Accepted refereed manuscript of:

Soria-Alcaraz JA, Ochoa G, Sotelo-Figeroa M & Burke EK (2017) A methodology for determining an effective subset of heuristics in selection hyper-heuristics, *European Journal of Operational Research*, 260 (3), pp. 972-983.

DOI: 10.1016/j.ejor.2017.01.042

© 2017, Elsevier. Licensed under the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International
http://creativecommons.org/licenses/by-nc-nd/4.0/

A methodology for determining an effective subset of heuristics in selection hyper-heuristics

Jorge A. Soria-Alcaraz[a,*], Gabriela Ochoa[b], Marco A. Sotelo-Figeroa[a], Edmund K. Burke[1,c]

[a] *Universidad de Guanajuato, Guanajuato, Mexico*
[b] *University of Stirling, Stirling, Scotland, UK*
[c] *Queen Mary University of London, London, UK*

Abstract

We address the important step of determining an effective subset of heuristics in selection hyper-heuristics. Little attention has been devoted to this in the literature, and the decision is left at the discretion of the investigator. The performance of a hyper-heuristic depends on the quality and size of the heuristic pool. Using more than one heuristic is generally advantageous, however, an unnecessary large pool can decrease the performance of adaptive approaches. Our goal is to bring methodological rigour to this step. The proposed methodology uses non-parametric statistics and fitness landscape measurements from an available set of heuristics and benchmark instances, in order to produce a compact subset of effective heuristics for the underlying problem. We also propose a new iterated local search hyper-heuristic using *multi-armed bandits* coupled with a change detection mechanism. The methodology is tested on two real-world optimisation problems: course timetabling and vehicle routing. The proposed hyper-heuristic with a compact heuristic pool, outperforms state-of-the-art hyper-heuristics and competes with problem-specific methods in course timetabling, even producing new best-known solutions in 5 out of the 24 studied instances.

Keywords: Metaheuristics, Hyper-heuristics, Adaptive Search, Combinatorial optimisation, Iterated Local Search

1. Introduction

Selection hyper-heuristics are search methodologies that operate at a high-level to coordinate a pool of low-level heuristics to solve optimisation problems [6, 7]. The objective is to apply the most effective low-level heuristic at each subsequent stage in order to solve a given problem instance. The result is an automated methodology producing a sequence of heuristics to improve or construct a solution. Hyper-heuristics raise the level of generality at witch optimisation systems can operate [6]. Low-level heuristics usually represent simple local search neighbourhoods (move operators) or abstractions of the rules used by human experts for constructing solutions (constructive heuristics).

The performance of a hyper-heuristic clearly depends on the quality of the pool of low-level heuristics selected [33] . Usually, the human expert, based on previous knowledge, chooses the composition and size of this pool with little if any methodological guidance. Hyper and meta heuristics research over the years, clearly indicates that having more than one operator or heuristic benefits the search process. However, an unnecessary large pool has been found to have a detrimental effect of adaptive approaches, given the overhead in time and memory required to record their performance history [43, 9]. There is a need for informed approaches to identify the most appropriate set of low-level heuristics in hyper-heuristic research and practice [33].

The main contribution of this work is precisely an empirical methodology for determining the most effective subset of low-level heuristics from a given larger set. Determining a subset requires both determining its size and composition. We also propose an efficient adaptive hyper-heuristic variant, which extends and improves our previously proposed learning hyper-heuristic [43]. The contributions of the paper are the following:

*corresponding author
Email addresses: jorge.soria@ugto.mx (Jorge A. Soria-Alcaraz), goc@cs.stir.ac.uk (Gabriela Ochoa), masotelof@ugto.mx (Marco A. Sotelo-Figeroa), e.burke@qmul.ac.uk (Edmund K. Burke)

1. An empirical methodology for determining the most effective subset of low-level heuristics, using fitness landscape probing techniques and non-parametric statistical tests.
2. A new variant of iterated local search hyper-heuristics using *multi-armed bandits* coupled with a change detection mechanisms for adaptively selecting low-level heuristics.
3. Empirical evidence supporting the effectiveness of the proposed methodology on two real-world scheduling problem, namely, educational course timetabling and vehicle routing.

The next Section overviews relevant background and related work in selection hyper-heuristics. Section 3 discusses the proposed methodology for selecting the most effective subset of heuristics. Section 4 describes the hyper-heuristic variant proposed incorporating dynamic multi-armed bandits as the selection rule. Section 5 describes the test problems considered: course timetabling and vehicle routing, while sections 6 and 7 showcases the application of the proposed methodos application on them. Finally, section 8 presents our conclusions and future work.

2. Background and related work

2.1. Selection hyper-heuristics

Hyper-heuristics have been defined as "automated methodologies for selecting or generating heuristics to solve computational search problems" [7]. The present study concentrates on methodologies for automatically selecting low-level heuristics, from an available pool, on-the-fly while solving the underlying optimisation problem. The selection mechanism considers only the history of domain-independent information from the search process. However, low-level heuristics which encapsulate domain-specific information can be, and usually are, incorporated in the pool of heuristics. Our approach considers improvement heuristics, also known as move operators. It starts from a complete initial solution and iteratively selects, from the available pool, appropriate heuristics to lead the search toward a promising direction. Several different strategies for dynamically selecting heuristics while solving the problem have been proposed [6]. These range from simple random and greedy strategies to sophisticated adaptive or reinforcement learning mechanisms incorporating various feedback from the search process.

Surprisingly, little work has been devoted to determine the "optimal" size and composition of the heuristic pool. This is normally done at the discretion of the researcher, without a clear methodology. Most hyper-heuristic implementations use in the order of 3 to 12 heuristics, which the authors design themselves or incorporate from the literature. An exception is the approach by [12] where a large number of low-level heuristics are assembled from parametrised components, followed by a step-by-step pool reduction strategy operating on a specific instance at execution time. The approach requires fixing the minimum number of heuristics desired (20 in [12]) from a larger available pool of assembled heuristics, and a fixed number of iterations from which a reduction step is executed. When a reduction step occurs, the low-level heuristics with the smallest total improvement over all previous iterations is discarded.

Our reduction methodology, described in detail in section 3 differs from [12] in two important ways. First, it does not require any extra parameters other than the complete set of initial heuristics which are supplied by the user. Second, it acts as a prepossessing step considering performance data from a set of instances of the underlying problem. Therefore, once it is completed, no additional overhead is required to solve new instances of the same problem.

2.2. Automated algorithm configuration (offline tuning)

Several frameworks have been proposed in the literature for automated parameter tuning and algorithm configuration [4, 19, 15, 18]. We describe here *ParamILS*, a framework for automated algorithm configuration achieved with local search in the configuration space. The key idea behind ParamILS is to combine a stochastic local search algorithm (iterated local search) with mechanisms for exploiting specific properties of algorithm configuration. [19] present extensive evidence that ParamILS can find substantially improved parameter configurations of complex and highly optimised algorithms.

Offline tuning tools are relevant to hyper-heuristic research. In selection hyper-heuristics, a selection probability is associated to each operator or heuristic, and these are dynamically adapted during the search process. However, these probabilities can alternatively be considered as fixed (static) parameters of the hyper-heuristic algorithm, and this can then be automatically tuned with an offline tuning tool. Thereafter, during the hyper-heuristic tun, operators are selected by a roulette-wheel mechanism based on these statically-tuned probabilities. We applied this method as a

control mechanism to compare against the proposed adaptive hyper-heuristic in [43], and found that the adapting the selection probabilities online produced significantly better results.

Our proposed methodology uses ParamILS in two stages. First, to contrast tuned operator selection probabilities against adaptive methods, regardless of how many heuristics are considered in the pool. Second, as a method for estimating the performance of all the available individual heuristics, and thus serve as an alternative, although computationally expensive, ranking mechanism for the first step of the proposed reduction methodology described in the next section.

3. Methodology for determining an effective heuristic subset

The proposed methodology requires an available pool of heuristics and a set of benchmark instances of the underlying problem. It uses fitness landscape metrics and non-parametric statistical tests to rank the heuristics. The ranked heuristics are thereafter grouped incrementally until the best subset is determined. Specifically, the methodology comprises the following three stages:

1. *Statistical ranking of low-level heuristics*: non-parametric statistical tests (described in more detail in section 3.1) are used to rank the estimated performance of each low-level heuristic. This requires a metric for estimating the heuristic's individual performance. We chose two alternative metrics to probe the search landscape. The first is based on the notion of *evolvability* [38] and the second on the concept of *landmarking*, where the performance of simple and quick algorithms is used to characterise the relative difficulty of the problem instances [39]. Section 3.2 gives more detailed descriptions of the landscape probing techniques employed.
2. *Grouping low-level heuristics*: subsets of increasing size are formed according the the statistical ranking of the low-level heuristics. Specifically, the size of the subsets is increased one by one starting from the top two heuristics according the non-parametric ranking, then the top three, and so on, until the whole set is considered. The hyper-heuristic under study is executed using each heuristic subset.
3. *Selecting the best subset of heuristics*: the subset of heuristics producing the best hyper-heuristic performance from step 2 is considered as the best subset. The performance metric employed will depend on the problem under consideration. Our study uses the best subset configuration obtained after a fixed running time.

