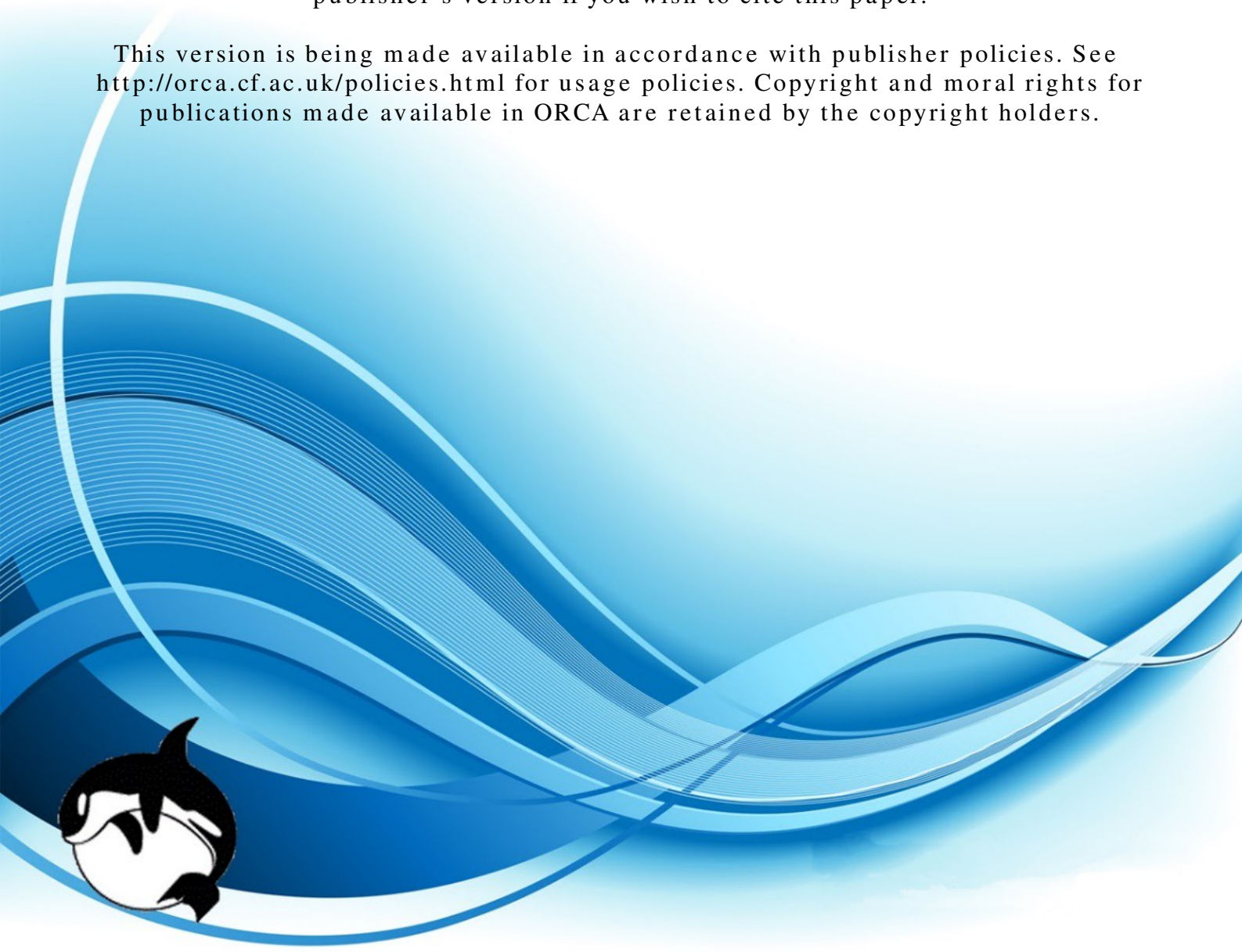


An Iterated Multi-stage Selection Hyper-heuristic

Ahmed Kheiri[1a], Ender Özcan[a]

[a] *University of Nottingham*
School of Computer Science
Jubilee Campus, Wollaton Road, Nottingham, NG8 1BB, UK
{*psxak1,ender.ozcan*}*@nottingham.ac.uk*

Abstract

There is a growing interest towards the design of reusable general purpose search methods that are applicable to different problems instead of tailored solutions to a single particular problem. Hyper-heuristics have emerged as such high level methods that explore the space formed by a set of heuristics (move operators) or heuristic components for solving computationally hard problems. A selection hyper-heuristic mixes and controls a predefined set of low level heuristics with the goal of improving an initially generated solution by choosing and applying an appropriate heuristic to a solution in hand and deciding whether to accept or reject the new solution at each step under an iterative framework. Designing an adaptive control mechanism for the heuristic selection and combining it with a suitable acceptance method is a major challenge, because both components can influence the overall performance of a selection hyper-heuristic. In this study, we describe a novel iterated multi-stage hyper-heuristic approach which cycles through two interacting hyper-heuristics and operates based on the principle that not all low level heuristics for a problem domain would be useful at any point of the search process. The empirical results on a hyper-heuristic benchmark indicate the success of the proposed selection hyper-heuristic across six problem domains beating the state-of-the-art approach.

Keywords: Heuristics, Combinatorial Optimisation, Hyper-heuristic, Meta-heuristic, Hybrid Approach

[1]Ahmed Kheiri (a.kheiri@exeter.ac.uk) is currently an Associate Research Fellow at the College of Engineering, Mathematics and Physical Sciences, University of Exeter.

1. Introduction

Most of the current decision support systems tend to use expert knowledge at their core, and are often custom tailored to a specific application domain. As a result, they cannot be reused for solving a problem from another domain. On the other hand, there has been some significant scientific progress in developing automated general purpose systems that can learn, adapt and improve their behaviour on the fly while solving a given problem. *Hyper-heuristics* are such methodologies which perform search over the space formed by a set of low level heuristics (move operators) which operate on solutions (Burke et al., 2013). An aim in hyper-heuristic research is to increase the level of generality of solution methodologies by *selecting* and/or *generating* heuristics automatically during the search process (Burke et al., 2010b). One of the key features of a hyper-heuristic is that there is a logical separation between the problem domain and the high level hyper-heuristic methodology that operates at the top level. This study focuses on selection hyper-heuristics which combine *heuristic selection* and *move acceptance* methods as their components under a single point iterative search framework which processes a single complete solution at each step (Burke et al., 2003; Özcan et al., 2008; Burke et al., 2012). A selection hyper-heuristic chooses a heuristic from a predefined set of low level heuristics and applies it to a candidate solution. The new solution is then considered and a decision is made whether it will be accepted or not. If accepted, the new solution replaces the current solution and the search continues iteratively. There is a growing number of hyper-heuristics (Chakhlevitch and Cowling, 2008; Burke et al., 2013) and they have been successfully applied to different hard computational problems, including examination timetabling (Pillay and Banzhaf, 2009; Rahman et al., 2014), course timetabling (Burke et al., 2007; Soria-Alcaraz et al., 2014), response time variability (García-Villoria et al., 2011) and water distribution problems (Kheiri et al., 2015).

It has been observed that a selection hyper-heuristic performs differently at different stages of the search process (Kheiri et al., in press; Asta et al., 2013a). Moreover, there is a strong empirical evidence indicating that the choice of heuristic selection and move acceptance combination influences the overall performance of a hyper-heuristic (Özcan et al., 2008; Bilgin et al., 2007). Lehre and Özcan (2013) conducted a theoretical study using a selection hyper-heuristic on a benchmark function showing that an improved runtime complexity can be obtained by mixing simple move acceptance criteria

rather than using each move acceptance method on its own. In that study, random choice is used as a heuristic selection and the algorithmic framework could be viewed as a multi-stage framework in which two hyper-heuristics with different move acceptance is employed. This situation points out the potential of a modified single point-based search framework for selection hyper-heuristics enabling multi-stage operation of multiple hyper-heuristics. The design of multi-stage selection hyper-heuristics then would require another hyper-heuristic level for managing those multiple hyper-heuristics. One of the frameworks proposed in (Özcan et al., 2006) that handles mutational and hill climbing heuristics separately by invoking a mutational heuristic first followed by a hill climber, actually performs a multi-stage search. The higher level in this framework employs two prefixed hyper-heuristics in which each hyper-heuristic controls diversification (exploration of the search space) managing mutational heuristics and intensification (exploitation the accumulated search experience) managing hill climbing heuristics only. In another study, Özcan and Burke (2009) conceptualised a *multilevel search* framework for selection hyper-heuristics with three levels. The proposed method combines multiple hyper-heuristics, each performing search over a set of low level heuristics. Considering the recursive nature of the hyper-heuristic definition of 'heuristics to choose heuristics' by Cowling et al. (2001), multiple hyper-heuristics managing low level hyper-heuristics is also possible requiring another level and so on, causing a hierarchical growth in the hyper-heuristic levels like a tree. The usefulness of such a structure is still under debate. An initial attempt to flatten the hierarchical growth in the hyper-heuristic levels was made by Özcan et al. (2013), assuming a particular situation in which there are multiple move acceptance methods for use. Instead of employing each move acceptance method individually or utilising them all in a multi-stage manner, the authors proposed an approach merging all move acceptance methods into a single method via a group decision making strategy.

To the best of our knowledge, the effectiveness of a general multilevel search framework which combines multiple selection hyper-heuristics based on the ideas in (Özcan and Burke, 2009) has not been investigated further. This work describes an iterated multilevel search framework which allows the use of multiple interacting hyper-heuristics cyclically during the search process. Given that one of the selection hyper-heuristics would be employed at each stage during the search process, we will refer to the overall approach as *multi-stage (selection) hyper-heuristic*. The additional level on top of the multiple selection hyper-heuristics will be referred to as *multi-stage level*.

The proposed multi-stage hyper-heuristic framework is general, reusable and useful in relieving the difficulty of choosing a hyper-heuristic method for solving a problem, by automating the process of selecting a hyper-heuristic at different point of the search process.

There are already some software tools supporting the rapid development of (meta-)hyper-heuristics, such as HyFlex (Hyper-heuristics Flexible framework) (Ochoa et al., 2012a) and Hyperion (Swan et al., 2011). The Java implementation of the HyFlex interface was used in a cross-domain heuristic search challenge, referred to as CHeSC 2011. The results form this competition and six HyFlex problem domain implementations became a benchmark in selection hyper-heuristic studies (see Section 4 for more details). The winning state-of-the-art approach of CHeSC 2011 is an elaborate but complicated algorithm which makes use of machine learning techniques (Misir et al., 2011). In this study, a novel iterated multi-stage selection hyper-heuristic based on the proposed framework is designed, implemented and analysed. The empirical results using the CHeSC 2011 benchmark across six problem domains indicate the success of our hyper-heuristic beating the state-of-the-art approach.

The sections are organised as follows. Section 2 covers some selection hyper-heuristics relevant to this study and introduces the concept of multi-stage selection hyper-heuristics summarising some recent studies. Section 3 describes the components of the proposed multi-stage selection hyper-heuristic framework. An overview of HyFlex and CHeSC 2011 is provided in Section 4. Section 5 presents the empirical results. Finally, Section 6 concludes the study.

2. Related Work

A variety of simple selection hyper-heuristic components were presented in (Cowling et al., 2001), including the following heuristic selection methods. *Simple Random* chooses a random heuristic at each time. *Random Gradient* selects a random heuristic and then applies it to the candidate solution as long as the solution is improved. *Random Permutation* applies low level heuristics in sequence based on a random permutation. *Random Permutation Gradient* combines the Random Permutation and Random Gradient strategies applying a heuristic from the sequence in a random permutation until that heuristic makes no improvement. *Greedy* chooses the best solution after applying all actively used low level heuristics to the current solution. *Choice*

Function is a learning heuristic selection method which gives a score to each low level heuristic based on their utility value. Cowling and Chakhlevitch (2003) suggested the use of a Tabu search based hyper-heuristic that provides a list that disallows heuristics with poor performance.

