Heuristic Selection in a Multi-phase Hybrid Approach for Dynamic Environments

Gönül Uludağ*, Berna Kiraz*, A. Şima Etaner Uyar†, Ender Özcan‡
* Institute of Science and Technology, Istanbul Technical University, Istanbul, Turkey 34469
Email: uludagg@itu.edu.tr, berna.kiraz@marmara.edu.tr
† Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey 34469
Email: etaner@itu.edu.tr
‡ School of Computer Science, University of Nottingham, Nottingham, UK NG8 1BB
Email: Ender.Ozcan@nottingham.ac.uk

***Abstract*—An iterative selection hyper-heuristic method controls and mixes a set of low-level heuristics while solving a given problem. A low-level heuristic is selected and employed for improving a (set of) solution(s) at each step. This study investigates the influence of different heuristic selection methods within a population based incremental learning algorithm and hyper-heuristic based hybrid multi-phase framework for solving dynamic environment problems. Even though the hybrid method delivers a good overall performance, it is superior in cyclic environments. The empirical results show that a heuristic selection method that relies on a fixed permutation of the underlying low-level heuristics, combined with a strategy that guarantees diversity when the environment changes is more successful than the learning approaches across different cyclic dynamic environments produced by a well known benchmark generator.**

I. Introduction

In some difficult optimisation problems, environment changes over time while the optimisation process is in progress. These types of problems are referred to as dynamic environment problems. When performing a search for the best solution in such environments, the dynamism is often ignored and traditional optimisation methodologies are employed. However, the key to success for an optimisation algorithm in dynamic environments is the capability of adapting itself with respect to the changes. There are many approaches in literature that are used for solving dynamic environment problems [1]–[3]. It has been observed that different approaches handle different types of dynamism in the environment better comparably. This implies that the properties of the dynamism needs to be known beforehand to be able to choose the most appropriate optimisation approach, however, this may not be the case given a dynamic environment problem.

Statistical Model-based Optimization Algorithms are adaptive in nature and are expected to be able to track the changes in the search space, if it occurs. Hence, they are potentially good approaches for solving dynamic environment problems. The use of such algorithms has been growing in recent years. Probabilistic model-based techniques, for example Estimation of Distribution Algorithms (EDAs) are among the most common ones used within these approaches. In EDAs, the probabilistic distribution model, learned based on the current best candidate solutions, are used to create new candidate solutions. Univariate marginal distribution algorithm (UMDA) [4], Bayesian optimization algorithm (BOA) [5] and population based incremental learning (PBIL) [6] are among the most commonly used EDAs in literature. There have been some studies also that apply improved EDAs in dynamic environments [7]–[13].

Heuristic and meta-heuristic approaches mostly utilize problem domain specific information and operate directly on the solution space. An EDA is a type of meta-heuristic approach which conducts population based search over the space of solutions. Hyper-heuristics [14] are more general methods designed for solving different computationally difficult problems even from different domains. They perform search over the space formed by a set of low-level heuristics which perturb or construct a (set of) candidate solution(s) [15], [16]. Hyper-heuristics operate at a higher level communicating with the problem domain through a domain barrier as they perform search over the heuristics. Any type of problem specific information is filtered through the domain barrier. Due to this feature, a hyper-heuristic can be directly employed in various problem domains without requiring any change, of course, through the use of appropriate domain specific low-level heuristics. This gives hyper-heuristics an increased level of generality. There are two main types of hyper-heuristics; methodologies that *generate* and *select* heuristics. This study focuses on the selection hyper-heuristic methodologies. An iterative selection hyper-heuristic consists of a *heuristic selection* and *move acceptance* components. A (set of) current solution(s) is modified by application of a heuristic to a solution in hand which is chosen by the heuristic selection method at each step. Then the new (set of) solution(s) is accepted or rejected using the move acceptance method. This process continues until the termination criteria are satisfied. More on hyper-heuristics including different heuristic selection and move acceptance components in literature can be found in [14]. There is strong empirical evidence showing that selection hyper-heuristics are able to quickly adapt without any external intervention in a given dynamic environment providing effective solutions [17]–[19].

In order to exploit the advantages of approaches with learning and those with model-building features in dynamic environments, we proposed a hybridization of EDAs with hyper-heuristics in the form of a multi-phase framework in [20]. In this study, we extend our previous studies and perform exhaustive tests to empirically analyze and explain the behavior of such an EDA and hyper-heuristic hybrid and try to determine a selection method which performs well within the previously proposed framework.

The rest of the paper is organized as follows. Section II describes the proposed multi-phase hybrid approach which combines selection hyper-heuristics and multi-population EDAs. The empirical analysis of this hybrid approach over a set of dynamic environment benchmark problems and the experimental design are provided in section III. Finally, section IV discusses the conclusion and future work.

