Five Phase and Genetic Hive Hyper-heuristics for the Cross-Domain Search

Tomasz Cichowicz, Maciej Drozdowski, Michał Frankiewicz, Grzegorz Pawlak, Filip Rytwiński, and Jacek Wasilewski

Institute of Computing Science, Poznan University of Technology, Poland
{Tomasz.Cichowicz,Michal.Frankiewicz,Filip.Rytwinski,
Jacek.Wasilewski}@student.put.poznan.pl
{Maciej.Drozdowski,Grzegorz.Pawlak}@cs.put.poznan.pl

Abstract. In this paper we present two hyper-heuristics: Five Phase Approach (5Ph) and Genetic Hive (GH), developed for the Cross-Domain Heuristic Search Challenge held in 2011. Performance of both methods is studied. Experience gained in construction of the hyper-heuristics is presented. Conclusions and recommendations for the future advancement of hyper-heuristic methodologies are discussed.

Keywords: hyper-heuristics, cross-domain heuristic search, HyFlex.

1 Introduction

Hyper-heuristics (HH) are supposed to bring a new quality to solving hard combinatorial problems. Instead of directly searching the space of various combinatorial optimization problems, hyper-heuristics explore the space of low level heuristics (LLHs). The LLHs perform moves in the space of solutions of a ground combinatorial optimization problem similarly to the classic local search methods. Thus, LLHs serve as an interface between the problem *domain* and the guiding algorithm of a hyper-heuristic. This approach has a potential advantage of automating construction and tuning of algorithms. That allows solution of a broad range of combinatorial problems (domains). Still, this general concept to be fruitful needs considering of at least two issues: Which LLHs concepts are general enough to be implemented in every domain, and how can they be controlled?

The first issue has been tackled in the HyFlex framework [1]. HyFlex is a Java library implementing four LLH types on six domains. The LLH types are: local search heuristics, mutational heuristics, ruin-recreate heuristics, crossover heuristics. The domains were: maximum satisfiability (Max-SAT), bin packing, flowshop (FS), personnel scheduling (PS), and later also traveling salesman problem (TSP), vehicle routing problem (VRP) [4]. HyFlex maintains a population of solutions initialized by randomized constructive heuristics. The objective functions are uniformly minimized in all domains. Some of the LLHs have additional parameters controlling, i.e. depth of search, intensity of mutation.

In this paper we report on two hyper-heuristics developed by a CS-PUT team of students and researchers of the Institute of Computing Science, Poznan University of Technology. Two methods were independently developed: Five Phase (5Ph) and Genetic Hive (GH).

2 Five Phase Approach

The idea was to build an algorithm which iteratively goes through three main phases: *intensification*, *stagnation*, and *diversification*. Moreover, to avoid getting stuck in some bad solution, the algorithm should work on a number of solutions in parallel applying also the *mutation* and *crossingover* phases (see Figure 1).

Solution Streams Initialization. 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.

LLH classification. The classification algorithm ran an LLH in its thread (solution stream) a predetermined number of times to collect the statistics. A linear regression was used to calculate the slope a_i of the linear approximation of the objective function in the repetition count, for each LLH i. The duration Δ_i of the classification period for LLH i was also recorded. The score of LLH i was $stat_i = -a_i/\Delta_i$. The LLH with $stat_i > 0$ was labeled as an *improver*, with $stat_i < -0.2$ LLH was classified as *masher*.

Intensification. A random LLH from the triplet(triple cluster of LLHs instead of just singleton LLHs) was applied in the solution. Probability of selecting the LLH was proportional to its score. The scores were calculated on the basis of statistics collected while running the thread. For each LLH i recent improvement of the objective function value ϕ_i ($\phi_i > 0$ means improvement), and execution time δ_i were recorded. The score of LLH i was $score_i = score_i * e^{\phi_i/\delta_i}$, where initial $score_i = 1$ - not selected firstly, might have been dominated and eliminated due to quickly growing score of just one LLH. To counter such effect, the LLHs that were not applied so far, had their score increased by 5% with each execution of any LLH in the thread. The selection of LLH and its run to stagnation was repeated $NoIt = \lceil gs/3+3 \rceil$ times, where gs is the number of global stagnations.

Stagnation. In the stagnation state, the improvement in the objective function stalled. It had been defined as a situation in which the solution did not improve in a number of consecutive iterations of LLH. Global stagnation phase occurred when, after 3 global iterations, the best objective function value was not changed.

