MISTA 2011

A new hyper-heuristic implementation in HyFlex: a study on generality

Mustafa Mısır · Katja Verbeeck · Patrick De Causmaecker · Greet Vanden Berghe

Abstract *Reusability* is a desired feature for search and optimisation strategies. Low-level, problem-dependent search mechanisms are far from being used on different problems while keeping them unchanged. Meta-heuristics have been employed for taking the heuristic design process to a higher level and for facilitating reusability. These approaches have been usually conceived as heuristic methods that need to be implemented and adapted with regard to the characteristics of the goal problem. Thus, the resulting algorithms are still problem-dependent and hard to apply to other problems. Hyper-heuristics take the search process into the heuristic level and manage the heuristic set instead of directly solving a problem. During this management process, any problem-dependent data exchange between hyper-heuristics and problems is disallowed. Although any knowledge about the problems is absent for the hyper-heuristics, the generality of hyper-heuristics has not been examined extensively. In a recently proposed high level framework, HyFlex, it is easy to test the generality of hyper-heuristics. HyFlex provides a set of problems with a number of instances as well as a group of low-level heuristics. In this study, the design process of a hyper-heuristic upon HyFlex will be discussed. A performance analysis based on the experimental results will be carried out.

1 Introduction

New problem domains or variants of existing problems have been progressively entering the literature. They draw the attention of researchers to develop effective solution strategies. The performance of the developed algorithms tends to vary over different problems or even over problem instances belonging to a particular problem. In order to alleviate this issue, algorithm selection strategies have been studied. The primary goal

Mustafa Mısır · Katja Verbeeck · Greet Vanden Berghe
KAHO Sint-Lieven, CODeS, Gebroeders Desmetstraat 1, 9000 Gent, Belgium
K.U.Leuven Campus Kortrijk, CODeS, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: {mustafa.misir,katja.verbeeck,greet.vandenberghe}@kahosl.be

Patrick De Causmaecker
K.U.Leuven Campus Kortrijk, CODeS, Etienne Sabbelaan 53, 8500 Kortrijk, Belgium
E-mail: patrick.decausmaecker@kuleuven-kortrijk.be

of these strategies is to determine the best algorithm for the target problem instances. Even if this is a useful approach, it misses the possible improvement opportunities due to varying algorithm performance in the course of the solution process. *Selection hyper-heuristics* follow a deeper selection approach by managing a number of given low-level search strategies during the search relying on their strength and weaknesses [7].

In the literature, the number of studies concerning hyper-heuristics is exponentially growing. In [8], these hyper-heuristics were classified based on the type of the provided feedback mechanisms and the nature of the heuristic search space. Based on their feedback mechanisms, three categories are considered, namely hyper-heuristics with *online learning*, *offline learning* and *no learning*. In online learning, the learning process occurs during the search process. Choice function [16], reinforcement learning [29,31], learning automata [28] are some examples of online learning. Contrarily, offline learning refers to learning before starting the search. In particular, case based reasoning [14] and learning classifier systems [32,26] work in an offline manner. It is also available some hyper-heuristic components without any learning device. An example is the simple random heuristic selection mechanism [16]. Hyper-heuristics are categorised with respect to the nature of the given heuristics as *selection hyper-heuristics* and *generation hyper-heuristics*. The aforementioned studies handled the selection hyper-heuristics that perform over constructive or perturbative heuristics. This type of hyper-heuristics also contain various meta-heuristic components such as tabu search [13], simulated annealing [18,11], genetic algorithms [20] and ant colony optimisation [12]. On the other hand, there exist some hyper-heuristics that aim to generate the low-level heuristics. In [10, 9,19,2,5], genetic programming was utilised to generate low-level heuristics designed for the problem instances.

According to the initial definition of hyper-heuristics, they have been designed to raise the level of generality [7]. Placing a domain barrier preventing any problem-dependent data transition from or to hyper-heuristics is the foremost rule for reaching generality. This basic principle leads hyper-heuristics to focus on managing low-level search strategies instead of directly solving a problem instance. Therefore, it is required to design a hyper-heuristic that has the ability to govern different heuristic sets while profiting maximally from their capabilities. HyFlex is a software framework that empowers hyper-heuristic developers to test their approaches across a range of problems [6]. In the current version, four problem domains, i.e. max SAT, bin packing, permutation flowshop and personnel scheduling, are available. Related to each problem domain, a number of perturbative heuristics that have been implemented from four main heuristic types, namely mutational heuristics, crossover operators, ruin-recreate heuristics and hill climbers. The detailed description of these problems, the characteristics and origins of instances as well as the definition of the heuristics were given in [17,22,21,35]. In addition to these features, HyFlex provides an opportunity to change the effect of mutational heuristics and hill climbers. Moreover, it is possible to keep some of the solutions in memory for further use.

In the present study, a selection hyper-heuristic is implemented on HyFlex and an experimental performance analysis on the available problems is conducted. The proposed hyper-heuristic has been developed as a general adaptive strategy for the four problem domains and the given heuristic sets. It consists of a dynamic heuristic selection mechanism and a move acceptance strategy based on the changing characteristics of the search environment. Experimental results confirm that the developed

Heuristic Type	Max SAT	Bin Packing	Perm. Flowshop	Personnel Scheduling
Mutation	2 (3,4)	2 (0,5)	5 (0,1,2,3,4)	1 (11)
Crossover	2 (7,8)	1 (6)	5 (11,12,13,14,15)	3 (8,9,10)
Ruin-recreate	1 (6)	2 (1,2)	2 (5,6)	3 (5,6,7)
HillClimber	4 (0,1,2,5)	2 (3,4)	4 (7,8,9,10)	5 (0,1,2,3,4)
Total	9	7	16	12

Table 1 The distribution of heuristic types for the given problem domains (heuristic indexes are shown in parentheses)

hyper-heuristic can provide significant performance improvement compared to other hyper-heuristics tested on the problem instances.

In the remainder of this paper, the generality requirements for hyper-heuristics are succinctly discussed in Section 2. The underlying components of the proposed hyper-heuristic is presented in Section 3. Next, the experimental results are discussed in Section 4. In the last section, the paper is concluded.

2 Generality requirements

A generic selection hyper-heuristic should be capable of managing a diverse range of heuristic sets utilised for solving distinct problems. Although the capability of solving as many problems as possible is the primarily mentioned focus, the main concern should be the management of different heuristic sets. This objective implicity embraces the aim of solving various problems anyway. The characteristics of the existing low-level heuristics may require distinct management strategies because each heuristic may have various advantages and disadvantages. These features should be interpreted relying on the dynamic performance of the heuristics and experimental limits such as the given execution time. And the heuristic set as a whole should be used in synergy. For this purpose, a heuristic selection mechanism should consist of particular analysis components to facilitate the adaptivity of the selection process.

2.1 Heuristic set feature

An analysis tool or a learning component should be designed based on a set of characteristics that determine the behavior of the heuristics. The first characteristic is related to the heuristic *set size*. A heuristic set with many heuristics has a higher probability of finding satisfactory solutions regarding a problem instance. Conversely, such a set can be hard to manage due to the availability of many options to select. The quality of heuristics is also critical about the set size. If the heuristics have similar performance, then the set size can make no difference. Since this option is uncommon, it is required to employ effective learning strategies. The second feature is the *speed* of the heuristics. This characteristic may affect the number of decision steps for the selection process. Hence, it is required to interpret this element with the *improvement capabilities* of the heuristics. The heuristic *specialisation* is another factor. A heuristic dedicated to solve a constraint or improve an objective can be considered in this category. In addition to that, heuristics generating only improving or equal quality solutions such as hill climbers or any other strict heuristic behavior come in this category. This feature list can obviously be extended. The main purpose should be the usage of the most relevant

and effective features [1, ch.6] in a collaborative way to predict the future performance of the heuristics.

