Selection Hyper-heuristics in Dynamic Environments

Berna Kiraz
Institute of Science and Technology, Istanbul Technical University, Turkey
A. Şima Uyar
Department of Computer Engineering, Istanbul Technical University, Turkey
Ender Özcan
School of Computer Science, University of Nottingham, UK

Abstract

Current state-of-the-art methodologies are mostly developed for stationary optimization problems. However, many real world problems are dynamic in nature, where different types of changes may occur over time. Population based approaches, such as evolutionary algorithms are frequently used in dynamic environments. Selection hyper-heuristics are highly adaptive search methodologies that aim to raise the level of generality by providing solutions to a diverse set of problems having different characteristics. In this study, thirty-five single point search based selection hyper-heuristics are investigated on continuous dynamic environments exhibiting various change dynamics, generated using the Moving Peaks Benchmark generator. Even though there are many successful applications of selection hyper-heuristics to discrete optimization problems, to the best of our knowledge, this study is one of the initial applications of selection hyper-heuristics for real-valued optimization as well as being among the very few which address dynamic optimization issues with these techniques. The empirical results indicate that selection hyper-heuristics with compatible components can react to different types of changes in the environment and are capable of tracking them. This shows the suitability of selection hyper-heuristics as solvers in dynamic environments.

Keywords: heuristics, meta-heuristics, hyper-heuristics, dynamic environments, moving peaks benchmark, decision support

1 Introduction

A *hyper-heuristic* is a methodology which explores the space of heuristics for solving complex computational problems [8, 40, 13, 11]. Although the term *hyper-heuristic* is introduced recently [21, 15], the initial ideas can be traced back to the 60s [18, 23]. There has been a growing interest in this field since then. There are two main types of hyper-heuristics in literature [10]: methodologies

that *select*, or *generate* heuristics. This study focuses on the former type of hyper-heuristics based on a single point search framework termed as a selection hyper-heuristic. A selection hyper-heuristic controls a set of low level heuristics and adaptively chooses the most appropriate heuristic to invoke at each step. This type of hyper-heuristics have been successfully applied to many combinatorial optimization problems ranging from timetabling to vehicle routing [9]. In this paper, from this point on we will use *hyper-heuristics* to denote *selection hyper-heuristics*.

Real world optimization problems are mostly dynamic in nature. To handle the complexity of dealing with the changes in the environment, an optimization algorithm needs to be adaptive and hence capable of following the change dynamics. From the point of view of an optimization algorithm, the problem environment consists of the instance, the objectives and the constraints. The dynamism may arise due to a change in any of the components of the problem environment. Existing search methodologies have been modified suitably with respect to the change properties to be dealt with in order to tackle dynamic environment problems. A key goal in hyper-heuristic research is raising the level of generality. To this end, approaches which generalize well and are applicable across a wide range of problem domains or different problems with different characteristics, have been investigated. Considering the adaptive nature of hyper-heuristics, they are expected to respond to the changes in a dynamic environment rapidly and hence be effective solvers in such environments regardless of the change properties. We conducted two preliminary studies on the applicability of hyper-heuristics in dynamic environments in [38, 31]. This study extends our previous work in [31] and further explores the performance of a set of hyper-heuristics in dynamic environments with more realistic change scenarios.

Hyper-heuristics have been mostly applied to discrete combinatorial optimization problems in literature [9]. In this study, they are applied to a set of real-valued optimization problems generated using the Moving Peaks Benchmark (MPB) generator. This benchmark generator is preferred as a testbed for our investigations mainly because it is one of the most commonly used benchmark generators in literature for creating dynamic optimization environments in the continuous domain [19].

The remainder of this paper is organized as follows: next section provides background information on hyper-heuristics and dynamic environments as well as a brief literature survey on the topic; section 3 gives the experimental design and results of the computational experiments, while Section 4 concludes the paper.

2 Background

2.1 Selection Hyper-heuristics

In a hyper-heuristic framework, an initial candidate solution is iteratively improved through two successive stages: *heuristic selection* and *move acceptance* [37]. Almost all such hyper-heuristics in literature perform a single point based search [9]. In the first stage, a heuristic is selected from a fixed set of low level perturbative heuristics and applied to the current candidate solution, generating a new one. The heuristic selection method does not use any problem domain specific knowledge while making this decision. Then, the new solution is either accepted or rejected based on an acceptance method. This process is repeated until the termination criteria are satisfied, after which, the best solution is returned. In the rest of the paper, a hyper-heuristic will be denoted as a *Heuristic Selection Method – Move Acceptance Method* pair.

Cowling et al. [15] defined hyper-heuristics as "heuristics to choose heuristics" and investigated the performance of different heuristic selection methods on a real world scheduling problem. These methods included *Simple Random* (SR), *Random Descent* (RD), *Random Permutation* (RP), *Random Permutation Descent* (RPD), *Greedy* (GR) and a more elaborate learning heuristic selection method, namely *Choice Function* (CF). Simple Random selects a low level heuristic randomly. Random Descent applies a randomly selected heuristic to the current solution repeatedly as long as the solution improves, then another heuristic is selected randomly. Random Permutation randomly orders all low level heuristics and applies each heuristic successively in turns. Random Permutation Descent selects a heuristic in the same way as Random Permutation, but it applies the selected heuristic repeatedly as long as the solution improves. Greedy applies all low level heuristics to the current solution and selects the one which generates the largest improvement.

Choice Function maintains a utility score for each low level heuristic H_i (Equation 1), measuring how well it has performed individually ($u_1(H_i)$ in Equation 2) and as a successor of the previously selected heuristic ($u_2(H_i, H_{\text{selected}})$ in Equation 3), and the elapsed time since its last call ($u_3(H_i)$ in Equation 4). The heuristic with the maximum score is selected at each iteration (H_{selected}). The score of each heuristic denoted as $score(H_i)$ gets updated after the heuristic selection process. Given that $\Delta f_n(y)$ ($\Delta f_n(x, y)$) denotes the change in the solution quality and $Time_n(y)$ ($Time_n(x, y)$) denotes time spent, when the n^{th} last time heuristic y was selected and

applied to the current solution (before the application of heuristic x):

$$\forall i,\ score(H_i) = \alpha u_1(H_i) + \beta u_2(H_i, H_{\text{selected}}) + \delta u_3(H_i) \quad (1)$$

$$\forall i,\ u_1(H_i) = \sum_n \alpha^{n-1} \frac{\Delta f_n(H_i)}{Time_n(H_i)} \quad (2)$$

$$\forall i,\ u_2(H_i, H_{\text{selected}}) = \sum_n \beta^{n-1} \frac{\Delta f_n(H_i, H_{\text{selected}})}{Time_n(H_i, H_{\text{selected}})} \quad (3)$$

$$\forall i,\ u_3(H_i) = elapsedTime(H_i) \quad (4)$$

Cowling et al. [16] provide a mechanism showing how the parameters α, $\beta \in (0, 1]$ and δ can be adjusted dynamically. In [15, 17], the authors combined all the above heuristic selection methods with the following move acceptance methods: *All Moves* (AM) accepted, *Only Improving* (OI) moves accepted and *Improving and Equal* (IE) moves accepted. The computational experiments resulted with the success of the Choice Function–All Moves hyper-heuristic.

Nareyek [36] applied *Reinforcement Learning* (RL) heuristic selection to Orc Quest and modified logistics domain problems. Reinforcement Learning maintains a utility score (weight) for each low level heuristic. Initially, all scores are the same for all heuristics, e.g., 0. If the selected heuristic improves the solution, its score is increased; otherwise it is decreased, e.g. by one. The scores are restricted to vary between certain lower and upper bounds. The author investigated different negative and positive adaptation strategies as well as heuristic selection methods based on the scores. *All Moves* was the acceptance method used in this study. The results showed that high negative and low positive adaptation rates are preferable. Moreover, the max strategy which selects a heuristic with the maximum score performs better than the softmax (roulette wheel) strategy which chooses a low level heuristic (H_i) randomly with a probability of $p(H_i) = score(H_i) / \sum_{\forall j} score(H_j)$.

Apart from the simple acceptance mechanisms, there are other more sophisticated ones. Improving moves are always accepted regardless of the nature of an acceptance mechanism. Kendall and Mohamad [30] applied a *Great Deluge* move acceptance based hyper-heuristic to a mobile telecommunications network problem. Great Deluge accepts a worsening move, if it is better than a dynamically changing threshold value which depends on the current step and overall duration of the experiment. Linearly decreasing the threshold value at each step is a common practice as illustrated in Equation 5 (for a minimization problem) to determine an acceptance range for a

worsening solution.

$$threshold_t = f_{final} + \Delta F \cdot (1 - \frac{t}{maxIterations}) \tag{5}$$

Here f_{final} is the expected final objective value, $maxIterations$ is the maximum number of steps (or total time), t denotes the current step (time), ΔF is an expected range for the maximum solution quality (fitness/cost) change.

Ayob and Kendall [1] proposed a set of Monte Carlo move acceptance methods inspired from the well known simulated annealing meta-heuristic. The results showed that Simple Random heuristic selection combined with *Exponential Monte Carlo With Counter* move acceptance (EMCQ) performs well. EMCQ accepts a worsening move with a probability given in Equation 6,

$$e^{-\frac{\Delta f \cdot m}{Q}}, \tag{6}$$

where Q is a counter for successive worsening moves and m is the unit time in minutes that measures the duration of the heuristic execution, Δf is the difference in the quality between new and current solutions. Q is reset if the quality of the solution improves, otherwise it is incremented. m is incremented at every B steps.

