Chapter III

Novel Enhanced Bio Inspired Harmony Search Algorithms for Solving Continuous Optimization Problems
CHAPTER 3
NOVEL ENHANCED BIO INSPIRED HARMONY SEARCH ALGORITHMS FOR SOLVING CONTINUOUS OPTIMIZATION PROBLEMS

3.1 INTRODUCTION

Global optimization is the branch of applied mathematics and numerical analysis that focuses on, well, optimization. The goal of global optimization is to find the best possible elements \( x^* \) from a set \( X \) according to a set of criteria \( F = \{ f_1, f_2, \ldots, f_n \} \). These criteria are expressed as mathematical functions, the so-called objective functions.

**Objective Function:** An objective function \( f: X \rightarrow Y \) with \( Y \subseteq \mathbb{R} \) is a mathematical function which is subject to optimization.

The codomain \( Y \) of an objective function as well as its range must be a subset of the real numbers (\( Y \subseteq \mathbb{R} \)). The domain \( X \) of \( f \) is called problem space and can represent any type of elements like numbers, lists, construction plans, and so on. It is chosen according to the problem to be solved with the optimization process. The objective of any Global optimization is to find the best elements \( x^* \) in \( X \) with respect to such criteria \( f \in F \) [39].

3.2 CLASSIFICATION OF OPTIMIZATION METHODS

This section classifies the optimization methods according to methods of operation and properties.

3.2.1 According to Methods of Operation

Generally, optimization algorithms can be divided into two basic classes: Deterministic and Probabilistic algorithms. Deterministic algorithms are most often used if a clear relation between the characteristics of the possible solutions and their utility for a given problem exist. Then, the search space can efficiently be explored
using, for example, a divide and conquer scheme [10]. Figure 3.1 sketches a rough taxonomy of global optimization methods.

**Determinism:** In each execution step of a deterministic algorithm, there exists at most one way to proceed. If there is no way to proceed, the algorithm will be terminated.

If the relation between a solution candidate and its “fitness” is not so obvious or too complicated or if the dimensionality of the search space is very high, it becomes harder to solve a problem deterministically. Trying it would possibly result in exhaustive enumeration of the search space, which is not feasible even for relatively small problems [39]. There, probabilistic algorithms come into play.

**Randomized Algorithm:** A randomized algorithm includes at least one instruction that acts on the basis of random numbers. In other words, a randomized algorithm violates the constraint of determinism. Randomized algorithms are also often called probabilistic algorithms.

An especially relevant family of probabilistic algorithms are the metaheuristic approaches. They trade in guaranteed correctness of the solution for a shorter runtime. One of the attractive features of metaheuristics is its simplicity and robustness. They can be developed even if deep mathematical properties of the problem domain are not at hand, and they still can provide reasonably good solutions, much better than those obtainable by simple heuristics. Moreover, in many cases, their performance will not degrade much even if the problem structures and/or parameters of algorithms are changed.
Fig. 3.1 A Taxonomy of Global Optimization Methods
**Metaheuristic:** A metaheuristic is a method for solving very general classes of problems. It is a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality [72].

These metaheuristic algorithms are based on the iteration of the following two steps:

1. Search new solutions on the basis of the previous history, and

2. Evaluate the solutions generated in Step 1 and extract necessary information for the future search.

Therefore, metaheuristics can be considered as the collection of ideas of how to use the search history to generate new solutions and how to extract the necessary information from the generated solutions.

The term meta-heuristic was coined by Fred Glover in 1986 [24] and has been widely applied in the literature. An important class of probabilistic metaheuristics is Evolutionary Computation. It encompasses all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. The search strategies proposed by metaheuristic methodologies result in iterative procedures with the ability to escape local optimal points. Metaheuristics have been developed to solve complex optimization problems in many areas especially with combinatorial optimization problems.

### 3.2.2 According to Properties

Speed and Precision are two conflicting objectives of any optimization algorithm. A general rule of thumb is that improvement in accuracy can be gained in optimization only by investing more time. When time is taken as a constraint, two main types of optimization viz., Online Optimization and Offline Optimization are considered [72].

**Online Optimization:** Online optimization problems are tasks that need to be solved quickly in a time span between ten milliseconds to a few minutes. In order to find a solution in this short time, optimality is normally traded in for speed gains.
Examples for online optimization are robot localization, load balancing, services composition for business processes and updating a factory’s machine job schedule after new orders came in. The online optimization tasks are often carried out in a way that minimizes the computation time in getting the result.

**Offline Optimization:** In offline optimization problems, time is not so important and a user is willing to wait even for days in order to get an optimal or close-to-optimal result.

Data mining and creating long-term schedules for transportation crews are examples for Offline Optimization problems. These optimization processes are usually be carried out so as to maximize accuracy of the result.

### 3.3 HARMONY SEARCH

Harmony search is a music-based metaheuristic optimization algorithm. It was inspired by the observation that the aim of music is to search for a perfect state of harmony [19]. This harmony in music is analogous to find optimality in an optimization process. The search process in optimization can be compared to a jazz musician’s improvisation process. On the one hand, the perfectly pleasing harmony is determined by the audio aesthetic standard. A musician always intends to produce a piece of music with perfect harmony. Similarly, an optimal solution to an optimization problem should be the best solution available to the problem under the given objectives and limited by constraints. Both processes intend to produce the best or optimum.

Such similarities between two processes can be used to develop new algorithms by learning from each other. Harmony Search is just such a successful example by transforming the qualitative improvisation process into some quantitative rules by idealization, and thus turning the beauty and harmony of music into an optimization procedure through search for a perfect harmony, namely, the Harmony Search (HS) or Harmony Search algorithm. In the HS algorithm, each musician (decision variable) plays (generates) a note (a value) for finding a best harmony (global optimum) altogether.

The HS is good at identifying the high performance regions of solution space within a reasonable time. But it is not efficient in performing local search in numerical
optimization applications [47,56]. Thus a few variants were developed for enhancing solution accuracy and convergence rate. Mahdavi et al. [47] presented an improved HS algorithm, denoted as IHS, by introducing a strategy to dynamically tune the key parameters, whereas Omran and Mahdavi [54] proposed a global best HS algorithm, denoted as GHS, by borrowing the concept from swarm intelligence. Their performance has revealed that both improved variants could find better solutions when compared to the basic HS algorithm. Particularly, the GHS outperformed the IHS algorithm [54].