2.2 Parameter and rule settings

Parameter-free strategies are interesting from a generality purpose perspective. Even though such methodologies are called parameter-free, their behaviour depends on some predetermined values or rules. For instance, simple random is a parameter-free heuristic selection mechanism that gives an equal selection chance to each heuristic. The simple random is parameter-free since its parameters are set from the beginning. Another example is the improving or equal move acceptance mechanism. It accepts only better or equal quality solutions. This method is based on a predetermined rule. That is, the move acceptance method is parameter free, yet not rule-free. All similar methods show that providing a totally independent mechanism seems impossible. Instead, the proposed algorithms should be able to control their parameters or rules according to the search space and the environmental settings with the aim of decreasing the user effect on the algorithm's performance.

3 Hyper-heuristic

A traditional selection hyper-heuristic requires a selection mechanism to determine the best heuristic to apply at each decision step. In addition, it needs a move acceptance strategy to check whether the constructed/visited solution is accepted with regard to its quality and the current state of the search. These consecutive operations are performed until the stopping condition is met. In this study, a new heuristic selection mechanism and a move acceptance strategy including additional components are proposed. In the following paragraphs, these submechanisms are explained in detail.

3.1 Heuristic selection

3.1.1 Dynamic heuristic set

In [27,34], a dynamic heuristic set strategy for the heuristic selection process was studied. The motivation behind this approach is determining elite heuristic subsets during specific iteration intervals. Similar approaches aiming to eliminate heuristics were studied in [13,15,23,24]. The dynamic heuristic set strategy was carried out by eliminating heuristics that are expected to perform worse compared to the rest and keep the best ones determined via some performance criteria. These criteria reflect the performance changes of the heuristics during the search. Performance changes are determined based on the number of new best solutions found, the total fitness improvement and disimprovement per unit execution time. These elements are used according to their importance for the search process. For instance, finding a new best solution is more important than improvement without a new best solution. That is, a heuristic that reaches a new best solution is considered as a higher quality heuristic. This information was gathered during a phase composed of a predetermined number of iterations. In the new selection strategy, i.e. adaptive dynamic heuristic set (ADHS), an updated version of this performance metric is proposed.

A weighted sum of different performance elements was used to determine the quality of different search strategies. In equation (1), the details of the performance measurement for each heuristic is given. In this equation, $C_{p,best}(i)$ denotes the number of new best solutions found. $f_{imp}(i)$ and $f_{wrs}(i)$ show the total improvement and worsening provided. $f_{p,imp}(i)$ and $f_{p,wrs}(i)$ refer to the same measurement be it in a particular phase only. t_{remain} refers to the remaining time to finish the whole search process. $t_{spent}(i)$ and $t_{p,spent}(i)$ are the time spent by heuristic i until now, the same measurement during a phase respectively. And each w_i denotes the weight of a performance element. The weights are set as $\{w_1 >> w_2 >> w_3 >> w_4 >> w_5\}$ to provide a strict priority between the given performance elements. Thus, for instance, if the first performance element of a heuristic has the highest value, than the rest of the performance elements have no effect on the overall p_i of the corresponding heuristic.

$$\begin{aligned} p_i = {} & w_1\Big[\Big(C_{p,best}(i)+1\Big)^2\Big(t_{remain}/t_{p,spent}(i)\Big)\Big] + \\ & w_2\Big(f_{p,imp}(i)/t_{p,spent}(i)\Big) - w_3\Big(f_{p,wrs}(i)/t_{p,spent}(i)\Big) + \\ & w_4\Big(f_{imp}(i)/t_{spent}(i)\Big) - w_5\Big(f_{wrs}(i)/t_{spent}(i)\Big) \end{aligned} \tag{1}$$

In this performance criterion, the first performance element is related to the heuristic's capability of finding a new best solution. In the aforementioned studies, this element was just a counter of new best solutions found by a heuristic. However, it may cause some difficulties especially if some heuristics can find new best solutions only after a long time. For that reason, it can be useful to apply a heuristic that may find more new best solutions during the given execution time. In addition to that, since this strategy works based on the exclusion of heuristics, it can be useful to have plus 1 for the first part. Otherwise, a relatively slow heuristic can stay in the heuristic set regardless of its speed because of finding new best solutions. The second element is used to select improving heuristics and the third element is employed to choose heuristics that deteriorate solutions less. The last two elements have a similar aim, but their values are independent from phases. They are calculated using the values collected until that moment.

The number of phases where a heuristic stays out of an elite heuristic set is denoted by tabu duration. For decreasing user dependency, the tabu duration and the number of iterations for one phase are calculated based on the number of heuristics available in the elite heuristic subset. The tabu duration is set to $d = \sqrt{2n}$ and the phase length (pl) is defined as the product of the tabu duration and a constant value ($ph_{factor} = 500$). These values are recalculated at the end of each phase. Since the calculated p_i values are a combination of different quantities and they are noisy, the performance values are converted into a quality index ($QI \in [1, n]$) value. A heuristic with the lowest p_i gets 1, the others get one unit more based on their order. The average (avg) of these QI values is calculated to determine the heuristics that will be excluded. Tabu heuristics are also attend this calculation with $QI = 1$.

$$avg = \left\lfloor \Big(\sum_{i}^{n} QI_i\Big)/n \right\rfloor \tag{2}$$

Tabu duration adaptation: The tabu duration of a specific heuristic is increased if it is successively excluded. In other words, the tabu duration is specifically determined for

each heuristic using d as the based value. This is required since the proposed exclusion procedure gets a tabu heuristic back whenever its tabu status expires.
Phase length adaptation: The number of iterations per phase is determined as $pl = d * ph_{factor}$. However, this value is updated based on the speed of the non-tabu heuristics as whole for fairness. For that purpose, at the end of each phase, the spent time per move concerning each non-tabu heuristic is calculated and the total of all (t_{subset}) is used to determine the next phase length. In the following equations, the simple formula for calculating the new phase length is given. A predefined constant value is assigned to $ph_{requested}$ that denotes the number of phases requested during the whole search process. For this study, $ph_{requested} = 100$. Using it, a duration of a phase ($ph_{duration}$) is calculated by dividing the total execution time by $ph_{requested}$. The resulting value is divided by t_{subset} to reach the new pl'. If the calculated value is smaller than pl, then it is utilised as the new phase length. This process is repeated at the end of each phase.

$$\begin{aligned} ph_{duration} &= t_{total}/ph_{requested} \\ pl' &= ph_{duration}/t_{subset} \end{aligned} \tag{3}$$

The utilised constant values as well as the provided rules for adaptation purposes regarding the ADHS are set based on the idea of giving chance to the existing heuristics in the heuristic set to show their performance.