II. A PBIL and Hyper-heuristic Hybrid

In our previous study we proposed a hybrid framework [20], which combined aspects of hyper-heuristics and multi-population EDA approaches. The proposed framework can combine any multi-population EDA with various selection hyper-heuristics. In our preliminary study, we hybridised a two population PBIL approach with hyper-heuristics. We called this approach HH-PBIL2. HH-PBIL2, which is inspired by SPIL2 introduced in [12], consists of two phases:

In the first phase, probability vectors $\overrightarrow{P}list$ corresponding to a set of different environments are learned in an offline manner using standard PBIL (SPBIL). Then, those learned probability vectors are stored for later use in the second phase of HH-PBIL2. In SPBIL, a posterior probability distribution model of promising solutions is built using statistical information obtained from the population of solution candidates. Two main steps, learning and sampling are used in SPBIL. SPBIL algorithm is initialized with the central probability vector that creates the initial population by sampling. The real-valued probability vector $\overrightarrow{P}(t) = \{p_1, p_2, ..., p_l\}$ (l is the binary-encoded length) is learned by using the best sample(s) $\overrightarrow{B}(t)$ at each t iteration as $p_i(t+1) := (1-\alpha)p_i(t) + \alpha B_i(t), \quad i = \{1, 2, ..., l\}$, where α is the learning rate. A bitwise mutation is applied to the probability vector for conserving the diversity. Then a set $S(t)$ of n candidate solutions are sampled from the updated probability vector.

In the second phase, the probability vectors serve as the low-level heuristics the hyper-heuristic manages. Figure 1 shows a simple diagram illustrating the execution of HH-PBIL2. As seen in Figure 1, $\overrightarrow{P}list$ represents the probability vectors learned during the offline learning phase.

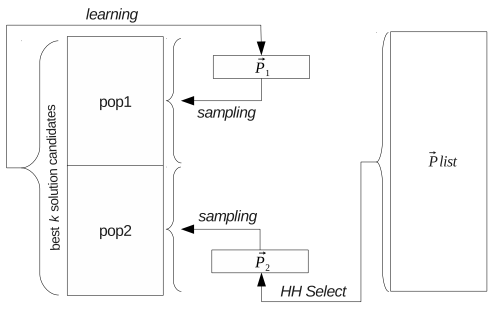

Figure 1: The framework of HH-PBIL2

The second phase of HH-PBIL2 framework uses a multi-population scheme for SPBIL named SPBIL2. The population is divided into two sub-populations and two probability vectors, one for each sub-population, are used in parallel. As seen in Figure 1, $pop1$ represents the first sub-population and $\overrightarrow{P}_1$ is its corresponding probability vector, $pop2$ represents the second sub-population and $\overrightarrow{P}_2$ is its corresponding probability vector. The pseudocode of the proposed HH-PBIL2 is shown in Algorithm 1.

In HH-PBIL2, the first probability vector $\overrightarrow{P}_1$ is initialized to the central probability vector, and the initial second probability vector $\overrightarrow{P}_2$ is selected from the $\overrightarrow{P}list$ randomly. Initial sub-populations are sampled independently from their own probability vectors and their sizes are equal. After the fitness calculation, the sub-population sample sizes are slightly adjusted within the range $[0.3 * n, \ 0.7 * n]$ according to their best fitness values. $\overrightarrow{P}_1$ is learned towards the best solution candidate(s) in the first sub-population and mutation is applied to $\overrightarrow{P}_1$. $\overrightarrow{P}_2$ is selected using the heuristic selection methods from $\overrightarrow{P}list$. Then, two sub-populations are sampled based on their respective probability vectors. The approach repeats the cycle until some termination criteria are met. In the HH-PBIL2 framework, different heuristic selection methods can be used for selecting the second probability vector from $\overrightarrow{P}list$.

There are many heuristic selection methods proposed in literature [15], [16]. Some of these methods include Simple Random (SR), Random Descent (RD),

Algorithm 1 Pseudocode of the proposed HH-PBIL2 approach

```
1:  $t := 0$
2:  initialize $\overrightarrow{P}_1(0) := \overrightarrow{0.5}$
3:  $\overrightarrow{P}_2(0)$ is selected from $\overrightarrow{P}list$
4:  $S_1(0) := sample(\overrightarrow{P}_1(0))$ and $S_2(0) := sample(\overrightarrow{P}_2(0))$
5:  while (termination criteria not fulfilled) do
6:      evaluate $S_1(t)$ and evaluate $S_2(t)$
7:      adjust next population sizes for $\overrightarrow{P}_1(t)$ and $\overrightarrow{P}_2(t)$ respectively
8:      place $k$ best samples from $S_1(t)$ and $S_2(t)$ into $\overrightarrow{B}(t)$
9:      send best fitness from whole/second population to heuristic
        selection methods
10:     learn $\overrightarrow{P}_1(t)$ toward $\overrightarrow{B}(t)$
11:     mutate $\overrightarrow{P}_1(t)$
12:     $\overrightarrow{P}_2(t)$ is selected using heuristic selection
13:     $S_1(t) := sample(\overrightarrow{P}_1(t))$ and $S_2(t) := sample(\overrightarrow{P}_2(t))$
14:     $t := t + 1$
15: end while
```