Diversification. In this phase masher LLHs were chosen randomly and applied for a predetermined time period. It was proposed to use short clusters - triplet- of LLHs on the solutions instead of just singleton LLHs. The architecture of 5Ph is depicted in Figure 1.

Triplet Mutation. After the intensification phase LLH triples were mutated. For each thread the algorithm of LLH mutation proceeded in two steps: 1. Randomly selecting a triplet from some other thread. Probability of drawing a triplet

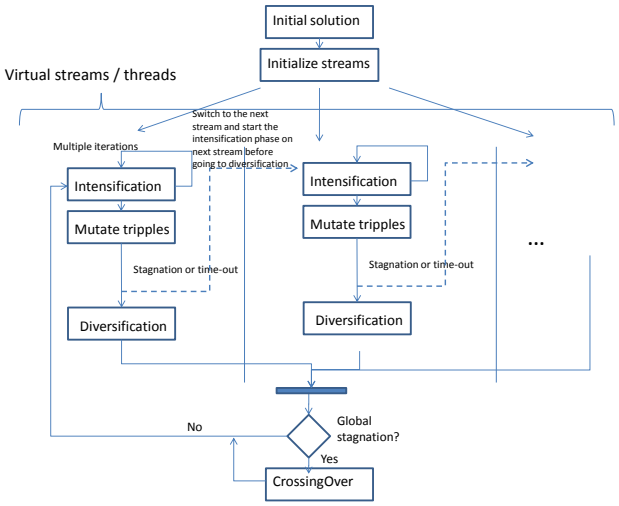

Fig. 1. Architecture of 5Ph algorithm

was proportional to the mean value of the triplets LLH scores; 2. Replacing the worst LLH in the current thread with the best LLH from the drawn one.

Solution Crossingover. If the state of global stagnation was reached, 5Ph applied a random LLH of the crossover class on the solutions from the threads. The two solutions to the crossover were selected randomly - the first solution with the probability proportional to the quality of the solution and the second with the probability inversely proportional to the solution quality. At the end of this phase, the global stagnation counter gs was reset to 0, and 5Ph restarted in the intensification phase.

3 Genetic Hive Algorithm

This algorithm consisted in parallel search of the solution space using evolving sequences of low level heuristics. It was inspired by the Bees Algorithm presented in [2] and genetic algorithms mentioned in [3] imitating the behavior of bees searching for food.

In the algorithm, bees correspond with the LLH sequences. They will be labeled as agents. Searching for locations correspond with searching for current problem solutions. The agents (bees) in the hive remain passive, while the agents outside attempt to improve the current solutions. Thus, this algorithm is a combination of evolutionary approach and simulation of agent colony searching for resources. Let us denote by H the set of all agents, by L the set of active agents (outside the hive), and by $B \subseteq L$ the set of agents continuing their search in their locations.

```
GHHH-SEARCH()
 1  INITSEARCH()
 2  while timeSpent < timeLimit
 3        do
 4           for every agent a ∈ L
 5                 do EVALUATE-AGENT(a)
 6           ▷ Select agents with the best scores that stay in search locations
 7           B ← bSize best agents from set L
 8           ▷ Select additional agents that stay in hive to simulate partial extinction
 9           S ← sSize random agents from set H \ B
10           ▷ Evolve new agents and update hive with them
11           O ← EVOLVE(B,oSize)
12           H ← B ∪ S ∪ O
13           ▷ Assign random agents from hive to free search locations
14           L ← B ∪((lSize − bSize) random agents from set H − B);
15           timeSpent ← currentTime − startTime
16           End.
```

where: *hSize* - population size - the total number of agents, *lSize* - number of search locations, *bSize* - number of agents staying in their search locations, *sSize* - number of additional agents that don't evolve, *oSize* - number of agents replaced by the offspring, size of the agent - number of LLHs in agent definition, probability of mutation.

4 Computational experiments

The aforementioned GH and 5Ph hyper-heuristics, as well as some other hyper-heuristics created by the team were subject to a number of tests comparing their performance. The results of that comparisons are summarized in Table 1. The first two are ad hoc methods. *RND LLH* - for each of 10 parallel solutions

Table 1. Pairwise comparison of CS-Put HHs

	Hyper Heuristics	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	GenHiv-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	GenHiv-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	GenHiv-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

randomly choose LLH and apply it to a solution. *Each LLH* - means executing each LLH and choosing the best solution. Further suffixes denote: LS means using *local search*, LLHs in the intensification phase *3pl* means that the triplets of LLHs were introduced. The particular method can be executed with the different parameters settings. The values presented in the table are the numbers of wins against each other. The total number of evaluated instances was 40. For some entries, the total number of wins is smaller than 40. It means that there were ties and that both methods gave the same solution. In Table 2, the two final HH

