# A Comprehensive Analysis of Hyper-heuristics

Ender Özcan, Burak Bilgin, Emin Erkan Korkmaz
Yeditepe University
Department of Computer Engineering
İnönü Cad. Kayışdağı Mah.
34755 Kadıköy/İstanbul Turkey
{eozcan, bbilgin, ekorkmaz}@cse.yeditepe.edu.tr

**Abstract.** Meta-heuristics such as simulated annealing, genetic algorithms and tabu search have been successfully applied to many difficult optimization problems for which no satisfactory problem specific solution exists. However, expertise is required to adopt a meta-heuristic for solving a problem in a certain domain. Hyper-heuristics introduce a novel approach for search and optimization. A hyper-heuristic method operates on top of a set of heuristics. The most appropriate heuristic is determined and applied automatically by the technique at each step to solve a given problem. Hyper-heuristics are therefore assumed to be problem independent and can be easily utilized by non-experts as well. In this study, a comprehensive analysis is carried out on hyper-heuristics. The best method is tested against genetic and memetic algorithms on fourteen benchmark functions. Additionally, new hyper-heuristic frameworks are evaluated for questioning the notion of problem independence.

## 1. Introduction

It is difficult to develop problem-specific heuristics or algorithms for solving a broad class of computational problems. Meta-heuristics have been proposed to provide a general framework that can be applicable to different domains. However, the efficiency of meta-heuristics depends on the neighborhood operators provided by the user and the best alternative for a problem domain can only be formalized by an expert. Hyper-heuristics provide a new approach which is proposed to overcome the problem of such dependencies in meta-heuristics. The aim of this study is to provide a comprehensive analysis of hyper-heuristics. Different frameworks that can be utilized for hyper-heuristics are also analyzed and the notion of problem-independence is questioned for this emerging technique.

Hyper-heuristic methods are described in [11] as “knowledge poor” heuristics that are used to choose the most appropriate low level heuristic from a set of heuristics during the search. Hyper-heuristics represent a class of methods that are non-problem specific. The decision for a heuristic selection is based on problem independent measures, such as the change in the quality of a solution when the selected heuristic is used. Once implemented, they can be directly used in another problem domain. Hyper-heuristics are presented as an alternative to the meta-heuristics that are mostly developed for a particular problem and require fine tuning of related parameters. Therefore, they can be developed and deployed by even non-specialized programmers with little or no experience of the problem domain.

The process in a hyper-heuristic can be divided into two phases: *heuristic selection* and *move acceptance*. Although there are many heuristic selection and move acceptance mechanisms, there is no comprehensive study to compare the performances of these different mechanisms in depth. Moreover, there is almost no discussion whether it is possible to make better use of hill climbers, or the affect of hill climbing algorithm choices within a hyper-heuristic framework. In this paper, the characteristics of the hyper-heuristic method are analyzed in different dimensions. All possible combinations of seven heuristic selection methods and five different acceptance criteria are tested on a group of benchmark problems. These are well-known problems widely used in the search and optimization community. The aim is to obtain a comprehensive analysis of the alternative selection and acceptance mechanisms that can be used with hyper-heuristics. The method is also tested against other meta-heuristics like genetic algorithms (GAs) and memetic algorithms (MAs) that are based on Darwinian evolution and population genetics. The comparisons presented in the paper provide insight about the effectiveness of this newly proposed method relative to meta-heuristics.

Heuristics are black-box systems that modify a provided candidate solution. The hyper-heuristic framework [5] allows the use of any type of heuristics. Two different types of heuristics can be identified: *mutational heuristics* and *hill climbers*. Hill climbers as local search components evaluate the quality of the candidate solution at hand. The aim of a hill climber is to produce a better candidate solution at each step, while mutational heuristics are not expected to produce a better candidate solution after they are applied. Mutational heuristics dwell on random perturbation. The classical framework used with hyper-heuristics selects a mutational heuristic or a hill climber at each step and applies it to the current candidate solution. However, it is possible to come up with alternative frameworks for this selection mechanism. Depending on the structure of the search space, it might be desirable to apply a hill climber whenever a mutational heuristic is used. A framework which guarantees this cycle would increase the efficiency in some domains. In this study, three alternative main frameworks and their possible variants are proposed for the hyper-heuristic methodology. Each framework utilizes the hill climbers and mutational heuristics in a different way. The experiments denote that the structure of the framework affects the performance significantly depending on the problem domain. It can be stated that hyper-heuristics do provide a certain level of independence from the problem domain. It is possible to apply the same set of heuristics to different problems when the hyper-heuristic method is used. However, this situation does not provide an escape from the "no free lunch" theorem [45]. The experiments carried out denote that, this time the framework used for the methodology has the tendency to become problem dependent. Hence, the characteristics of the problem should still be considered even if the hyper-heuristic approach is used. All search algorithms have a problem dependent nature. This characteristic appears at a higher level of abstraction with the hyper-heuristic approach.

Another contribution of this study is related to the use of hyper-heuristic method in meta-heuristics. Note that, the MAs use a hill-climber after generic operators are applied. Determining the appropriate hill climber is again problem-dependent. In this study, a hyper-heuristic approach utilizing a set of hill climbers is embedded in the MAs. Hence, the hill climber to be used is dynamically determined by the hyper-heuristic. The experiments carried out denote that the proposed approach can increase the performance of the MA.

This study combines and extends the initial studies provided in [34] and [36]. Section 2 provides some background on hyper-heuristics and hyper-heuristic frameworks. Then the benchmark functions used during the experiments and the heuristics utilized within the algorithms are introduced in Section 3. Memetic algorithms are presented in section 4. The details of the experiments are discussed in Section 5. Finally, the conclusions are presented in Section 6.

## 2. Hyper-heuristics

Hyper-heuristics have been used in solving search and optimization problems increasingly [11],[3],[6]. A hyper-heuristic acts as a heuristic scheduler over a set of heuristics that does the scheduling in a *deterministic* or a *non-deterministic* way. For example, the deterministic round-robin strategy schedules the next heuristic in a queue at each turn. A nondeterministic strategy schedules the next heuristic based on some probability distribution. More complicated and viable hyper-heuristics can be designed by making use of a learning mechanism that gets a feedback from the previous choices to select the right heuristic at each step. Furthermore, meta-heuristics can be used as a hyper-heuristic or a hyper-heuristic can be used within a meta-heuristic.

In [22], the genetic algorithm based a hyper-heuristic is used for managing a set of heuristics to solve chicken catching and transportation problem. In [10], eleven hyper-heuristics are experimented, including a set of greedy, simple random, peckish and variants of a tabu-search on two personnel scheduling problems. As a total of 95 heuristic combinations for event and resource scheduling are proposed to work underneath the hyper-heuristics. In [6], a hyper-heuristic that contains a tabu-search heuristic selection mechanism is proposed in order to rank low level heuristics used for timetabling problem. Each heuristic is initially assigned to the same rank. Rank of a heuristic is modified based on a reinforcement learning mechanism that considers the change in the quality of a candidate solution. Ranks are allowed to change within a predetermined range. A tabu list is used for maintaining the low level heuristic(s) generating worsening moves. *Tabu duration* is used to set the maximum number of iterations for which a heuristic can stay in the tabu list. In [40], a messy-GA like algorithm is used as a hyper-heuristic and experiments on a set of timetabling problem instances with three different fitness measures are proposed. In [7], a simple generic hyper-heuristic is introduced which utilizes constructive heuristics (graph coloring heuristics) to tackle timetabling problems. A tabu-search algorithm chooses among permutations of constructive heuristics according to their ability to construct complete, feasible and low cost timetables. In [12], hyper-heuristics are successfully merged with ant-colony optimization algorithm for solving 2D bin packing problems. In [8], a case based heuristic selection is used where the method is based on a knowledge discovery method to find the problem instances and situations where a specific heuristic works well. The proposed method explores the similarities between the problem instance to be solved and the source case, to predict the heuristic that will perform the best. In [3], the performance of a large set of hyper-heuristics is investigated on examination timetabling benchmark problems. A set of smart mutational heuristics are used within a generic hyper-heuristic framework. The experimental results show that there is a significant performance variance among hyper-heuristics.

### *2.1. Simple hyper-heuristics*

In this study, seven heuristic selection mechanisms and five acceptance criteria are paired up as simple hyper-heuristics that are obtained from the previous studies as presented in

Table 1. As a result, a broad range of hyper-heuristic variants are obtained. They are tested on a set of mathematical benchmark functions. Most of the simple methods to form a simple hyper-heuristic are discussed in [11]. *Simple Random* (SR) heuristic selection mechanism as its name suggests chooses a low level heuristic randomly based on a uniform probability distribution. *Random Descent* (RD) chooses a low level heuristic randomly and applies it repeatedly until no improvement is achieved. *Random Permutation* (RP) generates a random initial permutation of the low level heuristics and at each step applies the next low level heuristic in the provided order. *Random Permutation Descent* (RPD) works similar to the *Random Permutation*, except that it applies the low level heuristic in turn repeatedly as long as it produces improving results. The *Greedy* (GR) method allows all heuristics to process a given candidate solution and chooses the one that generates the most improved solution. *Choice Function* (CF) analyzes both the performance of each low level heuristic and each successively applied pair of low level heuristics. This analysis is based on the quality improvement, execution time and the overall performance. In [17], a study on the CF based hyper-heuristics is presented, where generalized low-level heuristics and utilization of parallel computing environments for hyper-heuristics are used. Two simple acceptance criteria are provided in [11]; *AM*, that accepts all moves and *OI*, that accepts only improving moves. It is not clear in their study whether OI accepts equal quality solutions or not. In this paper, OI refers to the mechanism that does not accept such solutions. They experiment combinations of heuristic selection and move acceptance criteria on a single sales summit problem. The results indicate that CF_AM approach is promising. In [3], *IE* move acceptance mechanism which rejects only the worsening moves, is used in the experiments. If a heuristic generates a solution that has an improved or equal quality as compared to the previous solution then it is accepted.

