An Investigation and Extension of a Hyper-heuristic Framework

Prapa Rattadilok
Assumption University, Thailand
E-mail: prapa.rattadilok@gmail.com

Keywords: choice function, dynamic configuration, hierarchical controller, hyper-heuristic framework, low-level heuristics, timetabling.

Received: October 8, 2009

Three modifications to the framework within which hyper-heuristic approaches operate are presented. The first modification automates a self learning mechanism for updating the values of parameters in the choice function used by the controller. Second, a procedure for dynamically configuring a range of low-level heuristics is described. Third, in order to effectively use this range of low-level heuristics the controller is redesigned to form a hierarchy of sub-controllers. The second and third modifications improve the inflexibility associated with having a limited number of low-level heuristics available to the controller. Experiments are used to investigate features of the hyper-heuristic framework and the three modifications including comparisons with previously published results.

Povzetek: Opisane so tri modifikacije hiper-hevrističnih pristopov.

1 Introduction

As the complexity of optimisation problems increases methods which guarantee optimal solutions place excessive demands on computation time and computer resources. Alternative approaches have been developed including: heuristics, meta-heuristics, combinations of meta-heuristics referred to as hybrids, and more recently hyper-heuristics. Generally, these approaches do not guarantee optimal solutions but instead provide solutions of acceptable quality obtained with acceptable demands on algorithm development, tuning time, computation time, and computer resources. Surveys and comparisons among these approaches are presented in [1-9].

Heuristic approaches use rules derived from experience or intuition as opposed to those derived from mathematical formulations and they produce reasonable computational performance with conceptual simplicity. Problem specific knowledge is applied at the heuristic design phase and increases effectiveness but limits reusability for problems in other domains. Heuristic approaches have been applied successfully to a variety of specific problems including: resource investment [10]; resource usage [11]; project finance scheduling [12]; flow-shop scheduling [13]; graph colouring [14]; and train pathing [15]. Meta-heuristic approaches employ artificial intelligence methods and are different from simple heuristics in the manner in which the problem is modelled by attempting to prescribe more generic structures. Simulated annealing [16], tabu search [17], genetic algorithms [18], ant colony [19] and particle swarm optimisation [20], hill climbing and local search [21], and differential evolution [22] are well known meta-heuristic approaches. Interest in meta-heuristics has generated the development of hybrid approaches [8] and recent significant advances have combined meta-heuristics with other problem solving paradigms and improved their use in important application areas [23]. However, due to the evolutionary nature of meta-heuristic approaches the computation time may be unpredictable and there is often a need for a training period in order to tune the approach to the problem.

The aim of hyper-heuristic approaches is to be able to use the same procedures within and across problem domains without the need for extensive change to the basic components thus handling classes of problems rather than addressing one type of problem [24-28]. While most applications of meta-heuristics explore a search space of problem solutions hyper-heuristics explore a search space of low-level heuristics in order to select and apply an appropriate low-level heuristic. The framework in which hyper-heuristic approaches operate is presented in Figure 1 where at each stage of the search the controller uses information about the past performance of the low-level heuristics in order to select one to be used in the next stage. The selection is often made using a choice function and this process continues until a stopping condition is satisfied and the best solution is determined based on the value of the cost function.

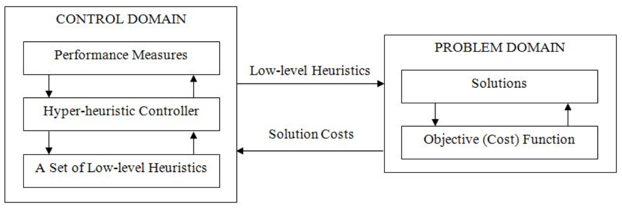

Figure 1: Hyper-heuristic framework.

The set of low level heuristics used by the controller are: pre-designed; limited in number; often involve add, drop, and swap operations; and remain the same throughout the search. They are problem specific and have limited reusability [29, 30]. Problems often include soft constraints, which may be violated, and hard constraints, which must not be violated, and these are usually assigned low and high positive weights, respectively, by the user. For any solution the value of the cost function is the sum of the weights associated with the constraints which are violated and for a feasible solution all of the hard constraints are satisfied.

The purpose of this article is to investigate three modifications to the hyper-heuristic framework The first modification (section 2.1) introduces a self learning mechanism for updating the values of choice function parameters so that the selection of low-level heuristics may be intensified or diversified appropriately. The second (section 2.2) introduces a procedure for dynamically configuring low-level heuristics in order to make a range of low-level heuristics available to the controller. In order to select effectively from this range of low-level heuristics the controller is redesigned (section 2.3) to form a hierarchy of sub-controllers each using the choice function described in section 2.1 and the dynamic configuration procedure described in section 2.2. Section 3 presents the results of experiments related to each of the three modifications including comparisons with previously published results. Section 4 discusses the results and draws conclusions. An Appendix is used to present details associated with the updating of choice function parameters.

2 Modifications to the hyper-heuristic framework

This section describes the three modifications to the hyper-heuristic framework in Figure 1.

2.1 Modifications to the choice function

The choice function proposed by Cowling et al. [30] and Soubeiga [31] is modified to allow the values of parameters to be updated automatically independently of any problem specific knowledge. The procedures work with complete rather than partial solutions and there is no need for an initial training period. The adjustment of parameters allows the search procedure to be intensified or diversified thus enhancing its applicability within and between problem domains. In cost minimisation problems the choice function selects a low-level heuristic by assessing the efficiency of the past performance of each of the low-level heuristics in decreasing the value of the cost function. Some may consistently decrease the value of the cost function and selection may be intensified on them. However, this may result in convergence to a local rather than global optimum and in such cases the choice function needs to select a low-level heuristic that diversifies the search to other parts of the solution space. Thus a suitable choice function should include factors which intensify or diversify the search appropriately. If at each stage of the search the low-level heuristics $H_1, H_2, H_3, \ldots, H_m$ are available to the choice function (F) then a value of F is computed for each low-level heuristic using,

$$F(H_j) = f_1(H_j) + f_2(H_j) + f_3(H_j),$$
$$\text{for } j = 1, 2, 3, \cdots, m. \qquad (1)$$

The three factors in (1) represent: the past performance of the low-level heuristic (f_1); the paired past performance of the low-level heuristic (f_2); and the time since the low-level heuristic was last selected (f_3). The first two factors are associated with intensifying the search while the last is associated with diversifying the search. In the Appendix section A1 each of the three factors is defined and the procedures for modifying parameters are presented in section A2. At the start of the search a solution is determined and one of the low-level heuristics is selected at random and applied to that solution. Information required in equations (A1), (A2), (A3), and (1) is updated and stored. The controller uses this information in (1) to determine the low-level heuristic with the largest value for F and then using the procedures to adjust parameters this low-level heuristic or a different one is determined and used in the next iteration of the search. Subsequent iterations are conducted in the same manner until a stopping rule is satisfied and then the best solution among all of the solutions is selected as the final solution. The process is stochastic and a transition from one solution to another in the solution space is made using information about all of the previous transitions. Consequently, the process is not a Markov process and probabilistic equilibrium among the solutions is not attained [32]. Unless stated otherwise the choice function in (1) is used in all of the subsequent modifications and experiments.