The idea of applying different hyper-heuristics at different stages has been studied previously. Hyper-heuristics often employ a learning mechanism Burke et al. (2013). Online learning hyper-heuristics receive feedback during the search process potentially influencing their decision in heuristic selection and move acceptance. Offline learning hyper-heuristics generally operate in a train and test fashion in two fixed successive stages. Mostly, latter type of hyper-heuristics are used to generate heuristics Burke et al. (2009). There are studies on offline learning selection hyper-heuristics as well. For instance, Chakhlevitch and Cowling (2005) proposed a two-stage hyper-heuristic using a different hyper-heuristic at each stage. The first stage hyper-heuristic employs a Greedy approach which is used to reduce the number of low level heuristics. In the following stage, a simple random hyper-heuristic accepting non-worsening moves is used. The authors reported that using both greedy and tabu search in combination with the aim of linearly reducing the number of the best performing low level heuristics, is promising.

The studies particularly on iterated multi-stage selection hyper-heuristics which enable the cyclic use of multiple selection hyper-heuristics in a staged manner are limited. Asta and Özcan (2015) used a data science technique to partition the low level heuristics which perform well together under a hyper-heuristic using a certain move acceptance method. Then each partition is associated with a relevant move acceptance method. Hence learning takes place during this phase. In the following phase, the approach employs an iterated multi-stage search with two hyper-heuristics, each combining the simple random heuristic selection and a different move acceptance method with the associated low level heuristics. There is no learning mechanism used during this phase.

Kalender et al. (2013, 2012) applied an iterated online learning multi-stage greedy gradient (GGHH) hyper-heuristic to the curriculum-based university course timetabling and high-school timetabling problems. This approach performs search iteratively going through two successive stages, which contain Greedy and Reinforcement Learning based hyper-heuristics, both using simulated annealing move acceptance. Both hyper-heuristics embed an online learning mechanism. Reinforcement Learning oriented hyper-heuristic uses the cost change as the score for each low level heuristic choosing the one

with the highest score at each step. Since Greedy stage is costly, this stage gets invoked dynamically if all heuristics start generating poor performance.

An iterated multi-stage hyper-heuristic which combines a *Dominance-based heuristic selection* method and a Random Descent hyper-heuristic with Naïve Move Acceptance (DRD) is proposed in (Özcan and Kheiri, 2012). The dominance-based selection method employs a greedy approach aiming to determine an active subset of low level heuristics taking into account the trade-off between the change in the objective value and the number of steps taken to achieve that result. The second stage applies random descent hyper-heuristic using that active subset of low level heuristics from the first stage to improve the quality of a solution in hand. If the second stage hyper-heuristic stagnates, then the first stage restarts with a given probability for detecting a new active subset of low level heuristics. This hyper-heuristic is tested using HyFlex producing the best performance when compared to the 'default' hyper-heuristics provided with HyFlex.

An iterated multi-stage hyper-heuristic known as *Robinhood* hyper-heuristic (RHH) combining three hyper-heuristics is proposed in (Kheiri and Özcan, 2013). The three hyper-heuristics use the same heuristic selection method but they differ in the move acceptance component. The selection heuristic component employs round-robin strategy-based neighbourhood method and applies the mutational and ruin and re-create heuristics on the candidate solution, then crossover heuristics, and then hill climbing heuristics and assigns equal time for each low level heuristic. Three move acceptance criteria (one per each hyper-heuristic) including only improving, improving or equal, and an adaptive acceptance methods are used in this approach. In the adaptive acceptance method, a move that improves the quality of the current solution is always accepted. Deteriorating moves are accepted according to a probability that is adaptively modified at different stages throughout the search. Each hyper-heuristic in RHH is applied for a fixed duration of time and therefore The transition between the three stages is static. This approach outperforms the 'default' hyper-heuristics of HyFlex, and took the fourth place with respect to the twenty approaches from CHeSC 2011.

An iterated multi-stage hyper-heuristic, referred to as *Hyper-heuristic Search Strategies and Timetabling* (HySST) with two stages, each using a different hyper-heuristic is designed to solve a variety of high school timetabling problems from different countries (Kheiri et al., in press). The Simple Random combined with Adaptive Threshold move acceptance and Simple Random combined with Accept All Moves hyper-heuristics are utilised only with

mutational and hill-climbing operators, respectively. The transition between two stages of HySST is deterministic. The following stage starts if the input solution cannot be improved at all in a given stage after a certain duration. This solver is tested on a set of real-world instances and competed at the 3 rounds of the third International Timetabling Competition (ITC 2011)[2]. In round 1, HySST generated the best solutions for three instances, and took the second place in rounds 2 and 3 (Kheiri et al., in press).

Asta et al. (2013a) proposed another iterated multi-stage hyper-heuristic combining two hyper-heuristics, namely; Dominance-based hyper-heuristic and Roulette Wheel selection with Adaptive Threshold move acceptance (DRW). The dominance-based hyper-heuristic reduces the set of low level heuristics using a greedy-like approach in a given stage. This learning approach considers the trade-off between number of steps versus achieved solution quality, discovering the most useful low level heuristics in improvement and at the same time generates their selection probabilities. Then the other hyper-heuristic gets invoked using the reduced set of low level heuristics and associated selection probabilities to improve a given solution at a stage. Similar to HySST, the transition between stages in DRW is deterministic. This multi-stage hyper-heuristic is used to improve pool of solutions as a local search. The approach won the MISTA 2013 challenge[3] at which the purpose was to solve a multi-mode resource-constrained multi-project scheduling problem (MRCMPSP).

In this study, our approach extends the previous multi-stage hyper-heuristics and makes use of the *relay hybridisation* (Misir et al., 2011; Kheiri and Keedwell, 2015) technique which applies a low level heuristic to a solution generated by applying a preceding heuristic. We propose an online learning iterated multi-stage hyper-heuristic which is evaluated on six HyFlex problem domains. Its performance is compared to the other multi-stage hyper-heuristics from the scientific literature and also to the competing hyper-heuristics from CHeSC 2011.

[2]ITC2011 website: http://www.utwente.nl/ctit/hstt/

[3]MISTA2013 challenge website: http://allserv.kahosl.be/mista2013challenge/

3. Methodology

3.1. A Multi-stage Selection Hyper-heuristic Framework

The traditional *single-stage* selection hyper-heuristic framework employs a single heuristic selection and a single move acceptance method. If a different hyper-heuristic component is used during the search process, this constitutes a different *stage* enabling the design of multi-stage hyper-heuristics as illustrated in Figure 1. Allowing the use of multiple hyper-heuristic components interchangeably under a multi-stage framework opens up richer design options, such as the possibility of having several hyper-heuristics controlling different sets of low level heuristics cooperatively. As discussed in Section 2, there is empirical evidence that multi-stage hyper-heuristics has the potential to deliver better performance than the traditional (meta)hyper-heuristics (Kalender et al., 2013, 2012; Özcan and Kheiri, 2012; Asta and Özcan, 2015; Kheiri et al., in press; Asta et al., 2013a). A multi-stage framework requires inclusion of an additional upper level which will be referred to as *multi-stage level* within the selection hyper-heuristic framework as shown in Figure 1. The multi-stage level allows the transition between available hyper-heuristics and their automated control at different points during the search process. Algorithm 1 provides the pseudocode of a multi-stage hyper-heuristic algorithm based on the framework in Figure 1.

3.2. An Iterated Multi-stage Hyper-heuristic Using Dominance and Relay Hybridisation

This study introduces an iterated multi-stage selection hyper-heuristic utilising two interacting hyper-heuristics cyclically and controlled by the multi-stage level as provided in Algorithms 2, 3 and 4. In one stage, a subset of "useful" low level heuristics, each associated with a score is determined by a hyper-heuristic embedding a greedy heuristic selection method (Algorithm 2, lines 15-28). Only that subset of low level heuristics is then used in the other stage (Algorithm 2, lines 9-11) and at each step, a heuristic is selected using a roulette wheel strategy based on those scores. As a move acceptance component of the multi-stage hyper-heuristic, a threshold move acceptance method is used (Asta et al., 2013a; Kheiri et al., in press) in both stages (Algorithm 3, line 13 and Algorithm 4, line 10), however the threshold values are treated in a different way in each stage as explained in this section.

In this study, we assume that the number of low level heuristics for a given problem domain is already provided. We form "new" heuristics by pairing

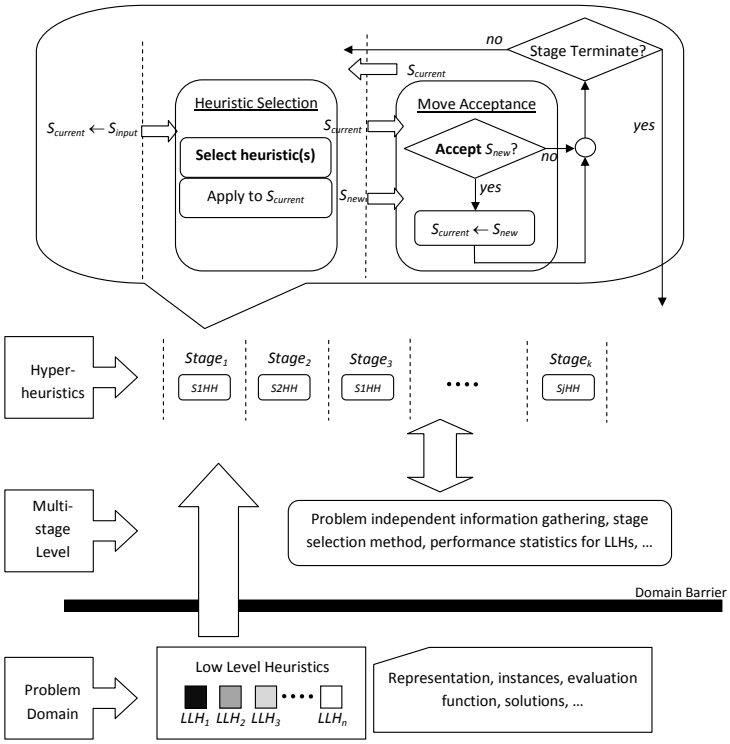


Figure 1: A multi-stage hyper-heuristic framework.

up each low level heuristic and invoking them successively. The technique of combining two heuristics is known as *relay hybridisation* which applies the second low level heuristic to the solution generated by the preceding low level heuristic. The motivation behind relay hybridisation is that although a low level heuristic that does not generate any improvement can still be useful when employed in combination with another low level heuristic. There is indeed empirical evidence that this technique is useful when embedded into a selection hyper-heuristic, considering that relay hybridisation is part of the winning selection hyper-heuristic of CHeSC 2011 (Misir et al., 2011). The proposed selection hyper-heuristic method employing relay hybridisation ignores the nature of the low level heuristics and does not make explicit use of the heuristic type information. Consequently, given n low level heuristics ($\{LLH_1, ..., LLH_n\}$), we end up with (n+n^2) low level heuristics in the overall ($\{LLH_1, ..., LLH_n, LLH_1 + LLH_1, LLH_1 + LLH_2, ..., LLH_n + LLH_n\}$), where $LLH_i + LLH_j$ denotes the combined heuristic of the pair LLH_i and LLH_j.

Algorithm 1: Multi-stage hyper-heuristic framework

```
1  Let HH = {S1HH, S2HH, ..., SjHH} represent set of all hyper-heuristics;
2  Let S_input represent set of input solutions;
3  Let S_output represent set of output solutions;
4  Let s_best represent the best solution;
5  repeat
6      SiHH ← SelectHH(HH);            /* i^th hyper-heuristic is chosen */
7      Update1();  /* set/update relevant parameter/variable values before
       entering into a stage */
8      while notSatisfied(SiHH–TerminationCriteria) do
9          S_output, s_best ← ApplyHH(SiHH, S_input);
10         Update2();      /* set/update relevant parameter/variable values
           during a stage */
11     end
12     Update3();   /* set/update relevant parameter/variable values after
       finishing a stage */
13 until TerminationCriterionSatisfied();
14 return s_best;
```

Iterated Local Search (ILS) is an iterative metaheuristic seeking an improved (optimal) solution through two main steps of perturbation and local search (Lourenço et al., 2010). ILS explicitly balances diversification/exploration (capability of jumping to other promising regions of the search space) and intensification/exploitation (capability of performing a thorough search within a restricted promising region) employing those steps, respectively. The relay hybridisation technique has the potential to combine a mutational and a hill climber as a new low level heuristics and if chosen, such a low level heuristic would yield an ILS-like behaviour in an iteration.

