Expert Systems With Applications 81 (2017) 423–443

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Selective Refining Harmony Search: A new optimization algorithm

Mahdi Shabani[a], Seyed Abolghasem Mirroshandel[b,*], Hadi Asheri[c]

[a] Department of Computer Engineering, Islamic Azad University, Science and Research Branch, Tehran, Iran
[b] Department of Computer Engineering, University of Guilan, Rasht, Iran
[c] Department of Electrical and Computer Engineering, College of Engineering, University of Tehran, Tehran, Iran

ARTICLE INFO

Article history:
Received 8 September 2016
Revised 19 March 2017
Accepted 20 March 2017
Available online 31 March 2017

Keywords:
Optimization
Evolutionary algorithm
Meta-heuristics
Population refinement
Selective Harmony Search
Exploitation
Exploration

ABSTRACT

While music improvisation is an NP-hard problem, it has always been handled by musicians successfully. Harmony search is a class of meta-heuristics inspired by the improvisation process of musicians. Inexperienced musicians usually make several harmonies until they find the desired one. However, experienced musicians more rely on their knowledge and experience instead of brute-force searching for a desired harmony. When making a harmony, they are able to distinguish the undesired notes of the current harmony and just modify them instead of throwing away the total harmony and making a new one. This approach of experienced musicians was adopted in this paper to allow the harmony search algorithm to exploit the knowledge and experience accumulated in the harmony memory to refine current harmonies. The underlying algorithm is called Selective Refining Harmony Search in which a new harmony memory update has been utilized. The main differences between the proposed method and the original harmony search are the integration of selection in the improvisation step and introduction of refinement concept. The improvements provided by the proposed method originate from its superior ability to imitate the behavior of musicians in the sense that instead of total harmonies, decision variables are considered during the memory update process. This modification provided outstanding performance to minimize the objective functions of the underlying problems, while computational loads are also better than other methods in most cases. During refinement procedure, two new parameters were employed to make a trade-off between effectiveness and efficiency of the algorithm. During the experiments, the proposed method exhibited robust performance against its two new parameters. Several algorithms including original harmony search and its state-of-the-art variants were implemented to conduct comprehensive comparisons. All of the algorithms were evaluated over IEEE CEC 2010 suite, one of the well-known and challenging benchmark test sets. Several studies have evidenced that the utilization of harmony search has been fostered in diverse application fields. Thus, the improvements to harmony search algorithm provided by the proposed method makes it more efficient for those fields and possible to be employed for more challenging problems.

1. Introduction

Optimization is a collection of mathematical algorithms with a wide range of successful engineering applications (Floudas & Pardalos, 2008). Some of these applications are too complex to be solved by traditional mathematical methods and has led to the emergence of a new branch of optimization called meta-heuristic optimization, the field of general nature-inspired algorithms. While heuristics are problem-dependent techniques and usually are adapted to the problem at hand, a meta-heuristic is a collection of algorithmic concepts utilized to define heuristic methods aimed to solve different types of problems. In other words, a metaheuristic is a general framework which is usually applied to a broad range of optimization problems in a way that relatively few modifications are required to adapt them to a specific problem (Bianchi, Dorigo, Gambardella, & Gutjahr, 2009; Osman & Kelly, 2012). Most meta-heuristic algorithms are nature-inspired (Gendreau & Potvin, 2010; Talbi, 2009; Yang, 2010), including particle swarm optimization (Kennedy, 2010), ant colony optimization (Dorigo & Birattari, 2010), cuckoo search (Gandomi, Yang, & Alavi, 2013; Shokri-Ghaleh & Alfi, 2014b) and simulated annealing (Leung, Zhang, Zhou, & Wu, 2012; Rodriguez, Garcia-Martinez, & Lozano, 2012). These nature-based algorithms have been success-

* Corresponding author.
E-mail addresses: mahdishabani2002@gmail.com (M. Shabani), mirroshandel@guilan.ac.ir (S. Abolghasem Mirroshandel), h.asheri@ut.ac.ir (H. Asheri).

http://dx.doi.org/10.1016/j.eswa.2017.03.044
0957-4174/

fully employed in different fields from engineering to health care systems (Bhattacharyya & Bandyopadhyay, 2015; Bueno, Jino, & Wong, 2014; Devika, Jafarian, Hassanzadeh, & Khodaverdi, 2014; Garcia, Montiel, Castillo, Sepúlveda, & Melin, 2009; Ho, Shih, Liao, & Chu, 2012; Landa-Torres, Manjarres, Salcedo-Sanz, Del Ser, & Gil-Lopez, 2013; Minetti, Leguizamón, & Alba, 2014; Moh'd Alia, Mandava, & Aziz, 2011; Shokri-Ghaleh & Alfi, 2014a).

In the literature of meta-heuristics, Harmony Search (HS) was proposed as a novel population-based algorithm aimed to handle the optimization of water distribution networks problem (Geem, Kim, & Loganathan, 2001). In fact, HS algorithm is a variation of evolutionary algorithms (EA) in which each member of the population is considered as a musical harmony. Recently, it has attracted great attention in different areas like mechanical engineering (Fesanghary, 2009), system control (Cao & Wang, 2011; Khooban, Abadi, Alfi, & Siahi, 2013), signal processing (Guo, Wang, Gao, & Tanskanen, 2012; dos Santos Coelho & de Andrade Bernert, 2009), scheduling (Akkoyunlu, Engin, & Buyukozkan, 2015; Yuan, Xu, & Yang, 2013) and many other engineering fields (Forsati & Mahdavi, 2010; Forsati & Shamsfard, 2015; Hadwan, Ayob, Sabar, & Qu, 2013; Liu & Zhou, 2013; Shreem, Abdullah, & Nazri, 2014; Turky & Abdullah, 2014). The core idea behind HS has been borrowed from musicians who are always looking for state of harmonies in the musical improvisation process. In fact, a musician has a lot of music pitches in his memory and usually examines many possible combinations of them to make a perfect harmony. The procedure of finding optimal solutions to engineering problems can be taught as an analogous to this kind of search for a desired harmony. Compared to traditional techniques, HS has achieved a great success when it comes to deal with complex optimization problems (Lee & Geem, 2005). HS algorithm holds several advantageous (Al-Betar, Khader, Geem, Doush, & Awadallah, 2013; Geem, 2008; Mahdavi, Fesanghary, & Damangir, 2007) over many other optimization methods: (1) HS can handle discontinuous functions as well as continuous ones because it does not need differential gradients; (2) HS is able to deal with discrete variables as well as continuous ones; (3) HS does not need the variables to be initialized; (4) Its convergence of is guaranteed; (5) HS has the ability to escape local optima with high probability; (6) HS has a novel stochastic derivative applied to discrete variables, which uses musician's experiences as a search direction.

During the musical improvisation process, experienced musicians are usually more favored to employ their knowledge to determine suitability of the harmonies and improve them with some modifications. Now, one might ask an interesting question: Is there a way to make HS algorithm able to modify the undesired notes of current harmonies to improve them, just similar to musicians? In order to find an answer to this question, a new HS algorithm called *Selective Refining Harmony Search (SRHS)* is proposed in this paper. SRHS algorithm offers a novel method to utilize existing knowledge and experience in the harmony memory to modify undesired variables (notes) of the current harmonies. In fact, SRHS is aimed to improve the quality and fitness of the population through refinement of undesired notes of the current harmonies. Original HS algorithm tries to improve the population through generating new random harmonies or through random modifications made on a randomly chosen member of the harmony memory. Previous works on HS algorithm have proved that relying on the best harmony improves the performance of HS algorithm (Omran & Mahdavi, 2008; Ouyang, Gao, Li, & Kong, 2015; Wang & Huang, 2010). Due to this fact, SRHS picks the best harmony as an appropriate representative of the accumulated knowledge and experience in the harmony memory. Then, it utilizes this harmony in a new step called *population refinement* to refine current members of the harmony memory. SRHS also introduces to new parameters related to population refinement step: *refinement period* (RP) and *subset size* (SS). RP determines the frequency of *population refinement* action, i.e. the intervals or iterations in which the harmony memory will be refined. Within each refinement step, first a number of members of harmony memory are selected and then their decision variables get refined. In order to refine the selected harmonies, a subset of decision variables of best harmony are utilized. The length of this subset is determined by SS.

