A Benchmark Set Extension and Comparative Study for the HyFlex Framework.

Steven Adriaensen, Gabriela Ochoa and Ann Nowé

***Abstract*—In this work we conduct a comparative study of several publicly available, state-of-the-art hyper-heuristics for HyFlex in order to assess their generality across domains. To this purpose we extend the HyFlex benchmark set with 3 new problem domains: The 0-1 Knap Sack, Quadratic Assignment and Max-Cut Problem. To our knowledge, this is the first public extension of the benchmark since the CHeSC 2011 competition. In addition, this is the first study testing the Fair-Share Iterated Local Search (FS-ILS) method, designed in prior research, using a semi-automated design approach, on new unseen problem domains. We show that, of the methods compared, Adap-HH (CHeSC 2011 winner) clearly perfoms the most consistently, overall. In addition, we identify a weakness of, as well as a way to further simplify the FS-ILS method. Finally, we found that, overall, the state-of-the-art methods compared, generalized much better than a naive baseline.**

I. Introduction

Many interesting combinatorial optimization problems cannot be solved in polynomial time, i.e. they are NP-hard. A classical example is the Traveling Salesman Problem. Luckily, in practice, optimal solutions are often not required and non-exact methods can be used instead. One approach that recently received a lot of attention are the so called meta-heuristic optimization methods [1], [2]. These methods often quickly find good solutions to large and otherwise intractable problems, by iteratively trying to improve a (set of) candidate solution(s).

In recent history, most research in this area has been presented in a problem-specific way. From a theoretical point of view this approach was motivated by the *No Free Lunch Theorem* [3], stating that on average, performance over all instances is the same for every method, and therefore advocating a made-to-measure approach. From a practical perspective, benchmarking accross multiple domains is a challenging endeavor. To do so, we need implementations for all these domains, and to set up a meaningful comparison we also require benchmark instances and solutions that can serve as a baseline. As a consequence of this problem-specific approach, existing methods are not readily applied to newly encountered problems, or even new instances of the same problem. Furthermore, the cost of developing and applying made-to-measure meta-heuristic solutions is high, resulting in only few practical applications.

Steven Adriaensen and Ann Nowé are with the Department of Computer Science, Vrije Universiteit Brussel, Elsene, Brussels, Belgium (email: steven.adriaensen@vub.ac.be).

Gabriela Ochoa is a Lecturer of Computer Science at the University of Stirling, Stirling, Scotland, UK.

Steven Adriaensen is funded by a Ph.D grant of the Research Foundation Flanders (FWO).

Therefore, recently, there is a renewed interest in more general methods. Rather than attempting to outperform made-to-measure methods, these methods provide a cheap off-the-peg alternative. A popular approach in this renaissance are hyper-heuristics [4]. A hyper-heuristic combines a given set of low level heuristics, to solve a given problem instance, in a problem independent way. Here, low level heuristics are domain-specific construction, perturbation and recombination heuristics. A hyper-heuristic can be applied to any domain, given a set of low level heuristics is provided for it first. While the potential of hyper-heuristics in more general, problem-independent search was recognized early on[1] [5], due to the aforementioned practical challenges, the empirical analysis of most hyper-heuristic methods focused on a comparison of peak-performance, on a single domain. The creation of the HyFlex framework [6] in 2010, built to support the CHeSC 2011 competition, has greatly simplified cross-domain benchmarking and has been used in the implementation of many hyper-heuristics ever since. Not only does it provide problem-specific components and benchmark instances for 6 different combinatorial optimization problems, the results obtained by the competition's 20 contestants, serve as a good baseline for comparisons.

To fairly assess a method's generality, however, it is important to test it on as many *new* domains as possible, as the potential variability in domains is vast [7]. Here the word *new* is crucial, i.e. the method/domain should not have been tested using this domain/method during any phase of its design, as otherwise we risk over-fitting. Even more so, since generative methods are being used to design hyper-heuristics [8], [9]. During the original CHeSC competition this was taken into account by making only 4 out of the 6 domains available before the competition. In later work, however, it is often unclear whether such separation is considered.

In this paper we perform a comparative study of several publicly available hyper-heuristics for HyFlex in order to assess their generality across domains. To this purpose we extend the HyFlex benchmark set, providing benchmark instances and problem specific components for 3 additional problem domains.[2] To the authors knowledge, this is, to date, the first public extension of the HyFlex benchmark since the CHeSC 2011 competition. In addition, this is the first study testing the Fair-Share Iterated Local Search (FS-ILS) method [10], designed using a semi-automated design approach, on new (unseen) problem domains.

[1]Note that the No Free Lunch Theorem is no argument against hyper-heuristic methods as on every problem domain, the domain-specific aspects of the method (low level heuristics) differ, i.e. a different method is used.

[2]Available here https://github.com/Steven-Adriaensen/hyflext (domains)

The remainder of the paper is organized as follows. Section II introduces the HyFlex framework. In Section III we describe the methods considered in our comparison. Section IV introduces each of the new problem domains. In Section V we compare the performance of several hyper-heuristics on all 9 domains. Finally, in Section VI we conclude.

II. HyFlex

In this section we briefly describe the HyFlex framework and how it became and still is an important benchmark for selection hyper-heuristics.

HyFlex is a modular and flexible Java class library for developing and testing iterative general-purpose heuristic search algorithms. HyFlex currently provides 6 different problem domains: Maximum Satisfiability, Bin Packing, Permutation Flow Shop, Personnel Scheduling, Traveling Salesman (TSP) and Vehicle Routing Problem (VRP). Each of which consists of:

- A set of 10-12 benchmark instances, to be solved.
- An evaluation function, measuring the cost of a candidate solution, to be minimized.
- A set of low-level heuristics: One construction heuristic and multiple perturbation and recombination heuristics, sub-divided in 4 categories:
  1) **mutation:** Perform a small modification on the solution, by changing some solution components.
  2) **ruin-recreate:** Partly destroy a solution to rebuild or recreate it afterwards.
  3) **local-search:** Similar to mutation, but may include an iterative improvement process and guarantees that the output solution is at least as good as the input.
  4) **crossover:** Combine two solutions, into a new solution.