Bai et al. [3] showed that *Simulated Annealing* (SA) as a move acceptance was promising. Bilgin et al. [5] compared the performances of many heuristic selection and move acceptance combinations in hyper-heuristics. The results show that a standard simulated annealing move acceptance performs the best, especially combined with Choice Function. Simulated Annealing accepts all improving moves and a worsening move with a probability given in Equation 7.

$$e^{-\frac{\Delta f}{\Delta F(1-\frac{t}{maxIterations})}}, \tag{7}$$

Bai et al. [2] investigated the performance of a Reinforcement Learning – Simulated Annealing with Reheating (SA+RH) hyper-heuristic on nurse rostering, university course timetabling and one-dimensional bin packing problems. The formula $e^{-\frac{\Delta f}{T}}$ is used while deciding whether or not to accept a worsening move. The temperature (T) is reduced using the nonlinear formula, $T = \frac{T}{1+\gamma T}$ [34], where

$$\gamma = \frac{(t_0 - t_{final}) iter_{temp}}{maxIterations \cdot t_0 \cdot t_{final}} \tag{8}$$

Here, $iter_{temp}$ is the number of iterations at a temperature. During the reheating phase, the temperature is increased using the formula $T = \frac{T}{1-\gamma T}$ and the system reenters the annealing phase. This hyper-heuristic generates a better performance when compared to the other meta-heuristic solutions in each problem domain. The same acceptance was also used by Dowsland et al. [22] as a part of a hyper-heuristic which hybridized Tabu Search with Reinforcement Learning as a heuristic selection method. This hyper-heuristic performed well on a shipper rationalization problem.

Burke et al. [12] compared the performance of different Monte Carlo move acceptance methods over a set of benchmark examination timetabling problems. EMCQ as a move acceptance method delivered a poor performance as compared to Simulated Annealing based methods. SA+RH turned out to be very promising as a move acceptance component of a hyper-heuristic. Özcan et al. [39] experimented with Great Deluge based hyper-heuristics on examination timetabling. It was observed that Reinforcement Learning–Great Deluge delivers a promising performance, when an additive/subtractive adaptation rate is used for rewarding/punishing. Similarly, Gibbs et al. [24] reported the success of Reinforcement Learning–Great Deluge and Reinforcement Learning–Simulated Annealing for solving sports scheduling problems. More on hyper-heuristics can be found in [8, 9, 13, 40].

2.2 Dynamic Environments

A dynamic environment is made up of components, such as, the problem instance, the objectives and the constraints, each of which may change in time individually or simultaneously. A change in a component can be categorized based on its characteristics as given in [7]: (i) *Frequency of change* defines how often the environment changes. (ii) *Severity of change* defines the magnitude of the change in the environment. (iii) *Predictability of change* is a measure of correlation between changes. (iv) *Cycle length/cycle accuracy* is a property that defines whether the optima return exactly to previous locations or close to them.

When designing an optimization algorithm for dynamic environments, one of the main issues for the algorithm to deal with is tracking the moving optima as closely as possible after a change occurs. Another one is being able to react to a change in the environment quickly and adapting to the new environment as fast as possible. Several strategies have been proposed to be used as a part of existing search methodologies for dynamic environments depending on the change properties. These strategies can be grouped into four main categories [29]: (i) maintain diversity at all times,

(ii) increase diversity after a change , (iii) use memory, (iv) work with multiple populations.

For the approaches which maintain diversity at all times, e.g., as in the *random immigrants* approach [25], achieving and preserving the right level of diversity is crucial. These approaches are generally more successful in environments where the changes are severe and the change frequency is relatively high. Approaches, such as *hypermutation* [14] and *variable local search* [42] increase diversity by increasing the mutation rate when the environment changes. It has been observed that too much diversity disrupts the search process, while too little may not be sufficient to prevent premature convergence. These approaches are more suitable for environments where changes are not too severe.

Some approaches make use of memory, as in [33, 20, 45, 43], where the evolutionary algorithm remembers solutions which have been successful in the previous environments. These approaches are particularly more useful if a change occurs periodically and a previous environment is re-encountered during the search process at a later stage. There are also other approaches with a good performance in dynamic environments, which make use of multiple populations, such as [7, 41]. In these approaches, the population is divided into subpopulations, where each subpopulation explores a different part of the search space. Often, the focus of such an algorithm is tracking several optima simultaneously in different regions of the search space. Further details about dynamic environments can be found in [7, 19, 35, 44].

2.2.1 The Moving Peaks Benchmark

The *Moving Peaks Benchmark* (MPB) generator introduced by Branke [6], is used in this study for analyzing and comparing the performance of different approaches. MPB is a dynamic benchmark function generator which is not as simplified as most of the toy problems in literature. Moreover, MPB exhibits similar properties to real world problems.

The MPB generator provides multidimensional and multimodal landscapes with a variety of different peak shapes. In MPB, the most commonly used peak shape is the cone. The height, width and the location of each peak is altered whenever a change in the environment occurs. A dynamic benchmark function generated using MPB with cone shaped peaks is formulated as follows:

$$F(\vec{x},t) \quad = \quad \max_{i=1..m} \{H_i(t) - W_i(t) * \sqrt{\sum_{j=1}^{d} (x_j - X_{ij}(t))^2}\} \tag{9}$$

where m is the number of peaks, d is the number of dimensions, X_{ij} are the coordinates of the peaks in each dimension, H_i and W_i are the heights and widths of the peaks respectively. For example, assume that the current peak coordinates, height and width values of two peaks in a 2-dimensional landscape at the given time t_c are as given in Table 1. The function value of a real-valued vector (candidate solution) located at $\vec{x} = (x_1, x_2) = (10.0, 3.0)$ is calculated as follows:

$$\begin{aligned} F((10.0, 3.0), t_c) &= \max\{50.0 - 0.1 * \sqrt{((10.0 - 2.0)^2 + (3.0 - 2.0)^2)}, \\ &\qquad 70.0 - 0.5 * \sqrt{((10.0 - 20.0)^2 + (3.0 - 20.0)^2)}\} \\ F((10.0, 3.0), t_c) &= \max\{49.19, 60.14\} \\ F((10.0, 3.0), t_c) &= 60.14 \end{aligned}$$

Table 1: Example peak coordinate, height and width values of a 2-dimensional landscape with two peaks

Peak i	$X_{i1}(t_c)$	$X_{i2}(t_c)$	$W_i(t_c)$	$H_i(t_c)$
1	2.0	2.0	0.1	50.0
2	20.0	20.0	0.5	70.0

In some applications, a time-invariant base function $B(\vec{x})$ is used as part of the benchmark function. In this case, the new MPB function, denoted as $G(\vec{x}, t)$ becomes $G(\vec{x}, t) = \max\{B(\vec{x}), F(\vec{x}, t)\}$.

When working with the MPB, firstly, the coordinates, heights and widths of the peaks are initialized. Then, every Δe iterations, the heights and the widths of the peaks are changed by adding a normally distributed random variable, while the location of the peaks are also shifted by a vector $\vec{v}$ of fixed length $vlength$ in a random direction. During the search, the height, width and location of each peak are changed according to the following equations:

$$\rho \in N(\mu, \sigma^2) \tag{10}$$

$$H_i(t) = H_i(t-1) + height\_severity \cdot \rho \tag{11}$$

$$W_i(t) = W_i(t-1) + width\_severity \cdot \rho \tag{12}$$

$$\vec{X}_i(t) = \vec{X}_i(t-1) + \vec{v_i}(t) \tag{13}$$

where ρ is a random value drawn from a Gaussian distribution $N(\mu, \sigma^2)$, where μ and σ^2 denote its mean and variance set to 0 and 1, respectively and $v_i(t)$ is the shift vector which is the linear combination of the previous shift vector $v_i(t-1)$ and a random vector $\vec{r}$ normalized to $vlength$.

The *height_severity*, the *width_severity* and *vlength* parameters determine the severity of the change in the heights, widths and locations of the peaks respectively. Δe determines the frequency of changes in the environment. The shift vector at time t is calculated as:

$$\vec{v_i}(t) = \frac{vlength}{|\vec{r} + \vec{v_i}(t-1)|}((1-\phi)\vec{r} + \phi\vec{v_i}(t-1)) \quad (14)$$

where the random vector $\vec{r}$ is created by drawing uniformly distributed random numbers for each dimension and normalizing its length to *vlength*, and ϕ is the *correlation coefficient*. The higher values of ϕ indicates a higher correlation between the current and previous shift vectors.

Figure 1 gives an example of an initial fitness landscape on which various types of changes are applied. The fitness landscapes in the figure are generated using MPB with a basis function of $B(\vec{x}) = 0$. Figure 1(a) shows the initial 2-dimensional fitness landscape with 2 peaks ($m = 2$). Each of the rest of the sub-figures shows a specific type of change applied on this initial fitness landscape.