2.2 Dynamically configured low-level heuristics

Swap-based low-level heuristics are used often and instead of generating a solution from scratch these low-level heuristics perform an exchange of attribute(s) between at least two swap candidates. For example, in a university timetabling problem an exchange may include swapping the days on which 2 classes are scheduled. Such low-level heuristics normally use problem specific knowledge in their design and applying them to different types of problems without any modification is usually infeasible. Different swap-based heuristics may be designed by choosing different configuration options at each of a set of configuration decision points. Examples of configuration options that may be selected at 4 commonly used configuration decision points are shown in Table 1.

Configuration Decision Points 1: The Number of Swap Candidates (λ)
Example Configuration Options:

Configuration Decision Points 1: The Number of Swap Candidates (λ)
1. Two swap candidates? 2. More than two swap candidates? *Comments: Determines the number of swap candidates involved in any trial swap process.*
Configuration Decision Point 2: Formation of λ Swap Candidate Sets
Example Configuration Options: 1. Non-violated assignments? 2. Violated assignments? *Comments: Specifies the swap candidate for each of the candidate sets.*
Configuration Decision Point 3: Ordering Candidates in the λ Swap Candidate Sets
Example Configuration Options: 1. Slot number? 2. Ascending cost? *Comments: Specifies the order in which the swap candidate from each candidate set enters the trial swap process.*
Configuration Decision Point 4: Acceptance Criteria
Example Configuration Options: 1. Best Solution? *Comments: The trial swap process terminates when a solution satisfies the acceptance criteria and then the solution is returned to the controller.*

Table 1: An example of configuration options associated with 4 configuration decision points.

From Table 1 it is seen that the number of configuration options at decision points 2 and 3 depends on the number of swap candidates (λ) chosen at decision point 1 and two or three swap candidates are commonly used. When forming swap candidate sets at decision point 2 the swap candidates may be shared among the sets formed.

The restrictions of using a fixed and limited number of problem specific low-level heuristics may be addressed by dynamically configuring swap-based low-level heuristics and using a hierarchical design for the controller. Dynamic configuration is discussed next and the design of a hierarchical controller is presented in section 2.3.

Figure 2(a) elaborates on elements of the framework in Figure 1 and represents a non-dynamic approach where the controller uses the choice function to select a low-level heuristic from a fixed set of usually no more than 10 swap-based low-level heuristics. Figure 2(b) presents the framework for an approach where the swap-based low-level heuristics are dynamically configured by the controller which selects configuration options at decision points as illustrated in Table 1 using a choice function of the same form as that used by the controller Figure 2(a) but with low-level heuristics replaced by configuration options. Dynamically configured low-level heuristics are generated and applied to the current solution and performance measures for configurations of these low-level heuristics are accumulated.

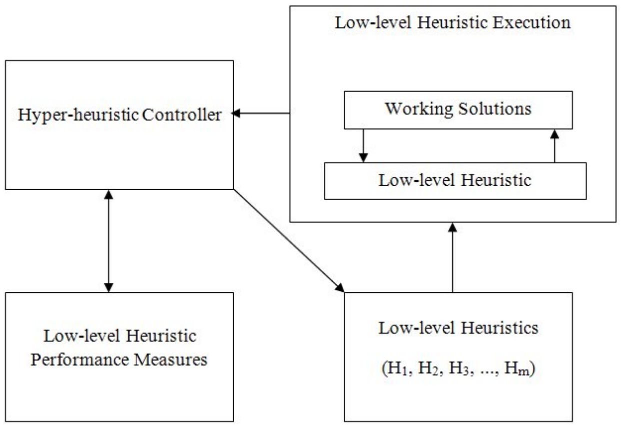
Low-level Heuristic Execution
Hyper-heuristic Controller
Working Solutions
Low-level Heuristic
Low-level Heuristic Performance Measures
Low-level Heuristics
(H_1, H_2, H_3, …, H_m)

Figure 2 (a): Non-dynamic approach.

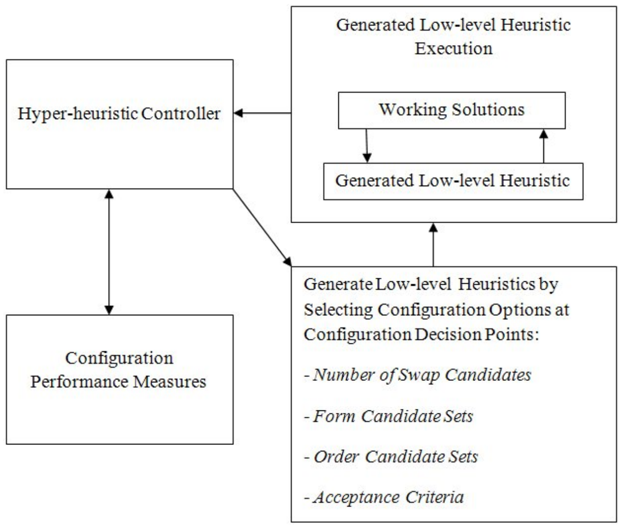
Generated Low-level Heuristic Execution
Hyper-heuristic Controller
Working Solutions
Generated Low-level Heuristic
Configuration Performance Measures
Generate Low-level Heuristics by Selecting Configuration Options at Configuration Decision Points:
- Number of Swap Candidates
- Form Candidate Sets
- Order Candidate Sets
- Acceptance Criteria

Figure 2 (b): Dynamically configured approach.

The use of a single choice function in the dynamic approach limits the total number of configuration options that the controller can work with effectively. Consequently, in order to improve the effectiveness of the dynamic approach the design of the controller needs to be reconsidered.

2.3 A hierarchical controller design

A new hierarchical design which operates in the controller component in Figure 2(b) is shown in Figure 3.

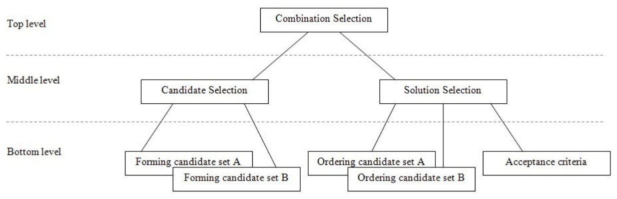

Figure 3: Hierarchical controller design.

