BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES, Vol. 60, No. 4, 2012
DOI: 10.2478/v10175-012-0093-7

VARIA

Hyper-heuristics for cross-domain search

T. CICHOWICZ, M. DROZDOWSKI, M. FRANKIEWICZ, G. PAWLAK[*],
F. RYTWINSKI, and J. WASILEWSKI

Institute of Computing Science, Poznań University of Technology, 2 Piotrowo St., 60-965 Poznań, Poland

Abstract. In this paper we present two hyper-heuristics developed for the Cross-Domain Heuristic Search Challenge. Hyper-heuristics solve hard combinatorial problems by guiding low level heuristics, rather than by manipulating problem solutions directly. Two hyper-heuristics are presented: Five Phase Approach and Genetic Hive. Development paths of the algorithms and testing methods are outlined. Performance of both methods is studied. Useful and interesting experience gained in construction of the hyper-heuristics are presented. Conclusions and recommendations for the future advancement of hyper-heuristic methodologies are discussed.

Key words: hyper-heuristics, cross-domain heuristic search, HyFlex.

1. Introduction

In the recent decades, the study of computational methods solving hard combinatorial problems was concentrated on building metaheuristic algorithms. Many metaheuristics exploiting intimate knowledge of the problem features were developed with great success. The dependence on the particular problem (also called a *domain*) is both a strength, and a weakness. Namely, the dependence on the problem, the need for tuning, and hence human interference, make the metaheuristics hardly portable and bound to a particular problem/domain (see for example [1–5]).

To overcome this difficulty the idea of Hyper-heuristics (HH) was proposed. There are two levels of abstraction. On the higher level hyper-heuristics explore the space of low level heuristics (LLHs) instead of directly operating on the space of problem solutions. The low level heuristics perform moves in the space of the ground combinatorial problem solutions similarly to the classic local search methods. The LLHs are an interface between the problem domain and the domain-independent guiding algorithm. Thus, hyper-heuristics serve as a general concept of constructing methods applicable in solving broad range of hard problems. Examples in which dedicated two-level heuristics were applied to the solve selected problems can be found in [6, 7]. Hyper-heuristic approach has a potential advantage of, on one hand, unifying porting and automating tuning processes and on the other hand, limiting generality of the meta-heuristics. It in turn, should allow solving a broad range of combinatorial problems on the basis of the same methodology. The challenge that represents developing hyper-heuristics is to construct methods capable of solving different kinds of combinatorial problems when problem formulation and its features are not directly known. It means that at least two issues must be adequately addressed: Which LLHs concepts can be implemented in every domain? Which algorithm can guide the LLHs efficiently?

To foster new ideas for hyper-heuristics a Cross-Domain Heuristic Search Challenge (CHeSC) was held in 2011 [8]. In this paper we report on our achievements in CHESC, lessons learnt, and conclusions for the future directions of hyper-heuristic developments. Two hyperheuristics were developed: Five Phase Approach (5Ph) and Genetic Hive (GH). The 5Ph method evolved into a complex LLH control architecture. GH was developed as alternative and competitive solution. Since the team was restricted to submit only one method, GH was finally chosen to be qualified in the challenge as slightly more effective. Though, we were not particularly successful in the competition (we were qualified in the middle of the ranking), we would like to share the experience of constructing the two hyperheuristics. HH research area is relatively new, and we progressed from the state of initial misconceptions towards some comprehension of HH operation. We believe that it is a valuable contribution to the research area.

The remainder of the paper is organized as follows. In the next section we introduce HyFlex, and the rules of CHESC competition. Section 3 is dedicated to the 5Ph hyperheuristic. Section 4 presents Genetic Hive hyperheuristic. Computational results are reported in Sec. 5. Conclusions and recommendations for future research are provided in the last section.

2. HyFlex and CHESC

HyFlex [9] is a Java library embodying a concept of hyper-heuristics using four types of low level heuristics: mutational, ruin-recreate, local search, and crossover LLHs. It is assumed that a pool of solutions is maintained by the HH. The HH must decide which LLH to apply and on which solution. Since the area for solutions is limited, HH has to decide which solutions to replace. We describe the function of the LLHs on the examples evolving from bin packing domain. Three local search LLHs are provided in HyFlex for bin packing: swap from the lowest bin, split a bin, optimize one bin. The first LLH takes

