Iterated Local Search vs. Hyper-heuristics: Towards General-Purpose Search Algorithms

Edmund Burke, Tim Curtois, Matthew Hyde, Graham Kendall,
Gabriela Ochoa, Sanja Petrovic , José A. Vázquez-Rodríguez and Michel Gendreau

***Abstract*— An important challenge within hyper-heuristic research is to design search methodologies that work well, not only across different instances of the same problem, but also across different problem domains. This article conducts an empirical study involving three different domains in combinatorial optimisation: bin packing, permutation flow shop and personnel scheduling. Using a common software interface (*HyFlex*), the same algorithms (high-level strategies or hyper-heuristics) can be readily run on all of them. The study is intended as a proof of concept of the proposed interface and domain modules, as a benchmark for testing the generalisation abilities of heuristic search algorithms. Several algorithms and variants from the literature were implemented and tested. From them, the implementation of *iterated local search* produced the best overall performance. Interestingly, this is one of the most conceptually simple competing algorithms, its advantage as a robust algorithm is probably due to two factors: (i) the simple yet powerful exploration/exploitation balance achieved by systematically combining a perturbation followed by local search; and (ii) its parameter-less nature. We believe that the challenge is still open for the design of robust algorithms that can learn and adapt to the available low-level heuristics, and thus select and apply them accordingly.**

I. Introduction

A hyper-heuristic can be seen as a (high-level) methodology which, when presented with a particular problem instance or class of instances, and a number of low-level heuristics (or its components), automatically produces an adequate combination of the provided components to effectively solve the given problem(s). The term hyper-heuristic was first used in 1997 [1] to describe a protocol that combines several Artificial Intelligence methods in the context of automated theorem proving. The term was independently used in 2000 [2] to describe 'heuristics to choose heuristics' in the context of combinatorial optimisation. In this context, the first journal paper to use the term was [3]. The idea of automating the design of heuristics, however, is not new. It can be traced back to the early 1960s [4] and 1970s [5]. Moreover, several related areas are currently being investigated: reactive search optimisation [6], automated parameter control [7], and adaptive memetic algorithms [8], to name a few.

In order to compare the performance of different hyper-heuristics or high-level search strategies across different domains, we propose using a common software interface (which we call *HyFlex*) for dealing with different combinatorial optimisation problems. HyFlex provides a set of routines for generating and evaluating solutions; and applying simple (or low-level) heuristics (such as move or mutation operations, ruin & recreate heuristics, and hill-climbers), on these solutions. Currently, HyFlex provides the implementation of the routines mentioned under a common interface for the following problem domains: one-dimensional bin packing, permutation flow shop and personnel scheduling. The present study uses Hyflex in order to assess the comparative performance of different state of the art hyper-heuristics, and newly proposed variants against a general version of an iterated local search algorithm. The goal of the study is to test the HyFlex concept, and its suitability as a benchmark for assessing the generalisation abilities of heuristic search algorithms. The idea is to determine, using this benchmark, which algorithms have the best generalisation abilities, that is, the best overall performance across the three studied domains.

The article is structured as follows. Section II describes hyper-heuristics in more detail discussing a recently proposed classification of these approaches. Section III presents the software framework used for conducting our empirical study: *HyFlex*. Section IV describes the competing algorithms, namely, an iterated local search implementation and a set of iterative or perturbation based hyper-heuristics. The experimental setup is described in section V, while section VI presents the results. Finally, section VII discusses our main results and suggests directions for future work.

II. Hyper-heuristics

In [9], a unified classification and definition of hyper-heuristics is presented which captures some of the work that is being undertaken in this field. Two main hyper-heuristic categories are identified: heuristic *selection* and heuristic *generation*. This article is concerned with heuristic selection methodologies based on perturbation low-level heuristics. These methods start with a complete solution, generated either randomly or using simple construction heuristics, and thereafter try to iteratively improve the current solution. The hyper-heuristic framework is provided with a set of neighbourhood structures and/or simple local searchers, and the goal is to iteratively select and apply them to the current complete solution. This process continues until a stopping condition has been met. This class of hyper-heuristics has

E. K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic and J. A. Vázquez-Rodríguez, are with the Automated Scheduling, optimisAtion and Planning (ASAP) Research Group, School of Computer Science, University of Nottingham, UK (email: {ekb, tec, mvh, gxk, gxo, sxp, jav}@cs.nott.ac.uk) M. Gendreau is with the Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT) Montreal, Canada (email: michelg@crt.umontreal.ca)

978-1-4244-8126-2/10/$26.00 ©2010 IEEE

the potential to be applied successfully to different combinatorial optimisation problems, since general neighbourhood structures or simple local searchers can be made available for each problem. Hyper-heuristics based on perturbation have been applied to personnel scheduling [3], [2], timetabling [3], shelf space allocation [10], [11], packing [12] and vehicle routing problems [13]. Hyper-heuristics of this type, are generally single-point algorithms in that they maintain a single incumbent solution in the solution space. However, it is worth noticing that these approaches are closely related to adaptive memetic algorithms [8] and adaptive operator selection in genetic algorithms [14], [15], so clearly population based approaches can also be seen as hyper-heuristics.