Each component in Figure 3 is regarded as a sub-hyper-heuristic controller each of which uses a choice function where the low-level heuristic H_j now represents a configuration option or a combination of configuration options depending on the level at which the sub-controller is operating. Information about the performance of configuration options or combinations of them is used to choose configuration options at different configuration decision points in the same manner as low-level heuristics were selected in the non-dynamic situation. There are 3 levels of sub-controllers in the hierarchy. The number of sub-controllers at the bottom level depends on the number of swap candidate sets formed (λ in Table 1). Figure 3 shows the case where λ = 2 and five sub-controllers are used: two for forming swap candidate sets; two for ordering swap candidate sets; and one for acceptance criteria. The middle level sub-controller chooses combinations of configuration options based on their performance in trails using the sets of configuration options generated at the bottom level. These configuration options are combined to form a low-level heuristic at the top level which is used in the next stage of the search.

3 Experiments

Published data sets and results for two different sets of problems are used in the experiments: international university timetabling competition problems (www.idsia.ch/Files/ttcomp2002/); and transportation services timetabling problems [33]. In order to allow comparisons experiments are designed to conform to the conditions associated with the published experimental results.

3.1 Experiments 1: The choice function

Two methods are investigated for generating an initial solution for a university timetabling problem: a random approach, which assigns random events (classes) to random slots (day, time, and room); and a greedy algorithm, which assigns an event to its best slot. On average across 5 experimental runs there are 1000 hard constraint violations in a randomly generated initial solution but only 200 for a greedily assigned solution. Consequently, greedy assignment is proposed for generating the initial solution used with a choice function.

Both of these methods are examined further by considering the average percentage of improvement

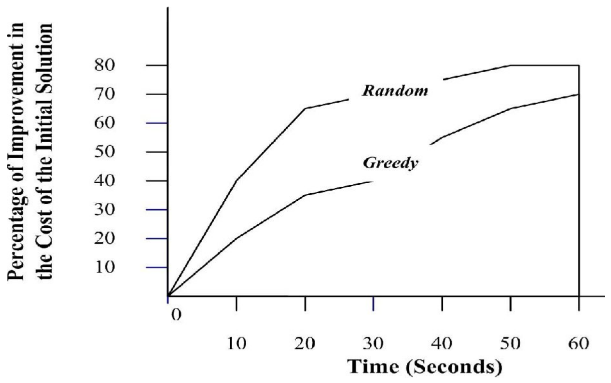

Figure 4: The percentage of improvement in the cost of the initial solution during the first minute.

in the cost of the initial solution if the search is allowed to continue for 1 minute and the results are shown in Figure 4. Table 2 shows the average number of hard and soft constraint violations for both methods at the end of 5 and 7 minutes.

Time Limit: 5 Minutes		
Methods	**Number of HCV**	**Number of SCV**
Random	8.7	1152.9
Greedy	0	921.6
Time Limit: 7 Minutes		
Methods	**Number of HCV**	**Number of SCV**
Random	0	767.8
Greedy	0	598.4

Table 2: The number of hard and soft constraint violations (HCV and SCV respectively) after 5 and 7 minutes.

From Figure 4 and Table 2 it is seen that beyond the initial solution the greedy assignment method continues to produce better results than the random method. In particular, the patterns in Figure 4 demonstrate the more general result that as the number of constraint violations decreases it becomes more difficult to reduce the number of constraint violations.

The results in Table 3 show the average costs of solutions across 10 experiment runs on each of 17 university timetabling problems using the choice function with and without automatic parameter modification. The values for the parameters α , β , and δ in equations (A1), (A2), and (A3) are set randomly at 0.7, 0.5 and 0.1, respectively, at the start of the search and the same set of 7 low-level heuristics is used for all of the problems.

Problems	**Without Parameter Modification**	**With Parameter Modification**
1	118.2	89.5
2	104.1	77
3	117.5	80.8
4	234	175.9
5	199.2	139.5
6	255	188.5
7	120.8	84.4

Problems	Without Parameter Modification	With Parameter Modification
8	103.6	71.9
9	124.3	89
10	122.2	85.2
11	155.7	107.6
12	185.1	127.3
13	96.8	75.1
14	255.9	186
15	91	64.2
16	195.1	188.8
17	141.7	99.3

Table 3: The cost of solutions with and without parameter modification in the choice function.

From Table 3 it is seen that modification of the parameters reduces the average cost of the solutions for every problem. Although not shown here the same outcome occurs when the initial values of the parameters vary. From these experiments it is evident that automatic parameter modification is a useful enhancement to the choice function.

The next experiments examine the effect of varying the number of low-level heuristics available to the controller. In order to ensure that the results are not affected by the quality of the low-level heuristics used in each experiment all low-level heuristics are idle except for one which performs a simple swap on the solution. The idle heuristics may be selected by the controller and vary in terms of the time they take to execute but they have no effect on the solutions. Two problems are used from the university timetabling competition (U1, U2) and the transportation services timetabling (T1, T2) data sets. The number of low-level heuristics varies from 5 to 40 and in each case results are averaged across 5 experimental runs. The entries in Table 4 represent the percentage of calls received by the non-idle low-level heuristic above the percentage expected if it is called at random. For example, in problem U1 with 20 low-level heuristics the non-idle low-level heuristic received on average 27 percent of all of the calls which is 22 percent above the 5 percent expected if 20 low-level heuristics are called at random.

Problems	Number of Low-level Heuristics							
	5	10	15	20	25	30	35	40
Problem U1	56	52	30	22	12	11	10	10
Problem U2	51	49	34	25	20	15	11	11
Problem T1	41	44	39	33	25	20	15	14
Problem T2	38	42	38	31	27	23	19	18

Table 4: The effects of increasing the number of low-level heuristics.

For each problem in Table 4 it is seen that as the number of low-level heuristics increases the idle low-level heuristics, which contribute nothing to the quality of the solution, are being called increasingly and the selection of low-level heuristics becomes almost random when there are a large number of low-level heuristics.

Figure 5 illustrates the effect over time on the cost of the solution of increasing the number of low-level heuristics.