3.1. Non-parametric tests

Parametric statistical tests have been used to contrast the performance of heuristic search methods. However, they assume independence, normality and homoscedesticity of the data, which are not guaranteed in the case of heuristic algorithms. Non-parametric statistical procedures overcome this limitation and can be used for comparing this type of algorithms. We used CONTROLTEST [14], a tool specially designed for non-parametric comparison among heuristic algorithms, and considered the three tests there proposed: Friedman, Aligned Friedman, and Quade. The Friedman test uses the arithmetic mean, Alignment Friedman uses a value of location computed as the average performance achieved by all algorithms in each problem, and Quade considers that some problems might be more difficult than others. All these tests consider ranks, therefore, the lower the reported value the better the performance achieved.

3.2. Fitness landscape probing techniques

The two fitness landscape probing techniques described below are proposed to estimate the performance of individual operator.

Evolvability: The notion of evolvability loosely refers to the capacity of an individual or population to evolve. The first formalisation of this notion is attributed to [1] who defined evolvability as "the ability of the genetic operator/representation scheme to produce offspring that are fitter than their parents." We are interested in measuring the evolvability of specific search operators. In a previous study [40] used three evolvability metrics to determine the credit of operators while solving a given problem instance. Here, we use the best performing metric in that study, termed E_a, which was originally proposed by [38]. This metric represents the probability that the fitness of neighbouring solutions is better than or equal to the fitness of the incumbent solution. It is simply measured as the ratio between the number of neighbours with improved or equal fitness as compared to the incumbent solution, and the size of the whole neighbourhood (when sampling this is the sample size). In order to estimate the evolvability of a

given operator on a given problem, we use a sampling procedure considering the whole set of available instances. The estimated operator's evolvability is the average evolvabilities across the instance set. On a given instance, the procedure for estimating the the evolvability of operator (heuristic) h_i is as follows.

1. Generate an initial solution, s_0, uniformly at random
2. Generate 100 representative neighbours of s_0 by single applications of h_i. The representative neighbours are sampled with a *Metropolis-Hastings* approach which gives prevalence to best points (i.e. those with higher fitness) [44]
3. Count, c, how many of these 100 representative neighbours have better or equal fitness than that of s_0
4. Calculate the evolvability as the ratio $E_a = \frac{c}{100}$

The procedure above is repeated 500 times from different initial solutions, and the evolvability of the operator h_i on the given instance is the average of these 500 measures. The number of representative neighbours was set to 100 in order to achieve statistical significance.

Landmarking: As a second metric, we consider the performance of the simplest possible search algorithm using a single operator h_i as a probing technique. This corresponds to a first-improvement hill-climbing method, outlined in Algorithm 1. The stopping condition is set to 100 iterations in order to have similar computational budget as that used in the evolvability probing technique. The result of a run is summarised by the fitness difference between the initial and the final solution. On each instance, the hill-climbing algorithm is run 500 times from different initial randomly generated solutions, and the quality of the operator h_i is the average of these 500 fitness difference values. The estimated operator's quality is the average of qualities across the whole instance set.

Algorithm 1 First-improvement Hill-climbing

Require: $InitialSolution\ s0, Heuristic\ h_i, FitnessFunction\ f$

```
s ← s0
while !StopCriteria() do
    s* = apply(h_i, s)
    if f(s*) < f(s) then
        s = s*
    end if
end while
return ls
```

3.3. Alternative ranking of low-level heuristics

In order to test the benefits of the proposed statistical ranking of low-level heuristics (first step of the methodology), we used ParamILS (see section 2.2) as an alternative approach to rank the heuristics. The idea is to learn a selection probability for each heuristics in the pool using all the benchmark instances as the training set. The experiment is, therefore, very computationally expensive (running times are reported in the case studies below). Each ParamILS iteration executes the hyper-heuristic under consideration with the complete heuristic pool on the whole training set. The fixed heuristic selection probabilities are thus evolved by the ParamILS search process, which runs for 1000 iterations. To achieve statistical significance, 35 times runs are considered. The outcome of the experiment is a list of selection probabilities for the heuristics in pool. Thereafter, heuristics are ranked in decreasing order of their selection probabilities.

4. Dynamic multi-armed bandit hyper-heuristic

This section describes the proposed hyper-heuristic variant, which extends our recent proposal [43]. A selection hyper-heuristic algorithm has three main components: the high-level search strategy, the pool of operators, and the adaptive or control mechanism to dynamically select the operator to apply at each search step. We describe below the high-level strategy and adaptive operator selection mechanism used. The pool of operators is normally problem-specific. Sections 5.1 and 5.2 describe the operators used for the two selected problem domains, respectively.

4.1. High-level strategy

An iterated local search framework is used as the high-level strategy (Algorithm 2). We choose Iterated Local Search (ILS) as it is a simple yet powerful search strategy that has produced outstanding results in practice [25]. ILS operates by iteratively alternating between applying a move (perturbation) operator to the incumbent solution and restarting local search from the perturbed solution. Moreover, a number of adaptive variants of multi-neighbourhood iterated local search (also called iterated local search hyper-heuristics) have been recently proposed [30, 45, 43] with encouraging results in various problem domains.

In our implementation (outlined in Algorithm Algorithm 2), the adaptive control mechanism is applied to the improvement stage, in which a local search heuristic is selected from the available pool and then applied to the incumbent solution (line 5). The perturbation stage uses a fixed randomised operator, and the acceptance condition simply accepts all improvements. This implementation differs from our previous ILS hyper-heuristic [43] in the operator control mechanism for selecting heuristics, which is detailed in Section 4.2.

Algorithm 2 High-level strategy: Iterated local search

1: $s_0 = GenerateInitialSolution$
2: $s^* = ImprovementStage(s_0)$
3: **while** $!StopCriteria()$ **do**
4: $\quad s' = SimpleRandomPerturbation(s^*)$
5: $\quad s^{*'} = ImprovementStage(s')$
6: $\quad$ **if** $f(s^{*'}) < f(s^*)$ **then**
7: $\quad\quad s^* = s^{*'}$
8: $\quad$ **end if**
9: **end while**
10: **return** s^*

4.2. Operator selection

We borrow ideas from *adaptive operator selection* (AOS) in evolutionary algorithms [17, 16]. Two cooperating mechanisms are required in this process: A *selection rule*, which defines how the next operator or low-lever heuristic to be applied next should be chosen according to its estimated quality; and a *credit assignment* mechanism, which defines how to estimate operators quality based on the impact brought by their most recent application. For the selection rule, we used dynamic multi-armed bandits (DMAB) proposed by [13]. The multi-armed bandit framework is commonly used in game theory for studying the *exploration vs. exploitation* dilemma. It involves N arms and a decision making-algorithm for selecting one arm at each time step with the goal of maximising the cumulative reward gathered along time. The exploration vs. exploitation balance is relevant for heuristic search. Indeed, adaptive operator selection can be formulated using multi-armed bandits with arms corresponding to search operators [13, 16]. Specifically, the *upper confidence bound* multi-armed bandit [2] was used as it provides optimal maximisation of cumulative rewards. The use of DMAB as an operator selection rule has reported encouraging results within selection hyper-heuristics [43, 40, 36]. For credit assignemnt, we use an *extreme value* criterion, as it has produced better results than other criteria in both evolutionary algorithms [17, 16] and hyper-heuristics [40]. These mechanisms are described in detail below. The most common way of assigning credit is to account for the fitness improvement brought by the operators. That is, the fitness difference of the generated offspring with respect to a reference value, which is generally taken as the parent.

DMAB selection rule: Each operator is viewed as an arm. Let $n_{i,t}$ denote the number of times the i^{th} arm has been played, and $\hat{p}_{i,t}$ the average empirical reward it has received up to time t. At each time step t, from K alternative arms, the algorithm selects the arm maximising the quantity computed by equation 1.

$$\hat{p}_{j,t} + C\sqrt{\frac{2log(\sum_{i=1}^{K} n_{i,t})}{n_{j,t}}} \tag{1}$$

Two considerations were required to use this framework for adaptive operator selection. First, a scaling factor C is needed, in order to properly balance the trade-off between selecting the operator with best empirical behaviour

(exploitation, first term in Eq.1) and giving other operators the opportunity to be selected (exploration, second term in Eq.1). Second, The multi-armed bandit framework is combined with the *Page-Hinkley* statistical test for detecting changes in the operator selection dynamic. This hybridisation introduces two additional parameters associated to the *Page-Hinkley* test: γ, which controls the trade-off between false alarms and unnoticed changes; and δ, which enforces the robustness of the test when dealing with slowly varying environments. Parameters C and γ need to be tuned for every problem. We found in preliminary experiments that $C = 10$, $\gamma = 100$ for course timetabling, and $C = 8$, $\gamma = 105$ for vehicle routing produced consistently good results. For the parameter δ, we used the value suggested in [16] ($\delta = 0.15$) for all the experiments. We selected DMAB as the selection rule due its hybridisation with the Page-Hinkle test, and its good performance achieved in previous works. [16, 17, 43]

Extreme value credit assignment: Rewards are updated as follows, when a heuristic h is selected, it is applied to the current solution. The fitness of this new solution is computed and the change in fitness Δ_f is added to a FIFO list of size W. A separate list is kept for each operator. Thereafter, the operator reward is updated to the maximal fitness improvement in the list. More formally, let t be the current step and $\Delta_f(t)$ the fitness improvement observed at time t, the expected reward for heuristic h is computed using equation 2.

$$\hat{r}_t = argmax\{\Delta_f(t_i), i = 1...W\} \tag{2}$$

5. Test problems

Hyper-heuristics are intended to be general approaches with good performance across several domains. However, many hyper-heuristic articles in the literature consider a single problem domain. We therefore, tested our approach on two challenging real-world problems with available benchmark instances: course timetabling and vehicle routing.