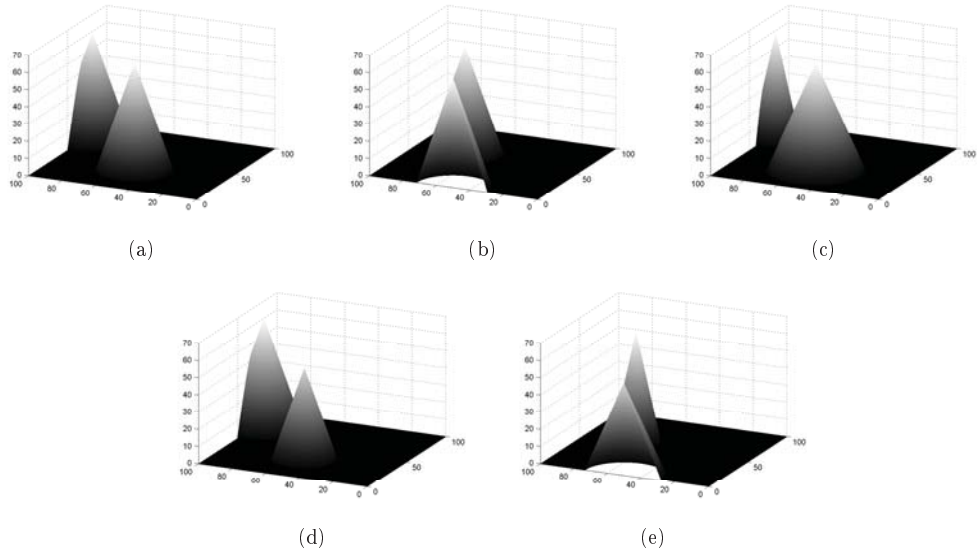


Figure 1: A 2-dimensional fitness-landscape with two peaks is given in (a). The following changes are applied on this landscape: (b) the peaks are shifted, i.e. their locations are changed, but their heights and widths remain fixed, (c) the widths of the peaks are changed, but their locations and heights remain fixed, (d) the heights of the peaks are changed, but their locations and widths remain fixed, (e) the heights, widths and locations of the peaks are changed.

An initial landscape with five peaks is generated to demonstrate the effect of the changes on the landscape further. 20 consecutive changes are applied to this initial landscape. For simplicity, only the heights of the peaks are modified as a change, but their locations and widths are fixed.

Figure 2 gives the height of each peak including the optimum after each change.

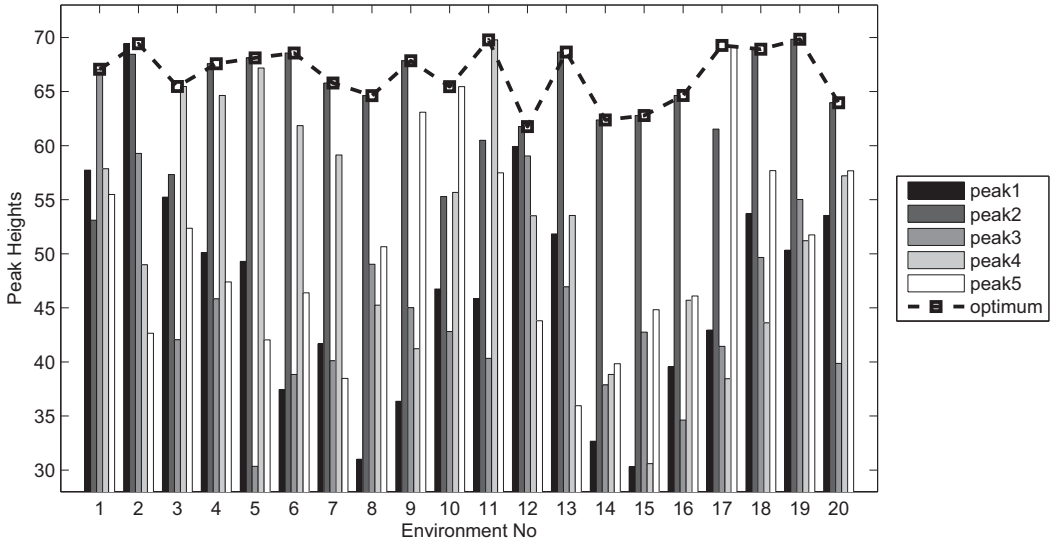


Figure 2: The heights of all the peaks given for each stationary environment over 20 changes.

2.3 Selection Hyper-heuristics in Dynamic Environments

Özcan et al. [38] is the first study which proposed a hyper-heuristic for solving dynamic environment problems to the best of our knowledge. The authors applied a Greedy hyper-heuristic to five well known benchmark functions. The Greedy heuristic selection method was chosen as a hyper-heuristic component with the hope that it would respond to the changes in the environment quickly. The results indeed showed that this selection hyper-heuristic is capable of adapting itself to the changes.

In [31], the authors compared the performance of different heuristic selection mechanisms within the selection hyper-heuristic framework. The hyper-heuristics combined the Improving and Equal acceptance with five heuristic selection methods controlling a set of mutational low level heuristics in a very simple dynamic environment. The landscape was only allowed to shift in this environment, and its general features remained the same. The Moving Peaks Benchmark was used during the experiments. Choice Function–Improving and Equal delivered the best average performance.

Kiraz and Topcuoglu [32] proposed a population based search framework embedding a variety of hyper-heuristics which combine {Simple Random, Random Descent, Random Permutation,

Random Permutation Descent, Choice Function} with {All Moves, Only Improving}. The behavior of these hyper-heuristics is investigated over a set of dynamic generalized assignment problem instances. The authors used an evolutionary algorithm operating on two subpopulations: *search* and *memory*. The individuals in the search subpopulation are perturbed using a heuristic selected by a hyper-heuristic and the other one is evolved using a standard evolutionary algorithm updating the memory periodically. The results showed that the Random Permutation Descent–All Moves and Choice Function–All Moves hyper-heuristics performed well in general.

There is already empirical evidence showing that different combinations of hyper-heuristic components yield different performances [5, 37] for solving discrete optimization problems. This study extends our previous work in [31] further and provides a complete empirical analysis of different hyper-heuristics coupling well known heuristic selection and move acceptance methods in dynamic environments. There is no previous study investigating a single point based search hyper-heuristic framework for solving dynamic environment problems and moreover, to the best of our knowledge, this is one of the first studies which investigates the application of hyper-heuristics to a real-valued optimization problem.

3 Computational Experiments

In this study, we explore the performance of a set of hyper-heuristics in dynamic environments exhibiting different change characteristics, which are generated using the MPB generator. The experiments consist of four parts. In the first part, a simple dynamic environment scenario is investigated, where only the locations of the peaks are changed but their heights and widths remain the same. We will refer to these set of experiments as EXPSET1. In the second part, denoted as EXPSET2, the approaches are compared in environments of different change frequencies and change severities, where peak locations as well as peak heights and widths are changed. In the third part, we explore the tracking ability of the approaches. In the last part their scalability is investigated through experiments where the number of peaks and the number of dimensions are increased.

3.1 Experimental Design

We experiment with thirty five hyper-heuristics composed of five heuristic selection methods {Simple Random, Greedy, Choice Function, Reinforcement Learning, Random Permutation Descent}

combined with seven move acceptance methods {All Moves, Only Improving, Improving and Equal, Exponential Monte Carlo with Counter, Great Deluge, Simulated Annealing, Simulated Annealing with Reheating}. All these hyper-heuristic components have different properties. Simple Random uses no feedback. Greedy selects the best solution at each step. Choice Function and Reinforcement Learning incorporate an online learning mechanism. Random Permutation Descent makes a random choice, but converts the framework into a hill climber, since the same heuristic is invoked repetitively as long as the solution improves. Great Deluge, Exponential Monte Carlo with Counter, Simulated Annealing and Simulated Annealing with Reheating are non-deterministic acceptance methods for which the acceptance decision depends on a given step. On the other hand, All Moves, Only Improving, Improving and Equal acceptance methods are deterministic.

The hyper-heuristics used in this study are applied to a set of real-valued dynamic function optimization instances produced by the Moving Peaks Benchmark (MPB) generator. A candidate solution is a real-valued vector representing the coordinates of a point in the multidimensional search space for a given instance, for which the length of the vector is the number of dimensions. In order to perturb a given candidate solution, a parameterized Gaussian mutation, $N(0, \sigma^2)$, where σ denotes the standard deviation, is implemented. Seven mutation operators based on seven different standard deviations; {0.5, 2, 7, 15, 20, 25, 30} are used as low-level heuristics within the hyper-heuristic framework during the experiments. A low level heuristic draws a random value from the relevant Gaussian distribution for each dimension separately and this random value is added to the corresponding dimension of a candidate solution to generate a new one.

3.1.1 Approaches Used in Comparisons

The performances of different hyper-heuristics are compared to well known techniques from literature including a Hypermutation [14] based approach (HM), $(1,\lambda)$-Evolutionary Strategies (ES) [4] and (μ,λ)-Covariance Matrix Adaptation Evolution Strategy (CMAES) [26, 28, 27]. These techniques are chosen since they are well known approaches to real-valued optimization and all use a different mutation adaptation scheme to deal with the dynamics in the environment. Hypermutation adapts the mutation rate whenever the environment changes. ES adapts the mutation rate based on the success or failure of the ongoing search. In CMAES, adaptation is based on the adaptation of the covariance matrix. The parameter settings of HM, ES and CMAES are determined as a result of a series of preliminary experiments.

Hypermutation performs a Gaussian mutation with a fixed standard deviation of 2 during

the stationary periods. When a change occurs, the standard deviation is increased to 7 for 70 consecutive *fitness evaluations*. Afterwards, the standard deviation is reset to 2.