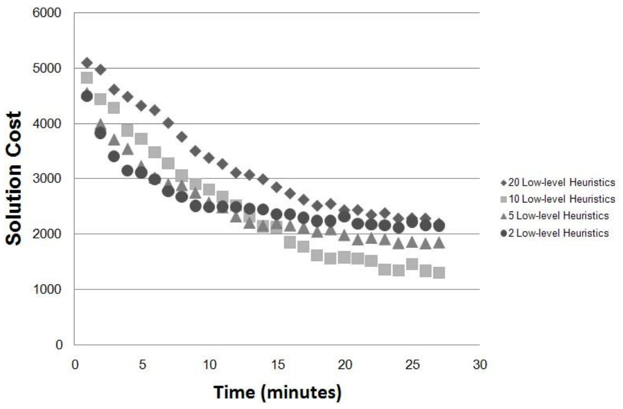

Figure 5: The number of low-level heuristics and the solution cost.

In Figure 5 it is seen that as the number of low-level heuristics increases it takes longer to establish good performance measures for them. This is evidenced by the flatter curve for 20 low-level heuristics compared to the curves for 5 or 10. For only 2 low-level heuristics performance measures are established earlier than in the other cases but there is much less opportunity to diversify the search and this makes it more difficult to escape from a local optimum. Based on the results in Table 3 and Figure 5 it is appropriate to recommended that the number of low-level heuristics should not be more than 10 or less than 5.

Figure 6 compares the average percentage of improvement across 5 experimental runs in the cost of the initial solution using the choice function approach, greedy selection, and random selection where for the greedy selection method low-level heuristics are selected and applied until no improvement is obtained and then a new low-level heuristic is selected. A university timetabling competition problem data set is used.

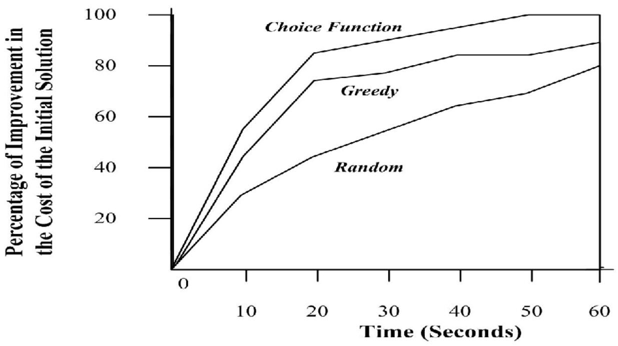

Figure 6: Improvement in the cost of the initial solution.

From Figure 6 it is seen that the choice function method consistently improves the initial solution more than either of the other methods. It is unlikely that the random approach will obtain similar quality solutions as the choice function even if more time is allowed. Random selection has a smoother improvement curve than greedy selection which has sharp improvement which flattens out quickly. The choice function has an even sharper improvement. On average it reduces almost 90 percent of the cost of the initial solution within the first 20 seconds and reduces it by almost 100 percent after 50 seconds.

3.2 Experiments 2: Dynamic configuration of low-level heuristics

Using problems from the university timetabling competition these experiments examine different configurations in terms of: their ability to improve, worsen, or not change solution costs; their effect on different constraints; and their performance on different problems. Table 5 lists the configuration options used for forming candidate sets and acceptance criteria where the swap candidates in both sets are ordered based on their cost.

Index	Forming Candidate Configuration Options
0	All slots
1	Occupied slots
2	Empty slots
3	Feasible slots
4	Infeasible slots
5	Constraint violated slots (specific constraint)
6	Non violated slots
Index	**Acceptance Criteria**
0	First cost improvement
1	First feasibility improvement
2	Minimum cost
3	Maximum feasibility
4	Minimum cost improvement
5	Maximum feasibility improvement

Table 5: Configuration options for forming candidate sets and acceptance criteria.

In Table 5 the forming candidate configuration options specify which slots in the solution will be selected and used as the swap candidates. The slots can be divided into occupied and empty slots. Empty slots do not have events assigned to them and are therefore always feasible. Occupied slots can be feasible or infeasible. Infeasible slots are those assignments that violate hard constraints. These occupied slots can be further divided based on the number of violations for specific types of constraints and can also be assignments that have no constraint violations. At the end of each trial swap the acceptance criteria are checked. The acceptance criteria used improve either the cost or feasibility of the solution. The acceptance criteria can be specified to accept the first solution that satisfies one or both of these objectives. The minimum cost and maximum feasibility acceptance criteria include the selection of solutions that decrease the value of the cost function. Using the 7 configuration options and the ordering of swap candidates based on cost for each of the 2 swap candidate sets and the 6 acceptance criteria options a total of 294 (i.e. 7x1x7x1x6) low-level heuristics are generated and more are generated if there is more than 1 option for ordering swap candidates in the 2 candidate sets.

The configurations derived from Table 5 are categorised according to their performance. If the largest percentage of all of the calls made on a configuration produce an improved solution then the configuration is categorised as 'improving' and similarly configurations may be categorised as 'unchanging' or 'worsening'. Table 6 shows the results for the first problem in the university timetabling competition when random selection is used to dynamically configure the configuration options. The top 3 configurations in each of the 3 categories are shown. The meaning of the entries in Table 6 is explained using the example of the best configuration in the 'improving' category (i.e. 2-5-2 (90.95%)). From Table 5 this means that the configuration is generated by choosing 'Empty slots' as the candidates for the first candidate set, 'Constraint violated slots' as the candidates for the second candidate set, 'Minimum cost' as the acceptance criteria, and on average 90.95 percent of the times when it is called this configuration improves the solution.

Unchanging	Worsening	Improving
2-3-3 (93.13%)	5-0-2 (62.12%)	2-5-2 (90.95%)
2-6-3 (91.27%)	5-0-3 (60.83%)	2-0-5 (90.57%)
3-6-3 (89.95%)	5-6-3 (59.77%)	2-5-5 (90.47%)

Table 6: Top 3 configurations for performance categories.

From Table 6 it is seen that some configurations do not lead to improvement in the cost but they may be used for diversifying the search. For example, using configurations that are associated with worsening costs would lead to diversification and this is often desirable. Configurations with high chances of improving costs are appropriate when search intensification is desired especially near the end of a search.

Table 7 presents a different view of the configurations where the columns represent 5 different types of constraints: 'Room' (an event must be assigned to a room that has all of the resources needed); 'Student' (a student cannot attend more than one event at any one time); 'One Per Day' (a student attends only one event per day); 'More Than Two' (a student attends more than 2 classes consecutively); and 'Late' (a student attends an event at the last period of the day). The configurations are represented in the same manner as in Table 6 but the percentage now indicates the average amount of improvement they produced in the cost of the initial solution each time they were used. The best 3 configurations are shown for each constraint.