5.1. Course timetabling

Course timetabling requires the assignment of a fixed number of subjects into a number of time-slots. The main objective is to obtain a timetable minimising the number of student conflicts. Our formulation uses a generic modelling approach where solutions are represented as vectors of integer numbers of length equal to the number of events (courses). Positions in the vector represent events, and their integer values are indices in a set of data structures encoding pairs of valid time-slots and classrooms for each event [43].

Many approaches have been proposed for solving variants of educational timetabling using metaheuristics, [5, 26, 11, 24] and hyper-heuristics [8, 34, 42, 41]. Recent surveys have also been published [35, 32, 33, 3].

Our implementation considers the set of 9 low-level heuristics proposed in our recent state-of-the-art hyper-heuristic for course timetabling [43]. They range from simple randomised exchange or swap neighbourhoods to greedy and more informed procedures. Table 1 provides a brief description of these heuristics, more details can be found in [43].

5.2. Vehicle routing

Vehicle routing problems (VRP) require that a fleet of vehicles serves a number of request in order to minimise costs. A number of variants have been proposed. Our formulation follows that implemented by [45], for the HyFlex hyper-heuristic framework [29]. Specifically, we consider the vehicle routing problem with time windows, whereby a customer must be served between two time points for a solution to be valid. The objective function balances the dual objectives of minimising the number of vehicles needed and minimising the total distance travelled. It is defined as follows:

$$VrpFitness = c \times numVehicles + distance$$

where c is a constant empirically set to 1000 to give higher importance to the number of vehicles in a solution.

Several heuristic search approaches have been successfully applied to vehicle routing, surveys have also been published [21, 31].

Our implementation uses the HyFlex hyper-heuristic framework [29], which includes low-level heuristics of different types: mutational, ruin-recreate, improvement and crossover. Improvement heuristics refer to greedy local search procedures that guarantee an improvement of the incumbent solution. The vehicle routing implementation [30]

Table 1: Course timetabling low-level heuristics.

Operator	Description
MLC	Locates the variable producing the most conflicts and changes its value to the that causing the minimum possible conflict.
BSP	Chooses a variable following a sequential order and changes its value to that producing the minimum conflict.
WMLC	Locates the variable producing the worst conflict and changes its value to that causing the minimum possible conflict.
MLS	Changes the value of a given variable to that causing the event to move to the less occupied time-slot.
Swap	Selects two variables uniformly at random and interchanges their values if possible. Otherwise leaves the solution unchanged.
Simple Mut	Chooses uniformly at random a variable and changes its value for another one inside its feasible domain.
SDP	Chooses a variable following a probability distribution based on the frequency of variable selection in the last k iterations.
DDP	Similarly to SDP Selects a new variable with a probability inversely proportional to its frequency of selection in the last k iterations. It differs from SDP in that it internally maintains an additional solution (which is a copy of the first initialised solution) and makes random changes to it following the same distribution.
Two Points	Selects uniformly at random two indexes in the integer string representation and modifies all variables between the indexes randomly.

provides a total of 12 low level heuristics across the four categories. From these, we selected 8 that were relevant to our implementation. These correspond to the all the mutational and ruin-recreate heuristics avialable (4 and 2, respectively) and 2 of the improvement heuristics available. The four heuristics left out, correspond to 2 crossover operators, and 2 local search heuristics that implement iterative versions of two of the already included mutational heuristcis (specifically, *shift* and *interchange*). A short description of these heuristics is provided in Table 2, more details can be found in [30].

6. Course timetabling: results and discussion

We first apply the proposed methodology and hyper-heuristic variant to the post-enrolment course timetabling problem. The experimental conditions resemble those of the *International timetabling competition (ITC) 2007* track 2 (post-enrolment course timetabling) [23]. A total of 24 instances are available. The objective function minimises the sum of hard and soft constraint violations. The number of hard constraints violations is termed the *distance to feasibility* metric, and it is defined as the number of students that are affected by unplaced events.

Table 3 reports the statistical rankings of the 9 low-level heuristics according to the two landscape probing metrics discussed in section 3.2, and the alternative ranking using ParamILS as described in section 3.3 in the last column. Heuristics are ordered according to the Quade rank and the hill-climbing landmarking metric, Lnd_{Hill}. The Quade test was selected as it considers both algorithm performance and the dispersion of results as evidence of their robustness (algorithms with higher dispersion are considered worse). The hill-climbing landmarking metric was selected as it is more informative. This is illustrated in Figures 1 and 2, showing the distribution of evolvability and hill-climbing landmarking metrics, respectively. Both metrics give an indication of the heuristics comparative performance. However, evolvability represents the heuristics' quality in the range from 0 to 1, in consequence, heuristics with high evolvability have similar values near to 1. Hill-climbing landmarking, instead, measures differences in fitness values,

Table 2: Vehicle routing low-level heuristics.

Operator	Description
Two-opt	Swaps two adjacent locations within a single route.
Or-opt	Moves two adjacent locations to a different place, within a single route.
Shift	Moves a single location from one route to another.
Interchange	Swaps two locations from different routes.
LocRR	Removes a number of locations based on location proximity, reinserting into the best route possible.
TimeRR	Removes a number of locations based on time window proximity, reinserting into the best route possible.
Two-opt*	Takes the end sections of two routes, and swaps them to create two new routes.
GENI	A location is taken from one route, and placed into another route, between the two locations of that route which are closest to it. Re-optimisation is then performed on the route.

Table 3: Course timetabling: non-parametric ranking of low-level heuristics (columns 2 - 6), and selection probabilities as tuned by ParamILS (last column). E_a stands for evolvability and Lnd_{Hill} for hill-climbing landmarking. Rankings are ordered according to the Quade test and Lnd_{Hill} metric.

Heuristic	Friedman		Aligned Friedman		Quade		ParamILS
	E_a	Lnd_{Hill}	E_a	Lnd_{Hill}	E_a	Lnd_{Hill}	*Prob.*(%)
MLC	2.7	1.09	42.4	20.52	2.51	1.16	30.3%
MLS	2.9	2.59	44.3	40.26	2.88	2.53	18.1%
BSP	3.04	2.73	42.71	48.21	3.36	2.70	22.7%
MWLC	1.14	4.23	40.57	73.90	1.233	3.99	12.1%
Two Points	5.19	4.38	100.61	83.85	5.33	4.67	6.0%
DDP	7.04	6.90	137.42	139.95	6.92	7.05	1.5%
SDP	6.09	7.04	133.42	139.95	6.88	7.05	3.0%
Simple Mut	6.08	7.28	134.53	142.19	6.85	7.24	4.5%
Swap	9.0	8.71	178.9	166.14	9.0	8.54	1.5%
$p-value$ (E_a)	1.10E-10		0.0168		1.94E-37		
$p-value$ (*Hill*)	1.28E-75		0.0161		6.77E-36		

and thus is not restricted to a specific interval. This property of Hill-climbing landmarking is important for the non-parametric ranking since this ranking is determined by performance differences between operators. This evidence suggest that Hill-climbing landmarking produces most suitable information to carry out the operator ranking. Another disadvantage of evolvability is that it uses the Metropolis-Hastings inner sample validation, which incurs in around 30% more fitness evaluations than those required to calculate the hill-climbing landmarking metric.

The computational time required by the heuristic subset selection methodology is detailed in Table 4 measured as CPU hours in Core i7 processor with 8 Gigabytes of RAM. Using ParamILS for ranking heuristics requires about ten times more computational effort than using the landscape probing metrics followed by statistical ranking.

Table 5 summarises the second step of the proposed methodology. It shows the results of executing the proposed hyper-heuristic considering heuristic subsets of increasing size, where the heuristics are ordered according to Table 3. The cost of a solution is denoted with a pair of values, (hv, sv), where hv and sv are the sum of hard and soft constraint violations, respectively. In order to compare two or more solutions, the pairs (hc, sc) are ranked in a lexicographical ascending order. Following the *ITC-2007* rules, 10 independent runs per instance were conducted, and results are reported as the average and standard deviation of the pair (hc, sc). The stopping condition for each run is is set to 300 CPU seconds, which is half of the alloted running time in the competition, as this is a pre-processing stage. Bold

Table 4: Course timetabling. Computational time in CPU hours required by the heuristic sub-set selection methodology.

Statistical Ranking			*Grouping Heuristics*
Landmarking	Evolvability	ParamILS	
20 hrs	25 hrs	195 hrs	120 hrs

Table 5: Course timetabling. Hyper-heuristic performance, as average of 10 independent runs, with heuristic subsets of increasing size. *hc* accounts for hard constraint and *sc* for soft constraint violations, which are to be minimised. Decimals are rounded due to space limitations. Bold fonts highlight best performance, which consistently occurs with the subsets of 4 and 5 heuristics.