  Each of these may furthermore be parametrized by the *intensity of mutation* (α) and *depth of search* (β) parameters.

One of HyFlex's core design principles is that all access to these domain-specific components must occur through a problem independent interface. Thanks to this explicit separation, any method using HyFlex can be readily applied to any instance of another problem domain implemented in HyFlex, without alterations.

HyFlex has been used to support the first Cross-domain Heuristic Search Challenge (CHeSC 2011), during which all 20 contestants were tested (31, 10 min. runs) on 30 instances, 5 from each of the 6 problem domains implemented in HyFlex. To test generalization to new instances, 2 of the 5 instances used in the competition were not available before submission, i.e. were hidden. To test generalization to new domains, the TSP and VRP domains were hidden as well. The winner [11] was the algorithm obtaining the highest accumulated score across all these instances.

After the competition, HyFlex became a benchmark for selection hyper-heuristics and has been used in the implementation of many ever since. Figure 1 shows the number

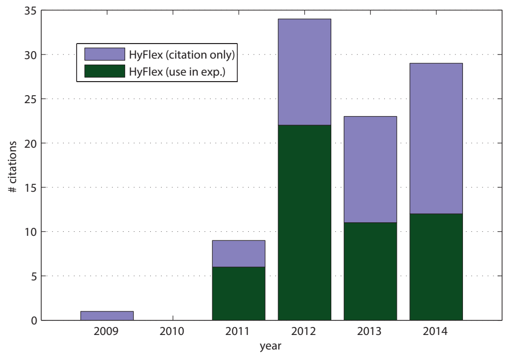

Fig. 1. An Overview of the use of HyFlex in published research

of citations[3] and use[4] of HyFlex over the last 5 years. In comparing these numbers to those of other frameworks for testing cross-domain hyper-heuristics [12], Hyperion [13] and hMod [14], which are cited 17 and 5 times respectively, we find HyFlex (cited 95 times, since 2011) to be the most widely adopted to date. Note that this doesn't mean HyFlex is the best or without flaws. On the contrary, we'd like to argue that due to some unfortunate design choices, HyFlex shows only a glimpse of what is possible using hyper-heuristics. In particular, it provides too little information about domains, instances and solutions that the high-level search method can exploit. Furthermore, providing a good implementation for a problem domain requires a lot of effort and expert knowledge, which might be available for classical benchmark problems, but less so for real-world problems.

III. Hyper-heuristic Methods

In this section we describe the methods considered in our comparison in more detail. Table I gives an overview of these methods and their properties. Note that we include the Lines Of Code metric (loc), as an indication of (code-)complexity.

TABLE I. Overview of the Methods Compared

method	author	style	loc	license
Adap-HH	M. Misir	Select-Accept	4477	GNU GPL
EPH	D. Meignan	Population-based	1158	Apache 2.0
FS-ILS	S. Adriaensen	Iterated Local Search	216	MIT
NR-FS-ILS	S. Adriaensen	Iterated Local Search	166	MIT
ANW-HH	S. Adriaensen	Select-Accept	27	MIT
AA-HH	S. Adriaensen	Select(-Accept)	21	MIT

A. Adap-HH

Adap-HH [11], made publicly available under the name GIHH,[5] is the method that won the CHeSC 2011 competition. At the highest level it follows a rather classical single point iterative selection and acceptance scheme, at a lower level it successfully combines various adaptive mechanisms. Adap-HH maintains an *Adaptive Heuristic Set*. Here, performance metrics are maintained for each of the low level heuristics and after a number of iterations (phase) the performance of each heuristic is re-assessed and a quality

[3]# Articles referencing [6], as listed by Google Scholar™.

[4]# Articles that use HyFlex in their experiments.

[5]http://allserv.kahosl.be/∼mustafa.misir/gihh.html

index is assigned. Within each phase, heuristics are applied with a frequency dependent on the assigned quality index. Also, each phase, heuristics of poor quality are temporarily (or even indefinitely) excluded. An acceptance condition, called *Adaptive Iteration Limited List-based Threshold Accepting*, is used to decide whether to accept the solutions, generated by these heuristics, as new incumbent solution or not. Here, a list of evaluation values of previously found new best solutions is maintained and only proposals no worse than the k^{th} element of this list are accepted. The value of k is adapted dynamically during the search. Next to these *basic* features Adap-HH also implements a restart condition, heuristic adaptation mechanism for α and β and a hybridization scheme (considering pairs of heuristics).

B. EPH

An Evolutionary Programming Hyper-heuristic [15] with co-evolution.[6] This population-based approach ended up 5^{th} in the CHeSC competition. The method maintains two populations, one of candidate solutions, another of heuristic sequences. Both populations co-evolve in the sense that the new solutions introduced in the population are generated by applying the heuristic sequences, and the fitness of the heuristic sequences is related to how they perform on the current solutions. The solution population evolves as follows: First an initial population is generated using the construction heuristic. Then the heuristic sequences are applied to these solutions (in order to evaluate their performance). The resulting solution replaces the worst solution in the population if it has a cost different from all others in the population and lower than the worst. A heuristic sequence consists of a set of perturbation heuristics (from categories 1, 2 and 4), followed by a set of local-search heuristics. Local-search heuristics are either applied once, or using a Variable Neighbourhood Descent (VND) strategy. Each perturbation and local-search heuristic has an associated α and β value. The population of heuristic sequences is initialized randomly and each generation the population is first doubled (by recombination and mutation) and then N individuals are selected based on their fitness using tournament selection. Parameters of the method (e.g. population sizes, use of VND) are determined during a dedicated initialization phase.