3.1.2 Learning automata

A learning automaton is a finite state machine that aims at learning the optimal action out of a set of actions ($A = \{a_1, \ldots, a_n\}$) through interaction with an environment in a trial and error manner [33]. During a learning process, the environmental response ($\beta(t) \in [0, 1]$) to the selected action is used to update a probability vector p consisting of the selection probabilities of the actions. The update operation is carried out using an update scheme (U). The update scheme of the learning automata is based on equations (4) and (5). Equation (4) refers the update operation for the applied action and Equation (5) is used to update the probabilities of the rest of the actions.

$$\begin{aligned} p_i(t+1) = p_i(t) \; & +\lambda_1 \; \beta(t)(1 - p_i(t)) \\ & -\lambda_2(1 - \beta(t))p_i(t) \\ & \text{if } a_i \text{ is the action taken at time step } t \end{aligned} \tag{4}$$

$$\begin{aligned} p_j(t+1) = p_j(t) \; & -\lambda_1 \; \beta(t)p_j(t) \\ & +\lambda_2(1 - \beta(t))[(r-1)^{-1} - p_j(t)] \\ & \text{if } a_j \neq a_i \end{aligned} \tag{5}$$

In [28], a learning automaton with a linear update scheme, *linear reward-inaction*, was employed as a heuristic selection mechanism. During the learning process, actions were considered as low-level heuristics. Heuristics that find new best solutions were rewarded. The changing heuristic probabilities were used like roulette wheel selection. In this study, *linear reward-punishment* scheme is used to update heuristic

probabilities with respect to finding a new best solution, improving the current solution, worsening the current solution and finding a solution with equal quality as the current solution. Related learning rates were set in a decreasing manner in the given order as $\{\lambda_1 = 0.1, \lambda_1 = 0.01, \lambda_2 = 0.002, \lambda_2 = 0.001\}$ without extensive tuning. The first two cases refer to the learning rates that are used for rewarding and the last two options show the learning rates for punishment. Differently from the above mentioned application of the learning automaton, it is just used to keep track of performance changes during different phases of the dynamic heuristic set process. In the beginning of each phase, the learning probabilities are reset. In addition, different learning rates are used for different heuristics. The underlying idea behind this approach is to project the speed of heuristics to the probability updates because it would be unfair to use the same reward/punishment for a very slow and a fast heuristic [4]. For instance, HyFlex contains local search heuristics that only return improving or equal solutions. They are generally expected to be more time consuming compared to other heuristic types. For that reason, the performance differences among heuristics should be interpreted based on their speed. The learning rate of the heuristics are determined using Equation (6). In this equation, C_{max} refers to the number of moves spent per unit time by the fastest heuristic. $C_{i,move}$ denotes the same value for heuristic i. The resulting value (λ_{mult}) is used as a multiplier for the initial learning rate. If this multiplier makes the related learning rate increase more than its predetermined limit, then the learning rate is set to a predetermined value.

$$\lambda_{mult} = C_{max}/C_{i,move} \tag{6}$$

3.1.3 Relay hybridisation

In [30], heuristics were divided into two types: mutational heuristics and hill climbers. They were used within four different frameworks. One of the frameworks, F_C, offers selecting a mutational heuristic first and applying a pre-determined hill climber to the visited solution by the selected mutational heuristic. The experimental results showed that F_C is a very effective strategy to use. In this study, we proposed a relay hybridisation approach to determine a pair of heuristics that can be used consecutively, like F_C, to find superior results. For that purpose, a list of the best heuristics, when applied just after a specific heuristic, is determined. This list is updated in *First-In-Last-Out* manner, by adding a new heuristic to the end of the list and removing the first one to keep the size of the heuristic set fixed if the list is full. After applying a relatively worse performing heuristic based on the provided performance metric, a heuristic is randomly chosen from its list and applied to the solution generated by this heuristic. Since the heuristic list can include different heuristics more than once, a random selection strategy chooses a heuristic among weighted heuristics. For instance, in a list size of 10, a specific heuristic may be present 5 times while the rest of the list may be occupied by the other heuristics. In such a list, the probability of selecting the 5-times occuring heuristic is 50%. That is, even if the selection mechanism is random, the end product is a weighted selection mechanism like roulette wheel.

This strategy is applied based on the value of (C_{phase}/pl). C_{phase} denotes the number of iterations passed within a phase. A random probabilistic value $p \in [0:1]$ is generated and the relay hybridisation is applied if $p \leq (C_{phase}/pl)$. The pseudocode of the method is depicted in Algorithm 1.

Algorithm 1: Relay hybridisation

Input: $l_{size} > 0 \wedge p, p' \in [0:1]$
if $p \leq (C_{phase}/pl)$ **then**
1 | select LLH_{tabu} and apply to $S \rightarrow S'$;
2 | **if** $size(l_i) > 0$ *and* $p' <= 0.5$ **then**
3 | | select a LLH from l_i and apply to $S' \rightarrow S''$;
4 | **else**
5 | | select a $LLH_{nonTabu}$ and apply to $S' \rightarrow S''$;
| **end**
end

In Figure 1, an example of the relay hybridisation is shown. In this example, applying heuristic pairs involving $LLH_2 \rightsquigarrow LLH_5$ and $LLH_2 \rightsquigarrow LLH_3$ found new best solutions before. Whenever LLH_2 is called and if the relay hybridisation is decided to be used, then it is determined whether a consecutive heuristic is randomly chosen from the heuristics that are currently available in the heuristic subset or from its pair list. The selected heuristic is consecutively applied to the solution at hand. If the resulting solution is a new best solution, then the selected heuristic is added to the end of the list.

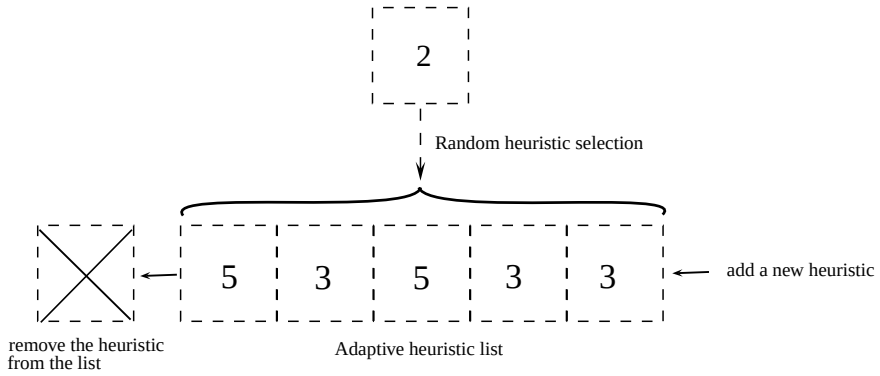

Fig. 1 Relay hybridisation

For some problem domains with some heuristic sets, it may not be possible to find effective heuristic pairs. Therefore, a similar tabu strategy that is used for the heuristic exclusion process is employed. If no new best solution is found by the relay hybridisation after a phase, then this feature is disabled for one phase. Furthermore, if this consecutively happens, then the tabu duration is increased by 1. This value stays the same whenever the tabu duration reaches its upper bound. If it can find a new best solution again, then the tabu duration is set to its initial value, namely 1.

Heuristic adaptation: HyFlex provides an opportunity to modify some of the heuristics in an informed manner. It is possible to increase or decrease the perturbation effect of a mutational heuristic. In addition to that, it is also allowed to change the depth of the search for local search heuristics. In the proposed approach, the deterioration

effects of the mutational heuristics and the search depth of the hill climbers was updated only for the relay hybridisation. If a hill climber can find a new best solution more than fives times, then its search depth is set to 1.0. Otherwise, its value is set to $(0.1 + (ph_{passed}/pl)0.4)$. ph_{passed} is the number of iterations passed during the current phase. The second part is used also for the mutation operators.

3.2 Move acceptance

Move acceptance mechanisms have more influence on the performance of a hyper-heuristic [30]. They determine the way to traverse the search space. One of the main concerns of the move acceptance mechanisms is the acceptability of worsening solutions. Accepting a worsening solution is a widely accepted strategy to prevent from getting stuck around a solution. In [28], a move acceptance strategy was proposed, i.e. iteration limited threshold accepting (ILTA) which accepts worsening solutions in a controlled manner. ILTA immediately accepts improving or equal solutions. If the hyper-heuristic cannot find new best solutions during a pre-determined number of iterations, then, a worsening solution is accepted. This operation is decided based on the value of the best solution found and a constant value determining a range. In [27], an adaptive version of ILTA (AILTA) was proposed. In this move acceptance, a second and higher range value is determined. If the hyper-heuristic cannot find a new best solution using a given range value, then this range value is increased to enable accepting much worse solutions. This is required to get rid of local optima.

