Choosing the Fittest Subset of Low Level Heuristics in a Hyperheuristic Framework

Konstantin Chakhlevitch and Peter Cowling

MOSAIC Research Centre,
Department of Computing,
University of Bradford,
Bradford BD7 1DP, United Kingdom
{K.Chakhlevitch, P.I.Cowling}@Bradford.ac.uk
http://www.mosaic.brad.ac.uk

Abstract. A hyperheuristic is a high level procedure which searches over a space of low level heuristics rather than directly over the space of problem solutions. The sequence of low level heuristics, applied in an order which is intelligently determined by the hyperheuristic, form a solution method for the problem. In this paper, we consider a hyperheuristic-based methodology where a large set of low level heuristics is constructed by combining simple selection rules. Given sufficient time, this approach is able to achieve high quality results for a real-world personnel scheduling problem. However, some low level heuristics in the set do not make valuable contributions to the search and only slow down the solution process. We introduce learning strategies into hyperheuristics in order to select a fit subset of low level heuristics tailored to a particular problem instance. We compare a range of selection approaches applied to a varied collection of real-world personnel scheduling problem instances.

1 Introduction

Personnel scheduling involves the allocation of the available workforce to timeslots and locations and the assignment of jobs to members of staff. Real-world personnel scheduling problems are NP-hard combinatorial optimisation problems with many complex constraints and a huge number of feasible solutions. Heuristic methods able to find solutions of a good quality in a limited time are often used in personnel scheduling. A recent review of staff scheduling models and methods is given in [1] where the authors identify the development of more general and flexible algorithms as one of the principal research goals. Such algorithms should be robust enough to cope with the changes in problem specifications and scheduling environment [1].

In [2], Burke et al. present an overview of the latest work in the area of *hyperheuristics* – an important direction towards the generalisation of the search methodology. A hyperheuristic is an approach which operates at a higher level of abstraction than a metaheuristic and intelligently chooses the appropriate low level heuristic from a given set depending upon the current state of the

problem [3]. Problem-specific information is concentrated in a set of low level heuristics (which represent simple local search neighbourhoods or dispatching rules) and in the objective function(s) of the problem. All that a hyperheuristic requires to make its decisions is the objective value(s) following the application of each low level heuristic and possibly the CPU time used by the low level heuristic. This makes a hyperheuristic robust and highly flexible.

Several hyperheuristic approaches for real-world scheduling problems use metaheuristics as high level procedure. Cowling et al. [8] develop a hyper-GA approach for a personnel scheduling problem. Their approach is based on genetic algorithm (GA) where the chromosome represents a sequence of low level heuristics. Heuristics are applied in the order given by the chromosome. Han et al. [9] further improve the hyper-GA's performance by enabling the chromosome length to change adaptively during the hyperheuristic run. The method of evolving heuristic choice is successfully implemented by Hart et al. in [7] for a complex scheduling problem of chicken catching and transportation. Burke et al. [10] present a tabu search based hyperheuristic applied to nurse rostering and university timetabling problems where a tabu list of low level heuristics with poor performance is maintained.

Another group of hyperheuristics employs learning mechanisms for making decisions. Gratch and Chien [4] consider a statistical approach to adaptively solve the real-world problem of scheduling satellite communications. Nareyek [5] presents a weight adaptation method which learns how to select attractive low level heuristics during the search. Ross et al. [6] use a learning classifier system based on an evolutionary algorithm to learn a solution process for various instances of the bin-packing problem. Their method determines an order in which simple heuristics should be applied to solve particular problem instance. Cowling et al. ([3], [11]) investigate a hyperheuristic based on statistical ranking of low level heuristics. In this method, historical information about the recent performance of low level heuristics is accumulated in a choice function. The selection of low level heuristic at each decision point depends on the current value of the corresponding choice function.