C. (NR-)FS-ILS

Fair-Share Iterated Local Search[7] is a hyper-heuristic following an Iterated Local Search scheme. First, it generates an initial candidate solution using the construction heuristic. Then it iteratively selects a perturbation heuristic (from categories 1 and 2) and applies it to the incumbent candidate solution, followed by iterative improvement, applying the heuristics from the Local-search category in a tabu-portfolio, to generate a proposal candidate solution. Then an adaptation of the metropolis acceptance condition is used to decide whether to accept it as new incumbent solution or not. Finally either a new iteration is performed or the search is restarted. A key algorithmic concept is the acceptance rate proportional, selection of the perturbation heuristics. Unlike Adap-HH and EPH, FS-ILS did not contest in the CHeSC 2011 competition, rather it was obtained in later research using a semi-automated design approach [8]. During its design the 30 CHeSC competition instances were used. While [10] did test FS-ILS's generality on new instances and showed it to be competitive to Adap-HH on the HyFlex benchmark set, no evaluation on instances of new domains was performed. Also, accidental complexity analysis suggested that the restart condition might not add sufficient value to motivate its complexity, therefore we also consider the variant without restart (NR-FS-ILS) in our experiments.

D. AA-HH and ANW-HH (X_{naive})

AA-HH and ANW-HH are 2 simple, single point hyper-heuristics. Each of them iteratively generates a proposal by applying a heuristic (from categories 1, 2 and 3) selected uniformly at random. They differ in that AA-HH Accepts All proposals, while ANW-HH Accepts No Worsening proposals as new incumbent solution. We include these (rather naive) methods in our comparison because they are simple. Therefore, it is easier to interpret their results and the state-of-the-art methods described above (hereinafter collectively referred to as X_{soa}), to motivate their complexity, should clearly generalize better. For the sake of reproduction, these methods are also made publicly available.[8]

E. ASAP Default Hyper-heuristics

To avoid biasing our domains to a particular method, we didn't use any of the aforementioned methods during the design and testing phase. Instead we used the 8 ASAP Default Hyper-heuristics, which were developed during the preparation and testing of the CHeSC 2011 competition software. These methods were inspired by state-of-the-art approaches and the design principles of some of these hyper-heuristics, can be found in [16]. Results for these hyper-heuristics were made available for the 4 public domains and were used in a rehearsal competition which was conducted weekly, prior to the competition.

IV. Additional Problem Domains

In this section we extend the HyFlex benchmark set, adding 3 new domains. In what follows we briefly introduce each of them. Table II gives an overview of # heuristics provided by each domain, per category. A description of the heuristics provided by these new domains can be found in the Appendix. More information about the original domains can be found on the CHeSC 2011 website.[9]

TABLE II. # HEURISTICS IN EACH DOMAIN, PER CATEGORY

problem domain	abbrev.	init	ls	mut	rr	xo	total
Max-SAT	SAT	1	2	4	1	2	10
Bin Packing	BP	1	2	3	2	1	9
Permutation Flow Shop	PFS	1	4	5	2	3	15
Personnel Scheduling	PSP	1	4	1	3	3	12
Traveling Salesman	TSP	1	6	5	1	3	16
Vehicle Routing	VRP	1	4	4	2	2	13
0-1 Knap Sack	KP	1	6	5	2	3	17
Quadratic Assignment	QAP	1	2	2	3	2	10
Max-Cut	MAC	1	3	2	3	2	11

[6]http://www.lalea.fr/public/index.php?cmd=smarty\&id=11_len
[7]https://github.com/Steven-Adriaensen/FS-ILS
[8]https://github.com/Steven-Adriaensen/hyflext (naive)
[9]http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html

A. 0-1 Knapsack Problem (KP)

Given a set of n items I, with associated weight $w : I \to \mathbb{R}$ and profit $p : I \to \mathbb{R}$ functions, and knapsack capacity V. Find the subset K that maximizes the total profit $\sum_{i \in K} p(i)$ and that satisfies the capacity constraint, i.e. $\sum_{i \in K} w(i) < V$. Here the search-space S is the subset of 2^I where each candidate solution satisfies the capacity constraint. The cost function $c : S \to \mathbb{R}$ maps each subset K to its negated total profit. This domain provides 10 benchmark instances, generated using the generator[10] used in [17]. Table III gives the optimal solution qualities (f_{opt}) as well as the parameters[11] used to generate each instance.

TABLE III. INSTANCES PROVIDED IN THE KP DOMAIN

index	n	type (t)	seed (i)	f_{opt}*
0	1K	no small weights (15)	12	104046
1	2K	uncorrelated (1)	13	1263861
2	2K	weakly corr. (2)	14	243145
3	2K	almost str. corr. (5)	17	431363
4	2K	no small weights (15)	23	396167
5	5K	uncorrelated (1)	24	4417737
6	5K	weakly corr. (2)	25	954172
7	5K	uncorr., similar weights (9)	32	1577175
8	5K	almost str. corr. (5)	28	1530536
9	5K	no small weights (15)	34	1467454

B. Quadratic Assignment Problem (QAP)

Given a set of n facilities F, a set of n locations L, $d : L \times L \to \mathbb{R}$ a function specifying the distance between each pair of locations and $f : F \times F \to \mathbb{R}$ a function specifying the flow between each pair of facilities. Find an assignment of facilities to distinct locations that minimizes the sum of the distances multiplied by the corresponding flows. The search space S consists of all bijections $F \to L$. The cost function is $c(s) = \sum_{x,y \in F} f(x,y) d(s(x), s(y))$. This domain provides 10 benchmark instances, taken from the QAPLIB library [18]. The properties and best known solution qualities (f_{prev}) of these instances, are summarized in Table IV.

TABLE IV. INSTANCES PROVIDED IN THE QAP DOMAIN

index	name	n	f_{prev}
0	sko100a	100	152002
1	sko100b	100	153890
2	sko100c	100	147862
3	sko100d	100	149576
4	tai100a	100	21052466
5	tai100b	100	1185996137
6	tai150b	150	441786736
7	tai256c	256	43849646
8	tho150	150	7620628
9	wil100	100	273038

C. Max-Cut Problem (MAC)

Given a weighted graph G, with vertices V, edges $E \subset V \times V$ and weight function $w : E \to R$. Find a cut, i.e. a partition of V into two disjoint subsets, such that the sum of the weights of the edges crossing both partitions is maximized. The search-space consists of all possible cuts $V \to \{1,2\}$ of G. Let p be a cut of G and $E_\times = \{(v_i, v_j) \in E | p(v_i) \neq p(v_j)\}$ the set of crossing edges, the cost of p is then given by $-\sum_{e \in E_\times} w(e)$. This domain provides 10 benchmark instances: Instances 2-7 were generated using Rudy, a graph generator by Giovanni Rinaldi[12]. Instances 0-1, 8-9 are torus graphs taken from the 7th DIMACS Implementation Challenge.[13] The properties and best known solution qualities (f_{prev}) of these instances are summarized in Table V.