Following the previous work in (Kheiri and Özcan, 2013), any selected low level heuristic is executed for a certain duration, τ (Algorithm 3, lines 9-21 and Algorithm 4, lines 6-12).

Combining multiple hyper-heuristics always requires a decision on when to switch between selection hyper-heuristics. The proposed method handles this decision adaptively. If there is no improvement in consecutive stages while the first hyper-heuristic (denoted as S1HH) is used then the second hyper-heuristic (denoted as S2HH) may kick in with a pre-defined transition probability (P_{S2HH}), otherwise, with a probability of (1-P_{S2HH}), S1HH is re-invoked again. The transition from S2HH to S1HH is not probabilistic, meaning that S1HH is invoked for certain after applying S2HH. This is

Algorithm 2: *MultiStageLevel*

Let LLH_{all} represent set of all $(n + n^2)$ LLHs. Each associated with $score_i$
Let $S_{current}$ represent the candidate (current) solution; $S_{inputstage1}$ the input solution to S1HH; $S_{inputstage2}$ the input solution to S2HH; $S_{bestoverall}$ the best solution obtained so far; $S_{beststage}$ the best solution obtained in the relevant stage
Let P_{S2HH} represent the probability to apply S2HH
Let $f(x)$ represent the objective value of a solution x
$S_{current}, S_{inputstage1}, S_{inputstage2}, S_{bestoverall}, S_{beststage} \leftarrow S_{initial}$
$score_{all} \leftarrow \{LLH_1 = 1, \ldots, LLH_n = 1, LLH_1 + LLH_1 = 0, \ldots, LLH_n + LLH_n = 0\}$
Let $C = \{c_0, c_1, \ldots, c_{(|C|-1)}\}$ be the set of threshold values to be used by the move acceptance; $counter \leftarrow 0$; $timeS_{beststage}Improved \leftarrow$ `getTimeElapsed`()
while `notSatisfied`*(terminationCriteria)* **do**
    while `notSatisfied`*(S1HH–TerminationCriteria)* **do**
        $S_{current}, S_{bestoverall}, S_{beststage} \leftarrow S1HH(LLH_{all}, score_{all}, S_{inputstage1}, S_{bestoverall}, timeS_{beststage}Improved, c_{counter})$
    end
    if $counter = (|C| - 1)$ **then**
        $S_{beststage} \leftarrow S_{current}$; $counter \leftarrow 0$
    end
    if $Random(0, 1) < P_{S2HH}$ **then**
        `// Pre-processing steps of S2HH`
        if $f(S_{beststage}) \geq f(S_{inputstage2})$ **then**
            $S_{inputstage2} \leftarrow S_{beststage}$
            $counter \leftarrow counter + 1$
        end
        else
            $counter \leftarrow 0$; $S_{inputstage2} \leftarrow S_{beststage}$
        end
        while `notSatisfied`*(S2HH–TerminationCriteria)* **do**
            $S_{bestoverall}, S_{beststage}, S_{beststep}, paretoArchive \leftarrow S2HH(LLH_{all}, S_{inputstage2}, S_{bestoverall}, c_{counter})$
            $S_{inputstage2} \leftarrow S_{beststep}$
        end
        `// Post-processing steps of S2HH`
        $score_{all} \leftarrow$ `computeScoresBasedOnDominance`$(paretoArchive)$
    end
    else
        $score_{all} \leftarrow \{1, ..., 1, 0, ..., 0\}$; $\epsilon \leftarrow$ `updateEpsilon`$(c_{counter})$
    end
    $S_{inputstage1} \leftarrow S_{beststage}$
end
return $S_{bestoverall}$

explained in more details in the following subsections.

3.2.1. Stage One Hyper-heuristic (S1HH)

Algorithm 3: $S1HH$

input : $LLH_{all}, score_{all}, S_{inputstage1}, S_{bestoverall}, timeS_{beststage}Improved, c_{counter}$
output: $S_{current}, S_{bestoverall}, S_{beststage}$

1 Let $f(x)$ represent the objective value of x
2 $S_{current} \leftarrow S_{inputstage1}$
3 $hIndex \leftarrow$ `rouletteWheelSelection` $(LLH_{all}, score_{all})$
4 **if** (`getTimeElapsed`() $- timeS_{beststage}Improved$) *exceeds* d **then**
5 $\quad S_{current} \leftarrow S_{beststage}$
6 $\quad \epsilon \leftarrow$ `updateEpsilon`$(c_{counter})$
7 $\quad timeS_{beststage}Improved \leftarrow$ `getTimeElapsed`()
8 **end**
9 **while** `notExceeded`(τ) & `notExceeded`$(timeLimit)$ **do**
10 $\quad S_{new} \leftarrow$ `applyHeuristic`$(LLH_{hIndex}, S_{current})$
11 $\quad$ **if** S_{new} $isBetterThan$ $S_{current}$ OR $f(S_{new})$ $isBetterThan$ $(1+\epsilon)f(S_{beststage})$ **then**
12 $\quad\quad S_{current} \leftarrow S_{new}$
13 $\quad$ **end**
14 $\quad$ **if** $S_{current}$ $isBetterThan$ $S_{beststage}$ **then**
15 $\quad\quad S_{beststage} \leftarrow S_{current}$
16 $\quad\quad timeS_{beststage}Improved \leftarrow$ `getTimeElapsed`()
17 $\quad$ **end**
18 $\quad$ **if** $S_{current}$ $isBetterThan$ $S_{bestoverall}$ **then**
19 $\quad\quad S_{bestoverall} \leftarrow S_{current}$
20 $\quad$ **end**
21 **end**
22 $S_{inputstage1} \leftarrow S_{current}$
23 **return** $S_{current}, S_{bestoverall}, S_{beststage}$

In stage one (Algorithm 3), the roulette wheel selection based hyper-heuristic chooses and applies randomly a low level heuristic based on a score associated with each low level heuristic (Algorithm 3, lines 3 and 10). Assuming that the i^{th} low level heuristic LLH_i has a score of $score_i$, then the probability of that heuristic being selected is $score_i / \sum_{\forall k}(score_k)$. Initially, all single heuristics are assigned a score of 1, while the rest of the paired heuristics are assigned a score of 0. The stage one hyper-heuristic always maintains the best solution found during search process, denoted as $S_{beststage}$ and keeps track of the time since the last improvement. The move acceptance

approach directly accepts improving moves, while non-improving moves are accepted if the objective value of the candidate solution is better than $(1+\epsilon)$ of the objective value of the best solution obtained in the relevant stage. Whenever the best solution during a stage can no longer be improved for a duration of d, ϵ gets updated according to Equation 1.

$$\epsilon = \frac{\lfloor \log(f(S_{beststage})) \rfloor + c_i}{f(S_{beststage})} \tag{1}$$

where $f(S_{beststage})$ is the objective value of the best solution obtained during the stage and c_i is an integer value in $C=\{c_0, ..., c_i, ..., c_{(k-1)}\}$, where $c_{(i-1)} < c_i$ for $0 < i < k$ and $k = |C|$. If $f(S_{beststage})$ is less than 1, ϵ takes a small value ~ 0.

The value of c_i never changes in this stage but it might get updated in stage two as explained in the following section. In the first execution of stage one, c_0 is used by default.

If the overall given time limit ($timeLimit$) is exceeded, or there is no improvement in the quality of the best solution obtained during the stage for a duration of s_1, then the stage one hyper-heuristic terminates.

3.2.2. Stage Two Hyper-heuristic (S2HH)

The aim of this stage (Algorithm 4) is to reduce the set of low level heuristics and adjust their scores according to their "performance" using the idea of the dominance-based heuristic selection (Asta et al., 2013a; Özcan and Kheiri, 2012). A score of 0 indicates that the corresponding heuristic will not be used in the following stage. The reduced set of low level heuristics along with the associated score are fed into the stage one hyper-heuristic.

Firstly, ϵ is set using Equation 1 for once at the start of this stage. Having a sorted *circular* list of values $C=\{c_0, ..., c_i, ..., c_{(k-1)}\}$ enables adaptive control of the level of diversification and gives flexibility of relaxing the threshold further whenever necessary allowing larger worsening moves. Initially, c_i takes the value of c_0. If the best solution obtained after applying stage one does not improve for a stage and stage two hyper-heuristic is applied, the parameter takes the next value on the list, that is, for example, c_1, allowing a larger worsening move to be accepted, and so on. If stage one hyper-heuristic manages to improve the solution and then stage two hyper-heuristic is applied, the parameter is reset to c_0. By default, $S_{beststage}$ is fed as an input to the next stage. There might be a case when even the $(c_{(k-1)})$ value is not sufficient to escape from a local optimum and the current (candidate) solution

Algorithm 4: $S2HH$

input : $LLH_{all}, S_{inputstage2}, S_{bestoverall}, c_{counter}$
output: $S_{bestoverall}, S_{beststage}, S_{beststep}, paretoArchive$

```
1  Let f(x) represent the objective value of x
2  S_beststage ← S_inputstage2
3  ϵ ← updateEpsilon(c_counter)
4  for i = 0; i < (n + n^2); i++ do
5      S_current ← S_inputstage2
6      while notExceeded(τ) & notExceeded(timeLimit) do
7          S_new ← applyHeuristic(LLH_i, S_current)
8          if S_new isBetterThan S_current OR
           f(S_new) isBetterThan (1 + ϵ)f(S_beststage) then
9              S_current ← S_new
10         end
11         S_beststage, S_bestoverall, S_beststep, heur_beststep ← updateBestValues(S_current)
12     end
13     paretoArchive ← update(S_beststep, heur_beststep)
14 end
15 return S_bestoverall, S_beststage, S_beststep, paretoArchive
```

is worse than $S_{beststage}$. If the second stage gets executed at this point of the search process, then the current solution is fed into the next stage as input to allow further diversification. It is possible for a given problem domain that Equation 1 could return a value of 0, then c_i is assigned to one of the values in C at random. After c_i is updated, it does not get changed during the execution of the remaining steps of this stage.

A greedy hyper-heuristic is applied using LLH_{all} (Algorithm 4, lines 4-14) for a fixed number of steps s_2. At each step, all the objective values obtained by applying all the low level heuristics are recorded only if they generate solutions different in quality to the input solution. If all heuristics cannot generate a new solution, then they considered all to have the worst possible objective value. The greedy approach takes the best generated solution obtained at a step and feeds it as an input solution to the next step.

At the end of the stage, the non-dominated solutions each associated with the low level heuristic that generated it are determined from the archive (Algorithm 2, line 27). Then the score of each "non-dominated" low level heuristic is increased by 1. It is potentially possible that a low level heuristic could produce a non-dominated solution more than once and so get a higher score indicating the frequency of such success. In the case of a tie where

multiple low level heuristics produce the same non-dominated solution for a given step, their scores are all incremented.

Due to the costly run time that the second stage hyper-heuristic introduces, taking $s_2 \times (n+n^2)$ steps, a relatively low value for s_2 is preferred. Additionally, we have introduced a probability parameter (P_{S2HH}) (Algorithm 2, line 15) in order to limit the use of this stage often. The stage terminates if s_2 steps are fully executed or overall given time limit ($timeLimit$) is exceeded (Algorithm 2, line 23).