Table 1
Heuristic selection and move acceptance strategies that are investigated in this study

| *Label* | *Name of the strategy* | *Method type* | *Source(s)* |
|---|---|---|---|
| SR | Simple Random | Heuristic Selection | [11] |
| RD | Random Descent | Heuristic Selection | |
| RP | Random Permutation | Heuristic Selection | |
| RPD | Random Permutation Descent | Heuristic Selection | |
| CF | Choice Function | Heuristic Selection | [11], [17] |
| TS | Tabu Search | Heuristic Selection | [4] |
| GR | Greedy | Heuristic Selection | [11] |
| AM | All Moves | Move Acceptance | |
| OI | Only Improving | Move Acceptance | |
| IE | Improving and Equal | Move Acceptance | [2], [3], [36] |
| MC | Exponential Monte Carlo with Counter | Move Acceptance | [2] |
| GD | Great Deluge | Move Acceptance | [24] |

In [2], *Monte Carlo* mechanisms are proposed where all improving moves are accepted and the acceptance of non-improving moves are allowed based on a dynamically changing probability function, denoted by $p_t$. *Linear Monte Carlo* uses a negative linear ratio of the probability of acceptance to the fitness worsening. *Exponential Monte Carlo* (EMC) uses a negative exponential ratio of the probability of acceptance to the fitness worsening as shown in Equation (1). *Exponential Monte Carlo with Counter* (MC) is an improved version of EMC. Additional to the usual process, if no improvement can be achieved over a series of consecutive iterations then the probability starts increasing. As a heuristic selection mechanism, all mechanisms utilize SR. The authors compare different hyper-heuristics based on MC and CF for solving a real world problem. The IO mechanism suggested in their paper is in fact the IE mechanism. The results indicate the success of MC.

$$p_t = e^{-\frac{\Delta f / N}{1 - t/D}} \tag{1}$$

where $\Delta f$ is the fitness change during the $t^{\text{th}}$ iteration, $D$ is the maximum number of iterations, $N$ is an expected range for the maximum fitness change.

In [24], another stochastic acceptance mechanism is proposed to be used within hyper-heuristics. In the experiments carried out on a set of channel assignment problems, the hyper-heuristic uses *Great Deluge (GD)* as the acceptance criterion and SR as the heuristic selection method. In GD, all moves are accepted generating a better or equal objective value than a level computed at each step during the search. The initial level is set to the objective value of the initial candidate solution. At each step, the level is updated at a linear rate towards the expected objective value ($f_o$).

$$\theta_t = f_o + N \times \left(1 - \frac{t}{D}\right) \tag{2}$$

where $\theta_t$ is the threshold level during the $t^{\text{th}}$ iteration in a minimization problem, $D$ is the maximum number of iterations, $N$ is an expected range for the maximum fitness change.

## *2.2. Hyper-heuristic frameworks*

As noted in the previous sections, hyper-heuristic approach can be roughly divided into two different phases. These are the selection and the acceptance processes used by the method. The selection mechanism decides on the mutational heuristic or the hill climber that will be applied at each step. The traditional framework that has been used by the researchers combine all of the mutational heuristics and hill climbers in a single set and use a selection mechanism (SR, GR,…,etc.) to choose one from the set. However, recent studies presented in [34] and [38] show that in memetic algorithms, using a single efficient hill climber might yield better solutions compared to using a set of hill climbers where the operator selection is carried out self adaptively. The mutational heuristics and hill-climbers have different characteristics in terms of the effect they have on the search process. Mutational heuristics introduce random perturbations while hill-climbers try to exploit the search space. In some situations, after applying a mutational heuristic a hill climbing might be desirable. For example, if IE is used in the hyper-heuristic, then most of the mutational heuristic moves will be declined. As a result, the traditional framework is questionable and it is worth considering different alternative frameworks which can apply the heuristics in different styles and hence which can provide insight about the interaction of the mutational heuristics and hill-climbers. In this study, four different frameworks are defined and used for the selection mechanism. These frameworks are named as $F_A$, $F_B$, $F_C$ and $F_D$, and they are summarized in Figure 1.

$F_A$ is the traditional framework where a mutational heuristic or a hill-climber can be chosen at each step without regarding discrimination between the two groups. The other frameworks are the newly proposed ones. In these frameworks, a hill climber is utilized separately to make better use of diversity provided by mutational heuristics. Therefore, in $F_B$, if a mutational heuristic is chosen then a predefined hill climber is applied to the candidate solution. This guarantees that at each step a hill climbing process will be used during the search. Note that, the mutational heuristics and hill climbers still exist together in the selection set in $F_B$.

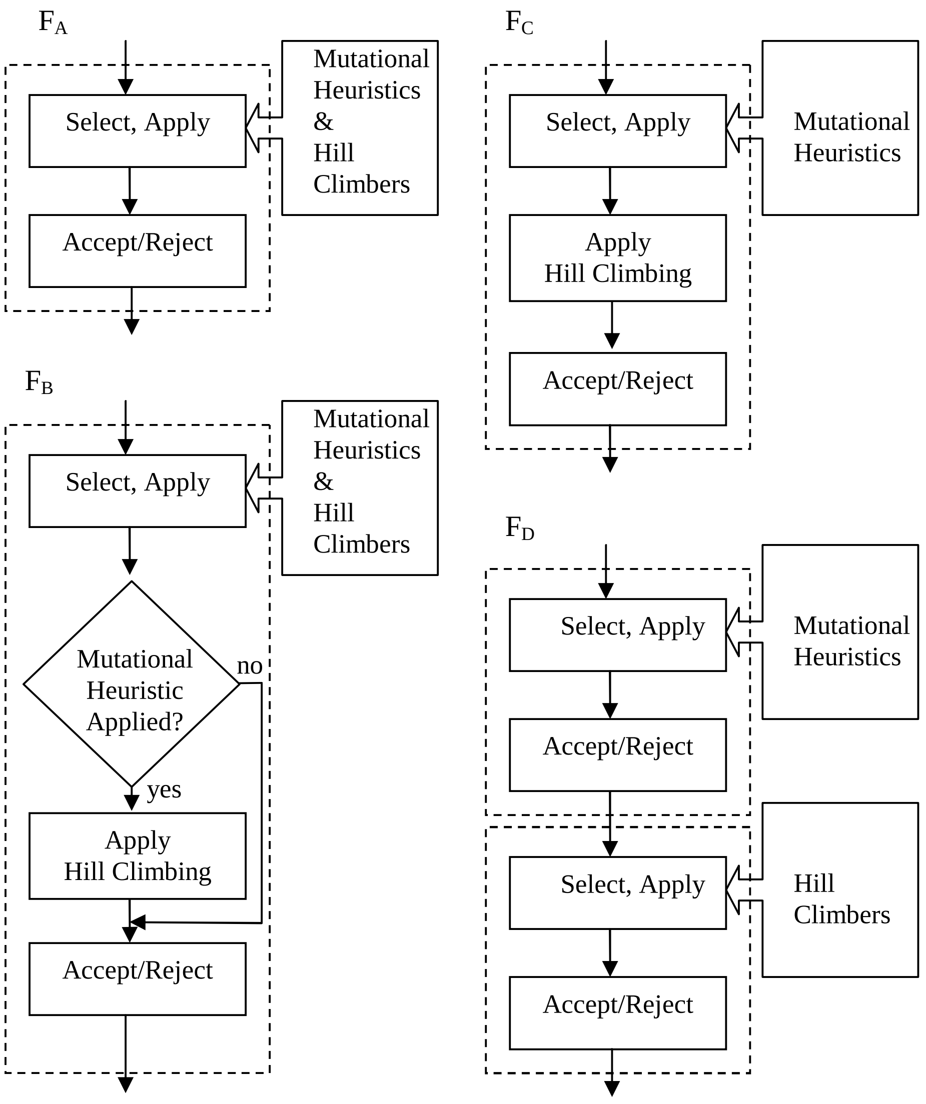


Fig. 1. Hyper-heuristic frameworks

In $F_C$, only mutational heuristics are in the selection set. So, first a mutational heuristic is applied and then a predefined hill climber is used. It is possible to use only a hill-climber at some steps in $F_B$, but using a mutational heuristic and then the hill-climber is guaranteed in $F_C$. $F_D$ is a more general form of $F_C$. Two hyper-heuristic modules are used; one for the mutational heuristics and one for the hill climbers. Hence, in $F_C$ the same hill-climber is used in all steps, but in $F_D$ different hill climbers can come into play.

