A new Hyperheuristic Algorithm for Cross Domain Search Problems

Andreas Lehrbaum and Nysret Musliu

Vienna University of Technology, Database and Artificial Intelligence Group
{lehrbaum, musliu}@dbai.tuwien.ac.at

Abstract. This paper describes a new hyperheuristic algorithm that performs well over a variety of different problem classes. A novel method for switching between working on a single solution and a pool of solutions is proposed. This method is combined with an adaptive strategy that guides the selection of the underlying low-level heuristics throughout the search. The algorithm was implemented based on the HyFlex framework and was submitted as a candidate for the Cross-Domain Heuristic Search Challenge 2011.

1 Introduction

Hyperheuristics (as introduced in [1]), are a way to incorporate existing problem-class specific simple low-level heuristics into a higher-level search strategy, which schedules and guides their execution. The main idea is that an ensemble of heuristics orchestrated by a top-level strategy is able to perform better on average at solving a wide range of problems, than any of the underlying heuristics alone. A good survey of existing hyperheuristic techniques is given in [2]. The HyFlex [3] framework offers an intuitive interface to utilise a set of given low-level search and mutation heuristics, easing the task of working purely on a high-level search strategy without any a priori knowledge about the problem instance or the heuristics available.

In this paper, we propose a hyperheuristic algorithm that consists of two distinct phases and uses a quality based selection strategy for the local-search heuristics using feedback from the search progress. The implementation is tailored towards the interface of the HyFlex framework and the specific requirements of the CHeSC competition rules[1]. Development and testing was performed on 4 different problem domains: Boolean Maximum Satisfiability, One-dimensional Bin Packing, Personnel Scheduling and Permutation Flow Shop. The provided low-level heuristics were used, implementing standard heuristic search and mutation operations for each of the domains.

[1] http://www.asap.cs.nott.ac.uk/chesc2011/rules.html

2 Algorithm Description

2.1 Overview

Following the classification of Burke et al. [2], our algorithm is an online learning hyperheuristic, working mainly on heuristic selection. The novelty of the proposed approach lies in the repeated switching between two search variants, namely a serial search phase working only on a single solution and a systematic parallel search phase working with a set of different solutions at the same time. We further propose a grading mechanism for the ordering of the low-level heuristics and a selection strategy for the available mutation heuristics.

After an initialisation phase, in which the preliminary scores of the available heuristics are determined, the search continues by systematically executing the local-search heuristics in order of their respective quality scores. The splitting in a serial and a parallel search phase balances the focus of the search between exploration of new parts of the search space and the exploitation of the quality of the currently best working solution. Throughout the search process, the performance and runtime characteristics of the low-level heuristics are measured and the scores responsible for their selection are updated accordingly. In addition to that, the overall search progress is monitored continuously and mechanisms such as the temporary blocking of ineffective heuristics or the restart of the algorithm from the last best solution are applied. A global tabu-list in form of a ringbuffer which always contains the last 40 visited distinct solutions is used to avoid cycles and to prune already explored branches of the search tree. If the tabu list already contains 40 elements, the next solution replaces the oldest entry. The fixed size of the tabu-list was chosen based on experimental data to balance memory and runtime requirements with the effectiveness of the tabu mechanism.

2.2 Detailed Description of the Algorithm

Initialisation Phase: The algorithm begins with calculating preliminary quality-scores of all the local-search heuristics available for the given problem instance. It does this by initialising the first solution and applying the heuristics in turn, recording their gain and their required runtime. The heuristics are called with the highest possible parameter settings for the depth of search and the intensity of the mutations (where applicable). The initial quality-score for each heuristic is calculated as gain per runtime. The best solution found so far is returned as the working solution for the subsequent phase.

Serial Search Phase: The available local-search heuristics $LS = \{ls_1, \ldots, ls_n\}$ are applied sequentially, in order of decreasing quality, to the current working solution s_{work}. After each application of a heuristic, the fitness function $f(s)$ of the resulting solution is evaluated. If the fitness value is better, the solution is accepted as the next working solution. If the local search heuristic resulted in a different solution with the same fitness value then it is accepted as well. Additionally, a global tabu-list T keeps track of the last 40 distinct solutions encountered and prohibits the acceptance of a solution that is already on the

tabu-list. In case a new working solution was accepted, the search continues with the application of the best local search heuristic, otherwise the next best heuristic is chosen from the set LS. The parameters for the depth of the search and the intensity of the mutation are set to random values before the application of each heuristic. This serial search phase ends whenever no further improvement could be found with all the available heuristics therefore resulting in a locally optimal solution. See algorithm 1 for a pseudocode implementation.