The steps in the procedure of harmony search are shown below:

Step 1. Initialize the problem and algorithm parameters

Step 2. Initialize the harmony memory

Step 3. Improvise a new harmony

Step 4. Update the harmony memory

Step 5. Check the stopping criterion

These steps are described in the next five subsections.

(1) **Initialize the problem and algorithm parameters**

In this step, the optimization problem is specified as follows:

Minimize $f(x)$ subject to $x_i \in X_i$, $i = 1, 2, \ldots, N$

where $f(x)$ is an objective function; $x$ is the set of each decision variable ($x_i$); $N$ is the number of decision variables, $X_i$ is the set of the possible range of values for each decision variable, that is $L_{x_i} \leq X_i \leq U_{x_i}$ and $L_{x_i}$ and $U_{x_i}$ are the lower and upper bounds for each decision variable. The HS algorithm parameters are also specified in this step. They are the Harmony Memory Size (HMS), or the number of solution vectors in the harmony memory; Harmony Memory Considering Rate (HMCR); Pitch Adjusting Rate (PAR); and the Number of Improvisations (NI), or stopping criterion. The Harmony Memory (HM) is a memory location where all the solution vectors (sets of decision variables) are stored. HMCR and PAR are used to improve the solution vector which are defined in Step 3.
(2) Initialize the harmony memory

In this step, the HM matrix is filled with as many randomly generated solution vectors as the HMS:

\[
\text{HM} = \begin{bmatrix}
    x_1^1 & x_2^1 & \cdots & x_{N-1}^1 & x_N^1 \\
    x_1^2 & x_2^2 & \cdots & x_{N-1}^2 & x_N^2 \\
    \vdots & \vdots & \ddots & \vdots & \vdots \\
    x_1^{HM-1} & x_2^{HM-1} & \cdots & x_{N-1}^{HM-1} & x_N^{HM-1} \\
    x_1^{HM} & x_2^{HM} & \cdots & x_{N-1}^{HM} & x_N^{HM}
\end{bmatrix}
\]

(3) Improvise a new harmony

A new harmony vector, \( x' = (x_1', x_2', \ldots, x_{N-1}', x_N') \), is generated based on three rules:

1. Memory consideration
2. Pitch adjustment
3. Random selection

Generating a new harmony is called ‘improvisation’. In the memory consideration, the value of the first decision variable \( x_1' \) for the new vector is chosen from any of the values in the specified HM range \( x_1^{1} - x_1^{HM} \). Values of the other decision variables \( x_2', \ldots, x_{N-1}', x_N' \) are chosen in the same manner. The HMCR, which varies between 0 and 1, is the rate of choosing one value from the historical values stored in the HM, while \( (1 - \text{HMCR}) \) is the rate of randomly selecting one value from the possible range of values.

\[
x_i' \leftarrow \begin{cases}
    x_i' \in \{ x_1^i, x_2^i, \ldots, x_i^{HM} \} \text{ with probability } \text{HMCR}, \\
    x_i' \in X_i \text{ with probability } (1-\text{HMCR})
\end{cases}
\]
Every component obtained by the memory consideration is examined to determine whether it should be pitch-adjusted. This operation uses the PAR parameter, which is the rate of pitch adjustment as follows:

\[
\text{Pitch adjusting decision for } x_i' \leftarrow \begin{cases} 
\text{Yes with probability PAR} \\
\text{No with probability (1 - PAR)} 
\end{cases}
\]

The value of (1 - PAR) sets the rate of doing nothing. If the pitch adjustment decision for \(x_i'\) is YES, \(x_i'\) is replaced as follows:

\[
x_i' = x_i' \pm \text{rand}() \times \text{BW},
\]

where \(\text{BW}\) is an arbitrary distance bandwidth

\(\text{rand}()\) is a random number between 0 and 1

(4) Update harmony memory

If the new harmony vector, \(x' = (x_1', x_2', \ldots, x_{n'}', x_n')\) is better than the worst harmony (\(X_w\)) in the HM, judged in terms of the objective function value which determines the fitness, \(\text{fitness}(X_i)\), the new harmony is included in the HM and the existing worst harmony is excluded from the HM.

(5) Check stopping criterion

If the stopping criterion is satisfied, computation is terminated.
Otherwise, Steps 3 and 4 are repeated.

(6) Computational procedure

The procedure for the Harmony Search algorithm is summarized as follows [41]:

Step 1: Set the parameters HMS, HMCR, PAR, BW and NI.

Step 2: Initialize the HM and calculate the objective function value of each harmony vector.
Step 3: Improvise a new harmony $X_{\text{new}}$ as follows:

for ($j = 1$ to $N$) do
  if ($r_1 < \text{HMCR}$) then
    $X_{\text{new}}(j) = X_a(j)$ where $a \in \{1,2,\ldots,\text{HMS}\}$
    if ($r_2 < \text{PAR}$) then
      $X_{\text{new}}(j) = X_a(j) \pm r_3 \times \text{BW}$ where $r_1, r_2, r_3 \in (0,1)$
    endif
  else
    $X_{\text{new}}(j) = \text{LB}_j + r \times (\text{UB}_j - \text{LB}_j)$, where $r \in (0,1)$
  endif
endo
3.5 ARTIFICIAL BEE COLONY ALGORITHM

Artificial Bee Colony (ABC) algorithm, proposed by Karaboga in 2005 for real parameter optimization, is an optimization algorithm which simulates the foraging behaviour of bee colony [32]. The minimal model of swarm-intelligent forage selection in a honey bee colony, that ABC algorithm adopts, consists of three kinds of bees: employed bees, onlooker bees, and scout bees. Half of the colony comprises employed bees and the other half includes the onlooker bees. Employed bees are responsible for exploiting the nectar sources explored before and giving information to the other waiting bees (onlooker bees) in the hive about the quality of the food source site which they are exploiting. Onlooker bees wait in the hive and decide a food source to exploit depending on the information shared by the employed bees. Scout bees randomly search the environment in order to find a new food source depending on an internal motivation or possible external clues or randomly.

