosome. The comparison of these four versions can test the robustness of ALChyper-GA under a range of conditions.

Thirty individuals are generated for the initial population by randomly selecting numbers ranged from 0 to 13 for each gene of the chromosome. After empirical testing over a range of parameter rates [4], 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.

5. 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 data sets to test the suitability of the algorithm, which describe realistic problem instances having differing degrees of difficulty [4]. 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.

We compare each of our heuristics over the five problem instances. The four versions of ACLhyper-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.

Table 1 presents our results. We compare the result of ALChyper-GA to hyper-GA, Genetic and memetic algorithm so as to see the efficiency and the robustness. The result of a mutation only approach (with only the two new mutation operators in ALChyper-GA) is also presented in the table. We also present the result 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. The upper bound is calculated by solving a relaxed knapsack problem [14] where we ignore travel penalties.

We find that the ALChyper-GA performs better than the hyper-GA for most problem instances. The latter is found producing better result than genetic and memetic algorithm, and the fast, greedy heuristics H1,…, H5. From table 1 we can see that the improvement in objective in few staff (1) data set and non-restricted staff data set is bigger than other three data sets, which suggests that ALChyper-GA works well, even on more difficult problems. Although results of the mutation only approach in table 1 are not as good as most applications of the four versions of ALChyper-GA, the results and the run time of the approach supports that the work of the two new mutation operators can improve the efficiency of the algorithm.

Comparing ALChyper-GA with hyper-GA supports the idea that if hyper-GA was able to find the optimal length chromosome, it will perform better. However, the ALChyper-GA has the ability to find a good quality length chromosome as it evolves. Indeed, as the search progresses the optimal length of the chromosome changes and the ALChyper-GA is able to react to this need. Figure 3 illustrates the change in length of the chromosome. The top part of the figure is for the basic data set, while the bottom part is for the very few staff data set. From the top part of the figure, we see that ALChyper-GA settles on a stable length chromosome for each individual in the population from generation 24 to generation 66 (the average is 6), at which time ALChyper-GA is at a local optimum with respect to the objective function (1955.28). Then, the length is changed by generation 67 and the algorithm .

Heuristics	*Basic data set*	*Very few staff*	*Few staff (1)*	*Few staff (2)*	*Non-restricted staff*
Upper bound (priority/number)	2261.57/345.7	2179.40/332.25	2124.12/323.80	2244.17/337.33	2179.53/332.25
GA (30, 100)	1796.19/1628	1633.96/1629	1589.34/1641	1706.28/1721	1644.7/1699
MA (30, 100)	1832.14/2064	1678.85/2054	1617.03/2129	1769.69/2254	1698.43/2133
Hyper-GA (30, 200) PPPN	*1959.09/1456*	*1780.15/1387*	*1749.33/1404*	*1858.92/1496*	1742.13/1422
Hyper-GA (30, 200) PPPA	1939.38/1448	1754.41/1461	1712.3/1306	1854.47/1475	*1814.38/1571*
Hyper-GA (30, 200) FTPN	1943.81/1411	1770.55/1437	1673.79/1436	1803.93/1422	1774.96/1434
Hyper-GA (30, 200) FTPA	1951.52/1420	1731.85/1424	1738.84/1436	1769.69/1427	1770.52/1419
ALCHyper-GA (30, 200) PPPN	**1961.64/1357**	**1788.49/1250**	**1816.15/1163**	1831.94/1591	**1822.94/1437**
ALCHyper-GA (30, 200) PPPA	1933.40/1638	1757.99/1644	1795.36/1325	**1862.00/1506**	1804.33/1638
ALCHyper-GA (30, 200) FTPN	1949.00/1450	1780.35/1365	1781.83/1277	1821.42/1638	1813.62/1488
ALCHyper-GA (30, 200) FTPA	1954.11/1526	1764.34/1496	1766.20/1364	1799.04/1583	1799.51/1419
ALCHyper-GA with only mutation	1880.50/1486	1769.71/1188	1780.13/1083	1783.15/1413	1788.13/1383
H1	1958.96/20.19	1629.44/20.78	1619.79/21.06	1724.08/20.93	1651.77/19.98
H2	1937.56/21.53	1597.59/20.97	1602.73/21.38	1692.25/21.02	1644.7/20.31
H3	1949.26/20.74	1617.07/20.42	1622.94/22.07	1706.28/21.94	1652.02/20.59
H4	1944.25/21.37	1629.48/21.00	1578.85/21.64	1660.97/21.80	1637.36/21.06
H5	1959.09/20.76	1582.06/21.35	1597.39/20.44	1647.42/20.58	1595.3/20.27