Random Permutation (RP), Random Permutation Descent (RPD), Reinforcement Learning (RL), Greedy selection (GR), and Choice Function (CF). SR, randomly selects a low-level heuristic and applies it to the candidate solution. RD applies a randomly selected heuristic to the current candidate solution repeatedly as long as solution improves, otherwise another low-level heuristic is selected randomly. RP randomly orders all low-level heuristics and applies each heuristic in turn. RPD selects a low-level heuristic in the same way as RP, but it applies the selected heuristic repeatedly as long as the solution improves. In RL, each low-level heuristic has a utility score. The scores of each heuristic are initialized to the same value and updated during the search process based on its performance. At each step, the low-level heuristic with the maximum score is selected. If the selected heuristic produces a better solution than the previous one, it is rewarded by increasing its score, otherwise it is penalized by decreasing its score. The scores are restricted to vary between predetermined lower and upper bounds. These heuristic selection methods can be used for hybridization in HH-SPBIL2.

GR applies all low-level heuristics one by one to the candidate solution and the best new solution and so the heuristic is selected. For HH-PBIL2, this means that a whole sub-population will be sampled for each vector and the resulting individuals will be evaluated to select the best low-level heuristic. In CF, when scoring a heuristic, the difference between the fitness values of the offspring and the current candidate solution is taken into account. In a dynamic environment setting, this means that whenever a change occurs, the current candidate solution has to be re-evaluated in the new environment. For HH-PBIL2, this involves re-evaluating all the candidate solutions in the current population. Both GR, in its basic form as explained above, and CF are computationally inefficient when used in HH-PBIL2. Therefore, we do not employ these two heuristic selection methods in the rest of the study.

III. Experiments

In this study, we perform experiments in two parts. First, we conduct a series of tests to do a comprehensive performance analysis for explaining the behavior of HH-PBIL2. Then, we experiment with a set of selection heuristics to analyze their behavior and determine the one that performs best when used within HH-PBIL2. In some of the tests, SPBIL, SPBIL2 and HH-PBIL2 will be used for comparisons.

In order to generate dynamic environments, we use the XOR generator [6], [21]. In the offline learning stage, first a set of M XOR masks are randomly generated. Then, for each mask (i.e. environment), SPBIL is executed for 100 independent runs where each run consists of $10,000$ generations. During offline learning, each environment is stationary. At the end, the probability vector producing the best solution found so far over all runs for each environment, is stored in $\overrightarrow{P}list$.

We use three Decomposable Unitation-Based Functions (DUFs) [12] together with the XOR generator. All DUFs are composed of 25 copies of 4-bit building blocks. Each building block is denoted as a unitation-based function $u(x)$ which gives the number of ones in the corresponding building block. Its maximum value is 4. The fitness of a bit string is calculated as the sum of the $u(x)$ values of the building blocks. The optimum fitness value for all DUFs is 100. DUF1 is the *OneMax* problem whose objective is to maximize the number of ones in a bit string. $DUF2$ has a unique optimal solution surrounded by four local optima and a wide plateau with eleven points having a fitness of zero. $DUF2$ is more difficult than $DUF1$. $DUF3$ is fully deceptive. The mathematical formulations of the DUFS, as given in [12], can be seen below.

$$f_{DUF1} = u(x) \tag{1}$$

$$f_{DUF2} = \begin{cases} 4 & \text{, if } u(x) = 4 \\ 2 & \text{, if } u(x) = 3 \\ 0 & \text{, if } u(x) < 3 \end{cases} \tag{2}$$

$$f_{DUF3} = \begin{cases} 4 & \text{, if } u(x) = 4 \\ 3 - u(x) & \text{, if } u(x) < 4 \end{cases} \tag{3}$$

SPBIL, SPBIL2 and HH-PBIL2 share some common settings which are used as suggested in [12]. The problem consists of 25 building blocks of length 4, therefore solution candidates are of length 100. Mutation shift is taken as 0.05 and 3 best candidate solutions are used in the online learning of probability vectors. The population size is set to 100. Both in SPBIL2 and HH-PBIL2, each sub-population size is initialized as 50 and they are allowed to vary between 30 and 70.

In the first phase of HH-PBIL2, probability vectors corresponding to a set of different environments are learned offline using SPBIL. During this phase, mutation rate and learning rate are taken as 0.02 and 0.25 respectively.