There are two other alternatives for the heuristic selection mechanism in (*in-mode*) $F_B$, $F_C$ and (*independent*) $F_D$. Unless mentioned these frameworks will be assumed to operate as described previously. In $F_B$ and $F_C$, the processes between heuristic selection and acceptance mechanism can be moved outside the hyper-heuristic module in Figure 1. In that manner, the heuristic selection mechanism makes its decisions by evaluating the performance of a mutational heuristic, ignoring the combined effect of the hill climber (*out-mode* operation). In $F_D$ *with feedback*, the heuristic selection mechanism on top of the mutational heuristics can get an additional feedback from the output obtained after applying a selected hill climber; meanwhile the heuristic selection mechanism on top of the hill climbers can get an additional feedback from the previously utilized mutational heuristic.

## 3. Benchmark functions and heuristics

### *3.1. Benchmark functions*

Different benchmark problems that exhibit different and controllable characteristics are useful in measuring the performance of optimization algorithms, such as, genetic algorithms, memetic algorithms or hyper-heuristics. In this work, fourteen different benchmark functions, presented in Table 2, are selected from the literature. Each objective function evaluates a bit string that represents a candidate solution during the execution of an algorithm to locate the global optimum.

Table 2
Benchmark functions used during the experiments

| *Label* | *Name* | *Formula* | *Source* |
|---|---|---|---|
| F1 | Sphere | $f(\vec{x}) = \sum_{i=1}^{n} x_i^2$ | [15] |
| F2 | Rosenbrock | $f(\vec{x}) = \sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$ | [15] |
| F3 | Step | $f(\vec{x}) = 6 \cdot n + \sum_{i=1}^{n} \lfloor x_i \rfloor$ | [15] |
| F4 | Quartic *with noise* | $f(\vec{x}) = \sum_{i=1}^{n} (i \cdot x_i^4 + U(0,1))$ | [15], [43] |
| F5 | Foxhole | $f(\vec{x}) = \dfrac{1}{0.002 + \sum_{j=1}^{25} \dfrac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6}}$ | [15] |

$$a_{1j} = \begin{cases} -32 \rightarrow \mathrm{mod}(j,25) = 1 \\ -16 \rightarrow \mathrm{mod}(j,25) = 2 \\ 0 \rightarrow \mathrm{mod}(j,25) = 3 \\ 16 \rightarrow \mathrm{mod}(j,25) = 4 \\ 32 \rightarrow \mathrm{mod}(j,25) = 0 \end{cases}$$

$$a_{2j} = \begin{cases} -32 \rightarrow j > 0 \wedge j \le 5 \\ -16 \rightarrow j > 5 \wedge j \le 10 \\ 0 \rightarrow j > 10 \wedge j \le 15 \\ 16 \rightarrow j > 15 \wedge j \le 20 \\ 32 \rightarrow j > 20 \wedge j \le 5 \end{cases}$$

| | | | |
|---|---|---|---|
| F6 | Rastrigin | $f(\vec{x}) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2\pi x_i))$ | [39] |
| F7 | Schwefel | $f(\vec{x}) = 418.9829 \cdot n + \sum_{i=1}^{n} x_i \cdot \sin(\sqrt{\lvert x_i \rvert})$ | [41] |
| F8 | Griewangk | $f(\vec{x}) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod \cos(\frac{x_i}{\sqrt{i}}) + 1$ | [21] |
| F9 | Ackley | $f(\vec{x}) = 20 + e - 20 \cdot e^{-0.2 \cdot \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi x_i)}$ | [1] |
| F10 | Easom | $f(\vec{x}) = -(\prod_{i=1}^{n} \cos(x_i)) \cdot (e^{-\sum_{i=1}^{n} (x_i - \pi)^2})$ | [16] |
| F11 | Schwefel's Double Sum | $f(\vec{x}) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j^2)$ | [41] |
| F12 | Royal Road | $f(\vec{x}) = \sum_{s \in S} order(s) \sigma_s(\vec{x}),$ where $\sigma_s(\vec{x}) = \begin{cases} 1 & \text{if } \vec{x} \text{ is an instance of } s \\ 0 & \text{otherwise} \end{cases}$ and $s$ is a schema | [30] |
| F13 | Goldberg | (table below) $f(\vec{x}) = \sum_{i=1}^{n} Value(x_i)$, where $x_i$ is the $i^{\text{th}}$ 3-bit string | [18], [19] |
| F14 | Whitley | (table below) | [44] |

F13 Goldberg:

| | | | | |
|---|---|---|---|---|
| *String* | 000 | 001 | 010 | 011 |
| *Value* | 1 | 3 | 3 | 8 |
| *String* | 100 | 101 | 110 | 111 |
| *Value* | 5 | 8 | 8 | 0 |

$$f(\vec{x}) = \sum_{i=1}^{n} Value(x_i),$$

where $x_i$ is the $i^{\text{th}}$ 3-bit string

F14 Whitley:

| | | | | |
|---|---|---|---|---|
| *String* | 0000 | 0001 | 0010 | 0011 |
| *Value* | 2 | 4 | 6 | 12 |

| *String* | 0100 | 0101 | 0110 | 0111 |
|---|---|---|---|---|
| *Value* | 8 | 14 | 16 | 30 |
| *String* | 1000 | 1001 | 1010 | 1011 |
| *Value* | 10 | 18 | 20 | 28 |
| *String* | 1100 | 1101 | 1110 | 1111 |
| *Value* | 22 | 26 | 24 | 0 |

$$f(\vec{x}) = \sum_{i=1}^{n} Value(x_i)\ ,$$

where $x_i$ is the $i^{\text{th}}$ 4-bit string

The characteristics of these benchmark functions are well studied and explicit (Table 2, **Error! Reference source not found.**). The modality property indicates the number of optima in the search space (i.e. between bounds). Unimodal benchmark functions have a single optimum. Multimodal benchmark functions contain more than one optimum in their search space. Such functions contain at least one additional local optimum in which a search method can get stuck. In some of the functions, the evaluation process can be separated into a series of independent evaluations of each dimensional encoding. As an example, the sphere function is a separable function. This characteristic makes delta evaluation possible in the case of a localized modification within a dimension. For example, if a single bit flip occurs in a candidate solution, in order to calculate the overall fitness, there is no need for decoding all dimensions for a separable function. By remembering the fitness contribution of the previous value for the dimension, the overall fitness computation becomes a simple subtraction of the previous contribution and addition of the current contribution for the dimension in question to the overall fitness.

Table 3
Characteristics of the benchmark functions used during the experiments

| *label* | *range of* $x_i$ | *dimension* | *optimum* | *isContinuous* | *isSeparable* | *isMultimodal* |
|---|---|---|---|---|---|---|
| F1 | -5.12,5.12 | 10 | 0 | yes | yes | no |
| F2 | -2.048,2.048 | 10 | 0 | yes | yes | no |
| F3 | -5.12,5.12 | 10 | 0 | yes | yes | no |
| F4 | -1.28,1.28 | 10 | 1 | yes | yes | yes |
| F5 | -65.536,65.536 | 2 | 1 | yes | no | yes |
| F6 | -5.12,5.12 | 10 | 0 | yes | yes | yes |
| F7 | -500,500 | 10 | 0 | yes | yes | yes |
| F8 | -600,600 | 10 | 0 | yes | no | yes |
| F9 | -32.768,32.768 | 10 | 0 | yes | no | yes |
| F10 | -100,100 | 6 | -1 | yes | no | no |
| F11 | -65.536,65.536 | 10 | 0 | yes | no | no |
| F12 | n/a | 8 | 0 | no | yes | n/a |
| F13 | n/a | 30 | 0 | no | yes | n/a |
| F14 | n/a | 6 | 0 | no | yes | n/a |

The benchmark set consists of continuous and discrete functions. Gray encoding is used to represent multidimensional candidate solutions for the continuous benchmark functions. Royal Road, Goldberg's 3 bit Deceptive Function and Whitley's 4 bit Deceptive Function are the discrete functions, whereas the rest are continuous functions. The deceptiveness of Goldberg and Whitley functions are due to the large hamming distance between the local optima and the global optimum.

### *3.2. Heuristics*

Four hill climbers are utilized within both hyper-heuristics and memetic algorithms: Steepest Descent (SDHC), Next Descent (NDHC), Davis' Bit Hill Climbing (DBHC), Random Mutation Hill Climbing (RMHC). Three different mutational heuristics are implemented to be used within hyper-heuristics: Mutation (MUTN), Swap Dimension (SWPD), Dimensional Mutation (DIMM) and Hyper-mutation (HYPM).

SDHC generates all possible neighboring solutions from a given candidate solution within a Hamming distance of 1 by inverting each bit. If an improvement is achieved, then the solution that has the best fitness replaces the current one. In a single step of NDHC, the bit in question of a candidate solution is inverted. If this modification generates an improvement, the new solution is accepted as the current candidate solution and the iteration continues with the neighboring bit. In the implementation, NDHC scans the whole candidate solution starting from the most significant bit towards the least significant bit. Consecutive bit inversions are repeated for a factor of the bit-string length. DBHC is similar to NDHC. The order of inversions is predetermined randomly as a permutation of indices of the bit-string in DBHC. In a RMHC step, a random bit is chosen and inverted. Similarly, the modified candidate solution becomes the current candidate solution, if the fitness improves.  More details on these hill climbers can be found in [13] and [30]. Each hill climber, SDHC, NDHC, RMHC and DBHC will be referred as MA0, MA1, MA2, MA3, respectively, if used within a memetic algorithm.