Main steps of the ABC algorithm simulating this behaviour are given below:

Step 1: Initialize the food source positions.

Step 2: Each employed bee produces a new food source in her food source site and exploits the better source.

Step 3: Each onlooker bee selects a source depending on the quality of her solution, produces a new food source in selected food source site and exploits the better source.

Step 4: Determine the source to be abandoned and allocate its employed bee as scout for searching new food sources.

Step 5: Memorize the best food source found so far.

Step 6: Repeat steps 2-5 until the stopping criterion is met.

3.6 PROPOSED WORK

One of the main objectives of this research work is to obtain a significant improvement in the performance of the music inspired metaheuristic optimization
algorithm, Harmony Search (HS) and its variant Global-best Harmony Search (GHS) in terms of solution quality and convergence speed.

HS has a remarkable advantage of algorithm simplicity and has gained increasing attention in tackling optimization problems [17,18,20,29,36,37]. However, HS and its other variants fail to concentrate on the following crucial factors:

- mutating the harmony vectors with better values in the improvisation step (where they are just pitch adjusted),
- monitoring the fitness of the harmony vectors
- discarding the immutable or worst harmony vectors while retaining the better

This research work proposes two optimization algorithms, with appropriate enhancements in the exploitation and exploration phases of HS and GHS, namely, Enhanced Bio inspired Harmony Search algorithm (EBHS) and Enhanced Bio inspired Global-best Harmony search algorithm (EBGHS), which have employed the factors mentioned above. The concept of mutating the harmony vectors has been inspired by the bees’ behaviour of ABC algorithm. Mutation of the harmony vectors with better values triggers the exploitation part of the search towards better solutions while discarding immutable harmony vectors from the harmony memory and replacing their places with new harmony vectors encourage the explorative part of the search.

3.6.1 Strategy – An outline

The strategy used in the proposed method is as follows:

- The exploitation phase of the HS method has been refined with the feature inspired from the food source exploitation behaviour of honey bees and the selection process used has been enhanced in such a manner that better individuals are promoted to the next generation
- In the exploitation phase, instead of merely considering and using the already existing information of the harmony memory, better information which can
be derived from its neighbouring solutions has been used to orient the search more towards the goal

- In the exploration phase, the immutable harmony vectors are removed whenever they are found to be not mutable or pitch adjustable after a predefined limit and replaced by randomly created new vectors and hence improves the exploration. Exploration helps the algorithm to quickly search the new regions of a large search volume

- The idea has been applied on the original HS as well as on one of its efficient variant GHS to ensure the wide range of its applicability

- Two novel enhanced Harmony Search algorithms have been proposed in order to solve optimization problems more efficiently, accurately and reliably

3.6.2 The Framework

The proposed methods employ the following schemes:

- A new Improvisation Scheme which modifies the Memory Consideration rule of the HS and GHS algorithms

- A new Monitoring Scheme which keeps track of the quality of the harmony vectors of the Harmony Memory

- An Abandoning Scheme which discards the immutable or not pitch adjustable harmony vectors from the Harmony Memory and includes new randomly generated harmony vectors

3.6.2.1 New Improvisation Scheme

In the memory consideration step, the HS algorithm chooses the design variable value from historically stored values in the HM with probability HMCR. The intensification or exploitation in HS is mainly controlled by HMCR and a high HMCR value indicates that the good solutions from the History/Memory are more likely to be selected or inherited. If the generated uniform random number \( r_1 \) is less than HMCR, the decision variable \( X_{\text{new}}(j) \) is generated by the memory consideration. That is, any
one of the vector stored in the harmony memory is selected randomly and considered, without any refinement, for further exploitation process. In addition, the fitness value of the selected vector is not at all considered and so it may lead to the possibility of selecting a worse vector for further exploitation. This situation leads to the low convergence rate as there are possibilities for the searching process getting trapped in locally optimal solutions. So, to overcome these problems, in the proposed algorithms, the memory consideration rule of original HS, \( X_{\text{new}}(j) = X_a(j) \) has been modified using the equation (1)

\[
X_{\text{new}}(j) = X_a(j) + \Phi_a(j)(X_a(j)-X_k(j)), \quad 1 \leq j \leq N
\]

where \( a, k \in \{1,2,\ldots,\text{HMS}\} \) and \( a \neq k \), \( \Phi a(j) \) is a random number between \((-1,1)\), HMS is the size of the Harmony Memory. Although \( k \) is selected randomly, it has to be different from \( a \). A new vector based on the value of a neighboring value within the lower and upper values has been created.

The role of \( \Phi a(j) \)

- It controls the production of New Harmony with a value within the possible range of values for the corresponding decision variable. i.e. it ensures \( Lx_i \leq X_{\text{new}}(j) \leq Ux_i \), where, \( Lx_i \) and \( Ux_i \) are the lower and upper bounds for each decision variable.

The role of the term \( (X_a(j)-X_k(j)) \)

- During the early stage of the local search process, the difference between \( X_a(j) \) and \( X_k(j) \) is high due to the dissimilarity between the candidate solutions and as the search approaches the optimum solution in the search space, the step length is adaptively reduced. This is because of the developed similarity in the solution vectors of the harmony memory. Hence, it is proved that the term \( \Phi a(j)(X_a(j)-X_k(j)) \) in equation (1) has great influence in improving the exploitation process in the early iterations and hence improves the members of the harmony memory and so the convergence rate.

The proposed approaches find a new solution in the neighbourhood of the currently considered solution. They also compare the new solution with the current solution and the better one is memorized and hence they attempt to retain the good solutions in the Harmony Memory and at the same time replacing the worst solution.
with a better one. This concept helps in improving the convergence rate considerably in the process of finding near optimal solutions.

### 3.6.2.2 New monitoring scheme

In the improvisation step of the HS, the harmony vectors are merely selected and pitch adjusted. The quality or the fitness value of the selected or the pitch adjusted vectors are not monitored and hence there is a possibility for the selection/inclusion of a harmony vector with low fitness value from/into the Harmony Memory. This affects the overall fitness quality of the harmony memory and hence leads to low convergence rate. In order to avoid this, the new monitoring scheme of the proposed method inspects the quality of the selected, mutated and the pitch adjusted harmony vectors. The immutable and not pitch adjustable harmony vectors are monitored and the parameter $\text{Trial}$ is used to record the number of times each harmony vector is not mutated or pitch adjusted. A harmony vector is said to be immutable or not pitch adjustable if its $\text{Trial}$ value exceeds a threshold value $\text{TLimit}$.