The working of perturbation or iterative hyper-heuristics can be separated into two processes: (i) (low-level) heuristic selection, and (ii) move acceptance strategy [9], [16]. The heuristic selection can be done in a *non-adaptive* (simple) way: either randomly or along a cycle, based on a prefixed heuristic ordering [2], [17]. No learning is involved in these approaches. Alternatively, the heuristic selection may incorporate an *adaptive* (or on-line learning) mechanism based on the probabilistic weighting of the low-level heuristics [3], [18], or some type of performance statistics [2], [17]. Both non-adaptive and adaptive heuristic selection schemes, are generally embedded within a single-point local search high-level heuristic. The acceptance strategy is an important component of any local search heuristic. Many acceptance strategies have been explored within hyper-heuristics. Move acceptance strategies can be divided into two categories: *deterministic* and *non-deterministic*. In general, a move is accepted or rejected, based on the quality of the move and the current solution during a single point search. Well known meta-heuristic components are commonly used as non-deterministic acceptance methods, such as those of great deluge [19] and simulated annealing [11], [12].

III. The HyFlex framework

HyFlex is an object oriented framework for the implementation and comparison of different perturbation-based iterative hyper-heuristics. The framework is inspired by the notion of a domain barrier between the low-level heuristics and the hyper-heuristic [20], [2]. HyFlex provides a software interface between the hyper-heuristic and the problem domain layers, thus allowing a clearly defined separation, and communication protocol between the domain specific and the domain independent algorithm components. An important feature of the framework, is that it provides the implementation (currently in Java) of a diverse set of combinatorial problems, including their solution representation, evaluation function and a set of useful and varied low-level heuristics. These domain modules encapsulate the problem specific algorithm components, and thus liberate the hyper-heuristic designer from knowing the details of the underlying applications. The creative and implementation efforts will instead be focused on the higher-level hyper-heuristic.

HyFlex extends the hyper- heuristic framework described in [20], [2] in two ways. First, as proposed in [21], a memory or list of solutions is maintained in the domain layer, instead of a single incumbent solution. This extension enriches the possibilities for the hyper-heuristic designer, allowing, for example, the implementation of population based approaches. Second, a large set of low-level heuristics of different types is provided. Specifically, we consider four types of low-level heuristics:

- *Mutational or perturbation heuristics*: perform a (generally) small change in the solution, by swapping, changing, removing, adding or deleting solution components.
- *Hill-Climbing or local search heuristics*: iteratively make small changes (mutations or perturbations) to the solution, accepting improving or non-deteriorating solutions, until a local optimum is found or a stopping condition is met. These heuristics differ from mutational heuristics in that they incorporate an iterative improvement process, therefore they guarantee that a non-deteriorating solution will be produced.
- *Ruin & recreate heuristics*: partly destroy the solution and rebuild or recreate it afterwards. These heuristics can be considered as large neighbourhood structures. They are, however, different from the mutational heuristics in that they can incorporate problem specific construction heuristics to rebuild the solutions.
- *Crossover heuristics*: combine solution components from two input solutions (parents) to produce a new solution or solutions (offspring).

Currently, three problem domain modules have been implemented: permutation flow shop, one-dimensional bin packing, and personnel scheduling. This selection reflects the expertise of our research team. However, it also provides a set of rather different applications. Further extensions of the HyFlex framework will increase the number of application domains. For all the applications, several heuristics from the types discussed above are implemented. Moreover, a large number of well-known and/or real-world benchmark instances for each domain is provided. An overview of the main design choices for each domain, is presented below. More details of each module can be found as technical reports [22], [23], [24].

A. Permutation flow shop

The permutation flow shop problem requires finding the order in which n jobs are to be processed on m consecutive machines. The jobs are processed in the order machine 1, machine 2, ..., machine m. Machines can only process one job at a time and jobs can be processed by only one machine at a time. No job can jump over any other job, meaning that the order in which jobs are processed on machine 1 is maintained throughout the system. Moreover, no machine is allowed to remain idle when a job is ready for processing. All jobs and machines are available at time 0. Each job i requires a processing time on machine j denoted by p_{ij}.

Initialisation: Solutions are initialised using the well established NEH procedure [25]. This heuristic has been used as an important component of many effective meta-heuristics for the permutation flow shop problem.

TABLE I
PERMUTATION FLOW SHOP MODULE INSTANCES.

Instance number	Size
0-9	20×5
10-19	20×10
20-29	20×20
30-39	50×5
40-49	50×10
50-59	50×20
60-69	100×5
70-79	100×10
80-89	100×20
90-99	200×10
100-109	200×20
110-119	500×20

TABLE II
ONE DIMENSIONAL BIN PACKING INSTANCE SETS. EACH SET CONTAINS 20 INSTANCES.