Algorithm 2: AILLA Move Acceptance

Input: $i = 1 \wedge K \geq k \geq 0 \wedge l > 0$
for *i=0* **to** *l-1* **do** $best_{list}(i) = f(S_{initial})$
if $adapt\_iterations \geq K$ **then**
  if $i < l - 1$ **then**
    $i++$
  end
end
if $f(S') < f(S)$ **then**
  $S \leftarrow S'$
  $w\_iterations = 0$
  if $f(S') < f(S_b)$ **then**
    $S_b \leftarrow S'$
    $i = 1$
    $adapt\_iterations = 0$
    $best_{list}.remove(last)$
    $best_{list}.add(0, f(S_b))$
  end
else if $f(S') = f(S)$ **then**
  $S \leftarrow S'$
else
  $w\_iterations++$
  $adapt\_iterations++$
  if $w\_iterations \geq k$ *and* $f(S') \leq best_{list}(i)$ **then**
    $S \leftarrow S'$ and $w\_iterations = 0$
  end
end

In the proposed hyper-heuristic, the underlying idea behind list-based threshold accepting [25] was employed to decrease the parameter dependency of AILTA. The new move acceptance strategy, i.e. adaptive iteration limited list-based threshold accepting (AILLA), accepts worsening solutions using the fitness values of the previously visited best solutions. The best fitness visited previously is used as the first threshold value. If it is not good enough to find a new best solution, then a higher fitness from the list ($best_{list}$ with l size) is used to decide about the acceptability of the worsening solutions. The pseudocode of AILLA is presented in Algorithm 2. In addition, the iteration limit (k) is updated whenever a new best solution is found. In Equation (7), the details of this update process is shown. In this equation, $iter_{elapsed}$ is the number of iterations elapsed and t_{exec} refers to the total execution time.

$$k = \begin{cases} ((l-1) \times k + iter_{elapsed})/l, & \text{if } cw = 0 \\ ((l-1) \times k + \sum_{i=0}^{cw} k \times 0.5^i \times tf)/l, & \text{otherwise} \end{cases} \quad (7)$$
$$tf = (t_{exec} - t_{elapsed})/t_{exec}$$
$$cw = \lfloor iter_{elapsed}/k \rfloor$$

4 Experiments

Experiments were carried out on the four problems within HyFlex, namely max SAT [21], bin packing [22], permutation flow shop [35] and personnel scheduling [17] problems. HyFlex provides 10 instances for each problem. Each instance was tested for 10 runs. Each run is executed for 10 minutes using Pentium Core 2 Duo 3 GHz PCs with 3.23 GB memory. In addition to the developed hyper-heuristic, three additional hyper-heuristics were tested. They are the combinations of simple random (SR) selection with AILLA, simulated annealing (SA) [3] and improving or equal (IE) move acceptance mechanisms.

4.1 Computational Results

For the bin packing problem, it is possible to determine an effective hyper-heuristic approach. The experimental results are shown in Table 2. Based on the Wilcoxon test, ADHS-AILLA performs significantly better than the other hyper-heuristics. When the heuristic selection mechanism ADHS is changed to SR, then a significant performance decrease comes out. This shows that for the bin packing problem with the given heuristic set, a learning or adaptation strategy concerning the selection procedure is required. On the other hand, there is no a statistically significant difference between SR-AILLA and SR-IE. However, they both generate superior results compared to SR-SA. This performance difference reveals the necessity of a more selective acceptance strategy regarding accepting worsening solutions. In other words, immediate acceptance of worsening solutions causes missing improving neighboring solutions.

For the permutation flowshop scheduling problem, again ADHS-AILLA statistically shows the best performance. Since SR-AILLA performs worse than ADHS-AILLA, it is apparent that ADHS provides a better hyper-heuristic opportunity. On the contrary, SR-AILLA consistently yields better results than SR-SA and SR-IE. This is a proof

BP	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	0.00594	**0.00650**	0.00703	0.00032	0.01177	0.01513	0.01668	0.00172
Inst2	0.00353	**0.00652**	0.00727	0.00107	0.01234	0.01583	0.02139	0.00262
Inst3	0.02029	**0.02187**	0.02275	0.00089	0.02110	0.02258	0.02371	0.00091
Inst4	0.01969	**0.02138**	0.02413	0.00151	0.02039	0.02239	0.02477	0.00138
Inst5	0.00034	**0.00077**	0.00456	0.00133	0.00034	0.00298	0.00510	0.00228
Inst6	0.00306	**0.00314**	0.00338	0.00010	0.00312	0.00440	0.00844	0.00214
Inst7	0.01099	**0.01352**	0.01584	0.00232	0.02226	0.02797	0.03223	0.00357
Inst8	0.02459	**0.02716**	0.03063	0.00245	0.08273	0.08684	0.09032	0.00269
Inst9	0.07273	**0.07830**	0.08560	0.00360	0.11710	0.12361	0.13272	0.00533
Inst10	0.01692	**0.02292**	0.02997	0.00377	0.03755	0.04219	0.04982	0.00449
BP	**SR-SA**				**SR-IE**			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	0.04048	0.04171	0.04441	0.00142	0.01197	0.01630	0.02041	0.00204
Inst2	0.03999	0.04213	0.04521	0.00179	0.01192	0.01521	0.01678	0.00197
Inst3	0.02455	0.02618	0.02723	0.00088	0.02275	0.02403	0.02519	0.00104
Inst4	0.02686	0.02906	0.03291	0.00174	0.02347	0.02494	0.02651	0.00111
Inst5	0.02230	0.02592	0.02773	0.00196	0.00035	0.00508	0.00647	0.00173
Inst6	0.02500	0.02536	0.02592	0.00034	0.00325	0.00555	0.00844	0.00250
Inst7	0.11090	0.11628	0.12430	0.00441	0.02217	0.02738	0.03205	0.00321
Inst8	0.12023	0.12507	0.13022	0.00305	0.08070	0.08456	0.08816	0.00282
Inst9	0.11598	0.11973	0.12552	0.00346	0.11372	0.12454	0.13352	0.00584
Inst10	0.03335	0.03451	0.03620	0.00089	0.03738	0.04309	0.05192	0.00559

Table 2 The results of the hyper-heuristics on the bin packing problem

FS	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	6263	**6293**	6308	12,9	6285	6299	6325	10,66
Inst2	6251	6273	6309	18,56	6226	**6258**	6280	17,57
Inst3	6315	**6337**	6354	11,41	6326	6339	6365	12,58
Inst4	6307	6358	6366	18,16	6323	**6348**	6366	20,14
Inst5	6365	**6383**	6402	12,43	6355	6389	6412	16,97
Inst6	10494	**10499**	10506	3,36	10502	10511	10526	8,53
Inst7	10923	**10928**	10972	15,5	10923	10934	10958	14,49
Inst8	26261	**26301**	26345	30,38	26301	26384	26435	44,73
Inst9	26754	**26792**	26816	21,75	26809	26865	26923	38,49
Inst10	26618	**26676**	26710	24,92	26638	26714	26751	33,98
FS	**SR-SA**				**SR-IE**			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	6291	6326	6364	20,12	6321	6347	6388	22,26
Inst2	6249	6296	6322	24,62	6261	6303	6340	26,68
Inst3	6335	6359	6386	18,17	6324	6369	6428	27,29
Inst4	6323	6351	6366	17,73	6330	6357	6374	15,24
Inst5	6364	6406	6456	27,56	6374	6405	6446	19,17
Inst6	10501	10538	10556	19,46	10497	10533	10555	19,9
Inst7	10923	10947	11017	30,33	10923	10947	11017	26,55
Inst8	26485	26519	26564	32,24	26413	26500	26649	74,37
Inst9	26888	26964	27046	42,16	26881	26978	27093	70,01
Inst10	26736	26781	26890	45,14	26732	26806	26876	59,29

Table 3 The results of the hyper-heuristics on the permutation flowshop scheduling problem

for the effect of the proposed move acceptance strategy. Besides, the performances of SR-SA and SR-IE are similar.