Types of Constraints				
Room	**Student**	**One Per Day**	**More Than Two**	**Late**
2-4-2 (0.96%)	0-4-2 (24.36%)	4-2-2 (4.09%)	2-5-4 (1.50%)	2-1-2 (0.36%)
2-5-2	5-4-2	5-6-2	2-5-2	2-5-4

Types of Constraints				
Room	**Student**	**One Per Day**	**More Than Two**	**Late**
(0.96%)	(18.61%)	(3.45%)	(1.43%)	(0.28%)
0-4-2 (0.92%)	1-4-2 (17.31%)	4-3-2 (3.05%)	2-1-2 (1.28%)	2-5-2 (0.25%)

Table 7: The best configurations for different types of constraints.

From Table 7 it is seen that configuration 2-5-2 benefits 3 constraints while configurations 2-5-4, 2-1-2, and 0-4-2 all benefit 2 constraints. When 'Empty slot' is used to form one of the candidate sets the 'Minimum cost' acceptance criteria is commonly used. The 'Student' constraint violations are removed at a high rate each time, while 'Late' constraint violations are removed at a much lower rate. If the 'Room' and 'Student' constraints are hard constraints and the other three are soft constraints then configuration 0-4-2 appears to perform well on those hard constraints.

The first 3 problems in the international timetabling competition are used to obtain the results in Table 8. The entries in the table have the same meaning as those in Table 7.

Problem 1	**Problem 2**	**Problem 3**
0-4-4 (0.99%)	0-4-2 (1.53%)	2-1-3 (0.80%)
1-4-2 (0.94%)	1-4-2 (1.32%)	2-5-3 (0.74%)
2-4-5 (0.94%)	5-4-2 (1.21%)	2-4-5 (0.73%)

Table 8: Best configurations for 3 timetabling problems.

From Table 8 it is seen that the best configuration varies from one problem to another but using the 'Infeasible slots' configuration option for forming the second candidate set is beneficial across the 3 problems and combining it with the 'Empty slots' option for the first candidate set is beneficial for problems 1 and 3. The configuration 1-4-2 works well in problems 1 and 2 but with different average improvements and the 'Minimum cost' acceptance criteria is dominant for problem 2 and useful in problem 1 but not problem 3. Even when a random configuration is used consistent performance measures are observed when some configuration options are combined and these performance measures may be used to influence the configuration by the controller in much the same way that a human heuristic designer applies their past experience in selecting suitable low-level heuristics for a problem. If configuration options are good when combined then it is possible that there is a positive relationship between the options and making modifications to multiple configuration points simultaneously may assist the controller in making configurations.

3.3 Experiments 3: Using a hierarchical design for the controller

These experiments compare the non-dynamic approach in Figure 2(a), which uses a single choice function and a fixed set of low-level heuristics, with the dynamic approach in Figure 2(b), which incorporates the dynamic configuration of low-level heuristics and a hierarchical design for the controller as described in Figure 3. Data sets from the university timetabling competition and the transportation services timetabling problems are used in the experiments.

The non-dynamic approach uses the following 8 low-level heuristics: $\boldsymbol{H_1}$: Swap the highest cost feasible assignment with every other assignment (in ascending order based on their cost), select the best quality solution; $\boldsymbol{H_2}$: Same as H_1 but select the first improving quality solution; $\boldsymbol{H_3}$: Same as H_1 but the candidate assignments are ordered randomly; $\boldsymbol{H_4}$: Same as H_3 but select the first improving quality solution; $\boldsymbol{H_5}$: Swap the highest cost infeasible assignment with every other assignment (in ascending order based on their cost), select the best quality solution; $\boldsymbol{H_6}$: Same as H_5 but select the first improving quality solution; $\boldsymbol{H_7}$: Same as H_5 but the candidate assignments are ordered randomly; and $\boldsymbol{H_8}$: Same as H_7 but select the first improving quality solution.

For a fair comparison, the configuration options for the dynamic approach are limited to those that will generate low-level heuristics equivalent to the non-dynamic set. The configuration options for the 4 configuration points are: ***The Number of Swap Candidates (λ)***: 2; ***Forming λ Swap Candidate Sets***: Highest cost feasible assignment, Highest cost infeasible assignment; ***Ordering λ Swap Candidate Sets***: Ascending cost based, Random; and ***Acceptance Criteria***: Best quality, First improving quality. To ensure the same number of low-level heuristics as in the non-dynamic set, the ***Forming*** options selects a swap candidate for the first candidate set and the second candidate set contains all other assignments. Because there is only one assignment in the first candidate set the ***Ordering*** options are only used to order the candidates in the second candidate set.

Table 9 compares the 4 best results from the university timetabling competition (www.idsia.ch/Files/ttcomp2002/results.htm) with the results obtained using the non-dynamic and dynamic approaches where the solution costs are the averages from 10 experimental runs. It is noted that the results for the competition were obtained using algorithms specifically designed for these problems while the dynamic and non-dynamic approaches use generic configuration options and low-level heuristics, respectively.

Problem Data Set	**Approach**					
	Problem Specific Algorithms				**Hyper-heuristics**	
	1	**2**	**3**	**4**	**Non-Dynamic**	**Dynamic**
1	45	61	85	63	**80.1**	**79.5***
2	25	39	42	46	73	73.2
3	65	77	84	96	**77.8**	**77.6***
4	115	160	119	166	174.3	175.7
5	102	161	77	203	289.9	292
6	13	42	6	92	131.2	133.5

Problem Data Set	Approach					
	Problem Specific Algorithms				Hyper-heuristics	
	1	2	3	4	Non-Dynamic	Dynamic
7	44	52	12	118	180.2	170.9*
8	29	54	32	66	82.1	82.2
9	17	50	184	51	**68.9**	**69.6**
10	61	72	90	81	**83.3**	**83.3***
11	44	53	73	65	79.9	81.2
12	107	110	79	119	120.2	**118.1***
13	78	109	91	160	**101.2**	**103.5**
14	52	93	36	197	255.7	253.4*
15	24	62	27	114	119	123.6
16	22	34	300	38	**64.2**	**64.8**
17	86	114	79	212	**169.9**	**170.5**
18	31	38	39	40	61.3	61.3*
19	44	128	86	185	186.1	186.2
20	7	26	0	17	93.7	94.7

Table 9: University timetabling solution costs.

In Table 9 the highlighted values represent the 40 percent of cases where one or both of the hyper-heuristic approaches achieved a lower cost than at least one of the best 4 competition results and this is encouraging considering the problem specific nature of the algorithms used in the competition. The results for the hyper-heuristic approaches are very similar but the dynamic approach achieved the same or better results to the non-dynamic approach in 35 percent of cases (marked *).