Set name	Piece size distribution	Bin capacity	Number of pieces	Ref
bp1	Uniform [20,100]	150	1000	[32]
bp2	Triples [25,50]	100	501	[32]
bp3	Uniform [150,200]	1000	100	[33]

Low-level heuristics: A total of 14 low level heuristics were implemented. Specifically, 5 mutational, 2 ruin-recreate, 4 local search, and 3 crossover heuristics. The mutational and local search heuristics are inspired by those proposed in [26], [27], whilst the crossover heuristics are widely-known recombination operators for the permutation representation [28], [29]. The ruin-recreate heuristics incorporate the successful NEH procedure in the construction process. For more details see [24].

Instance data: A total of 120 instances from the widely known Taillard set [30], are provided. The instance sizes are given in Table I, in the format $n \times m$. The job processing times in all instances are uniformly distributed random integers in the range $[1, 99]$.

B. One dimensional bin packing

The one-dimensional bin packing problem involves a set of integer-size pieces L, which must be packed into bins of a certain capacity C, using the minimum number of bins possible. In other words, the set of integers must be divided into the smallest number of subsets so that the sum of the sizes of the pieces in a subset does not exceed C.

Initialisation: Solutions are initialised by first randomising the order of the pieces, and then applying the widely known 'first-fit' heuristic [31]. This is a constructive heuristic, which packs the pieces one at a time, each into the first bin that they will fit into, opening a new bin when necessary.

Low-level heuristics: 2 mutational, 2 ruin and recreate, repacked with best-fit, and 3 local search heuristics. For more details see [23].

Instance data: The problem instances are summarised in table II. There are 60 instances in total, 20 in each of three classes.

C. Personnel scheduling

The personnel scheduling problem involves deciding at which times and on which days (i.e. which shifts) each employee should work over a specific planning period. However, the personnel scheduling problem is actually a title for a group of very similar problems. There is no general personnel scheduling problem. Instead there is a group of problems with a common structure but which differ in their constraints and objectives. This creates an additional challenge in implementing a problem domain module for personnel scheduling. To overcome this we have designed a data file format for which each instance can select a combination of objectives and constraints from a wide choice. We then implemented a software framework containing all the methods for these constraints and objectives.

Initialisation: Initial solutions are created with a hill climbing heuristic which uses a neighbourhood operator that adds new shifts to the roster.

Low-level heuristics: 3 mutational (including $vertical$, $horizontal$ and new swaps, see [22]), 5 local search, 3 ruin and recreate, and 3 crossover heuristics. These heuristics are taken from previously proposed successful meta-heuristic approaches to nurse rostering problems [34], [35], [36], [37]. For more details see [22].

Instance data: The instances have been collected from a number of sources. Some of the instances are from industrial collaborators. These include: ORTEC an international consultancy and software company who specialise in workforce planning solutions and SINTEF, the largest independent research organisation in Scandinavia. Other instances have been provided by other researchers or taken from various publications. The collection is a very diverse data set drawn from eleven different countries. The majority of the instances are real world scenarios. An overview of the instances can be found in [22], they vary in the length of the planning horizon, the number of employees and the number of shift types. Each instance also varies in the number and priority of objectives present[1].

IV. THE COMPETING ALGORITHMS

A. Iterated local search

Iterated local search is a relatively straightforward algorithm. As often happens with many simple but sometimes very effective ideas, the same principle has been rediscovered multiple times, leading to different names [38], [39]. The term *iterated local search* was proposed in [40]. Our implementation operates by applying a randomly selected mutation or ruin and recreate heuristic to a solution and then applying all the local search heuristics to the resulting mutated solution. The available local search heuristics are applied in sequence (in a default predefined order). So each local search improves the output of the previous one. If the resulting new solution is better than the original solution then it replaces the original solution, otherwise the new solution is

[1]The instances can be downloaded from:
`http:///www.cs.nott.ac.uk/~tec/NRP/`

simply discarded. The mutation, followed by local search, is repeated inside a loop until the preset time limit is reached. The pseudo code is shown below (algorithm 1).

Algorithm 1 *Iterated Local Search*

```
Create a initial solution s
repeat
  Create a copy of the current solution: s' ← s
  Apply randomly selected mutation or ruin and recreate heuristic to s'
  Apply (in predefined default sequence) all local search heuristics to s'
  if f(s) > f(s') then
    s ← s'
  end if
until time limit is reached
```

B. Iterative hyper-heuristics

We implemented a set of iterative (or perturbation based) hyper-heuristics which conform to the general pseudo code shown below (algorithm 2).

Algorithm 2 *Iterative Hyper-heuristic*

```
Create a initial solution s
repeat
  Create a copy of the current solution: s' ← s
  Select a heuristic H and apply it to s'
  if Accept(s') then
    s ← s'
  end if
until time limit is reached
```

Two components of this algorithm need to be specified in order to have a concrete implementation of a hyper-heuristic. Namely, the strategy for *selecting* a heuristic from the available pool, and an *acceptance* criterion for the solution, obtained after applying the selected heuristic in each step. We implemented two methodologies for heuristic selection and three acceptance criteria, as described below. The combination of them provide us with 6 hyper-heuristic variants.

1) Heuristic selection mechanisms: The two heuristic selection mechanisms implemented are the following:

Simple random (RN): randomly selects one heuristic from the available pool following an uniform probability distribution. This mechanism serves as a base-line for comparison, however it has been found to produce good results when combined with an adequate acceptance criterion [2], [17].