MUTN scans a candidate solution once and flips a bit with a given *mutation probability*. SWPD swaps two different regions in a candidate solution each corresponding to a randomly selected dimension. DIMM inverts each bit in a randomly selected dimension with a probability of 0.5. HYPM is a restart operator representing random walk. Each bit in a candidate solution is inverted with a probability of 0.5.

## **4. Memetic algorithms**

Memetic algorithms (MAs) extend genetic algorithms (GAs) [20], [23] by embedding hill climbing as illustrated in Figure 2. Many researchers underlined the effectiveness of hill climbing when hybridized with GAs [31]. *Meme* is defined as a "contagious" piece of information in [14]. A meme is processed, understood, adapted and spread by each infected person. This meme transmission process carries some similarities with local improvement. Therefore, GAs using hill climbing are referred to as memetic algorithms, in which a meme denotes a specific hill climbing algorithm.

MAs are based on the principals of the Darwinian evolution and population genetics and they have been applied to various search and optimization problems [31], [34], [35], [37], [38].

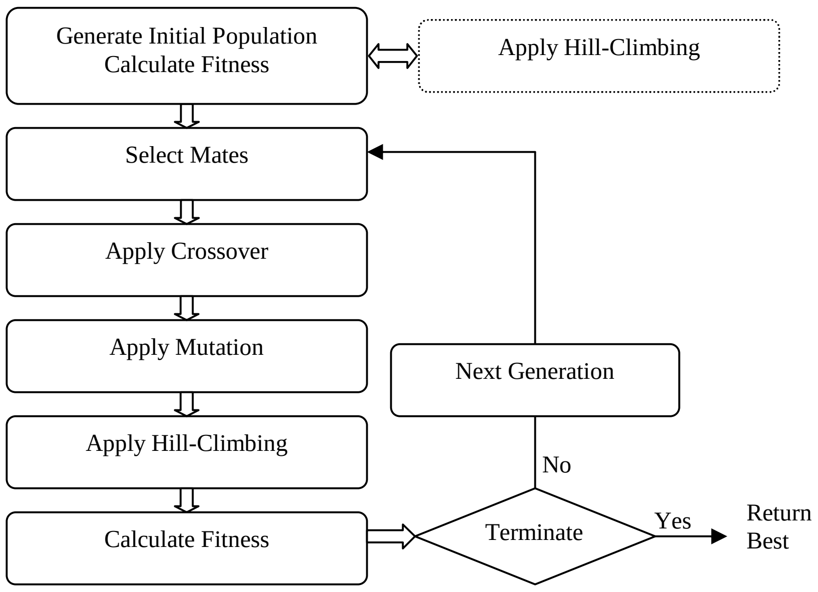


Fig. 2. A generic memetic algorithm.

In a generic MA (Figure 2), a candidate solution to an optimization problem at hand, called *chromosome* or *individual*, is represented using a binary bit string. The length of the binary string is referred to as *chromosome length*. Each locus on a chromosome is referred as *gene* that encodes a related parameter of the problem.  Each gene receives a value from the *allele* set {0, 1}. For example, assuming that the problem dealt with requires a binary encoding of length six, "101001" might represent an individual. Different representations can be used for encoding a candidate solution for a given problem. The search for an optimal solution is carried out on a set of individuals, called a *population*. The initial population is generated randomly and the quality of individuals is computed using a *fitness* (evaluation) *function*. If preferred, a hill climbing method can be applied to the initial population. In an evolutionary cycle, the population undergoes a set of genetic operators. First, *mates* are selected, and then selected mates are allowed to reproduce generating new candidate solutions. Mate selection is not a random process. A mechanism is used to ensure that a solution with a *better* fitness has a higher chance to get selected as a mate. As an example, in *tournament mate selection*, randomly selected group of individuals compete based on their fitness values and the winner is selected as a mate. The size of the group is referred as *tour-size*. The *crossover* operator exchanges genetic material between selected mates with a given probability. One-point crossover (1PTX) divides two mates into two sub-parts at a randomly selected cut point and concatenates the left and right-hand side parts each from different mates, generating two new candidate solutions, named as *offspring*. Assuming that the cut point is three, the application of 1PTX to the selected mates "101|001" and "000|111" results with two offspring: "101111" and "000001" After the crossover, they are *mutated*. The mutation operator inverts a bit in a candidate solution with a given mutation probability. For example, assuming a mutation probability of 1/6, "101111" and "000001" might be mutated into"101101" and "010011", respectively. Note that the choice of the mutation probability as 1/chromosome-length implies that on average a single bit will be inverted. In a generic MA, the hill-climbing operator is applied to the individuals, right after the mutation. This evolutionary cycle terminates whenever a set of criteria is satisfied. Finally, then the best candidate solution in the last population is returned. In this paper, all utilized MAs are generic MAs.

## *4.1. Coordinating a set of memes in memetic algorithms*

Adaptation issues in GAs have been discussed by many researchers, [9], [15]. In [42], a review of self-adaptive operators and parameter adaptation is provided. In a traditional MA, a single hill climbing algorithm is used. It is possible that more than one hill climbing algorithm can be devised to be used within the MAs for solving a problem. In such a case, either each MA using a different hill climber can be tested over a set of problem instances to determine the hill climber with the best mean performance or an adaptive and a self-adaptive strategy that chooses the hill climber to invoke during the hill climbing step can be utilized to coordinate all hill climbers for an improved performance.

Multimeme Memetic Algorithms (MMAs) represent a set of self-generating (co-evolving) MAs as discussed in detail in [25], [27], [29]. A meme (-plex) is defined in a wider spectrum in [26]. In this study, additional to the chromosome, each individual carries a meme which is basically an identifier for the hill climber that will be invoked during the hill climbing step. The meme of each individual is randomly generated for the initial population. Then genetic and memetic materials are co-evolved. Assuming that “101001+M2” represents an individual, “101001” is the genetic material and “M2” is the memetic material. “M2” indicates that whenever the hill climbing is invoked, $2^{nd}$ hill climber is to be utilized. The crossover and mutation processes proceed in the traditional way for the genetic material as described in the previous section. Memes also go through the inheritance and mutation processes. A meme is inherited to the offspring during the crossover by a mechanism proposed in [28], called as *Simple Inheritance Mechanism* (SIM). SIM passes on the meme of the mate with a better fitness to both offspring. If the fitness of the mates is the same, then a random meme is transmitted to each offspring. Applying 1PTX to the mates “101|001+M1” and “000|111+M2” yields two offspring: “101111+M2” and “000001+M2”, assuming that the former mate has a better fitness as compared to the latter one. During the mutation, a random hill climber is selected as a new meme from the set of hill climbers with a probability, called the *innovation rate* (IR). Each hill climber has an equal chance of being chosen. The details of the multimeme approach along with related test results of applying it on different theoretical and real world problem instances can be found in [25],[26]. The adaptation ability of MMA on the dynamic OneMax problem is tested in [28] and it has been observed that MMA successfully tracked changes in this dynamic environment. In another study [33], tests are conducted on three benchmark functions using two new adaptive methods that they proposed for selecting the appropriate meme within the MAs. Their biased roulette wheel strategy turned out to be the most viable one.

The self-adaptive multimeme approach can be also used within the other heuristics, meta-heuristics and even hyper-heuristics. This is still an open research area. Moreover, remembering that hyper-heuristics perform a search on a set of heuristics, they represent a set of highly promising strategies that can be used to manage a set of hill climbers within the MAs as well.

## 5. Experimental results

The experiments carried out can be roughly divided into three groups. The analysis carried out in this study has been based on different dimensions. As noted in the first section, the first aim of the work is to propose a comprehensive comparison between the selection mechanisms and acceptance criteria that have been used in the literature. Hence, the first set of experiments presented in this section try to provide an overall picture related to the selection and acceptance alternatives. Then, the notion of problem independence is analyzed for hyper-heuristics by experimenting on different frameworks. This forms the second set of experiments in this section. Lastly, the performance of hyper-heuristics is tested against other meta-heuristics throughout the last set of experiments. Additionally, the idea of using the hyper-heuristic approach to utilize the memes in MAs is tested.

Unless mentioned, the experiments are performed in a laboratory, denoted as *Lab*#1, on Pentium IV 2 GHz Linux machines having 256 Mb memories. The second laboratory used during the experiments, denoted as *Lab*#2, is equipped with Pentium IV 3 GHz Windows machines having 2 Gb memories. Fifty runs are performed during each test on a benchmark function. For a fair comparison between all algorithms, the experiments are terminated if the number of evaluations exceeds $3x10^6$ in 600 CPU seconds or the expected global optimum is achieved.

### *5.1 Evaluation criteria*

Following evaluation criteria are used during the comparisons of the algorithms. *Success rate, s.r.*, is the ratio of successful runs in which the expected fitness is achieved to the total number of runs. The average number of fitness evaluations is used as the performance criterion for the experiments with *full success* (s.r.=1.00). Moreover, the *rank* of an algorithm is determined with respect to the success rates of algorithms that are in comparison considering the ties.