Table 10 compares the same hyper-heuristic approaches with the problem specific algorithm BOOST [32] for transportation services timetabling problems. The results are the average solution costs from 10 experimental runs.

Problem Data Set	Approach		
	Problem Specific Algorithm	Hyper-Heuristics	
	BOOST [32]	Non-Dynamic	Dynamic
1	492	**492**	**492***
2	1376	**1376**	**1376***
3	1678	**1678**	**1678***
4	1641	1761	1756*
5	1396	**1396**	**1396***
6	1389	1421	1434
7	1465	1606	1604*
8	1858	2045	2044*
9	3409	**3409**	3411
10	3502	**3502**	3533
11	14919	15598	15632
12	6028	6268	6272
13	21963	23987	24132
14	12510	14498	14498*

Table 10: Transportation services timetabling solution costs.

In Table 10 the highlighted values represent the 43 percent of cases where one or both of the hyper-heuristic approaches achieved the same cost as BOOST which is specifically designed for the transportation problems while the hyper-heuristic approaches are using generic configurations and low-level heuristics. The results for the hyper-heuristic approaches are very similar but the dynamic approach achieved the same or better results compared to the non-dynamic approach in 57 percent of cases (marked *).

From the results in Tables 9 and 10 it is seen that the dynamic approach has performed well across 2 different types of problems using a generic set of configuration options. It was not expected that the hyper-heuristic approaches would achieve better results than algorithms specifically designed for these problems but their performance is acceptable and compares favourably with the specific algorithms. In addition, for the dynamic approach increasing the number of configuration options increases the possible number of configurations. Therefore, a longer time is required for the controller to establish reliable performance measures and it is expected that the dynamic approach may obtain equally good solutions in all cases to the non-dynamic approach given a longer search time.

The sequence of the trips in a transportation services timetabling problem determines the feasibility of the solution where no trip precedes an earlier one. A candidate selection configuration option may be added where instead of forming the second candidate set by selecting every other swap candidate these candidates must be the slots on different buses from the first candidate set. This limits the number of candidates in the second candidate set and minimises the number of swap trials needed especially when the sequence of all assignments is time feasible. The 5 problems (10 – 14) in Table 10 with the highest cost are used in the next set of experiments which examine the effect of making this simple modification to the dynamic approach based on information specific to the timetabling problem. Table 11 shows the average cost of solutions from 10 experimental runs using the dynamic approach with and without this modification and the corresponding costs for BOOST as shown in Table 10.

Approach	Problem Data Set (as in Table 10)				
	10	11	12	13	14
Modified Dynamic	**3502**	**15568**	**6066**	24132	**14467**
Dynamic (as in Table 10)	3533	15632	6272	24132	14498
BOOST (as in Table 10)	3502	1419	6028	21963	12510

Table 11: Transportation services timetabling solution costs with modified dynamic approach.

From the highlighted costs in Table 11 it is seen that the modification has improved the solution using the dynamic approach in 4 of the 5 problems. For problem 10 the modified dynamic approach has an equal cost to BOOST and for problem 13 the cost has not changed. The modification has improved the performance of the dynamic approach for these transportation services problems but, although the results are not shown, it was not as beneficial for the university timetabling problems. However, it does demonstrate that often with the dynamic approach it is easy to insert problem specific knowledge into the configuration options with beneficial results.

4 Conclusion

The framework within which hyper-heuristics operate has been investigated and three modifications have been developed and tested using experiments and comparisons with published results.

The first modification introduced a self learning mechanism into the choice function to modify the values of parameters in the function as the search progresses in order to allow intensification and diversification of the search. Experimental evidence showed that the modification improved the performance of the choice function which performed better than either a greedy or random method for selecting low-level heuristics. Other experiments showed that a greedy algorithm is an appropriate means of developing an initial solution and no more than 10 or less than 5 low-level heuristics should be used in the non-dynamic approach.

The second and third modifications represent two steps toward addressing the inflexibility associated with a non-dynamic approach where there is a fixed and limited number of pre-designed problem specific low-level heuristics available to a controller using a single choice function. The second modification introduced procedures for dynamically configuring low-level heuristics and the third modification redesigned the controller using a hierarchy of sub-controllers working together at different levels to generate and combine configurations. The combination of these two modifications resulted in a dynamic approach.

Experiments examined the procedure for dynamically configuring low-level heuristics in terms of: their effect on solution costs; their effect on different constraints; and their performance on different problems. The procedure was shown to be feasible but it was observed that a large number of configurations were generated and that it may be possible to combine those with desirable characteristics. However, with dynamic configuration the effectiveness of a controller using a single choice function was questionable and the controller was redesigned to form a hierarchy of sub-controllers. Experiments compared the performance of the new dynamic hyper-heuristic approach, the non-dynamic hyper-heuristic approach, and published results for algorithms that were specifically designed for the particular problems. The problems represented two different timetabling tasks and the dynamic and non-dynamic approaches used generic configuration options and low-level heuristics, respectively. It was not expected that either of the hyper-heuristic approaches would achieve better results than the problem specific algorithms but for 40 percent of the university problems and 43 percent of the transportation problems the hyper-heuristic approaches achieved a lower cost than problem specific algorithms. The results for the non-dynamic and dynamic approaches were very similar but the dynamic approach achieved the same or better results on 57 percent and 35 percent of the transportation and university problems, respectively. The dynamic approach performed well across these two different types of problems using a generic set of configuration options. For the dynamic approach increasing the number of configuration options increases the number of configurations and the controller takes longer to establish reliable performance measures so it is possible that the dynamic approach may perform even better compared to the non-dynamic approach given a longer search time. For a subset of transportation problems it was demonstrated that a simple modification to configuration options using problem specific knowledge produced an improvement in the solutions generated by the dynamic approach.

Hyper-heuristic approaches are relatively new and the findings for the modifications investigated in this study are promising. In particular, the new dynamic approach developed here is encouraging but further studies are needed to: verify its applicability in other problem domains; develop a more intelligent controller able to identify the best configuration options for particular problems; and further investigate methods suggested by Rattadilok et al. [34] to allow the search to be carried out simultaneously on multiple processors.

References

[1] V. Maniezzo, S. Vob, P. Hansen, (Editors) (2009) Special Issue on Mathematical Contributions to Metaheuristics, *Journal of Heuristics*, 3, pp.197-312.

[2] R. Qu, E.K. Burke, B. McCollum, (2009) Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems, *European Journal of Operational Research*, 198(2), pp.392-404.

[3] R. Qu, E.K. Burke, B. McCollum, L.G.T. Merlot, S.Y. Lee, (2009) A Survey of Search Methodologies and Automated System Development for Examination Timetabling, *Journal of Scheduling*, 12(1), pp.55–89.