Reinforcement learning with Tabu search (TS): the low-level heuristics are selected according to learnt scores (ranks). The mechanism incorporates a dynamic tabu list of low-level heuristics that are temporarily excluded from the available heuristics. The algorithm deterministically selects the low-level heuristic with the highest rank (max strategy) that is not included in the tabu list. If the selected heuristic produces an improvement, the heuristic rank is increased, otherwise the rank is decreased, and the heuristic is placed into the tabu list. This mechanism was proposed and successfully applied in timetabling and personnel scheduling problems [3].

2) Acceptance criteria: The three criteria implemented are the following:

Naïve acceptance (NV): accepts all improvements. Deteriorations are accepted with 50% probability.

Adaptive acceptance (AA): accepts all improvements. Deteriorations are accepted according to a rate ($acceptanceRate$), which changes depending on whether the search appears to be progressing or stuck in a local optimum. The $acceptanceRate$ value begins at zero, thus, initially, it is an accept-only-improving strategy. However, if the solution does not improve for 0.1 seconds, then the $acceptanceRate$ is increased by 5%, making it more likely that a worse solution is accepted. It is increased to 10% if there is no further improvement in the next 0.1 seconds. Conversely, if the search is progressing well, with no decrease in fitness in the last 0.1 seconds, then the $acceptanceRate$ is reduced by 5%, making it less likely for a worse solution to be accepted. These modifications are intended to help the search navigate out of local optima, and to focus the search if it is progressing well. This mechanism was proposed and tested on the HyFlex bin packing domain [23].

Great Deluge (GD): the great deluge algorithm [41], can be seen as a deterministic variant of the better known simulated annealing algorithm. At each iteration, any solution is accepted which is not much worse than an expected objective value referred to as *level* that changes at a linear rate every step. If $f(s_{new}) < level = f(s_0) - (t\Delta F)/T$, then move is accepted at step t, where $f(s_0)$ is the objective value of the initial solution, ΔF is the difference between the objective values of $f(s_0)$ and the expected final objective value and T is the maximum number of iterations.

V. Experimental setup

Each problem domain is treated as a competition between the implemented hyper-heuristics. Ten problem instances were selected for each domain, which represent 'events' in the competition. For each instance, and competing algorithm, 30 runs were conducted, and the average best objective function obtained represents the algorithm's measurement for the event. Two running time limits were set which represent a $short$ and a $long$ track. These times were set in CPU seconds as indicated in table III. The experiments were performed on the University of Nottingham computing cluster, on Intel Xeon (Harpertown) 3GHz quad core CPUs. Each run was performed on one CPU.

A. Quality measures

We conceive this comparison as a preliminary version of a competition or challenge where the winning algorithm will be the one having the best generalisation ability. Using a sports metaphor, we are interested in conducting the

TABLE III
THE RUNNING (TIME-OUT) TIMES USED FOR EACH PROBLEM DOMAIN.

Short track	*Long* track	Problem domains
10 CPU seconds	20 CPU seconds	bin packing and flow shop
20 CPU seconds	40 CPU seconds	personnel scheduling

$Dechathlon$ challenge of search heuristics [42]. Since different domains and instances are considered, normalised quality measures or scores are required. We used here two alternative normalised measures in order to assess their suitability as scoring systems for the competition.

Simple ranking: This consists of simply ranking the algorithms on an event (instance) according to their measurement (average best objective value). The best performing algorithm will rank 1 and the worst will rank n (the number of competing algorithms).

ROADEF score: The ROADEF challenge[2] is an operational research challenge dedicated to industrial applications, organised every two years by the French Operational Research and Decision Support Society (ROADEF). In this challenge the score is computed as follows: let $z(M, I)$ denote the objective function value obtained by Method M on Instance I. Let $z_b(I)$ and $z_w(I)$ denote the best and worst objective function values found on instance I, respectively. The normalised score obtained by Method M on Instance I is given by:

$$Score(M, I) = 100 \times \frac{z_w(I) - z(M, I)}{z_w(I) - z_b(I)} \quad (1)$$

Multiplying by 100 gives a more manageable range of scores from 0.0 to 100.0. These measures are thereafter averaged for each instance and domain.

B. The competing algorithms

As discussed in section IV, seven algorithms participate in our competition: the iterated local search implementation (ILS), and six variants of the iterative hyper-heuristic framework, resulting from combining the available two heuristic selection mechanisms and three acceptance criteria. Using the notation from section IV-B, these algorithms are: *TS+NV*, *TS+AA*, *TS+GD*, *RN+NV*, *RN+AA* and *RN+GD*.

VI. RESULTS

Tables IV, V and VI report the rank and ROADEF score measures on the three studied domains, respectively. The measurements are reported for the seven competing algorithms on the 10 selected instances for each domain. Average measurements are also reported, and bold fonts indicate the best quality values.