We experiment with two main types of dynamic environments: randomly changing environments and cyclic environments. To generate dynamic environments showing different dynamism properties, we consider different change frequencies τ, change severities ρ and cycle lengths CL. The values used in the experiments for these parameters are determined through a set of preliminary experiments. In the cyclic environments, we assume that the environments return to their exact previous locations. We determined the change periods to correspond to low frequency (LF), medium frequency (MF) and high frequency (HF) changes as 50 generations, 25 generations and 5 generations respectively for DUF1 and DUF2 and as 100 generations, 35 generations and 10 generations respectively for DUF3. In convergence plots, these settings for LF, MF and HF correspond respectively to stages where the algorithm has been converged for some time, where it has not yet fully converged and where it is very early on in the search. In randomly changing environments, for change severities, 0.1, 0.2, 0.5 and 0.75 are chosen to denote low severity (LS), medium severity (MS), high severity (HS) and very high severity (VHS) changes respectively, for random dynamic environments. These are determined based on the definition of the XOR generator. For cyclic environments, the cycle lengths CL are selected as 2, 4 and 8.

For each run of the algorithms, 128 changes occur after the initial environment. Therefore, the total number of generations in a run is calculated as $maxGenerations = changeFrequency * changeCount$. In order to compare the performance of the algorithms, the results are reported in terms of offline error [1], which is calculated as the cumulative average of the differences between the best values found so far and the optimum value at each time step.

$$\frac{1}{T}\sum_{t=1}^{T} \mid opt_t - e_t* \mid \tag{4}$$

$$e*_t = max(e_\tau, e_{\tau+1}, ..., e_t)$$

where T is the total number of evaluations and τ is the last time step $(\tau < t)$ when change occurred.

Fitness values are calculated using the corresponding DUF definitions given above. In all our experiments, while the location of the global optimum may change, its fitness value remains the same and is always 100. The main aim in the optimisation here is to minimize the offline error. All reported results are averages of final offline errors achieved at the end of the runs, over 100 independent runs.

One-way ANOVA and Tukey HSD tests at a 95% confidence level are performed to observe whether the pairwise performance variations between the approaches are statistically significant or not. Where statistical significance tests are reported, the following notation is used: Given A vs B, $s+$ ($s-$) denotes that A (B) is performing statistically better than B (A), while $A \geq B$ ($A \leq B$) indicates that A (B) performs slightly better than B (A) and this performance difference is not statistically significant.

In the first set of experiments, we investigate the effect of the number of low-level heuristics, i.e. the learned probability vector counts. In our previous study [20], we compared two variants of HH-PBIL2 denoted RL-PF and RL-P2 with the SPBIL and SPBIL2 approaches proposed in [12]. In the RL-PF variant, the RL heuristic selection method is used for hybridization, and the best performing candidate solution(s) from the whole population, i.e. from the two populations combined, are used to update the score. In the RL-P2 variant, the best performing solution candidate(s) from only the second population is used to update the score. The results showed that the RL-P2 variant outperforms the others. Therefore, for this experiment we only used the RL-P2 variant. As the number of low-level heuristics, we test the following values: $8, 16, 32, 64$. For this experiment, mutation rate P_m and learning rate α are taken as 0.02 and 0.25 respectively. The RL settings are taken as recommended in [22]. The results of the ANOVA and Tukey's HSD tests for statistical significance are reported in Table I. In the table, each entry shows the total number of times the corresponding approach achieves the corresponding significance state ($s+$, $s-$, $\geq$ and $\leq$) over the others on the three DUFs for different change severity and frequency settings in randomly changing environments and for different cycle length and change frequency settings in cyclic environments. From the table, we can see that $M = 8$ is better. So, for the rest of the experiments, the number of low-level heuristics, i.e. learned probability vector counts, is taken as 8.

Table I: Overall counts for the RL-P2 approach.

M	$s+$	$s-$	$\geq$	$\leq$
8	79	57	20	33
16	66	59	37	27
32	71	56	36	26
64	37	81	32	39

In the second set of experiments, we explore the performance of the RL-PF, RL-P2, SPBIL, and SPBIL2 approaches with different combinations of restart schemes. All restart schemes are scored according to the Formula 1 scoring system. For each algorithm, the median offline error value of 100 runs is calculated. Then, the results are ordered by median values and the top 8 algorithms are assigned a score of 10, 8, 6, 5, 4, 3, 2 and 1 point respectively [23]. Considering both random and cyclic environments there are 63 different problems, therefore, the maximum overall score that an algorithm can get is 630.

For HH-PBIL2 the following restart schemes are used. The original scheme (there is no restart) is denoted as $Case0$. In the $Case1$ scheme, only the first probability vector and in the $Case2$ scheme, only the second probability vector is reset to the central probability vector when environmental change is detected. In the $Case3$ scheme, the second probability vector is initialized to the central probability vector at the beginning of the run but is not reset when change occurs. In the $Case4$ scheme, like in the previous case, the second probability vector is initialized to the central probability vector at the beginning of the algorithm. However, here, the central probability vector is also included in the list of low-level of heuristics $\overrightarrow{Plist}$. For SPBIL, the probability vector is reset to the central probability vector when change is detected. This is denoted as $Case1$. For SPBIL2, when environmental change is detected, in the $Case1$ scheme, only the first probability vector is reset to the central probability vector and in the $Case2$ scheme, only the second probability vector is initialized randomly. In the $Case3$ scheme, the first probability vector is restarted with the central probability vector and the second probability vector is initialized randomly when environmental change is detected.