The rest of the paper is organized as follows: the original HS algorithm is presented in Section 2. Some benchmark and state-of-the-art variants of HS are reviewed in Section 3. SRHS is depicted in Section 4. Section 5 contains a comprehensive description of experimental setup, test functions, comparisons, statistical tests and sensitivity analyses. Finally, conclusions and future works are summarized in Section 6.

2. Harmony search

An optimization problem is usually defined as follows:

$$\arg \min_{\overline{X}=(x_1,x_2,\cdots,x_N)} f(\overline{X}), \tag{1}$$

in which $f(\overline{X})$ is the objective function, $\overline{X}$ is the solution vector composed of decision variables $x_i \in \mathcal{X}_i (i \in \{1,2,\cdots,n\})$, $\mathcal{X}_i$ is the set of all possible values of ith decision variable, and n is the dimension of the solution vector, i.e. the number of decision variables. The idea behind HS is inspired by music improvisation process in which a musician tries to find the perfect harmony with examining a wide range of combinations of many possible pitches. Such combinations usually obey the following rules (Das, Mukhopadhyay, Roy, Abraham, & Panigrahi, 2011): (i) An adjacent pitch of one pitch from the memory must be played; (ii) A random pitch from the possible range must be played; (iii) Any one pitch from the memory must be played. Surprisingly, HS algorithm imitates the same procedure during the variable selection step in which all the following constraints must be satisfied: (i) An adjacent value from the HS memory must be chosen; (ii) A random value from the possible range of values must be chosen; (iii) Any value from the HS memory must be chosen. In order to imitate musical improvisation scheme, HS algorithm is composed of the following steps:

(1) *Initializing the parameters*: the set of parameters of HS algorithm include harmony memory size (HMS), harmony memory considering rate (HMCR), pitch adjusting rate (PAR), and termination criterion (e.g. maximum number of improvisations).
(2) *Initializing the harmony memory*: random solution vectors are generated and stored in harmony memory. The capacity of HM is determined by HMS:

$$HM = \left[\begin{array}{c|c} X^1 & f(X^1) \\ X^2 & f(X^2) \\ \vdots & \vdots \\ X^{\mathrm{HMS}} & f(X^{\mathrm{HMS}}) \end{array}\right] = \left[\begin{array}{cccc|c} x_1^1 & x_2^1 & \cdots & x_n^1 & f(X^1) \\ x_1^2 & x_2^2 & \cdots & x_n^2 & f(X^2) \\ \vdots & \vdots & \cdots & \vdots & \vdots \\ x_1^{\mathrm{HMS}} & x_2^{\mathrm{HMS}} & \cdots & x_n^{\mathrm{HMS}} & f(X^{\mathrm{HMS}}) \end{array}\right], \tag{2}$$

where X^i is a candidate solution and $1 \le i \le \mathrm{HMS}$.
(3) *Improvising a new solution*: A new solution vector $X^{new} = [x_1^{new}, \cdots, x_N^{new}]$ is generated based on *Harmony Memory Considering Rate* (HMCR) parameter. The value of

Algorithm 1. Harmony search algorithm.

1: **procedure** HARMONY SEARCH
2: Initialize the harmony memory with HMS random solutions
3: **repeat**
4: Create a new solution as follows:
5: **for all** decision variables x_j^{new} ($j \in \{1, 2, \cdots, n\}$) **do**
6: $x_j^{new} \leftarrow \begin{cases} x_j^k \text{ with probability HMCR for } k \in \{1, 2, \cdots, \text{HMS}\} \\ \text{a random value within the allowed range,} \end{cases}$
7: **if** $x_j^{new} \in$ HM **then**
8: $x_j^{new} = \begin{cases} x_j^{new} \pm rand() \times \text{bw} & \text{with probability PAR} \\ x_j^{new} & \text{with probability 1-PAR,} \end{cases}$
9: **end if**
10: **end for**
11: **if** the new solution is better than the worst solution in HM **then**
12: Replace the worst solution by the new one
13: **end if**
14: **until** Termination condition is fulfilled
15: **return** The best solution in the harmony memory
16: **end procedure**

each decision variable of X^{new} is assigned as follows:

$$x_i^{new} = \begin{cases} x_i^{int(rand()*HMS)+1} & \text{with probability HMCR} \\ \text{a random value within the allowed range} & \text{with probability1-HMCR} \end{cases} \quad \text{for } i \in \{1, \cdots, n\}, \tag{3}$$

where *rand*() is a uniform distribution. In fact, HMCR is the probability of selecting corresponding variable of one of the present HM members and considering it as x_i^{new} and 1-HMCR is the probability of x_i^{new} randomly being assigned.

(4) *Pitch Adjustment*: If x_i^{new} comes from HM, it might be further modified according to *Pitch Adjustment Ratio* (PAR), the probability of mutating the decision variable:

$$x_i^{new} = \begin{cases} x_i^{new} \pm rand() \times \text{bw} & \text{with probability PAR} \\ x_i^{new} & \text{with probability 1-PAR} \end{cases} \quad \text{for } i \in \{1, \cdots, n\}, \tag{4}$$

where bw is the amount of maximum change in pitch adjustment.

(5) *Updating the harmony memory*: Let X^w be the member of HMwith worst fitness:

$$w = \arg \min_{j \in \{1,2,\cdots,\text{HMS}\}} f(X^j). \tag{5}$$

If $f(X^{new}) \le f(X^w)$ then X^w will be replaced by X^{new}.

(6) *Making decision*: Repeat Steps 3–5 until the underlying termination criterion (e.g. maximum iterations) is satisfied.

As mentioned in Geem et al. (2001), HMCR and PAR parameters provide the HS algorithm the opportunity to find locally globally improved solutions, respectively. Although HS algorithm is a random search technique similar to genetic algorithm (Davis, 1991) and particle swarm optimization (PSO) (Clerc, 2010), it is considerably different from those population-based evolutionary methods due to its single evolving search memory. The pseudo-code of HS algorithm is shown in Algorithm 1.

3. Related work

In this section, the some major modifications to HS algorithm along with some of its state-of-the-art variants are briefly discussed. The original HS (Algorithm 1) does not utilize any selection procedure in the search process. The major part of the literature of HS algorithms similarly have no selection scheme. However, recent variants of HS algorithm incorporate a selection scheme in their improvisation phase. The first category of HS algorithms are explained in Section. 3.1, and the latter ones are presented in Section. 3.2.

Algorithm 2. Modification proposed by SAHS algorithm.

1: **for all** decision variables x_i^{new} ($i \in \{1, 2, \cdots, n\}$) **do**
2: $x_j^{new} = \begin{cases} x_j^{new} + \left[\max_j - x_j^{new}\right] \times rand() & \text{with probability p=0.5} \\ x_j^{new} - \left[\min_j - x_j^{new}\right] \times rand() & \text{with probability 1-p} \end{cases} \quad \text{for } j \in \{1, 2, \cdots, n\},$
3: **end for**

Algorithm 3. Modification proposed by GBHS algorithm.