The experimental results (Table 4) on the personnel scheduling problem are hard to interpret because the number of iterations spent during the given execution time is limited. That is, there is not much time for exploration. Thus, a hyper-heuristic that reacts fast may have a higher chance to provide the best solutions. With the given

PS	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	3300	**3326**	3388	27.46	3317	3352	3391	22.93
Inst2	2115	**2231**	2495	108.26	2070	2362	2592	151.26
Inst3	295	**338**	380	25.41	405	1039	1885	677.92
Inst4	10	**21**	30	5.27	22	29	35	4.2
Inst5	17	**23**	29	3.88	26	30	36	3.06
Inst6	19	**25**	30	3.33	19	32	38	5.47
Inst7	1106	**1159**	1216	50.19	1117	1240	1404	78.35
Inst8	2176	2268	2359	56.46	2179	**2255**	2369	59.76
Inst9	3130	**3252**	3435	102.2	3166	3316	3527	118.08
Inst10	9438	**9624**	9781	107.33	9526	9642	9826	103.53
PS	**SR-SA**				**SR-IE**			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	3313	3368	3426	37.22	3307	3355	3432	36.6
Inst2	2295	2556	2725	135.63	2158	2558	3430	348.21
Inst3	480	1560	3160	1047.48	435	1033	1870	683.33
Inst4	17	27	36	5.1	25	30	36	3.44
Inst5	21	29	38	5.02	23	30	38	5.25
Inst6	21	31	43	6.33	24	32	39	5.44
Inst7	1100	1212	1411	102.46	1116	1268	1510	130.83
Inst8	2186	2256	2334	36.83	2183	2266	2464	73.47
Inst9	3159	3289	3347	61.69	3164	3296	3442	87.98
Inst10	9490	9646	9824	101.65	9380	9669	10050	192.04

Table 4 The results of the hyper-heuristics on the personnel scheduling problem

SAT	ADHS-AILLA				SR-AILLA			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	8	15	20	4.65	8	11	17	2.73
Inst2	24	29	37	4.62	20	25	29	2.82
Inst3	23	25	30	2.55	17	21	29	3.27
Inst4	7	9	11	1.51	7	10	13	1.84
Inst5	2	6	10	2.76	3	4	6	1.16
Inst6	3	6	11	2.25	3	5	7	1.45
Inst7	9	15	23	4.6	12	15	19	2.67
Inst8	8	18	32	8.17	8	24	39	11.64
Inst9	2	5	9	2.2	1	3	6	1.49
Inst10	2	5	8	1.89	2	3	6	1.35
SAT	**SR-SA**				**SR-IE**			
	MIN	AVG	MAX	STD	MIN	AVG	MAX	STD
Inst1	6	**9**	14	2.26	11	14	21	3.27
Inst2	20	**24**	27	2.21	24	27	31	2.07
Inst3	16	**18**	22	1.63	19	23	28	2.75
Inst4	5	**6**	7	0.82	18	24	27	2.95
Inst5	1	**3**	4	0.97	5	10	12	2.41
Inst6	1	**3**	4	1.14	6	11	14	2.46
Inst7	9	**12**	15	2.23	26	30	36	3.34
Inst8	3	**9**	16	4.04	37	44	50	3.84
Inst9	1	**3**	5	1.16	3	5	8	1.73
Inst10	0	**2**	3	0.82	3	5	7	1.51

Table 5 The results of the hyper-heuristics on the Max SAT problem

learning strategy, ADHS-AILLA performs the best. There is no statistically different performance between the rest of the hyper-heuristics.

In Table 5, the results on Max SAT are given. For this problem, SR-SA provides significant performance improvement differently from the other three problems. Besides, SR-AILLA takes the second place and leads to superior results compared to ADHS-AILLA and SR-IE. These results show that ADHS worsens the performance of

the hyper-heuristic for Max SAT. Moreover, SA functions way better than AILLA and IE.

	BP	**MSAT**	**FS**	**PS**	**OverAll**
ADHS-AILLA	**85.0**	**80.0**	**87.0**	56.0	**308.0**
HH1	38.0	37.0	27.5	**63.5**	166.0
HH2	39.0	70.5	24.0	63	196.5
HH3	76.0	47.5	19.5	19	162.0
HH4	59.0	34.5	61.0	54	208.5
HH5	9.0	0.5	10.0	37.0	56.5
HH6	61.0	67.0	71.0	1.0	200.0
HH7	22.0	42.0	26.0	58.0	148.0
HH8	1.0	11.0	64.0	38.5	114.5

Table 6 The comparison of ADHS-AILLA and the hyper-heuristics (HH1⇝HH8) from the CHeSC competition website (higher values are better)

In Table 6, ADHS-AILLA was compared to the hyper-heuristic results announced at the CHeSC 2011 competition website[1]. The comparison was made via the Formula 1 scoring system. The best hyper-heuristic gets 10, the rest get 8,6,5,4,3,2,1 based on their performance order. That is, only top eight hyper-heuristics earn points. The comparisons are carried out with only one run per instance. The comparison results show that ADHS-AILLA wins bin packing, Max SAT and the flowshop scheduling tracks. It comes fourth for the personnel scheduling problem. According to the over all performance, ADHS-AILLA wins.

4.2 Relay hybridisation

In Figure 2, the effect of the proposed relay hybridisation on different problems is shown. Consecutively applied heuristics reach new best solutions during the whole search process for the bin packing problem. On the other hand, for the max SAT and flowswhop scheduling problems, this happens only at the early stages of the search. For personnel scheduling, the effect of the hybridisation is also limited. However, as was mentioned before, the number of iterations spent for this problem is quite small compared to the other problems. For the bin packing problem, mostly the crossover operator is useful for generating the first solution that is sent to the second heuristic. As the second heuristic, one of the local search heuristics (LLH_4) performs as the best second heuristic within pairs. One can consider LLH_0, LLH_2 and LLH_3 as the other second heuristics. For max SAT, mutational heuristics (LLH_3) LLH_4, with the following local search heuristics LLH_0 and LLH_1 help the hyper-heuristic for finding new best solutions. For the permutation flowshop, mutational (LLH_0, LLH_1) and ruin-recreate (LLH_6, LLH_6) heuristics with local search heuristics (LLH_7, LLH_8, LLH_{10}) construct effective heuristic pairs. For personnel scheduling, again mutational (LLH_{11}) and ruin-recrate heuristics (LLH_5, LLH_6) with one particular local search heuristic (LLH_3) perform effectively. All these heuristic combinations indicate that the proposed hybridisation mechanism can determine effective heuristic pairs. As expected, heuristic pairs mostly behave as memetic algorithms.