As Table IV reflects, there is a best performing algorithm: *ILS*, for the first 5 bin packing instances. Indeed, there is a regular ranking of the competing algorithms in this group of instances (t501-15 to t501-19)[3], which suggests that they are very similar to each other. A different, but also regular pattern can be observed for the second group of instances (I15 to I19) where the best performing algorithm (in all but instance $I15$) is *TS+GD*. The two average quality measurements suggest a different winning algorithm in this domain; while the average ranking favors the *ILS* algorithm, the average ROADEF score favors *TS+GD*. However, the two measurements are generally in good agreement.

The permutation flow shop results (Table V) suggest that the first 2 instances are relatively easy to solve, as most of the competing algorithms produce good results on them. These two instances are amongst the smallest in the Taillard set (see Table I). For the larger instances ILS has an edge. However, on average the *RN+GD* hyper-heuristic produces better scores, although it is closely followed by ILS.

Personnel scheduling is the domain containing the most varied set of instances, as each of them correspond to real-world scenarios from different institutions as discussed in section III-C. Table VI suggests that different algorithms are better suited for the different instances. We can see, however, that *TS+GD* and *RN+GD* are consistently ranking low in all the instances. The best average performance in this domain is achieved by *RN+NV*, closely followed by *RN+AA*.

We have discussed the performance of the competing algorithms separately in the 3 domains. However, we are interested in the overall performance across all the domains. Moreover, the analysis so far is based on the long computational times (see Table III). The results for the shorter running times were rather similar, therefore, we only present a summary of the averages in Table VII. As Table VII indicates, both running times produced a similar ranking of the competing algorithms, especially for the top places. For both quality measurements ILS is the winning algorithm, followed in second place by *RN+AA*. The third place is given to the *TS+AA* algorithm by the rank measurement, whereas this place is occupied by *TS+GD* according to the ROADEF score measurement. The algorithm with worst overall performance was *RN+GD*.

VII. DISCUSSION AND FUTURE WORK

We have conducted a large empirical study involving three different domains in combinatorial optimisation: bin packing, permutation flow shop and personnel scheduling. Using a common software interface (HyFlex) we were able to run the same algorithms (high-level strategies or hyper-heuristics) on all of them. The study was intended as a proof of concept of a competition or challenge where the winners are those algorithms with better generalisation abilities across the different instances and domains: the *Decathlon* challenge of search heuristics.

From the competing algorithms we found that our simple implementation of Iterated Local Search (ILS) produced

[2]The French Operational Research and Decision Support Society (ROADEF)challenge (`http://challenge.roadef.org`)

[3]Instances t501-15 to t501-19 belong to the set named *bp*1, and instances I15 to I19 to *bp*2 in Table II. These files are available at `http://paginas.fe.up.pt/~esicup/`

TABLE IV

BIN PACKING DOMAIN: RANKING AND SCORE MEASUREMENTS FOR THE LONG RUNNING TIMES. THE SCORE (SHOWN BETWEEN BRACKETS) IS CALCULATED ACCORDING TO EQUATION 1. THE BEST MEASUREMENTS ARE SHOWN IN BOLD FONT.

Instance	*TS+NV*	*TS+AA*	*TS+GD*	*RN+NV*	*RN+AA*	*RN+GD*	*ILS*
t501-15	5 (82.54)	2 (99.73)	4 (84.49)	6 (13.49)	3 (96.02)	7 (0.00)	**1 (100.0)**
t501-16	5 (83.16)	2 (99.45)	4 (90.20)	6 (10.84)	3 (97.31)	7 (0.00)	**1 (100.0)**
t501-17	5 (81.81)	2 (99.49)	4 (83.87)	6 (9.64)	3 (95.89)	7 (0.00)	**1 (100.0)**
t501-18	5 (82.99)	2 (99.21)	4 (87.92)	6 (6.70)	3 (96.85)	7 (0.00)	**1 (100.0)**
t501-19	5 (80.92)	2 (99.52)	4 (84.98)	6 (5.39)	3 (97.34)	7 (0.00)	**1 (100.0)**
I15	4 (92.81)	2 (95.43)	3 (92.81)	6 (38.56)	5 (92.81)	7 (0.00)	**1 (100.0)**
I16	5 (33.22)	3 (58.18)	**1 (100.0)**	7 (0.00)	4 (54.71)	6 (15.04)	2 (79.67)
I17	5 (15.73)	2 (53.32)	**1 (100.0)**	7 (0.00)	4 (21.62)	6 (6.63)	3 (48.40)
I18	5 (32.58)	3(42.05)	**1 (100.0)**	6 (0.38)	4 (34.09)	7 (0.00)	2 (75.00)
I19	6 (17.84)	2 (99.70)	**1 (100.0)**	7 (0.00)	3 (86.59)	5 (35.21)	4 (83.69)
Average	5.0 (60.36)	2.2 (84.61)	2.7 **(92.43)**	6.3 (8.50)	3.5 (77.32)	6.6 (5.69)	**1.7** (88.68)

TABLE V

PERMUTATION FLOW SHOP DOMAIN: RANKING AND SCORE MEASUREMENTS FOR THE LONG RUNNING TIMES. THE SCORE (SHOWN BETWEEN BRACKETS) IS CALCULATED ACCORDING TO EQUATION 1. THE BEST MEASUREMENTS ARE SHOWN IN BOLD FONT.