1: **for all** decision variables x_i^{new} ($i \in \{1, 2, \cdots, n\}$) **do**
2: $x_i^{new} = \begin{cases} x_i^{int(rand()*\text{HMS})+1} & \text{with probability HMCR} \\ \min_i + rand() \times (\max_i - \min_i) & \text{with probability 1-HMCR} \end{cases}$
3: **if** x_i^{new} has been chosen from HM **then**
4: $x_i^{new} = x_i^{best}$ with probability PAR
5: **end if**
6: **end for**

3.1. HS variants without selection procedure

3.1.1. Improved harmony search

Authors of Mahdavi et al. (2007) have claimed that fixed values for PAR and bw parameters of the HS algorithm impose limitations on its accuracy and convergence rate. They mentioned that such fixed parameters affect the number of iterations to reach to the optimal solution. To remedy the situation, they proposed *Improved Harmony Search (IHS)*. The main idea behind IHS were the following facts:

- Small PAR along with large bw values might lead to poor performance and large number of iterations for the HS algorithm;
- While small bw values in final generations provide fine-tuned solution vectors, its early values can be larger to enforce the algorithm to increase the diversity of the solution vectors;
- Large PAR along with small bw values possess a high potential to improve the solution vectors during the final generations.

To consider these facts in HS algorithm, they proposed dynamic adjustment of the parameters as follows:

$$\text{PAR}\ (g) = \text{PAR}_{\min} + \frac{\text{PAR}_{\max} - \text{PAR}_{\min}}{\text{NI}} g, \tag{6}$$

$$\text{bw}(g) = \text{bw}_{\max} . \exp\left(\frac{\text{Ln}\left(\frac{\text{bw}_{\min}}{\text{bw}_{\max}}\right)}{\text{NI}} g\right), \tag{7}$$

where g indicates the current generation, $\text{PAR}(g)$ is the pitch adjusting rate of the gth generation, $\text{PAR}_{\min}$ and $\text{PAR}_{\max}$ are the minimum and maximum values of PAR respectively, NI is the maximum number of improvisations, $\text{bw}(g)$ is the distance bandwidth for the gth generation, and $\text{bw}_{\min}$ and $\text{bw}_{\max}$ are the minimum and maximum values of bw respectively.

3.1.2. Self adaptive harmony search

The idea of dynamic adjustment of parameter has also been investigated in *Self Adaptive Harmony Search* (SAHS) algorithm (Wang & Huang, 2010). SAHS algorithm ignores bw parameter and replaces Eq. (4) with the following:where $\max_j$ and $\min_j$ are the highest and lowest values of the jth decision variable in the HM. SAHS provides the chance to produce finer adjustments of HM with gradually approaching to the optimal value.

Algorithm 4. Modification proposed by SAGBHS algorithm.

for all decision variables x_i^{new} $(i \in \{1, 2, \cdots, n\})$ **do**
  $x_i^{new} = \begin{cases} x_i^{int(rand()*\text{HMS})+1} \pm rand().\text{bw} & \text{with probability HMCR} \\ \min_i + rand() \times (\max_i - \min_i) & \text{with probability 1-HMCR} \end{cases}$
  if x_i^{new} has been chosen from HM **then**
    $x_i^{new} = x_i^{best}$ with probability PAR
  end if
end for

Algorithm 5. Modification proposed by NGHS algorithm.

for all decision variables x_i^{new} $(i \in \{1, 2, \cdots, n\})$ **do**
  x_{iL} = the lowest legal value of i^{th}*decisionvariable*
  x_{iU} = the highest legal value of i^{th}*decisionvariable*
  $x_R = 2 \times x_i^{best} - x_i^{worst}$
  if $x_R < x_{iL}$ **then**
    $x_R = x_{iL}$
  end if
  if $x_R > x_{iU}$ **then**
    $x_R = x_{iU}$
  end if
  $x_i^{new} = x_i^{worst} + rand() \times \left(x_R - x_i^{worst}\right)$ ▹ **position updating**
  if $rand() \le p_m$ **then**
    $x_i^{new} = x_{iL} + rand() \times (x_{iU} - x_{iL})$ ▹ **genetic mutation**
  end if
end for

3.1.3. Global-Best Harmony Search

Another considerable modification to original HS algorithm was proposed by *Global-Best Harmony Search* (GBHS) (Omran & Mahdavi, 2008). It was inspired by the search process in PSO(Eberhart & Kennedy, 1995; Kennedy, 2010) in which there exists a stronger bias toward the best solution found so far. In a similar manner, GBHS tries to set decision variables to the corresponding variable in currently best solution found with a higher probability. In fact, GBHS modifies the pitch adjustment step of original HS such that the new harmony mimics the best harmony in the HM. Such modification makes GBHS efficient for both continuous and discrete problems. GBHS requires the same number of iteration as the IHS and modifies Step 3 of HS algorithm as follows:

3.1.4. Self-Adaptive Global-Best Harmony Search

The idea of *GBHS* algorithm was the main inspiration for authors of *Self-Adaptive Global-Best Harmony Search* (SAGBHS) algorithm (Pan, Suganthan, Tasgetiren, & Liang, 2010), in which HMCR and PAR are dynamically adapted by their defined learning mechanisms and bw is decreased when moving to new generations. Their proposed adjustment procedure provided consistent performance on different types of problems. In fact, the dynamic procedure allowed their algorithm to more explore during the initial steps and then exploit within the final stages of the search process. Improvisation procedure of SAGBHS is mentioned in Algorithm 4.

3.1.5. Novel Global Harmony Search

The main idea of PSO is that each individual member of the population tries to mimic its successful companion. This aspect allowed the authors of Zou, Gao, Wu, Li, and Li (2010) to propose a new HS algorithm called *Novel Global Harmony Search* (NGHS) in which the improvisation step is modified as shown in Algorithm 5. The *position updating* step of NGHS algorithm is aimed to make the worst harmony of HM closer to the best one at every iteration. It is trivial that such modification may cause the search process converge to the locally optimal solutions. In order to avoid this problem, *Improved Novel Global Harmony Search* (INGHS) algorithm was proposed in Ouyang et al. (2015). INGHS makes one more modification in the updating operation of the NGHS algorithm. If the fitness of the new improvised harmony is better than that of the randomly selected harmony, the new harmony replaces the selected one.

3.1.6. Island-Based Harmony Search

Island model concepts (Kushida, Hara, Takahama, & Kido, 2013) have inspired authors of Al-Betar, Awadallah, Khader, and Abdalkareem (2015) to propose a new modification to HS algorithm, called *Island-Based Harmony Search* (IBHS). In IBHS algorithm, individual harmonies of HM are divided to I_m sub-groups (islands) of size I_s (Algorithm 7). Therefore, after division step, there exist a set of m harmony memories, i.e. $\text{HM} = (\text{HM}_1, \text{HM}_2, \cdots, \text{HM}_m)$. Then, improvisation process is executed for each island and its HM will be updated separately. After specific generations, a migration process allows a number of individuals to be exchanged between islands. In fact, the migration step allows the interaction between islands and also provides the diversity of the population.

3.2. HS variants with selection procedure

Within EA framework, the fitness value of a solution vector is able to influence the evolution process via a selection procedure. Biological evolution has inspired EA to uses mechanisms such as reproduction, mutation, recombination, and selection (Zimmer & Emlen, 2013). Due to this fact, some HS variants have utilized selection schemes that are designed based on *survival of the fittest* (Spencer, 1896), a way of describing the mechanism of *Natural Selection* (Darwin, 1859).