[*]e-mail: grzegorz.pawlak@cs.put.poznan.pl

the largest piece from the least filled bin, and exchanges it with a smaller piece from a randomly selected bin. If packing is invalid, two pieces of smaller size are used. If still no valid packing is possible, then the LLH performs nothing. Mutational LLHs in bin packing domain are: swap, and repack the lowest filled bin. The swap LLH exchanges two different pieces at random. If the destination bin cannot accommodate the new piece, then the piece is put into a new empty bin. Two ruin-recreate heuristics are provided: destroy x highest bins, destroy x lowest bins. In the former LLH the x most filled bins are repacked using best-fit heuristic. Note that parameter x determines the intensity of changes. The crossover LLH is exon shuffling crossover [6]. Ruin-recreate, mutational, and some crossover LLHs may have their own parameters controlling 'depth of search' or 'intensity of mutation'. HyFlex evaluates solutions and the objective functions are uniformly minimized. The problem domains covered by HyFlex are: maximum satisfiability (Max-SAT), bin packing (BP), permutation flowshop (FS), personnel scheduling (PS), traveling salesman problem (TSP), and vehicle routing problem (VRP).

The Cross-Domain Heuristic Search Challenge (CHeSC) [10] used 4 domains known at the development stage (Max-SAT, Bin Packing, FS, PS), and 2 hidden domains (TSP, VRP). The hidden domains remained unknown to the competitors at the development stage. HyFlex provided 10 training instances at the development stage. During the competition the HH algorithm was not aware which domain and instance it was solving. For the known domains, 3 randomly chosen training instances and 2 hidden instances were used. For the hidden domain 5 instances were used. The run times were limited to 10 minutes for each instance. Since LLHs and HHs exploited randomized algorithms, quality of the solutions was a median from 31 runs on one instance. It will be shown in Sec. 5 that the choice of the median was an important design decision. The scoring method used "Formula 1" metric which assigned 10, 8, 6, 5, 4, 3, 2, 1 points to the top 8 consecutive performers on each instance (race). The scores were collected over all tests. Thus, for 6 domains (4 known and 2 hidden) and 5 instances, the maximum score was 300 points. Let us observe that the rules of the competition required domain- and instance-independent construction of the HH algorithm. It was possible for HH to judge progress in solving an instance only by changes of the objective function, and LLH execution times only.

3. Five phase approach

In this section we sketch the development path of the Five Phase Approach (5Ph) starting from its initial Three Phase form (3Ph).

3.1. Initial three phase approach. There were two initial ideas for the construction of the Three Phase method (3Ph): To avoid being stacked in one unsuccessful search trajectory, the hyperheuristic should follow many alternative paths in parallel. The search should undergo a repetitive pattern of intensifying (improving) the current solution, and diversifying the search after stagnation. Since the rules of the competition disallowed multi-threading, the algorithm switched between virtual streams handling different solutions, thus providing an equivalent of the parallel threads. The architecture of Three Phase algorithm (3Ph) is shown in Fig. 1. In the following, we outline components of 3Ph.

