An Investigation of a Tabu Assisted 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-This paper investigates a tabu assisted genetic algorithm based hyper-heuristic (hyper-TGA) for personnel scheduling problems. We recently introduced a hyper-heuristic genetic algorithm (hyper-GA) with an adaptive length chromosome which aims to evolve an ordering of low-level heuristics in order to find good quality solutions to given problems. The addition of a tabu method, the focus of this paper, extends that work. The aim of adding a tabu list to the hyper-GA is to indicate the efficiency of each gene within the chromosome. We apply the algorithm to a geographically distributed training staff and course scheduling problem and compare the computational results with our previous hyper-GA.

1. Introduction

Metaheuristic approaches have been successfully applied to a range of personnel scheduling problems, which involve the allocation of staff to timeslots and possibly locations [22]. For example, Burke et al [2] and Dowsland [12] used tabu search to solve nurse rostering problems. Aickelin and Dowsland [1] solved a nurse rostering problem in a large UK hospital utilising a genetic algorithm. Their approach gave fast and robust results which proved flexible and able to solve the large rostering problems within hospitals, coping with a range of objectives and constraints. Easton and Mansour [13] conducted an experiment on a distributed genetic algorithm to solve deterministic and stochastic labour scheduling problems. Their procedure was run in parallel, on a network of workstations. In order to maintain the search close to the feasible region, they used a combination of feasibility and penalty methods to help exploit favourable adaptations in infeasible offspring. Their approach was applied to three different sets of published test suites of labour scheduling problems. Their results were compared to those problems which had been solved by other metaheuristics and conventional heuristics. The comparison found that their results outperformed other methods, on average.

Indirect genetic algorithms have also been studied widely. For example, Terashima-Marin, Ross and Valenzuela-Rendon [21] designed an indirect GA to solve an examination timetabling problem. They design strategies which are encoded along with the parameters for guiding the search, into a 10-position array. Their chromosome represents how to construct a timetable rather than representing the timetable itself. The indirect chromosome representation can help avoid the limitation of a direct chromosome, which is known as the coordination failure between different parts of a solution when solving examination timetabling problems. Corne and Ogden [11] compared their indirect and direct GA for a Methodist preaching timetabling problem and found the former is more efficient.

Although genetic algorithms have been shown to be effective in solving a range of problems, they are usually computationally expensive and require domain knowledge to be present. Normally the chromosome of a genetic algorithm is either the solution of the given problem or a structure of the solution. This makes problem specific knowledge essential in the design of a chromosome. This results in the algorithm being difficult to reuse for different problems because of the heavy dependence of domain knowledge. In order to overcome this disadvantage, and have a reusable, robust and fast-to-implement approach, applicable to a wide range of problems and instances, we have designed genetic algorithms which use an indirect chromosome representation based on evolving a sequence of heuristics for a trainer scheduling problem, which is a type of personnel scheduling problem that can be regarded as the allocation of staff to timeslots and possibly locations [22]. The idea 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. Cowling et al described them [7] as follows: "*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*." A set of low level heuristics and a high level heuristic selector define their hyper-heuristic. They present a general framework for the hyper-heuristic to select which low-level heuristic to apply at a given choice point. The hyper-heuristic has to maintain a state to record the amount of time each low-level heuristic takes and also record the change in the evaluation function. The 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. No domain knowledge is present in the hyper-heuristic. Each low level heuristic communicates with the hyper-heuristic using a common, problem independent, interface architecture [7]. In order to improve the performance of the general framework, Cowling et al designed a *choice function*. The choice function is calculated using information from the recently called low-level heuristics: the improvement of each individual heuristic, the improvement of each pair of heuristics, and the CPU time of each heuristic. They applied the approach to a sales summit scheduling problem [8], a project presentation scheduling problem [9] and a nurse scheduling problem

[10], and found that those problems can be solved effectively.

Burke and his colleagues have designed a tabu search based hyper-heuristics to solve timetabling and rostering problems [4, 5]. In the framework of their hyper-heuristic, heuristics compete using rules based on the principles of reinforcement learning. A tabu list of heuristics is maintained which prevents certain heuristics from being chosen at certain times during the search. The basic idea of the tabu list is to prevent a poorly performing heuristic from being chosen again too soon. The approach successfully has solved a university course timetabling problem and a nurse rostering problem in a major UK hospital.