In (1,λ)-ES, λ *offspring* (new candidate solutions) are generated from one *parent* (current solution in hand) by a Gaussian mutation with zero mean and a standard deviation of σ. The initial value for σ is set to 2. Whenever the environment changes, σ is reset to this initial value. During the stationary period of the search, σ is adapted according to the classical 1/5 success rule [4] as shown in Equation 15 at every k iterations. If the percentage of successful mutations, denoted as p_s is greater than 1/5, σ is increased, otherwise it is decreased. After λ offspring are obtained, a solution is selected from them to replace the parent. The value of k is set to 7. This evolutionary process repeats until a maximum number of iterations is completed.

$$\sigma = \begin{cases} \sigma/c & \text{if } p_s > 1/5 \\ \sigma.c & \text{if } p_s < 1/5 \\ \sigma & \text{if } p_s = 1/5 \end{cases} \tag{15}$$

During the experiments, the value of the parameter c is set to $0.9 \in [0.85, 1)$ as suggested in [4].

CMAES is the state-of-the-art algorithm for global optimization. It is based on the adaptation of the covariance matrix. In CMAES, offspring at generation $g+1$ are generated by sampling the multivariate normal distribution [28], i.e. $k = 1, \ldots, \lambda$

$$x_k^{(g+1)} = \langle x \rangle_w^{(g)} + \sigma^{(g)} \sim N(0, C^{(g)}) \tag{16}$$

where $\langle x \rangle_w^{(g)}$ is the weighted mean of the μ best individuals at generation g, σ is the mutation step size, $C^{(g)}$ is the covariance matrix at generation g. The covariance matrix C is adapted via the evolution path. The step size σ is initialized to $\sigma = 0.3$ and is then updated using a cumulative step-size adaptation (CSA) approach, in which a conjugate evolution path is constructed [28]. Further details on CMAES can be found in [26, 28, 27].

The initial value of μ is set to 1 for CMAES for a fair comparison with the other single point search methods [28], while the value of λ for ES and CMAES is set to 7 for a fair comparison with the Greedy hyper-heuristic which makes 7 evaluations at each step.

3.1.2 Parameter Settings of Hyper-heuristics

Some of the heuristic selection and acceptance methods have parameters which require initial settings.

- In Reinforcement Learning, the initial scores of all heuristics are set to 15. Their lower and upper bounds are set to 0 and 30, respectively as suggested in [39]. If the current heuristic produces a better solution than the previous one, its score is increased by 1, otherwise it is decreased by 1.
- In Choice Function, α, β, and δ are set to 0.5 and updated by ± 0.01 at each iteration.
- In Exponential Monte Carlo with Counter, the value of B is set to 60, 10, 2 for LF, MF and HF changes, respectively.
- In Great Deluge, Simulated Annealing and Simulated Annealing with Reheating, the expected range is calculated as $\Delta F = initialError - optimumError$, where $optimumError = 0$. Also in Simulated Annealing with Reheating, the starting and final temperatures are set to $t_0 = -\Delta F/log(0.1)$ and $t_{final} = -\Delta F/log(0.005)$, respectively.

It is assumed that all programs are aware of the time when a change occurs during the experiments. As soon as the environment changes,

- the current solution is re-evaluated.
- the Exponential Monte Carlo with Counter parameters m and Q are reset to 1.
- the expected range(ΔF) is recalculated for Great Deluge and Simulated Annealing.
- the system enters the reheating phase for Simulated Annealing with Reheating.

On the other hand, the parameters of the heuristic selection methods Choice Function and Reinforcement Learning are not updated at all when the environment changes.

3.1.3 Experimental Settings

Each run is repeated 100 times for a given setting. Each problem instance contains 20 changes in a given environment, i.e. there are 21 consecutive stationary periods. The total number of iterations per run ($maxIterations$) is determined based on the change period as given in Equation 17,

$$maxIterations = (NoOfChanges + 1) * ChangePeriod \tag{17}$$

where there are $(NoOfChanges+1)$ stationary periods with a length of $ChangePeriod$, including the initial environment before the first change.

Table 2 lists the fixed parameters of the Moving Peaks Benchmark used during the experiments. These parameter settings are taken from [6, 7]. In the scalability experiments, dimension and peak counts are changed while the rest of the settings are kept the same.

Table 2: Parameter settings for the Moving Peaks Benchmark

Parameter	Setting	Parameter	Setting
Number of peaks p	5	Number of dimensions d	5
Peak heights	$\in [30, 70]$	Peak widths	$\in [0.8, 7.0]$
Peak function	cone	Basis function	not used
Range in each dimension	$\in [0.0, 100.0]$	Correlation coefficient ϕ	0

In this study, we experimented with combinations of two change characteristics, namely the frequency and the severity of the changes. We performed some initial experiments to determine the settings for various change frequencies and severities.

First, we utilized the Simple Random heuristic selection as a basis to determine change frequency settings. We allowed a Simple Random–Improving and Equal hyper-heuristic to run for long periods without any change in the environment. Based on the resultant convergence behavior, we determined the change periods[1] as 6006 fitness evaluations for low frequency (LF), 1001 for medium frequency (MF) and 126 for high frequency (HF). In the convergence plot, 6006 fitness evaluations correspond to a stage where the algorithm has been converged for some time, 1001 corresponds to a time where the approach has not yet fully converged and 126 is very early on in the search.

In MPB, the severity of the changes in the locations of the peaks, their heights and widths are controlled by three parameters, namely *shift length*, *height severity* and *width severity*, respectively. We determined low severity (LS), medium severity (MS) and high severity (HS) change settings based on the Moving Peaks Benchmark formulation given in Equation 9. The parameter settings used in the experiments for different levels of severity are provided in Table 3.

Table 3: MPB parameter settings for each severity level

Setting	LS	MS	HS
Shift length	1.0	5.0	10.0
Height severity	1.0	5.0	10.0
Width severity	0.1	0.5	1.0

[1]Since we have 7 low level heuristics and the Greedy heuristic selection method evaluates all at each step, these values are determined as multiples of 7 to give each method an equal number of evaluations during each stationary period.

3.1.4 Performance Evaluation

The performance of the approaches is compared based on the *offline error* [7] metric. The *error value* of a candidate solution $\vec{x}$ at time t represents its distance to the optimum in terms of the objective/functional value at a given time as given in Equation 18.

$$err(\vec{x}, t) = |optimum(t) - F(\vec{x}, t)| \tag{18}$$

Here $optimum(t)$ and $F(\vec{x}, t)$ are the function values of the global optimum solution and a given candidate solution $\vec{x}$ (see Equation 9) at time t, respectively (MPB provides the location and the function value of the current global optimum). The offline error is calculated as a cumulative average of $err(\vec{x}_b, t)^*$ which denote the error values of the best candidate solutions ($\vec{x}_b$) found so far since the last change until a given time t, as provided in Equation 20. An algorithm solving a dynamic environment problem aims to achieve the least overall offline error value obtained at the end of a run.

$$offline\_error = \frac{1}{T_{eval}} \sum_{t=1}^{T_{eval}} (err(\vec{x}_b, t)^*) \tag{19}$$

$$err(\vec{x}_b, t)^* = \min\{err(\vec{x}_b, \tau), err(\vec{x}_b, \tau + 1), \ldots, err(\vec{x}_b, t)\} \tag{20}$$

Here T_{eval} is the total number of evaluations, τ is the last time step ($\tau < t$) when change occurred, and x_b is the best solution found so far until the time step t since the last change at time τ.

3.2 Results and Discussion

All trials are repeated for 100 times using each approach for each test case. The results are provided in terms of average offline error values in the tables. The performances of the approaches are compared under a variety of change frequency-severity pair settings where each setting generates a different dynamic environment. In the result tables, the best performing approach is marked in bold. The comparisons based on One-way ANOVA and Tukey HSD tests at a 95% confidence level are performed to show whether the observed pairwise performance variations are statistically significant or not. We illustrate the tracking ability of the approaches as well as their scalability, only using EXPSET2 in this section, since we have observed the same behavior for EXPSET1 and EXPSET2.

3.2.1 Results for EXPSET1

Table 4 summarizes the results of EXPSET1 using MPB in which only the peak locations change in time. The performance of all methods degrades as the change frequency increases. Moreover, the offline error becomes particularly high when the change frequency is high. Performance also degrades for almost all methods as the severity of change increases. These observations are somewhat expected, based on the fact that the methods are provided with a very limited time to respond to the changes in the environment.

We performed statistical significance tests to determine the overall best heuristic selection and best move acceptance methods. Considering all hyper-heuristic runs where a different heuristic selection method is used, Improving and Equal acceptance consistently performs the best over all frequency-severity settings. However, when considering all hyper-heuristic runs where a different move acceptance method is used, there is more variation among the best performing heuristic selection methods for different frequency-severity settings:

- Greedy performs the best when combined with the All Moves acceptance.
- Choice Function is the best as a heuristic selection method to be combined with the Improving and Equal, Only Improving and EMCQ acceptance methods.
- Greedy seems to perform the best for low frequency changes, while the heuristic selection methods that rely on randomness, i.e., RPD and Simple Random perform better for higher frequency changes when combined with Simulated Annealing and Simulated Annealing with Reheating.
- Great Deluge based hyper-heuristics perform similarly regardless of the heuristic selection.

Overall, considering the average offline error results given in Table 4 and the statistical significance tests, Choice Function–Improving and Equal is the best performing hyper-heuristic for EXPSET1. Hypermutation performs the best when combined with the Improving and Equal and Only Improving acceptance methods. However, overall it is one of the heuristic selection methods which delivers very poor performance. Evolutionary Strategies performs well in the cases for which the change frequency is low. Its performance deteriorates as the frequency increases. CMAES performs the best only when both the change frequency and severity are low. For this particular case, ES is the second best performing approach and they are both better than Choice Function–Improving and Equal. For all the remaining frequency-severity settings, Choice Function–Improving and Equal performs the best.