Algorithm 1: Serial search working on a single solution s_{work}

```
1:  for i = 1 → |LS| do
2:    setDepthOfSearch(random(0 ... 1))
3:    setIntensityOfMutation(random(0 ... 1))
4:    s_temp ← applyHeuristic(ls_i, s_work)
5:    if f(s_temp) < f(s_work) or (f(s_temp) = f(s_work) and s_work ≠ s_temp)
      then
6:      if s_temp ∉ T then
7:        s_work ← s_temp
8:        T ← T ∪ s_work
9:        i ← 1
10:     end if
11:   end if
12: end for
13: return s_work
```

Quality Updates: In fixed time intervals of 5 seconds (or after every call to a heuristic, in case it takes longer to complete), the qualities of the local-search heuristics are updated to reflect their performance during the whole search process. This can result in the re-ordering of the search sequence of the heuristics. The quality metric is calculated as the number of times the heuristic resulted in an accepted solution divided by the total runtime of this heuristic so far. The time interval between updates was determined experimentally for the CHeSC setting to balance the cost of the updates and the possible gain of reordering.

Generation of Mutated Solutions: The available mutation and ruin-recreate heuristics are placed in a roulette-wheel reflecting their relative performance. Initially all mutation heuristics have the same chance of being selected. The performance of a mutation heuristic is judged based on the solution quality of the mutated solution after the subsequent serial search phase has finished. This method is used in order to assess the likelihood of a given mutation heuristic to result in an improvement during the further course of the search. Given the current working solution as input, 7 mutated offsprings are generated by applying a set of mutation heuristics chosen by the roulette-wheel process. The mutation intensity and search depth parameters are again set to a new random value each time before the mutation heuristics are applied. However, the probability of strong mutations is lowered towards the end of the search.

Algorithm 2: Parallel search working on a set of solutions $s_{1...7}$

```
1: for i = 1 → 7 do
2:    h_i ← 1 // set working heuristic index h_i to the best LS heuristic
3: end for
4: repeat
5:    candidatesLeft ← false
6:    for i = 1 → 7 do
7:       if h_i ≤ |LS| then
8:          setDepthOfSearch(random(0 ... 1))
9:          setIntensityOfMutation(random(0 ... 1))
10:         s_temp ← applyHeuristic(ls_{h_i}, s_i)
11:         if f(s_temp) < f(s_best) then
12:            s_best ← s_temp
13:            T ← T ∪ s_temp
14:            return s_best
15:         else
16:            candidatesLeft ← true
17:            if f(s_temp) < f(s_i) then
18:               s_i ← s_temp
19:               T ← T ∪ s_temp
20:               h_i ← 1 // continue with best LS heuristic
21:            else
22:               h_i ← h_i + 1 // continue with the next best LS heuristic
23:            end if
24:         end if
25:      end if
26:   end for
27: until candidatesLeft = false
28: return SELECTSOLUTION(s_{1...7})
```

Parallel Search Phase: Starting from the set of 7 mutated solutions, the parallel search phase begins to work on the candidate solutions one after another. It begins with applying the best local-search heuristic to the first candidate. If an improvement is found, the new solution is accepted, otherwise it is discarded. Afterwards the search continues with the next candidate solution and the local-search heuristic scheduled for this solution. Whenever a global improvement is found (i.e. the result is better than the currently best found solution so far), it is immediately accepted as the working solution for the next serial search phase and the parallel search is aborted. Otherwise the search goes on until all solutions have reached a local optimum with respect to all available local-search heuristics. See algorithm 2 for a pseudocode implementation.

Working Solution Selection: Assuming no global improvement was found during the parallel search phase, a solution is selected from the pool of solutions containing the locally optimal output from the serial search phase as well as the parallel search phase. Solutions with a better quality have higher probability of being selected. If all solutions in the set are contained in the global tabu-list,

a random mutation will be applied to the best solution until the result is not contained in the buffer anymore.

Excluding Inefficient Heuristics: If a local-search heuristic is found to be ineffective (determined by a low count of successful applications) it is excluded for a single iteration from the search process in both the serial and the parallel phase with a 50% chance.