A hyper-heuristic method was also developed by Hart, Ross and Nelson [17]. They developed an evolving, heuristically driven schedule builder to solve a real-life chicken catching and transportation problem. They divided the problem into two sub-problems and solved each using a separate genetic algorithm. The result of the two genetic algorithms is a strategy for producing schedules, rather than producing a schedule itself. All the information collected from the company is summarised as 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 onto 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 large number of live chickens. They compared the performance of their approach to hill-climbing and simulated annealing, and found their approach to be superior.

Randall and Abramson [20] designed a general purpose metaheuristic based solver for combinatorial optimisation problems. They used linked list modelling to represent a problem, and then 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 [19] provided an approach that was able to learn how to select promising heuristics during the search process. The learning was based on weight adaptation. In addition, 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 [14] developed an adaptive problem solving system to select proper heuristic methods from a space of heuristics after a period of adaptation and applied it successfully to a network scheduling problem.

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 to solve a problem [6] and further improved this approach to an adaptive length chromosome hyper-GA (ALChyper-GA), which is more parameter-adaptive [15] than the fixed length version. We observe the performance 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 add a tabu method to our hyper-GA. This addition will help prevent the hyper-GA from calling those inefficient low-level heuristics in one specific chromosome, where each chromosome is a sequence of low-level heuristics which are applied, in the given order, to the problem at hand.

3. Problem Description

The problem we tackle is to schedule geographically distributed trainers to deliver geographically-distributed courses over a period of several weeks, where each course is given a priority which indicates how important it is that the course is delivered. We wish to maximise the total priority of courses which are delivered in the period, whilst minimising the amount of travel time for each trainer. We have 25 staff, 10 training centres (or locations) and 60 timeslots. Each course is to be delivered by one member of staff from a number of staff members who are competent to deliver that course. Each staff member can only work up to 60% of his/her working time (i.e. 36 timeslots). Each course is to be scheduled for 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 the limited number of rooms at each site. The starting time of each course must be 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 deliver a course. The objective function is to maximise the total priority for scheduled courses minus the total travel penalty for trainers. A mathematical model for the problem is shown in figure 1:

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

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

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

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

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

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

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

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

E*: the set of courses;* ***S****: the set of staff members;*
T*: the set of timeslots;*
L*: the set of locations;* $\mathbf{dur}_i$*: the duration of course* e_i *;*
$\mathbf{d}_{sl}$*: the distance penalty for staff member s delivering a course at location l;*
$\mathbf{w}_i$*: the priority of course* e_i*;*
$\mathbf{c}_l$*: the number of room at location l*

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

4. Low-Level Heuristics

As stated in section 2, a hyper-heuristic consists of a set of low-level heuristics and a high level selector. In our approach we have designed fourteen problem-specific low-level heuristics, which accept a current solution, and modify it locally. At each generation the hyper-GA can call upon the set of low-level heuristics and apply them in any sequence. These low-level heuristics are grouped into: *add, add-swap, add-remove* and *remove.* The first group comprise five heuristics that can be sub-divided into two sub-groups. *Add-first, add-random* and *add-best* try to add unscheduled courses 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 swap courses in the timetable in order to schedule one more course, which is selected from the unscheduled list by descending priority; *add-swap-first-improvement* and *add-swap-best-improvement* do the same but consider the unscheduled list in a random order. The third group (*add-remove-first, add-remove-random,* and *add-remove-worst*) select one course from the unscheduled course list by descending priority, if the course is in conflict with course(s) in the timetable (none of the course's possible staff members are able to work in the possible timeslots), and the course's fitness is higher than the fitness(es) of the conflicting course(s), delete the conflicting course(s) and add the unscheduled course. *Remove-first* and *remove-random* comprise the last group. These two heuristics try to remove courses 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 Adaptive Length Hyper-GA (ALChyper-GA)

5.1 Hyper-GA

Hyper-GA and the adaptive length chromosome hyper-GA (ALChyper-GA) are hyper-heuristics that use a GA to select low-level heuristics so as to produce high quality solutions for the given problem. The GA uses an indirect representation. This is a sequence of integers each of which represents a single low-level heuristic. Each individual chromosome in a GA population gives a sequence of heuristic choices that tell us which low-level heuristics to use and in what order they are applied. Figure 2 is an example of hyper-GA, where integer *0* in the chromosome represents the low-level heuristic *add-first, 1* means *add-random,* up to *13* which refers to *remove-random* 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

The adaptive length chromosome hyper-GA [15] assumes that a fixed length chromosome is not always the optimal length and it aims to evolve good combinations of low-level heuristics without explicit consideration of the chromosome length. The behaviour of a particular low-level heuristic combination could be very promising, while another low-level heuristic (or combination) could perform poorly within the same chromosome. Thus, we suggest that if we remove the poor-performing heuristics from a chromosome or inject efficient heuristics from one chromosome to another, the quality of solution can be improved. As a result, the length of chromosomes in each generation changes as genes are inserted or removed. A new crossover operator and two new mutation operators were designed to carry out the removal/injection operations. The new crossover, called *best-best crossover,* will select the best group of genes (the call of low-level heuristics by these genes that gives the most improvement to the objective function) from either selected parent 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 which gives the larges 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. An example of the best-best crossover is shown in figure 3. The grey area in parent 1 is the best group of genes in the chromosome, while the black area in parent 2 is the best group of genes in this chromosome. The two groups of genes are selected and exchanged to form child 1 and 2.