Figure 2 provides an example of how stage two hyper-heuristic executes with $n = 2$, i.e., combining two heuristics $\{LLH_1, LLH_2\}$ via relay hybridisation yielding a set of six low level heuristics,$LLH_{all} = \{LLH_1,\ LLH_2,\ LLH_3{=}LLH_1{+}LLH_1,\ LLH_4{=}LLH_1{+}LLH_2,\ LLH_5{=}LLH_2{+}LLH_1,\ LLH_6{=}LLH_2{+}LLH_2\}$ and where $s_2 = 4$. After running all low level heuristics in a greedy fashion for 4 steps, the number of low level heuristics is automatically reduced to three using the Pareto front which considers the trade-off between number of steps a heuristics executes and improvement in the quality of the resultant solution. Also, the selection probability of each heuristic for the use in the next stage is determined based on the information derived from the Pareto front. The low level heuristics on the Pareto front are $\{LLH_1, LLH_2\}$, $\{LLH_1\}$ and $\{LLH_3\}$. Hence, the scores of the fourth, fifth and sixth low level heuristics are zeros; and scores of LLH_1, LLH_2 and LLH_3 are assigned to 2, 1 and 1, respectively. Given that the probability of a heuristic in S1HH being selected is $score_i / \sum_{\forall k}(score_k)$, therefore, in this example the probability of selecting LLH_1 becomes 50%, while it is 25% for LLH_2 and LLH_3. This means that LLH_4, LLH_5 and LLH_6 are not expected to perform well and hence disabled. LLH_1 is expected to perform better than LLH_2 and LLH_3 and hence its selection probability is higher than LLH_2 and LLH_3.

4. Hyper-heuristics Flexible Framework - HyFlex

HyFlex[4] is an interface which supports the selection hyper-heuristic development. This interface including six problem domains was implemented in Java as version v1.0 for a competition, referred to as Cross-Domain Heuristic Search Challenge, CHeSC 2011[5]. The aim of CHeSC 2011 was to determine the state-of-the-art hyper-heuristic that generalises well across a set

[4] `http://www.hyflex.org/`

[5] CHeSC website: http://www.asap.cs.nott.ac.uk/chesc2011/

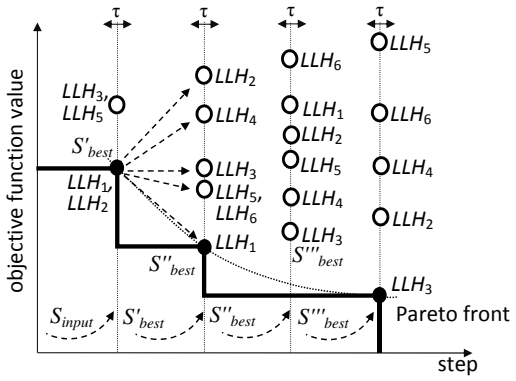


Figure 2: An example of how the stage two hyper-heuristic works. The commas seperate multiple low level heuristics achieving a solution with the same quality. For example, LLH_1 and LLH_2, as well as, LLH_3 and LLH_5 generate solutions with the same quality in the first step. Similarly, in the second step, LLH_5 and LLH_6 generate solutions with the same quality.

of problem instances from six different problem domains. Throughout this paper, "HyFlex" refers to that version. There is a new version of HyFlex which is version v1.1 supporting the batch mode operation of hyper-heuristics (Asta et al., 2013b), which is not within the scope of this study. HyFlex currently provides implementation of six minimisation problem domains: boolean satisfiability (SAT), one-dimensional bin-packing (BP), personnel scheduling (PS), permutation flow-shop (PFS), travelling salesman problem (TSP) and vehicle routing problem (VRP). The software package includes a set of low level heuristics (LLHs) and a number of instances associated with each domain. The code for SAT, BP, PS and PFS was released first along with some public instances, while the code and instances for TSP and VRP were released after the competition.

In HyFlex, the low level heuristics are perturbative heuristics processing and returning complete solutions after their application. Heuristics are categorised as *mutational* (MU) which modifies a solution in some way with no guarantee of improvement, *ruin and re-create* heuristic (RR) which destructs a given complete solution generating a partial solution and then reconstructs a complete solution, *hill climbing* (HC) which performs local search returning a solution which has the same or better quality of the input solution, and *crossover* (XO) which creates a new solution by combining some parts from two given solutions. HyFlex provides functionality for controlling the

intensity of the mutation and ruin and re-create operators, as well as, the *depth of the search* in local search operators. The setting of a parameter is allowed to range from 0.0 to 1.0. Table 1 provides an overview of the low level heuristics from each heuristic category for each HyFlex domain. A low level heuristic from a domain in HyFlex is given a unique ID. For example, low level heuristics from LLH_0 to LLH_5 in the SAT domain are all mutational heuristics.

Table 1: The category of low level heuristics, each indicated by its unique HyFlex ID from each problem domain.

	SAT	BP	PS	PFS	TSP	VRP
MU	0–5	0,3,5	11	0–4	0–4	0,1,7
RR	6	1,2	5,6,7	5,6	5	2,3
HC	7,8	4,6	0–4	7–10	6,7,8	4,8,9
XO	9,10	7	8,9,10	11–14	9–12	5,6

Before CHeSC 2011, the results of eight mock hyper-heuristics from literature over four of the HyFlex problem domains were put on the competition website. The description of the mock hyper-heuristics were not provided on the competition website, but it is reported in (Burke et al., 2010a) that the iterated local search which applies a sequence of heuristics in a predefined order has the best performance.

The ranking method used at CHeSC 2011 is inspired from the Formula 1 points scoring system. The top eight hyper-heuristics are determined after comparison of the median objective values that all hyper-heuristics achieve over 31 trials for each instance. Each algorithm is then awarded a score according to its ranking. The winner receives 10 points, the runner up gets 8 and then 6, 5, 4, 3, 2 and 1, respectively. In the case of a tie for a given instance, the corresponding points are added together and shared equally between each algorithm. In CHeSC 2011, five instances from each domain is used. The winner is the one which scores the maximum points over the thirty instances across all six problem domains.

The results of twenty participants in the competition along with the description of their algorithms were available from the website of the competition. Those results were based on five instances of all HyFlex problem domains. The winner, Mustafa Misir, developed an approach, denoted as AdapHH, which is a solver that applies an adaptive heuristic selection com-

bined with adaptive iteration limited list-based threshold move accepting method (Misir et al., 2011). The second place was taken by a hyper-heuristic based on Variable Neighborhood Search (VNS-TW) which applies shaking heuristics then hill-climber heuristics (Hsiao et al., 2012).

There has been a growing number of studies on selection hyper-heuristics which are evaluated on HyFlex problem domains since CHeSC 2011. We provide a brief overview of some of those single stage generic selection hyper-heuristics in here. An improved choice function hyper-heuristics is proposed in (Drake et al., 2012) showing that this approach is more successful than the traditional choice function heuristic selection. Drake et al. (2015) extended this previous study introducing crossover operators into the set of low level heuristics and a mechanism to control the input those binary operators, which slightly improved the overall performance of the hyper-heuristic. An adaptive iterated local search approach is proposed and applied on HyFlex problem domains in (Burke et al., 2011; Ochoa et al., 2012b). Jackson et al. (2013) evaluated variants of *late acceptance*-based selection hyper-heuristics. The authors point out the best configuration for the late acceptance strategy which accepts a solution if its quality is better than the quality of a solution obtained from a certain number of prior steps. None of those previously proposed selection hyper-heuristics perform better than AdapHH.

5. Experimental Results

We have evaluated the performance of the proposed dominance-based and relay hybridisation multi-stage hyper-heuristic, denoted as MSHH, across six problem domains of HyFlex. During our experimentation, crossover operators are ignored as low level heuristics, considering that the multi-stage hyper-heuristics operate under a single point based search framework. The Mann-Whitney-Wilcoxon test (Fagerland and Sandvik, 2009; Kruskal, 1957) is used as a statistical test for pairwise average performance of two given algorithms. We have used the following notation: Given two algorithms; A versus B, $>$ ($<$) denotes that A (B) is better than B (A) and this performance difference is statistically significant within a confidence interval of 95% and A $\geq$ B (A $\leq$ B) indicates that A (B) performs better on average than B (A) but no statistical significance.

In CHeSC 2011, the competing algorithms are run for 31 trials. Therefore, each experiment is repeated for 31 times unless it is mentioned otherwise. A benchmarking software tool provided at the CHeSC 2011 website is used

to obtain the equivalent time value (*timeLimit*) on the used machines that correspond to 600 nominal seconds according to the competition rule.

We have fixed the parameter values based on our previous work (Özcan and Kheiri, 2012; Kheiri and Özcan, 2013; Kheiri et al., in press; Asta et al., 2013a): $\tau = 15ms$, $d = 9s$, $s_1 = 20s$, $s_2 = 5$, $P_{S2HH} = 0.3$, $C = \{0, 3, 6, 9\}$. The experiments are performed using those settings as "regular" settings on all thirty instances from all domains used at CHeSC 2011. We compare the performance of our approach to each individual hyper-heuristic used in a stage, previously proposed multi-stage hyper-heuristics and competing hyper-heuristics of CHeSC 2011 including the state-of-the art hyper-heuristic which won the competition, respectively.

5.1. Parameter Settings

A set of experiments is performed on four arbitrarily chosen (first) instances of four public problem domains to observe the performance of the proposed algorithm under different parameter settings:

- $\tau = \{10, \mathbf{15}, 20, 30\}$ (in milliseconds)
- $d = \{7, \mathbf{9}, 10, 12\}$ (in seconds)
- $s_1 = \{10, 15, \mathbf{20}, 25\}$ (in seconds)
- $s_2 = \{3, \mathbf{5}, 10, 15\}$ (in steps/iterations)
- $P_{S2HH} = \{0.1, \mathbf{0.3}, 0.6, 0.9, 1.0\}$
- $C = \{\{0\}, \{3\}, \{6\}, \{9\}, \{\mathbf{0,\ 3,\ 6,\ 9}\}\}$

While testing a different setting for a given parameter, the remaining parameters are fixed with the values marked in bold which are our initial settings. MSHH is run with each setting for 10 trials on the selected instance from each public domain. Table 2 summarises the results based on the average performance of MSHH with various parameter settings when a given parameter value deviates from its regular setting. In all cases, MSHH with the "regular" parameter setting wins against another setting, however, mostly, this performance difference is not statistically significant. There are a few cases for which MSHH with a setting other than the proposed one yields a slightly better average performance on the BP and PS instances. For example, $\tau = 10$ performs slightly better than $\tau = 15$ on the BP instance, and

$P_{S2HH} = 0.6$ is a slightly better choice than $P_{S2HH} = 0.3$ for the PS instance. MSHH with the "regular" parameter setting always performs better than another setting on the PFS and SAT instances. The overall performance of a setting across all domains is important in cross domain search. Hence, MSHH with the "regular" parameter setting turns out to be indeed a good choice and so the same setting is used during the remaining experiments.

Recall that each low level heuristic in HyFlex has a parameter (intensity or depth of the search) that can take any value in the range from 0.0 to 1.0. Initially, we assigned the parameter value of 0.0 to each low level heuristic. The relevant parameter setting of a selected low level heuristic gets updated to a random value in case the move does not improve the candidate solution, otherwise the same setting is kept.

Table 2: The average performance comparison of MSHH for different parameter settings over 10 trials. MSHH with "regular" setting of a given parameter is compared to MSHH when that setting is changed to a given setting based on Mann-Whitney-Wilcoxon statistical test for each selected instance from a public domain.

Par.:	τ			d			C			
Dom.	10	20	30	7	10	12	{0}	{3}	{6}	{9}
SAT	$\geq$	$\geq$	$\geq$	$>$	$\geq$	$\geq$	$>$	$>$	$\geq$	$\geq$
BP	$\leq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\leq$	$\geq$	$\geq$
PS	$\geq$	$\geq$	$>$	$\leq$	$\geq$	$\geq$	$\leq$	$\geq$	$\geq$	$\geq$
PFS	$>$	$>$	$>$	$\geq$	$>$	$\geq$	$>$	$>$	$\geq$	$\geq$
wins	3	4	4	3	4	4	3	3	4	4
Par.:	s_1			s_2			P_{S2HH}			
Dom.	10	15	25	3	10	15	0.1	0.6	0.9	1
SAT	$\geq$	$\geq$	$\geq$	$\geq$	$>$	$\geq$	$>$	$\geq$	$\geq$	$\geq$
BP	$\leq$	$>$	$\leq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\leq$	$\geq$
PS	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$\geq$	$>$	$\leq$	$>$	$\geq$
PFS	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$>$	$\geq$	$\geq$
wins	3	4	3	4	4	4	4	3	3	4

5.2. Performance Comparison to the Constituent Hyper-heuristics

We have experimented with the hyper-heuristics used at each stage, denoted as S1HH and S2HH, respectively, run on their own and compare their performances to the performance of the proposed multi-stage hyper-heuristic. Tables 3 presents the results. MSHH obtains the best solution in 31 trials for 27 out of 30 of the CHeSC 2011 instances, which include all instances from

the SAT, BP and TSP domains and exclude one instance from the remaining domains. On average, MSHH still performs better than the constituent hyper-heuristics of S1HH and S2HH run on their own on the 22 instances across all six problem domains. The standard deviation associated with the average objective value from MSHH is the lowest in all cases on the SAT and TSP problem domains.