The monitoring scheme in the improvisation step is presented in bold letters as follows:

```plaintext
if ($r_1 < \text{HMCR}$) then // $r_1$ is a random number which takes the value between 0 and 1/
    $X_{\text{new}}(j) = X_a(j) + \Phi_a(j)(X_a(j)-X_i(j))$ // where $a,k \in (1,2,\ldots,\text{HMS})$ and $a \neq k$,
    $\Phi_a(j)$ is a random number between (-1,1), $r_1 \in (0,1)$/
    Evaluate the Objective function value of $X_{\text{new}}(j)$ and $X_a(j)$
    if (fitness ($X_{\text{new}}(j)$) > fitness ($X_a(j)$)) then
        $X_a(j) = X_{\text{new}}(j)$
    else
        $\text{Trial}(X_a(j)) = \text{Trial}(X_a(j)) + 1$
    endif
endif

if ($r_2 < \text{PAR}$) then // pitch adjusting the new Harmony
    $X_{\text{new}}(j) = X_a(j) \pm r_3 \times \text{BW}$ where $r_2, r_3 \in (0,1)$
    Evaluate the Objective function value of $X_{\text{new}}(j)$ and $X_a(j)$
    if (fitness ($X_{\text{new}}(j)$) > fitness ($X_a(j)$)) then
        $X_a(j) = X_{\text{new}}(j)$
    else
        $\text{Trial}(X_a(j)) = \text{Trial}(X_a(j)) + 1$
    endif
endif
```

```
3.6.2.3 New abandoning scheme

This scheme introduces a new step after the improvisation step. A vector which exceeds the threshold trial limit, **TLimit**, if any, is found and replaced by a randomly generated harmony vector. This scheme enhances the explorative part of the search as there exists another chance for the new vectors to be included in the Harmony Memory by replacing the immutable and not pitch adjustable vectors. The Pseudo code of the abandoning scheme is given below:

Randomly select a number *j* between 1 and *n* where *n*= number of decision variables

\[ X_{\text{new}}(j) = LB_j + r \times (UB_j - LB_j), \text{ where } r \in (0, 1) \]

Maxtrial = Trial\((X_1(j))\) // Trial\((X_1(j))\) denotes the number of times the harmony vector \((X_1(j))\) is not mutated or pitch adjusted //

// The harmony vector with maximum Trial value is found //

\[ aband = 1 \]

\[ \text{for (} k = 2 \text{ to HMS) do} \]

\[ \text{if Trial (} X_k(j)\text{) > Maxtrial} \]

\[ Aband = k \]

\[ \text{Maxtrial = Trial (} X_k(j)\text{)} \]

\[ \text{endif} \]

\[ \text{endfor} \]

// The harmony vector is replaced by a newly generated vector if it exceeds a threshold value TLimit //

If Maxtrial > TLimit

\[ X_{\text{aband}}(j) = X_{\text{new}}(j) \]

3.7 The EBHS algorithm

The Enhanced Bio inspired Harmony Search is shown as follows:

**Step 1:** Set the Parameters HMS, HMCR, PAR and NI, TLimit, Trial (of size HM)

**Step 2:** Initialize the HM with a set of random solutions \(X_a\) and Trial with zeros

**Step 3:** Improvise a new harmony as follows:

\[ \text{for (} i = 1 \text{ to } n \text{) do} // n = number of decision variables \]

\[ \text{if } (r_1 < \text{HMCR}) \text{ then} // r_1 \text{ is a random number which takes the value between 0 and 1} //\]

\[ X_{\text{new}}(j) = X_a(j) + \Phi_a(j)(X_a(j)-X_d(j)) // \text{ where } a,k \in (1,2,\ldots,\text{HMS}) \text{ and } a \neq k, \]

\[ \Phi a(j) \text{ is a random number between (-1,1), } r_1 \in (0,1) //\]

Evaluate the Objective function value of \(X_{\text{new}}(j)\) and \(X_a(j)\)

\[ \text{if (fitness } (X_{\text{new}}(j)) > \text{fitness } (X_a(j)) \text{ then} \]

\[ X_a(j) = X_{\text{new}}(j) \]
\[ \text{if (} r_2 < \text{PAR} \text{) then} \] // pitch adjusting the new Harmony
\[ X_{\text{new}}(j) = X_{\text{a}}(j) \pm r_3 \times BW \quad \text{where } r_2, r_3 \in (0,1) \]

Evaluate the Objective function value of \( X_{\text{new}}(j) \) and \( X_{\text{a}}(j) \)

\[ \text{if (fitness} (X_{\text{new}}(j)) > \text{fitness} (X_{\text{a}}(j)) \text{) then} \]
\[ X_{\text{a}}(j) = X_{\text{new}}(j) \]
\[ \text{Trial}(X_{\text{new}}(j))=0 \]

\[ \text{else} \]
\[ \text{Trial}(X_{\text{a}}(j)) = \text{Trial}(X_{\text{a}}(j)) +1 \]
\[ \text{endif} \]
\[ \text{endif} \]

Step 4: Randomly select a number \( j \) between 1 and \( n \) where \( n \) = number of decision variables

\[ X_{\text{new}}(j) = L B_{j} + r \times (U B_{j} - L B_{j}) \text{, where } r \in (0, 1) \]

Maxtrial = Trial(\( X_{\text{a}}(j) \))// Trial(\( X_{\text{a}}(j) \)) denotes the number of times the harmony vector \( (X_{\text{a}}(j)) \) is not mutated or pitch adjusted//

// The harmony vector with maximum Trial value is found//

aband = 1

for \( k = 2 \) to HMS do

If Trial (\( X_{\text{a}}(j) \)) > Maxtrial

Aband = \( k \)

Maxtrial = Trial (\( X_{\text{a}}(j) \))

endif

endfor

// The harmony vector is replaced by a newly generated vector if it exceeds a threshold value TLimit//

if Maxtrial > TLimit

\[ X_{\text{aband}}(j) = X_{\text{new}}(j) \]

\[ \text{Trial}(X_{\text{new}}(j))=0 \]

endif
Step 5: Update the HM as $X_w = X_{new}$ if fitness ($X_{new}$) > fitness ($X_w$)

Step 6: Terminating criteria

Case 1: if optimum value is known apriori:
Calculate $|f(x_{best}) - f^*|$ where $f(x_{best})$ is the best-of-the-run value and $f^*$ is the actual optimum
If $|f(x_{best}) - f^*| < \varepsilon$ then return the Best Harmony vector, $X_{best}$ in the HM else go to step 3.

Case 2: if optimum value is not known apriori:
if NI is reached then return the Best Harmony vector, $X_{best}$ in the HM else go to step 3.

3.8 The EBGHS algorithm

The Enhanced Bio inspired Global-best Harmony Search is shown as follows:

Step 1: Set the Parameters HMS, HMCR, PAR and NI, TLimit, Trial (of size HM)
Step 2: Initialize the HM with a set of random solutions $X_a$ and Trial with zeros
Step 3: Improvise a new harmony as follows:
for (j = 1 to n) do // n = number of decision variables
if ($r_1 < HMCR$) then // $r_1$ is a random number which takes the value between 0 and 1//
$X_{new}(j) = X_a(j) + \Phi_a(j)(X_{best}(j)-X_a(j))$ // where $a,k \in (1,2,\ldots,HMS)$ and $a \neq k$,
$\Phi_a(j)$ is a random number between (-1,1), $r1 \in (0,1)$//
Evaluate the Objective function value of $X_{new}(j)$ and $X_a(j)$
if (fitness ($X_{new}(j)$) > fitness ($X_a(j)$)) then
$X_a(j)=X_{new}(j)$
Trial($X_{new}(j)$)=0
else
Trial($X_a(j)$) = Trial($X_a(j)$) + 1
endif
if ($r_2 < PAR$) then // pitch adjusting the new Harmony
$X_{new}(j) = X_{Best}$ where $r_2 \in (0,1)$ and $X_{Best}$ is the Global best solution in the Harmony memory
endif
else
$X_{new}(j) = LB_j + r \times (UB_j-LB_j)$, where $r \in (0,1)$
Trial($X_{new}(j)$)=0
endif
endfor
Step 4: Randomly select a number $j$ between 1 and $n$ where $n$ = number of decision variables
$X_{new}(j) = LB_j + r \times (UB_j-LB_j)$, where $r \in (0,1)$
Maxtrial = Trial($X_a(j)$)// Trial($X_a(j)$) denotes the number of times the harmony vector ($X_a(j)$) is not mutated or pitch adjusted
// The harmony vector with maximum Trial value is found//
aband = 1
for (k = 2 to HMS) do
    if Trial (X_k(j)) > Maxtrial
        Aband = k
        Maxtrial = Trial (X_k(j))
    endif
endfor

// The harmony vector is replaced by a newly generated vector if it exceeds a threshold value TLimit//
if Maxtrial > TLimit
    X_aband(j) = X_new(j)
    Trial(X_new(j)) = 0
endif

Step 5: Update the HM as X_w = X_new if fitness (X_new) > fit (X_w)

Step 6: Terminating criteria

Case 1: if optimum value is known apriori:
Calculate |f(x_best) - f*| where f(x_best) is the best-of-the-run value and f* is the actual optimum
if |f(x_best) - f*| < ε then return the Best Harmony vector, X_best, in the HM else go to step 3.

Case 2: if optimum value is not known apriori:
if NI is reached then return the Best Harmony vector, X_best, in the HM else go to step 3.

3.9 PERFORMANCE ANALYSIS

The performance of the EBHS and EBGHS algorithms is analysed in this section.

3.9.1 Experimental setup

To test the performance of the proposed algorithms, an extensive experimental evaluation and comparison with the HS and a variant of HS, GHS, are provided based on a set of eleven global optimization problems depicted in figures from Fig. 3.2 to Fig. 3.11 as follows:

A. Sphere function, defined as

\[
\min f(x) = \sum_{i=1}^{n} x^2(i)
\]

where global optimum \( x^* = 0 \) and \( f(x^*) = 0 \) for \(-100 \leq x(i) \leq 100\)
B. Schwefel’s problem 2.22, defined as

\[
\min f(x) = \sum_{i=1}^{n} |x(i)| + \prod_{i=1}^{n} |x(i)|
\]

where global optimum \(x^* = 0\) and \(f(x^*) = 0\) for \(-10 \leq x(i) \leq 10\)

C. Rosenbrock function, defined as

\[
\min f(x) = \sum_{i=1}^{n-1} (100(x(i + 1) - x^2(i))^2 + (x(i) - 1)^2),
\]

where global optimum \(x^* = (1,1,1,...1)\) and \(f(x^*) = 0\) for \(-30 \leq x(i) \leq 30\)
D. Himmelblau function, defined as
\[ \min f(x,y) = (x^2 + y - 11) - (x + y^2 - 7)^2 \]
where there are 4 equally valid solutions, each resulting in \( f(x^*) = 0 \) for \(-5 \leq x,y \leq 5\) 
The four points are: (-3.779,-3.283), (-2.805,-3.131), (3,2), (3.584, -1.848)

E. Step function, defined as
\[ \min f(x) = \sum_{i=1}^{n} (|x(i) + 0.5|)^2 \]
where global optimum \( x^* = 0 \) and \( f(x^*) = 0 \) for \(-100 \leq x(i) \leq 100\)

F. Rotated hyper-ellipsoid function, defined as
\[ \min f(x) = \sum_{i=1}^{n} \left( \sum_{j=1}^{i} x(j) \right)^2 \]
where global optimum \( x^* = 0 \) and \( f(x^*) = 0 \) for \(-100 \leq x(i) \leq 100\)
G. Schwefel’s problem 2.26, defined as

$$\min f(x) = 418.9829n - \sum_{i=1}^{n} (x(i) \sin(\sqrt{x(i)}))$$

where global optimum $$x^* = (42.9687, 420.9687, \ldots, 420.9687)$$ and $$f(x^*) = 0$$ for $$-500 \leq x(i) \leq 500$$

H. Rastrigin function, defined as

$$\min f(x) = \sum_{i=1}^{n} (x^2(i) - 10 \cos(2\pi x(i)) + 10),$$

where global optimum $$x^* = 0$$ and $$f(x^*) = 0$$ for $$-5.12 \leq x(i) \leq 5.12$$
I. **Ackley function, defined as**

\[
\min f(x) = -20 \exp \left( -0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x^2(i)} \right) - \exp \left( \frac{1}{n} \sum_{i=1}^{n} \cos(2\pi x(i)) \right) + 20 + e
\]

where global optimum \( x^* = 0 \) and \( f(x^*) = 0 \) for \(-32 \leq x(i) \leq 32\)

---

J. **Griewank function, defined as**

\[
\min f(x) = \frac{1}{4000} \sum_{i=1}^{n} x^2(i) - \prod_{i=1}^{n} \cos \left( \frac{x(i)}{\sqrt{i}} \right) + 1
\]

where global optimum \( x^* = 0 \) and \( f(x^*) = 0 \) for \(-100 \leq x(i) \leq 100\)
Fig. 3.10 2D Griewank function

K. Six-hump Camel-back function, defined as

\[
\min f(x) = \left( 4x^2(1) - 2.1x^4(1) + \frac{1}{3} x^6 (1) + x(1) \ast x(2) - 4x^2(2) + 4x^4(2) \right),