Instance	*TS+NV*	*TS+AA*	*TS+GD*	*RN+NV*	*RN+AA*	*RN+GD*	*ILS*
ta012	**1 (100.0)**	5 (84.62)	**1 (100.00)**	5 (84.62)	**1 (100.0)**	**1 (100.0)**	7 (0.00)
ta022	**1 (100.0)**	**1 (100.0)**	7 (0.00)	**1 (100.0)**	**1 (100.0)**	**1 (100.0)**	6 (57.14)
ta032	7 (0.00)	6 (13.00)	2 (70.00)	4 (46.00)	5 (36.00)	**1 (100.0)**	3 (56.00)
ta042	7 (0.00)	4 (9.43)	3 (58.30)	5 (6.42)	6 (3.96)	2 (67.36)	**1 (100.0)**
ta052	4 (6.76)	5 (4.72)	2 (77.94)	6 (1.07)	7 (0.00)	3 (72.42)	**1 (100.0)**
ta062	7 (0.00)	5 (33.33)	6 (32.05)	2 (96.15)	**1 (100.0)**	2 (96.15)	4 (34.62)
ta072	6 (1.65)	4 (19.78)	2 (97.80)	7 (0.00)	5 (18.68)	3 (95.88)	**1 (100.0)**
ta082	4 (23.06)	6 (19.47)	**1 (100.0)**	7 (0.00)	5 (22.29)	3 (71.48)	2 (87.62)
ta092	5 (9.42)	6 (6.69)	2 (96.51)	7 (0.00)	4 (11.40)	3 (82.37)	**1 (100.0)**
ta096	7 (0.00)	6 (6.62)	3 (74.14)	5 (22.78)	4 (29.76)	2 (90.93)	**1 (100.0)**
Average	4.9 (24.09)	4.8 (29.77)	2.9 (70.67)	4.9 (35.70)	3.9 (42.21)	**2.1 (87.66)**	2.7 (73.54)

TABLE VI

PERSONNEL SCHEDULING DOMAIN: RANKING AND SCORE MEASUREMENTS FOR THE LONG RUNNING TIMES. THE SCORE (SHOWN BETWEEN BRACKETS) IS CALCULATED ACCORDING TO EQUATION 1. THE BEST MEASUREMENTS ARE SHOWN IN BOLD FONT.

Instance	*TS+NV*	*TS+AA*	*TS+GD*	*RN+NV*	*RN+AA*	*RN+GD*	*ILS*
BCV-3.46.2	**1 (100.0)**	4 (55.56)	5 (26.67)	3 (66.67)	2 (75.56)	7 (0.00)	6 (22.22)
BCV-4.13.1	5 (72.73)	2 (90.91)	6 (63.64)	2 (90.91)	**1 (100.0)**	7 (0.00)	4 (81.82)
ORTEC01	5 (66.40)	7 (0.00)	3 (88.51)	2 (93.50)	6 (65.13)	4 (74.86)	**1 (100.0)**
GPost	**1 (100.0)**	5 (78.06)	7 (0.00)	2 (97.50)	4 (87.27)	6 (5.27)	3 (93.73)
QMC-2	4 (82.14)	**1 (100.0)**	7 (0.00)	5 (78.57)	3 (85.71)	6 (10.71)	2 (96.43)
Ikegami-2Shift	6 (24.22)	3 (55.28)	7 (0.00)	**1 (100.0)**	2 (74.53)	4 (51.55)	5 (45.34)
Ikegami-3Shift	4 (77.87)	7 (0.00)	5 (50.27)	**1(100.00)**	2 (97.27)	3 (88.25)	6 (47.54)
Valouxis-1	2 (98.46)	**1 (100.0)**	7 (0.00)	5 (80.00)	3 (92.31)	6 (16.92)	4 (87.69)
Azaiez	**1 (100.0)**	**1 (100.0)**	7 (0.00)	**1 (100.0)**	**1 (100.0)**	6 (45.00)	5 (92.50)
SINTEF	5 (73.04)	4 (74.78)	6 (17.39)	2 (99.13)	**1 (100.0)**	7 (0.00)	3 (93.04)
Average	3.4 (79.49)	3.5 (65.46)	6.0 (24.65)	**2.4 (90.63)**	2.5 (87.78)	5.6 (29.26)	3.9 (76.03)

TABLE VII
THE RANKING OF THE COMPETING ALGORITHMS WITH RESPECT TO THE TWO RUNNING TIMES, AND THE TWO QUALITY MEASUREMENTS.