Table 2. Comparison results for 5Ph and GH - No. of wins out of 10 instances

SAT	FS	PS	BP	SAT	FS	PS	BP
5Ph				**GH**			
Minimum							
1	1	1	2	2	8	9	6
Median							
8	0	4	7	2	10	4	3
Average							
10	10	9	10	0	0	0	0

of CS-PUT team are compared on all the HyFlex instances. Minima, medians and averages of the objective function are presented. A value presents number of wins for particular method and instance out of 10 evaluated instances. From the results gathered in the Table 2 one can conclude that GH method is generally better, according to the competition rules, than 5Ph. On contrary, 5Ph is better in averages and there is a tie in medians. Hence, distributions of the results are different. This demonstrates that a single performance index may be insufficient to show the complexity of the results.

5 Conclusions

Controlling the search process is a complex problem. One of the difficulties was to decide when to stop applying the current LLH. If the current LLH exhausts its potential for improving the current solution the search arrives in the state of "stagnation". However, simple statistical methods of detecting stagnation, we applied, were insufficient to quickly discover that the search had already stalled. Similar difficulty is guiding the diversification process to move the solution away from the local minimum. On one hand, it is necessary to leave quickly the current part of the solution space, on the other hand, the good parts of the current solution should be preserved to avoid rebuilding the solution from the scratch. We attempted to improve the performance of 5Ph by diversifying the population of the solutions supplied to the crossover: some of the solutions were the local minima, some were random, and some were the worst solutions visited so far.

Still, the results were not satisfactory and in the tunning the number of crossovers gradually decreased to 1. It seems that convergence of crossover was too slow for the rules of the competition.

A HH, applying certain types of LLH, iteratively performed well in one domain, but was unsuccessful in other domains. This suggests that the sequence of the LLH types should be broken, and the type of LLH currently applied should be varied. To avoid being trapped in a bad solution path, we introduced "parallel" search threads in 5Ph. In the course of tunning, the number of threads decreased from over 100 to 3-7. It can be noted that the chance of avoiding being stacked up by parallelism is not bigger than by the use of other diversification tools. Thus, massive parallelism does not provide real advantage in HH search.

Performance of LLHs is domain-, instance-, and solution-dependent. This has consequences in reasoning about LLHs and HHs: Classifying LLHs by their average behavior makes no sense, and such classifications cannot be applied to guide HHs. Since, a HH does not "know" what instance in what domain is solved and the LLHs can perform so unpredictably, each instance becomes a unitary combinatorial optimization problem. Consequently, unless the domain is fixed and the instances are similar, HHs cannot be preconditioned for efficient solving of any instance in any domain. All the information fed to the AI of HH must be collected while solving the actual instance.

The performance of hyper-heuristic search has at least two criteria: time and quality of solutions. However, the issue of time was more involved. The HH should be perceived as combination of three elements: architecture, control parameters, and guiding algorithm. The architecture dictates the mode of LLH usage. The architecture can be, i.e., Tabu Search, Memetic Algorithm, 5Ph, GH, etc. In the classic meta-heuristics, the control parameters are set in the tuning process. Consequently, HH with sophisticated architecture and a lot of control parameters (as in 5Ph) need more time and more data to tune to the solved instances. This leads to a conclusion that for better understanding of HH search, it would be more advantageous to start with a rudimentary HH.

Acknowledgments. Research partially supported by Polish National Science Center (No. 519 643340)

References

1. Burke, E., Curtois, T., Hyde, M., Ochoa, G., Vazquez-Rodriguez, J.A.: HyFlex: A Benchmark Framework for Cross-domain Heuristic Search. ArXiv e-prints (2011), http://adsabs.harvard.edu/abs/2011arXiv1107.5462B,
2. Pham DT, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M.: The Bees Algorithm. Manufacturing Engineering Centre, Cardiff University, UK, (2005)
3. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent Developments. In: C. Cotta et al. (Eds.): Adaptive and Multilevel Metaheuristics, SCI, vol. 136, pp. 3–29. Springer, Heidelberg (2008)
4. Hyde, M., Ochoa, G., Parkes, A.: Cross-domain Heuristic Search Challenge. (2011) http://www.asap.cs.nott.ac.uk/chesc2011/
5. Hyde, M., Ochoa, G.: ASAP Default Hyper-heuristics. (2011) http://www.asap.cs.nott.ac.uk/chesc2011/defaulthh.html