	2		3		4		5		6		7		8		9	
	hc	*sc*	*hc*	*sc*	*hc*	*sc*	*hc*	*sc*	*hc*	*sc*	*hc*	*sc*	*hc*	*sc*	*hc*	*sc*
1	387	2484	117	2252	**104**	**2132**	131	2209	113	2257	121	2311	104	2259	159	2361
2	425	2340	135	2408	**108**	**2496**	140	2360	123	2088	126	2318	192	2268	165	2278
3	82	2766	19	1835	20	1866	**14**	**1898**	29	2079	26	2060	19	2032	44	1985
4	134	2459	44	2134	47	2555	**29**	**2010**	36	2497	38	2515	26	2162	60	2444
5	171	1349	82	1225	**55**	**1242**	80	1277	58	1270	77	1304	71	1273	85	1440
6	184	1401	78	1298	**62**	**1270**	71	1235	74	1155	39	1228	46	1267	76	1294
7	74	1478	22	1189	**17**	**1067**	22	1020	23	1170	25	1276	25	1129	28	1116
8	63	1335	12	1075	**13**	**1168**	17	1301	20	1115	16	1008	20	1168	14	1108
9	371	2369	111	2362	**95**	**2270**	116	2364	102	2332	99	2417	97	2248	114	2378
10	402	2417	125	2190	**109**	**2261**	158	2526	122	2281	137	2184	147	2322	185	221
11	104	2690	36	1993	30	2039	**17**	**2079**	24	1965	38	2194	24	2169	43	2437
12	132	2671	32	2396	**27**	**2218**	30	2316	42	2323	31	2294	55	2179	54	2442
13	241	1458	105	1291	94	1356	**91**	**1374**	92	1235	104	1340	98	1423	128	1357
14	208	1377	71	1266	74	1344	**65**	**1270**	83	1273	78	1272	92	1325	88	1334
15	54	1259	12	1108	18	1176	**8**	**1070**	11	971	11	1092	16	1283	14	1041
16	48	1201	8	1026	3	922	**0**	**859**	5	955	1	963	1	938	6	931
17	28	1957	0	1115	0	805	**0**	**775**	0	990	0	1150	2	1183	2	1048
18	52	141	48	97	42	85	**36**	**73**	39	79	44	102	49	83	49	98
19	28	921	16	915	15	805	**12**	**475**	25	490	30	650	38	883	22	848
20	130	2769	17	1956	13	1954	**9**	**1893**	11	1936	21	1892	16	1934	23	1941
21	29	57	32	45	7	35	**0**	**23**	0	50	5	42	12	43	15	48
22	476	2311	239	2365	249	2149	**223**	**2228**	235	2231	236	2235	272	2192	292	2301
23	1011	5014	320	4951	**306**	**4697**	417	5003	387	4910	374	5276	268	4906	555	4770
24	197	2627	41	1987	38	1889	**29**	**1933**	48	2029	47	2033	44	1841	43	1871

fonts highlight the best results obtained, which indicate that the groups of 4 and of 5 heuristics produce consistently the best performance for all the instances. As the most effective subset, we select the group of 5 heuristics as it has slightly better performance and provides more options for the adaptive search strategy. Specifically, the 5 heuristic selected are: MLC, MLS, BSP, MWLC and Two Points. Table 3 orders the heuristic according to the Quade rank and the hill-climbing landmarking metric. However, the 5 heuristics selected correspond to the top performing according to all the rankings in Table 3, which gives strong evidence of the effectiveness of the subset selection.

6.1. Contrasting operator selection strategies

In order to assess whether adapting selection probabilities while searching is the best performing strategy, this section compares it with two alternative strategies, namely, selecting heuristics uniformly at random, and learning selection probabilities offline, which can be seen as a form of parameter tuning. The random strategy serves as a baseline; it simply selects uniformly at random a heuristic from the pool at each iteration. The off-line or *tuning* approach considers the operator selection probabilities as static parameters that can be tuned using existing tuning tools. As described in section 2.2, we use ParamILS as a tuning tool. The heuristic subset selected corresponds to the top 5 heuristic identified above, namely, MLC, MLS, BSP, MWLC, and Two Points. For the strategy using parameter tuning, Table 6 reports the static operator selection probabilities obtained using ParamILS, expressed as percentages.

Table 6: Course timetabling: Static operator selection probabilities (expressed as percentages) obtained with parameter tuning.

MLC	MLS	BSP	MWLC	TwoP
33.1%	17.5%	21.3 %	18.6%	9.5%

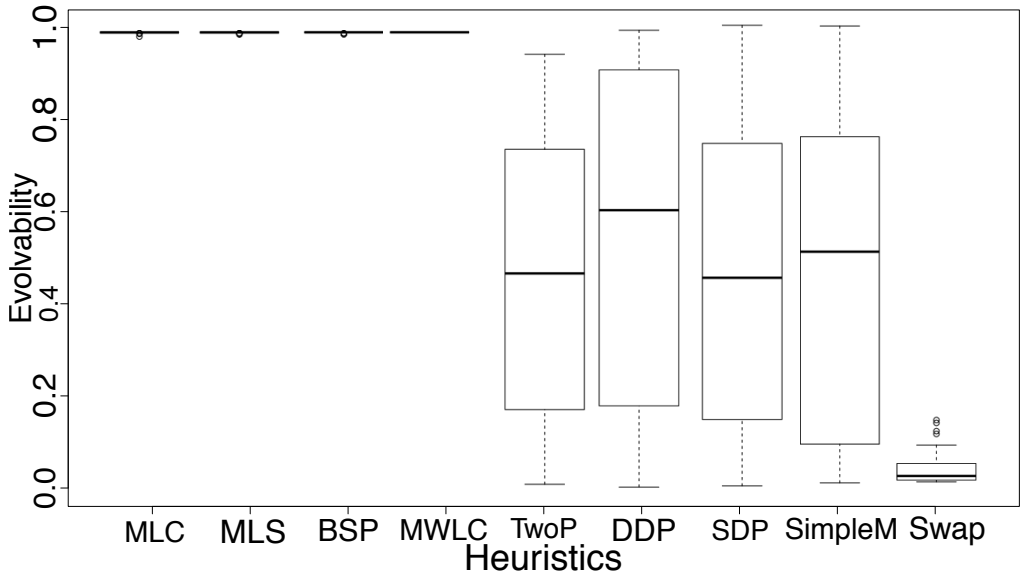

Figure 1: Distribution of evolvability E_a values for each heuristic across 500 experiments. Course timetabling instance *ITC-2007-4*.

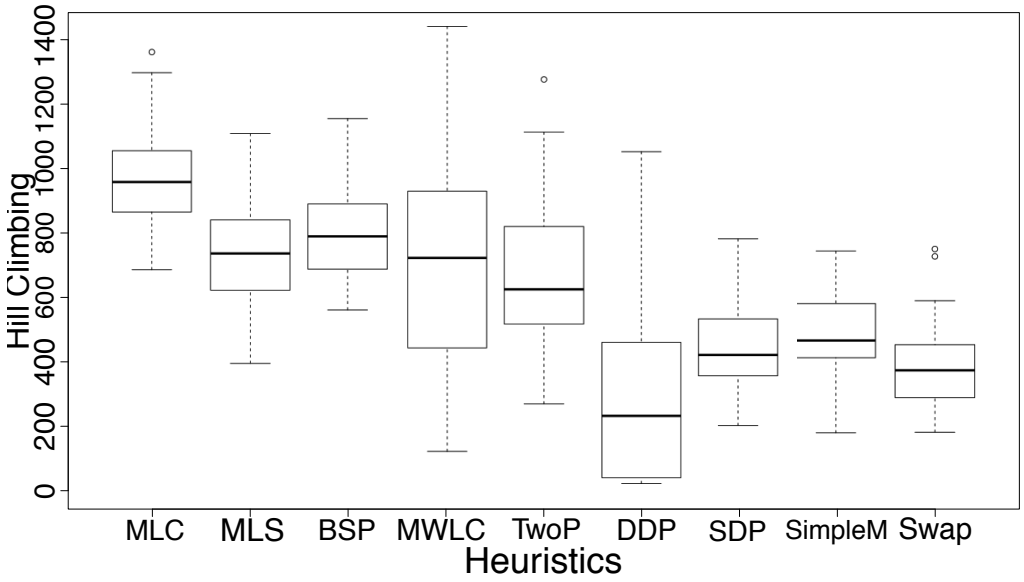

Figure 2: Distribution of hill-climbing landmarking values for each heuristic across 500 experiments. Course timetabling instance *ITC-2007-4*.

Following the ITC-2007 rules, each hyper-heuristic variant is run 10 times, and the stopping condition corresponds to a time limit of about 10 minutes following the benchmark algorithm provided in the competition website[1]. Table 7 compares the three operator selection strategies: random, tuned and adaptive within the proposed hyper-heuristic using the effective subset of 5 low-level heuristics. Values correspond to averages of 10 runs. The best results are achieved consistently using the proposed adaptive strategy, which attains the minimum possible cost for the hard