[4] Z. Rios, (Editor) (2009) Special Issue on Heuristic Research: Advances and Applications, *Journal of Heuristics*, 2, pp.105-196.

[5] E. Alba, E. Talbi, A.J. Nebro, (Editors) (2008) Special Issue on Advances in Metaheuristics for Multiobjective Optimization, *Journal of Heuristics*, 5, pp.311- 412.

[6] G.I. Zobolas, C.D. Tarantilis, G. Ioannou, (2009) Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm, *Computers and Operations Research*, 36(4), pp. 1249-1267.

[7] B. McCollum, (2007) A Perspective on Bridging the Gap between Research and Practice in University Timetabling, in E.K. Burke, H. Rudova, (Editors) Practice and Theory of Automated

Timetabling VI, *Lecture Notes in Computer Science*, 3867, pp.3-23.

[8] C. Blum, A. Roli, (2003) Metaheuristics in combinatorial optimisation: overview and conceptual comparison, *ACM Comput. Sruv.*, 35, pp.268-308.

[9] F.W. Glover, G.A. Kochenberger, (2003) *Handbook of Metaheuristics*, International Series on Operational Research and Management Science, Vol.57, Springer.

[10] A.A. Najafi, F. Azimi, (2009) A priority rules-based heuristic for resource investment project scheduling problem with discounted cash flows and tardiness penalties, *Mathematical Problems in Engineering*, Vol. 2009, article ID 106425.

[11] C. Weng, X. Lu, (2005). Heuristic scheduling for bag-of-tasks applications in combination with QOS in the computational grid source, *Future Generation Computer Systems*, 21(2), pp.271-280.

[12] A. Elazouni, (2009) Heuristic method for multi-project finance-based scheduling, *Construction Management and Economics*, 27(2), pp.199-211.

[13] J.P.O. Fan, G.K. Winley, (2008) A Heuristic Search Algorithm for Flow-shop Scheduling, *Informatica*, 32, pp.453-464.

[14] H.E. Mausser, M.J. Magazine, (1996) Comparison of neural and heuristic methods for a timetabling problem, *European Journal of Operational Research*, 93(2), pp.271-287.

[15] Y. Lee, C. Chen, (2009) A heuristic for the train pathing and timetabling problem, *Transportation Research Part B: Methodological*, 43(8-9), pp.837-851.

[16] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, (1983) Optimization by Simulated Annealing, *Science*, 220(4598), pp.671-680.

[17] F. Glover, (1989) Tabu Search - Part I, *ORSA Journal on Computing*, 1(3), pp.190-206.

[18] D.E. Goldberg, (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Boston, MA: Kluwer Academic Publishers.

[19] M. Dorigo, T. Stützle, (2004) *Ant Colony Optimization*. MIT Press.

[20] J. Kennedy, R.C. Eberhart, Y. Shi, (2001) *Swarm Intelligence*. Morgan Kaufmann Publishers.

[21] B. Selman, H. Kautz, B. Cohen, (1993) Local Search Strategies for Satisability Testing, *Proceedings of 2nd DIMACS Challenge Workshop on Cliques, Color-ing, and Satisability*, Rutgers University, pp.290-295.

[22] K.V. Price, M.R. Storn, J.A. Lampinen, (1998) *Differential Evolution: A Practical Approach to Global Optimization*. Springer.

[23] T.G. Crainic, M. Gendreau, L-M. Rousseau, (Editors) (2010) Special Issue on Recent Advances in Metaheuristics, *Journal of Heuristics*,16, pp.235-535.

[24] R. Bai, J. Blazewicz, E.K. Burke, G. Kendall, B. McCollum, (2007) A simulated annealing hyper-heuristic methodology for flexible decision support, *Technical Report*, School of Computer Science, University of Nottingham (available at: www.asap.cs.nott.ac.uk/publications/pdf/Bai_et_al_2007-8.pdf).

[25] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, (2007) A Graph-Based Hyper Heuristic for Educational Timetabling Problems, *European Journal of Operational Research*, 176(1), pp.177-192.

[26] E.K. Burke, S. Petrovic, R. Qu, (2006) Case-based heuristic selection for timetabling problems, *Journal of Scheduling*, 9(2), pp.115-132.

[27] E.K. Burke, G. Kendall, E. Soubeiga, (2003) A tabu search hyperheuristic for timetabling and rostering, *Journal of Heuristics*, 9(6), pp.451-470.

[28] P. Cowling, G. Kendall, E. Soubeiga, (2000) A Hyperheuristic Approach to Scheduling a Sales Summit, in E.K. Burke, W. Erben, (Editors) Proceedings of the Third International Conference on the Practice and Theory of Automated Timetabling, *Lecture Notes in Computer Science*, 2079, pp.176 – 190.

[29] P. Cowling, G. Kendall, E. Soubeiga, (2002) Hyperheuristics: A Robust Optimisation Method Applied to Nurse Scheduling, in Parallel Problem Solving from Nature VI (PPSN 2002), *Lecture Notes in Computer Science*, pp.851–860.

[30] P. Cowling, G. Kendall, E. Soubeiga, (2001) Hyperheuristic: A Tool for Rapid Prototyping in Scheduling and Optimisation, *Proceedings of the Second European Conference on Evolutionary Computing for Combinatorial Optimisation* (EvoCop 2002), pp.1–10.

[31] E. Soubeiga, (2003) *Development and Application of Hyperheuristics to Personnel Scheduling*, Ph.D. thesis, University of Nottingham School of Computer Science.

[32] S. Karlin, (1972) *A First Course in Stochastic Processes*, Academic Press, New York.

[33] R.S.K. Kwan, M.A. Rahim, (1999) Object Oriented Bus Vehicle Scheduling – the BOOST System, in N.H.M. Wilson, (Editor) *Computer-Aided Transit Scheduling of Public Transport*, Springer, pp.177-179.

[34] P. Rattadilok, A. Gaw, R.S.K. Kwan, (2005) Distributed Choice Function Hyper-heuristics for Timetabling and Scheduling, in E. Burke, M. Trick, (Editors) Practice and Theory of Automated Timetabling V, *Lecture Notes in Computer Science*, 3616, pp.51-67.