The HS algorithms of Section 3.1 convey the selection in a random manner because they don't utilize the fitness of the solutions in the improvisation step (Al-Betar, Doush, Khader, & Awadallah, 2012). To equip the HS algorithms with *Natural Selection* characteristic, some recent works have proposed to append a selection scheme into the algorithm (Al-Betar et al., 2012; Al-Betar et al., 2013; Castelli, Silva, Manzoni, & Vanneschi, 2014; Karimi, Askarzadeh, & Rezazadeh, 2012). They have also analyzed the effect of this modification on the performance of the algorithm. Several selection schemes have been implemented and their effects on the HS algorithm are studied in Al-Betar et al. (2012), in which the random selection of the memory consideration step of HS algorithm was substituted by some selection schemes. When the memory consideration step was revised with a new selection step, the overall algorithm was a new variant of HS algorithm. The authors examined different selection schemes including random selection (Geem et al., 2001), global-best selection (Omran & Mahdavi, 2008), proportional selection (Holland, 1992), tournament selection (Goldberg, Korb, & Deb, 1989), linear ranking selection (Baker, 1985) and exponential ranking selection (Hancock, 1994). One of the most promising proposed variants was based on tournament selection and was called *tournament harmony search* (THS).

Therefore, the selective variants of HS algorithm avoid the random selection of a harmony and utilize a selection scheme to choose a harmony for memory consideration process. one of the most promising selection-based HS algorithms is called *Tournament Selection-based Harmony Search* (TSHS) algorithm (Karimi et al., 2012), which exploits the *Tournament Selection* scheme (Miller & Goldberg, 1995). Except the selection step, other parts of TSHS algorithm are similar to IHS algorithm. In the selection step, a number of harmonies are chosen to participate in a tournament. Then, the harmony with the best quality is the winner of the tournament and it is the selected harmony for improvisation step. The parameter determining the number of chosen harmonies to participate in the tournament is called *tournament size*. It is necessary to note that the harmonies with better fitness are more probable to be selected in tournament selection scheme. In Castelli et al. (2014), the authors proposed *Geometric Selective Harmony*

Algorithm 6. Modification proposed by INGHS algorithm.

for all decision variables x_i^{new} $(i \in \{1, 2, \cdots, n\})$ **do**
  x_{iL} = the lowest legal value of $i^{th}decisionvariable$
  x_{iU} = the highest legal value of $i^{th}decisionvariable$
  if $rand() \leq O(k)$ **then**
    $x_R = 2 \times x_i^{best} - x_i^{worst}$
  else
    $x_R = (1 + rand()) \times x_i^{best} - (1 - rand()) x_i^{int(rand()*\mathrm{HMS})+1}$
  end if
  if $x_R < x_{iL}$ **then**
    $x_R = x_{iL}$
  end if
  if $x_R > x_{iU}$ **then**
    $x_R = x_{iU}$
  end if
  $x_i^{new} = x_i^{int(rand()*\mathrm{HMS})+1} + rand() \times \left(x_R - x_i^{int(rand()*\mathrm{HMS})+1}\right)$ ▹ **position updating**
  if $rand() \leq p_m$ **then**
    $x_i^{new} = x_{iL} + rand() \times (x_{iU} - x_{iL})$ ▹ **genetic mutation**
  end if
end for

Algorithm 7. Division process of Island-Based Harmony Search.

```
1:  procedure DIVIDEHM
2:      for i = 1 : HMS do
3:          repeat
4:              k = U(1 ··· I_m)
5:              if HM_k ≤ I_s then
6:                  I_j = k
7:              end if
8:          until I_j is assigned to the island k
9:      end for
10: end procedure
```

Algorithm 8. Modification proposed by GSHS algorithm.

for all decision variables x_i^{new} $(i \in \{1, 2, \cdots, n\})$ **do**
  $X^{p_{i_1}}$ = tournament_selection (HM)
  $X^{p_{i_2}}$ = tournament_selection (HM)
  $\alpha = rand()$

$$x_i^{new} = \begin{cases} \alpha X^{p_{i_1}} + (1-\alpha) X^{p_{i_2}} & \text{with probability HMCR} \\ \min_i + rand() \times (\max_i - \min_i) & \text{with probability 1-HMCR} \end{cases}$$

end for

Search (GSHS) and compared it with THS and TSHS methods. The experimental results proved that GSHS outperforms both THS and TSHS methods for many benchmark problems. GSHShas established following modifications to the HS algorithm: (1) Selection procedure is inspired by the tournament selection of Genetic Algorithms; (2) A memory consideration process is proposed using a recombination operator; (3) A new mutation operator is also suggested. During the memory consideration process, to assign the ith decision variable of the new harmony X^{new}, GSHS has defined two harmonies $X^{p_{i_1}}$ and $X^{p_{i_2}}$ that are selected by a tournament of size *ts*. As it has been experienced in EA (Bäck, 1994), the selection procedure is generally expected to make a trade-off between exploration and exploitation of the underlying algorithm and promote its convergence to the optimal solution. To achieve this goal, GSHS offers the improvisation procedure shown in Algorithm 8. Similar to the evolutionary tournament selection, the tournament_selection(HM) first selects *ts* harmonies from the HM randomly and then returns the one with the best fitness.

4. Selective Refining Harmony Search

Exploration and exploitation are the two cornerstones of EA (Črepinšek, Liu, & Mernik, 2013). The former is aimed to search for new solutions in new regions and span the search spaces as widely as possible. The latter means intensifying and relying on existing solutions to make the search space narrow enough to make the algorithm converge to optimal solution. As an EA progresses, the new solutions are expected to become more reliable (with higher fitness). Due to this fact, while exploration is more desired in the early stages of an EA, it makes sense to exploit the search space in the later steps. In other words, the obtained knowledge and experience of the population increases iteratively and exploitation becomes more effective in the subsequent iterations. It is interesting that experience is utilized in musical improvisation in similar manner to EA. Musicians rely on their knowledge to improvise desirable harmonies. The primary question is how musicians modify the undesirable harmonies. If a harmony does not make desired voice, the first option might be to explore a different set of notes iteratively until the desired one is obtained. However, experienced musicians exploit a more tricky way to improvise a perfect harmony. They usually doesn't throw away the whole harmony, but they are more favored to refine it. In fact, they exploit their experience to detect undesirable notes of the underlying harmony and then try to replace them with some appropriate ones. In order to incorporate musicians' refinement procedure into the HS algorithm, we propose the *Selective Refining Harmony Search* (SRHS) algorithm which is constructed by three major steps:

(i) *Representation*: Provides an appropriate representation for existing knowledge and experience of the algorithm.
(ii) *Selection*: Is focused on how to select some harmonies for refinement.
(iii) *Refinement*: Provides a procedure to refine the selected harmonies.

Representation: In global best PSO systems, first a swarm of particles traverse through the search space. Then the position of each particle is determined with respect to its own experience (current position) and the experience of the swarm (the position

of the best particle) (Eberhart & Kennedy, 1995; Kennedy, 2010). Inspired by this phenomena in PSO models, some HS variants have successfully relied on the best harmony to improve the performance of the HS algorithm (Omran & Mahdavi, 2008; Ouyang et al., 2015; Wang & Huang, 2010). Such successful utilizations of best harmony in recent HS algorithms inspired SRHS algorithm to pick the best harmony as a candid representative of the current experience and knowledge accumulated in HM.