In [13], we present a range of hyperheuristics where greedy and random methods are mixed in order to maintain a good balance between intensification and diversification of the search. We also show how hyperheuristic approaches can be enhanced by adding tabu lists of recently applied low level heuristics or recently modified events. Since for complex real-world problems there is generally no obvious choice of low level heuristics, we introduce a scheme for designing a large set of low level heuristics for a personnel scheduling problem. Low level heuristics represent all possible combinations of simple selection rules for events and resources. The results of hyperheuristic runs for a real-world personnel scheduling problem are promising. The disadvantage of the hyperheuristic methods described in [13] is that they are relatively slow since selection of low level heuristic to apply at each decision point involves examining all heuristics from a large set. In [14], we analyse the behaviour of individual low level heuristics and their contribution to the construction of the solution. We conclude that some low level

heuristics from the set are fitted better for the particular problem instance than others. The effectiveness of each low level heuristic may vary a great deal from instance to instance and is hard to predict. These conclusions have motivated us to investigate learning strategies for choosing the subset of the fittest low level heuristics. This is the main contribution of this paper. Learning low level heuristics allows us not only to get rid of a significant amount of redundancy in our approach, but, given similar amount of CPU time, to improve further the results previously achieved for all instances of the trainer scheduling problem.

The paper is organised as follows. In the next three sections we briefly formulate the problem and describe our set of low level heuristics and hyperheuristic algorithms. The learning approaches are introduced in Section 5. In Section 6 we analyse the results of all approaches. Section 7 concludes the paper.

2 The trainer scheduling problem

The trainer scheduling problem arises in a large financial institution which regularly organises training for its staff. The problem involves assigning a number of compound events of three different types (i.e. training courses, meetings and projects) to a limited number of training staff, locations, and timeslots. Each timeslot represents a day of the week. We must provide a schedule for 3 months activity. A numerical priority is specified for each event which reflects management's view of the event's utility. The travel of each trainer is penalised depending on the time taken to travel from the home location of the trainer to the location where the event is conducted. The objective is to maximise the total priority for scheduled events while minimising the total travel penalty for the training staff.

The problem is heavily constrained due to a number of limitations related to possible trainers and locations for the events, availability of trainers and rooms, room types and capacities, time windows and durations for the events, workloads for trainers etc (see [13] for details). The integer programming formulation of a much simplified version of a similar problem is given in [8].

3 Low level heuristics

We divide events into two subsets: already scheduled and not yet scheduled. For each category of events we develop separate lists of event selection rules and resource selection rules. Selection rules are chosen in such a way as to reflect the subgoals of the objective function and the constraints with some random selection for diversification. For example, the criteria for selection events which are not yet scheduled can be highest priority, smallest number of possible trainers or locations etc. For already scheduled events we employ such criteria as highest travel penalty, widest time window, largest number of possible trainers or locations and others. The resource selection rules concern the selection of alternative locations, trainers, timeslots or their combinations. Selecting different resources for the event, we intend to reduce the travel penalty for the event and free up

resources which can be used later for scheduling other events. We refer to [13] for further details and complete lists of selection rules.

Combining ("multiplying") event selection rules with resource selection rules for each category of events, we construct a set of 95 low level heuristics (25 heuristics for not scheduled events and 70 heuristics for scheduled events). Such an approach is quite easy to implement since different heuristics can use the same event or resource selection rules as their components. We create only 27 pieces of code representing different event/resource selection mechanisms, and only 5 of these pieces of code are substantially different to each other.

4 Hyperheuristics

In this section we describe our hyperheuristics.

Hyperheuristic Greedy selects and applies at each iteration the low level heuristic either providing the greatest improvement to the objective function or leading to the smallest deterioration (or yielding zero improvement) if there are no improving heuristics. Note that improvements upon the current objective value, not upon the best value found so far are considered for all hyperheuristics in this section. Ties are broken randomly.

Hyperheuristics from the "peckish" group combine random and greedy methods [12]. Hyperheuristic Peckish1 (P1) randomly selects a low level heuristic at each iteration from the candidate list of low level heuristics which improve the current solution or from the whole set of low level heuristics if the candidate list is empty. Hyperheuristic Peckish2 (P2) randomly selects a low level heuristic from the candidate list which contains the n best (not necessarily improving) heuristics. Hyperheuristic Peckish3 (P3) attempts to form the candidate list of only improving heuristics and if such a list is not empty, randomly selects a low level heuristic from it. Otherwise, random selection from the candidate list of n best non-improving heuristics is applied. In hyperheuristic Peckish4 (P4) the candidate list size n is dynamically changed. It is initially set to 1. If at some iteration the improving low level heuristics exist and one of them is applied, the candidate list size n is reset to 1. Otherwise, a low level heuristic is randomly selected from n best non-improving heuristics and candidate list size is incremented. The candidate list size n determines how "greedy" and how "random" the peckish hyperheuristic is – increasing n adds randomness and decreasing n makes the hyperheuristic more greedy.