Appendix

A1 Factors in the choice function

Factor f_1***:*** A measure of the past performance of the low-level heuristic H_j is calculated using,

$$f_1(H_j)=\sum_{n=1}^{l}\alpha^n\left(\frac{I_n(H_j)}{T_n(H_j)}\right). \quad \text{(A1)}$$

$I_n(H_j)$ is the change in the cost function the n^{th} last time H_j was used, *l* refers to the first time that H_j was selected, and if $I_n(H_j)$ > 0 then the value of the cost function was decreased. $T_n(H_j)$ is the amount of CPU time in milliseconds from the time the low-level heuristic H_j was used the n^{th} last time until the time when it returned a solution to the controller. The parameter α is normalised to have a value in the interval (0, 1) and it assigns a decreasing geometric sequence of weights to the past performance measures of H_j. The initial value of α is determined randomly and if necessary it is automatically modified during the search as described below.

Factor f_2***:*** The performance of a low-level heuristic may be affected by the low-level heuristic that was used immediately before it. Suppose that H_k was used at the last iteration and the use of H_j next is being considered. Then the measure of the past performance of the pair (H_k, H_j) is calculated using,

$$f_2(H_k,H_j)=\sum_{n=1}^{l}\beta^n\left(\frac{I_n(H_k,H_j)}{T_n(H_k,H_j)}\right). \quad \text{(A2)}$$

$I_n(H_k,H_j)$ is the change in the cost function the n^{th} last time the pair (H_k,H_j) was used, *l* refers to the first time in the search that H_j was used immediately after H_k, and if $I_n(H_k,H_j)$ > 0 then the value of the cost function decreased. $T_n(H_k,H_j)$ is the amount of CPU time in milliseconds from the time the pair (H_k,H_j) was used the n^{th} last time until the time when a solution was returned to the controller. The parameter β is normalised to have a value in the interval (0, 1) and it assigns a decreasing geometric sequence of weights to the past performance measures of the pair (H_k,H_j). The initial value of β is determined randomly and if necessary it is automatically modified during the search as described below.

Factor f_3***:*** The two factors f_1 and f_2 intensify the search on low-level heuristics which have performed well in the past. The third factor f_3 diversifies the search by considering low-level heuristics that may not have been used for some time and this is relevant in situations where the search is stuck at a local optimum. The value of f_3 is calculated for each low-level heuristic H_j using,

$$f_3(H_j)=\delta.\tau(H_j). \quad \text{(A3)}$$

$\tau(H_j)$ is the amount of CPU time in milliseconds since the low-level heuristic H_j was last used and each time H_j is used $\tau(H_j)$ is reset to zero. The initial value of δ is selected randomly in the interval (0, 1) and if necessary it is automatically modified during the search as described below.

A2 Modification of parameters in the choice function

Suppose that there are *m* low-level heuristics $H_1, H_2, H_3, \ldots, H_m$, H_k has just been used, and the choice function suggests the use of H_j at the next iteration. Before using H_j determine which of the factors $f_1(H_j)$, $f_2(H_j)$, and $f_3(H_j)$ has the largest value L.

1. If $L = f_1(H_j)$ (or $f_2(H_j)$) then use H_j in the next iteration and modify the value of α to $\alpha(1+\varepsilon)$ (or β to $\beta(1+\varepsilon)$) where $\varepsilon = \frac{I_1(H_j)}{mc_0}$ $\left(\text{or } \frac{I_1(H_k,H_j)}{mc_0}\right)$ and *c₀* is the value of the cost function for the low-level heuristic used at the start of the search. Thus the value of α (or β) increases as confidence grows in the forecasts provided by the choice function and decreases when a low-level heuristic cannot be found that has decreased the value of the cost function the last time it was used.

If $I_1(H_j)$ = 0 (or $I_1(H_k,H_j)$ = 0) and this has not been occurring regularly then no change in the value of the cost function is preferable to a decrease and the value of α (or β) needs to be decreased by a small amount where $\varepsilon = \frac{-T_1(H_j)}{m^2 n_j}$ $\left(\text{or } \frac{-T_1(H_k,H_j)}{m^2 n_j}\right)$ and n_j is the number of times H_j has been used in the search. Here ε is proportional to the time that might be wasted by using H_j and it is a small value if n_j is large which means H_j (or the pair (H_k,H_j)) has often performed well in the past. If $I_1(H_j)$ = 0 (or $I_1(H_k,H_j)$ = 0) and this has been occurring regularly (as defined by the user) then the value of δ is modified as in part 4 of the modification procedures below.

2. If $L = f_3(H_j)$ then determine the trial low-level heuristic H_i which maximises the value of $f_1(H_h) + f_2(H_h)$ for $h = 1, 2, 3, \cdots, m$. Use H_i as a trial and if $F(H_i) < F(H_k)$ then decrease the value of δ to $\delta(1-q)$ and accept that H_i is the low-level heuristic to use in the next iteration. This means that diversification of the search using H_j has been suggested prematurely.

Before the trial use of H_i is conducted $F(H_j) > F(H_i) > f_1(H_j) + f_2(H_j) + f_3(H_i)$ which means that $f_3(H_j) > f_3(H_i)$. If the use of H_i decreases the value of the cost function then it is preferred to H_j and the value of δ needs to be decreased in order to lessen the effect of the factor f_3 in the choice function. If H_j has been suggested prematurely then it is desirable to use H_i and have $F(H_i) > F(H_j)$ which means that if the value of δ changes to $\delta(1-q)$ then $F(H_i) - qf_3(H_i) > F(H_j) - qf_3(H_j)$ and so $q > \dfrac{F(H_j) - F(H_i)}{f_3(H_j) - f_3(H_i)} > 0$. Thus an appropriate value for q is $\dfrac{F(H_j) - F(H_i)}{f_3(H_j) - f_3(H_i)} + \gamma$, where γ is an arbitrarily selected small positive number.

Otherwise, use H_j as suggested by the choice function and do not change the value of δ. This means that diversification of the search using H_j is appropriate.

3. If the values of $f_1(H_j)$, $f_2(H_j)$, and $f_3(H_j)$ are the same then use H_j as suggested by the choice function and do not change the values of α, β, and δ.

4. Regardless of the value of L if the suggested low-level heuristic H_j has been selected and used many times in recent iterations and continually fails to decrease the value of the cost function then increase the value of δ to $\delta + p$ in order to diversify the search using H_n which maximises the value of $\tau(H_h)$ for $h = 1, 2, 3, \ldots, m$. H_n is the low-level heuristic which was last used the longest time ago and $F(H_j) > F(H_n)$. However, it is desirable to diversify the search so that $F(H_n) + p\tau(H_n) > F(H_j) + p\tau(H_j)$ and so $p > \dfrac{F(H_j) - F(H_n)}{\tau(H_n) - \tau(H_j)} > 0$. Thus and an appropriate value for p is $\dfrac{F(H_j) - F(H_n)}{\tau(H_n) - \tau(H_j)} + v$, where v is an arbitrarily selected small positive number.