Selection: Selection is a key factor of EA where the evolution process is influenced by fitness values (Al-Betar et al., 2012). Reliable selection methods are mainly based on the Darwinian principle of natural selection, i.e. *Survival of the Fittest*. The selection procedure in EA is responsible to make the search process consider better individuals with a higher probability, enforce an appropriate diversity onto the population and avoid premature convergence (Bäck & Hoffmeister, 1991; Miller & Goldberg, 1996). In order to get equipped with such desirable characteristics, , SRHS algorithm includes a selection step. There exist well-known selection schemes for HS algorithm including random selection (Geem et al., 2001), global-best selection (Omran & Mahdavi, 2008), proportional selection (Holland, 1992), tournament selection (Goldberg et al., 1989), linear ranking selection (Baker, 1985) and exponential ranking selection (Hancock, 1994). In this paper, tournament selection scheme has been utilized due to its robustness and popularity in the literature (Miller & Goldberg, 1996).

Refinement: Let's assume that enough number of generations have been improvised by the HS algorithm and thus there exist reliable experience and knowledge in HM. Also suppose that $X^{\min} = [1, 1, \cdots, 1, 1]$ is the optimal solution for the underlying objective function ($f(X^{\min}) \leq f(X) \forall X \in \text{HM}$). Moreover, let X^1 and X^2 be two solution vectors of HM with corresponding fitness values as follows:

$$X^1 = [1.0, 1.0, \cdots, 1.0, 100] \Rightarrow f(X^1) = 9.9, \tag{8}$$

$$X^2 = [1.5, 1.5, \cdots, 1.5, 1.5] \Rightarrow f(X^2) = 8.5. \tag{9}$$

Although X^2 has a better fitness than X^1, it is obvious that X^1 has a higher chance to get closed to the optimal solution by minor modifications because it is different from $X^{\min}$ just in the last variable. Despite the fact depicted by this example, almost all HS algorithms evaluate and select the harmonies just based on their fitness values. In other words, despite to musicians that don't throw away the whole harmony and refine the undesirable notes, HS algorithms decide just based on the fitness values and a harmony even with all, except few, desired decision variables might be ignored. So, we have modified the HS algorithm such that decisions are made in the decision variable level.

The refinement procedure proposed by SRHS is shown in Algorithm 9. Every decision variable x_i of X is replaced by a decision variable x_j of X^{best} that provides maximum change in the fitness of X. It is obvious that X^{best} plays the role of musicians' experience for SRHS algorithm. This refinement procedure allows the memory updating step to modify unsuitable decision variables of each harmony with the most appropriate variable of the best harmony. As it can be deduced from Algorithm 9, the complexity of refining one harmony is $\mathcal{O}(n^2)$, in which n is the number of decision variables. The overall procedure of SRHS is depicted in Algorithm 10.

Despite to depicted advantages of refinement procedure, Algorithm 10 suffers from a considerable problem. The computational complexity, in terms of number of fitness evaluations, is increased by the comparisons performed in refinement step between all the variables of X^{best} and all the variables of each selected HM member at every iteration. The complexity of SRHS will

Algorithm 9. Refinement step of SRHS algorithm.

$X^{best} = \arg\min_{X \in \text{HM}} f(X)$
for $i = 1, 2, \cdots, n$ **do**
  for $j = 1, 2, \cdots, n$ **do**
    $X^{modified} = X$
    $x_i^{modified} = x_j^{best}$
    if $f(X^{modified}) \leq f(X)$ **then**
      $X = X^{modified}$
    end if
  end for
end for

be as follows:

$$\text{Complexity(SRHS)} \propto \mathcal{O}\left(\text{NI} + \text{NS} \times \text{NI} \times n^2\right), \tag{10}$$

where NS is the number of selected harmonies to be refined, NI is the number of iterations of HS algorithm and n is the number of decision variables (dimension) of a harmony. It is obvious that such complexity might be undesired for many common problems. In order to overcome this problem, two following methods will be integrated into the algorithm:

(I) *Iteration Selection*: This method determines the iterations in which the refinement action must be executed. Performing refinement not in all, but some iterations, is one way to simultaneously make utilize the refinement advantageous and also prevent too much increase in complexity. As a result, the number of iterations containing refinement decreases from NI to mod(NI, RP), where RP is the *Refinement Period* parameter that controls the period in which refinement occurs. The *Iteration Selection* procedure is depicted in Algorithm 11.

(II) *Subset Selection*: Another way to decrease the computational complexity imposed by refinement is utilizing a subset, not all, of decision variables of X^{best}. This subset of decision variables is selected by the *Subset Selection* method. Similar to *Iteration Selection*, this method allows the algorithm to avoid too much complexity without losing the benefits of refinement. The number of selected decision variables is determined by a parameter called *Subset Size (SS)*, which is usually considered much smaller than n. The *Subset Selection* procedure is depicted in Algorithm 12.

Therefore, to avoid too much fitness evaluations consumed by 10, *Iteration Selection* and *Subset Selection* steps will be also included by SRHS algorithm. Given such modification, the number of necessary fitness evaluations consumed by SRHS algorithm changes from $\mathcal{O}\left(\text{NI} + \text{NS} \times \text{NI} \times n^2\right)$ into:

$$\text{Complexity(SRHS)} \propto \mathcal{O}\left(\text{NS} \times \lfloor \frac{\text{NI}}{\text{RP}} \rfloor \times n \times \text{SS}\right). \tag{11}$$

Utilizing RP and SS parameters, the final version of SRHS algorithm is depicted in Algorithm 13. As it can be observed in the algorithm, to even utilize the best harmony more, SRHS employs the pitch adjustment proposed in Omran and Mahdavi (2008).

5. Experimental results

This section evaluates the performance of SRHS algorithm and compares it with several HS algorithms. To make comprehensive comparisons, we have implemented original HS algorithm (Geem et al., 2001) and some of its state-of-the-art variations including IHS (Mahdavi et al., 2007), SAHS (Wang & Huang, 2010), GBHS (Omran & Mahdavi, 2008), NGHS (Zou et al., 2010), and GSHS (Castelli et al., 2014). All of the algorithms were examined over

Algorithm 10. Selective Refining Harmony Search algorithm.

procedure MHS
  Initialize the harmony memory with HMS random solutions
  $g = 0$
  repeat
    Create a new solution as follows:
    for all decision variables x_j^{new} $(j \in \{1, 2, \cdots, n\})$ **do**
      $x_j^{new} \leftarrow \begin{cases} x_j^k \text{ with probability HMCR for } k \in \{1, 2, \cdots, \text{HMS}\} \\ \text{a random value within the allowed range,} \end{cases}$
      if x_j^{new} comes from HM **then**
        $x_j^{new} = \begin{cases} x_j^{new} \pm rand() \times \text{bw} & \text{with probability PAR} \\ x_j^{new} & \text{with probability 1-PAR,} \end{cases}$
      end if
    end for
    if the new solution is better than the worst solution in HM **then**
      Replace the worst solution by the new one
    end if
    $X^{best} = \arg\min_{X \in \text{HM}} f(X)$ ▷ **Representation Step**
    $\mathcal{X}$ = tournament_selection(HM) ▷ **Selection Step**
    for all solution vectors $X \in \mathcal{X}$ **do** ▷ **Population Refinement Step: Start**
      for $i = 1, 2, \cdots, n$ **do**
        for $j = 1, 2, \cdots, n$ **do**
          $X^{modified} = X$
          $x_i^{modified} = x_j^{best}$
          if $f(X^{modified}) \leq f(X)$ **then**
            $X = X^{modified}$
          end if
        end for
      end for
    end for ▷ **Population Refinement Step:End**
    $g = g + 1$
  until Termination condition is fulfilled
  return The best solution in the harmony memory
end procedure