A total of 16 different schemes are scored, but due to lack of space, only the best 8 schemes according to the ranking results are shown in Table II. In [23], the scoring was only done based on a ranking performed using the median values. To give a better idea of the behavior of the approaches, here, we calculate three scores based on the median values, the best values and the average values. In all rankings, even though the ordering may be slightly different, the first 8 approaches were the same. Looking at the results in the table we can see that the first 6 schemes achieved much better scores than the others. Therefore, for the next step of the experiments, we will only consider these 6 schemes.

After performing the ranking, we also counted the overall significance states ($s+$, $s-$, $\geq$ and $\leq$) as before, on the three DUFs for different change severity and frequency settings in randomly changing environments and for different cycle length and change frequency settings in cyclic environments for the best 6 schemes. According to the results in Table II, the RL-P2-Case1 scheme outperforms the others.

In all approaches (SPBIL, SPBIL2, HH-PBIL2), there are two important parameters, which may affect

Table II: The best 8 schemes according to the Formula 1 ranking based on median, best and average offline error values.

Algorithms	Median	Best	Average
RL-P2-Case1	386	363	378
SPBIL-Case1	368	333	369
RL-PF-Case1	282	294	289
RL-P2-Case3	222	218	212
SPBIL2-Case2	218	205	220
RL-P2-Case0	201	222	206
SPBIL2-Case1	165	150	165
RL-P2-Case4	146	156	148

Table III: Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for the 6 selected schemes

Algorithms	$s+$	$s-$	$\geq$	$\leq$
RL-P2-Case1	185	64	24	42
SPBIL-Case1	174	116	21	4
RL-PF-Case1	131	125	29	30
SPBIL2-Case2	108	160	23	24
RL-P2-Case0	93	159	29	34
RL-P2-Case3	92	159	36	28

the performance of the algorithms: the mutation rate P_m and the learning rate α. In order to investigate the effects of the selection of these parameters, different P_m values are tested in combination with different α values for the RL-P2-Case1 scheme. For these set of experiments, {0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1} values and {0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95} values are tested for P_m and for α respectively. The experiments are performed for medium severity (MS) and medium frequency (MF) settings for randomly changing environments and changes at a cycle length of CL=4 and medium frequency (MF) for cyclic environments. Figure 2 and Figure 3 show the plots for the offline error values versus the different α values with different P_m values in randomly and cyclic changing environments respectively for only DUF2. For lack of space, the results for the other DUFs are not given, however, they show similar tendencies. The plots show that, the best P_m and α combination is 0.1 for P_m and 0.35 for α. However, it can also be seen that the setting of P_m is not very critical, especially in the cyclic environments. For a good setting of α, all P_m values give similar results. The setting of α seems to be more effective in performance, however, the plots show that it is not very sensitive, i.e. values between 0.2 and 0.6 produce acceptable results.

In the final set of experiments, we explore the effects of the heuristic selection methods on performance using the results obtained in the previous set of experiments. For these tests, we use 0.1 for P_m and 0.35 for α as selected in the previous tests. We experiment with five heuristic selection methods SR, RD, RP, RPD,

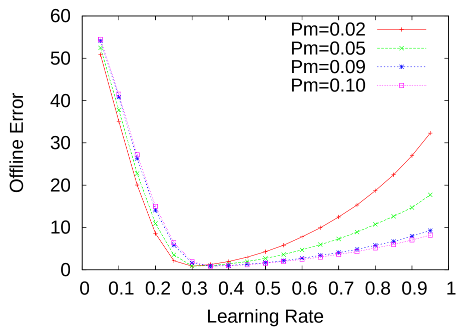

Figure 2: Offline error values versus the different α values with different P_m values in randomly changing environments for MF and MS changes on DUF2

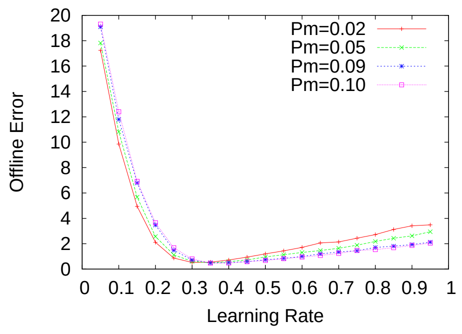

Figure 3: Offline error values versus the different α values with different P_m values in cyclic environments for MF and MS changes on DUF2

RL for the XX-P2-Case1 hyper-heuristic scheme (XX stands for the heuristic selection method). For each heuristic selection scheme, we looked at the offline errors achieved at the end of a run, averaged over 100 runs, for all DUFs.