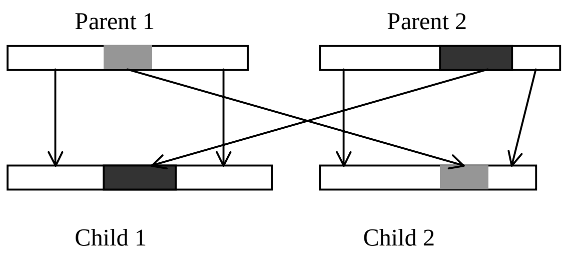

Figure 3 best-best crossover

The results from utilising this approach are reported in [15].

6. Addition of Tabu Method to Hyper-GAs

In the hyper-GA, presented in section 5, the processing involved in determining good/bad gene combination is computationally expensive. In this paper we aim to develop a cheaper and easier method to improve the efficiency of each low level heuristic call. Instead of removing or injecting genes, we aim to observe the performance of each gene and make tabu poorly performing genes for a period of time. To implement this idea, we add a tabu method to the hyper-GA.

The way we use tabu is to penalise genes which do not change the objective function. For example, if gene 2 (*add-best*) does not make any change to the objective function, it will not be allowed to call its low level heuristic during following *n* generations. Thus, we add an additional parameter to each gene, and the representation of each chromosome becomes (figure 4):

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

Figure 4. Hyper-TGA representation

The super script (or tabu tenure) of each heuristic in figure 4 shows how many generations the gene will be forbidden from calling the low level heuristic it represents. For example, gene 2^3 in figure 4 means that the gene cannot call *add-best* (heuristic 2) for 3 generations. We can see from figure 4 that the same gene may have different value for its super script (Such as 9^0 and 9^1 in figure 4). This could be because the calling of a low-level heuristic (heuristic 9) changes the value of objective function, whilst the calling the same heuristic later in the chromosome did not alter the objective function *n-1* generations ago.

To use the tabu method, we have to decide the length of the tabu list. We have experimented with lengths between 1 and 10, and found that a list length between 3 and 6 gives good results.

Pseudo-code of the tabu assisted hyper-GA (hyper-TGA) is presented as follows:

```
1. Generate an initial solution ($S$) randomly.
2. Generate 30 initial chromosomes; initialise the tabu
   tenure $t_j$ ($j$ is the position of the gene) for each gene
   to 0. Store these chromosomes into a pool
3. For each chromosome $k$ ($0 \leq k<30$),
     i.   Apply low-level heuristics to $S$ according to the
          order of the chromosome, when the script of
          corresponding gene equals to 0.
     ii.  Record the solution $S_k$ ($k$ is the position of the
          chromosome in the pool).
     iii. Record the change each applied low-level
          heuristic makes to the objective function: $C_j$.
     iv.  For each $t_j>0$, $t_j = t_j - 1$;
     v.   If $C_j=0$ then $t_j=n$. ($n$ is the length of the tabu list).
4. Compare each $S_k$ to $S$: if $S_k>S$, then $S=S_k$.
5. Apply one-point crossover and mutation (see section
   7) to current chromosomes
6. Add 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 step 3.
```

7. 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 to test the suitability of the algorithm, which describe realistic problem instances having differing degrees of difficulty [6]. 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. For more details of the problem please refer to [6].

There are 4 versions of hyper-GA, two with adaptive mutation and crossover rate and two without. 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 as follows:

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

and the crossover rate will be decreased as follows :

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

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:

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

There are two types of objective function in our algorithm. One uses total priority minus total travelling penalty resulting from applying the heuristics given by the chromosome to the best solution found so far. The formula is:

$$\sum \Pr iority \; - \sum Travelling\ \ Penalty \tag{11}$$

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 formula for this objective function is:

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

The consideration of CPU time is so that we can easily compare the efficiency of each individual sequence of low-level heuristic.

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 is adapted using (7)-(10).