While assessing hyper-heuristics, *utilization rate* and *acceptance rate* are also used. *Utilization rate* is the ratio of the number of moves of a specific heuristic to the overall number of moves in a run. Let (*move*) *acceptance rate* denote the ratio of the number of accepted moves of a heuristic to the total number of moves made by the same heuristics in a run. Utilization rate of a heuristic indicates how many times a given heuristic is selected by the hyper-heuristic, while the acceptance rate indicates what percentage of such moves is accepted. The acceptance rate might not be informative for some hyper-heuristics. For example, for any hyper-heuristic using AM acceptance criterion the acceptance rate is 1.00 for all heuristics used. Similarly, if a hyper-heuristic uses hill climbers only as heuristics and IE as an acceptance criterion, then the acceptance rate again will be 1.00 for all hill climbers.

Additionally, *average evolutionary activity*, obtained during an experiment is considered for the evaluation of the memes used within an MMA. *Evolutionary activity* is the total number of emergences of a specific meme within each individual between the initial generation and the current one during a run. Average evolutionary activity is achieved by averaging the evolutionary activity over a number of runs at each generation. It is a monotonically increasing function. The slope of the average evolutionary activity versus generation plot shows the degree of preference to a meme. The steeper the slope is for a meme, the more it is invoked.

### 5.2. *Comparison of simple hyper-heuristics*

The first set of experiments is performed to compare simple hyper-heuristics to determine the best one. The $F_B$ framework is used. The heuristic set contains {SWPD, DIMM, HYPM, MUTN, NDHC, SDHC, RMHC, DBHC} and as a hill climber DBHC is chosen.

Experimental results show that there is almost no significant difference among the performance of heuristic selection methods. Yet, IE as an acceptance criterion performed better when compared to the rest of the move acceptance criteria. The average performance of CF_IE pair as a hyper-heuristic is slightly better than the rest of the hyper-heuristics as shown in Table 4.

Table 4
Best performing heuristic selection-acceptance criterion combination(s) for each problem instance; * indicates that there are other hyper-heuristics generating a similar performance

| *Label* | *Best Hyper-heuristic*(*s*) |
|---|---|
| F1 | RP_OI, CF_IE* |
| F2 | SR_OI, CF_IE* |
| F3 | RD_IE, CF_IE* |
| F4 | RD_IE, CF_IE* |
| F5 | TABU_OI, CF_IE* |
| F6 | CF_MC* |
| F7 | CF_IE* |
| F8 | CF_MC* |
| F9 | CF_IE* |
| F10 | CF_IE* |
| F11 | RPD_IE, CF_IE* |
| F12 | SR_IE, CF_IE* |
| F13 | CF_IE* |
| F14 | RP_OI, CF_IE* |

### 5.3. *Choice of meme in $F_B$ and $F_C$ hyper-heuristic frameworks*

In order to compare the performance of each meme in the set {NDHC, SDHC, RMHC, DBHC}, more experiments are carried out. Each meme is employed within the frameworks $F_B$ and $F_C$ using the hyper-heuristic CF_IE when working in the in-mode. As mutational heuristics {SWPD, DIMM, HYPM, MUTN} are used.

The choice of meme combined with the choice of mutational heuristics affects the performance of the hyper-heuristic drastically. Similar outcomes are obtained for both frameworks. The experimental results show that DBHC and RMHC are the best meme choices for both $F_B$ and $F_C$ as illustrated in Table 5.

Table 5
The rank of each meme utilized within *in-mode* $F_B$ and $F_C$ for each benchmark function.

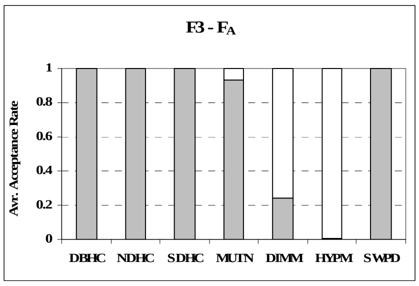
| | *$F_B$* | | | | *$F_C$* | | | |
|---|---|---|---|---|---|---|---|---|
| *La-bel* | *SDHC* | *NDHC* | *RMHC* | *DBHC* | *SDHC* | *NDHC* | *RMHC* | *DBHC* |
| F1 | 4 | 2 | 2 | 2 | 4 | 2 | 2 | 2 |
| F2 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 | 2.5 |
| F3 | 4 | 2 | 2 | 2 | 4 | 2 | 2 | 2 |

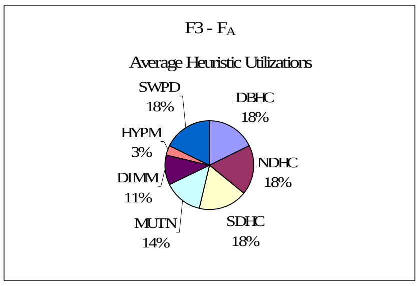
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| F4 | 3.5 | 3.5 | 1 | 2 | 3 | 3 | 1 | 3 |
| F5 | 4 | 3 | 1.5 | 1.5 | 4 | 3 | 1.5 | 1.5 |
| F6 | 4 | 3 | 2 | 1 | 3.5 | 3.5 | 1.5 | 1.5 |
| F7 | 4 | 3 | 1.5 | 1.5 | 4 | 3 | 1.5 | 1.5 |
| F8 | 4 | 1 | 3 | 2 | 4 | 1 | 3 | 2 |
| F9 | 4 | 2 | 2 | 2 | 4 | 2 | 2 | 2 |
| F10 | 4 | 3 | 2 | 1 | 4 | 3 | 2 | 1 |
| F11 | 3.5 | 3.5 | 1.5 | 1.5 | 3.5 | 3.5 | 1.5 | 1.5 |
| F12 | 4 | 3 | 1.5 | 1.5 | 3.5 | 3.5 | 1.5 | 1.5 |
| F13 | 3.5 | 3.5 | 1.5 | 1.5 | 3.5 | 3.5 | 1.5 | 1.5 |
| F14 | 3 | 4 | 1.5 | 1.5 | 4 | 3 | 1.5 | 1.5 |
| *Avr.* | 3.71 | 2.79 | 1.82 | 1.68 | 3.68 | 2.75 | 1.79 | 1.79 |

NDHC performs significantly better than DBHC only for F8 in both frameworks. DBHC locates the global optimum for F2 in none of the trials in both frameworks. SDHC is the worst meme among all. The reason seems to be the choice of mutational heuristics. In particular, MUTN becomes useless, possibly because it does not introduce any new state that SDHC does not visit for a given candidate solution.

## *5.4. Evaluation of mutational heuristics*

During this set of experiments RMHC (MA2) is excluded from the heuristic set, increasing the number of mutational heuristics over the number of hill climbers. This way, three hill climbers with significantly different performances are kept to evaluate the performance of mutational heuristics. The results show that all mutational heuristics provide a positive contribution during the search while CF_IE hyper-heuristic is utilized, except HYPM. As an example, for F3, F7, F11 functions, within the $F_A$ and $F_B$, almost none of the moves by HYPM are accepted and it is the least utilized heuristic among all heuristics as illustrated in Figure 3. Hence, during the next set of experiments, HYPM is excluded from the mutational heuristics. Due to the IE acceptance mechanism, all candidate solutions generated by hill climbers are accepted.