[1] http://www.cs.qub.ac.uk/itc2007/

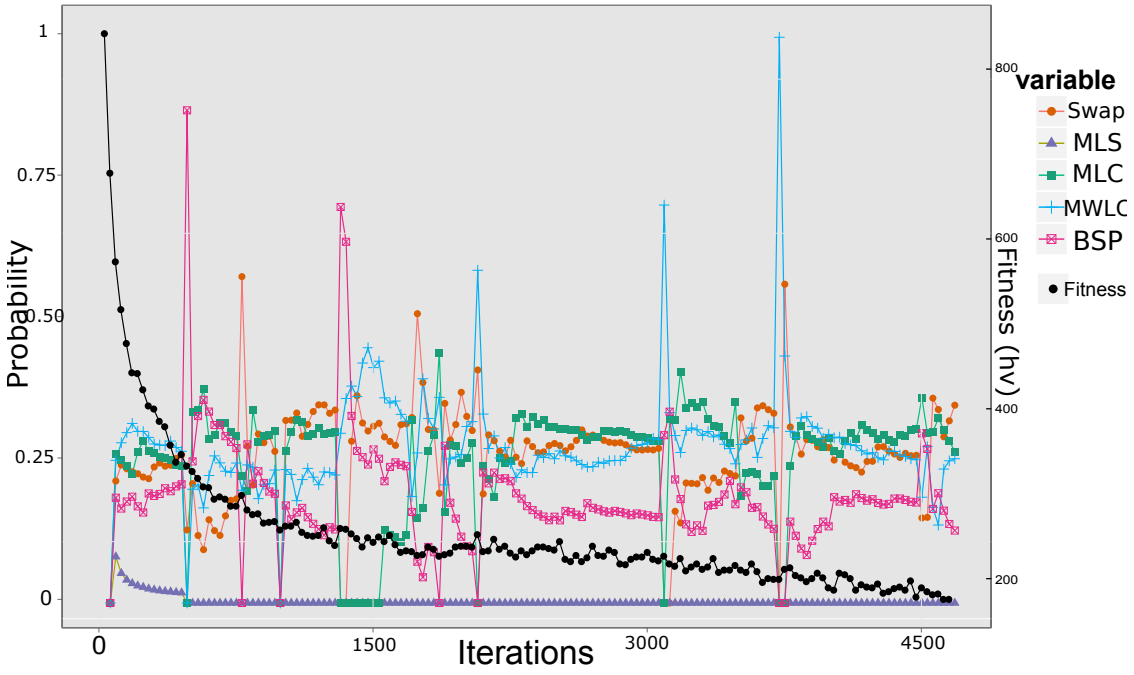

Figure 3: Course timetabling, *ITC-2007* instance 1: Operator selection dynamic with the selected sub-set of 5 heuristics. Curves show probabilities over time. The fitness over time is also displayed (black line)

constraint violation in all instances. We argue that this method successfully adjusts operator selection probabilities for each instance under consideration. An example of adaptation can be seen in Figure 3, where the selection probabilities of each operator over time are visualised. The curves reveal a fast adaptation to changes in the operator capacities to guide the search process. The spikes in the curves are evidence of the change detection mechanism at work within the dynamic multi-armed bandit strategy.

6.2. Contrasting against the state-of-the-art

This section compares the proposed hyper-heuristic, which we now term *HHDMAB*, with state-of-the-art methods on the course timetabling problem. The comparison includes both recent hyper-heuristics and problem specific approaches in the literature. Specifically, the contenders are:

Cambazard: the winner of the *ICT-2007* competition [10], a multi-stage local search algorithm considering several neighbourhoods.

Ceschia: a single-step meta-heuristic approach based on simulated annealing, with a neighbourhood composed of moves that reschedule one event or swap two events [11].

Lewis: a 3-stage local search algorithm, in which a constructive phase is followed by two separate simulated annealing phases [22].

Jat & Yang: a hybrid approach, which applies a guided genetic algorithm integrating local search techniques [20].

AdapExAP: the previous version of our adaptive iterated local search hyper-heuristics [43], which uses the full set of 9 neighbourhoods described in Table 1 coupled with an adaptive mechanism based on the *adaptive pursuit* selection rule.

HHADL: an iterated local search hyper-heuristic with Add-Delete list, this algorithm generate heuristics based on a fixed number of add and delete operations [41].

Table 7: Course timetabling: Contrasting operator selection strategies: Random, Tuned and Adaptive, using the effective subset of heuristics. *hc* accounts for hard constraint and *sc* for soft constraint violations, which are to be minimised. The Adaptive strategy clearly outperforms its counterparts.

ITC-2007	Random		Tuned		Adaptive	
Instance	*hc*	*sc*	*hc*	*sc*	*hc*	*sc*
1	129	2439.12	35.18	1773.12	0	860.12
2	142.3	2213.56	29.77	1856.14	0	530.15
3	22.11	2174.55	7.43	762.83	0	315.12
4	49.12	2504.75	27.45	562.88	0	559.12
5	62.31	1285.15	0	635.32	0	0
6	81.22	1301.52	0	350.65	0	4.6
7	23.91	1106.73	0	456.23	0	5.2
8	11.1	1085.27	0	441.93	0	0
9	127.52	2268.65	35.13	1674.75	0	854.3
10	171.66	2408.8	15.93	954.25	0	615.12
11	25.78	2515.72	17.1	775.14	0	355.12
12	43.75	2190.52	12.65	1208.14	0	278.43
13	98.15	1263.93	0	588.14	0	15.2
14	93	1309.15	0	516.34	0	25.12
15	19.53	1128.32	0	788.37	0	44.16
16	6.3	822.15	2.14	598.15	0	73.76
17	0	1365.12	0	332.9	0	3
18	35	1365.12	0	162.9	0	28.4
19	35	485.12	15	231.5	0	183.12
20	38.43	2229.15	15.34	1298.45	0	311.07
21	75	161.12	5	62.9	0	12.09
22	277.36	2301.55	0	1190.34	0	354.4
23	436.15	4809.76	42.15	2766.14	0	515.23
24	46.33	2128.69	37.44	692.49	0	337.34

Table 8 shows the best achieved results as well as the average and standard deviation of our experiments, our algorithm *HHDMAB* presents consistently better results than *AdapExAP* who is the initial approach from which HHDMAB was developed, also HHDMAB algorithm presents also new best-know solutions for instances 3, 9,12,20 and 22 from Course Timetabling state of art.

7. Vehicle routing: results and discussion

The second case study considers the well known vehicle routing problem. The formulation and experimental setting follow the rules of the *Cross-Domain Heuristic Search Competition (CHeSC) 2011* competition [29]. CHeSC Instances were taken from [37] and include 5 from the Solomon data set and 5 from the Gehring and Homberger data set. Both data sets include three types of instances: Random, Clustered, and mixed Random-Clustered; according to the way in which the customers' locations are determined, more details can be found in the Appendix.

Table 9 summarises the first step of the proposed methodology, that is, the statistical rankings of the 8 low-level heuristics according to the two landscape probing metrics discussed in section 3.2, and the alternative ranking using ParamILS (section 3.3) in the last column. Rankings are ordered following the Quade test and the hill-climbing landmarking metric, Lnd_{Hill}, as discussed in the previous case study.

The computational time required by the heuristic subset selection methodology is detailed in Table 10 measured as CPU hours in Core i7 processor with 8 Gigabytes of RAM. As for the course timetabling study, using ParamILS for ranking heuristics requires about ten times more computational effort than using the landscape probing metrics.

Figures 4 and Figure 5, show the distribution of evolvability and hill-climbing landmarking metrics, respectively. As in the previous case study, both metrics give a similar indication of the heuristics comparative performance. How-

Table 8: Course Timetabling. Comparison against state-of-the-art. Values indicate the best soft constraint results , in all cases solutions are feasible, i.e. the hard constraints are 0. Average and standard deviation results are also reported in brackets in the form ($\bar{s}_{\sigma}$) for the two hyper-heuristic approaches. Bold font indicates the best result per instance.

ITC-2007	*Cambazard*	*Ceschia*	*Lewis*	*Jat & Yang*	*AdapExAP*	*HHADL*	*HHDMAB*
1	571	**59**	1166	501	$650(780.45_{148.5})$	630	630 $(860.12_{110.7})$
2	993	**0**	1665	342	$470(960.7_{270.4})$	450	380 $(530.15_{130.6})$
3	164	148	251	3770	$290(337_{88.7})$	300	**137** $(315.12_{57.15})$
4	310	**25**	424	234	$600(815_{42.6})$	602	75 (559.12)
5	5	**0**	47	**0**	$35(39.16_{9.3})$	6	**0**(0_0)
6	**0**	**0**	412	**0**	$20(29.4_{7.3})$	**0**	**0** $(4.6_{1.4})$
7	6	**0**	6	**0**	$30(33.74_{2.1})$	**0**	**0** $(5.2_{1.1})$
8	**0**	**0**	65	**0**	$0(0_0)$	**0**	**0** (0_0)
9	1560	**0**	1819	989	$630(861.1_{127.4})$	640	**602** (854.3.15)
10	2163	**3**	2091	499	$2349(2458.2_{185.2})$	663	$482(615.12_{72.14})$
11	178	142	288	246	$350(405.7_{57.3})$	344	**159**$(355.12_{49.12})$
12	146	267	474	172	480 $(506.4_{27.4})$	198	**140** $(278.43_{25.9})$
13	**0**	1	298	**0**	$46(77.37_{49.2})$	**0**	**0** $(15.2_{12.7})$
14	1	**0**	127	**0**	$80(108.3_{33.5})$	35	$20(25.12_{17.4})$
15	**0**	**0**	108	**0**	**0** $(5.75_{9.4})$	**0**	12 $(44.16_{16.0})$
16	2	**0**	138	**0**	$0(2.22_{4.1})$	140	**0**$(73.76_{33.4})$
17	**0**	**0**	**0**	**0**	$0(0_0)$	**0**	**0**$(3_{1.5})$
18	**0**	**0**	25	**0**	$20(25.16_6)$	**0**	**0** $(5.3_{2.1})$
19	1824	**0**	2146	84	$360(404.51_{39.1})$	400	$133(214.6_{33.6})$
20	445	543	625	297	$150(177.12_{37.1})$	150	**106** $(311.07_{31.5})$
21	**0**	5	308	0	**0** $(3.78_{5.7})$	**0**	**0**$(2.5_{2.3})$
22	29	5	x	1142	$33(45.71_{12.7})$	32	**25** $(354_{18.5})$
23	**238**	1292	3101	963	$1007(1378.45_{319.4})$	238	$267.4(515.23_{89.4})$
24	21	**0**	841	274	$0(45.88_{60.0})$	640	$76(337.34_{51.9})$

Table 9: Vehicle routing. Non-parametric ranking of low-level heuristics (columns 2 to 6), and selection probabilities as tuned by ParamILS (last column) E_a stands evolvability and Lnd_{Hill} for hill-climbing landmarking. Rankings are ordered according to the Quade test and Lnd_{Hill} metric.