PPPN, PPPA, FTPN, and FTPA are simply mnemonic names we give to each of the four version of hyper-GA.

The comparison of these four versions test the robustness of hyper-GA under a range of conditions.

Thirty individuals are generated for the initial population by randomly selecting integers ranged from 0 to 13 for each gene of the chromosome. The length of each chromosome is 14. After empirical testing over a range of parameter rates [6], we use 0.6 for the crossover rate (one-point crossover is used in experiments), 0.1 for the mutation rate (the mutation randomly selects some positions in one chromosome and mutates integers at these positions to other values ranging from 0 to 13), a population size of 30 and we run for 200 generations (100 generations gives equally good results, but we use 200 to see the further change of low-level heuristics' distribution).

We compare each of the four versions of hyper-TGA over five problem instances (each result is the average of 5 runs). The results of the hyper-TGA tests on the 5 data sets is shown in table 1. The results of our previous work: application of hyper-GA and ALChyper-GA to the same data is also shown. We also compare with the results of a genetic algorithm and a memetic algorithm. The upper bound is calculated by solving a relaxed knapsack problem [18] where we ignore travel penalties. The length of the tabu list is 5 for hyper-TGA's results in table 3. (If the tabu list length is too short, those poor-performing low-level heuristics are not blocked for long enough, and the efficiency of the algorithm is not improved. If the length is too long, only a few heuristics are available in each generation, and the effectiveness of the algorithm is reduced.) We can see again, from the table that most of our new algorithms are superior to previous results.

Results from our guided-operator hyper-GA [16] are also included in table 1. Although the guided-operator hyper-GA produces superior results, the hyper-TGA is competitive whilst using about 20% less processing time. 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. In addition, the hyper-TGA is a lot easier to implement as the identification of best/worst genes is not easy to implement and it is open to a number of interpretations. Thus, we can use the hyper-TGA to find a quick promising solution, or use the guided-operator hyper-GA to achieve better quality solutions.

Heuristics	*Basic data*	*Very few staff*	*Few staff 1*	*Few staff 2*	*Non -restricted*
Upper bound (priority)	2261	2179	2124	2244	2179
GA	1796/1628	1633/1629	1589/1641	1706/1721	1644/1699
MA	1832/2064	1678/2054	1617/2129	1769/2254	1698/2133
Hyper-GA PPPN	1959/1456	1780/1387	1749/1404	1858/1496	1742/1422
Hyper-GA PPPA	1939/1448	1754/1461	1712/1306	1854/1475	1814/1571
Hyper-GA FTPN	1943/1411	1770/1437	1673/1436	1803/1422	1774/1434
Hyper-GA FTPA	1951/1420	1731/1424	1738/1436	1769/1427	1770/1419
ALCHyper-GA PPPN	1961/1357	1788/1250	1816/1163	1831/1591	1822/1437
ALCHyper-GA PPPA	1933/1638	1757/1644	1795/1325	1862/1506	1804/1638
ALCHyper-GA FTPN	1949/1450	1780/1365	1781/1277	1821/1638	1813/1488
ALCHyper-GA FTPA	1954/1526	1764/1496	1766/1364	1799/1583	1799/1419
Hyper-TGA PPPN	**1966/972**	**1789/911**	**1820/834**	**1866/1004**	**1824/941**
Hyper-TGA PPPA	1959/958	1782/931	1804/864	1852/996	1809/970
Hyper-TGA FTPN	1960/963	1784/933	1799/856	1852/1012	1814/982
Hyper-TGA FTPA	1965/985	1782/942	1811/892	1857/997	1804/930
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

Table 1. Comparison of Guided Operator Hyper-GA with Other Algorithms (Objective (maximise)/Time)