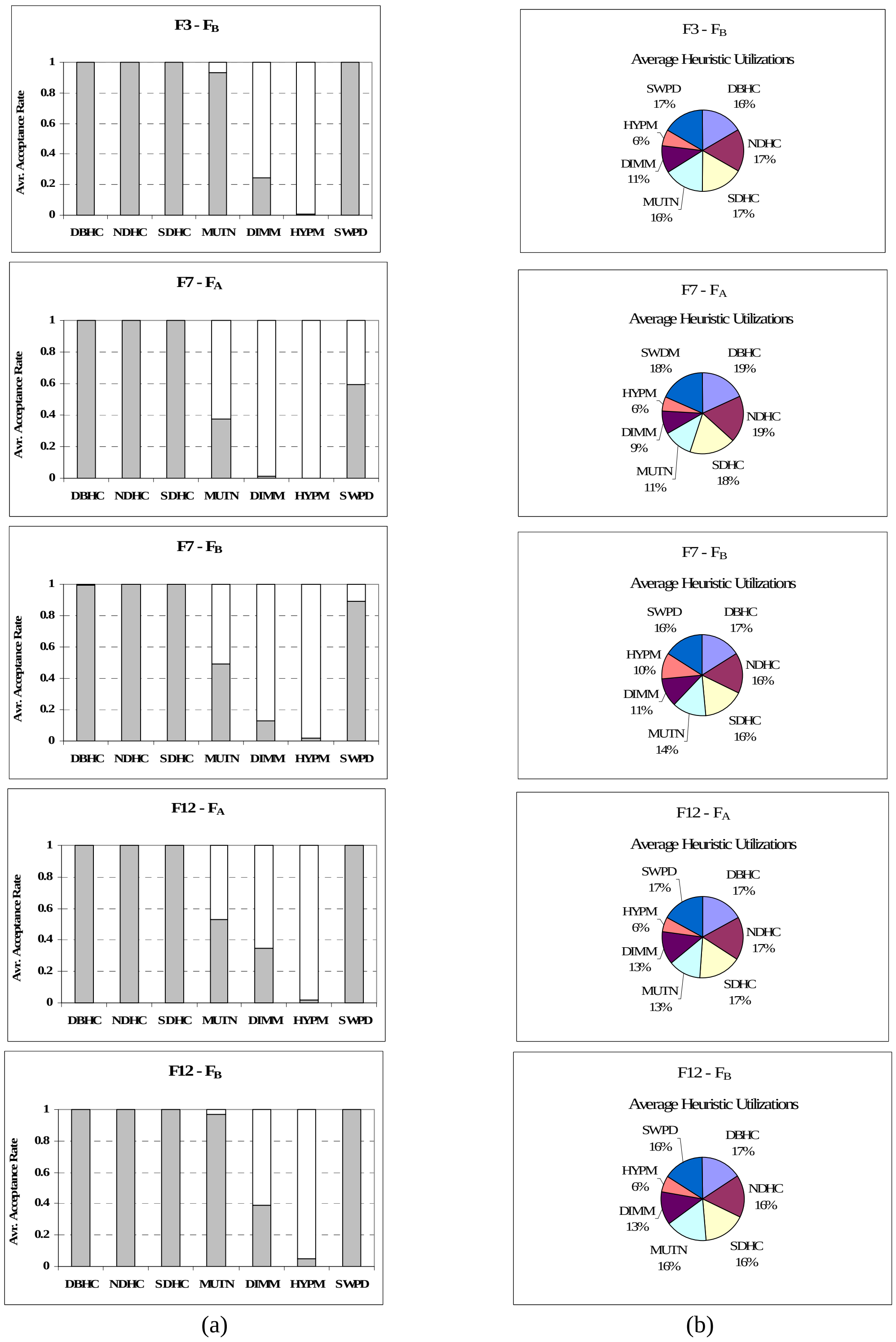


Fig. 3. Average acceptance rate and utilization of each heuristic within the frameworks $F_A$ and $F_B$ over fifty runs for a subset of benchmark functions

## *5.5. Comparison of hyper-heuristic frameworks*

This set of experiments is different then the ones reported in [36]. A reduced heuristic set, discarding the worst performing heuristics as determined by the previous experiments, is used within the frameworks described in Section 2.2. Mutational heuristic set contains {SWPD, DIMM, MUTN}, while the hill climbers consist of {NDHC, RMHC, DBHC}. $F_A$ and $F_B$ use the combined set of mutational heuristics and hill climbers. $F_B$ and $F_C$ employ DBHC as the single hill climber. $F_D$ uses CF_AM and CF_IE as a hyper-heuristic for the mutational heuristics and hill climbers, respectively. Experimental results presented in Table 6 show that $F_C$ performs significantly better as compared to the other frameworks considering the success rates.

Table 6
The success rate of each hyper-heuristic framework for each benchmark function.

| *Label* | $F_A$ | $F_B$ | $F_C$ | $F_D$ |
|---|---|---|---|---|
| F1 | 1.00 | 1.00 | 1.00 | 1.00 |
| F2 | 0.00 | 0.00 | 0.00 | 0.00 |
| F3 | 1.00 | 1.00 | 1.00 | 0.00 |
| F4 | 0.00 | 0.02 | 0.02 | 0.02 |
| F5 | 0.76 | 1.00 | 1.00 | 0.54 |
| F6 | 0.08 | 1.00 | 1.00 | 0.00 |
| F7 | 0.92 | 0.98 | 1.00 | 0.00 |
| F8 | 0.00 | 0.30 | 0.90 | 0.90 |
| F9 | 1.00 | 1.00 | 1.00 | 0.96 |
| F10 | 0.02 | 0.44 | 0.54 | 0.02 |
| F11 | 0.00 | 1.00 | 1.00 | 0.06 |
| F12 | 1.00 | 1.00 | 1.00 | 0.00 |
| F13 | 0.00 | 1.00 | 1.00 | 0.00 |
| F14 | 0.82 | 1.00 | 1.00 | 0.06 |
| *Avr.* | 0.47 | 0.77 | 0.82 | 0.25 |

The results obtained during this set of experiments confirm those obtained in the previous study, [36].The elimination of HYPM and SDHC from the heuristic set did not affect the performance of any related framework. Experiments are repeated for $F_D$ using SR_AM for the mutational heuristics on the benchmark functions without changing any other setting. This $F_D$ achieved almost the same performance as the previous one, but for F8 the latter choice yields the best success rate of 0.92 among all frameworks. Additionally, for $F_A$ another set of heuristics is used for comparison; {SWPD, DIMM, MUTN, DBHC}. Using this set did not improve the performance of the framework $F_A$ much with an average success rate of 0.49 over the benchmark functions.

## *5.6. Memetic algorithms versus CF_IE hyper-heuristic in $F_C$*

This set of experiments is performed in *Lab*#2 and a run is terminated if the execution time exceeds 600 CPU seconds or the expected global optimum is achieved. During a single iteration in a generic hyper-heuristic framework, either a mutational heuristic or a hill climber can be chosen to be applied to the current candidate solution, but in an MA, a hill climber is applied after mutation to each candidate solution. All GA and MA parameters are arbitrarily chosen with respect to the chromosome length *l* that is the product of dimensions and the number of bits used (

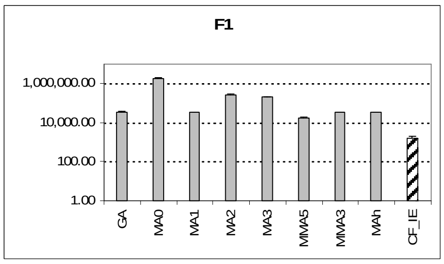
Table 7). All of them use a tournament mate selection strategy, one point crossover, traditional mutation and a trans-generational MA with a replacement strategy that keeps only two best individuals from the previous generation.

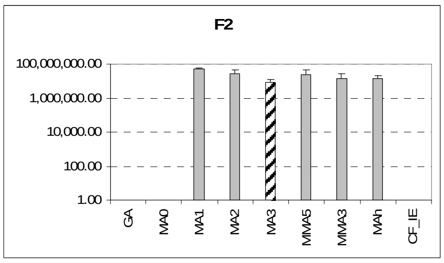
Table 7
Parameter-value pairs for the GA and MAs

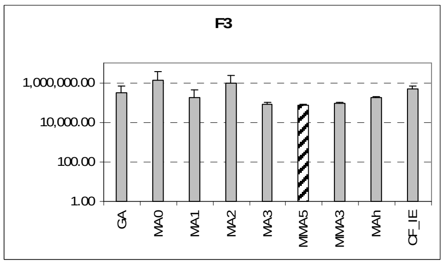
| *Parameter* | *Value* |
|---:|:---|
| Population Size = | max{*l*/5, 20} |
| Max. no. of hill climbing steps = | 2*l* |
| Tour-size = | 2 |
| Crossover probability = | 1.00 |
| Mutation probability = | 1/*l* |

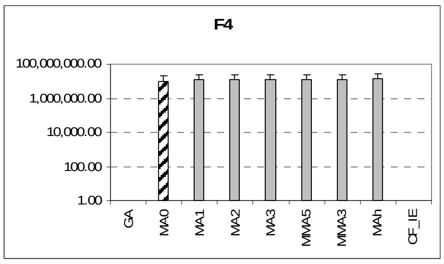
Experiments are performed in the following order. GA is compared to each MA utilizing a different meme from the set {MA0, MA1, MA2, MA3}. Additionally, MMA using five effective memes {GA, MA0, MA1, MA2, MA3} are tested for each benchmark function, denoted as MMA5. GA meme represents the case for which value the hill climbing step is skipped. The study in Ozcan and Basaran (2006) showed that using low number of memes might boost the performance of the multi-meme memetic algorithms. Hence, MMA3 is implemented with a reduced set of three memes {MA0, MA1, MA3}. The MAh denotes the MA that utilizes SR_IE hyper-heuristic to manage the memes {MA0, MA1, MA3}.

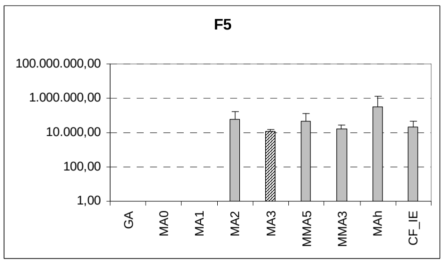
The experimental results are presented in Figure 4. Bars indicate the average number of evaluations required by each algorithm for the corresponding benchmark function. They appear only if full success is achieved during the runs. All memetic algorithms successfully locate the global optimum in all runs for each benchmark function. The experimental results show that MAs perform much better than the GA. Meme choice affects the performance of the MA as suggested in [32]. Considering the MAs using a single hill climber, MA3 performs significantly better than the rest of the memes when used as a single hill climber within the MA on average. MA2 has the worst performance among all memes. Hence, that's the reason why MA2 is not used within the MMAs and MAh. There is no statistically significant difference between the performances of MMA5, MA3, MMA3 and MAh. MA3 performs slightly better than MMA5. MMA3 performs slightly better than MA3 and MMA5 on average. Using a hyper-heuristic within an MA to manage a set of hill climbers also seems to be a viable strategy. MAh ranks first, when the average number of evaluations over all benchmark functions is considered. CF_IE seems to be successful in solving deceptive problems. Furthermore, the performance of the CF_IE in $F_C$ hyper-heuristic framework falls in between the MMA and GA with no significant variance from MAs.