Hyperheuristics from the next group are based on the ideas of tabu search (see [15]). Hyperheuristic TabuHeuristic (TH) employs a tabu list of recently called heuristics. The size of the tabu list is fixed and set to some prespecified value. The algorithm greedily selects the best low level heuristic at each iteration. If such a heuristic leads to an improved objective value, it is always applied and released from the tabu list if there; a non-improving heuristic is chosen only if it is not in the tabu list and immediately becomes tabu after its application. Hyperheuristic TabuEvent (TE) is similar to TabuHeuristic but the tabu list holds recently selected events. In hyperheuristics TabuHeuristicAdaptive

(THA) and TabuEventAdaptive (TEA) the tabu list size is changed adaptively as the search progresses (see [13]).

Note that all hyperheuristics above ensure that the solution is *changed* at each iteration by discarding low level heuristics failing to find any alternative resources for the selected event.

5 Learning techniques

We have tested two approaches to select the subset of promising low level heuristics from a large superset:

1. Warming up approach (WU): the hyperheuristic identifies the specified number of the fittest low level heuristics during some warm up period (given by the number of iterations) and then either continues its run until a stopping condition is met (WU-C) or restarts from a known initial solution with a reduced set of low level heuristics (WU-R).
2. Step-by-step reduction (SSR) approach: the hyperheuristic gradually reduces the set of low level heuristics during its run until some number of the fittest low level heuristics remain in the set and continues with a reduced set until a stopping condition is met.

The following parameters should be specified for the step-by-step reduction:

- $h_{\min}$ – the minimum possible number of low level heuristics remaining in the subset. In other words, when after several reduction steps the number of the fittest low level heuristics in the reduced set reaches $h_{\min}$, no further reductions are allowed.
- s – reduction step, i.e. the number of iterations between two successive reductions.
- $f \in [0.9; 1)$ – reduction factor, i.e. $h_{after} = [h_{before} * f]$, where h_{after} and h_{before} are the number of low level heuristics in the set after and before reduction respectively and $[x]$ denotes the integer nearest to x.

Selecting different values of s and/or f, we can control the speed of reduction. Increasing s or f makes the reduction slower, decreasing one of the parameters speeds up the process.

Note that both WU and SSR approaches preserve the same ratio of low level heuristics for not yet scheduled events and for already scheduled events in the reduced set as in the whole set of 95 low level heuristics. Since this ratio in the superset is approximately $1 : 3$, the algorithm consecutively takes off three low level heuristics for already scheduled events from the set before one low level heuristic for not scheduled events is discarded.

We use *total improvement* as a fitness criterion for learning low level heuristics. For each low level heuristic, the hyperheuristic accumulates the corresponding value of improvement returned at each iteration. When a reduction occurs, the low level heuristics with the smallest total improvement over all previous iterations are discarded.

Table 1. Real-world datasets for the trainer scheduling problem

	Dataset1	Dataset2	Dataset3	Dataset4	Dataset5	Dataset6
Courses	224	147	224	147	83	161
Meetings	0	0	37	71	27	58
Projects	0	0	71	169	31	171
Trainers	53	54	53	54	54	47
Locations	16	16	16	16	19	16
Rooms	37	39	37	39	35	36

6 Experiments and results

This section describes input data and experimental settings for our methods and contains the comparative analysis of the results of computational experiments.

Input data. We use real datasets provided by a financial institution. Table 1 presents a summary for 6 datasets used in our experiments. The first two datasets have been used in our previous work (see [13]) and represent simplified versions of datasets 3 and 4 since only training courses have to be scheduled. This paper adds realism to the model of [13] by including project and meeting events and their associated constraints. The meetings are particularly difficult to schedule since the presence of a large number of trainers may be required on the same day and at the same location.

Implementation. The problem model, low level heuristics and hyperheuristics were implemented in Microsoft Visual C++ making good use of object orientation and the experiments were run on a Pentium 4 1600MHz PC with 640Mb RAM running under Windows 2000. A single experiment for hyperheuristic consists of 10 runs with different random seeds and starting from the same initial solution. The stopping condition is 500 iterations for the experiments with the whole set of low level heuristics and 1000 iterations when the reduced sets are used. The set of 95 low level heuristics is reduced down to 20 heuristics when learning strategies are applied and the warming up period length is 200 iterations.