Heuristic	Friedman		Aligned Friedman		Quade		ParamILS
	E_a	Lnd_{Hill}	E_a	Lnd_{Hill}	E_a	Lnd_{Hill}	Prob(%)
TimeRR	3.9	1.40	26.1	15.2	3.76	1.52	19.4%
TwoOptStar	1.1	2.59	16.9	21.1	1.10	2.70	14.5 %
locRR	2.8	2.6	22.6	19.1	2.98	2.65	18.1%
ShiftMutate	2.3	3.4	18.79	28.59	2.23	3.10	14.5 %
GENI	5.3	5.0	40.0	58.50	5.42	5.0	1.9 %
Interchange	5.9	6.5	51.79	59.99	5.8	6.45	9.7%
TwoOpt	6.85	6.64	65.85	60.15	6.93	6.75	14.5 %
OrOpt	7.75	7.85	72.85	61.34	7.7	7.79	5.8 %
$p-value$ (E_a)	1.67E-30		0.27		6.68E-17		
$p-value$ $(Hill)$	6.13E-11		0.026		5.22E-16		

ever, hill-climbing landmarking gives more information in terms of the difference between heuristics, which is a desired property for the next stage of the proposed methodology.

Table 11 summarises the second step in the selection methodology. It shows the performance of the proposed hyper-heuristic variant considering heuristic subsets of increasing size. Notice that the best results are mostly achieved when using the group of 4 heuristics. Therefore this group is selected as the best pool for the vehicle routing domain. The running time in this pre-processing state is set as use half the time prescribed by CHeSC rules (300 CPU seconds validated trough a benchmark program [28].)

Table 10: Vehicle routing. Computational time in CPU hours required by the heuristic sub-set selection methodology.

Statistical Ranking			*Grouping Heuristics*
Landmarking	Evolvability	ParamILS	
15hrs	16.5 hrs	137 hrs	95 hrs

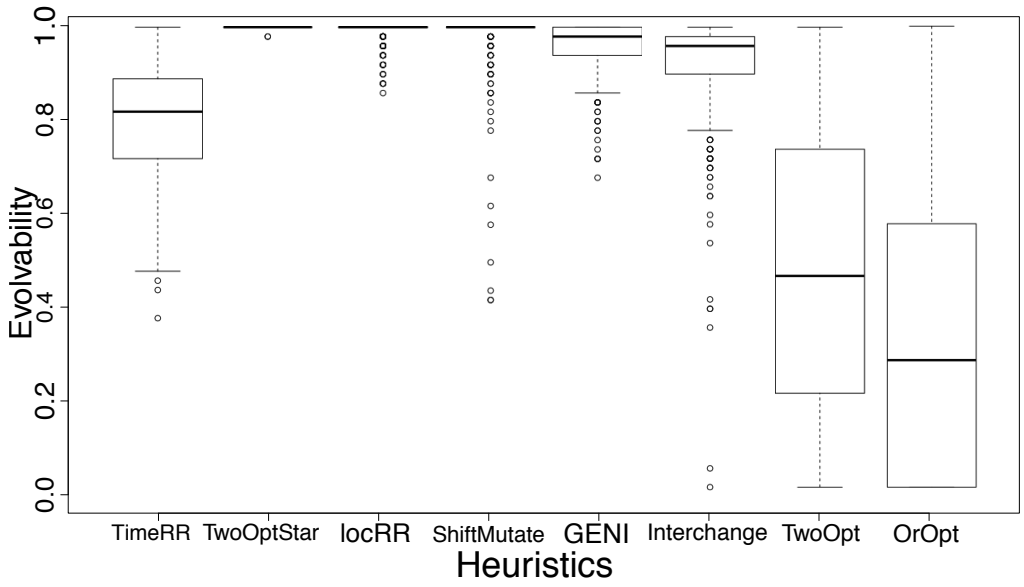

Figure 4: Distribution of evolvability E_a values for each heuristic across 500 experiments. Vehicle routing instance 3.

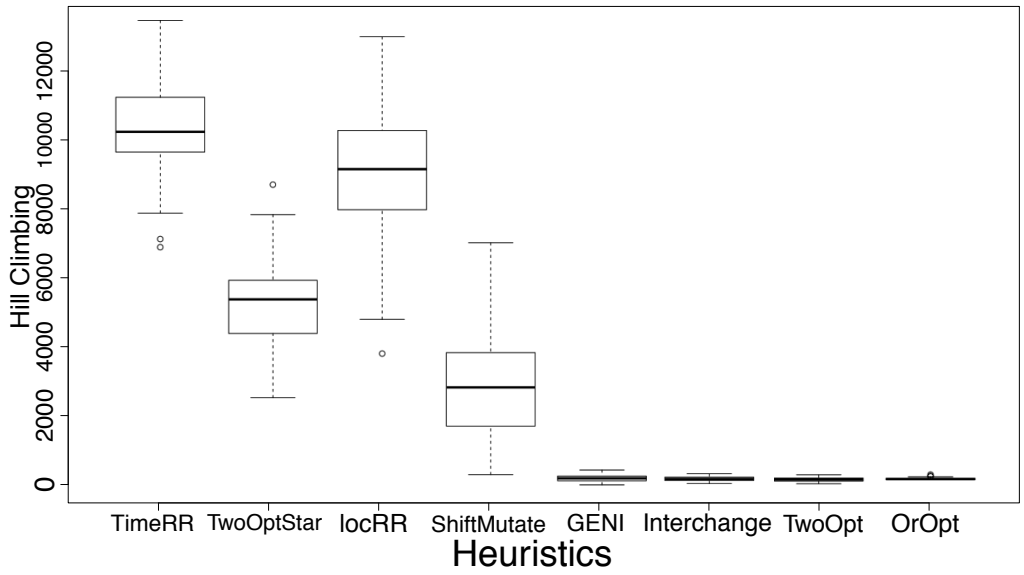

Figure 5: Distribution of hill-climbing landmarking values for each heuristic across 500 experiments. Vehicle routing instance 3.

Table 11: Vehicle routing. Hyper-heuristic performance, as average of 31 independent runs, 300 CPU seconds per run, with heuristic subsets of increasing size. Bold fonts highlight best performance, which mostly occurs with the subset of 4 heuristics.

Instance	2	3	4	5	6	7	8
0	5571.76	5294.29	5249.89	**5233.23**	5276.42	5313.06	5366.26
1	22519.96	21972.70	**21775.03**	21871.82	21959.49	21985.76	21989.71
2	14747.22	14408.75	**14190.83**	14228.91	14385.65	14343.09	14461.58
3	6550.24	5739.74	**5616.24**	5722.01	5939.19	6070.90	6092.82
4	15423.92	15214.17	**14995.28**	15063.98	15099.32	15101.64	15136.26
5	160453.599	160673.41	**160903.68**	160156.86	160119.81	161336.22	160516.81
6	70613.13	70522.17	**70033.93**	70047.44	70124.60	71063.20	70615.11
7	172858.89	170666.18	**170207.01**	170344.21	171124.51	171387.77	171407.28
8	167524.63	167241.77	**165473.22**	165868.30	165598.62	167967.05	169647.40
9	154978.08	154924.18	**155362.17**	154453.02	154580.84	155794.78	155196.08

Table 12: Vehicle routing: Static operator selection probabilities (expressed as percentages) obtained with parameter tuning.

TimeRR	TwoOptStar	locRR	ShiftMutate
35.8%	25.6%	23.2%	14.4%

7.1. Contrasting operator selection strategies

The proposed adaptive hyper-heuristic is compared against their counterparts with both uniformly at random selection of operators, and tuned operator probabilities. Table 12 reports the static operator probabilities (expressed as percentages) obtained using ParamILS (as discussed in section 2.2). The tuning process considered the whole set of 10 instances.

Table 13 compares the three operator selection strategies: random, adaptive and tuned, within the proposed hyper-heuristic using the effective subset of 4 low-level heuristic. Values correspond to averages of 31 runs, where each run lasted 10 minutes according to the benchmark algorithm provided in the competition website [28]. The adaptive strategy consistently produces the best (smallest) fitness values.

Table 13: Vehicle routing: Contrasting operator selection strategies. Values account for average best fitness at the end of the run, which are to be minimised. The Adaptive strategy clearly outperforms its counterparts.

Instance	Random	Tuned	Adaptive
0	5331.78	5295.16	5166.38
1	21708.60	21632.78	21031.87
2	14488.76	13873.78	13325.16
3	5471.77	5452.12	5421.25
4	14336.96	14329.71	14302.21
5	169142.68	161336.12	157824.57
6	77747.04	76052.12	73166.96
7	170575.26	166397.15	162402.47
8	171693.95	159373.13	154662.08
9	152699.63	152587.32	152395.76

Figure 6 shows the the dynamics of operator selection achieved by the adaptive strategy in a selected vehicle routing instance (number 9). In this case, 3 of the 4 heuristics dominate the search process, with TwoOptStar having the highest selection probability.

7.2. Contrasting against the state-of-art

This section contrasts the proposed hyper-heuristic with the effective low-level heuristics subset against both state-of-the-art hyper-heuristics and problem specific approaches.