[1] `http://www.asap.cs.nott.ac.uk/chesc2011/` – results on 15/01/2011

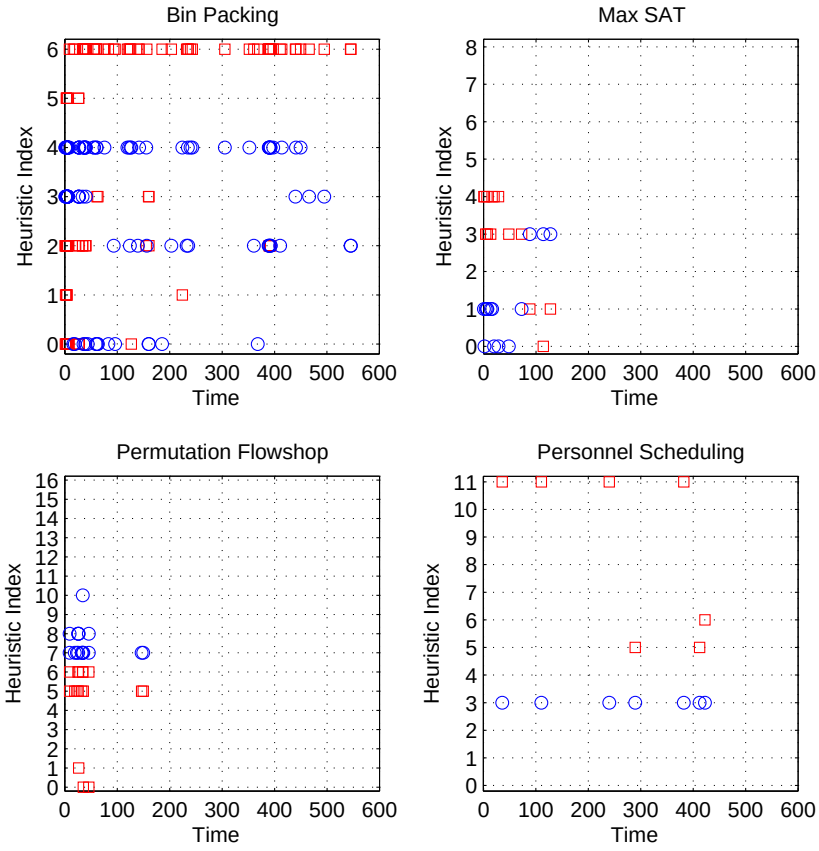

Fig. 2 Heuristic combinations from relay hybridisation (squares and circles show the applied heuristics consecutively)

4.3 Heuristic selection

In Figure 3, the effect of the proposed heuristic selection mechanism is shown. For the bin packing and permutation flowshop scheduling problems, ADHS calls more heuristics than SR during the given execution time. This situation reverses for the max SAT and personnel scheduling problems. This shows that the ADHS selection mechanism may invest time in time consuming but effective heuristics if it is required. In Figure 4, the number of heuristic calls of each heuristic over time is illustrated. For the given heuristic sets, the proposed selection strategy determines a distinction between heuristic performances. For the personnel scheduling and permutation flowshop scheduling problems, performance differences look stable. However, for the bin packing and max SAT problems, some heuristics behave differently during the search. In other words, some heuristics started to be selected more and others less. This shows that, for solving the bin packing and max SAT problems, considering performance changes compared to each other over time can be effective.

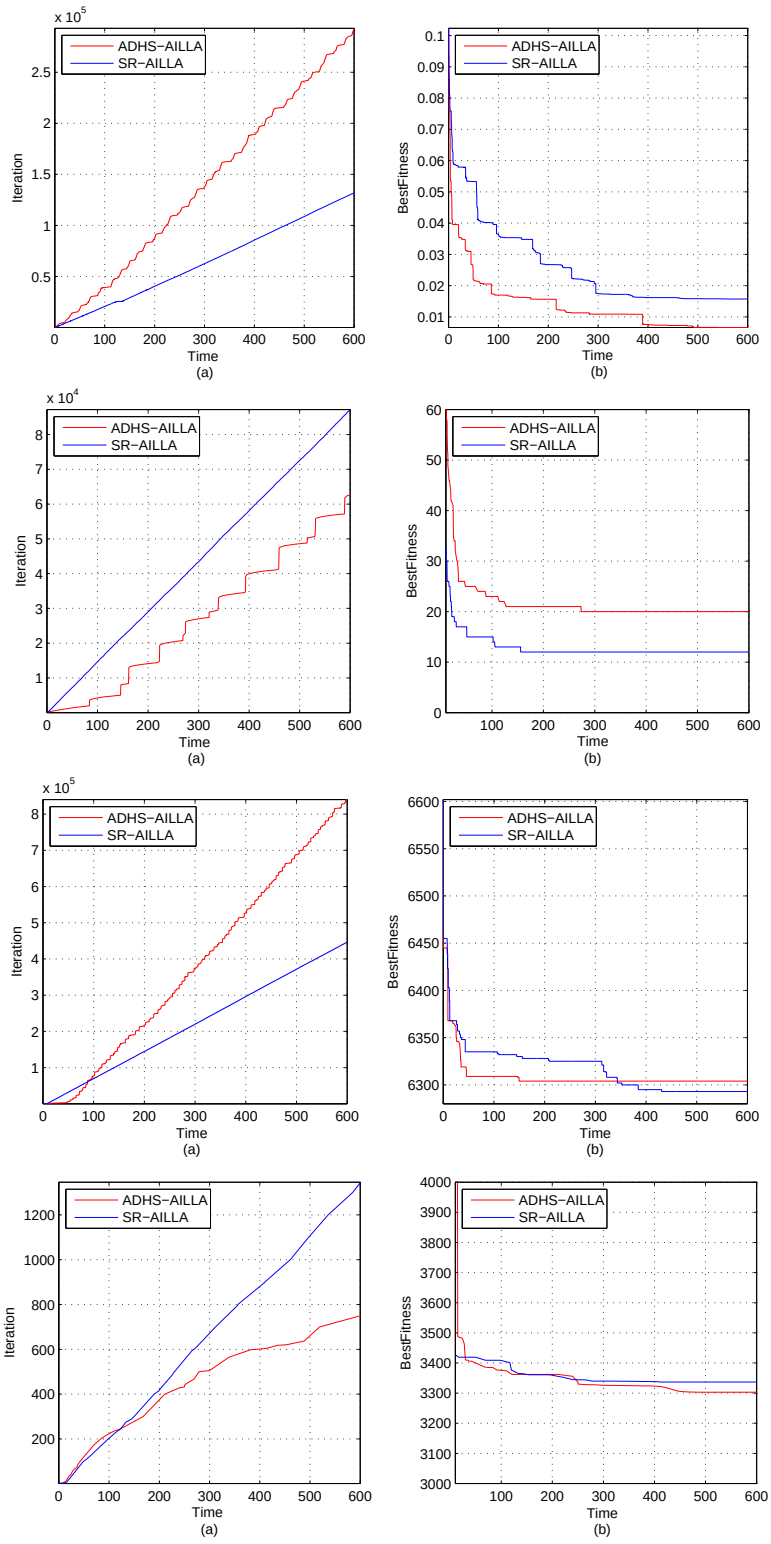

Fig. 3 The effect of the proposed selection mechanism (the chart pairs belong to the bin packing, max SAT, flowshop scheduling and personnel scheduling problems from top to bottom)