TABLE V. INSTANCES PROVIDED IN THE MAC DOMAIN

index	name	type	weights	$\|V\|$	$\|E\|$	f_{prev}
0	g3-8	torus	$\mathbb{Z}$	512	1536	41684814
1	g3-15	torus	$\mathbb{Z}$	3375	10125	283206561
2	g14	planar	1	800	4694	3064
3	g15	planar	1	800	4661	3050
4	g16	planar	1	800	4672	3052
5	g22	random	1	2000	19990	13359
6	g34	torus	1, -1	2000	4000	1384
7	g55	random	1	5000	12498	10299
8	pm3-8-50	torus	1, -1	512	1536	458
9	pm3-15-50	torus	1, -1	3375	10125	3014

V. COMPARATIVE STUDY

A. Experimental Setup

As during the CHeSC competition, 31 runs were performed on each instance, for each method, using the default parameter settings. In total 6 different methods were tested on 98 (68 old, 30 new) instances of 9 domains. All experiments were performed on the same machine, equipped with Intel Xeon E5320 (1.86GHz) processors, 8 GB RAM and running Scientific Linux 5.5 (64-bit). Each run, a method was given a time limit, corresponding to 10 minutes on the machine used during the CHeSC competition.[14] As such it is, to some extent, possible to compare results obtained on other machines, to those reported in this paper. To avoid bias, none of the methods compared were ever tested on the new instances prior to our experiments. In addition, to cancel out the influence of potential periodic variations in system performance, the runs of different methods were interleaved.

B. Results

This section is structured as follows: First we compare the methods on the original HyFlex benchmark (see Section V-B1), next we compare them on the 3 new domains (see Sections V-B2,V-B3 and V-B4 resp.). Finally, in Section V-B5 we have a look at their performance over all instances. For each instance of the new domains median results are reported. Due to space constraints median results for the original HyFlex domains are omitted.[15] In addition we report the following performance measures for each domain: Let $C_{x,i}$ the set of results obtained by a method $x \in X$ on an instance $i \in P$, where X and P are the sets of methods and instances considered, respectively.

- **rank**: The rank of each method $x \in X$ on instance $i \in P$, ranked by increasing μ_{rank}.

[10]Available here http://www.diku.dk/~pisinger/codes.html

[11]The range of coefficients is $[1, r = 10\text{K}]$ and # tests is always 1K

[12]The full set can be found at http://web.stanford.edu/~yyye/yyye/Gset/

[13]http://dimacs.rutgers.edu/Challenges/Seventh/Instances/

[14]Using the benchmark program provided on the CHeSC 2011 website

[15]You find them here https://github.com/Steven-Adriaensen/hyflext (data)

- $\boldsymbol{\mu_{rank}}$: The average rank of the median cost $\tilde{c}_{x,i}$ obtained by each method $x \in X$ on an instance i, ranked by increasing cost.
- $\boldsymbol{\mu_{norm}}$: The average normalized evaluation function value [19]. Let $c_i^{min} = MIN_{r \in C_{x,i}, \forall x \in X}(r)$ be the minimum and $c_i^{max} = MAX_{r \in C_{x,i}, \forall x \in X}(r)$ be the maximum cost obtained by any method on some instance i, then $f_{norm}(r) = \frac{r - c_i^{min}}{c_i^{max} - c_i^{min}}$ and $\mu_{norm}(x) = AVG_{r \in C_{x,i}, \forall i \in P}(f_{norm}(r))$.
- $\boldsymbol{\sigma^{run}_{norm}}$: $AVG_{i \in P}(STD_{r \in C_{x,i}}(f_{norm}(r)))$
- $\boldsymbol{\sigma^{\pi}_{norm}}$: $STD_{i \in P}(AVG_{r \in C_{x,i}}(f_{norm}(r)))$
- **best**: The # instances $i \in P$ for which $\tilde{c}_{x,i} = \tilde{c}^{min}_{x,i}$.
- **worst**: The # instances $i \in P$ for which $\tilde{c}_{x,i} = \tilde{c}^{max}_{x,i}$.

Note that σ^{run}_{norm} gives a measure of variation in the solution quality obtained in different runs on the same instance, while σ^{π}_{norm} measures variations in performance on different instances of the same domain.