Initial schedules are constructed using a greedy heuristic which takes events one by one and assigns the combination of resources yielding the lowest travel penalty for each event until all events have been tried. The events are pre-sorted in descending order of their priority. If the meetings are present in a dataset, ties in priority are broken by descending order of the number of trainers involved to ensure that the meetings have been considered early in schedule construction. The upper bound of all possible schedules for each problem instance represents the solution for the relaxed version of the problem where all constraints on availability of trainers and rooms, room types and capacities, on starting times for the events and their time windows are ignored (see [13] fore more details).

Table 2. Average performance of hyperheuristics applied to a whole set of 95 low level heuristics (WS) and after learning the fittest low level heuristics using step-by-step reduction approach (SSR) given in distance from the upper bound

Hyper-heuristic	Dataset 1		Dataset 2		Dataset 3		Dataset 4		Dataset 5		Dataset 6	
	WS	SSR	WS	SSR	WS	SSR	WS	SSR	WS	SSR	WS	SSR
Initial	4352		6727		13880		24210		9968		29610	
Greedy	751	541	2382	1507	4931	3941	8197	6911	3188	2197	5741	5064
P1	517	503	2398	2033	4856	4336	8085	**5091**	3687	3690	5944	**3406**
P2(25)	1061	556	2842	2092	5910	4901	9432	6644	4091	2295	4998	4090
P3(25)	567	269	2793	2044	5634	4403	7750	5586	3988	2296	5281	4497
P4	941	465	2584	1782	4675	4114	7533	5468	4285	2194	5870	4195
TH(30)	746	362	1990	1770	4623	3760	7775	7106	3687	2595	6022	5378
THA(45)	749	276	2289	1600	4435	3945	7115	7863	4287	3089	6340	5025
TE(N/2)	553	**256**	1973	**1317**	4237	**3241**	7642	6525	3093	1799	4946	4670
TEA(45)	555	360	2185	1693	4259	3281	7568	7027	3289	**1797**	5526	5047

Results. The average results for hyperheuristics managing a large collection of 95 low level heuristics (denoted by WS) and a reduced set of low level heuristics selected by step-by-step reduction approach (SSR) are compared in Table 2. We choose SSR as the approach producing better results on average than both versions of the WU method (see Figure 1). For SSR method the following values of parameters are choosen based on empirical tests: $h_{\min} = 20$, $s = 30$, $f = 0.90$ (fast reduction). These settings provide the most consistent outcomes (although not necessarily the best results for every hyperheuristic and dataset). The figures in Table 2 represent the distances from the corresponding upper bound. The numbers in parentheses after some of hyperheuristics' names are the values of either candidate list size or tabu list size. Hyperheuristics are not particularly sensitive to the values of these parameters and we use moderate values for illustration purposes. The only exception is hyperheuristic TabuEvent which produces the most consistent outcomes when the tabu list may contain up to a half of all the events in the dataset. This is denoted by $N/2$ in Table 2, where N is the number of events in the dataset. It is evident from the table that the learning technique embedded into a hyperheuristic leads to significant improvements in the quality of the schedules. The small deviations in the objective values observed represent practically very significant differences in trainer inconvenience due to additional travel, or additional low priority scheduled events. Analysing the performance of individual hyperheuristics with learning, we can notice from Table 2 that hyperheuristics whose behaviour is primarily greedy (Greedy and tabu list based) outperform their peckish counterparts for Datasets 1,2,3 and 5

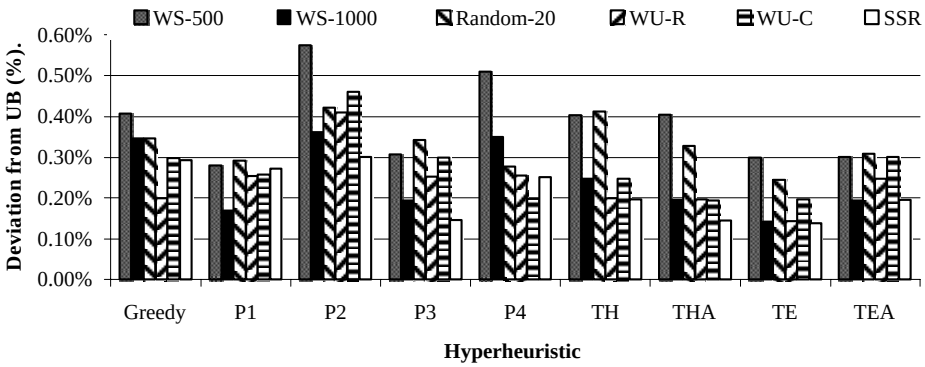

Fig. 1. Comparison of hyperheuristic approaches for Dataset 1 (deviation in % from the upper bound)