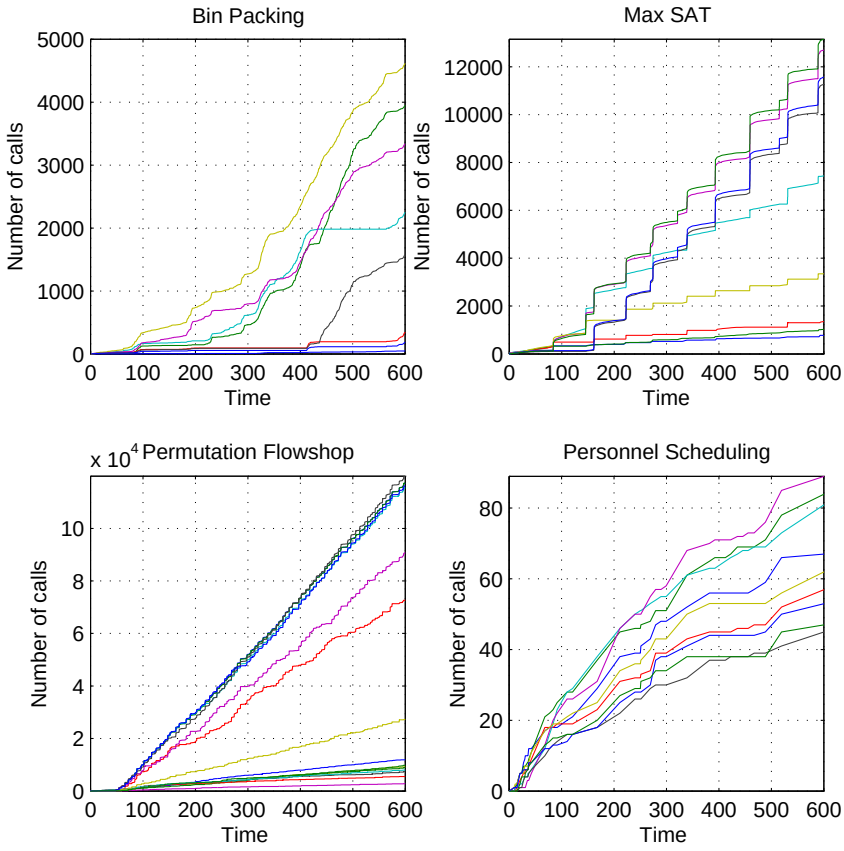

Fig. 4 The number of calls for each heuristic over the given problem domains by ADHS-AILLA

4.4 Varying performance of the heuristics

In Figure 5, the varying QI values of each heuristic during the search process are illustrated. The charts show that even if LLH_3 is a hill climber, it is frequently excluded from the heuristic set with the only crossover operator, LLH_6. For the remaining heuristics, it is possible to see performance changes based on the employed performance metric. During early iterations, a mutational heuristic (LLH_0), a ruin-recreate heuristic (LLH_2) and a hill climber (LLH_4) take the lead to improve the solution at hand. However, after a while, the other ruin-recreate heuristic (LLH_1) with a mutational heuristic (LLH_5) start to perform better. These results show that the performance variation of different heuristics should be regularly monitored and analysed.

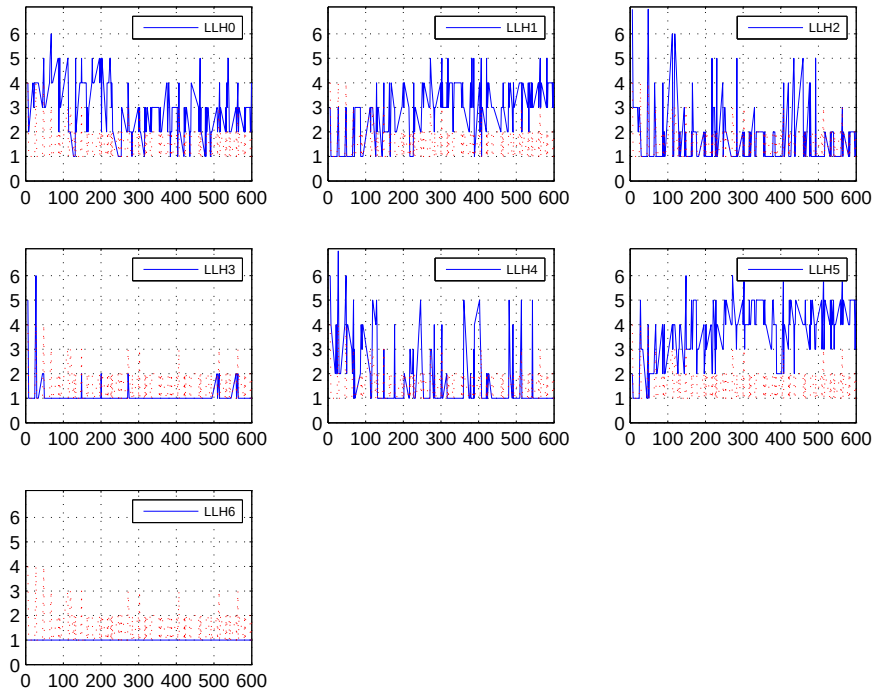

Fig. 5 *QI-avg* values of the bin packing heuristics (Quality index (QI) values regarding the performance of the heuristics are shown as solid lines, average (avg) of the QI values are shown as dotted lines. The heuristics with a lower QI than avg are excluded.)

4.5 Set size

In Figure 6, changes of the heuristic set size over time is demonstrated. For the bin packing problem, the exclusion process eliminates at most 3 heuristics and the set size generally changes between 4 and 5. For max SAT, fluctuations of the set size is considerably less. It can be explained by the number of phases because the existing SAT heuristics can spend more time than bin packing heuristics. The differences on the number of iterations per time is shown in Figure 3. For the flowshop scheduling problem, the set size is reduced to 6 out of 16 heuristics and it changes mainly between 6 and 9. For the last problem domain, even one phase could not be completed due to the previously mentioned speed issues.

5 Conclusion

In the literature, there exist different kinds of problems with various solution strategies. Many of these problems are hard to solve. Hence, there is a tendency of dedicated developing heuristic methods for solving these problems. Among these approaches, it is expected that some of them work exceptionally well on some problems or problem instances, but generate inferior results on others. Hyper-heuristics search over such

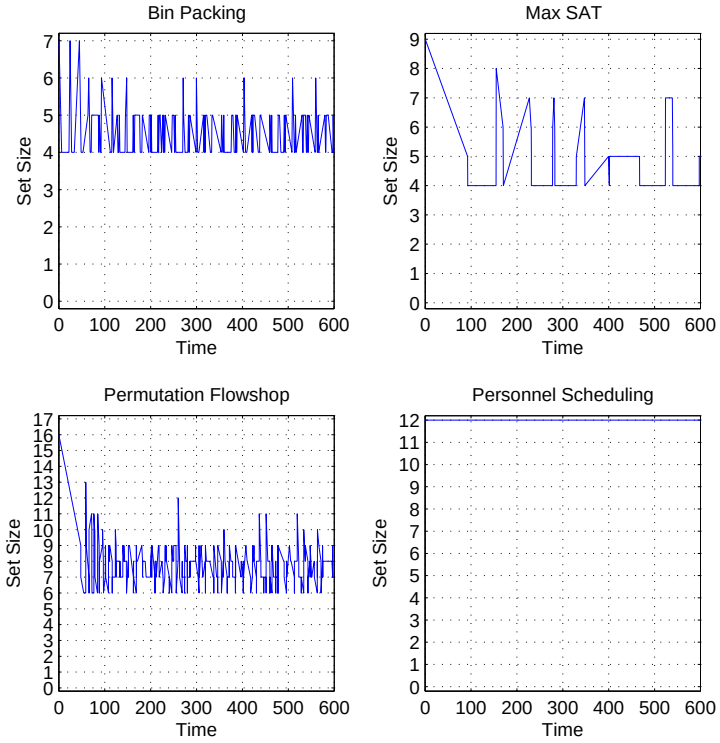

Fig. 6 Changes on the heuristic set size over time by ADHS-AILLA

low-level algorithms to provide generality by using their strengths. With problem-independency, their generality level increases. In this research, a new selection hyper-heuristic was designed and implemented on the hyper-heuristic software framework, i.e. HyFlex. The developed approach consists of a dynamic heuristic set strategy that determines the best heuristic subsets along a number of iterations. For increasing the effectiveness of this elimination method, a learning automaton and a pairwise heuristic hybridisation mechanism were employed. Also, a new threshold based dynamic move acceptance strategy was accommodated. A set of experiments was carried out with three additional hyper-heuristics over four problem domains with their specific heuristic sets. The experimental results indicated that the designed hyper-heuristic is an effective strategy for the given problem instances and the heuristic sets. Although the hyper-heuristic performs well on average, it still has some issues. For the Max SAT problem, SA generates superior results compared to AILLA. Moreover, ADHS performs worse than SR for the same problem. The related results should be analysed to increase the performance of the proposed hyper-heuristic. In addition, since there is room for improvement concerning the personnel scheduling problem, the missing elements or misleading algorithm settings should be determined.