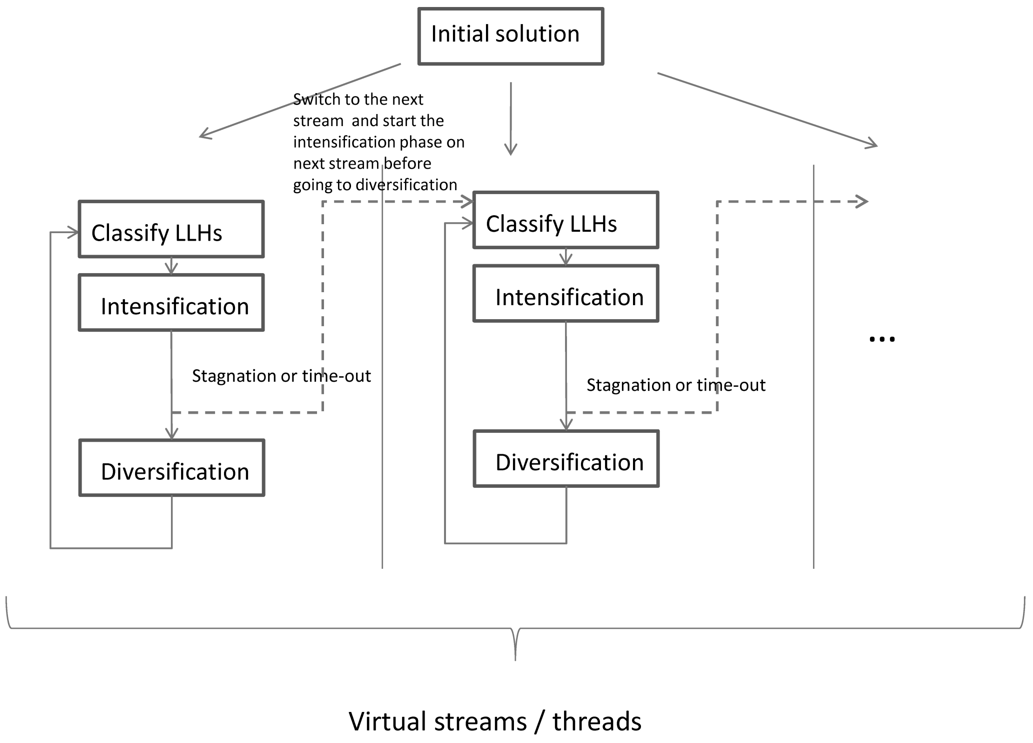


Fig. 1. Architecture of 3Ph algorithm

LLH classification. To identify which LLHs can improve solutions and which can diverge from the current search point, an online classifier of the LLH behavior was necessary. The classification algorithm ran an LLH a predetermined number of times, and the values of the objective function were recorded. For LLH i, $score_i = -a_i/\Delta_i$ was calculated, where a_i was a slope of the linear approximation of the objective function values recorded in the test LLH executions, Δ_i was the execution time. Two types of LLHs were distinguished: *improvers* with $score_i > 0$ applied in the intensification, and *mashers* with $\mathbf{score}_i < -0.2$ applicable in diversification stage. The remaining LLHs were excluded from use in the next algorithm cycle.

Intensification. In each solution stream a randomly chosen improver LLHs were executed for a predetermined time (experimentally set to 5 sec.). The probability of choosing certain LLH i was proportional to its score $stat_i$ as an improver. In a thread, intensification finished after the assumed time period or after detecting (local) stagnation (see the next paragraph). The intensification phase reached global end when all streams reached the time limit or all were in the stagnation state.

Stagnation. Stagnation of a thread was declared if the solution did not improve in a number of consecutive iterations of the chosen LLH.

Diversification. To depart from the current solution (possibly from a local optimum), a randomly chosen LLH was executed for a predetermined time.

Preliminary tests exposed a number of misconceptions and 3Ph performance deficiencies: After just 2 min, 3Ph could not improve the objective function. Since the rules of the competition allowed 10 min. runtime, 80% of the time limit remained unexploited. Classification of LLHs consumed one third of the time limit, but the resulting assignment did not persist between the iterations. Hence, LLH classification was time-consuming, and counterproductive. The LLH performance volatility demonstrate that their behavior depends on the solutions on which they operate. Since the LLH classification is so short-lived, it is better to combine classification with the actual search. The intensification phase almost always reached stagnation before the 5 sec. limit. Many threads (solution streams) produced the same solution. Only a small number of threads produced good results. This implied that the diversification process was insufficient. The number of solution streams needed not to be big if suitably guided.

An intermediate algorithm 4Ph was constructed using crossover LLHs. This, in turn, was substituted by the 5Ph algorithm.

3.2. Final five phase approach. 5Ph inherited the overall control structure of 3Ph (cf. Fig. 2). Intensification, stagnation, diversification and solution streams were preserved in 5Ph. It was observed that small groups of LLHs (clusters) have more diversified behavior and hence bigger potential for improving the objective function than singleton LLHs. The clusters should mutate to diversify their search capabilities. The length of LLH clusters was experimentally set to three, hence we call them triplets. The number of threads was reduced to 7 because the runtime limits restricted spawning unproductive threads. Greater diversity was achieved by changing the use of LLHs than by brute force thread spawning. In the following we refer to global iterations as to the iterations encompassing all the phases from intensification to diversification. 5Ph ran iteratively to the 10 min. time limit.