TABLE VI. COMPARISON OF PERFORMANCE ON HYFLEX

Maximum Satisfiability (SAT)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	FS-ILS	1.33	0.02	0.01	0.02	12	0
2	NR-FS-ILS	2.25	0.03	0.02	0.03	4	0
3	Adap-HH	2.42	0.03	0.02	0.02	4	0
4	EPH	4.0	0.1	0.04	0.04	0	0
5	ANW-HH	5.0	0.39	0.06	0.14	0	0
6	AA-HH	6.0	0.86	0.08	0.03	0	12
Bin Packing (BP)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	EPH	2.33	0.09	0.04	0.07	6	0
2	Adap-HH	2.92	0.17	0.07	0.22	2	1
3	NR-FS-ILS	2.92	0.12	0.04	0.1	1	0
4	ANW-HH	3.17	0.16	0.04	0.17	3	0
5	FS-ILS	3.75	0.13	0.04	0.11	0	0
6	AA-HH	5.92	0.89	0.03	0.18	0	11
Personnel Scheduling (PSP)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	EPH	1.92	0.13	0.09	0.08	7	0
2	NR-FS-ILS	2.21	0.15	0.07	0.11	3	0
3	FS-ILS	3.08	0.15	0.06	0.12	2	0
4	Adap-HH	3.46	0.18	0.1	0.15	0	0
5	ANW-HH	4.58	0.3	0.15	0.2	1	3
6	AA-HH	5.75	0.48	0.12	0.25	0	9
Permutation Flow Shop (PFS)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	NR-FS-ILS	1.54	0.13	0.07	0.05	10	0
2	FS-ILS	2.21	0.16	0.07	0.06	3	0
3	Adap-HH	2.75	0.18	0.07	0.07	3	0
4	EPH	3.5	0.21	0.08	0.06	2	0
5	AA-HH	5.25	0.6	0.06	0.09	0	3
6	ANW-HH	5.75	0.7	0.14	0.07	0	9
Traveling Salesman (TSP)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	NR-FS-ILS	2.4	0.05	0.02	0.02	1	0
2	EPH	2.45	0.06	0.02	0.03	3	0
3	Adap-HH	2.55	0.05	0.02	0.03	4	0
4	FS-ILS	2.6	0.06	0.02	0.03	3	0
5	AA-HH	5.5	0.53	0.16	0.12	0	5
6	ANW-HH	5.5	0.5	0.19	0.17	0	5
Vehicle Routing (VRP)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	FS-ILS	2.2	0.07	0.04	0.06	2	0
2	NR-FS-ILS	2.3	0.09	0.03	0.08	3	0
3	Adap-HH	2.9	0.07	0.02	0.09	2	0
4	EPH	3.3	0.17	0.1	0.13	3	0
5	ANW-HH	4.7	0.31	0.09	0.22	0	2
6	AA-HH	5.6	0.78	0.05	0.31	0	8
HyFlex domains (P_{old})							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	NR-FS-ILS	2.26	0.1	0.04	0.09	22	0
2	FS-ILS	2.54	0.1	0.04	0.09	22	0
3	Adap-HH	2.84	0.12	0.05	0.14	15	1
4	EPH	2.92	0.13	0.06	0.09	21	0
5	ANW-HH	4.76	0.39	0.11	0.24	4	19
6	AA-HH	5.68	0.69	0.08	0.25	0	48

1) Original HyFlex domains: Table VI summarizes the performance of each method on all 68 instances in the original HyFlex benchmark set (P_{old}), grouped per domain. Overall we find that (NR-)FS-ILS outperforms Adap-HH, which in turn outperforms EPH. Note that all of these methods, in the past, have been tested on (at least) the subset of instances used during the CHeSC 2011 competition (P_{CHeSC}). We find the relative performance of these methods on each domain to be largely consistent with the outcome of the CHeSC 2011 competition and that reported in [10]. This with some exceptions: EPH seems to be slightly more competitive, in particular we observe that it outperforms Adap-HH on the BP domain, while on P_{CHeSC}, Adap-HH clearly outperforms EPH. As σ^{π}_{norm} of Adap-HH is large, this is most likely caused by Adap-HH performing very good on some, while poorly on other BP instances and P_{CHeSC} just happening to contain more of the former. Also, we find that NR-FS-ILS performs better than FS-ILS (Overall, and on 4 out the 6 domains), while [10] showed FS-ILS to perform significantly better on P_{CHeSC}. Similar to what was observed during the rehearsal competition, we find that Adap-HH, EPH and (NR-)FS-ILS (X_{soa}) clearly outperform rather naive alternatives such as AA-HH and ANW-HH (X_{naive}).

2) 0-1 Knap Sack Problem (KP): Table VII summarizes the performance and VIII gives the median cost obtained by each method on the 0-1 Knap Sack Problem. Here we observe that Adap-HH and EPH perform well, while (NR-)FS-ILS does not. Key to understanding why (NR-)FS-ILS (and ANW-HH) perform poorly, is the observation that the variability per run (σ^{run}_{norm}) is high for these methods. Also, we find that (NR-)FS-ILS performs worse for larger, more difficult instances. The iterative improvement phase on these instances (starting from an empty solution) can last up to several minutes and the quality of the candidate solution generated varies strongly.

TABLE VII. COMPARISON OF PERFORMANCE ON KP

0-1 Knap Sack Problem (KP)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	Adap-HH	1.6	0.03	0.02	0.03	8	0
2	EPH	1.85	0.05	0.03	0.08	5	0
3	AA-HH	3.5	0.15	0.01	0.26	2	0
4	NR-FS-ILS	4.65	0.36	0.19	0.19	1	6
5	ANW-HH	4.65	0.33	0.15	0.32	0	4
6	FS-ILS	4.75	0.39	0.22	0.21	1	2

TABLE VIII. MEDIAN SOLUTION QUALITIES OBTAINED FOR KP

π	Adap-HH	FS-ILS	NR-FS-ILS	EPH	AA-HH	ANW-HH
0	**-104046**	**-104046**	**-104046**	**-104046**	-104025	-103350
1	**-1258634**	-1220103	-1231767	-1253074	-1209914	-1208666
2	**-242104**	-236813	-239578	-240663	-238397	-232545
3	**-431351**	-431297	-431312	-431333	-431311	-431304
4	**-396167**	-395941	-395654	**-396167**	**-396167**	-396013
5	**-4328770**	-3756992	-3697266	-4283926	-4248962	-4252143
6	**-937868**	-906490	-895516	-936200	-923973	-923425
7	**-1577175**	-1572999	-1572999	**-1577175**	**-1577175**	-1572999
8	-1530463	-1347297	-1346608	**-1530471**	-1530453	-1530453
9	-1467353	-1463681	-1462759	**-1467357**	-1467353	-1466892

3) Quadratic Assignment Problem: Table IX summarizes the performance and X gives the median cost obtained by each method on the Quadratic Assignment Problem. Here, (NR-)FS-ILS performs best and the X_{soa} perform better than the X_{naive} methods. Note that AA-HH clearly outperforms ANW-HH, which performs worst on all QAP instances.