\]

where global optimum \( x^* = (0.08983, 0.7126) \) and \( f(x^*) = -1.0316285 \) for \(-5 \leq x(i) \leq 5\)

Fig.3. 11 2D Six-hump Camel-back function

Among the above mentioned 11 benchmark problems, Sphere function, Schwefel’s problem 2.22, Step function, Rotated hyper-ellipsoid function are unimodal. Rosenbrock function, Schwefel’s problem 2.26, Rastrigin function, Ackley function, Griewank are difficult multimodal functions where the number of local minima increases with the problem dimensions. Six-hump camel-back function and Himmelblau function are low dimensional functions with a few local optima.
3.9.2 Parameter Setting

For HS and for GHS algorithms, the following parameters have been set to the values recommended in [41] and [54] respectively as these values lead to optimal solutions with significant convergence rate: Size of the Harmony Memory = 15, $HMCR = 0.9$, $PAR = 0.3$, $BW = 0.01$ and the maximum improvisation number is 40000 for all test problems. For the proposed algorithms the same parameter setting has been maintained. Each problem has been run 30 independent replications. The average and standard deviations over these 30 replications for dimensions equal to 25 and 50 have been reported in Table 3.1 and Table 3.2 respectively. The significant best solutions are shown in bold letters. The optimal solutions obtained using each method for the corresponding functions are listed in Table 3.3 and the best solutions are shown in bold letters.

3.9.3 Results and Discussion

The Harmony search is a stochastic metaheuristic algorithm in nature and it uses two random probability distribution functions for determining the probability values of the two random variables, $r_1$ and $r_2$, in the Harmony Memory Consideration phase of the algorithm. So, the performance analysis of the proposed algorithm has been done based on the empirical results. The comparative study presented on the global optimization functions focuses on the following performance metrics:

a) The quality of the final solution

b) The convergence speed

c) The convergence characteristics

To illustrate the proposed algorithms’ convergence characteristics, two functions from each category have been considered. Schwefel’s problem 2.22 and Step function were for unimodal, Rosenbrock function and Rastrigin functions were for multimodal and Six-hump camel-back function and Himmelblau function were for low dimensional functions with a few local optima.

a) Comparison of the Quality of the final solution: To judge the accuracy of different algorithms, they were run for a very long time over every benchmark function, until the number of iterations exceeds a given upper limit (fixed here as $4 \times 10^5$).
The mean and the standard deviation of the best-of-the-run errors for 30 independent runs of each of the four algorithms are presented in Table 3.1 and Table 3.2. Note that the best-of-the-run error corresponds to the absolute error difference between the best-of-the-run value $f(x_{best})$ and the actual optimum $f^*$ of a particular objective function, i.e., $|f(x_{best}) - f^*|$. 

The experiments reported here are for the number of dimensions 25 and 50.

**Observations**

- Tables 3.1 and 3.2 indicate that the EBHS algorithm generates 11 best results out of 11 functions compared to HS and EBGHS generates 11 best results out of 11 functions compared to GHS.

- EBGHS outperforms HS, GHS and EBHS for all the optimization functions except for the Six-Hump Camel-Back function and Himmelblau function where EBHS outperforms the other competitors. In [54], it has been stated that, for the Six-Hump Camel-Back function, the GHS has performed relatively worse than HS because the dimension of the problem is too low and in case of low dimensionality, the GHS’s pitch adjustment step suffers. But, the proposed EBGHS algorithm performs better for the Six-Hump Camel-Back function than that of GHS and HS. Table 3.1 depicts that the EBHS algorithm outperforms the other three while solving low dimensional functions with a few local optima.

- All the algorithms improvised a near optimum solution but the result that has been obtained by the EBGHS is better than the results of the other algorithms both in low and high dimensions.
Table 3.1 Mean and Standard Deviation of the benchmark function optimization results (n=25) in the format Mean ± STD

<table>
<thead>
<tr>
<th>Function</th>
<th>Global optimum</th>
<th>HS</th>
<th>EBHS</th>
<th>GHS</th>
<th>EBGHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(0,0,……,0)</td>
<td>0.000196 ± 0.000025</td>
<td>0.000013 ± 0.000009</td>
<td>0.000054 ± 0.000019</td>
<td>0.000001 ± 0.000000</td>
</tr>
<tr>
<td>B</td>
<td>(0,0,……,0)</td>
<td>0.215230 ± 0.0750921</td>
<td>0.000149 ± 0.000178</td>
<td>0.163593 ± 0.096807</td>
<td>0.000092 ± 0.000074</td>
</tr>
<tr>
<td>C</td>
<td>(1,1)</td>
<td>210.339526 ± 108.565294</td>
<td>83.043394 ± 0.016259</td>
<td>68.373275 ± 79.677317</td>
<td>7.679725 ± 13.02415</td>
</tr>
<tr>
<td>D</td>
<td>(3.584427, -1.848126)</td>