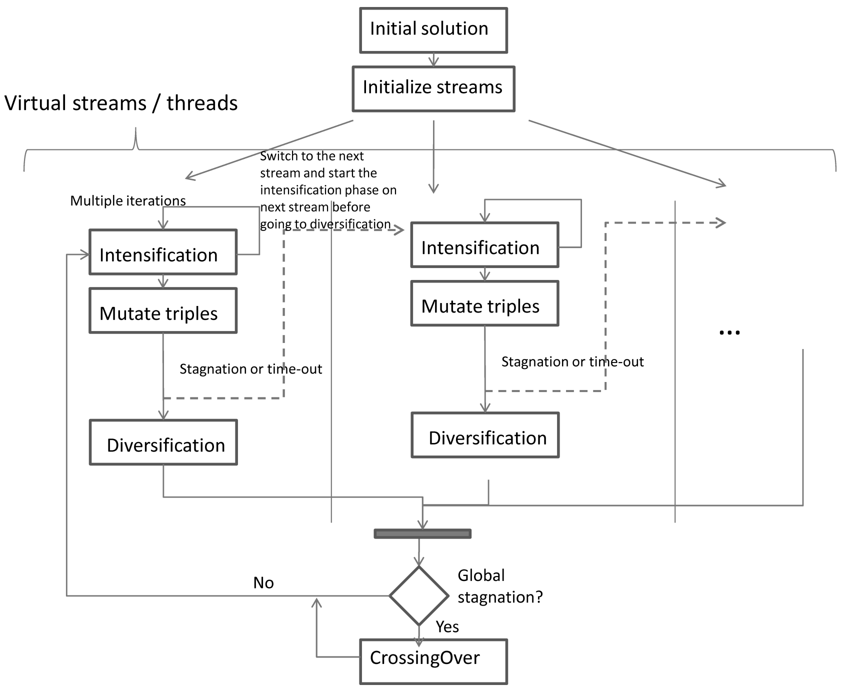


Fig. 2. 5Ph architecture

Solution streams initiation. This step was applied once for each thread to scatter the search paths into different areas of the solutions space. Random LLHs were applied for five seconds on every thread. The same procedure was applied in the diversification phase described in the following.

Intensification and local stagnation. The triplets comprised local search LLHs only. LLH classification was done on the fly. Let ϕ_i denote the improvement of the objective function and δ_i the execution time in the last run of LLH i. The score of LLH i was $score_i = score_i * e^{\phi_i/\delta_i}$, where initial $score_i = 1$, $\phi_i > 0$ represents improvements. LLH scores were preserved between global iterations. An LLH from the triplet was selected with probability proportional to its score and applied on the solution. The selected LLH was run to its stagnation in the local thread. As a side effect some LLHs were eliminated by just one LLH with quickly growing score. To prevent such a situation, the LLHs that were not applied so far, had their scores increased by 5% with each execution of any LLH in the thread. An LLH stagnation was declared if the objective function was not improved in four consecutive runs of the elected LLH or 5 sec. time limit was reached. The selection of a new LLH and its run to stagnation was repeated $NoIt = \lceil gs/3 + 3 \rceil$ times, where gs is the number of global stagnations (see the stagnation paragraph). Thus, the pressure on intensification increased with the number of global iterations. After *NoIt* iterations we reached thread stagnation.

Triplet mutation. For each thread a triplet from some other thread was selected with the probability proportional to the average score of the three LLHs. The worst LLH in the current triplet was replaced with the best LLH from the foreign triplet. After thread stagnation and triplet mutation in one thread, 5Ph switched to the next thread.

Diversification. If local stagnation was detected in all the threads, then diversification was applied on each thread separately in the same way as it was described in the initiation step.

Global stagnation. Global stagnation occurred if in 3 global iterations, i.e. after running all threads three times through stagnation, mutation and diversification, the objective function did not improve.

Solution crossingover. After global stagnation an attempt was made to combine solution features by applying crossover operators. A randomly chosen LLH of crossover type was applied on two solutions from the threads. One solution was chosen with probability proportional to the objective value, the other solution with the probability inversely proportional to the objective value. Thus, on average good solutions were crossed over with bad solutions. During the tests the number of crossovers was gradually decreased from 10 to 1. At the end of this step the global stagnation counter gs was reset to 0, and 5Ph restarted in the intensification phase.

4. Genetic Hive algorithm

Genetic Hive algorithm (GH) was inspired by Bees Algorithm [11] and by genetic algorithms [12]. Bee population split into groups: those actively searching for food locations and the other part remaining in the hive. After some time, the searching bees return to the hive and those previously in the hive, set out to explore the food locations. The bees finding larger amounts of food have bigger chances to proliferate. Evolving sequences of LLH are the equivalent of bees searching for food. Searching for food locations corresponds with searching solution space by the sequence of LLHs. GH embodies the search by using evolutionary and agent colony methods. The idea of the algorithm is one of the original contributions of the paper. A pseudocode of GH is shown in Fig. 3.

```
1  INITSEARCH()
2  while $RuneTime < TimeLimit$
3      do
4        for each agent $a \in L$
5            do EVALUATE-AGENT($a$)
6        $B \leftarrow bSize$ best agents from set $L$
7        $S \leftarrow sSize$ random agents from set $H \setminus B$
8        $O \leftarrow$ EVOLVE($B$, $oSize$)
9        $H \leftarrow B \cup S \cup O$
10       $L \leftarrow B \cup ((lSize - bSize)$ random agents from set $H \setminus B)$;
```