For the hyper-heuristic comparison, we consider the contestants in the *Cross-Domain Heuristic Search Competition (CHeSC) 2011* [28], and follow the competition rules. For each testing instance (5 out of the 10 available instances are used for test), 31 runs each lasting 600 CPU seconds according to benchmark tool provided, are conducted. Table 14 reports the mean and standard deviation raw values obtained by our Hyper-heuristic in the 5 CHeSC test instances. The algorithms are ranked according to their median performance, and receive points according to a system inspired by the Formula-1 [29]. The top eight preforming algorithms receive 10, 8, 6, 5, 4, 3, 2, and 1 points, respectively. In case of ties, the points of the concerned positions are summed and equally shared. Following this system, our method achieves 25 points (see Table 15(a)) which means an overall rank of 3^{rd} when compared against the CHeSC contestant HyFlex hyper heuristics as reported by [27]. These are encouraging results as our proposed hyper-heuristic was not designed specifically for HyFlex and the CHeSC competition.

For comparing against problem-specific approaches, we consider the best-known solutions for the Solomon and Homberger instances, as reported in the Transportation Optimization Portal - TOP [37], maintained by SINTEF Applied Mathematics. SINTEF is the largest independent research organisation in Scandinavia. Table 15(b) reports comparative results in terms of number of *Vehicles* and *Distance* travelled. Results suggest that HHDMAB is competitive, specially in instances R101 and R201. This evidence supports the idea that algorithms with increased autonomy and generality can be competitive against human designed problem-specific algorithms.

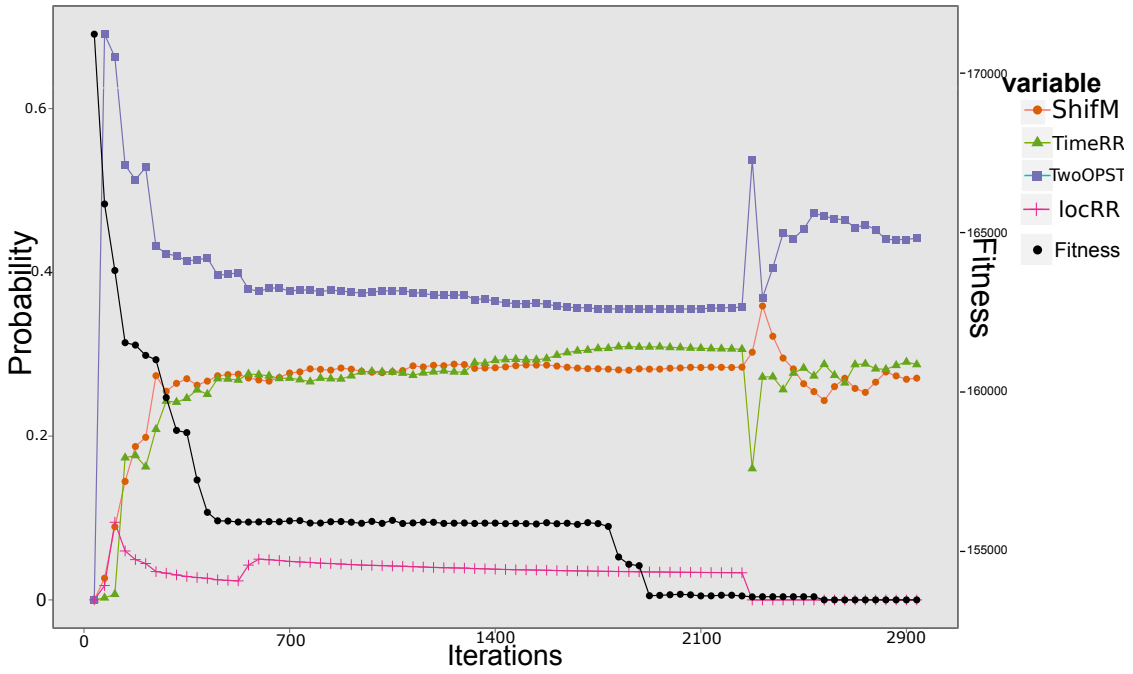

Figure 6: Vehicle routing, instance 9: Operator selection dynamics with the selected sub-set of 4 heuristics. Curves show probabilities over time. The fitness over time is also displayed (black line).

Table 14: Vehicle routing raw results on the CHeSC competition instances. Values account for the average and standard deviation of objective function values (out 31 runs) in form $\bar{x}, \sigma$

	Test 1	Test 2	Test 3	Test 4	Test 5
$\bar{x}$	60773.4	13121.6	148378.4	20654.5	148678.5
σ	58033.1	12277.4	144005.3	20651.1	146053.1
Points	8	6	4	4	3

Table 15: Comparing the proposed approach HHDMAB against CHeSC 2011 hyper-heuristic entries (a), problem specific approaches (b).

(a) Comparison with CHeSC 2011 contestants.

Rank	Algorithm	Score
1	PHUNTER	33
2	HAEA	28
3	**HHDMAB**	**25**
4	KSATS-HH	23
5	ML	22
6	AdapHH	16
6	HAHA	16
8	EPH	12
9	AVEG-Nep	10
10	GISS	6
10	GenHive	6
10	VNS-TW	6
13	ISEA	5
13	XCJ	5
15	SA-ILS	5
16	ACO-HH	2
17	DynILS	1
18	NA-SLS	0
18	SelfSearch	0
18	Ant-Q	0
18	MCHH-S	0

(b) Comparison with best-known solutions as reported in [37]

Instance	**HHDMAB**		Best Know	
RC207	4	1094.14	3	1061.14
R101	19	1655.98	19	1650.80
RC103	12	1395.01	11	1261.67
R201	4	1275.30	4	1252.37
R106	13	1296.86	11	1424.73
C1-10-1	107	43956.7	100	424478.9
RC2-10-1	22	31163.6	20	30278.5
R1-10-1	101	55345.7	100	53501.3
C1-10-8	113	49366.4	92	44092.7
RC1-10-5	98	49154.5	90	45564.8

8. Conclusions

We proposed a generic empirical methodology for selecting an effective heuristic pool in selection hyper-heuristics using non-parametric statistics and fitness landscape probing techniques. From the two probing landscape metrics studied, hill-climbing landmarking proved more informative and computationally efficient. However, operators' evolvability seems to be a reliable metric to quickly distinguish good from bad heuristics in a given domain; and can be used as a pre-processing step to chose a sub-set. The selection methodology proposed outperforms in both efficiency and solution quality, the use of off-the-shelf tuning tools. Our case studies consider improvement heuristics also known as move operators, but in principle the methodology can be applied to selection hyper-heuristics with constructive heuristics, as long as their individual performance can be estimated.

A new variant of iterated local search hyper-heuristics was also proposed, which incorporates dynamic multi-armed bandits. Both the heuristic pool selection method and the hyper-heuristic variant were successfully tested on two complex optimisation problems: course timetabling and vehicle routing, which offers some evidence of generality. Our results on well studied benchmark instances, indicate that the proposed approach outperforms state-of-the-art hyper-heuristics on both domains. Moreover, on the course timetabling domain, it also successfully competes with state-of-the-art problem specific approaches, producing 5 new best-known solutions.

Future work will explore rigorous ways to not only determine the size of the heuristic pool, but importantly, if the pool is effective for the underlying domain, considering for example the complementarity and cooperation among the constituent heuristics. We also foresee the application of the methodology to selection hyper-heuristics with constructive heuristics and new problem domains.