Table 4: The offline error generated by each approach during the EXPSET1 experiments for different combinations of frequency and severity of change.

Algorithm	LF			MF			HF		
	LS	MS	HS	LS	MS	HS	LS	MS	HS
GR-AM	24.92	24.69	24.77	38.09	37.69	37.90	63.92	63.04	63.51
GR-OI	1.24	2.22	3.38	3.15	7.42	12.15	13.55	22.58	31.56
GR-IE	1.26	2.23	3.39	3.06	7.36	12.18	13.86	22.95	31.52
GR-GD	2.07	4.10	5.99	4.02	8.34	13.59	14.73	24.19	31.96
GR-EMCQ	2.69	3.67	4.76	4.89	8.20	12.96	14.16	22.72	31.72
GR-SA	3.52	7.28	13.20	11.47	18.50	23.71	38.49	43.33	46.48
GR-SA+RH	6.18	6.74	8.00	15.05	16.30	18.60	55.97	55.86	56.11
CF-AM	117.90	117.80	118.10	155.73	157.35	155.95	194.89	190.75	182.40
CF-OI	0.64	0.69	**0.79**	**1.25**	1.58	**2.17**	4.77	7.04	**11.45**
CF-IE	0.63	0.69	0.79	1.28	**1.52**	2.17	**4.49**	**6.69**	11.51
CF-GD	3.18	4.60	7.16	3.93	6.27	10.78	8.57	11.74	18.56
CF-EMCQ	0.81	0.88	1.00	1.51	1.79	2.46	4.94	7.15	11.67
CF-SA	6.59	11.45	19.74	38.60	42.23	58.41	140.04	136.46	140.97
CF-SA+RH	13.11	13.19	13.92	26.62	28.88	28.79	66.83	59.19	76.07
SR-AM	35.05	34.93	35.23	52.96	53.09	52.76	86.68	86.45	85.55
SR-OI	0.97	1.19	1.37	1.83	2.99	4.21	5.44	11.29	18.41
SR-IE	0.97	1.18	1.38	1.87	3.01	4.23	5.25	11.47	18.06
SR-GD	2.06	4.02	6.62	3.33	6.34	10.29	6.95	13.07	20.76
SR-EMCQ	1.68	2.08	2.31	2.77	4.06	5.19	6.47	12.13	18.51
SR-SA	3.70	9.72	15.96	6.79	15.02	24.01	40.63	42.20	48.23
SR-SA+RH	8.79	8.93	8.87	14.45	15.18	16.04	31.27	32.62	35.66
RL-AM	38.01	37.73	37.39	60.30	61.28	59.36	96.30	97.23	96.12
RL-OI	1.39	2.58	2.97	2.53	4.32	5.34	7.13	8.64	12.12
RL-IE	1.38	2.74	3.13	2.73	4.36	4.86	6.64	8.84	12.90
RL-GD	2.85	5.92	9.05	4.42	10.21	15.15	8.45	12.01	17.33
RL-EMCQ	2.59	3.15	3.46	5.05	6.13	6.01	7.33	9.47	13.58
RL-SA	6.39	12.18	17.57	12.52	19.53	27.69	47.51	54.81	61.05
RL-SA+RH	11.30	11.46	11.53	21.47	21.54	21.87	38.53	39.57	43.15
HM-AM	60.44	59.60	59.58	88.57	87.00	87.15	113.11	111.65	112.23
HM-OI	2.23	2.51	2.57	3.47	4.66	5.23	8.17	14.50	18.10
HM-IE	2.22	2.50	2.57	3.47	4.71	5.19	8.66	14.60	18.68
HM-GD	3.74	4.61	6.11	5.61	7.36	9.64	9.44	15.81	19.72
HM-EMCQ	2.57	2.78	2.86	3.92	4.95	5.50	9.37	14.76	18.49
HM-SA	5.14	9.14	14.87	9.79	15.51	23.90	56.10	65.38	70.23
HM-SA+RH	7.83	8.06	8.45	14.76	15.33	14.91	31.68	32.93	33.53
RPD-AM	36.60	36.90	36.43	54.86	54.28	54.40	88.81	88.96	89.45
RPD-OI	0.97	1.13	1.28	1.78	2.68	3.63	5.16	10.41	16.24
RPD-IE	0.96	1.13	1.28	1.78	2.68	3.70	5.09	10.27	16.36
RPD-GD	2.09	3.93	6.42	3.24	6.13	9.87	6.64	12.24	19.28
RPD-EMCQ	1.52	1.80	1.96	2.47	3.48	4.43	6.02	10.73	16.65
RPD-SA	3.56	9.19	15.04	6.27	14.16	23.00	39.41	42.66	48.91
RPD-SA+RH	8.14	8.07	8.50	13.83	14.19	14.96	30.75	32.66	35.28
ES	0.53	**0.65**	0.79	2.87	3.45	4.19	11.08	12.67	15.88
CMAES	**0.42**	1.59	3.15	1.96	5.60	10.06	9.66	13.57	19.97

3.2.2 Results for EXPSET2

Table 5 summarizes the results of EXPSET2 using MPB in which peak locations, their heights and widths are changed. Similar phenomena as in the previous part are observed during this set of experiments. The methods deteriorate in performance as the change frequency increases. We again performed statistical significance tests to determine the overall best heuristic selection and best move acceptance methods. Considering all hyper-heuristic experiments for which a different

move acceptance method is used, the Choice Function heuristic selection outperforms all others, except in the low frequency change cases. Here, RPD–EMCQ gives the better average results. In this set of experiments, the Improving and Equal, Only Improving and EMCQ acceptance methods all perform well. In most cases, there is no statistically significant difference between them when applied in combination with the Choice Function heuristic selection method.

Table 5: The offline error generated by each approach during the EXPSET2 experiments for different combinations of frequency and severity of change.

Algorithm	LF			MF			HF		
	LS	MS	HS	LS	MS	HS	LS	MS	HS
GR-AM	26.47	22.85	24.86	39.36	32.98	35.00	63.41	52.63	56.06
GR-OI	4.35	8.82	11.48	6.19	14.06	19.14	17.08	28.16	36.08
GR-IE	4.63	8.96	11.69	6.33	15.03	19.38	17.06	27.61	35.59
GR-GD	5.11	9.96	13.09	7.05	15.04	20.34	18.70	27.81	36.31
GR-EMCQ	5.35	8.52	11.47	8.35	13.37	19.50	17.60	28.80	36.47
GR-SA	5.52	11.10	17.17	15.88	20.19	25.86	43.31	40.30	45.14
GR-SA+RH	8.88	11.35	13.81	16.93	19.29	22.84	56.71	47.78	50.02
CF-AM	121.08	98.49	101.57	153.90	125.84	128.53	185.61	143.96	146.98
CF-OI	**3.56**	8.97	11.35	4.79	10.30	12.86	**7.46**	15.65	**23.24**
CF-IE	3.66	7.95	11.57	**4.38**	10.46	**12.37**	7.47	14.79	23.83
CF-GD	6.53	11.97	17.94	6.83	14.37	21.64	11.55	19.86	29.82
CF-EMCQ	4.27	9.09	12.71	4.69	**9.61**	13.75	8.60	**14.76**	24.78
CF-SA	10.30	18.47	29.43	37.01	62.23	78.52	140.89	119.09	125.79
CF-SA+RH	16.22	20.85	24.56	26.89	38.02	43.50	69.58	75.29	85.38
SR-AM	37.03	33.09	35.36	54.60	47.71	50.13	88.27	75.67	78.21
SR-OI	3.89	8.19	10.24	5.46	9.65	13.27	8.83	18.65	26.90
SR-IE	4.04	7.54	9.84	4.96	10.76	13.60	8.87	18.46	27.24
SR-GD	5.23	9.63	12.96	6.39	13.25	17.59	9.87	20.10	29.26
SR-EMCQ	4.78	7.84	10.23	5.79	10.04	14.04	10.03	18.83	28.16
SR-SA	5.36	12.79	19.78	9.06	18.05	27.41	44.63	41.90	50.21
SR-SA+RH	10.74	12.97	14.06	17.19	19.58	21.70	35.85	35.60	39.64
RL-AM	39.62	35.19	37.29	63.15	54.41	56.65	96.70	82.18	85.15
RL-OI	4.60	9.96	12.02	5.94	12.65	16.01	9.99	16.67	25.55
RL-IE	4.70	9.02	12.48	5.73	12.74	14.17	9.35	17.54	25.33
RL-GD	5.80	11.73	15.76	7.57	16.37	22.29	11.30	20.18	28.55
RL-EMCQ	5.29	8.71	10.81	8.00	12.64	15.94	10.66	16.94	25.64
RL-SA	8.76	14.86	21.26	16.27	22.86	33.97	54.44	57.35	66.82
RL-SA+RH	13.91	14.55	17.27	24.14	25.01	27.95	41.55	44.42	49.78
HM-AM	62.52	56.36	59.70	90.72	78.52	82.27	115.32	98.13	101.77
HM-OI	5.59	10.63	13.01	6.88	13.51	15.73	11.41	22.21	29.32
HM-IE	5.44	11.38	13.48	6.72	13.09	15.82	11.27	23.53	29.63
HM-GD	6.66	11.92	15.94	8.91	15.82	19.79	12.46	23.95	30.43
HM-EMCQ	5.80	9.90	12.49	7.09	12.95	15.48	12.59	22.39	29.49
HM-SA	7.50	13.82	22.14	12.10	20.13	32.12	62.87	72.03	81.57
HM-SA+RH	11.16	14.46	16.58	17.69	22.23	24.34	35.04	38.31	42.72
RPD-AM	38.80	33.99	36.77	56.75	48.90	51.39	90.22	76.93	80.70
RPD-OI	4.26	7.54	10.17	5.01	10.19	12.61	8.12	17.73	25.56
RPD-IE	4.14	8.12	10.28	5.00	9.67	12.54	8.31	16.65	26.20
RPD-GD	5.20	8.87	14.15	6.71	12.44	17.27	10.12	18.98	28.36
RPD-EMCQ	4.28	**7.42**	**9.34**	5.80	10.01	13.89	8.99	17.51	26.59
RPD-SA	5.16	12.32	19.30	8.51	17.44	26.67	42.81	42.63	51.06
RPD-SA+RH	10.26	12.30	13.78	16.50	19.09	21.29	33.40	34.82	39.29
ES	3.69	9.19	12.74	6.18	12.21	15.68	14.58	21.37	27.40
CMAES	6.20	13.78	17.01	7.86	17.25	21.28	15.53	24.17	31.73