Fig. 3. Pseudocode of GH algorithm

In the following section we refer to the sequences of LLHs as to agents and to procedures and code lines in Fig. 3. In the NITEARCH procedure set H of *hSize* random agents, and a *lSize* random solutions (search locations) are created. Agents from set H are chosen randomly to form set L of *lSize* agents assigned to the *lSize* locations. In line 5 agents in set L are applied on their solutions. Each agent obtains a score that is the relative solution improvement. In line 6 *bSize* best agents are selected to set B and remain in their locations. Agents in set $L \setminus B$ are moved to set H as bees returning to the hive. Set S of *sSize* invariant agents is randomly selected from $H \setminus B$ to simulate survival of some agents, and extinction of the remaining *hSize-bSize-sSize* agents. In procedure VOLVE *oSize* new agents are constructed using crossing-over and mutation of the agents in set B. Pairs of agents are selected to crossingover by roulette wheel method with probabilities proportional to the agent score. Next, a single point crossover is applied to the LLH sequences (i.e. the agents). In other words, the sequences are cut in two random places and their prefixes are exchanged. After that, *mutation* randomly changes LLHs in the agent sequence. Elements from sets B, S, O create new version of set H (line 9). The *lSzie-bSize* abandoned search slots are assigned new agents chosen from the new set H. GH algorithm repeats iteratively lines 2-10 until exploiting the time limit. *hSzie, lSize, bSize, sSize, oSize* are control parameters of the algorithm.

5. Computational experiments

Computational experiments and the testbed followed CHESC rules as closely as possible to guarantee comparability with the final competition results. In the following subsection we describe the testbed, results of the tests on individual LLHs, and on the final HHs.

5.1. Testing environment. To test the algorithms in reasonable time, and in manner allowing reproduction of the results the dedicated distributed testing system with a centralized management unit was designed. Since the comparability of the results was essential for further decisions, all computing nodes had the same hardware and software configurations. Each node was equipped with a modern quad-core x64 processor with ample memory(8GB). The experiments were run on Linux platform with disabled unnecessary system tasks to ensure non-preemtable run on a processor core in dedicated memory. Because HyFlex framework is Java module, all experiments had to be run on Java Virtual Machine (JVM). The JVM 1.6.0_26 version was used.

A shared disk space for storing the results was provided. To minimize I/Os communication and blocking on file, all the results were written to local log files using caching/buffering mechanisms and multi-part logs. We also proposed slim tasks management model by using system built-in commands. The centralized management unit was used for setting shared data such as equal Java random number generator seed in every node. After the experiment initialization, tasks were distributed between 40 cores (10 different instances * 4 domains = 40 cores). According to the competition rules, it took 10 minutes of parallel processing to execute the test. Each experiment was performed with fixed set of settings determined by the algorithm tested. Afterwards, the results were merged and exported to analysis tools.

5.2. Low level heuristic tests. We started our study with intuitive expectation that the LLHs preserve certain performance patterns. We expected patterns of the form: LLH i typically improves solution quality until arriving in stagnation, LLH j in domain X uniformly provides good solutions.

To determine the characteristics of the HyFlex LLHs [9], a number of tests has been conducted. 10 LLHs from the groups of mutational, ruin-recreate, and local search LLHs have been run on each of the ten instances from each domain. For mutational and ruin-recreate LLHs with control parameters (intensity of mutation, depth of search) the parameters were chosen randomly from range [0,1] with 0.1 grid. This gave a total of 1200 tests. In a test, the chosen LLH was repetitively executed for at least 300 sec. During the test the best objective function value obtained to this point and the current function value were recorded.

A more complex tests were needed for crossover LLHs to avoid crossover of a solution with itself. For this purpose 10 random solutions have been initialized and then randomly crossed-over. The result of crossover operation replaced one out of the previously chosen solutions. The time limit was modified to 60 sec. The best objective function value was chosen from the 10 stored solutions. This gave another 400 tests.

To verify if solution diversity changes behavior of LLH, analogous tests have been conducted after mixing the solution by 30 sec. run of a random LLH. The only observed difference was that the LLHs previously improving objective function had a slightly worse starting point. This was usually compensated within 30–60 sec. of the given LLH run time.

The results proved that the behavior of LLHs of a certain type is inconsistent between domains and instances in the same domain. Another observation was that some of the LLHs failed completely to change the objective function along their execution at all. LLH behavior may not correspond with common understanding of the LLH class name. In consequence, we concluded that the LLH performance is instance-, domain- and solution-dependent.

5.3. Hyper heuristics results. In this section we report on electing best HH submitted to the challenge, and then we comment on the results in the competition. In general, comparing performance of heuristics fairly is very difficult. In our case the challenge was smaller because we compared HHs in fully controllable conditions, on the same instances. A crucial decision was the choice of performance metric. Formula 1 scoring method chosen by the competition organizers gives a strong preference to the order of competitors, ignoring the difference in the objective function. Therefore, to elect the best of our HHs we used the number of wins against other HHs on the given 40 instances. The results are collected in Table 1. The second column in Table 1 provides names of the HHs. RND LLH maintains 10 independent solutions, and for each of them applies iteratively a randomly chosen LLH. Each LLH executes iteratively, in the given instance, one LLH. The best solution generated by any LLHs is returned. The remaining LLHs are variants of the methods described in Secs. 3 and 4. In the following columns, each entry represents the number of wins vs the number of defeats of the HH given in the row against the HH given in the column. As secondary criteria we used the number of times a given HH provided the best solution, and the number of unique best solutions. As a result, variants 5Ph-160-63 of 5Ph and GenHive-68 of GH were chosen for final comparison. The results of our finals are shown in Table 2.