<td>-1.848000 ± 0.000000</td>
<td>-1.848000 ± 0.000000</td>
<td>-1.869000 ± 0.000031</td>
<td>-1.848020 ± 0.000010</td>
</tr>
<tr>
<td>E</td>
<td>(0,0,……,0)</td>
<td>4.856072 ± 3.054675</td>
<td>0.003190 ± 0.000076</td>
<td>0.000000 ± 0.000000</td>
<td>0.000000 ± 0.000000</td>
</tr>
<tr>
<td>F</td>
<td>(0,0,……,0)</td>
<td>3862.031089 ± 983.767532</td>
<td>10.083410 ± 6.041984</td>
<td>4132.880910 ± 4621.948200</td>
<td>11.420098 ± 0.000582</td>
</tr>
<tr>
<td>G</td>
<td>(420.9687,420.9687,...,420.9687)</td>
<td>27.779129 ± 9.007291</td>
<td>0.000386 ± 0.000096</td>
<td>0.048619 ± 0.006381</td>
<td>0.000071 ± 0.000036</td>
</tr>
<tr>
<td>H</td>
<td>(0,0,……,0)</td>
<td>1.82640 ± 0.639510</td>
<td>0.074554 ± 0.042098</td>
<td>0.044802 ± 0.048120</td>
<td>0.003569 ± 0.000982</td>
</tr>
<tr>
<td>I</td>
<td>(0,0,……,0)</td>
<td>1.128030 ± 0.317013</td>
<td>0.002730 ± 0.000619</td>
<td>0.027318 ± 0.020619</td>
<td>0.000082 ± 0.000371</td>
</tr>
<tr>
<td>J</td>
<td>(0,0,……,0)</td>
<td>1.092072 ± 0.036018</td>
<td>0.004916 ± 0.003642</td>
<td>0.107630 ± 0.182073</td>
<td>0.000093 ± 0.000072</td>
</tr>
<tr>
<td>K</td>
<td>(-.08983,0.7126)</td>
<td>-1.031628 ± 0.000000</td>
<td>-1.031628 ± 0.000000</td>
<td>-1.031620 ± 0.000020</td>
<td>-1.031628 ± 0.000000</td>
</tr>
</tbody>
</table>
Table 3.2 Mean and Standard Deviation of the benchmark function optimization results (n=50) in the format Mean ± STD

<table>
<thead>
<tr>
<th>Function</th>
<th>Global optimum</th>
<th>HS</th>
<th>EBHS</th>
<th>GHS</th>
<th>EBGHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(0,……,0)</td>
<td>8.984062 ± 0.821126</td>
<td><strong>0.000846± 0.000674</strong></td>
<td>2.439871 ± 0.629840</td>
<td>0.000002± 0.000001</td>
</tr>
<tr>
<td>B</td>
<td>(0,……,0)</td>
<td>82.738130 ± 6.9527801</td>
<td><strong>0.0289540± 0.049028</strong></td>
<td>20.163593 ± 5.096807</td>
<td>0.005218± 0.000429</td>
</tr>
<tr>
<td>C</td>
<td>(1,1)</td>
<td>157898210.339526± 3231508.655762</td>
<td><strong>989.648391± 794.740180</strong></td>
<td>2637858.067023± 925378.564787</td>
<td>519.537811± 427.738400</td>
</tr>
<tr>
<td>D</td>
<td>(3.584427, -1.848126)</td>
<td>-1.848000 ± 0.000000</td>
<td><strong>-1.848000± 0.000000</strong></td>
<td>-1.869000± 0.000031</td>
<td>-1.848020± 0.000010</td>
</tr>
<tr>
<td>E</td>
<td>(0,……,0)</td>
<td>21356.856072± 2108.002995</td>
<td><strong>0.100128± 0.026482</strong></td>
<td>5236.943000± 1137.780188</td>
<td>0.009841± 0.020170</td>
</tr>
<tr>
<td>F</td>
<td>(0,……,0)</td>
<td>238162.098563± 29383.061767</td>
<td><strong>28451.00451± 19266.93519</strong></td>
<td>334132.880910± 42621.76844</td>
<td>17951.93100± 8457.92617</td>
</tr>
<tr>
<td>G</td>
<td>(420.9687,420.9687,... 420.9687)</td>
<td>7867.327634± 579.367295</td>
<td><strong>48.932700± 109.639102</strong></td>
<td>1281.3875± 402.084683</td>
<td>29.819621± 90.876000</td>
</tr>
<tr>
<td>H</td>
<td>(0,……,0)</td>
<td>348.24660± 28.107465</td>
<td><strong>10.873510± 0.6394219</strong></td>
<td>80.756004± 30.653809</td>
<td>0.107369± 0.01963073</td>
</tr>
<tr>
<td>I</td>
<td>(0,……,0)</td>
<td>13.878083 ± 0.317013</td>
<td><strong>0.000051± 0.000006</strong></td>
<td>8.756908 ± 0.907609</td>
<td>0.000000± 0.000000</td>
</tr>
<tr>
<td>J</td>
<td>(0,……,0)</td>
<td>195.874390 ± 0.332687</td>
<td><strong>0.012986± 0.019872</strong></td>
<td>56.081628± 18.765903</td>
<td>0.005193± 0.003902</td>
</tr>
<tr>
<td>K</td>
<td>(-.08983,0.7126)</td>
<td>-1.031628± 0.000000</td>
<td><strong>-1.031628± 0.000000</strong></td>
<td>-1.031620± 0.000020</td>
<td>-1.031628± 0.000000</td>
</tr>
</tbody>
</table>

b) Comparison of the convergence speed and success Rate: In order to compare the speeds of different algorithms, a threshold value of the error for each benchmark problem has been selected. For the benchmark functions used, this threshold is fixed as $10^{-5}$. Each algorithm was run on a function and stopped as soon as the best error value falls below the predefined threshold value. Then the number of iterations the algorithm takes was noted down. A lower number of iterations correspond to a faster algorithm. The maximum number of iterations for each algorithm was kept as $4 \times 10^4$ for all functions. Table 3.3 reports the number of iterations (the best value
among the 30 runs) taken by the algorithm to find the optimum solution (within the
given tolerance) and the same is depicted in Fig. 3.12 as a comparison graph.

Observations

- Table 3.3 shows that the proposed algorithms not only yield the most
accurate results for nearly all the benchmark problems, but also they do
so by taking less number of iterations. The Comparison graphs shown in
Fig. 3.12(a) and 3.12(b) clearly depict this. It is also observed from
Fig. 3.12(a) that EBGHS converges faster for multimodal functions to
the optimal solution when compared to the convergence speed of other
competitors. But, in case of low dimensional functions, EBHS converges
faster to optimal solution when compared to other algorithms and it is
shown in Fig.3.12(b). Further, Fig. 3.13(a) and 3.13(b) depict the