Hypermutation is again among the worst performing heuristic selection methods. Unlike in the

previous experiments, in EXPSET2, both ES and CMAES do not perform the best in any of the frequency-severity settings. ES is better than CF-EMCQ only when both the change frequency and severity are low. In other cases, ES and CMAES are outperformed by the Choice Function heuristic selection in combination with either Improving and Equal, Only Improving or EMCQ acceptance methods.

3.2.3 Dynamic Environment Heuristic Search Challenge

Recently, CHESC – Cross-domain Heuristic Search Challenge[2], a competition on hyper-heuristics was held, which used HyFlex, a tool implemented for research and rapid development of hyper-heuristics. In this competition, different hyper-heuristics competed for solving problem instances from six different problem domains. As a comparison and ranking method, the organisers adopted the Formula 1 scoring system. The top eight approaches are given a score of 10, 8, 6, 5, 4, 3, 2 and 1 points for each problem instance from the best to the worst, successively. The rest of the approaches receive a score of 0. The comparison and the ranking of the approaches are based on the median result generated by each approach over a given number of runs for an instance. The sum of scores over all problem instances determine the final ranking of an approach.

In order to evaluate the performance of hyper-heuristics across different dynamic environments and see their relative performance as compared to the state-of-the-art techniques, all approaches are scored in the same way as in CHESC. Considering both EXPSET1 and EXPSET2 with all change frequency-severity combinations, there are 18 different problems. Therefore, 180 is the maximum overall score an approach can get. The results are summarized in Table 6, where the overall scores of the best five approaches are included. As can be seen from the results, Choice Function–Improving and Equal is the clear winner. Figure 3 shows the histogram of scores for this hyper-heuristic which ranks the first, second and third among all approaches in a total of sixteen out of eighteen cases. Only when a high severity change occurs at a low frequency, Choice Function–Improving and Equal performs worse than some others. The top three hyper-heuristics use Choice Function as the heuristic selection component. All hyper-heuristics using All Moves, Great Deluge, Simulated Annealing or Simulated Annealing with Reheating as an acceptance component perform poorly with an overall score of 0 regardless of the heuristic selection component. ES ranks eighth with a score of 37, CMAES ranks thirteenth with a score of 11, while all the Hypermutation based methods receive a score of 0 in all cases.

[2] http://www.asap.cs.nott.ac.uk/external/chesc2011/

Table 6: The overall scores for the top five approaches.

Approach	Overall Score
Choice Function–Improving and Equal	136
Choice Function–Only Improving	119
Choice Function–EMCQ	86
Random Permutation Descent–Only Improving	72
Random Permutation Descent–Improving and Equal	71

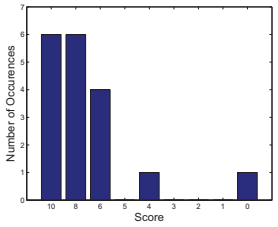


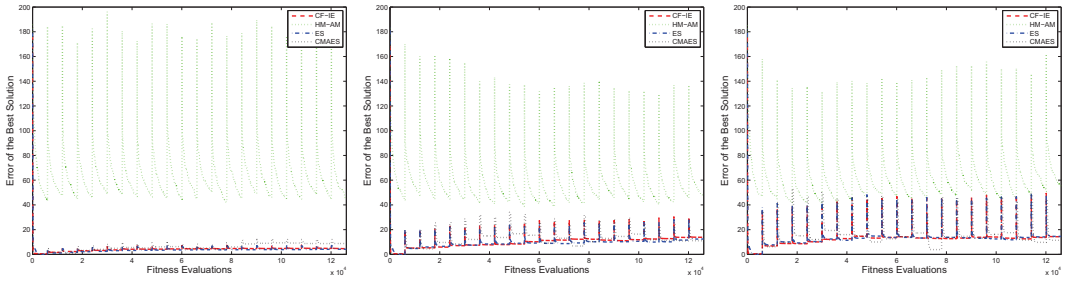
Figure 3: Histogram of scores for CF–IE over 18 dynamic environment cases.

3.2.4 Tracking Ability of the Approaches

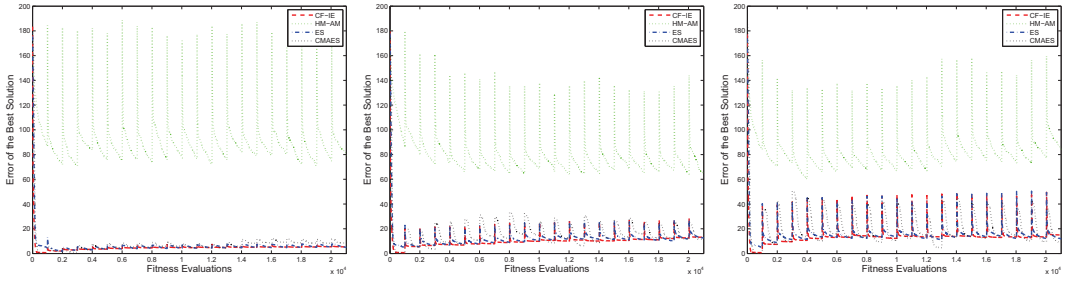
The error values of the best candidate solutions calculated using Equation 18 versus the number of evaluations based on different change frequency and severity combinations are plotted in Figure 4 for Choice Function–Improving and Equal, Hypermutation–All Moves, ES and CMAES to illustrate and compare their tracking ability when the environment changes. Choice Function–Improving and Equal is chosen as the best performing hyper-heuristic, while Hypermutation–All Moves is chosen as a poor approach. ES and CMAES are included as they are known to be among the best real-valued optimization approaches. Figure 6 shows the boxplots for the final offline error values of the corresponding approaches. In the boxplot, the minimum and maximum values obtained (excluding the outliers), the lower and upper quartiles and the median are shown. The outlier points are also marked.

To be able to demonstrate the tracking behavior of the approaches more clearly, we isolated the plots for a medium frequency and a medium severity change scenario from Figure 4, and plotted them in Figure 5. From Figures 4 and 5, it can be observed that when the environment changes, the error values of the best candidate solutions produced by Choice Function–Improving and Equal, ES and CMAES increase much less than that of Hypermutation–All Moves. Moreover, these approaches are able to recover much more quickly, following the optimum. This indicates that Choice Function–Improving and Equal, ES and CMAES display a good tracking behavior. However, the tracking behavior of Hypermutation–All Moves is poor. Choice Function-Improving

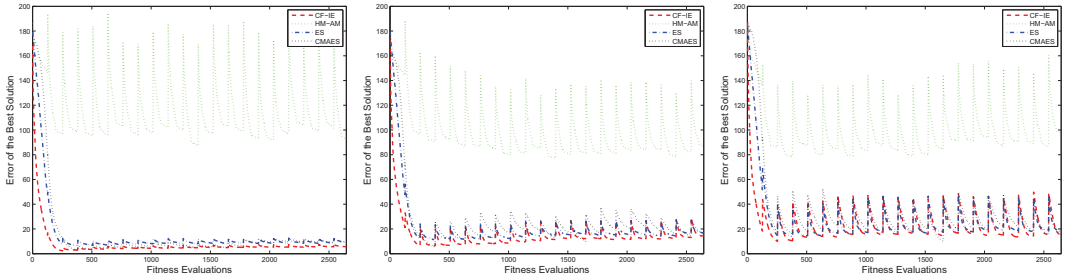
and Equal performs significantly better than the Hypermutation-All Moves, ES and CMAES on average during most of the environment changes as illustrated in Figure 6. The average performance of an approach reflects upon its tracking behaviour as well.



(a) LF (-LS, MS and HS)



(b) MF (-LS, MS and HS)



(c) HF (-LS, MS and HS)

Figure 4: Comparison of approaches (CF-IE, HM-AM, ES, and CMAES) for the combinations of (a) Low, (b) Medium, (c) High frequencies and severities of change based on the error values of the best candidate solution versus evaluation counts for EXPSET2.