The two final HHs were compared on all the HyFlex instances. Minima, medians and averages of the objective function are presented in Table 2. A value in the boldface is the better one of two values and the method producing it was winning. It can be seen in Table 2 that GH is better on minima, 5Ph is better on averages and there is a tie in medians. The reason is quite simple: distributions of the results are different. Hence, the choice of the HH performance metric is essential for the final winner. Moreover, this demonstrates that a single performance index is insufficient to show the whole complexity of the results. Since the rules of CHESC, by the Formula 1 scoring system, preferred methods giving the best solutions (minima), GH method was our final choice.

Table 1

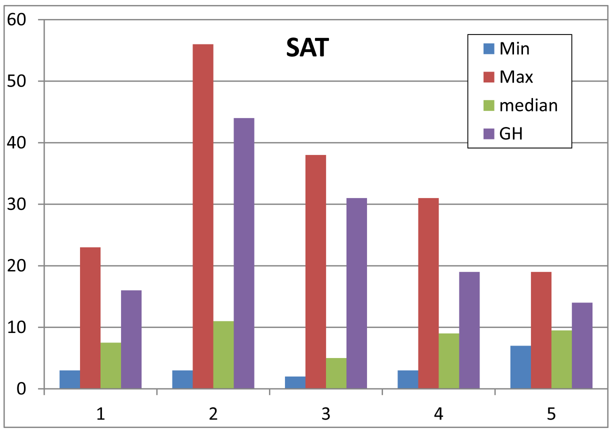
Pairwise comparison of CS-Put HHs

	Hyper Heuristic	1	2	3	4	5	6	7	8	9	10	11	12	13
1	RND LLH	x												
2	Each LLH	22/17	x											
3	4Ph-LS	36/2	35/5	x										
4	4Ph-RND LLH	25/14	21/17	6/32	x									
5	5Ph-LS 3pl	37/3	35/5	19/17	36/3	x								
6	5Ph-RND LLH 3pl	25/15	28/12	6/33	18/15	4/34	x							
7	5Ph-154	37/2	35/5	17/20	33/4	20/17	37/2	x						
8	5Ph-155	32/7	31/8	17/20	31/8	16/18	31/8	16/22	x					
9	5Ph-160-40	35/5	27/10	19/18	28/12	18/21	25/15	17/22	19/20	x				
10	5Ph-160-46	30/10	27/12	19/20	24/15	18/21	24/14	16/20	15/24	14/24	x			
11	5Ph-160-63	32/6	35/5	17/21	29/11	20/19	30/10	15/23	18/20	20/19	22/17	x		
12	GenHive-35	32/8	28/10	14/24	30/9	17/21	28/12	16/22	17/20	17/22	19/21	14/23	x	
13	GenHive-65	36/4	27/13	19/19	29/9	19/18	26/13	18/21	21/18	21/17	21/16	19/21	20/19	x
14	GenHive-68	35/5	29/11	21/17	30/10	24/15	30/10	19/19	26/14	22/18	24/15	23/15	26/11	23/15