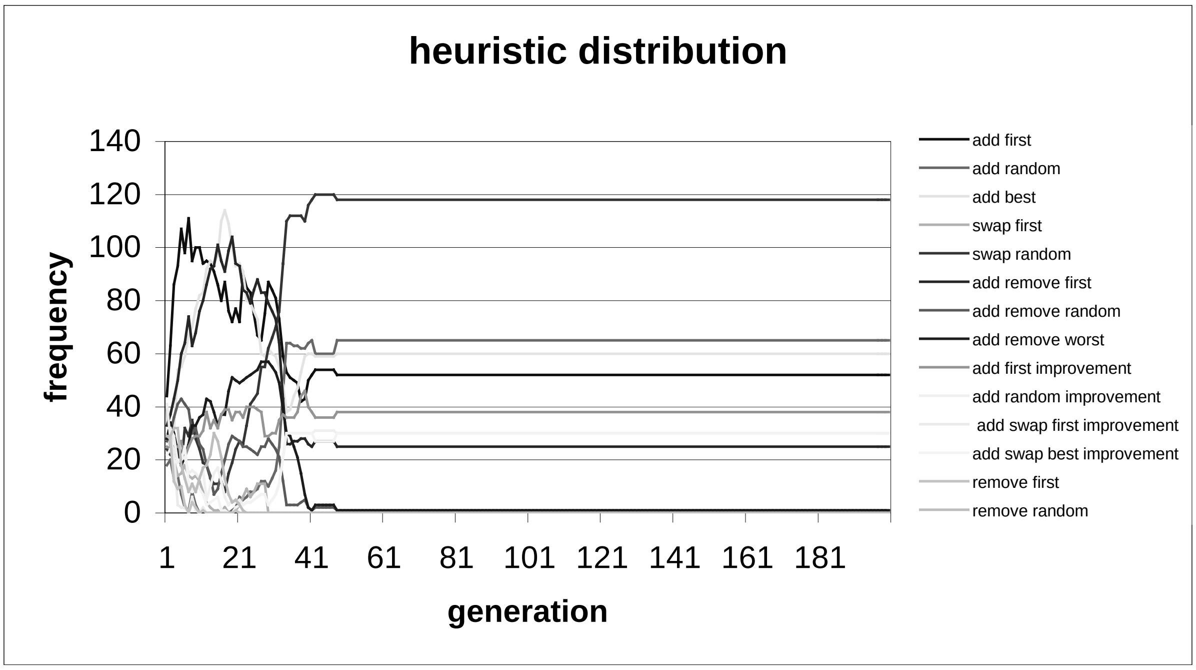

Figure 5. Distribution of low-level heuristic for PPPN

Figure 5 presents the frequency of hyper-TGA calling each low-level heuristic. We can see from the figure that the frequency changes randomly at the beginning of the evolution. All calls settle at generation 47 when the search reaches its best value of objective function (1966).

Hyper-TGA calls some low-level heuristics often during the whole evolution, such as *add-best* (gene 2) in the figure. This is because these low-level heuristics work actively to make changes to the objective function. Thus, calls to them are rarely forbidden by the tabu method in the algorithm. Some low-level heuristics' call frequencies change from very low to high. For example, *add-random* (gene 1) and *swap-random* (gene 4). The reason for this phenomenon is the effect of these heuristics alters from being unable to change the objective function to providing changes. The call to some heuristics, such as *add-first-improvement* (gene 8), remains low during the whole process. The reluctance to make change to the objective function forces them be tabued for most of the time during the evolution.

8. Conclusion

We introduced a tabu method assisted hyper-GA, which is a further improvement, with respect to computational time, of our previous work. It is a promising approach for solving personnel scheduling problems and other optimisation problems. The use of a tabu method to forbid poorly performing genes appears to help the search. Making calls to some heuristics also helps to reduce the CPU time. We have tried different lengths of tabu list, and find that a list which is too long or too short provides an ineffective search. Therefore, we decide to use 5 as the length of the list in this work, and find it improves the objective function and reduces the CPU time. 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] 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., 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] Burke, E.K., Kendall, G., and Soubeiga, E., A Tabu-Search Hyperheuristic for Timetabling and Rostering, 2003, to appear in Journal of Heuristics.

[5] Burke, E.K., Soubeiga, E., Scheduling nurses using a tabu-search hyperheuristic, 2003, Proceedings of the 1st M ultidisciplinary International Conference on Scheduling: Theory and Applications, MISTA 2003, Nottingham, UK, pp. 197-218.

[6] 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.

[7] Cowling, P.I., Kendall, G., Soubeiga, E., Hyperheuristic Approach to Scheduling a Sales Summit, 2001, Selected papers of *Proceedings of the Third International Conference of Practice And Theory of Automated Timetabling*, Springer LNCS vol 2079, pp. 176-190.

[8] 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

[9] 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.

[10] 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.

[11] 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*, LNCS:1408, pp. 142-155.

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

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

[14] 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.

[15] Han, L., Kendall, G., and Cowling, P., An adaptive length chromosome hyperheuristic genetic algorithm for a trainer scheduling problem, Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution And Learning, (SEAL'02), Orchid Country Club, Singapore, 18-22 Nov 2002, pp 267-271.

[16] Han, L., Kendall, G., Guided Operators for a Hyper-Heuristic GA, 2003, accepted to AI'03, Dec 2003, Perth, Australia.

[17] 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(6) pp. 61-80.

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

[19] 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.

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

[21] 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.

[22] 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.