improvement in the Convergence Speed of EBHS over HS and EBGHS
over GHS respectively.

- The lower mean and standard deviation values of the proposed methods
in Table 3.1 and 3.2 show their higher robustness (i.e., their ability to
produce similar results over repeated runs on a single problem)
compared to their other competitors.
Table 3.3 The Optimal solutions and the Iteration number* at which the solution is obtained (n=25)

<table>
<thead>
<tr>
<th>Function</th>
<th>Global optimum</th>
<th>HS</th>
<th>EBHS</th>
<th>GHS</th>
<th>EBGHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rosenbrock</td>
<td>1.1</td>
<td>(1.001501, 1.003031), 4649</td>
<td>(0.999359, 0.999817), 3217</td>
<td>(1.065781, 1.106731), 4059</td>
<td>(1.051525, 1.105706), 2365</td>
</tr>
<tr>
<td>Rastrgin</td>
<td>0,0,……,0</td>
<td>(0,0,……,0), 16956</td>
<td>(0,0,……,0), 8478</td>
<td>(0,0,……,0), 16027</td>
<td>(0,0,……,0), 8116</td>
</tr>
<tr>
<td>Himmelblau</td>
<td>(3.584427, -1.848126)</td>
<td>(3.584491, -1.847621), 2841</td>
<td>(3.584427, -1.848126), 753</td>
<td>(3.588519, -1.849269), 8105</td>
<td>(3.584106, -1.84805), 2010</td>
</tr>
<tr>
<td>Six-hump Camel-back</td>
<td>(-0.08983, 0.713279)</td>
<td>(-0.088263, 0.713279), 4132</td>
<td>(-0.089531, 0.713279), 2510</td>
<td>(-0.086483, 0.715506), 5814</td>
<td>(-0.089667, 0.713282), 4096</td>
</tr>
<tr>
<td>Step</td>
<td>0,0,……,0</td>
<td>(0,0,0,……,0), 14679</td>
<td>(0,0,0,……,0), 6433</td>
<td>(0,0,0,……,0), 7852</td>
<td>(0,0,0,……,0), 5179</td>
</tr>
<tr>
<td>Schwefel</td>
<td>0,0,……,0</td>
<td>(0,0,0,……,0), 17057</td>
<td>(0,0,0,……,0), 8472</td>
<td>(0,0,0,……,0), 12413</td>
<td>(0,0,0,……,0), 6529</td>
</tr>
</tbody>
</table>

* The best value among 30 runs

Fig. 3.12(a) Comparison of the convergence speed of the four algorithms for unimodal and multimodal functions
Fig. 3.12(b) Comparison of the Convergence Speed of the Four Algorithms for low dimensional functions

Fig. 3.13(a) The improvement in the convergence speed of EBHS over HS
Fig. 3.13(b) The improvement in the convergence speed of EBGHS over GHS

C) The Convergence Characteristics: The convergence characteristics of the proposed and the existing methods have been shown in Fig. 3.14 – Fig. 3.19 in terms of the objective function values (Absolute Error values) versus the number of iterations.

Fig. 3.14 Convergence curves for the Rosenbrock function
Fig. 3.15 Convergence curves for the Rastrigin function

Fig. 3.16 Convergence curves for the Himmelblau function
Fig. 3.17 Convergence curves for the Six-Hump Camel-Back function

Fig. 3.18 Convergence curves for the Step function
**Fig. 3.19 Convergence curves for the Schwefel function**

**Observations**

- The convergence curves of the proposed methods (EBHS and EBGHS) in Fig. 3.14 – Fig. 3.19 show that they have maintained a steady convergence rate from the start and finally finished at the optimum value, while the other two variants of HS have showed a much slower convergence.

- For high dimensional functions, the evolution curves of the EBGHS algorithm descend much faster and reach lower level than that of the other compared algorithms as shown in Fig. 3.14, 3.15, 3.18 and 3.19.

- In Fig. 3.16 and 3.17, it has been observed that the evolution curve of EBHS descends much faster to the lower values when compared to EBGHS which indicates that EBHS performs better than EBGHS only in lower dimensions.

- The evolution curve of EBHS algorithm descends much faster and reaches lower level in Fig. 3.14 – Fig. 3.19 when compared to HS.
Thus, it can be concluded that in overall the Bio inspired proposed algorithms outperform the other methods and are generating significantly better results than the original HS algorithm and its best variant GHS.

3.10 SUMMARY

This work has presented two novel algorithms EBHS and EBGHS each with a time complexity of O(n), where n is the number of decision variables, for solving continuous optimization problems. In these methods, with appropriate changes in the memory consideration rule, the improvisation process is improved. The exploitation process is carried out in a controlled way so that better harmony vectors enjoy higher selection probability. The immutable harmony vectors are removed whenever they are found to be not mutable or pitch adjustable after a predefined limit and replaced by randomly created new vectors and hence improves the exploration. These actions guide to the speedy occupation of the harmony memory with better solutions and hence cause the search to move rapidly towards the goal. These enhancements also avoid the problem of getting trapped into the local minima as the chance for selecting the same information repeatedly has been minimized. This results in a significant improvement in the performance in terms of solution quality and convergence speed. From the simulation results, it is concluded that the proposed algorithms have the ability to attain the global optimum. The proposed approaches were tested on eleven benchmark functions with different properties where they generally outperformed the HS and GHS algorithms.