Table 2

Results for 5Ph an GH

Ins.	SAT	FS	PS	BP	SAT	FS	PS	BP
		5Ph				**GH**		
Minimum								
1	2	6331	3310	**0.0062**	2	**6318**	**3303**	0.0064
2	19	6277	**2045**	0.0036	19	**6271**	2070	0.0068
3	**14**	**6340**	350	**0.0199**	15	6350	**320**	0.0207
4	1	6366	19	0.0205	**0**	**6339**	**11**	**0.0197**
5	1	6415	23	0.0003	1	**6404**	**17**	0.0003
6	2	10507	22	0.0032	**1**	**10501**	**18**	**0.0031**
7	5	10923	1120	0.0162	5	10923	**112**	**0.0057**
8	5	26329	2230	0.0260	5	**26292**	**2225**	**0.0172**
9	5	26838	3266	0.0459	5	**26780**	**3165**	**0.0452**
10	209	26655	10037	0.0070	209	**26639**	**9509**	**0.0037**
Median								
1	**10**	6389	**3347**	**0.0110**	12	**6377**	3360	0.0255
2	30	6330	**2350**	**0.0081**	**27**	**6323**	2540	0.0195
3	24	6409	**400**	**0.0228**	**23**	**6402**	465	0.0272
4	**10.5**	6391	28	**0.0242**	17	**6377**	**27.5**	0.0265
5	**7**	6471	31	**0.0063**	27	**6468**	31	0.0105
6	**18.5**	10545	31	**0.0084**	45	**10537**	31	0.0089
7	**6**	10972	**1358**	0.0899	9	**10970**	1379	**0.0660**
8	**6**	26455	2688	0.1051	9	**26434**	**2429**	**0.0738**
9	**9**	26943	3994	0.1000	12	**26928**	**3533**	**0.0802**
10	**211**	26761	26366	**0.0197**	217	**26751**	**11057**	0.0271
Average								
1	**14.8380**	**6386.1**	**3348.3**	**0.0101**	44.73	6421.87	14257.25	0.0447
2	**30.9225**	**6328.5**	**2352.1**	**0.0092**	37.89	6360.88	15639.17	0.0427
3	**25.4296**	**6406.5**	**403.2**	**0.0227**	57.98	6454.41	46504.29	0.0542
4	**9.9718**	**6388.6**	**29.5**	**0.0243**	34.28	6421.61	253.00	0.0513
5	**8.1479**	**6469.9**	**32.4**	**0.0063**	39.87	6506.50	336.82	0.0216
6	**18.9014**	**10542.3**	**32.7**	**0.0069**	64.18	10574.08	251.63	0.0195
7	**6.2535**	**10973.7**	**1646.5**	**0.0886**	17.38	11008.66	8599.89	0.0978
8	**5.9507**	**26447.2**	**3286.9**	**0.0990**	13.68	26499.88	11627.68	0.1080
9	**8.4296**	**26939.5**	**4598.5**	**0.0965**	19.45	26996.80	13495.29	0.1030
10	**211.4789**	**26758.4**	28463.8	**0.0211**	250.83	26814.96	**28253.26**	0.0322



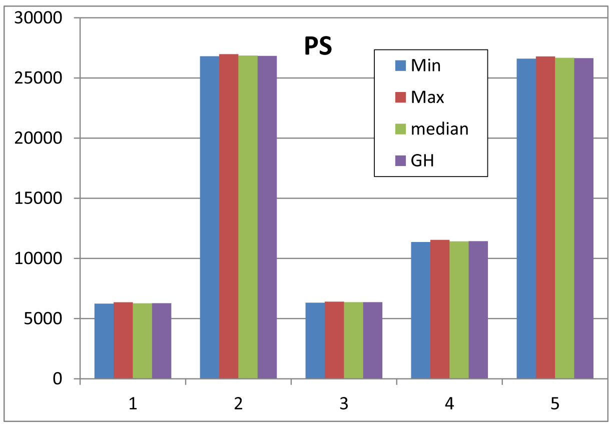


Fig. 4. Quality of GH in Max-SAT and Personnel Scheduling vs other contenders

It is also interesting to know how far our solution was from the winners. It is not easy to answer this question because in the final competition, the instances were randomly chosen. Hence, we did not know which ones were chosen and a direct comparison was impossible. Nevertheless, in the contest the same set of instances were used for all the submitted methods. For example, in Fig. 4 quality of solutions for Max-SAT (SAT) and Personal Scheduling (PS) is shown. Minima, maxima, medians of other contenders and the GH results are shown. For the Max-SAT problem the differences are quite evident, and there was space for improvement. In contrast, in the Personal Scheduling all results were fairly similar, and there were almost no space for improvement.

6. Conclusions

In this section we summarize experience and observations made in the process of developing, tuning, and electing our best hyperheuristic.

We started our endeavor with a set of misconceptions. Probably, they were formed by the earlier experience in the classic heuristic and metaheuristic search. We believed that in general, certain LLHs should perform well on certain problems. Some types of LLHs should behave similarly in all domains. Yet, it turned out to be a misconception. The LLHs of the same type behave completely differently in various domains. Also on one domain, or on one instance but for various current solutions, LLHs have different performance. It applies both to the solution quality and to the runtime. Thus, LLH performance is domain-, instance-, and solution-dependent. This has consequences for reasoning about LLHs, HHs, and HH search. Classifying LLHs by their typical behavior makes no sense, because such classifications are inevitably volatile. Considering the HH concept strictly, like in CHESC competition, as an HH does not "know" which instance in which domain is solved, then the LLHs can perform unpredictably and each instance becomes a *unitary combinatorial optimization problem*. As a result, HHs cannot be preconditioned for efficiently solving any instance in any domain. All the information needed to guide the HH must be collected while solving the actual instance. It might have been different if we knew which kind of domain was being solved, and which the particular LLH operated.