On average, MSHH outperforms S2HH and this performance is statistically significant for all instances, except for Inst2, Inst3 and Inst4 from the PS domain and Inst3 from the VRP domain. On the SAT and TSP domains, MSHH performs still significantly better than S1HH on all instances. On PS, MSHH is better than S1HH in four instances, but this performance variation is significant for two out of the four instances. MSHH performs slightly better than S1HH on the BP, PFS and VRP domains. However, S1HH performs better than MSHH only on two instances, Inst1 from BP and Inst2 from PFS for which the performance difference is statistically significant.

Our study empirically confirms that combining hyper-heuristics under a multi-stage framework can potentially lead to an improved overall performance.

5.3. Performance Comparison to the Previous Multi-stage Selection Hyper-heuristics

The performance of the proposed dominance-based and relay hybridisation multi-stage hyper-heuristic is compared to the performance of some previously proposed elaborate and successful multi-stage hyper-heuristics which are described in Section 2: greedy, random gradient and simulated annealing hyper-heuristic (also known as greedy-gradient) (GGHH) (Kalender et al., 2012), dominance-based and random descent hyper-heuristic (DRD) (Özcan and Kheiri, 2012), Robinhood selection hyper-heuristic (RHH) (Kheiri and Özcan, 2013), hyper-heuristic search strategies and timetabling approach (HySST) (Kheiri et al., in press), dominance-based and roulette wheel hyper-heuristic (DRW) (Asta et al., 2013a). Table 4 presents the results achieved after the application of all those multi-stage hyper-heuristics to the CHeSC 2011 domains under the same setting.

In the overall, MSHH turns out to be a viable general methodology outperforming the other multi-stage hyper-heuristic approaches in most of the HyFlex problem domains. The MSHH consistently performs the best in SAT, BP and TSP problem domains based on the average and minimum objective values obtained over 31 runs for each instance. Only for Inst1 from BP, DRD

Table 3: The performance comparison of MSHH, S1HH and S2HH based on the average (avg.), associated standard deviation (std.), minimum (min.) of the objective values over 31 trials and the pairwise average performance comparison of MSHH vs S1HH and MSHH vs S2HH based on Mann-Whitney-Wilcoxon for each CHeSC 2011 instance produced by each approach. The hyper-heuristic producing the best value for avr. and min. per each instance are highlighted in bold.

		MSHH					S1HH				S2HH		
Domain	Instance	avg.	std.	median	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.
SAT	Inst1	**0.9**	0.7	1.0	**0.0**	$>$	6.4	4.5	1.0	$>$	15.0	4.6	3.0
	Inst2	**3.1**	3.9	2.0	**1.0**	$>$	21.3	13.3	3.0	$>$	44.9	9.8	18.0
	Inst3	**0.7**	0.5	1.0	**0.0**	$>$	7.1	7.7	**0.0**	$>$	26.3	14.0	1.0
	Inst4	**1.7**	1.0	1.0	**1.0**	$>$	5.7	4.3	**1.0**	$>$	20.0	4.6	12.0
	Inst5	**7.6**	0.9	7.0	**7.0**	$>$	10.4	1.5	**7.0**	$>$	15.4	1.7	13.0
BP	Inst1	0.0163	0.0014	0.0163	**0.0136**	$<$	**0.0159**	0.0010	0.0137	$>$	0.0198	0.0015	0.0160
	Inst2	**0.0037**	0.0015	0.0030	**0.0025**	$>$	0.0061	0.0015	0.0034	$>$	0.0104	0.0021	0.0077
	Inst3	**0.0050**	0.0015	0.0049	**0.0025**	$\geq$	0.0054	0.0012	0.0027	$>$	0.0128	0.0011	0.0104
	Inst4	0.1084	0.0000	0.1084	**0.1083**	$\leq$	**0.1084**	0.0000	0.1083	$>$	0.1084	0.0000	0.1084
	Inst5	**0.0050**	0.0019	0.0044	**0.0032**	$\geq$	0.0055	0.0021	0.0032	$>$	0.0210	0.0015	0.0187
PS	Inst1	**25.5**	4.5	25.0	**16.0**	$>$	28.8	4.7	18.0	$>$	31.6	4.9	22.0
	Inst2	9668.9	217.8	9638.0	**9184.0**	$\leq$	**9645.3**	159.6	9334.0	$\leq$	9645.8	106.7	9391.0
	Inst3	**3283.7**	93.3	3270.0	**3132.0**	$\geq$	3304.8	99.6	3134.0	$\geq$	3309.9	110.2	3172.0
	Inst4	**1786.3**	172.1	1760.0	1545.0	$\geq$	1801.0	142.3	1570.0	$\geq$	1836.0	291.1	**1400.0**
	Inst5	**353.2**	21.2	350.0	**315.0**	$>$	724.4	657.3	320.0	$>$	810.7	621.5	360.0
PFS	Inst1	**6239.8**	14.9	6239.0	**6212.0**	$>$	6287.6	21.9	6249.0	$>$	6353.3	29.8	6301.0
	Inst2	26895.2	55.3	26889.0	**26775.0**	$<$	**26873.2**	30.7	26822.0	$>$	26976.9	54.7	26849.0
	Inst3	**6333.8**	19.0	6325.0	**6303.0**	$>$	6360.5	16.4	6323.0	$>$	6405.5	23.7	6369.0
	Inst4	**11363.8**	32.7	11359.0	**11320.0**	$>$	11429.9	43.8	11357.0	$>$	11529.3	35.9	11436.0
	Inst5	26711.9	47.0	26709.0	26630.0	$\leq$	**26693.1**	40.7	**26608.0**	$>$	26779.1	49.8	26702.0
TSP	Inst1	**48208.1**	31.8	48194.9	**48194.9**	$>$	50032.0	571.1	49263.1	$>$	50326.5	606.6	49221.6
	Inst2	**2.09e^{+7}**	9.05e^{+4}	2.09e^{+7}	**2.07e^{+7}**	$>$	2.14e^{+7}	1.12e^{+5}	2.12e^{+7}	$>$	2.13e^{+7}	1.05e^{+5}	2.11e^{+7}
	Inst3	**6809.1**	7.1	6808.8	**6796.6**	$>$	7012.5	30.4	6964.6	$>$	7040.2	31.3	6988.6
	Inst4	**66840.2**	276.5	66843.6	**66236.8**	$>$	68908.4	382.4	68159.9	$>$	70241.9	704.6	68791.0
	Inst5	**53011.4**	469.7	52910.2	**52341.3**	$>$	54411.1	595.1	53686.0	$>$	55814.8	946.4	53992.4
VRP	Inst1	70998.4	3840.3	70506.5	**63948.2**	$\leq$	**70223.0**	2960.2	64273.2	$>$	84103.9	7225.8	68958.3
	Inst2	**13421.8**	251.6	13359.6	**13303.9**	$\geq$	13658.0	471.4	13319.6	$>$	13695.8	473.9	13320.0
	Inst3	148498.2	1625.8	148436.2	145466.5	$\leq$	**148232.6**	1935.3	145426.5	$\geq$	149553.2	2377.8	**145362.7**
	Inst4	21016.4	488.2	20671.4	**20650.8**	$\leq$	**20991.3**	478.0	20653.5	$>$	21131.9	510.3	20657.5
	Inst5	**148813.7**	1272.5	149193.7	**146334.6**	$\geq$	148999.1	1217.1	146844.9	$>$	150282.6	1616.3	146666.9

performs better in terms of average and minimum objective values. MSHH achieves the best average results on three instances on the PS and PFS problem domains. MSHH performs the best on average only on the Inst1 VRP instance, while RHH and GGHH perform better on three instances and one VRP instance, respectively. This appears to be an indication that application of all low level heuristics and performing local search and accepting solutions which is the best at any given time is potentially a better approach on the VRP domain. DRD performs the worst on the SAT problem domain, but delivers a good average performance on the BP problem domain. GGHH and DRW manage to provide the best average results on a single instance of VRP and PFS, respectively. MSHH is better than HySST on all problem instances across all domains and this performance difference is statistically significant.

5.4. Performance Comparison to the CHeSC 2011 Hyper-heuristics

MSHH and the twenty competing hyper-heuristics from CHeSC 2011 are ranked under the same criteria used at the time of the competition. Table 5 presents the scores for each algorithm based on the Formula 1 scoring system with respect to the median objective values obtained during the 31 trials over all instances across the six domains. Although MSHH delivers a relatively poor "median" performance in the PS and VRP problem domains, the overall results reveal that MSHH is the winner with a total score of 163.60.

Another performance evaluation method was suggested by Di Gaspero and Urli (2012) to illustrate the relative performance variation of each hyper-heuristic in a given bunch. Considering a set of hyper-heuristics, denoted as P and the resultant solutions associated with their objective values obtained from running a hyper-heuristic, denoted as $j \in P$ for 31 trials on a given problem instance i, the median objective value, denoted as $med_j(i)$ is normalised to a value, $Nmed_j(i)$ in [0,1] using Equation 2:

$$Nmed_j(i) = \frac{med_j(i) - min_P(i)}{max_P(i) - min_P(i)}, \tag{2}$$

where $min_P(i)$ and $max_P(i)$ are the minimum and maximum median objective values achieved by running all hyper-heuristics in P on i, respectively. Normalising the median objective values also acts as a unification method for all hyper-heuristics on a given domain as well as all problem domains enabling visualisation of relative performance of different hyper-heuristics on

Table 4: The performance comparison of MSHH, GGHH, DRD, RHH, HySST and DRW multi-stage hyper-heuristics based on the average (avg.), associated standard deviation (std.), minimum (min.) of the objective values over 31 trials and the pairwise average performance comparison of MSHH vs (GGHH, DRD, RHH, HySST and DRW) based on Mann-Whitney-Wilcoxon for each CHeSC 2011 instance produced by each approach. The hyper-heuristic producing the best value for avr. and min. per each instance are highlighted in bold.