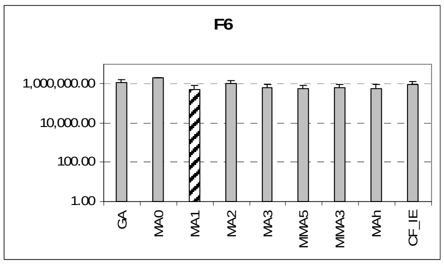

F1
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE

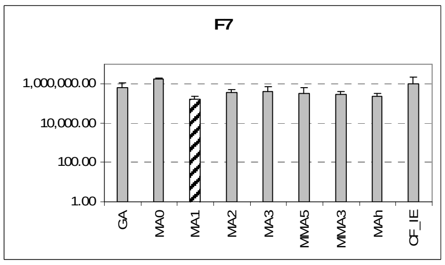

F2
100,000,000.00
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE

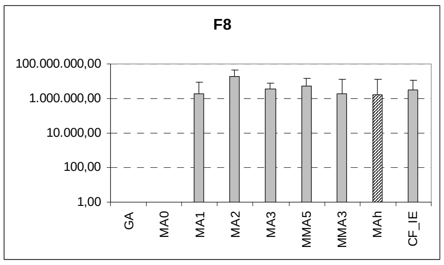

F3
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE

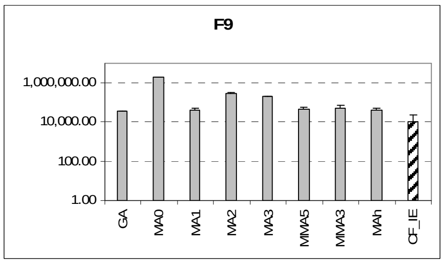

F4
100,000,000.00
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE

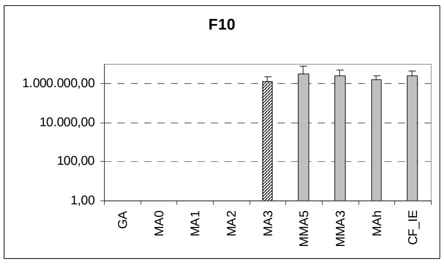

F5
100.000.000,00
1.000.000,00
10.000,00
100,00
1,00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE


F6
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE


F7
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE


F8
100.000.000,00
1.000.000,00
10.000,00
100,00
1,00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE


F9
1,000,000.00
10,000.00
100.00
1.00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE


F10
1.000.000,00
10.000,00
100,00
1,00
GA
MA0
MA1
MA2
MA3
MMA5
MMA3
MAh
CF_IE

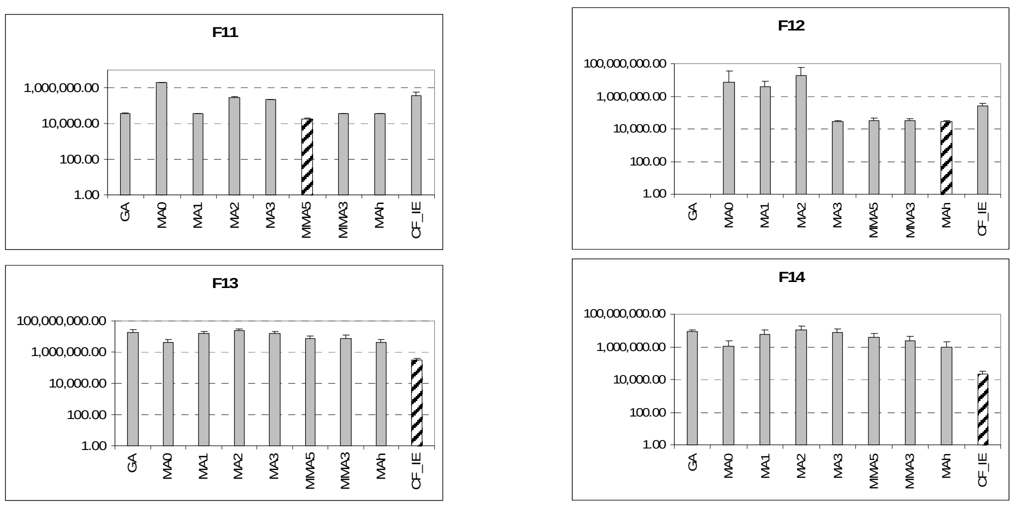
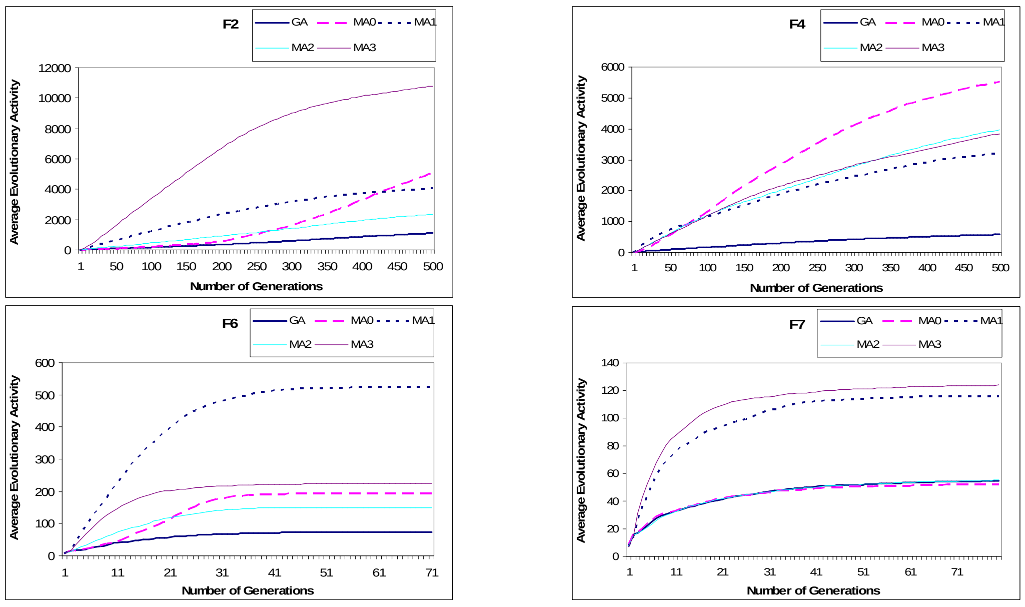
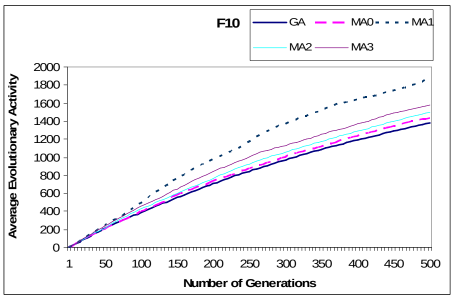


Fig. 4. Mean and the standard deviation of the number of evaluations per run in log scale, generated by each algorithm for each benchmark function

For six sample benchmark functions average evolutionary activity plots are provided in Figure 5. The MMA mechanism does not provide a synergy among a set of hill climbers, but instead, it successfully discovers the best meme to be utilized for the problem at hand. For example, Figure 4 shows that for F2, F4, F6 and F12, the best memes are MA3, MA0, MA1 and MA2, respectively. Figure 5 shows that the very same memes are utilized more than the rest in the MMA5. There are some cases another meme (other than the best meme) might be used more within the MMA. For example, for F7 and F10, the best memes are MA1 and MA3 respectively, but the MMA5 utilizes MA3 and MA1 more instead. Such cases seem to occur due to the small performance variances among the memes.





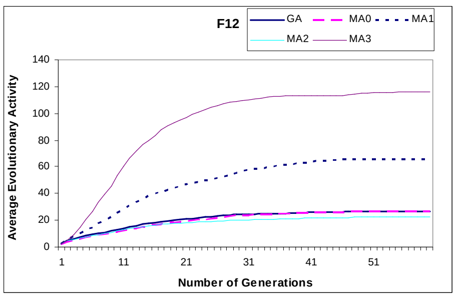


Fig. 5. Average evolutionary activity versus number of generations plot of each meme in the MMA5 for a selected subset of benchmark functions

## 6. Conclusions and Future Work

Generic hyper-heuristic iteration consists of selection and decision steps. A heuristic from a set is chosen using a problem independent performance measure. After employing the heuristic, a decision is made whether the new candidate solution will be accepted or not. In this process, experiments show that the acceptance mechanism significantly affects the performance as compared to heuristic selection. Based on the results obtained and previous studies, GD, MC and IE seem to be the best acceptance mechanism choices if a hyper-heuristic is to be used for solving a search and optimization problem. These acceptance mechanisms seem to perform well when combined with CF or SR heuristic selection mechanisms. For example, in [3], CF_MC and SR_GD produce the best performances for solving a set of examination timetabling problems.

Different hyper-heuristic frameworks perform differently. Proposed $F_C$ framework successfully combines mutational heuristics and a hill climber generating a significantly better performance as compared to the generic framework. The choice of meme also affects the performance of the proposed hyper-heuristic frameworks. The performance variances between the generic framework and $F_B$ and $F_C$ are significant. Hence, if there is a set of hill climbing algorithms for solving a problem at hand, the most suitable framework should be investigated. The heuristic set and the framework to be used within a hyper-heuristic might produce results worse than a GA, while the right choice might yield an improved performance even better than an MA.