with the best results consistently delivered by TabuEvent hyperheuristic. For the most hard-to-schedule Datasets 4 and 6 peckish hyperheuristics are more successful, especially Peckish1 where the random component dominates and more frequent calls of low level heuristics which worsen the objective value are possible. This is not surprising since for tightly constrained datasets it is difficult to find alternative resources without increasing travel penalties. Scheduling a new event may require a few rescheduling moves which worsen the current schedule by increasing penalties. Hyperheuristics with a greater degree of randomness accept such moves more often than more greedy hyperheuristics. This provides a smaller number of not scheduled events in the best schedule. On the other hand, the presence of a greedy component in a peckish hyperheuristic guarantees that a worse move will not be accepted too often and therefore the penalties will not be significantly damaged. For "easier" Datasets 1,2,3 and 5 there is more freedom to find resources and greedier hyperheuristics perform better.

Figure 1 demonstrates the difference in performance of hyperheuristic approaches with and without learning for Dataset 1. The results for other datasets follow similar patterns. It is clear that SSR is the best method in most occasions. In order to check the quality of low level heuristics selected by SSR algorithm, we have conducted a range of experiments with the reduced sets of 20 randomly selected low level heuristics ("Random-20" columns in Figure 1). The results of these experiments are rather erratic: the high quality solutions comparable to those produced by SSR approach are present along with very poor, unacceptable ones. For example, the worst objective value achieved by hyperheuristic Greedy for Dataset 1 using the Random-20 approach is unacceptable 3227 from the upper bound whereas SSR is never more than 1214 from the bound. These results give strong evidence for the importance of the right selection of low level heuristics in the subset. We can also notice from Figure 1 that WU strategies are less consistent than SSR and their performance varies widely for different hyperheuristics. Comparing the results of WS approach for 500 and 1000 itera-

tions (WS-500 and WS-1000 columns in Figure 1) with those of SSR approach, we can observe that although further improvements have been recorded during additional 500 iterations for the hyperheuristics applied to a large set of low level heuristics, the solutions obtained are generally slightly worse than that of hyperheuristics employing the SSR strategy. The WS-500 and SSR approaches use similar amounts of CPU time, although the SSR approaches carry out 1000 iterations in their time compared to WS-500's 500 iterations.

The low level heuristics in the reduced set are usually those yielding the most frequent and valuable improvements to the current solution since the selection criterion used in the SSR approach is the total improvement achieved by the low level heuristic. The most effective event selection rules for not scheduled events involve random selection of events and selection based on priority. Other event selection rules are also present in the reduced set but less often and depending on the hyperheuristic and dataset. The resource selection rules for not scheduled events are distributed quite evenly among the best low level heuristics. For already scheduled events we observe different preferences in selection rules. All event selection rules are usually represented in the reduced set of low level heuristics although random selection, selection of events with the highest priority and with the highest travel penalty are present in higher ratios. Only four resource selection rules for already scheduled events are contained in low level heuristics from the reduced set. Three of them are well suited to the total improvement criterion selecting appropriate trainers, locations or their combinations which reduce the travel penalty for the selected event. The fourth rule selects alternative timeslots for scheduled events and always yields zero improvement.

Summarising the above analysis, we observe that the reduced sets of low level heuristics obtained by applying our learning approaches are similar for different datasets. It appears that the choice of low level heuristics depends primarily on the method used to select the reduced subset and to a lesser extent on the problem instance and hyperheuristic. Successful individual runs of hyperheuristics with a subsets of randomly selected low level heuristics support the idea that even greater diversity could be useful. Therefore, the development of more advanced and comprehensive learning mechanisms is an interesting research direction, possibly employing weight adaptation schemes similar to those presented by Nareyek in [5] or advanced scoring systems for ranking low level heuristics.