Restarting the Search: Whenever the search continues for a certain amount of time (10% of the available runtime), producing only solutions which are worse than a preset threshold (130% of the so far best solution), the search continues with the generation of mutated solutions from the currently best solution. This prevents the search process from slowly generating ever worse solutions and wandering too far away from the best candidate found so far.

3 Results

Tables 1 and 2 show the final results of the 20 participating teams (with our algorithm named HAHA), as published by the organizers of the CHeSC competition[2]. The values represent the median resulting function value per instance of 31 subsequent runs with different random seeds and 600 seconds CPU runtime. All problem instances were minimisation tasks and selected by the organisers of the competition. The rank column denotes the final overall rank according to the official scoring system and the best value in each column is marked bold.

Rank	Algorithm	MS_1	MS_2	MS_3	MS_4	MS_5	BP_1	BP_2	BP_3	BP_4	BP_5	PS_1	PS_2	PS_3	PS_4	PS_5
1	AdapHH	**3**	5	**2**	3	8	**0.01607**	0.00360	**0.00356**	**0.10828**	**0.00354**	24	9667	3289	1765	325
2	VNS-TW	**3**	**3**	**2**	**3**	10	0.03696	0.00715	0.01671	0.10878	0.02776	19	9628	**3223**	1590	320
3	ML	5	10	3	9	8	0.04214	0.00753	0.01456	0.10852	0.02182	**18**	9812	3228	1605	**315**
4	PHUNTER	5	11	4	9	8	0.04787	0.00360	0.02012	0.10908	0.03948	25	10136	3255	1595	320
5	EPH	7	11	6	14	13	0.05042	0.00360	0.01127	0.10866	0.02238	22	10074	3232	1615	345
6	HAHA	**3**	4	**2**	5	8	0.08829	0.00726	0.01450	0.11023	0.02790	21	9666	3236	**1558**	335
7	NAHH	8	10	4	9	**7**	0.05504	0.00347	0.00473	0.10878	0.00554	27	9827	3246	1644	345
8	ISEA	5	11	4	9	11	0.03422	**0.00328**	0.00365	0.10862	0.00640	20	9966	3308	1660	**315**
9	KSATS	4	7	**2**	4	9	0.01923	0.00780	0.01149	0.10892	0.02199	22	9681	3241	1640	355
10	HAEA	6	12	5	12	11	0.04522	0.00363	0.01379	0.10873	0.02400	25	9795	3266	1699	345
11	ACO-HH	11	35	9	17	13	0.04771	0.00320	0.00388	0.10986	0.01486	26	11212	3346	1760	355
12	GenHive	16	44	31	19	14	0.02994	0.00708	0.01037	0.10859	0.02286	21	12708	3274	1727	330
13	DynILS	23	56	37	31	19	0.04027	0.00767	0.01016	0.10872	0.01285	33	9893	3324	1870	465
14	SA-ILS	13	23	12	15	9	0.07873	0.01153	0.01458	0.11039	0.02958	20	9750	3228	1625	340
15	XCJ	6	8	5	9	10	0.02201	0.01145	0.01569	0.10856	0.02850	30	33390	3277	1658	380
16	AVEGNep	8	10	5	9	**7**	0.08737	0.00773	0.01807	0.11139	0.03750	26	10230	3283	1765	360
17	GISS	16	21	13	17	9	0.06917	0.00837	0.03218	0.11259	0.05922	25	**9625**	3294	1785	370
18	SelfS	13	36	14	14	10	0.06642	0.00736	0.01441	0.10968	0.02391	26	9803	3249	1635	350
19	MCHH-S	8	14	8	8	9	0.06225	0.00729	0.01459	0.10976	0.02861	32	13297	3344	1785	370
20	Ant-Q	23	52	38	27	14	0.04909	0.01650	0.02102	0.10990	0.03765	33	73535	3348	1970	425

Table 1: Median results for the Max-SAT ($MS_{1...5}$), Bin Packing ($BP_{1...5}$) and Personnel Scheduling ($PS_{1...5}$) problem instances

4 Conclusion

Our algorithm was ranked 6th (out of 20 teams) in the final CHeSC competition. The good results at the completely different domains of Max-SAT and Person-