	Short track		Long track	
Place	Rank	Score	Rank	Score
1^{st}	*ILS* (2.73)	*ILS* (80.08)	*ILS* (2.77)	*ILS* (79.42)
2^{nd}	*RN+AA* (3.20)	*RN+AA* (70.18)	*RN+AA* (3.30)	*RN+AA* (69.10)
3^{rd}	*TS+AA* (3.83)	*TS+GD* (61.97)	*TS+AA* (3.50)	*TS+GD* (62.58)
4^{th}	*TS+GD* (4.07)	*TS+AA* (53.23)	*TS+GD* (3.87)	*TS+AA* (59.94)
5^{th}	*RN+NV* (4.50)	*TS+NV* (49.41)	*TS+NV* (4.43)	*TS+NV* (54.64)
6^{th}	*RN+GD* (4.60)	*RN+NV* (44.01)	*RN+NV* (4.53)	*RN+NV* (44.94)
7^{th}	*TS+NV* (4.73)	*RN+GD* (43.88)	*RN+GD* (4.77)	*RN+GD* (40.87)

the best overall performance, therefore, we declare it as the winner of our competition. It is interesting to notice, however, that ILS was not the best performing algorithm in all the instances of the three domains, but ranked consistently well on most of them. This indicates that there is arguably room for improving the design of adaptive hyper-heuristics. The more sophisticated mechanism for selecting the heuristics: reinforcement learning with Tabu search (described in section IV-B), occupied only the third and fourth places in the ranking. Similarly, the most sophisticated acceptance criterion: Great Deluge, did not perform as well as we expected. It is worth mentioning that Great Deluge has a single parameter that needs to be tuned. We selected a parameter value that produced good overall behaviour, but it may be the case that a carefully tuned Great Deluge for each instance and domain will produce better results on them. This is, of course, not in the interest of our challenge. The simple adaptive acceptance criteria implemented (also described in section IV-B) produced better results. We hypothesise that the advantage of *ILS* as a robust algorithm is due to two factors: (i) the simple yet powerful exploration/exploitation balance achieved by systematically combining a perturbation followed by local search; and (ii) its parameter-less nature. The favourable results achieved by *ILS* also suggest that the studied domains, although fundamentally different, may have some commonalities regarding the landscape structure, namely, a clustered distribution of local optima.

We observe that the quality measurement and set of instances selected for the competition, will impact upon its outcome. We selected two simple measurements and the results are generally well correlated. Selecting a diverse set of instances is important for the main goal of our challenge, that of highlighting the generalisation abilities of the competing algorithms.

There are many directions for further work. Firstly, the incorporation of additional problem domains and instances will increase the difficulty and quality of the challenge. Secondly, we believe the challenge is open for the design and implementation of robust algorithms that can learn and adapt to the available low-level heuristics, and thus select and apply them accordingly. An important group of algorithms that should be implemented and explored within the HyFlex framework are those based on a population of solutions, incorporating the available crossover heuristics. Our goal is to promote research into generally applicable search methodologies. To this end, we aim to extend the HyFlex framework with additional problem domains, and make it available to the research community.

REFERENCES

[1] J. Denzinger, M. Fuchs, and M. Fuchs, "High performance ATP systems by combining several ai methods," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, 1997, pp. 102–107.

[2] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach for scheduling a sales summit," in *Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000*, ser. Lecture Notes in Computer Science. Konstanz, Germany: Springer, August 2000, pp. 176–190.

[3] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[4] H. Fisher and G. L. Thompson, "Probabilistic learning combinations of local job-shop scheduling rules," in *Factory Scheduling Conference*, Carnegie Institue of Technology, May 10-12 1961.

[5] I. Rechenberg, *Evolution strategy: Optimization of technical systems by means of biological evolution*. Stuttgart: Fromman-Holzboog, 1973.

[6] R. Battiti, M. Brunato, and F. Mascia, *Reactive Search and Intelligent Optimization*, ser. Operations Research Computer Science Interfaces Series. Springer, 2008, vol. 45.

[7] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith, *Parameter Setting in Evolutionary Algorithms*. Springer, 2007, ch. Parameter Control in Evolutionary Algorithms, pp. 19–46.

[8] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, "Classification of adaptive memetic algorithms: a comparative study," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 36, no. 1, pp. 141–152, 2006.

[9] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J. Woodward, *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science. Springer, 2009, ch. A Classification of Hyper-heuristic Approaches.

[10] R. Bai, E. K. Burke, and G. Kendall, "Heuristic,meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation," *Journal of the Operational Research Society*, vol. 59, pp. 1387 – 1397, 2008.

[11] R. Bai and G. Kendall, "An investigation of automated planograms using a simulated annealing based hyper-heuristics," in *Metaheuristics: Progress as Real Problem Solver - (Operations Research/Computer Science Interface Serices, Vol.32)*, T. Ibaraki, K. Nonobe, and M. Yagiura, Eds. Springer, 2005, pp. 87–108.

[12] K. A. Dowsland, E. Soubeiga, and E. K. Burke, "A simulated annealing hyper-heuristic for determining shipper sizes," *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.

[13] D. Pisinger and S. Ropke, "A general heuristic for vehicle routing problems," *Computers and Operations Research*, vol. 34, pp. 2403–2435, 2007.

[14] A. Fialho, L. Da Costa, M. Schoenauer, and M. Sebag, "Extreme: dynamic multi-armed bandits for adaptive operator selection," in *GECCO '09: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*. New York, NY, USA: ACM, 2009, pp. 2213–2216.