3.2.5 Scalability Results

In this part, we investigate the scalability of the approaches for different frequency-severity settings. We performed experiments with different number of peaks and dimensions. Table 7 summarizes the results for analyzing the effect of the number of dimensions on performance using EXPSET2 for different change frequency and severity combinations. In these experiments, only the best hyper-heuristics {Choice Function–Improving and Equal, Choice Function–EMCQ} are considered

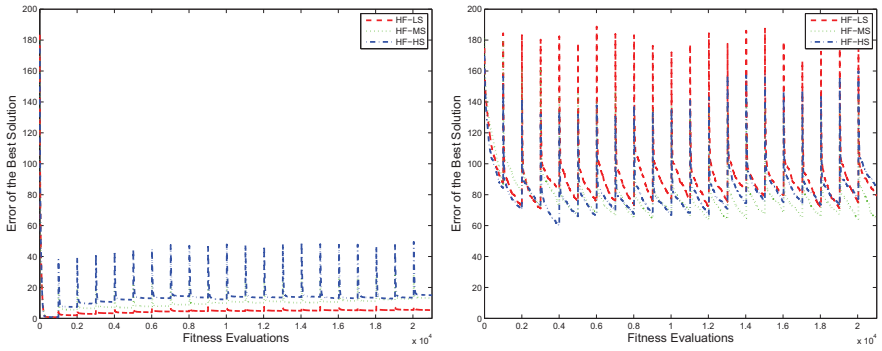
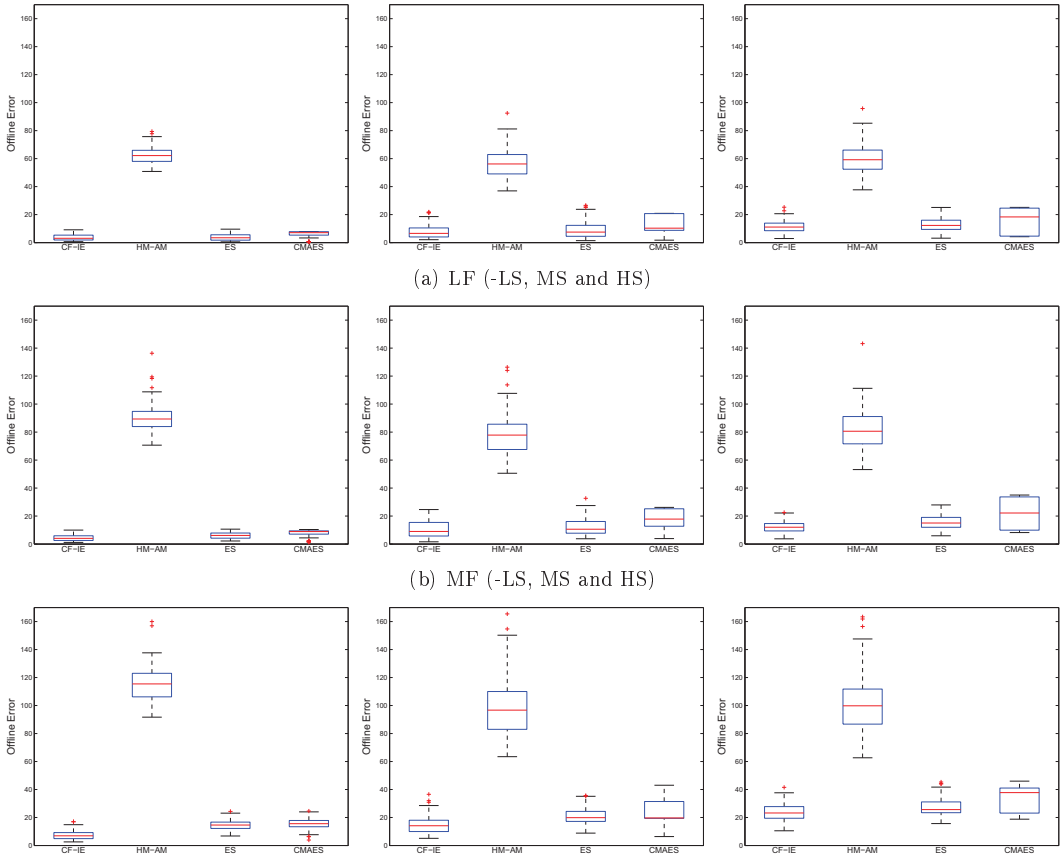


Figure 5: A sample plot of the error values of the best candidate solution versus evaluation counts based on medium change frequency and medium severity combination for EXPSET2. The left and right plots show the results for Choice Function–Improving and Equal and Hypermutation–All Moves, respectively.



(a) LF (-LS, MS and HS)

(b) MF (-LS, MS and HS)

(c) HF (-LS, MS and HS)

Figure 6: Box-plots of offline error values for a statistical comparison of the approaches (CF-IE, HM-AM, ES, and CMAES) for the combinations of (a) Low, (b) Medium, (c) High frequencies and severities of change using EXPSET2.

along with Hypermutation–Improving and Equal, Hypermutation–EMCQ, ES and CMAES. As expected, the performance of the hyper-heuristics and ES worsens as the number of dimensions increases. ES seems to be less affected from the dimensionality increase for lower frequency and severity settings. CMAES improves its performance when the number of dimensions is increased to 10. However, Choice Function–Improving and Equal scales better and is the best performing approach for higher change frequency and severity settings.

Table 7: Offline error generated by each approach in the experiments for analyzing the effect of number of dimensions for EXPSET2 for different frequency and severity combinations

		# of dimensions	CF-IE	CF-EMCQ	HM-IE	HM-EMCQ	ES	CMAES
LF	LS	5	**3.78**	4.52	5.39	5.74	4.14	5.98
		10	4.85	5.61	9.22	11.03	4.60	**2.30**
		20	8.22	10.34	17.76	24.09	5.65	**4.75**
	MS	5	**9.04**	9.29	12.40	10.06	9.73	12.05
		10	11.35	11.47	15.91	15.03	9.88	**7.08**
		20	15.46	16.37	26.22	25.76	13.73	**11.94**
	HS	5	**10.53**	12.02	12.42	12.43	12.16	12.48
		10	13.29	15.06	19.99	17.68	**12.67**	14.15
		20	18.21	21.27	32.57	29.62	**16.56**	21.84
MF	LS	5	**4.61**	4.79	6.66	7.20	6.35	7.94
		10	6.99	7.69	11.33	13.92	9.22	**5.35**
		20	13.34	15.45	21.99	28.51	17.66	**12.46**
	MS	5	10.91	**10.23**	13.11	13.07	13.10	17.94
		10	13.21	14.00	19.89	21.26	17.68	**12.31**
		20	**21.00**	22.14	33.63	36.96	30.54	22.83
	HS	5	**13.43**	14.06	15.69	15.11	15.67	24.23
		10	**17.26**	18.67	25.47	24.78	23.07	21.80
		20	**24.94**	27.32	41.16	44.28	37.12	33.39
HF	LS	5	**8.48**	8.50	11.68	13.03	14.04	14.70
		10	**17.38**	19.60	23.22	26.32	31.88	27.20
		20	49.53	52.28	**48.42**	53.73	68.98	73.81
	MS	5	15.82	**15.56**	22.62	23.33	21.26	23.18
		10	**27.40**	28.88	36.73	36.70	40.03	34.87
		20	**61.30**	61.68	64.35	63.72	78.68	86.49
	HS	5	**24.07**	24.39	29.81	29.55	28.32	42.36
		10	**37.32**	38.96	50.78	49.52	46.62	50.22
		20	**75.16**	76.25	78.59	75.79	85.02	86.55

Table 8 provides the results for analyzing the effect of the number of peaks in the environment on performance using EXPSET2 for different change frequency and severity combinations. The same hyper-heuristics {Choice Function–Improving and Equal, Choice Function–EMCQ, Hypermutation–Improving and Equal, Hypermutation–EMCQ}, ES and CMAES are included in the experiments. Again, the performance of the hyper-heuristics worsens as the number of dimensions increases. This time, ES performs similar to the hyper-heuristics. However, the effect of the increase in the number of peaks is less than the effect of the increase in dimensionality for ES. CMAES improves its performance as the number of peaks increases for all frequencies combined with low and medium severities. However, in almost all cases, it is no longer the best performing

approach. All methods seem to scale well with respect to the increase in the number of peaks in the environment.