[1] Altenberg, L., 1994. The evolution of evolvability in genetic programming. In: Advances in Genetic Programming. MIT Press, Cambridge, MA, USA, pp. 47–74.
[2] Auer, P., Cesa-Bianchi, N., Fischer, P., 2002. Finite-time analysis of the multiarmed bandit problem. Machine Learning 47 (2-3), 235–256.
[3] Babaei, H., Karimpour, J., Hadidi, A., 2015. A survey of approaches for university course timetabling problem. Computers & Industrial Engineering 86, 43 – 59, applications of Computational Intelligence and Fuzzy Logic to Manufacturing and Service Systems.
[4] Birattari, M., 2009. Tuning Metaheuristics: A Machine Learning Perspective. Springer.
[5] Burke, E., Eckersley, A., McCollum, B., Petrovic, S., Qu, R., 2010. Hybrid variable neighbourhood approaches to university exam timetabling. European Journal of Operational Research 206 (1), 46 – 53.
[6] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E., Qu, R., Dec 2013. Hyper-heuristics: A survey of the state of the art. Journal of the Operational Research Society 64 (12), 1695–1724.
[7] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J., 2010. Handbook of Metaheuristics. Vol. 146 of International Series in Operations Research & Management Science. Springer, Ch. A Classification of Hyper-heuristic Approaches, pp. 449–468, chapter 15.
[8] Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., Qu, R., 2007. A graph-based hyper-heuristic for educational timetabling problems. European Journal of Operational Research 176 (1), 177–192.
[9] Burke, E. K., Qu, R., Soghier, A., 2014. Adaptive selection of heuristics for improving exam timetables. Annals of Operations Research 218 (1), 129–145.
[10] Cambazard, H., Hebrard, E., O¢¢Sullivan, B., Papadopoulos, A., 2012. Local search and constraint programming for the post enrolment-based course timetabling problem. Annals of Operations Research 194, 111–135.
[11] Ceschia, S., Di Gaspero, L., Schaerf, A., 2012. Design, engineering, and experimental analysis of a simulated annealing approach to the post-enrolment course timetabling problem. Computers & Operations Research 39 (7), 1615 – 1624.
[12] Chakhlevitch, K., Cowling, P., 2005. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In: Evolutionary Computation in Combinatorial Optimization. Vol. 3448 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 23–33.
[13] Costa, L. D., Fialho, A., Schoenauer, M., Sebag, M., 2008. Adaptative operator selection with dynamic multi-armed bandits. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008). ACM, pp. 913–920.
[14] Derrac, J., Garcia, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm and Evolutionary Computation 1 (1), 3 – 18.
[15] Eiben, A., Smit, S., mar 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm and Evolutionary Computation 1 (1), 19–31.
[16] Fialho, A., Costa, L., Schoenauer, M., Sebag, M., 2009. Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms. In: Learning and Intelligent Optimization. Vol. 5851 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 176–190.
[17] Fialho, A., Costa, L. D., , Sebag, M., 2010. Analyzing bandit-based adaptive operator selection mechanisms. Ann. Math. Artif. Intell. 60 (1-2), 25–64.
[18] Hoos, H. H., 2012. Automated Algorithm Configuration and Parameter Tuning. In Autonomous Search. Springer Berlin Heidelberg, Ch. 3, pp. 37–71.
[19] Hutter, F., Hoos, H. H., Leyton-Brown, K., Stützle, T., 2009. Paramils: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36 (1), 267–306.
[20] Jat, S. N., Yang, S., 2011. A hybrid genetic algorithm and tabu search approach for post enrolment course timetabling. Journal of Scheduling 14(6), 617–637.
[21] Kumar, S. N., 2012. A Survey on the Vehicle Routing Problem and Its Variants. Intelligent Information Management 04 (03), 66–74.
[22] Lewis, R., 2012. A time-dependent metaheuristic algorithm for post enrolment-based course timetabling. Annals of Operations Research 194, 273–289.
[23] Lewis, R., Paechter, B., McCollum, B., et al., 2007. Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition. Cardiff Business School.
[24] Lewis, R., Thompson, J., 2015. Analysing the effects of solution space connectivity with an effective metaheuristic for the course timetabling problem. European Journal of Operational Research 240 (3), 637 – 648.
[25] Lourenço, H., Martin, O., Stützle, T., 2003. Iterated local search. In: Glover, F., Kochenberger, G., Hillier, F. S. (Eds.), Handbook of Metaheuristics. Vol. 57 of International Series in Operations Research & Management Science. Springer New York, pp. 320–353.
[26] Lu, Z., Hao, J.-K., 2010. Adaptive tabu search for course timetabling. European Journal of Operational Research 200 (1), 235 – 244.
[27] Mascia, F., Stützle, T., 2012. A Non-adaptive Stochastic Local Search Algorithm for the CHeSC 2011 Competition. Springer Berlin Heidelberg, Ch. 1, pp. 101–114.
[28] Ochoa, G., Hyde, M., 2011. The Cross-domain Heuristic Search Challenge (CHeSC 2011). Website, `http://www.asap.cs.nott.ac.uk/chesc2011/`.
[29] Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., Kendall, G., Parkes, A. J., Petrovic, S., Burke, E. K., 2012. Hyflex: a benchmark framework for cross-domain heuristic search. In: Proceedings of the 12th European conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP'12. Vol. 7245 of Lecture Notes in Computer Science. Springer-Verlag, pp. 136–147.
[30] Ochoa, G., Walker, J., Hyde, M., Curtois, T., 2012. Adaptive evolutionary algorithms and extensions to the hyflex hyper-heuristic framework. In: Parallel Problem Solving from Nature - PPSN 2012. Vol. 7492 of Lecture Notes in Computer Science. Springer, Berlin, pp. 418–427.
[31] Pillac, V., Gendreau, M., Guaret, C., Medaglia, A. L., 2013. A review of dynamic vehicle routing problems. European Journal of Operational Research 225 (1), 1 – 11.
[32] Pillay, N., 2013. A survey of school timetabling research. Annals of Operations Research 218 (1), 261–293.
[33] Pillay, N., 2014. A review of hyper-heuristics for educational timetabling. Annals of Operations Research, 1–36.
[34] Qu, R., Burke, E. K., McCollum, B., 2009. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. European Journal of Operational Research 198 (2), 392 – 404.
[35] Qu, R., Burke, E. K., McCollum, B., Merlot, L. T. G., Lee, S. Y., 2008. A survey of search methodologies and automated system development

for examination timetabling. Journal of Scheduling 12 (1), 55–89.
[36] Sabar, N. R., Ayob, M., Kendall, G., Qu, R., Feb 2015. A dynamic multiarmed bandit-gene expression programming hyper-heuristic for combinatorial optimization problems. IEEE Transactions on Cybernetics 45 (2), 217–228.
[37] SINTEF, 2008. Transportation optimization portal - TOP: Vehicle routing problem with time windows. Website, `https://www.sintef.no/vrptw`.
[38] Smith, T., Husbands, P., Layzell, P., O'Shea, M., 2002. Fitness landscapes and evolvability. Evolutionary Computation 10 (1), 1–34.
[39] Smith-Miles, K., Lopes, L., May 2012. Measuring instance difficulty for combinatorial optimization problems. Computers & Operations Research 39 (5), 875–889.
[40] Soria-Alcaraz, J., Ochoa, G., Carpio, J. M., Puga, H., 2014. Evolvability metrics in adaptive operator selection. In: Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014. pp. 1327–1334.
[41] Soria-Alcaraz, J., Ozcan, E., Swan, J., Kendall, G., Carpio, M., 2016. Iterated local search using an add and delete hyper-heuristic for university course timetabling. Applied Soft Computing 40, 581 – 593.
[42] Soria-Alcaraz, J., Terashima-Marin, H., Carpio, M., 2010. Academic timetabling design using hyper-heuristics. Advances in Soft Computing, ITT Springer-Verlag 1, 158–164.
[43] Soria-Alcaraz, J. A., Ochoa, G., Swan, J., Carpio, M., Puga, H., Burke, E. K., 2014. Effective learning hyper-heuristics for the course timetabling problem. European Journal of Operational Research 238 (1), 77 – 86.
[44] Vanneschi, L., Clergue, M., Collard, P., Tomassini, M., Verel, S., 2004. Fitness clouds and problem hardness in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation (GECCO 2004). Vol. 3103 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 690–701.
[45] Walker, J., Ochoa, G., Gendreau, M., Burke, E. K., 2012. Vehicle routing and adaptive iterated local search within the hyflex hyper-heuristic framework. In: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 265–276.

Appendix

This appendix give more details on the formulation and instance data of the two selected test domains.

Course timetabling: is an educational timetabling problem which requires to scheduling of a set of events using limited resources subject to a set of constraints. This study is focusses on the *post-enrollment course timetabling problem* in which the student enrollment is available before the timetabling process, in this variant is necessary to assign a set of courses into timeslots looking for the minimum number on conflicts i.e. Two courses or lectures assigned at the same time to a student. There are two main types of constraints in a timetabling problem: *hard* and $soft$ constraints. The hard constraints have to be satisfied in order to obtain a *feasible* solution, while violations of soft constraints are allowed, since they represent preferences.The main constraints of the post-enrolment course timetabling problem instances of the *International timetabling competition (ITC) 2007* track 2 are as follows:

- A set of n events that are scheduled into 45 time-slots.
- A set of r rooms, each which has a specific seating capacity.
- A set of *room-features* that are satisfied by rooms and required by events.
- A set of s students who attend various different combinations of events.

The hard constraints are:

- No student should be required to attend more that one event at the same time.
- In each case the room should satisfy the class requirements (be big enough for all the attending students and/or required class features).
- Only one event is put into each room in any timeslot.
- Events should only be assigned to timeslots that are pre-defined as available.
- Where specified, events should be scheduled to occur in the correct order.

The soft constraints are:

- Students should not be scheduled to attend an event in the last timeslot of a day.
- Students should not have to attend three or more events in successive timeslots.
- Student should not be required to attend only one event in a particular day.

Vehicle routing: The vehicle routing problem can be described as the task of meeting the demand of all customers, using as few vehicles as possible, and satisfying all constraints, such as vehicle capacity. Furthermore, the variant of the vehicle routing problem which is modelled here is the vehicle routing problem with time windows. This variant includes extra time window constraints, whereby a customer must be served between two time points for a solution to be valid. There is a base location, or depot, from where each vehicle must start and end its route. A route is a series of location visits for a single vehicle. The objective function for this domain balances the dual objectives of minimising the number of vehicles needed and minimising the total distance travelled. It was defined as follows:

$$objectivefunction = c \times numVehicles + distance$$

where c is a constant that we empirically set to 1000 to give higher importance to the number of vehicles in a solution. Instances are taken from two sources: the Solomon data set of 100 customer problems, and the Homberger data set of 1000 customer problems. These instances are available from the Transportation Optimization Portal - TOP [37], maintained by SINTEF Applied Mathematics. SINTEF is the largest independent research organisation in Scandinavia. For both data sets, there are three types of instances. Namely:

R: Random. The customers locations are determined in a uniformly random way.

C: Clustered. The customers locations are grouped in a number of clusters.

CR: Clustered Random. The customers locations are in a mix of random and clustered locations.

A total of 10 instances were chosen, 5 from each data set (Table 16). These are the instances available in the current version of the HyFlex hyper-heuristic framework [29]

Table 16: Used Instances in Vehicle Routing Domain

Instance	name	no-vehicles	vehicle capacity
0	Solomon/RC/RC207	25	1000
1	Solomon/R/R101	25	200
2	Solomon/RC/RC103	25	200
3	Solomon/R/R201	25	1000
4	Solomon/R/R106	25	200
5	Homberger/C/C1-10-1	250	200
6	Homberger/RC/RC2-10-1	250	1000
7	Homberger/R/R1-10-1	250	200
8	Homberger/C/C1-10-8	250	200
9	Homberger/RC/RC1-10-5	250	200