TABLE IX. COMPARISON OF PERFORMANCE ON QAP

Quadratic Assignment Problem (QAP)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	NR-FS-ILS	1.95	0.1	0.05	0.08	5	0
2	Adap-HH	2.5	0.1	0.05	0.07	2	0
3	FS-ILS	2.85	0.1	0.04	0.08	3	0
4	EPH	3.7	0.13	0.06	0.07	0	0
5	AA-HH	4.0	0.15	0.03	0.11	1	0
6	ANW-HH	6.0	0.63	0.15	0.07	0	10

TABLE X. MEDIAN SOLUTION QUALITIES OBTAINED FOR QAP

π	Adap-HH	FS-ILS	NR-FS-ILS	EPH	AA-HH	ANW-HH
0	152214	**152196**	**152196**	152388	152402	154520
1	154164	**154088**	154166	154390	154290	156642
2	**147970**	148002	147978	148122	148190	150930
3	149850	149858	**149828**	150144	149992	152090
4	21366690	**21309210**	21321550	21401250	21518130	21646580
5	1187876000	1187491000	**1187383000**	1189221000	1189321000	1237911000
6	502937700	503088700	502654000	502409100	**502293800**	514793700
7	**44858390**	44874030	44873020	44860940	44866880	44929370
8	8163764	8169250	**8162592**	8163304	8168990	8321922
9	273414	273362	**273336**	273630	273512	275644

4) Max Cut Problem: Table XI summarizes the performance and XII gives the median cost obtained by each method on the Max-Cut problem. Here we observe, somewhat surprisingly, that AA-HH performs best, followed by Adap-HH. Looking closer we find that AA-HH performs best on all instances with unit weight edges (2-9), but ranking only 5^{th} on instance 0. This most likely because the worsening proposed by exploratory operators in the weighted case tends to be much greater (and more variable) than in the unit case. EPH and especially ANW-HH perform clearly the worst. The performance of (NR-)FS-ILS varies strongly from instance to instance (high σ^{π}_{norm}). We find performance to be mediocre to poor on the larger instances (1,5,7,9), while competitive on the smaller.

TABLE XI. COMPARISON OF PERFORMANCE ON MAC

Max Cut Problem (MAC)							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	AA-HH	1.5	0.16	0.05	0.1	8	0
2	Adap-HH	2.25	0.19	0.07	0.07	1	0
3	NR-FS-ILS	3.1	0.31	0.07	0.2	0	0
4	FS-ILS	3.95	0.34	0.08	0.22	1	2
5	EPH	4.6	0.47	0.12	0.14	0	1
6	ANW-HH	5.6	0.7	0.1	0.08	0	7

TABLE XII. MEDIAN SOLUTION QUALITIES OBTAINED FOR MAC

π	Adap-HH	FS-ILS	NR-FS-ILS	EPH	AA-HH	ANW-HH
0	-41175603	**-41348693**	-41145032	-40953212	-40502841	-38772645
1	**-269692927**	-255265025	-257764081	-260608752	-263151470	-258016734
2	-3044	-3041	-3044	-3023	**-3046**	-2996
3	-3025	-3020	-3025	-3004	**-3033**	-2976
4	-3026	-3026	-3028	-3004	**-3035**	-2982
5	-13126	-13083	-13091	-13065	**-13177**	-12964
6	-1314	-1302	-1304	-1206	**-1322**	-1232
7	-9823	-9632	-9668	-9794	**-9878**	-9657
8	-450	-450	-450	-430	**-454**	-404
9	-2786	-2676	-2680	-2648	**-2814**	-2578

5) Summary: Table XIII summarizes the performance of each method on all 98 instances in the extended HyFlex benchmark set (P_{all}), partitioned in P_{old} and P_{new}. At first sight (*rank* and μ_{rank} on P_{all}) one might conclude that NR-FS-ILS generalizes best. When looking at μ_{norm}, however, Adap-HH performs best accross P_{all}. Furthermore, the σ_{norm} and *min* values suggest the performance of Adap-HH to be more reliable. Remark that Adap-HH ranks 2^{nd} or 3^{rd} on 7 out of the 9 domains (KP 1^{st} and PSP 4^{th}) and only performs worst on a single instance. In addition generalization should be measured w.r.t. new instances (not used during design), i.e. P_{new} for cross-domain generalization. Here we find that Adap-HH clearly outperforms all others. Curiously AA-HH ranks 2^{nd}, i.e. we can't generalize the clear superiority of X_{sao} over X_{naive} observed on P_{old} (see Section V-B1). However we'd like to argue that this seemingly dramatic realization should not be worrying. It is no "problem" that a naive method performs better on some particular domains, as long as the overall performance is worse, i.e. it generalizes poorly. Overall we find X_{naive} to be performing clearly worse than X_{sao} on P_{all} and this for all measures of performance. Also, both methods show similar performance, suggesting that the extended benchmark set is well-balanced (i.e. unbiased w.r.t. pure exploration, exploitation). Finally, as FS-ILS performs worse than (NR-)FS-ILS on P_{all} and nearly all domains considered (except for SAT), we can further simplify it by omitting the restart condition.

TABLE XIII. COMPARISON OF PERFORMANCE ACROSS THE EXTENDED HYFLEX BENCHMARK

HyFlex domains (P_{old})							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	NR-FS-ILS	2.26	0.1	0.04	0.09	22	0
2	FS-ILS	2.54	0.1	0.04	0.09	22	0
3	Adap-HH	2.84	0.12	0.05	0.14	15	1
4	EPH	2.92	0.13	0.06	0.09	21	0
5	ANW-HH	4.76	0.39	0.11	0.24	4	19
6	AA-HH	5.68	0.69	0.08	0.25	0	48
New domains (P_{new})							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	Adap-HH	2.12	0.11	0.04	0.09	11	0
2	AA-HH	3.0	0.15	0.03	0.18	11	0
3	NR-FS-ILS	3.23	0.26	0.1	0.2	6	6
4	EPH	3.38	0.22	0.07	0.2	5	1
5	FS-ILS	3.85	0.28	0.12	0.22	5	4
6	ANW-HH	5.42	0.55	0.13	0.25	0	21
Extended Hyflex Benchmark (P_{all})							
rank	**method**	μ_{rank}	μ_{norm}	σ^{run}_{norm}	σ^{π}_{norm}	**best**	**worst**
1	NR-FS-ILS	2.56	0.15	0.06	0.15	28	6
2	Adap-HH	2.62	0.11	0.05	0.12	26	1
3	FS-ILS	2.94	0.15	0.06	0.17	27	4
4	EPH	3.06	0.15	0.07	0.14	26	1
5	AA-HH	4.86	0.53	0.06	0.34	11	48
6	ANW-HH	4.96	0.44	0.12	0.26	4	40