TABLE 1. COMPARISON OF ALCHYPER-GA AND HYPER-GA (OBJECTIVE/TIME)

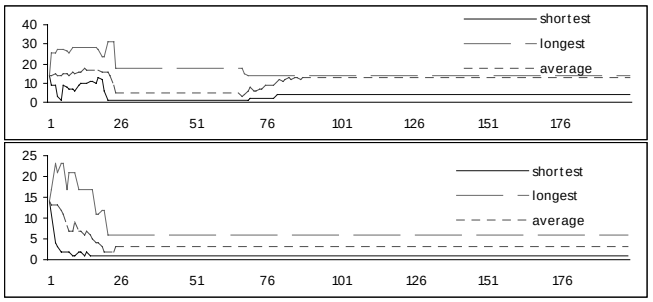

Fig 3. Change of chromosome length for the basic data set, and the very few staff data set (length/generation)

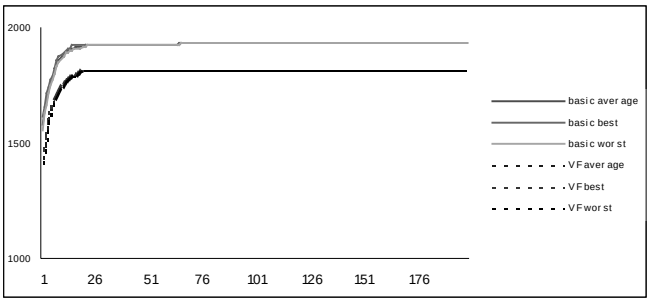

Fig 4. Objective function value for the basic data set, and the very few staff data set (objective/generation)

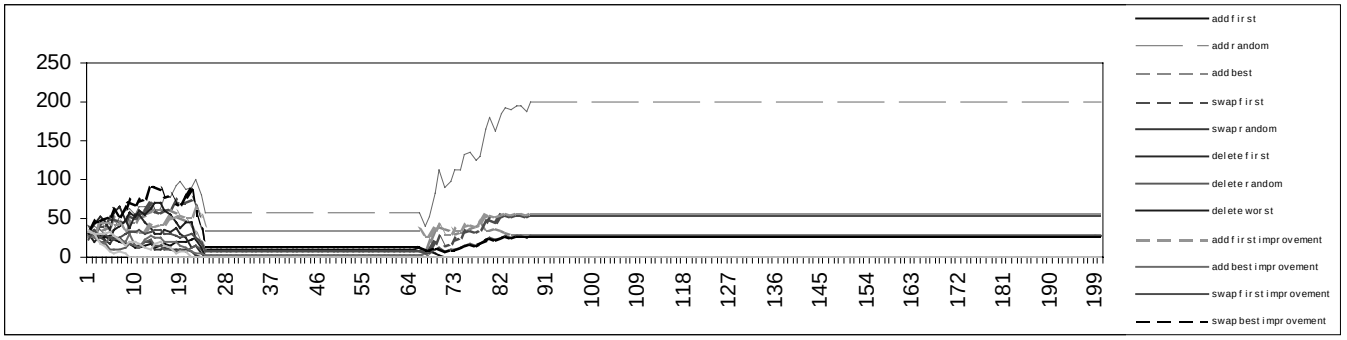

Fig 5. Heuristic distribution for the basic data set (frequency/generation)