An HH can be understood as three key components: architecture, control parameters, and the guiding AI algorithm. Examples of architecture are genetic algorithm (GA), tabu search (TS), simulated annealing (SA), 5Ph, GH. As the classic metaheuristics (GA, TS, SA) architecture dictates operations in the space of solutions, the HH architecture dictates what operations can be done in the space of LLHs. Control parameters of the classic metaheuristics are set in the tuning process. In HH search, setting control parameters is much more difficult, especially assuming a strict approach as to a unitary search problem. Tuning control parameters of an HH requires a good machine learning algorithm (the guiding AI), as well as a lot of time. Yet, this resource was very scarce in CHESC competition. An HH with complex control architecture, like 5Ph, needs a lot of time to be tuned (either manually, or automatically). On the contrary, simple HHs should be easier to control and self-tune. Hence, it seems a reasonable idea to start study of HH search from very simple architectures. There is one more, humorous, reason for starting future research in HH from something rudimentary. In our tests, sometimes a wrong implementation of a certain idea resulted in better performance than the correct implementation embodying the desired idea.

There are also a number of technical observations resulting from the use of HyFlex. One of the issues in HyFlex is that some LLHs have input parameters. Some of them have strong impact on LLH performance. For example, some 'depth of search' parameters control depth of search in exponential time algorithms. Controlling LLH input parameters again requires adequate methods. In our implementation, this issue was not satisfactorily addressed because each pair of an LLH and its parameter was treated as a unique LLH.

To avoid being trapped in one bad solution path, we introduced parallel search threads in 3Ph ... 5Ph. It turned out inefficient: only a few threads produced interesting solutions, the same solution was often constructed in many threads. On the contrary, limited parallelism of search in GH was more successful. It means that without diversification of the search paths, parallelism in itself is not helpful. Since the LLHs did not provide internally sufficient diversity of search, we turned our attention to more diverse use of the LLHs. Consequently the number of threads decreased from over 100 in 3Ph to 3–7 in 5Ph. Thus, brute force of massive parallelism by itself does not provide real advantage in HH search. Moderate parallelism with other mechanisms to breed diversity may be helpful.

In the above discussion we used concepts of search stagnation, diversification, in rather intuitive manner. These concepts, inevitably, had their instantiation in our HHs. It may be argued that they could be implemented differently, and hopefully better. However, defining them in domain-, LLH-independent way is difficult. To conclude we could say that constructing effective hyperheuristic working in the above introduced strict sense is a hard task. Only future may tell if this idea finally turns successful.

Acknowledgements. Research partially supported by the Polish National Science Center Grant No. NN519643340.

REFERENCES

[1] J. Blazewicz, E. Pesch, M. Sterna, and F. Werner, "Metaheuristic approaches for the two-machine flow-shop problem with weighted late work criterion and common due date", *Computers & Operations Research* 35, 574–599 (2008).

[2] J. Blazewicz, W. Domschke, and E. Pesch, "The job shop scheduling problem: Conventional and new solution techniques", *Eur. J. Operational Research* 93, 1–33 (1996).

[3] K.S. Hindi and E. Toczylowski, "Detailed scheduling of batch production in a cell with parallel facilities and common renewable resources", *Computers and Industrial Engineering* 28, 839–850 (1995).

[4] P. Jantos, D. Grzechca, and J. Rutkowski, "Evolutionary algorithms for global parametric fault diagnosis in analogue integrated circuits", *Bull. Pol. Ac.: Tech.* 60 (1), 133–142 (2012).

[5] S. Dinu and G. Bordea, "A new genetic approach for transport network design and optimization", *Bull. Pol. Ac.: Tech.* 59 (3), 263–272 (2011).

[6] P. Rohlfshagen and J. Bullinaria, "A genetic algorithm with exon shuffling crossover for hard bin packing problems", *Proc. 9th Annual Conf. on Genetic and Evolutionary Computation GECCO'07* 1, 1365–1371 (2007).

[7] M. Kaleta and E. Toczylowski, "Restriction techniques for the unit-commitment problem with total procurement costs", *Energy Policy* 36 (7), 2439–2448 (2008).

[8] M. Hyde, G. Ochoa, and A.Parkes, "Cross-domain heuristic search challenge", http://www.asap.cs.nott.ac.uk/chesc2011/ (2011).

[9] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, and E.K. Burke, "HyFlex: a benchmark framework for cross-domain heuristic search", *Eur. Conf. on Evolutionary Computation in Combinatorial Optimisation (EvoCOP 2012), Lecture Notes on Computing Science* 7245, 136–147 (2012).

[10] M.Hyde and G. Ochoa, "ASAP Default Hyper-heuristics", http://www.asap.cs.nott.ac.uk/chesc2011/defaulthh.html (2011).

[11] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M.Zaidi, "The bees algorithm", *Manufacturing Engineering Centre*, Cardiff University, Cardiff, 2005.

[12] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments", *Adaptive and Multilevel Metaheuristics*, SCI 136, 3–29 (2008).