VI. CONCLUSION

In this paper we've conducted a comparative study of several publicly available, state-of-the-art hyper-heuristics for HyFlex in order to assess their generality across domains. To this purpose we've extended the HyFlex benchmark set with 3 *new* problem domains, i.e. the 0-1 Knap Sack (KP), Quadratic Assignment (QAP) and Max-Cut Problem (MAC). In what follows we summarize our findings:

- Of the methods compared, Adap-HH clearly generalizes best to the new domains and performs most consistently overall.
- (NR-)FS-ILS performs poorly on the KP domain, most likely due to the long intensification phases it performs in uninteresting areas of the search space.
- On the full and extended benchmarks FS-ILS does not seem to benefit from its restart condition (on the contrary), we can therefore omit it without loss of performance (with as exception the SAT domain).
- While AA-HH is surprisingly competitive on the new domains, overall the state-of-the-art methods compared, clearly generalize better than their naive counterparts.

The main downside of Adap-HH is its high complexity. One might argue that "complexity is not an issue, as methods are used as a black-box". We'd like to argue to the contrary. For software in general, the *black-box* argument can be rephrased as "It does not matter how my code looks like, the user doesn't see it anyway". This is a known fallacy, as used software also needs to be maintained and therefore be maintainable. We believe this to be even more relevant in a research context. Therefore, interesting future research would be to apply accidental complexity analysis to Adap-HH.

Note that 2 out of the 3 authors of this paper, were also involved in the design of FS-ILS. This leads to an obvious risk of bias in our comparison. Furthermore, given our experimental results it wouldn't have been difficult to select domains and instances in such way that ours would have come out on top. In the setup of our experiments we've taken the nescessary precautions to avoid such bias and have made the reproducability of our experiments a main concern. The results reported for FS-ILS itself aren't that good. This is not problematic, on the contrary, it led us to identify a weakness of our method. Furthermore, we believe that it should be feasible to resolve this issue in future research.

When looking for community support, we found many authors to be interested in having their method included in our comparison. Including all, however, was beyond the scope of this work. Therefore, we plan to publish a more extensive comparison of the state-of-the-art in the near future.

VII. Acknowledgments

In this section we'd like to thank Mustafa Misir and David Meignan for their cooperation, allowing us to run additional experiments and include their methods in our comparison. Special thanks go to Fan Xue, who provided us with a partial implementation of the QAP domain, which served as a source of inspiration in our eventual implementation.

References

[1] H. H. Hoos and T. Stützle, *Stochastic local search: Foundations & applications*. Morgan Kaufmann, 2004.

[2] G. A. Kochenberger *et al.*, *Handbook in Metaheuristics*. Springer, 2003.

[3] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 1, no. 1, pp. 67–82, 1997.

[4] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[5] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," *International series in operations research and management science*, pp. 457–474, 2003.

[6] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic *et al.*, "Hyflex: a benchmark framework for cross-domain heuristic search," *Evolutionary Computation in Combinatorial Optimization*, pp. 136–147, 2012.

[7] M. Mısır, K. Verbeeck, P. De Causmaecker, and G. V. Berghe, "The effect of the set of low-level heuristics on the performance of selection hyper-heuristics," in *Parallel Problem Solving from Nature-PPSN XII*. Springer, 2012, pp. 408–417.

[8] S. Adriaensen, T. Brys, and A. Nowé, "Designing reusable metaheuristic methods: A semi-automated approach," in *Evolutionary Computation (CEC), 2014 IEEE Congress on*. IEEE, 2014, pp. 2969–2976.

[9] N. Sabar, M. Ayob, G. Kendall, and R. Qu, "The automatic design of hyper-heuristic framework with gene expression programming for combinatorial optimization problems," 2012.

[10] S. Adriaensen, T. Brys, and A. Nowé, "Fair-share ils: a simple state-of-the-art iterated local search hyperheuristic," in *Proceedings of the 2014 conference on Genetic and evolutionary computation*. ACM, 2014, pp. 1303–1310.

[11] M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe, "An intelligent hyper-heuristic framework for chesc 2011," in *Learning and Intelligent Optimization*. Springer, 2012, pp. 461–466.

[12] P. Ryser-Welch and J. F. Miller, "A review of hyper-heuristic frameworks," in *Proceedings of the 50th Anniversary Convention of the AISB*, 2014.

[13] J. Swan, E. Özcan, and G. Kendall, "Hyperion–a recursive hyper-heuristic framework," in *Learning and intelligent optimization*. Springer, 2011, pp. 616–630.

[14] E. Urra, D. Cabrera-Paniagua, and C. Cubillos, "Towards an object-oriented pattern proposal for heuristic structures of diverse abstraction levels," *XXI Jornadas Chilenas de Computación 2013, PROCEEDINGS*, 2013.

[15] D. Meignan, "An evolutionary programming hyper-heuristic with co-evolution for chesc11," in *OR53 Annual Conference*, 2011.

[16] E. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, J. A. Vazquez-Rodriguez, and M. Gendreau, "Iterated local search vs. hyper-heuristics: Towards general-purpose search algorithms," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[17] S. Martello, D. Pisinger, and P. Toth, "Dynamic programming and strong bounds for the 0-1 knapsack problem," *Management Science*, vol. 45, no. 3, pp. 414–424, 1999.

[18] R. E. Burkard, S. E. Karisch, and F. Rendl, "Qaplib–a quadratic assignment problem library," *Journal of Global optimization*, vol. 10, no. 4, pp. 391–403, 1997.