		MSHH				GGHH				DRD				RHH				HySST				DRW		
Domain	Instance	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.	vs.	avg.	std.	min.
SAT	Inst1	**0.9**	0.7	**0.0**	>	18.4	4.5	9.0	>	27.7	5.0	16.0	>	6.6	1.2	3.0	>	3.2	1.2	1.0	>	7.4	2.7	2.0
	Inst2	**3.1**	3.9	**1.0**	>	42.0	9.8	17.0	>	58.2	8.3	27.0	>	11.8	1.9	8.0	>	16.0	20.4	2.0	>	15.2	12.2	5.0
	Inst3	**0.7**	0.5	**0.0**	>	22.0	7.8	10.0	>	40.5	6.1	17.0	>	4.3	1.5	1.0	>	4.0	6.8	1.0	>	9.4	7.9	**0.0**
	Inst4	**1.7**	1.0	**1.0**	>	19.5	3.5	10.0	>	29.9	3.7	22.0	>	8.9	2.0	4.0	>	4.2	1.5	**1.0**	>	6.3	2.0	2.0
	Inst5	**7.6**	0.9	**7.0**	>	10.4	1.5	**7.0**	>	18.5	2.7	13.0	>	8.5	0.8	**7.0**	>	9.7	2.5	**7.0**	>	10.5	1.5	**7.0**
BP	Inst1	0.0163	0.0014	0.0136	>	0.0608	0.0038	0.0541	<	**0.0133**	0.0018	**0.0109**	≥	0.0167	0.0016	0.0136	>	0.0659	0.0066	0.0536	>	0.0228	0.0022	0.0190
	Inst2	**0.0037**	0.0015	**0.0025**	>	0.0105	0.0017	0.0073	>	0.0084	0.0014	0.0071	>	0.0071	0.0010	0.0036	>	0.0158	0.0036	0.0116	>	0.0070	0.0012	0.0036
	Inst3	**0.0050**	0.0015	**0.0025**	>	0.0209	0.0018	0.0179	>	0.0135	0.0011	0.0114	>	0.0070	0.0016	0.0046	>	0.0330	0.0043	0.0238	>	0.0071	0.0011	0.0050
	Inst4	**0.1084**	0.0000	**0.1083**	>	0.1132	0.0006	0.1116	≥	0.1084	0.0000	0.1083	>	0.1085	0.0000	0.1084	>	0.1268	0.0011	0.1226	>	0.1084	0.0000	0.1084
	Inst5	**0.0050**	0.0019	**0.0032**	>	0.0379	0.0023	0.0322	>	0.0209	0.0023	0.0166	>	0.0079	0.0020	0.0053	>	0.0632	0.0048	0.0542	>	0.0074	0.0022	0.0034
PS	Inst1	**25.5**	4.5	**16.0**	>	32.5	5.2	23.0	≥	27.0	4.1	20.0	≥	27.4	4.1	20.0	>	49.6	4.1	41.0	>	30.6	4.0	24.0
	Inst2	9668.9	217.8	**9184.0**	≥	9695.8	135.0	9415.0	≥	9684.7	110.3	9433.0	≤	**9644.6**	114.6	9405.0	>	10131.0	140.5	9905.0	≥	9723.3	147.7	9486.0
	Inst3	**3283.7**	93.3	**3132.0**	≥	3303.8	93.3	3182.0	≥	3321.0	112.3	3157.0	≥	3316.9	88.4	3158.0	>	3669.5	130.3	3461.0	≥	3292.8	87.5	3139.0
	Inst4	1786.3	172.1	1545.0	≥	1791.9	217.7	1505.0	≤	1773.5	173.8	1523.0	≤	**1738.0**	99.3	1555.0	>	58241.9	7050.4	47646.0	≤	1754.7	146.2	**1448.0**
	Inst5	**353.2**	21.2	**315.0**	>	696.9	519.6	340.0	≥	395.7	235.3	325.0	>	385.3	30.6	330.0	>	208250.2	10796.2	192036.0	>	769.0	838.7	355.0
PFS	Inst1	**6239.8**	14.9	**6212.0**	>	6302.8	11.3	6275.0	>	6333.1	22.5	6285.0	>	6285.3	8.8	6267.0	>	6341.2	25.5	6300.0	>	6255.8	14.3	6228.0
	Inst2	26895.2	55.3	26775.0	≤	26872.6	32.5	26805.0	>	26963.5	56.9	26860.0	>	26931.3	41.0	26846.0	>	27101.0	17.0	27068.0	<	**26807.6**	36.0	**26735.0**
	Inst3	**6333.8**	19.0	**6303.0**	>	6368.3	3.9	6352.0	>	6395.0	19.4	6359.0	>	6354.4	13.1	6326.0	>	6395.4	17.5	6369.0	>	6349.6	22.7	**6303.0**
	Inst4	**11363.8**	32.7	11320.0	>	11454.4	19.4	11420.0	>	11505.3	27.8	11455.0	>	11455.3	19.5	11415.0	>	11551.1	23.1	11482.0	≥	11380.8	34.6	**11319.0**
	Inst5	26711.9	47.0	26630.0	<	26682.0	27.0	26624.0	>	26756.3	54.2	26639.0	≥	26733.9	35.7	26632.0	>	26882.4	18.2	26841.0	<	**26644.0**	29.2	**26598.0**
TSP	Inst1	**48208.1**	31.8	**48194.9**	>	49189.7	280.2	48658.0	>	49561.2	513.7	48579.8	>	49946.2	354.9	48862.2	>	50633.4	2321.2	49356.2	>	50071.6	579.1	49028.9
	Inst2	**2.09e^{+7}**	9.05e^{+4}	**2.07e^{+7}**	>	2.14e^{+7}	1.60e^{+5}	2.12e^{+7}	>	2.13e^{+7}	1.05e^{+5}	2.11e^{+7}	>	2.32e^{+7}	1.82e^{+6}	2.13e^{+7}	>	2.49e^{+7}	1.20e^{+5}	2.47e^{+7}	>	2.13e^{+7}	8.91e^{+4}	2.11e^{+7}
	Inst3	**6809.1**	7.1	**6796.6**	>	7008.8	22.3	6928.2	>	7052.0	31.9	6993.6	>	7063.1	22.5	7013.8	>	8093.3	88.8	7934.9	>	7023.7	41.6	6963.6
	Inst4	**66840.2**	276.5	**66236.8**	>	71115.5	1245.2	69138.4	>	70161.9	445.4	69397.5	>	72505.3	2540.5	69036.3	>	78243.2	739.8	76803.1	>	68817.0	394.5	68088.7
	Inst5	**53011.4**	469.7	**52341.3**	>	57995.5	1643.8	54137.8	>	55144.2	744.3	53787.7	>	57138.0	2417.3	54477.8	>	60433.8	755.5	59470.7	>	54427.7	621.9	53601.3
VRP	Inst1	**70998.4**	3840.3	**63948.2**	>	81271.4	4660.9	73486.9	≥	71768.6	4168.4	65292.3	≥	71146.5	3673.4	65726.4	>	99928.7	2677.2	95327.9	>	97991.1	3940.5	91978.2
	Inst2	13421.8	251.6	13303.9	<	13418.5	41.3	13319.7	>	14523.1	680.4	13354.4	≤	**13405.4**	183.5	13327.0	>	13832.4	605.9	**12351.7**	>	14015.3	601.4	13367.0
	Inst3	148498.2	1625.8	145466.5	>	171937.4	3951.0	163675.7	>	151904.0	1871.8	148551.8	<	**146174.9**	1817.0	**142480.2**	>	350453.2	6010.8	336025.8	>	307301.2	15683.8	280087.2
	Inst4	21016.4	488.2	**20650.8**	≤	**20669.7**	9.7	20654.6	>	21910.4	564.5	20657.2	≤	20766.5	295.9	20653.8	>	24370.2	903.8	21048.8	>	22072.5	560.5	20827.7
	Inst5	148813.7	1272.5	146334.6	>	153931.3	1621.4	150687.8	>	149886.1	1382.4	147255.8	<	**147234.4**	622.8	**145552.5**	>	208923.3	5196.0	202032.9	>	196210.4	6374.2	183749.7

Table 5: Ranking (performance comparison) of MSHH and the 20 hyper-heuristic approaches competed at CHeSC 2011 across six problem domains based on the Formula 1 scoring system.

Label	SAT	BP	PS	PFS	TSP	VRP	Overall
MSHH	48.00	38.00	6.00	25.00	42.60	4.00	**163.60**
AdapHH	27.58	44.00	8.00	33.00	34.60	14.00	161.18
VNS-TW	27.08	2.00	39.50	30.00	13.60	6.00	118.18
ML	10.00	8.00	31.00	36.50	10.00	22.00	117.50
PHUNTER	7.00	2.00	11.50	6.00	21.60	33.00	81.10
EPH	0.00	6.00	10.50	18.00	30.60	12.00	77.10
HAHA	25.58	0.00	24.50	2.83	0.00	14.00	66.92
NAHH	10.50	16.00	2.00	19.50	9.00	6.00	63.00
ISEA	3.50	25.00	14.50	3.50	7.00	4.00	57.50
KSATS-HH	19.00	7.00	8.50	0.00	0.00	22.00	56.50
HAEA	0.00	1.00	1.00	7.33	8.00	27.00	44.33
GenHive	0.00	10.00	6.50	7.00	2.00	6.00	31.50
ACO-HH	0.00	17.00	0.00	6.33	6.00	1.00	30.33
SA-ILS	0.25	0.00	18.50	0.00	0.00	4.00	22.75
AVEG-Nep	9.50	0.00	0.00	0.00	0.00	9.00	18.50
XCJ	3.50	10.00	0.00	0.00	0.00	5.00	18.50
DynILS	0.00	9.00	0.00	0.00	8.00	0.00	17.00
GISS	0.25	0.00	10.00	0.00	0.00	6.00	16.25
SelfSearch	0.00	0.00	3.00	0.00	2.00	0.00	5.00
MCHH-S	3.25	0.00	0.00	0.00	0.00	0.00	3.25
Ant-Q	0.00	0.00	0.00	0.00	0.00	0.00	0.00

the same scale. Figures 3 and 4 provide the box plots of the normalised median objective values for the MSHH and the competitors' hyper-heuristics for each domain and overall, respectively. It is observed that the MSHH outperforms the other approaches overall and in SAT, PFS, and TSP problem domains, taking the second place in BP problem domain. However, the proposed hyper-heuristic delivers a relatively poor performance on the PS and VRP problem domains.

5.5. An Analysis of the Proposed Hyper-heuristic

We have repeated some experiments in order to track and interpret the behaviour of the proposed multi-stage hyper-heuristic. Each trial is repeated for 10 times during this set of experiments. The *percentage utilisation* is the ratio of the number of improvements that a low level heuristic generates over the best solution found so far to the total number of such improvements. Figure 5 shows the average percentage utilisation of the single and combined low level heuristics while an arbitrarily chosen representative instance from

each problem domain is solved. As one would expect, not all the low level heuristics can generate improvement over the best solution found so far during the search process. For example, in PS, surprisingly, LLH_0 and LLH_1 heuristics which are provided as hill climbers do not yield any improvement neither themselves individually nor in combination with another low level heuristic on the tested instance. On the other hand, LLH_3 and LLH_5 are not able to make any improvement on the best solutions while BP instance is being solved. This is not surprising, though, as those low level heuristics are mutational heuristics.

With the exception on SAT problem domain, most of the improving moves are due to hill climbers rather than mutational heuristics. The use of the combination of a mutational heuristic followed by a hill climbing heuristic, like the basic steps of iterated local search (Burke et al., 2010a), is automatically favoured by our hyper-heuristic in the PFS and TSP domains (Figure 5(d), (e)). Similarly, ruin and re-create followed by a hill climber is another favourite automatically detected pairing in the TSP problem domain. In TSP, LLH_1 does not seem to be that useful at the first glance, but considering the relay hybridisation technique, it seems to serve as a 'good' diversification component, improving the performance of the hill climbing heuristic (LLH_8) employed afterwards. In BP, MSHH favours the pairing of a mutational low level heuristic followed by a ruin and re-create heuristic. The relay hybridisation of low level heuristics seem to be useful, except for the PS and VRP domains, in which it has been observed that no generated heuristic pairs contributes towards the improvement of the best solutions. The proposed hyper-heuristic looses time by testing all pairs of given low level heuristics which could have been used in the search process. This could be one of the reasons why the proposed hyper-heuristic performs relatively poor on those domains.

The behaviour of MSHH considering the average threshold value of the move acceptance method and average objective values of the current solution in time is illustrated in Figure 6 for an arbitrarily selected instance from each problem domain. In some cases, MSHH improves the quality of the initially generated solution at the beginning of the search process rapidly. Then the improvement slows down, but still continues as in the BP problem domain (Figure 6(b)). While MSHH solves a given instance, it enters into what seems to be a "neutral" region getting stuck at a local optimum. Due to the employment of the adaptive move acceptance method, the MSHH managed to escape from the local optimum and further improvements to the candidate

solutions are obtained. MSHH seems to require partial restarts while solving problem instances from the SAT and PFS problem domains more than the others which definitely works and this could be one of the reasons for the success of MSHH on those problem domains.

Figure 7 depicts the average number of low level heuristics including individual and paired low level heuristics versus time, over 10 trials on a selected instance from each problem domain. Interestingly, on average, the number of low level heuristics are reduced to approximately less than 10% of the total in all six problem domains. This reduction occurs due to the employment of S2HH which aims to disable the poor performing heuristics. In PS problem domain, it is observed that all the single low level heuristics are used during the entire search process. The fluctuations in the number of used low level heuristics during the search process are very frequent in all the other problem domains. It has been observed that the number of low level heuristics never decreases to a single low level heuristic at any time in none of the domains. Figure 7 illustrates that different set of low level heuristics are useful at different parts of the overall search process. For example, at the start of the search process, the number of the low level heuristics stays the same for BP, then it starts decreasing towards the midst of the given time.