[15] J. Maturana, A. Fialho, F. Saubion, M. Schoenauer, and M. Sebag, "Extreme compass and dynamic multi-armed bandits for adaptive operator selection," in *Proc. IEEE Congress on Evolutionary Computation CEC '09*, 2009, pp. 365–372.

[16] E. Ozcan, B. Bilgin, and E. E. Korkmaz, "Hill climbers and mutational heuristics in hyperheuristics," in *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, ser. Lecture Notes in Computer Science, vol. 4193, Reykjavik, Iceland, September 2006, pp. 202–211.

[17] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation," in *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, ser. Lecture Notes in Computer Science, S. Cagoni, J. Gottlieb, E. Hart, M. Middendorf, and R. Goenther, Eds., vol. 2279. Kinsale, Ireland: Springer-Verlag, April 3-4 2002, pp. 1–10.

[18] A. Nareyek, "Choosing search heuristics by non-stationary reinforcement learning," in *Metaheuristics: Computer Decision-Making*, M. G. C. Resende and J. P. de Sousa, Eds. Kluwer, 2003, ch. 9, pp. 523–544.

[19] G. Kendall and M. Mohamad, "Channel assignment in cellular communication using a great deluge hyper-heuristic," in *Proceedings of the 2004 IEEE International Conference on Network (ICON2004)*, 2004, pp. 769–773.

[20] E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Kluwer, 2003, pp. 457–474.

[21] J. Woodward, A. Parkes, and G. Ochoa, "A mathematical framework for hyper-heuristics," 2008 September, workshop on Hyper-heuristics - Automating the Heuristic Design Process, in conjunction with the 10th International Conference on Parallel Problem Solving From Nature (PPSN X), Dortmund, Germany.

[22] T. Curtois, "A hyflex module for the personnel scheduling problem," School of Computer Science, University of Nottingham, Tech. Rep., 2009.

[23] M. Hyde, "A hyflex module for the one dimensional bin-packing problem," School of Computer Science, University of Nottingham, Tech. Rep., 2009.

[24] J. A. Vázquez-Rodríguez, "A hyflex module for the permutation flow shop problem," School of Computer Science, University of Nottingham, Tech. Rep., 2009.

[25] M. Nawaz, E. E. Jr., and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *OMEGA-International Journal of Management Science*, vol. 11, no. 1, pp. 91–95, 1983.

[26] R. Ruiz and T. G. Stützle., "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, pp. 2033–2049, 2007.

[27] ——, "An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," *European Journal of Operational Research*, vol. 187, no. 10, pp. 1143–1159, 2007.

[28] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985, pp. 136–140.

[29] D. E. Goldberg and J. R. Lingle., "Alleles, loci, and the traveling salesman problem." in *Proceedings of the 1st International Conference on Genetic Algorithms*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1985.

[30] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[31] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham, "Worst-case performance bounds for simple one-dimensional packaging algorithms," *SIAM Journal on Computing*, vol. 3, no. 4, pp. 299–325, December 1974.

[32] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of Heuristics*, vol. 2, pp. 5–30, 1996. [Online]. Available: citeseer.ist.psu.edu/falkenauer96hybrid.html

[33] P. Schwerin and G. Wäscher, "The bin-packing problem: A problem generator and some numerical experiments with ffd packing and mtp," *International Transactions in Operational Research*, vol. 4, no. 5, pp. 377–389, 1997.

[34] E. K. Burke, T. Curtois, R. Qu, and G. V. Berghe, "A scatter search for the nurse rostering problem," School of Computer Science, University of Nottingham, Tech. Rep., 2007.

[35] ——, "A time predefined variable depth search for nurse rostering," School of Computer Science, University of Nottingham, Tech. Rep., 2007.

[36] E. K. Burke, P. Cowling, P. D. Causmaecker, , and G. V. Berghe, "A memetic approach to the nurse rostering problem," *Applied Intelligence*, vol. 15, no. 3, pp. 199–214, 2001.

[37] E. K. Burke, T. Curtois, G. Post, R. Qu, and B. Veltman, "A hybrid heuristic ordering and variable neighbourhood search for the nurse rostering problem," *European Journal of Operational Research*, vol. 188, no. 2, pp. 330–341, 2008.

[38] J. Baxter, "Local optima avoidance in depot location," *Journal of the Operational Research Society*, vol. 32, pp. 815–819, 1981.

[39] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the TSP incorporating local search heuristics," *Operations Research Letters*, vol. 11, no. 4, pp. 219–224, 1992.

[40] H. R. Lourenco, O. Martin, and T. Stutzle, *Iterated Local Search*. Norwell, MA: Kluwer Academic Publishers,, 2002, pp. 321–353.

[41] G. Dueck, "New optimization hueristics: The great deluge algorithm and the record-to record travel," *Journal of Computational Physics*, vol. 104, pp. 86–92, 1993.

[42] E. K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, and J. A. Vázquez-Rodríguez, "Towards the decathlon challenge of search heuristics," in *Workshop on Automated Heuristic Design - In conjunction with the Genetic and Evolutionary Computation Conference (GECCO-2009)*, 2009, pp. 2205–2208.