Algorithm 11. Iteration selection.

procedure ITERATION_SELECTION(NI,RP)
  $\mathcal{I} = \emptyset$
  for $i = 1, 2, \cdots, \text{NI}$ **do**
    if $mod(i, \text{RP}) == 0$ **then**
      $\mathcal{I} = \mathcal{I} \bigcup \{i\}$
    end if
  end for
  return $\mathcal{I}$
end procedure

Algorithm 12. Subset selection.

procedure SUBSET_SELECTION(X,SS)
  index = rand(0,n − SS);
  $\mathcal{D} = X(index : index + \text{SS} - 1)$
  return $\mathcal{D}$
end procedure

20 benchmark problems of the *IEEE CEC 2010* (Tang, Xiaodong Li, & Zhenyu Yang, 2010), a well-established benchmark suite in the literature. All experiments are conducted on a machine with 12 CPU cores (2.93 GHz intel xeon family 6) and 132GB of RAM. Section. 5.1 explains the benchmark problems along with all details about experimental settings and configurations of related parameters. Comparison results are presented in Section. 5.2. Computational loads of all examined methods are compared in Section 5.3. The sensitivity of SRHS to its parameters is analyzed in Section. 5.4. Finally some concluding remarks are discussed in Section 5.5.

5.1. Benchmark problems

Usually, separable problems are considered the easiest ones to optimize, whereas the fully-nonseparable ones are the most difficult problems. There exist various kinds of partially separable problems that lie between these two extreme cases. In general, real-world optimization problems usually include different groups of parameters with strong within-group dependencies and little between-group interactions. Therefore, we must utilize benchmark problems that contain such dependencies among their parameters. This allows us to ensure that the comparisons made among mentioned algorithms on test problems, will be held in practical scenarios. With this in mind, we have employed *IEEE CEC 2010* (Tang et al., 2010) test suite, which includes four types of high-dimensional problems:

(1) Separable functions;
(2) Partially-separable functions, in which a small number of variables are dependent while the others are independent;
(3) Partially-separable functions that consist of multiple independent subcomponents. Each subcomponent is m-nonseparable; and
(4) Fully-nonseparable functions in which all parameters are coupled together.

This suite contains a diverse set of problems with different characteristics. Some problems of the suite are very difficult to optimize because of the too many number of local optima. The list of problems of *IEEE CEC 2010* is presented in Table 1.

The configuration of parameters used in SRHS and all other implemented methods is shown in Table 2. All of the parameters with assigned values within the experiments are replicated. Among all mentioned parameters in Table 2, there is a special aspect about the parameter NI. The value assigned to parameter NI in SRHS is

Algorithm 13. Selective Refining Harmony Search algorithm: enhanced.

procedure SRHS
  Initialize the harmony memory with HMS random solutions
  $g = 0$
  $\mathcal{I}$ = iteration_selection(NI)
  repeat
    Create a new solution as follows:
    for all decision variables x_j^{new} $(j \in \{1, 2, \cdots, n\})$ **do**
      $x_j^{new} \leftarrow \begin{cases} x_j^k \text{ with probability HMCR for } k \in \{1, 2, \cdots, \text{HMS}\} \\ \text{a random value within the allowed range,} \end{cases}$
      if x_i^{new} has been chosen from HM **then** ▹ **Pitch Adjustment of (Omran and Mahdavi, 2008)**
        $x_i^{new} = x_i^{best}$ with probability PAR
      end if
    end for
    if the new solution is better than the worst solution in HM **then**
      Replace the worst solution by the new one
    end if
    if $g \in \mathcal{I}$ **then** ▹ **Representation Step**
      $X^{best} = \arg\min_{X \in \text{HM}} f(X)$
      $\mathcal{X}$ = tournament_selection(HM) ▹ **Selection Step**
      for all solution vectors $X \in \mathcal{X}$ **do** ▹ **Population Refinement Step: Start**
        for $i = 1, 2, \cdots, n$ **do**
          $\mathcal{D}$ = subset_selection(X^{best})
          for all decision variables $x_j^{best} \in \mathcal{D}$ **do**
            $X^{modified} = X$
            $x_i^{modified} = x_j^{best}$
            if $f(X^{modified}) \leq f(X)$ **then**
              $X = X^{modified}$
            end if
          end for
        end for
      end for
    end if ▹ **Population Refinement Step:End**
    $g = g + 1$
  until Termination condition is fulfilled
  return The best solution in the harmony memory
end procedure

Table 1
Benchmark functions.

Function Category	Problem	Description
Separable	f_1	Shifted Elliptic Function
	f_2	Shifted Rastrigin's Function
	f_3	Shifted Ackley's Function
Single-group m-nonseparable	f_4	Single-group Shifted and m-rotated Elliptic Function
	f_5	Single-group Shifted and m-rotated Rastrigin's Function
	f_6	Single-group Shifted and m-rotated Ackley's Function
	f_7	Single-group Shiftedm-dimensional Schwefel's Problem
	f_8	Single-group Shiftedm-dimensional Rosenbrock's Function
$\frac{D}{2m}$-group m-nonseparable	f_9	$\frac{D}{2m}$-group Shifted and m-rotated Elliptic Function
	f_{10}	$\frac{D}{2m}$-group Shifted and m-rotated Rastrigin's Function
	f_{11}	$\frac{D}{2m}$-group Shifted and m-rotated Ackley's Function
	f_{12}	$\frac{D}{2m}$-group Shifted and m-dimensional Schwefel's Problem
	f_{13}	$\frac{D}{2m}$-group Shifted and m-dimensional Rosenbrock's Function
$\frac{D}{m}$-group m-nonseparable	f_{14}	$\frac{D}{m}$-group Shifted and m-rotated Elliptic Function
	f_{15}	$\frac{D}{m}$-group Shifted and m-rotated Rastrigin's Function
	f_{16}	$\frac{D}{m}$-group Shifted and m-rotated Ackley's Function
	f_{17}	$\frac{D}{m}$-group Shifted and m-dimensional Schwefel's Problem
	f_{18}	$\frac{D}{m}$-group Shifted and m-dimensional Rosenbrock's Function
Nonseparable	f_{19}	Shifted Schwefel's Problem
	f_{20}	Shifted Rosenbrock's Function

smaller than its value in other algorithms. In fact, NI has been encountered differently for SRHS algorithm to prevent unfair comparisons. As everyone knows, it is highly important to establish theories through correct and fair experimentations (Črepinšek, Liu, & Mernik, 2014; Mernik, Liu, Karaboga, & Črepinšek, 2015). The authors of Črepinšek, Liu, Mernik, and Mernik (2015) have mentioned that inexact replications of experiments must be avoided when comparing meta-heuristic algorithms as much as possible. They also mentioned the number of consumed fitness evaluations as one of the most important aspects that must be considered fairly, when comparing evolutionary algorithms (Črepinšek et al., 2015). The number of fitness evaluations consumed by SRHS algorithm is equal to $\mathcal{O}\left(\text{NS} \times \lfloor \frac{\text{NI}}{\text{RP}} \rfloor \times n \times \text{SS}\right)$. Given the mentioned settings (Table 2), SRHS consumes 65,000 fitness evaluations with $\text{NI} = 15,000$. However, the number of fitness evaluations of other algorithms is equal to NI. In other words, given the same NI for all algorithms, SRHS consumes a higher number of fitness evaluations than other methods. To provide fair comparisons, NI is set to $15,000$ for SRHS and $65,000$ for other algorithms.