7 Conclusions

Since a hyperheuristic accumulates knowledge about low level heuristic performance rather than directly about the problem, there is growing evidence that the approach is robust for a variety of problem instances and across problem domains. In this paper, we have added to this evidence. We have presented a range of hyperheuristic algorithms managing a large collection of low level heuristics constructed by combining simple selection rules. In order to make the methodology faster and more effective, we have introduced learning techniques into hyperheuristics which allow us to reduce the set of low level heuristics leaving

only those most likely providing regular improvements to the current solution for the particular problem instance. The experimental study shows that our hyperheuristics produce high quality results for difficult real-world instances of personnel scheduling problem. Hyperheuristics with learning outperform their counterparts dealing with a large set of low level heuristics both in terms of solution quality and CPU time.

The methodology presented in this paper has the potential to solve other complex real-world scheduling problems. By considering manual schedule generation and repair techniques, it is usually straightforward to design selection rules and therefore to form a set of low level heuristics, even when the problem structure is poorly understood. The choice of which selection rules will work in an automated system is difficult, and we have presented and compared several methods to identify effective low level heuristics from a large set in this paper. Investigating the robustness of the method across other problem domains is our primary goal for the near future.

References

1. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Sier, D.: Staff Scheduling and Rostering: A Review of Applications, Methods and Models. European Journal of Operational Research **153** (2004) 3-27
2. Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P., Schulenburg, S.: Hyperheuristics: An Emerging Direction in Modern Search Technology. In:Glover, F., Kochenberger, G.A. (eds.): Handbook of Metaheuristics. Kluwer Academic Publishers, Boston Dordrecht London (2003) 457-474
3. Cowling, P., Kendall, G., Soubeiga, E.: A Hyperheuristic Approach to Scheduling a Sales Summit. In: Burke, E., Erben, W. (eds.): Practice and Theory of Automated Timetabling III: PATAT2000. Lecture Notes in Computer Science, Vol. 2079. Springer-Verlag, Berlin Heidelberg New York (2000) 176-190
4. Gratch, J., Chien, S.: Adaptive Problem-Solving for Large-Scale Scheduling Problems: A Case Study. Journal of Artificial Intelligence Research **4** (1996) 365-396
5. Nareyek, A.: Choosing Search Heuristics by Non-Stationary Reinforcement Learning. In: Resende, M., de Souza, J. (eds.): Metaheuristics: Computer Decision-Making. Kluwer Academic Publishers, Boston Dordrecht London (2003) 523-544
6. Ross, P., Schulenburg, S., Marín-Blázquez, J. G., Hart, E.: Hyper-Heuristics: Learning to Combine Simple Heuristics in Bin-packing Problems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). Morgan Kauffmann (2002) 942-948.
7. Hart, E., Ross, P., Nelson, J.: Solving a Real-World Problem Using an Evolving Heuristically Driven Schedule Builder. Evolutionary Computation **6** (1998) 61-80
8. Cowling, P., Kendall, G., Han, L.: An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In: Proceedings of 2002 Congress on Evolutionary Computation (CEC2002). IEEE Computer Society Press, Honolulu, USA (2002) 1185-1190
9. Han, L., Kendall, G., Cowling., P.: An Adaptive Length Chromosome Hyperheuristic Genetic Algorithm for a Trainer Scheduling Problem. In: Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02). Orchid Country Club, Singapore (2002) 267-271

10. Burke, E., Kendall, G., Soubeiga, E.: A Tabu-Search Hyperheuristic for Timetabling and Rostering. Journal of Heuristics **9** (2003) 451-470
11. Cowling, P., Kendall, G., Soubeiga, E.: A Parameter-Free Hyperheuristic for Scheduling a Sales Summit. In: Proceedings of the Third Metaheuristic International Conference (MIC'2001). Porto, Portugal (2001) 127-131
12. Corne, D., Ross, P.: Peckish Initialisation Strategies for Evolutionary Timetabling. In: Burke, E., Ross, P. (eds.): Practice and Theory of Automated Timetabling. Lecture Notes in Computer Science, Vol. 1153, Springer-Verlag, Berlin Heidelberg New York (1995) 227-240
13. Cowling, P., Chakhlevitch, K.: Hyperheuristics for Managing a Large Collection of Low Level Heuristics to Schedule Personnel. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation (CEC'2003). IEEE Computer Society Press, Canberra, Australia (2003) 1214-1221
14. Chakhlevitch, K.: Hyperheuristics Which Manage Large Collections of Low Level Heuristics. PhD thesis, in preparation
15. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston Dordrecht London (1997)