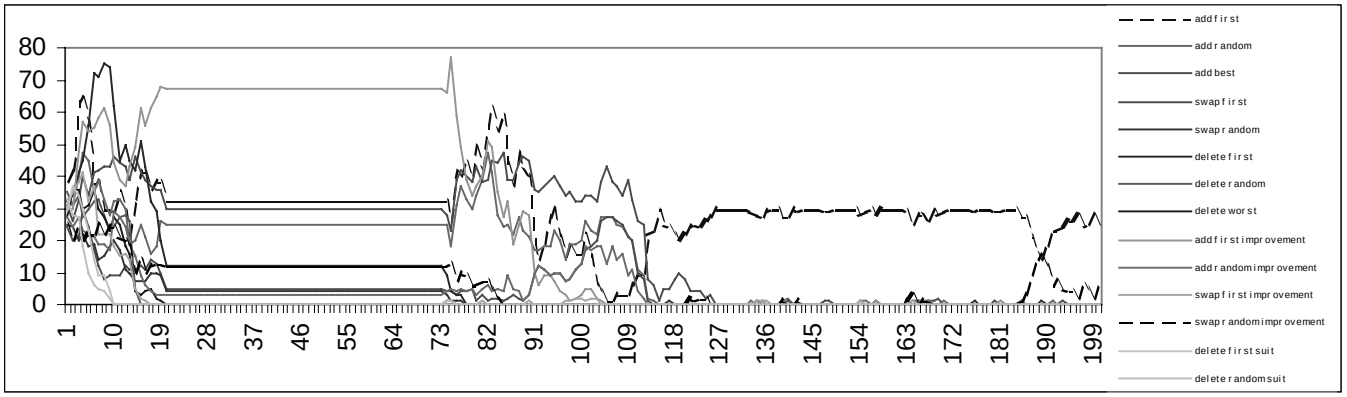

Fig 6. Heuristic distribution for the very few staff data set (frequency/generation)

behaviours changes at the same time. In the mean time, the length of individual chromosomes varying before becoming flat at generation 89. Similar performance can also be found in the bottom part of the figure, for a different dataset.

The top part of figure 3 and figure 5 compare the change in length of chromosome and the heuristic distribution for basic data set, while the bottom part of figure 3 and figure 6 compare for very few staff data set. The reason why we select the two data sets is that the performance for the few staff (1) and the few staff (2) is similar to basic data set, and the performance for the non-restricted data set is similar to the very few staff data set. From figures 3, 5, and 6 we find that the frequency of each low-level heuristic becomes flat as long as there is no change in the length of chromosome. However, when the chromosome length becomes static at generation 117, there are only two low-level heuristics left, (*add-first* and *swap-random-improvement),* and the two low-level heuristics keep on varying after generation 117.

Figure 4 illustrates the change of objective function. The upper group of lines are for the basic data set, while the lower group of lines are for the very few staff data set. We can also find the objective function of the basic data set is improved from 1955.28 to 1961.64 at generation 67 in the figure, while the objective function of the very few staff data set is improved from 1787.63 to 1788.49 at generation 75. Both of the two improvements happen at the generation when variation appears again.

From the top of figure 3 we can find another phenomenon in that the longest chromosome becomes shorter, the shortest chromosome becomes longer, and the average length increases by generation 67. While in the bottom of figure 3 both the length of the longest chromosome and average length drop down by generation 75 and all three lengths become 1 by generation 117. This is because of the work of our new mutation operators. The two mutation operators keep variations in each population, and try to keep selected chromosome efficient.

6. CONCLUSIONS AND FUTURE WORK

ALCHyper-GA is a promising approach to personnel scheduling and other optimisation problems. It is the further improvement of hyper-GA. The length of chromosome adapt after the identification of performance of individual heuristic or combination of heuristics selected by the chromosome. Three new operators: *best-best crossover, removing-worst mutation,* and *inserting-good mutatio*n help to dynamically insert or remove heuristics. This algorithm outperforms the hyper-GA, while the later has better performance than GA, MA and its component heuristics which were presented in (Cowling et al, 2002).

In future, we will consider different methods of parameter adaptation and maintain the diversity of population. We will also consider the combination of hyper-GA with other metaheuristics, such as tabu search and simulated annealing. We also plan to test the robustness of our algorithm to a range of real-world problems.

7. ACKNOLEDGEMENTS

We would like to thank the reviewers for their constructive comments on this paper.

8. 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] 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

[3] 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.

[4] Cowling, P.I., Kendall, G., Han, L., An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. Accepted for 2002 Congress on Evolutionary Computation (CEC2002), Hilton Hawaiian Village Hotel, Honolulu, Hawaii, May 12-17, 2002.

[5] 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.

[6] 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

[7] 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.

[8] 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.

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

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

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

[12] 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.

[13] 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.

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

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

[16] Nareyek, A., Choosing Search Heuristics by Non-Stationary Reinforcement Learning, 2001, submitted for the MIC post-proceedings book.

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

[18] 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

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

[20] 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.