5.2. Performance comparison

In this section, SRHS has been compared to other HS algorithms. Due to the randomness of the HS-related methods, 51 runs have been drawn for all methods and the statistics of the results are presented. The statistics of fitness values for each method include mean, standard deviation, best and worst. We

Table 2
Parameter settings.

Parameter	Value(s)
Harmony Memory Size (HMS)	7
Number of Decision Variables (NVAR or n)	1000
Harmony Memory Considering Rate (HMCR)	0.8
Pitch Adjusting Rate (PAR)	0.3
Minimum Pitch Adjusting Rate (PAR_{min})	0.01
Maximum Pitch Adjusting Rate (PAR_{max})	0.99
Minimum Bandwidth (bw_{min})	0.0001
Maximum Bandwidth (bw_{max})	$\frac{1}{x_U - x_L}$
Number of Improvisations (NI)	15000 for SRHS and 65000 for other algorithms*
Tournament Size(TS)	3
Refinement Period (RP)	10,000
Sample Size (SS)	50
Number of harmonies to be refined (NS)	1

Table 3
HS algorithms implemented for experimental analyses.

Algorithm	Method
HS	Harmony Search (Geem et al., 2001)
IHS	Improved Harmony Search (Mahdavi et al., 2007)
GBHS	Global-Best Harmony Search (Omran & Mahdavi, 2008)
SAHS	Self Adaptive Harmony Search (Wang & Huang, 2010)
NGHS	Novel Global Harmony Search (Zou et al., 2010)
GSHS	Geometric Selective Harmony Search (Castelli et al., 2014)

have implemented some HS benchmark variants along with some state-of-the-art ones as shown in Table 3. Within the selection free HS algorithms, SRHS is compared to original HS, IHS, GBHS, SAHS and NGHS methods. GSHS algorithm has been chosen as the representative of selection-based variants. Since it has been shown that GSHS outperforms other selective methods like THS and TSHS (Castelli et al., 2014), within our experiments it has been implemented as the dominant representative of selection-based HS methods.

As mentioned in Section 5.1, *IEEE CEC 2010* suite is constituted of 20 benchmark problems that are divided into five categories: separable, single-group m-nonseparable, $\frac{D}{2m}$-group m-nonseparable, $\frac{D}{m}$-group m-nonseparable and nonseparable. Implemented HS algorithms have been examined on all 20 test problems. The obtained results have been shown in Tables 4–8.

In general, the complexity of an optimization problem is proportional to its dimension such that it is far more difficult to find optimums of high dimensional problems (Bellman, Bellman, Bellman, & Bellman, 1961). However, There exits particular conditions that make even high dimensional problems to some extent easy to solve. For example, if dimensions of the solution vectors are independent of each other, the main problem can be decomposed into multiple sub-problems. Each sub-problem involves optimization over one decision variable while other decision variables are remained constant. The class of problems with mentioned independence property is known as separable problems (Auger, Hansen, Mauny, Ros, & Schoenauer, 2007). The performance of SRHS algorithm and other methods for separable problems is shown in Table 4. It can be observed that SRHS provides fitness values better than other methods in average for problem f_1. Moreover, its best and worst fitness values are also better than all other methods for this problem. Next to SRHS, GBHS produces the most appropriate mean, best and worst fitness values. When it comes to standard deviation, SRHS is not superior to other methods. GSHS and IHS are the two first methods with lowest deviations in fitness values. The obtained results for f_2 are similar to those obtained for f_1. Again, fitness values of SRHS are more desirable than other methods because of its better mean value. It has more suitable best and worst fitness values for f_2 problem. Once more, GBHS algorithm is the method ranked second with respect to mean, best and worst cases over f_2 function. Similar to the results for f_1, SRHS is not the superior method according to standard deviations. For problem f_2, GSHS and IHS are two first algorithm yielding the lowest standard deviation. The obtained results for problem f_3 almost match to those of f_1 and f_2 problems. SRHS has most proper mean, worst and best fitness values for f_3. After SRHS, GBHS exhibits better mean, best and worst fitnesses compared to other methods. Instead of SRHS, SAHS and IHS methods produced fitness values with the lowest standard deviations for f_3 problem.

Given the comparison results for separable functions, SRHS provides the most appropriate mean, worst and best fitness values for all three separable problems. Table 4 revealed that SRHS is not the best method just according to standard deviation criteria. This characteristic of SRHS may come from the fact that SRHS has a higher degree of randomness compared to other HS variants. As it can be deduced from Algorithm 10, SRHS has attached a random subset selection step to HS algorithm that makes the fitness values more varying.

A nonseparable function is considered as a fully-nonseparable function if any of two its parameters are not independent. As mentioned above, the separability criteria determines the extent to which the problem is difficult to solve. In other words, nonseparable functions are most difficult problems to optimize due to the coupling that exists between their parameters. The experimental results for nonseparable functions are depicted in Table 5. There are two test functions f_{19} and f_{20}. When comparing the statistics of fitness values for problem f_{19}, it can be observed that SRHS has been constantly better than other methods. Mean, best and worst of fitness values of SRHS are more appropriate than others for this function. After SRHS, GBHS and GSHS algorithms has shown the most appropriate performances for f_{19} problem, respectively. For f_{20} problem, mean, best and worst fitness values of SRHS are better than those of other methods and next to it GBHS is the second best method. It seems that SRHS reveales more appropriate standard deviation for nonseparable functions than for separable ones. For nonseparable function f_{19}, standard deviation of SRHS is lower than all other methods and GBHS is the second method with lowest deviation. Although SRHS does not possess the lowest deviation for f_{20}, it is the second method after GBHS. Therefore, comparing the results depicted in Tables 4 and 5, it is proved that performance of SRHS algorithm is better for nonseparable functions than for separable problems.

Various types of partially separable functions can be utilized to analyze the performance of algorithms in between two extreme cases of separable and fully-nonseparable functions (Griewank & Toint, 1982). In practice, objective function of real-world optimization problems consists of clusters of parameters with high within and low between cluster dependencies (Tang et al., 2010). This phenomena is reflected in *IEEE CEC 2010* with the introduction of partially-separable functions. In order to produce such problems, parameters are first assigned to different clusters. Then, for each cluster the variables are decided to be dependent or independent.

The next category of test problems includes m-nonseparable functions. These problems are nonseparable functions with at most m nonindependent parameters. This category includes both shifted unimodal and multimodal test problems. Statistics of the obtained results for this category of functions are shown in Table 6. The first function of the category is f_4 called *Single-group Shifted and m-rotated Elliptic Function*. The experiments demonstrated that mean, best and worst fitness values generated by SRHS are more beneficial than other methods for this problem. After SRHS algorithm, GBHS produced next mean, best and worst fitness values. Although SRHS is not the method with lowest variations in fitness values, it is ranked second after GSHS according to obtained standard devi-

Table 4
Fitness values for category of separable functions.