The experimental results also confirm that the MAs perform significantly better as compared to GAs. The meme choice affects the performance of an MA. MMAs based on a Lamarckian learning mechanism are promising and by using the right set of memes; their performance can be improved over generic MAs using a single hill climber. If possible, useless memes should be identified and eliminated for an improved performance in an MMA. There is however a performance cost for allowing the MMA to make such eliminations.

The newly proposed MA scheme that suggests the use of a hyper-heuristic to manage a set of hill climbers also seems to be promising. When it is time for hill climbing, the hyper-heuristic makes the choice among the memes. During the experiments a non-deterministic heuristic selection is used, providing the best average performance. Additionally, some previous studies indicate that MMAs performs well in dynamic environments. Instead of using the GA as a hyper-heuristic mechanism, embedding hyper-heuristics into a GA for selecting the best operator and analyzing the behavior of hyper-heuristics in dynamic environments are the future research directions of this study.

## Acknowledgement

Authors thank Mustafa MISIR for his help during the hyper-heuristic framework experiments. This research is supported by TUBITAK (The Scientific and Technological Research Council of Turkey) under the grant number 105E027.

## References

[1] D. Ackley, *An Empirical Study of Bit Vector Function Optimization*, In Proceedings of Genetic Algorithms and Simulated Annealing, 1987, 170–215.

[2] 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, 2003, 132–141.

[3] B. Bilgin, E. Ozcan, E.E. Korkmaz, *An Experimental Study on Hyper-Heuristics and Final Exam Scheduling*, In Proceedings of the 2006 International Conference on the Practice and Theory of Automated Timetabling, 2006, 123–140.

[4] E. Burke and E. Soubeiga, *Scheduling Nurses Using a Tabu-Search Hyperheuristic*, In Proceedings of the MISTA I, Nottingham, vol. 1, 2003, 197–218.

[5] E.K. Burke, G. Kendall, J. Newall, E.Hart, P. Ross, and S. Schulenburg, Hyper-heuristics an Emerging Direction in Modern Search Technology, *Handbook of Metaheuristics* (eds Glover F. and Kochenberger G. A.), 2003, 457–474.

[6] E.K. Burke, G. Kendall, and E. Soubeiga, A Tabu-Search Hyper-heuristic for Timetabling and Rostering, *Journal of Heuristics* Vol 9, No. 6 (2003) 451–470.

[7] E.K. Burke, A. Meisels, S. Petrovic, and R. Qu, A Graph-Based Hyper Heuristic for Timetabling Problems, *European Journal of Operational Research* (2007) 176:177–192.

[8] E. K. Burke S. Petrovic, and R. Qu, Case Based Heuristic Selection for Timetabling Problems, *Journal of Scheduling*, Vol.9 No2 (2006) 1094–6136.

[9] H.G. Cobb, *An investigation into the use of hypermutation as an adaptive operator in Genetic Algorithms Having Continuous, Time-dependent Nonstationary Environment*, NRL Memorandum Report 6760, 1990.

[10] P. Cowling, K. Chakhlevitch, *Hyperheuristics for managing a large collection of low level heuristics to schedule personnel*, In Proceedings of the Congress on Evolutionary Computation, vol.2, 2003, 1214–1221.

[11] P. Cowling, G. Kendall, and E. Soubeiga, A Hyper-heuristic Approach to Scheduling a Sales Summit, *LNCS* 2079, PATAT III, Konstanz, Germany, selected papers (eds Burke E.K. and Erben W), 2000, 176–190.

[12] A. Cuesta-Cañada, L. Garrido and H. Terashima-Marín, Building Hyper-heuristics Through Ant Colony Optimization for the 2D Bin Packing Problem, *LNCS* 3684, 2005, 654–660.

[13] L. Davis, *Bit Climbing, Representational Bias, and Test Suite Design*, In Proceedings of the 4th Int. Conference on Genetic Algorithms, 1991, 18–23.

[14] R. Dawkins, *The Selfish Genes*, Oxford University Press, 1976.

[15] K. De Jong, *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan, 1975.

[16] E.E. Easom, *A Survey of Global Optimization Techniques*. M. Eng. thesis, Univ. Louisville, Louisville, KY, 1990.

[17] A. Gaw, P. Rattadilok and R.S.K. Kwan, *Distributed Choice Function Hyperheuristics for Timetabling and Scheduling*," In Proceedings of the 5th International Conference on the Practice and Theory of Automated Timetabling, 2004, 495–498.

[18] D. E. Goldberg, Genetic Algorithms and Walsh Functions Part I, A Gentle Introduction, *Complex Systems* (1989) 129–152.

[19] D. E. Goldberg, Genetic Algorithms and Walsh Functions Part II, Deception and Its Analysis, *Complex Systems* (1989) 153–171.

[20] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading (MA), 1989.

[21] A.O. Griewangk, Generalized Descent of Global Optimization, *Journal of Optimization Theory and Applications*, (1981) 34: 11–39.

[22] E. Hart, P.Ross, J. Nelson, Solving a Real-World Problem Using an Evolving Heuristically Driven Schedule Builder, *Evolutionary Computation* 6 (1), 1998, 61–80.

[23] J.H. Holland, *Adaptation in Natural and Artificial Systems*, Univ. Mich. Press, 1975.

[24] G. Kendall, and M. Mohamad, *Channel Assignment in Cellular Communication Using a Great Deluge Hyper-heuristic*, In Proceedings of the IEEE International Conference on Network, 2004, 769–773.

[25] N. Krasnogor, and S. Gustafson, A Study on the use of "Self-Generation" in Memetic Algorithms, *Natural Computing*, vol 3. no 1 (2004): 53–76.

[26] N. Krasnogor, *Studies on the Theory and Design Space of Memetic Algorithms*, PhD Thesis, University of the West of England, Bristol, UK, 2002.

[27] N.Krasnogor and J.E. Smith, *Multimeme Algorithms for the Structure Prediction and Structure Comparison of Proteins*, In Proceedings of the Bird of a Feather Workshops, GECCO, 2002, 42–44.

[28] N.Krasnogor and J.E. Smith, *Emergence of Profitable Search strategies Based on a Simple Inheritance Mechanism*, In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, 2001, 432–439.

[29] N.Krasnogor and J.E. Smith, *A Memetic Algorithm With Self-Adaptive Local Search: TSP as a case study*, In Proceedings of the Genetic and Evolutionary Computation Conference, GECCO, 2000, 987–994.

[30] M. Mitchell, and S. Forrest, Fitness Landscapes Royal Road Functions, *Handbook of Evolutionary Computation*, Baeck, T., Fogel, D., Michalewiz, Z., (Ed.), Institute of Physics Publishing and Oxford University, 1997.

[31] P. Moscato, and M. G. Norman. A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems, *Parallel Computing and Transputer Applications*, 1992, 177–186.

[32] Z. Ning, , Y. S. Ong, K. W. Wong, and M. H. Lim, *Choice of Memes In Memetic Algorithm*, In Proceedings of the 2nd International Conference on Computational Intelligence, Robotics and Autonomous Systems, 2003.

[33] Y. S. Ong, and A.J. Keane, Meta-Lamarckian Learning in Memetic Algorithms, *IEEE Trans. Evolutionary Computation*, vol. 8, no. 2, (2004) 99–110.

[34] E. Ozcan, *An Empirical Investigation on Memes, Self-generation and Nurse Rostering,* In Proceedings of the 6th Int. Conf. on PATAT 2006, 246–263.

[35] E. Ozcan, Memetic Algorithms for Nurse Rostering, *LNCS* 3733, The 20th ISCIS, 2005, 482–492.

[36] E. Ozcan, B. Bilgin, and E.E. Korkmaz, Hill Climbers and Mutational heuristic, *LNCS* 4193, PPSN IX, 2006, 202–211.

[37] E. Ozcan, and C. Basaran, *A Case Study of Memetic Algorithms for Constraint Optimization: Multidimensional 0-1 Knapsack Problem*, CSE-2006-01, technical report, 2006.

[38] E. Ozcan, and E. Onbasioglu, Memetic Algorithms for Parallel Code Optimization, *International Journal of Parallel Processing*, vol. 35, no. 1 / February (2007) 33–61.

[39] L. A. Rastrigin, Extremal Control Systems, In *Theoretical Foundations of Engineering Cybernetics Series*, Moscow, Nauka, Russian, 1974.

[40] P. Ross, J.G. Marin-Blazquez, E. Hart, Hyper-heuristics applied to class and exam timetabling problems, In Proceedings of the Congress on Evolutionary Computation, vol.2, 2004, 1691–1698.

[41] H. P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons (1981), English translation of Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie, 1977.

[42] J. E. Smith, and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms, *Soft Computing*. vol 1. no 2 (1997): 81–87.

[43] D. Tasoulis, N. Pavlidis, V. Plagianakos and M. Vrahatis, *Parallel Differential Evolution*, In Proceedings of the IEEE Congress on Evolutionary Computation, 2004, 2023–2029.

[44] D. Whitley, Fundamental Principles of Deception in Genetic Search, In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, Morgan Kaufmann, San Matco, CA, 1991.

[45] D. Wolpert, and W.G. MacReady, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, 1(1), 1997, 67–82.