6. Conclusion

A selection hyper-heuristic is a general-purpose search methodology that mixes and controls a given set of heuristics for solving a computationally hard problem. Such high level methods do not require any modification while being applied to a new/unseen problem domain. Moreover, the component-based design of selection hyper-heuristics enables re-usability of those components as well.

Up to this date, most of the selection hyper-heuristics performing single point based search contains two key components: heuristic selection and move acceptance. This work is one of the initial studies that explicitly addresses whether it is useful to combine multiple hyper-heuristics or not and how. We present a general multi-stage hyper-heuristic framework and describe its main components. The proposed multi-stage hyper-heuristic framework is reusable and useful in relieving the difficulty of choosing a hyper-heuristic method for solving a problem. This framework is used as a basis to implement an iterated multi-stage hyper-heuristic with synergistic components embedding two hyper-heuristics with an adaptive threshold

move acceptance method. One of the hyper-heuristics aims to reduce the number of low level heuristics discovering the "potentially" useful ones at a given stage during the search process and adjust the probability of each low level heuristic being selected in the following stages. The hyper-heuristic extends the low level heuristic set first by creating "new" heuristics through relay hybridisation, and then a dominance based learning strategy is employed reducing the number of heuristics. The strategy captures the trade-off between the extent of improvement that a heuristic can generate and the number of steps it takes to achieve that improvement. Moreover, each chosen low level heuristic in the "reduced" set is associated with an adaptively decided selection probability to be used in the following stages. The second hyper-heuristic mixes the "reduced" set of low level heuristics with the given probabilities during the search process.

The proposed learning multi-stage selection hyper-heuristic with adaptive move acceptance is tested on a benchmark of problem domains. The results confirm its success when compared to each constituent hyper-heuristics, previously proposed other multi-stage hyper-heuristics as well as the state-of-the-art hyper-heuristic which won the CHeSC 2011 competition. Our hyper-heuristic is a relatively simple approach which is easy-to-implement and easy-to-maintain as compared to some previously proposed hyper-heuristics including the previous state-of-the-art hyper-heuristic, yet, it is extremely effective in cross-domain search delivering a superior performance. The success of the proposed multi-stage hyper-heuristic approach based on the proposed "simple" framework across a variety of domains indeed indicates the potential of utilising and mixing the existing or new selection hyper-heuristics.

References

Asta, S., Karapetyan, D., Kheiri, A., Özcan, E., Parkes, A. J., 2013a. Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. In: Kendall, G., Vanden Berghe, G., McCollum, B. (Eds.), Proceedings of the 6th Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA2013). Ghent, Belgium, pp. 836–839.

Asta, S., Özcan, E., 2015. A tensor-based selection hyper-heuristic for cross-domain heuristic search. Information Sciences 299, 412 – 432.

Asta, S., Özcan, E., Parkes, A. J., 2013b. Batched mode hyper-heuristics. In: Nicosia, G., Pardalos, P. (Eds.), Learning and Intelligent Optimization. Vol. 7997 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 404–409.

Bilgin, B., Özcan, E., Korkmaz, E. E., 2007. An experimental study on hyper-heuristics and exam scheduling. In: Burke, E. K., Rudová, H. (Eds.), Practice and Theory of Automated Timetabling VI. Vol. 3867 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 394–412.

Burke, E., Kendall, G., Misir, M., Özcan, E., 2012. Monte Carlo hyper-heuristics for examination timetabling. Annals of Operations Research 196 (1), 73–90.

Burke, E. K., Curtois, T., Hyde, M. R., Kendall, G., Ochoa, G., Petrovic, S., Rodríguez, J. A. V., Gendreau, M., 2010a. Iterated local search vs. hyper-heuristics: towards general-purpose search algorithms. In: IEEE Congress on Evolutionary Computation (CEC '10). pp. 1–8.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R., 2013. Hyper-heuristics: a survey of the state of the art. Journal of the Operational Research Society 64 (12), 1695–1724.

Burke, E. K., Gendreau, M., Ochoa, G., Walker, J. D., 2011. Adaptive iterated local search for cross-domain optimisation. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO '11). ACM, New York, NY, USA, pp. 1987–1994.

Burke, E. K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S., 2003. Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (Eds.), Handbook of Metaheuristics. Kluwer, pp. 457–474.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J. R., 2010b. A classification of hyper-heuristic approaches. In: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics. Vol. 146 of International Series in Operations Research and Management Science. Springer US, pp. 449–468.

Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., Woodward, J. R., 2009. Exploring hyper-heuristic methodologies with genetic

programming. In: Kacprzyk, J., Jain, L. C., Mumford, C. L., Jain, L. C. (Eds.), Computational Intelligence. Vol. 1 of Intelligent Systems Reference Library. Springer Berlin Heidelberg, pp. 177–201.

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.

Chakhlevitch, K., Cowling, P., 2005. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In: Raidl, G., Gottlieb, J. (Eds.), Evolutionary Computation in Combinatorial Optimization. Vol. 3448 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 23–33.

Chakhlevitch, K., Cowling, P., 2008. Hyperheuristics: recent developments. In: Cotta, C., Sevaux, M., Sörensen, K. (Eds.), Adaptive and Multilevel Metaheuristics. Vol. 136 of Studies in Computational Intelligence. Springer Berlin Heidelberg, pp. 3–29.

Cowling, P., Chakhlevitch, K., 2003. Hyperheuristics for managing a large collection of low level heuristics to schedule personnel. In: IEEE Congress on Evolutionary Computation (CEC '03). pp. 1214–1221.

Cowling, P., Kendall, G., Soubeiga, E., 2001. A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (Eds.), Practice and Theory of Automated Timetabling III. Vol. 2079 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 176–190.

Di Gaspero, L., Urli, T., 2012. Evaluation of a family of reinforcement learning cross-domain optimization heuristics. In: Hamadi, Y., Schoenauer, M. (Eds.), Learning and Intelligent Optimization. Vol. 7219 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 384–389.

Drake, J. H., Özcan, E., Burke, E. K., 2012. An improved choice function heuristic selection for cross domain heuristic search. In: Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (Eds.), Parallel Problem Solving From Nature (PPSN XII). Vol. 7492 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 307–316.

Drake, J. H., Özcan, E., Burke, E. K., May 2015. A modified choice function hyper-heuristic controlling unary and binary operators. In: Evolutionary Computation, 2015. CEC '15. IEEE Congress on. p. to appear.

Fagerland, M. W., Sandvik, L., 2009. The Wilcoxon–Mann–Whitney test under scrutiny. Statistics in Medicine 28 (10), 1487–1497.

García-Villoria, A., Salhi, S., Corominas, A., Pastor, R., 2011. Hyper-heuristic approaches for the response time variability problem. European Journal of Operational Research 211 (1), 160–169.

Hsiao, P.-C., Chiang, T.-C., Fu, L.-C., 2012. A VNS-based hyper-heuristic with adaptive computational budget of local search. In: IEEE Congress on Evolutionary Computation (CEC '12). pp. 1–8.

Jackson, W. G., Özcan, E., Drake, J. H., 2013. Late acceptance-based selection hyper-heuristics for cross-domain heuristic search. In: 13th UK Workshop on Computational Intelligence (UKCI2013). IEEE, pp. 228–235.

Kalender, M., Kheiri, A., Özcan, E., Burke, E. K., 2012. A greedy gradient-simulated annealing hyper-heuristic for a curriculum-based course timetabling problem. In: 12th UK Workshop on Computational Intelligence (UKCI2012). IEEE, pp. 1–8.

Kalender, M., Kheiri, A., Özcan, E., Burke, E. K., 2013. A greedy gradient-simulated annealing selection hyper-heuristic. Soft Computing 17 (12), 2279–2292.

Kheiri, A., Keedwell, E., 2015. A sequence-based selection hyper-heuristic utilising a hidden Markov model. In: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference. GECCO '15. ACM, New York, NY, USA, pp. 417–424.

Kheiri, A., Keedwell, E., Gibson, M. J., Savic, D., 2015. Sequence analysis-based hyper-heuristics for water distribution network optimisation. Procedia Engineering 119, 1269–1277, computing and Control for the Water Industry (CCWI2015) Sharing the best practice in water management.

Kheiri, A., Özcan, E., 2013. A hyper-heuristic with a round robin neighbourhood selection. In: Middendorf, M., Blum, C. (Eds.), Evolutionary

Computation in Combinatorial Optimization. Vol. 7832 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 1–12.

Kheiri, A., Özcan, E., Parkes, A. J., in press. A stochastic local search algorithm with adaptive acceptance for high-school timetabling. Annals of Operations Research.

Kruskal, W. H., 1957. Historical notes on the Wilcoxon unpaired two-sample test. Journal of the American Statistical Association 52 (279), 356–360.

Lehre, P. K., Özcan, E., 2013. A runtime analysis of simple hyper-heuristics: to mix or not to mix operators. In: Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms (FOGA XII '13). ACM, New York, NY, USA, pp. 97–104.

Lourenço, H. R., Martin, O. C., Stützle, T., 2010. Iterated local search: framework and applications. In: Gendreau, M., Potvin, J.-Y. (Eds.), Handbook of Metaheuristics. Vol. 146 of International Series in Operations Research and Management Science. Springer US, pp. 363–397.

Misir, M., Verbeeck, K., De Causmaecker, P., Vanden Berghe, G., 2011. A new hyper-heuristic implementation in HyFlex: a study on generality. In: Fowler, J., Kendall, G., McCollum, B. (Eds.), Proceedings of the 5th Multidisciplinary International Scheduling Conference: Theory and Application (MISTA2011). pp. 374–393.

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J. A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A. J., Petrovic, S., Burke, E. K., 2012a. HyFlex: a benchmark framework for cross-domain heuristic search. In: Hao, J.-K., Middendorf, M. (Eds.), Evolutionary Computation in Combinatorial Optimization. Vol. 7245 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 136–147.

Ochoa, G., Walker, J., Hyde, M., Curtois, T., 2012b. Adaptive evolutionary algorithms and extensions to the HyFlex hyper-heuristic framework. In: Coello, C. A. C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (Eds.), Parallel Problem Solving from Nature (PPSN XII). Vol. 7492 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 418–427.

Özcan, E., Bilgin, B., Korkmaz, E. E., 2006. Hill climbers and mutational heuristics in hyperheuristics. In: Runarsson, T. P., Beyer, H.-G., Burke, E., Merelo-Guervós, J. J., Whitley, L. D., Yao, X. (Eds.), Parallel Problem Solving from Nature (PPSN IX). Vol. 4193 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 202–211.

Özcan, E., Bilgin, B., Korkmaz, E. E., 2008. A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis 12 (1), 3–23.

Özcan, E., Burke, E. K., 2009. Multilevel search for choosing hyper-heuristics. In: Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory and Application (MISTA2009). pp. 788–789.

Özcan, E., Kheiri, A., 2012. A hyper-heuristic based on random gradient, greedy and dominance. In: Gelenbe, E., Lent, R., Sakellari, G. (Eds.), Computer and Information Sciences II. Springer London, pp. 557–563.

Özcan, E., Misir, M., Kheiri, A., 2013. Group decision making hyper-heuristics for function optimisation. In: 13th UK Workshop on Computational Intelligence (UKCI2013). IEEE, pp. 327–333.

Pillay, N., Banzhaf, W., 2009. A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. European Journal of Operational Research 197 (2), 482–491.

Rahman, S. A., Bargiela, A., Burke, E. K., Özcan, E., McCollum, B., McMullan, P., 2014. Adaptive linear combination of heuristic orderings in constructing examination timetables. European Journal of Operational Research 232 (2), 287–297.

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.

Swan, J., Özcan, E., Kendall, G., 2011. Hyperion - a recursive hyper-heuristic framework. In: Coello, C. A. C. (Ed.), Learning and Intelligent Optimization. Vol. 6683 of Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 616–630.