In the future, the mentioned drawbacks will be investigated. Then, a feedback mechanism will be settled between the selection process and the move acceptance part. In addition, a two-phase heuristic selection mechanism consisting of selecting a

heuristic subset and choosing a heuristic from the selected heuristic set in connection with a feedback mechanism will be utilised. Furthermore, the adaptive characteristic of the move acceptance strategy will be boosted.

References

1. E. Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2nd edition, 2010.
2. M. Bader-El-Den, R. Poli, and S. Fatima. Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework. *Memetic Computing*, 1(3):205–219, 2009.
3. B. Bilgin, P. Demeester, M. Misir, W. Vancroonenburg, G. Vanden Berghe, and T. Wauters. A hyper-heuristic combined with a greedy shuffle approach to the nurse rostering competition. In *the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10) - the Nurse Rostering Competition*, 2010.
4. M. Bowling and M. Veloso. Rational and convergent learning in stochastic games. In *International Joint Conference on Artificial Intelligence*, volume 17, pages 1021–1026. Citeseer, 2001.
5. E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward. A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. *IEEE Transactions on Evolutionary Computation*, 14(6):942–958, 2010.
6. E.K. Burke, T. Curtois, M. Hyde, G. Kendall, G. Ochoa, S. Petrovic, and J.A. Vazquez-Rodriguez. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *Proceedings of the 4th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA'09)*, pages 790–797, Dublin, Ireland, August 10–12 2009.
7. E.K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg. *Handbook of Meta-Heuristics*, chapter Hyper-Heuristics: An Emerging Direction in Modern Search Technology, pages 457–474. Kluwer Academic Publishers, 2003.
8. E.K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Ozcan, and J.R. Woodward. A classification of hyper-heuristic approaches. *Handbook of Metaheuristics*, pages 449–468, 2010.
9. E.K. Burke, M. Hyde, G. Kendall, and John Woodward. Automatic heuristic generation with genetic programming: Evolving a jack-of-all-trades or a master of one. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO'07)*, pages 1559–1565, London, England, July 12–16 2007.
10. E.K. Burke, M.R. Hyde, and G. Kendall. Evolving bin packing heuristics with genetic programming. In T.P. Runarsson, H-G.B., E. Burke, J.J. Merelo-Guervos, L.D. Whitley, and X. Yao, editors, *Proceedings of the 9th Parallel Problem Solving from Nature (PPSN'IX)*, volume 4193 of *LNCS*, pages 860–869, Reykjavik, Iceland, September 9–13 2006. Springer-Verlag.
11. E.K. Burke, G. Kendall, M. Misir, and E. Ozcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, pages 1–18, 2010. 10.1007/s10479-010-0782-2.
12. E.K. Burke, G. Kendall, D.L. Silva, R. O'Brien, and E. Soubeiga. An ant algorithm hyper-heuristic for the project presentation scheduling problem. In *Proceedings of the Congress on Evolutionary Computation 2005 (CEC'05). Volume 3*, pages 2263–2270, 2005.
13. E.K. Burke, G. Kendall, and E. Soubeiga. A tabu-search hyper-heuristic for timetabling and rostering. *Journal of Heuristics*, 9(3):451–470, 2003.
14. E.K. Burke, S. Petrovic, and R. Qu. Case based heuristic selection for timetabling problems. *Journal of Scheduling*, 9(2):115–132, 2006.
15. K. Chakhlevitch and P. Cowling. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In G. Raidl and J. Gottlieb, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3448 of *LNCS*, pages 23–33. Springer Berlin / Heidelberg, 2005.
16. P. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In *PATAT '00: Selected papers from the Third International Conference on Practice and Theory of Automated Timetabling III*, pages 176–190, London, UK, 2001. Springer-Verlag.
17. T. Curtois, G. Ochoa, M. Hyde, and J. A. Vazquez-Rodriguez. A hyflex module for the personnel scheduling problem. Cs technical report, Univeristy of Nottingham, 2010.

18. K.A. Dowsland, E. Soubeiga, and E.K. Burke. A simulated annealing hyper-heuristic for determining shipper sizes. *European Journal of Operational Research*, 179(3):759–774, 2007.
19. A.S. Fukunaga. Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, 16(1):31–61, 2008.
20. L. Han and G. Kendall. An investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Proceedings of Congress on Evolutionary Computation (CEC'03)*, volume 3, pages 2230–2237, 2003.
21. M. Hyde, G. Ochoa, T. Curtois, and J. A. Vazquez-Rodriguez. A hyflex module for the maximum satisfiability (max-sat) problem. Cs technical report, Univeristy of Nottingham, 2010.
22. M. Hyde, G. Ochoa, T. Curtois, and J. A. Vazquez-Rodriguez. A hyflex module for the one dimensional bin packing problem. Cs technical report, Univeristy of Nottingham, 2010.
23. G. Kendall and N.M. Hussin. An investigation of a tabu-search-based hyper-heuristic for examination timetabling. In *Multidisciplinary scheduling: theory and applications: 1st International Conference, MISTA'03: Nottingham, UK, 13-15 August 2003: selected papers*, page 309. Springer Verlag, 2005.
24. G. Kendall and N.M. Hussin. A tabu search hyper-heuristic approach to the examination timetabling problem at the mara university of technology. In *Proceedings of the 5th Practice and Theory of Automated Timetabling (PATAT'04)*, volume 3616 of *LNCS*, pages 270–293. Springer, 2005.
25. D.S. Lee, V.S. Vassiliadis, and J.M. Park. List-based threshold-accepting algorithm for zero-wait scheduling of multiproduct batch plants. *Industrial & Engineering Chemistry Research*, 41(25):6579–6588, 2002.
26. J.G. Marn-Blzquez and S. Schulenburg. A hyper-heuristic framework with xcs: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In T. Kovacs, X. Llor, K. Takadama, P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *IWLCS*, volume 4399 of *LNCS*, pages 193–218. Springer, 2007.
27. M. Misir, K. Verbeeck, P. De Causmaecker, and G. Vanden Berghe. Hyper-heuristics with a dynamic heuristic set for the home care scheduling problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'10)*, pages 2875–2882, Barcelona, Spain, July 18–23 2010.
28. M. Misir, T. Wauters, K. Verbeeck, and G. Vanden Berghe. A new learning hyper-heuristic for the traveling tournament problem. In *Proceedings of the 8th Metaheuristic International Conference (MIC'09)*, Hamburg, Germany, 2009.
29. A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer Decision-Making*, pages 523–544. Kluwer Academic Publishers, 2003.
30. E. Ozcan, B. Bilgin, and E.E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.
31. E. Ozcan, M. Misir, G. Ochoa, and E.K. Burke. A reinforcement learning - great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing*, 1(1):39–59, 2010.
32. P. Ross, S. Schulenburg, J.G. Marín-Blázquez, and E. Hart. Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems. In W. B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, A. C. Schultz, J. F. Miller, E. Burke, and N. Jonoska, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'02)*, pages 942–948, New York, July 9–13 2002. Morgan Kaufmann Publishers.
33. M.A.L. Thathachar and P.S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.
34. W. Vancroonenburg, M. Misir, B. Bilgin, P. Demeester, and G. Vanden Berghe. A hyper-heuristic approach for assigning patients to hospital rooms. In *Proceeedings of the 8th International Conference on the Practice and Theory of Automated Timetabling (PATAT'10)*, Belfast, Northern Ireland, August 10–13 2010.
35. J. A. Vazquez-Rodriguez, G. Ochoa, T. Curtois, and M. Hyde. A hyflex module for the permutation flow shop problem. Cs technical report, Univeristy of Nottingham, 2010.