The results for randomly changing environments are provided IV. For randomly changing environments, all heuristic selection schemes performed well and there were no statistically significant differences between the results.The results for the cyclic environments are summarized in Table V. For each column, the best scheme is highlighted. The results show that for DUF1 and DUF2, in the tested cyclic environments, RP performs the best as a heuristic selection method in the HH-PBIL2 framework. For DUF3, RPD seems to produce better results than RP, however, when the actual values are observed, it can be seen that they are very close. This result was contrary to our initial intuition that a heuristic selection scheme with learning would be better, especially in cyclic environments. However, this is not surprising. In the cases where the period of the change is larger than the number of heuristics, the RP approach becomes equivalent to GR, since each heuristic is applied to each environment at least once, regardless of the permutation. Therefore, the heuristic, i.e. the probability vector that was specifically learned for that environment, ends up being applied at least once in that environment. As the offline error calculation considers the improving results at each step, the offline error produced by the heuristic trained specifically for that environment, which is expected to be very low, gets included. Thus, the final offline error at the end of the run is lower. This confirms our findings in [19] that the best heuristic selection mechanism that works with the acceptance scheme that accepts all moves[1] is GR. This is also supported by the fact that, in the table the results for LF and MF cases produce very low offline error values, but the values become high for HF. In the experiments for DUF1 and DUF2, 50 generations, 25 generations and 5 were used for LF, MF and HF changes. For the HF case, the environment changes every 5 iterations, which is lower than the number of heuristics ($M = 8$). So for some environments the corresponding heuristic may not get applied.

Based on the offline errors achieved at the end of a run, averaged over 100 runs, we also counted the overall significance states ($s+$, $s-$, $\geq$ and $\leq$) as before, on the three DUFs for different change severity and frequency settings in randomly changing environments and for different cycle length and change frequency settings in cyclic environments, for the different hyper-heuristics incorporating different heuristic selection schemes. The results are given in Table VI. The results show that, overall, the hyper-heuristic, in which *Random Permutation* is used as the heuristic selection mechanism is statistically significantly better than the others in a much higher number of cases.

Table VI: Overall ($s+$, $s-$, $\geq$ and $\leq$) counts for the different hyper-heuristics incorporating different heuristic selection schemes.

Algorithms	$s+$	$s-$	$\geq$	$\leq$
RP	113	20	65	54
RPD	78	29	82	63
SR	57	55	59	81
RD	30	81	69	72
RL	18	111	59	64

IV. Conclusion and Future Work

In this study we explored the effects of heuristic selection methods to determine the best one for the HH-PBIL2 framework which was proposed earlier in [20]. We performed an analysis of our approach under different parameter settings first, to achieve this goal. We looked at the effects of the number of low-level

[1]Our proposed approach does not use any acceptance scheme within the HH framework. This is equivalent to accepting all moves.

Table IV: Offline errors generated by each heuristic selection methods averaged over 100 runs, on the three DUFs for different change severity and frequency settings in randomly changing environments.

Alg.	LF				MF				HF			
	LS	MS	HS	VHS	LS	MS	HS	VHS	LS	MS	HS	VHS
DUF1												
SR-P2	**0.06**	**0.06**	**0.08**	0.09	**0.17**	**0.25**	**0.85**	**0.98**	21.98	23.64	**26.78**	28.29
RD-P2	**0.06**	**0.06**	**0.08**	0.09	**0.17**	**0.25**	0.89	1.05	21.97	23.63	26.83	28.42
RP-P2	**0.06**	**0.06**	**0.08**	**0.08**	**0.17**	0.26	0.86	0.99	21.99	23.61	**26.78**	**28.25**
RPD-P2	**0.06**	**0.06**	**0.08**	0.09	**0.17**	0.26	0.88	1.05	21.96	**23.60**	26.80	28.38
RL-P2	**0.06**	**0.06**	**0.08**	0.09	**0.17**	0.26	0.89	1.04	**21.93**	23.61	26.85	28.42
DUF2												
SR-P2	**0.12**	0.16	0.52	0.56	0.43	0.85	4.13	4.60	42.89	45.81	50.93	52.98
RD-P2	**0.12**	**0.15**	0.53	0.62	**0.42**	0.85	4.33	4.91	42.98	45.79	50.93	53.13
RP-P2	**0.12**	0.16	**0.49**	**0.52**	0.43	**0.84**	**4.11**	**4.57**	42.89	45.86	50.90	**52.97**
RPD-P2	**0.12**	**0.15**	0.53	0.61	**0.42**	0.85	4.36	4.97	**42.84**	45.79	**50.89**	53.10
RL-P2	**0.12**	0.16	0.55	0.61	**0.42**	0.85	4.36	4.99	42.98	**45.78**	51.03	53.23
DUF3												
SR-P2	19.38	18.38	16.07	14.18	19.86	18.98	17.27	15.50	38.30	39.86	41.06	40.50
RD-P2	19.26	18.35	16.05	14.16	19.70	**18.84**	**17.24**	15.52	38.29	39.64	40.66	40.23
RP-P2	19.39	18.35	16.07	14.21	19.80	18.96	17.29	**15.48**	38.42	39.97	41.38	40.68
RPD-P2	19.33	**18.27**	16.03	14.19	19.73	18.99	17.25	15.56	38.40	39.77	40.79	40.33
RL-P2	**19.25**	18.30	**16.00**	**14.15**	**19.58**	18.89	17.28	15.53	**38.16**	**39.44**	**40.54**	**40.01**