[2] `http://www.asap.cs.nott.ac.uk/chesc2011/results.html`

Rank	Algorithm	FS_1	FS_2	FS_3	FS_4	FS_5	TSP_1	TSP_2	TSP_3	TSP_4	TSP_5	VRP_1	VRP_2	VRP_3	VRP_4	VRP_5
1	AdapHH	**6240**	26814	6326	**11359**	26643	**48194**	**20822145**	**6810**	66879	53099	60900	13347	148516	20656	148689
2	VNS-TW	6251	26803	6328	11376	**26602**	**48194**	21042675	6819	67378	**54028**	76147	13367	148206	21642	149132
3	ML	6245	**26800**	**6323**	11384	26610	**48194**	21093828	6820	66893	54368	80671	13329	**145333**	20654	148975
4	PHUNTER	6253	26858	6350	11388	26677	**48194**	21246427	6813	67136	52934	64717	**12290**	146944	**20650**	148658
5	EPH	6250	26816	6347	11397	26640	**48194**	21064606	6811	**66756**	52925	74715	13335	162188	**20650**	155224
6	HAHA	6269	26850	6353	11419	26663	48414	21291914	6917	69324	56039	65498	13317	155941	20654	148655
7	NAHH	6245	26885	**6323**	11383	26671	**48194**	20971771	6841	67418	53097	65398	13358	157242	20654	152081
8	ISEA	6262	26844	6366	11419	26663	**48194**	20868203	6832	67282	54129	70471	13339	149149	20657	150474
9	KSATS	6292	26860	6366	11466	26683	48578	21557455	6947	72027	58738	64495	13296	156577	20655	**147124**
10	HAEA	6261	26826	6353	11408	26651	**48194**	20925949	6824	67488	54144	**60608**	13342	146951	20655	147283
11	ACO-HH	6249	26904	6353	11393	26724	48200	21137472	6851	67202	53428	73348	14371	149672	21663	151610
12	GenHive	6279	26835	6366	11434	26648	48271	21083157	6868	67236	56022	67475	13353	167297	20718	147960
13	DynILS	6269	26875	6365	11419	26670	**48194**	20987358	6823	67308	54100	69798	14359	149869	21654	150060
14	SA-ILS	6336	26886	6390	11514	26703	49046	21281226	6994	70614	57607	64185	13390	162642	20667	152271
15	XCJ	6271	26910	6366	11481	26710	48412	21162559	6884	68005	54967	63654	13354	152321	20658	153110
16	AVEGNep	6322	26952	6379	11507	26743	48639	21520601	6969	70194	57998	77884	12397	184710	20655	166742
17	GISS	6329	26979	6385	11516	26758	49010	21651052	7001	72630	59804	61580	13352	162266	20657	149590
18	SelfS	6287	26859	6369	11443	26678	49043	21040810	6984	69646	56647	73894	14386	203667	20687	153590
19	MCHH-S	6336	26937	6397	11527	26716	49412	21504030	6997	70685	57836	72005	13534	207891	20850	160303
20	Ant-Q	6358	26971	6407	11545	26792	49613	21277953	7016	69987	55314	76678	14382	193827	21656	160684

Table 2: Median results for the Flow Shop ($FS_{1...5}$), Travelling Salesman ($TSP_{1...5}$) and Vehicle Routing ($VRP_{1...5}$) problem instances

nel Scheduling indicate a rather good general problem solving capability. The algorithm has however some shortcomings in domains where different fast local search heuristics all result in small improvements most of the time (like the tested Bin Packing instances). It seems that the rather strong bias for the local-search heuristic with the best quality score can be both a strength and a weakness, indicating that a more adaptive process could be advantageous. More extensive study is needed to assess the further potential of this algorithmic approach.

Acknowledgments: The research herein is partially conducted within the competence network Softnet Austria II (www.soft-net.at,COMETK-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT). Additionally, this work was partially supported by the Austrian Science Fund (FWF): P20704-N18.

References

1. P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," *Practice and Theory of Automated Timetabling III*, pp. 176–190, 2001.
2. E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "A survey of hyper-heuristics," *Computer Science Technical Report No. NOTTCS-TR-SUB-0906241418-2747, School of Computer Science and Information Technology, University of Nottingham*, 2009.
3. E. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, and J. Antonio, "HyFlex: A Flexible Framework for the Design and Analysis of Hyper-heuristics," in *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland*, 2009, p. 790.