Table 8: Offline error generated by each approach in the experiments for analyzing the effect of number of peaks for EXPSET2 for different frequency and severity combinations

		# of peaks	CF-IE	CF-EMCQ	HM-IE	HM-EMCQ	ES	CMAES
LF	LS	5	**3.76**	4.00	5.40	5.79	3.95	5.85
		10	4.75	4.82	6.27	6.32	**4.65**	5.11
		15	4.83	5.33	6.95	6.83	4.87	**3.38**
	MS	5	9.67	**9.05**	9.90	10.16	9.07	13.88
		10	11.04	11.06	12.72	**10.53**	10.95	13.14
		15	11.47	11.15	12.26	**9.73**	11.36	11.06
	HS	5	11.55	**11.27**	13.56	12.41	11.33	14.06
		10	13.74	14.16	14.55	**13.22**	14.34	16.06
		15	14.52	13.51	15.08	**12.28**	13.72	17.48
MF	LS	5	4.58	**4.43**	6.48	7.17	6.23	7.28
		10	**4.93**	5.22	7.36	7.95	6.82	6.88
		15	5.87	5.67	7.41	8.33	7.60	**5.20**
	MS	5	**9.63**	10.42	12.31	11.14	11.46	17.53
		10	**11.06**	12.23	14.45	12.57	13.62	17.36
		15	12.41	**11.59**	14.31	12.31	13.50	16.28
	HS	5	**12.96**	13.66	15.82	15.48	15.69	22.58
		10	**14.77**	15.43	17.27	16.47	17.62	23.97
		15	**15.12**	15.73	16.20	16.38	17.36	26.29
HF	LS	5	8.14	**7.84**	11.06	12.08	14.54	15.97
		10	**8.08**	8.50	11.82	12.92	13.77	13.79
		15	**8.20**	8.74	11.81	12.55	13.52	10.37
	MS	5	**15.32**	16.38	21.93	23.49	21.77	28.37
		10	**16.31**	17.24	23.62	22.38	21.63	23.64
		15	**16.81**	17.00	21.97	21.89	21.82	23.35
	HS	5	24.70	**24.38**	29.06	29.67	27.34	33.46
		10	**24.83**	26.50	29.79	28.81	28.67	30.28
		15	**25.11**	26.04	27.59	27.50	28.13	35.18

4 Conclusion

In this study, we investigated the performance of thirty five hyper-heuristics combining five heuristic selection methods {Simple Random, Greedy, Choice Function, Reinforcement Learning, Random Permutation Descent} and seven move acceptance methods {All Moves, Only Improving, Equal and Improving, Exponential Monte Carlo With Counter, Great Deluge, Simulated Annealing, Simulated Annealing with Reheating}. A hypermutation based single point search method, combined with these seven acceptance schemes, (1+λ)-ES and the state of-the-art real valued optimization approach (μ,λ)-Covariance Matrix Adaptation Evolution Strategy are also included in the tests. The Moving Peaks Benchmark, a multidimensional dynamic function generator, is used for the experiments. Different dynamic environments are produced by changing the height, width and location of the peaks in the landscape with desired change frequencies and severities. Even

though there are many successful applications of selection hyper-heuristics to discrete optimization problems, to the best of our knowledge, this study is one of the initial applications of selection hyper-heuristics for real-valued optimization as well as being among the very few which address dynamic optimization issues with these techniques.

The empirical results show that learning selection hyper-heuristics perform well in dynamic environments, especially when combined with the proper acceptance method. The learning selection hyper-heuristics can react rapidly to different types of changes in the environment and they are capable of tracking them closely. The acceptance criteria relying on some algorithmic parameter settings such as Simulated Annealing did not perform well as part of a hyper-heuristic in dynamic environments. This is possibly because the relevant parameters of such non-deterministic or stochastic acceptance methods often require a search for tuning. In dynamic environments, as a result of the changes in the environment, another level of complexity is added on top of the search process for the best (optimum) solution, also increasing the size of the search space.

The overall results also show that accepting all moves is the worst strategy regardless of the heuristic selection method for solving dynamic environment problems. As an online learning approach which receives feedback during the search process, the Choice Function–Improving and Equal hyper-heuristic ranks performance-wise the first among all others. Evolutionary Strategies, Covariance Matrix Adaptation Evolution Strategy and Hypermutation perform mostly worse than the learning selection hyper-heuristics when compared across a range of dynamic environments exhibiting a variety of change properties. A goal in hyper-heuristic research is to design automated methodologies that are applicable to a range of stationary problems. This study shows that learning selection hyper-heuristics are sufficiently general, which makes them viable approaches to solve not only dynamic problems regardless of the change dynamics in the environment, but also continuous optimisation problems. The results are promising which promote further study.

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

References

[1] M. Ayob and G. Kendall. A monte carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In *Proceedings of the Int. Conf. on Intelligent*

Technologies, pages 132–141, 2003.

[2] R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum. A simulated annealing hyper-heuristic methodology for flexible decision support. Technical Report NOTTCS-TR-2007-8, School of CSIT, University of Nottingham, 2007.

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

[4] H. G. Beyer and H. P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

[5] B. Bilgin, E. Özcan, and E. E. Korkmaz. An experimental study on hyper-heuristics and exam timetabling. In *Proceedings of the 6th International Conference on Practice and Theory of Automated Timetabling*, pages 123–140, 2006.

[6] J. Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Congress on Evolutionary Computation CEC 99*, volume 3, pages 1875–1882. IEEE, 1999.

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

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

[9] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and Q. Rong. A survey of hyper-heuristics. Technical report, 2009.

[10] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research and Management Science*, pages 449–468. Springer, 2010.

[11] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. Exploring hyper-heuristic methodologies with genetic programming. In Janusz Kacprzyk, Lakhmi C. Jain, Christine L. Mumford, and Lakhmi C. Jain, editors, *Computational Intelligence*, volume 1 of *Intelligent Systems Reference Library*, pages 177–201. Springer, 2009.

[12] E. K. Burke, G. Kendall, M. Misir, and E. Özcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, pages 1–18, 2010.

[13] K. Chakhlevitch and P. Cowling. Hyperheuristics: Recent developments. In Carlos Cotta, Marc Sevaux, and Kenneth Sirensen, editors, *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer Berlin Heidelberg, 2008.

[14] H. G. Cobb. An investigation into the use of hypermutation as an adaptive operator in genetic algorithms having continuous, time-dependent nonstationary environments. Technical Report AIC-90-001, Naval Research Lab., Washington, DC, 1990.

[15] P. Cowling, G. Kendall, and E. Soubeiga. A hyper-heuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III : Third International Conference, PATAT 2000*, volume 2079 of *LNCS*. Springer, 2000.

[16] P. Cowling, G. Kendall, and E. Soubeiga. A parameter-free hyper-heuristic for scheduling a sales summit. In *Proceedings of the 4th Metaheuristic International Conference*, pages 127–131, 2001.

[17] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In *EvoWorkShops*, volume 4193 of *Lecture Notes in Computer Science*, pages 1–10. Springer, 2002.

[18] W. B. Crowston, F. Glover, G. L. Thompson, and J. D. Trawick. Probabilistic and parametric learning combinations of local job shop scheduling rules. *ONR Research memorandum, GSIA, Carnegie Mellon University, Pittsburgh*, (117), 1963.

[19] 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*, 15:1427–1448, 2011.

[20] A. Ş. Uyar and A. E. Harmanci. A new population based adaptive domination change mechanism for diploid genetic algorithms in dynamic environments. *Soft Computing*, 9:803–814, 2005.

[21] J. Denzinger, M. Fuchs, and M. Fuchs. High performance ATP systems by combining several AI methods. In *4th Asia-Pacific Conf. on SEAL*, pages 102–107, 1997.

[22] 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.

[23] H. Fisher and G. L. Thompson. Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth and G. L. Thompson, editors, *Industrial Scheduling*, pages 225–251. Prentice-Hall, 1963.

[24] J. Gibbs, G. Kendall, and E. Özcan. Scheduling english football fixtures over the holiday period using hyper-heuristics. In Robert Schaefer, Carlos Cotta, Joanna Kolodziej, and Günter Rudolph, editors, *Parallel Problem Solving from Nature - PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 496–505. Springer Berlin / Heidelberg, 2011.

[25] J. J. Grefenstette. Genetic algorithms for changing environments. In *Proceedings of Parallel Problem Solving from Nature*, pages 137–1446, 1992.

[26] N. Hansen. The CMA evolution strategy: A tutorial. Technical report, june 28, 2011.

[27] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol*, 11(1):1–18, 2003.

[28] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[29] Y. Jin and J. Branke. Evolutionary optimization in uncertain environments-a survey. *IEEE Trans. on Evolutionary Comp.*, 9(3):303–317, 2005.

[30] G. Kendall and M. Mohamad. Channel assignment in cellular communication using a great deluge hyperheuristic. In *IEEE Int. Conf. on Network*, pages 769–773, 2004.

[31] B. Kiraz, A. Ş. Uyar, and E. Özcan. An investigation of selection hyper-heuristics in dynamic environments. In *Proceedings of EvoApplications 2011*, volume 6624 of *LNCS*. Springer, 2011.

[32] 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*, pages 1487–1492, 2010.

[33] J. Lewis, E. Hart, and G. Ritchie. A comparison of dominance mechanisms and simple mutation on nonstationary problems. In *Proceedings of Parallel Problem Solving from Nature*, pages 139–148, 1998.

[34] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical Programming*, 34:111–124, 1986.

[35] R. W. Morrison. *Designing evolutionary algorithms for dynamic environments*. Springer, 2004.

[36] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In *Metaheuristics: Computer Decision-Making*, pages 523–544. Kluwer Academic Publishers, 2001.

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

[38] E. Özcan, A. Ş. 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*, pages 2201–2204, 2009.

[39] 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*, 1(1):39–59, 2010.

[40] P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. Springer, 2005.

[41] R. K. Ursem. Multinational GA optimization techniques in dynamic environments. In *Proceedings of the Genetic Evol. Comput. Conf.*, pages 19–26, 2000.

[42] F. Vavak, K. Jukes, and T. C. Fogarty. Adaptive combustion balancing in multiple burner boiler using a genetic algorithm with variable range of local search. In *Proceedings of the Int. Conf. on Genetic Algorithms*, pages 719–726, 1997.

[43] S. Yang. Genetic algorithms with memory and elitism based immigrants in dynamic environments. *Evolutionary Computation*, 16:385–416, 2008.

[44] S. Yang, Y. S. Ong, and Y. Jin, editors. *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Int.* Springer, 2007.

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