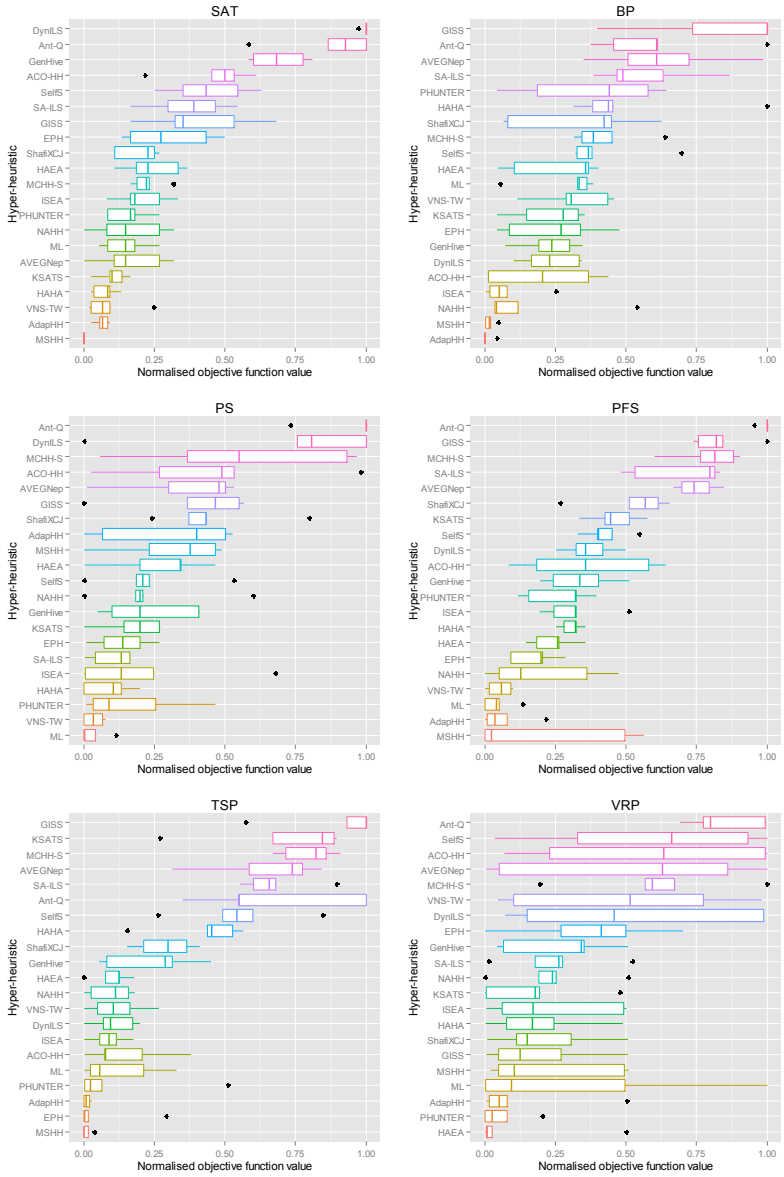


Figure 3: Ranking (performance comparison) of MSHH and CHeSC 2011 hyper-heuristics for each HyFlex problem domain based on the median results converted to the normalised objective values. The dots in the box plots are outliers.

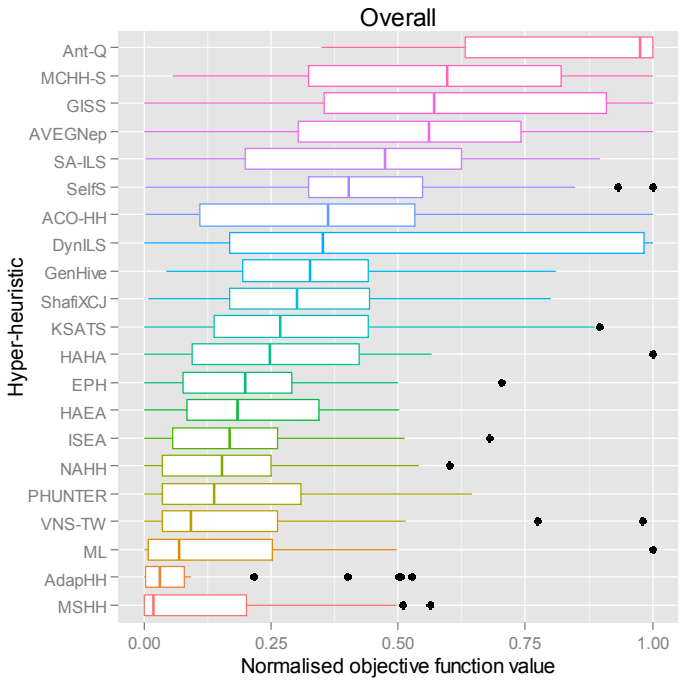
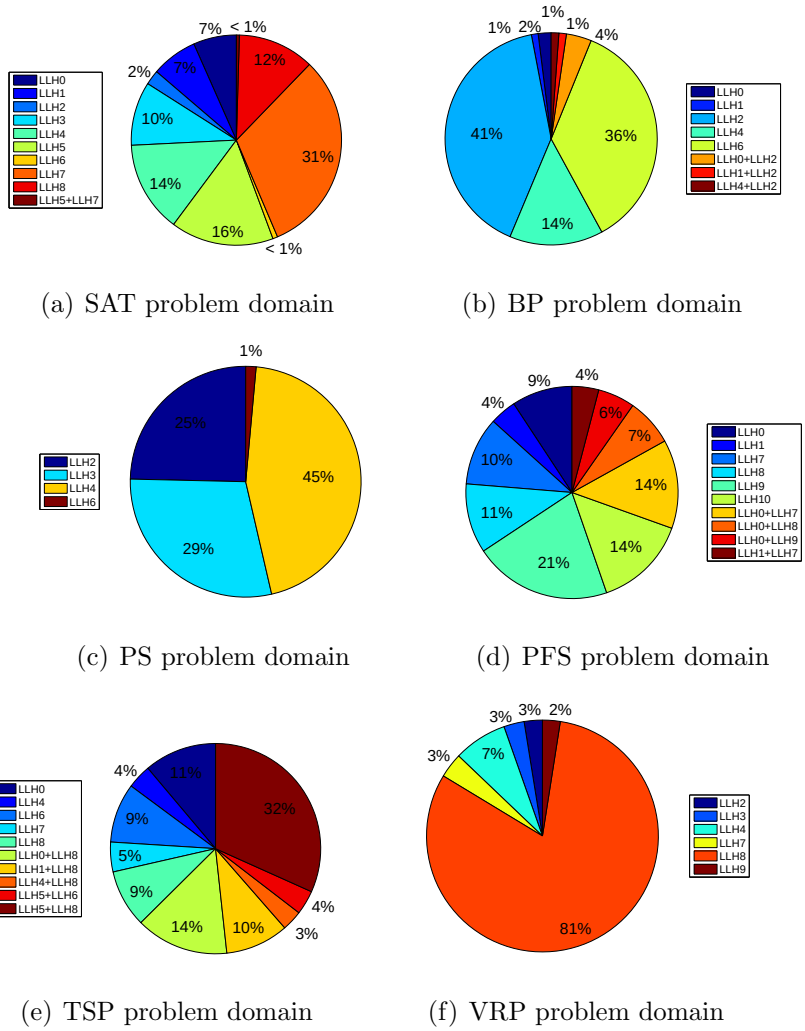


Figure 4: Ranking (performance comparison) of MSHH and CHeSC 2011 hyper-heuristics in overall based on the median results converted to the normalised objective values. The dots in the box plots are outliers.

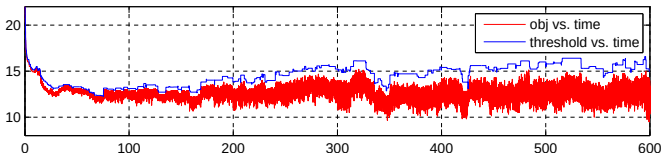


(a) SAT problem domain (b) BP problem domain

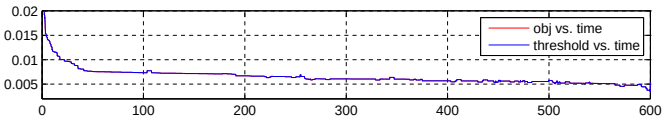
(c) PS problem domain (d) PFS problem domain

(e) TSP problem domain (f) VRP problem domain

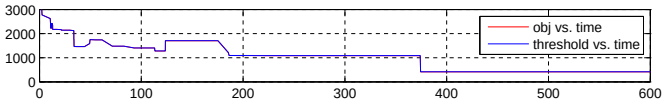
Figure 5: Average percentage utilisation of single/combined low level heuristics over 10 trials while solving a sample instance representing each problem domain.



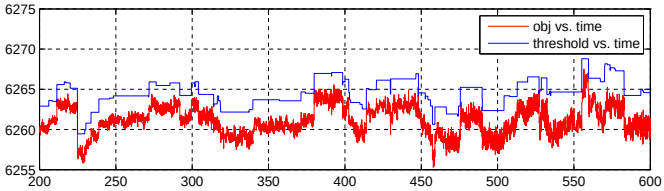
(a) SAT problem domain



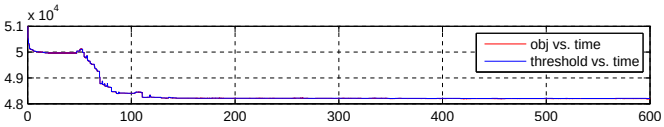
(b) BP problem domain



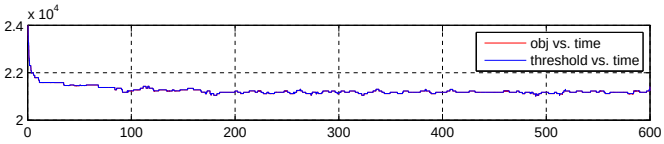
(c) PS problem domain



(d) PFS problem domain

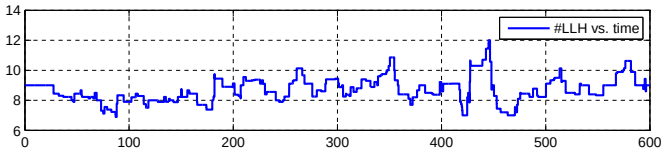


(e) TSP problem domain

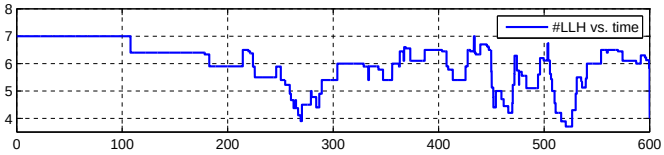


(f) VRP problem domain

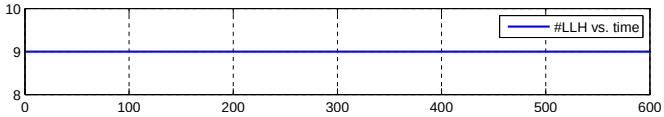
Figure 6: Plots of the average objective and threshold level values over 10 trials versus time while solving a sample instance representing each problem domain.



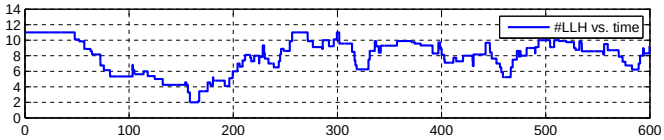
(a) SAT problem domain



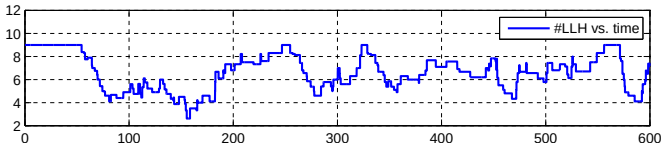
(b) BP problem domain



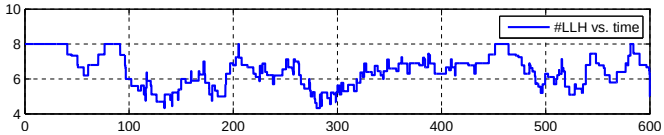
(c) PS problem domain



(d) PFS problem domain



(e) TSP problem domain



(f) VRP problem domain

Figure 7: Plots of the average of changes in the number of single/combined low level heuristics versus time from 10 trials while solving a sample instance representing each problem domain.