Table V: Offline errors generated by each heuristic selection methods averaged over 100 runs, on the three DUFs for different cycle length and change frequency settings in cyclic environments.

Alg.	LF			MF			HF		
	CL=2	CL=4	CL=8	CL=2	CL=4	CL=8	CL=2	CL=4	CL=8
DUF1									
SR-P2	0.03	0.03	0.03	0.10	0.08	0.09	15.09	15.00	15.29
RD-P2	0.04	0.04	0.04	0.15	0.12	0.14	15.72	15.57	15.85
RP-P2	**0.02**	**0.02**	**0.02**	**0.05**	**0.05**	**0.04**	**14.25**	**13.83**	**13.87**
RPD-P2	0.03	0.03	0.03	0.06	0.06	0.06	14.76	14.71	14.40
RL-P2	0.03	0.04	0.03	0.21	0.16	0.19	16.09	17.17	16.85
DUF2									
SR-P2	0.06	0.06	0.05	0.23	0.20	0.24	28.93	29.05	29.59
RD-P2	0.07	0.07	0.07	0.42	0.38	0.42	29.19	29.52	29.05
RP-P2	**0.04**	**0.04**	**0.04**	**0.09**	**0.08**	**0.09**	**27.37**	**27.02**	27.24
RPD-P2	0.06	0.06	0.05	0.11	0.11	0.12	27.75	28.61	**25.59**
RL-P2	0.07	0.07	0.07	0.61	0.52	0.57	29.90	32.44	31.29
DUF3									
SR-P2	10.16	11.36	11.35	11.09	12.13	12.15	24.14	24.25	24.40
RD-P2	10.21	11.37	11.37	11.26	12.14	12.17	23.53	24.36	23.93
RP-P2	**10.09**	11.36	11.33	**10.35**	11.60	11.58	22.74	22.59	23.05
RPD-P2	10.11	**11.33**	**11.32**	10.36	**11.50**	**11.49**	**21.23**	**22.22**	**21.42**
RL-P2	10.43	11.51	11.49	12.01	12.98	12.82	24.03	28.84	26.96

heuristics (number of probability vectors), the mutation rate and the learning rate. The results showed that the approach is not very sensitive to the mutation and learning rate. The choice of the number of low-level heuristics also isn't very sensitive, however, the best performance is achieved when the change period, i.e. the number of iterations between the changes, is greater than or equal to the number of low-level heuristics.

Then we introduced the resetting of probability vectors into the hybrid algorithm. We experimented with several versions where different resetting schemes are used when the environment changes. The tests show that this mechanism is important for the approach to be successful in all types of dynamic environments we experimented with. This suggests that diversity is one of the key factors for an improved performance. However, since SPBIL and SPBIL2 that use the same resetting mechanisms are not the winners in our experiments considering their overall performances, it is observed that the hybridization with a hyper-heuristic approach is beneficial.

Even though the hybrid method provides good performance overall, it generates an outstanding performance particularly in cyclic environments. This is somewhat expected, since the hybridization technique acts similar to a memory scheme, which is already known to be successful in cyclic dynamic environments

[12]. Using the results of the previous experiments, our final set of tests focused on the effect of the heuristic selection methods.The results revealed that, contrary to our initial intuition, the heuristic selection mechanism with learning isn't the most successful one for the HH-PBIL2 framework. The selection scheme that relies on a fixed permutation of the underlying low-level heuristics (RP) is the most successful one. For the cases when the change period is long enough to allow all the vectors in the permutation to be applied at least once, the RP heuristic selection mechanism becomes equivalent to *Greedy Selection*. In HH-PBIL2, the move acceptance stage of a hyper-heuristic is not used. This is the same as using the *Accept All Moves* strategy. This move acceptance scheme is known to perform best with the *Greedy Selection* method [19].

As future work, we will experiment with other types of more complex EDA based methods within the HH-PBIL2 framework. We will also verify our findings in a real-world problem domain, for example the dynamic vehicle routing problem.

Acknowledgments

This work is supported in part by the EPSRC, grant EP/F033214/1 (The LANCS Initiative Postdoctoral Training Scheme) and Berna Kiraz is supported by the TÜBİTAK 2211-National Scholarship Programme for PhD students.