[19] L. Di Gaspero and T. Urli, "Evaluation of a family of reinforcement learning cross-domain optimization heuristics," in *Learning and Intelligent Optimization*. Springer, 2012, pp. 384–389.

[20] K. Chan and H. Tansri, "A study of genetic crossover operations on the facilities layout problem," *Computers & Industrial Engineering*, vol. 26, no. 3, pp. 537–550, 1994.

[21] Q. Wu and J.-K. Hao, "A memetic approach for the max-cut problem," in *Parallel Problem Solving from Nature-PPSN XII*. Springer, 2012, pp. 297–306.

Appendix

0-1 Knapsack Problem (KP)

Initialization:
Constructs a candidate solution representing the empty set.

Local search Heuristics:

0) PackRandom: Packs a randomly fitting item.
1) PackBest: Packs most valuable fitting item.
2) PackPPP: Packs a randomly fitting item, improving the *Profit Per Pound* $\frac{\sum_{j \in K} p(j)}{\sum_{j \in K} w(j)}$ of the solution.
3) PackLight: Packs the lightest fitting item.
4) SwapFirst: Substitute a random packed piece by a random more expensive fitting piece.
5) SwapBest: Perform the substitution of 2 pieces, improving the total profit most.

Mutational Heuristics:

6) SWAPRANDOM: Randomly substitutes 2 pieces.
7) SWAPPPP: Randomly substitutes 2 pieces, improving the *Profit Per Pound* of the solution.
8) REMOVERANDOM: Unpacks a random piece.
9) REMOVEWORST: Unpacks the cheapest piece.
10) REMOVEHEAVY : Unpacks the heaviest piece.

The SWAP, REMOVE mutations are repeated $\lceil 5\alpha \rceil$ and $\lceil \alpha |K| \rceil$ times, respectively.

Ruin-Recreate Heuristics:
These heuristics first unpack $\lceil \alpha |K| \rceil$ randomly selected pieces, subsequently they pack pieces, till no more fit.

11) REPACKBEST: Each time packs the most valuable fitting piece.
12) REPACKPPP: Each time packs the fitting piece, improving the *Profit Per Pound* of the solution most.

Crossover Heuristics:

13) INTERSECTION: The output solution packs the items packed in both input solutions.
14) RANDOMUNION: Starting from the empty solution, each time packs a random fitting piece, packed in either of the input solutions.
15) BESTUNION: Starting from the empty solution, each time packs the fitting piece, packed in either of the input solutions, improving the total profit most.

Quadratic Assignment Problem (QAP)

Initialization
Assigns facilities to locations, uniformly at random.

Local search Heuristics:

0) SWAPFIRSTII: Iteratively swaps the locations of 2 random facilities, improving the solution.
1) SWAPBESTII: Iteratively swaps the locations of the 2 facilities, improving the solution most.

Both local-search heuristics terminate at a local optimum.

Mutational Heuristics:

2) SWAPRANDOM: Swaps the locations of 2 random facilities, repeated $\lceil 5\alpha \rceil$ times.
3) SWAPBEST: Swaps the locations of the 2 facilities, resulting in the best solution quality. This operation is repeated $\lceil 1000\alpha^3 \rceil$ times or until an improving solution is found. Within a single application the same pair of facilities are never swapped twice.

Ruin-Recreate Heuristics:

4) RREASSIGN: Unassigns $\lceil \frac{\alpha n}{2} \rceil$ facilities and reassigns them uniformly at random.
5) GREEDYREASSIGNF: First unassigns $\lceil \alpha n \rceil$ facilities. Next re-assigns facilities x to a location, in order of decreasing sum of flows $\sum_{y \in F} f(x,y) + f(y,x)$, minimizing the cost of the partial assignment $c(s') = \sum_{x,y \in F'} f(x,y) d(s(x), s(y))$, with $F' \subset F$ the assigned facilities.
6) GREEDYREASSIGNL: As 5, but instead free locations x are re-assigned to facilities in order of decreasing sum of distances $\sum_{y \in L} d(x,y) + d(y,x)$.

Crossover Heuristics:
The domain provides the 2 classic permutation crossover operations, shown to perform best on QAP in [20].

7) PMX: Partially Matched Crossover.
8) OX: Order Crossover.

Max-Cut Problem (MAC)

Initialization:
Vertices are greedily inserted in a random order in the partition that minimizes the cost of the cut on the sub-graph $G_{partial}$ of G, containing only the vertices and edges between vertices, that were already inserted.

Local search Heuristics:

0) SWAPFIRST: Changes the partition of a random vertex that improves the quality of the solution.
1) SWAPBEST: Changes the partition of the vertex improving the quality of the solution most.
2) SWAPNEIGHBOURS: Changes the partition of the 2 neighbouring vertices improving the quality of the solution most. This move is performed only once.

The SWAPFIRST and SWAPBEST heuristics are repeated for $\lceil 100\beta \rceil$ iterations or until no improving move exists.

Mutational Heuristics:

3) SWAPRANDOM: Changes the partition of a randomly selected vertex (repeated $\lceil 10\alpha \rceil$ times).
4) SWAPRANDOMNEIGHBOURS: Changes the partition of the 2 randomly selected neighbouring vertices (repeated $\lceil 5\alpha \rceil$ times).

Ruin-Recreate Heuristics:

5) RANDOMRR: Removes $\lceil 50\alpha \rceil$ random vertices and re-inserts them in a random partition.
6) GREEDYRR: Removes $\lceil 50\alpha \rceil$ random vertices and re-inserts them greedily, inserting the vertex in the partition, resulting in the best sub-cut.
7) RADIALRR: Removes $\lceil 5\alpha \rceil$ random vertices and all their neighbours, and re-inserts them as in 6.

Crossover Heuristics:

8) ONEPOINTXO: Performs the one point crossover on the partitioning of vertices (ordered by *id*).
9) MULTIPLEPARENTXO: Performs the multiple parent crossover described in [21].