References

[1] J. Branke, *Evolutionary optimization in dynamic environments*. Kluwer, 2002.

[2] C. Cruz, J. Gonzalez, and D. Pelta, "Optimization in dynamic environments: a survey on problems, methods and measures," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 15, pp. 1427–1448, 2011.

[3] S. Yang, Y.-S. Ong, and Y. Jin, Eds., *Evolutionary Computation in Dynamic and Uncertain Environments*, ser. Studies in Computational Int. Springer, 2007, vol. 51.

[4] A. Ghosh and H. Muehlenbein, "Univariate marginal distribution algorithms for non-stationary optimization problems," *Int. J. Know.-Based Intell. Eng. Syst.*, vol. 8, no. 3, pp. 129–138, 2004.

[5] M. Kobliha, J. Schwarz, and J. Ocenek, "Bayesian optimization algorithms for dynamic problems," in *EvoWorkshops*, ser. Lecture Notes in Computer Science, vol. 3907. Springer, 2006, pp. 800–804.

[6] S. Yang and X. Yao, "Experimental study on population-based incremental learning algorithms for dynamic optimization problems," *Soft Comput.*, vol. 9, no. 11, pp. 815–834, Nov. 2005.

[7] G. J. Barlow and S. F. Smith, "Using memory models to improve adaptive efficiency in dynamic problems," in *IEEE Symposium on Computational Intelligence in Scheduling, , CISCHED*, 2009, p. 714.

[8] C. M. Fernandes, C. Lima, and A. C. Rosa, "Umdas for dynamic optimization problems," in *Proc. of the 10th conference on genetic and evolutionary computation*, ser. GECCO '08. ACM, 2008, pp. 399–406.

[9] Y. Wu, Y. Wang, X. Liu, and J. Ye, "Multi-population and diffusion umda for dynamic multimodal problems," *Journal of Systems Engineering and Electronics*, vol. 21, no. 5, pp. 777–783, 2010.

[10] S. Yang and H. Richter, "Hyper-learning for population-based incremental learning in dynamic environments," in *in Proc. 2009 Congr. Evol. Comput*, 2009, pp. 682–689.

[11] X. Peng, X. Gao, and S. Yang, "Environment identification-based memory scheme for estimation of distribution algorithms in dynamic environments," *Soft Comput.*, vol. 15, pp. 311–326, 2011.

[12] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *IEEE Trans. on Evolutionary Comp.*, vol. 12, pp. 542–561, 2008.

[13] B. Yuan, M. E. Orlowska, and S. W. Sadiq, "Extending a class of continuous estimation of distribution algorithms to dynamic problems," *Optimization Letters*, vol. 2, no. 3, pp. 433–443, 2008.

[14] E. K. Burke, M. Gendreau, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: A survey of the state of the art," *to appear in the Journal of the Operational Research Society*, 2012.

[15] P. I. Cowling, G. Kendall, and E. Soubeiga, "A hyper-heuristic approach to scheduling a sales summit," in *Practice and Theory of Automated Timetabling III : 3rd Int. Conference, PATAT 2000*, ser. LNCS, vol. 2079. Springer, 2000.

[16] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, pp. 3–23, 2008.

[17] E. Özcan, Şima Uyar, and E. Burke, "A greedy hyper-heuristic in dynamic environments," in *GECCO 2009 Workshop on Automated Heuristic Design: Crossing the Chasm for Search Methods*, 2009, pp. 2201–2204.

[18] B. Kiraz and H. R. Topcuoglu, "Hyper-heuristic approaches for the dynamic generalized assignment problem," in *Intelligent Systems Design and Applications (ISDA), 2010 10th International Conference on*, 2010, pp. 1487–1492.

[19] B. Kiraz, Şima Uyar, and E. Özcan, "An investigation of selection hyper-heuristics in dynamic environments," in *Proc. of EvoApplications 2011*, ser. LNCS, vol. 6624. Springer, 2011.

[20] G. Uludağ, B. Kiraz, A. Şima Etaner Uyar, and E. Özcan, "A framework to hybridise PBIL and a hyper-heuristic for dynamic environments," in *Parallel Problem Solving from Nature - PPSN*. Under review, 2012.

[21] S. Yang, "Constructing dynamic test environments for genetic algorithms based on problem difficulty," in *In Proc. of the 2004 Congress on Evolutionary Computation*, 2004, pp. 1262–1269.

[22] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning - great-deluge hyper-heuristic for examination timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, 2010.

[23] G. Ochoa, M. R. Hyde, T. Curtois, J. A. V. Rodríguez, J. D. Walker, M. Gendreau, G. Kendall, B. McCollum, A. J. Parkes, S. Petrovic, and E. K. Burke, "Hyflex: A benchmark framework for cross-domain heuristic search," in *EvoCOP*, 2012, pp. 136–147.