Problem	Fitness	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_1	mean	1.066863E11	1.070119E11	4.526664E10	1.419054E11	2.45089E11	9.216948E10	**1.247741E10**
	std	6.05347E9	4.48593E9	**2.745057E9**	4.317981E9	1.933125E10	4.546117E9	5.0882E9
	best	9.036875E10	9.737452E10	3.838337E10	1.343646E11	2.046613E11	7.744879E10	**5.230155E9**
	worst	1.189569E11	1.16452E11	5.034045E10	1.57622E11	2.861131E11	1.00049E11	**3.058692E10**
f_2	mean	1.791134E4	1.792043E4	1.309101E4	1.984757E4	1.893789E4	1.728503E4	**2.93325E3**
	std	1.820724E2	1.557008E2	1.562081E2	2.132755E2	2.283827E2	**1.533215E2**	1.018951E3
	best	1.737875E4	1.755365E4	1.280729E4	1.943069E4	1.83659E4	1.671766E4	**2.227243E3**
	worst	1.825856E4	1.834548E4	1.347286E4	2.026915E4	1.937573E4	1.758916E4	**5.593051E3**
f_3	mean	2.113424E1	2.113424E1	2.053363E1	2.127957E1	2.128091E1	2.115158E1	**1.530481E1**
	std	1.781372E−2	1.285721E−2	4.714934E−2	**1.229205E-2**	2.549426E−2	2.166297E−2	1.321188E0
	best	2.105998E1	2.110682E1	2.042451E1	2.125918E1	2.121328E1	2.110173E1	**1.350195E1**
	worst	2.116022E1	2.11657E1	2.06538E1	2.130292E1	2.133655E1	2.120667E1	**1.851869E1**

Table 5
Fitness values for category of nonseparable functions.

Problem	Fitness	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_{19}	mean	4.940754E7	5.068611E7	3.108686E7	6.933636E7	1.308182E8	3.648773E7	**1.465916E7**
	std	4.942569E6	5.019079E6	3.354141E6	7.996354E6	3.47872E7	3.36868E6	**3.231652E6**
	best	3.546776E7	3.889341E7	2.471397E7	5.389771E7	7.684002E7	2.471708E7	**9.832108E6**
	worst	6.085569E7	5.986493E7	3.824505E7	8.289908E7	2.596269E8	4.252389E7	**2.288571E7**
f_{20}	mean	3.70268E12	3.676698E12	8.556263E11	4.917577E12	3.632894E12	2.656115E12	**1.305559E11**
	std	1.137572E11	1.171115E11	**3.083083E10**	1.321775E11	1.273782E11	1.010248E11	7.435749E10
	best	3.385991E12	3.40314E12	8.040281E11	4.705804E12	3.335519E12	2.455571E12	**6.193702E10**
	worst	3.98029E12	3.86494E12	9.77193E11	5.175821E12	3.863205E12	2.868779E12	**3.489898E11**

Table 6
Fitness values for category of single-group m-nonseparable functions.

Problem	Fitness	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_4	mean	4.504266E14	4.867815E14	2.368543E14	6.754532E14	5.52533E15	2.667785E14	**1.944188E14**
	std	1.051361E14	9.447829E13	7.067113E13	1.348722E14	1.715443E15	**5.724101E13**	5.906014E13
	best	2.359441E14	2.568657E14	1.11009E14	3.845249E14	2.506035E15	1.74031E14	**1.023736E14**
	worst	7.202767E14	6.908181E14	3.945823E14	1.005609E15	1.004835E16	4.62046E14	**3.498734E14**
f_5	mean	4.658049E8	4.645725E8	4.103485E8	4.975817E8	7.667285E8	4.682087E8	**3.539655E8**
	std	2.390363E7	**2.082447E7**	6.161791E7	2.560079E7	5.273541E7	2.548939E7	5.760978E7
	best	4.014207E8	4.054421E8	2.993045E8	4.299641E8	6.38965E8	4.038705E8	**2.257338E8**
	worst	5.311363E8	5.069993E8	5.421335E8	5.368507E8	8.553665E8	5.219919E8	**4.661294E8**
f_6	mean	1.28559E7	1.309042E7	1.171697E7	1.487423E7	2.059229E7	1.176203E7	**1.166505E7**
	std	5.756797E5	5.067207E5	1.751834E6	4.782195E5	**1.563666E5**	4.96981E5	1.891673E6
	best	1.189292E7	1.188952E7	8.567494E6	1.381154E7	2.011071E7	1.042742E7	**7.851949E6**
	worst	1.421852E7	1.405377E7	1.79191E7	1.587118E7	2.087381E7	**1.286196E7**	1.747816E7
f_7	mean	6.698044E10	6.762783E10	4.839154E10	1.014701E11	2.73404E12	4.955274E10	**4.638889E10**
	std	1.021626E10	1.133562E10	7.948418E9	1.315667E10	7.531368E11	8.774136E9	**7.01217E9**
	best	4.086824E10	4.335932E10	3.19715E10	7.520076E10	1.506647E12	**3.168877E10**	3.534871E10
	worst	8.748941E10	9.830212E10	7.339568E10	1.318484E11	4.797717E12	7.047239E10	**6.460992E10**
f_8	mean	5.582752E14	6.508102E14	3.862213E14	1.802478E15	6.735565E16	4.048966E14	**2.846766E14**
	std	1.378804E14	1.320636E14	2.382299E14	4.374931E14	1.399116E16	**1.209133E14**	2.562655E14
	best	2.966776E14	3.450646E14	1.008191E14	1.098732E15	3.805531E16	1.580181E14	**2.21577E13**
	worst	9.927221E14	9.946583E14	9.967106E14	2.777537E15	1.031422E17	**7.566161E14**	1.449109E15

ations for f_4. For f_5 problem, SRHS is the first method with most desirable average fitness values and next to it GBHS is better than other methods. Two first best fitness values are those produced by SRHS and GBHS. According to worst fitness values, SRHS and IHS are ranked first and second, respectively. It is interesting that after the IHS, original HS algortihm has produced results with smallest variations. For f_6 problem, SRHS has produced the most desirable mean and best fitness values and next to it is GBHS algorithm. GSHS and IHS algorithms are the two first ones with superior worst case fitnesses. NGHS and SAHS are the methods with lowest variations in fitness values within 51 runs. Results obtained over f_7 problem also proves the superiority of SRHS algorithm. With respect to average of generated fitness values, SRHS and GBHS are the best algorithms. These methods are again two first ones according to best fitness values, respectively. Worst case of SRHS is better than all other methods for f_7 function. After SRHS, the method with better worst fitness is GSHS algorithm. Unlike the previous problems, SRHS is the algorithm with lowest variations in the generated fitness values. After SRHS, GBHS is the second method with minimum standard deviation. In the case of f_8 test problem, SRHS has shown superior performance according to mean and best fitness measures and the next outperforming method is GBHS method. GSHS and IHS provided lowest variations in fitness

Table 7
Fitness values for category of $\frac{d}{2m}$-group m-nonseparable functions.

Problem	Fitness	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_9	mean	1.497344E11	1.475499E11	5.556283E10	1.844882E11	2.719067E11	1.210156E11	**6.644236E9**
	std	7.321596E9	5.760122E9	3.340207E9	8.476749E9	1.596963E10	5.181691E9	**3.30108E9**
	best	1.337454E11	1.353286E11	4.827565E10	1.649154E11	2.480754E11	1.087793E11	**3.703954E9**
	worst	1.629778E11	1.591303E11	6.329239E10	2.038445E11	3.13732E11	1.339905E11	**1.813489E10**
f_{10}	mean	1.875406E4	1.869964E4	1.52128E4	2.048106E4	1.931525E4	1.768677E4	**5.664637E3**
	std	**1.758771E2**	2.23459E2	2.034716E2	2.261166E2	2.48385E2	2.009016E2	8.56949E2
	best	1.816488E4	1.803291E4	1.484375E4	1.99381E4	1.866151E4	1.719509E4	**5.016251E3**
	worst	1.909332E4	1.911075E4	1.577443E4	2.093145E4	1.979335E4	1.819718E4	**8.263027E3**
f_{11}	mean	2.294709E2	2.29508E2	2.217223E2	2.335691E2	2.330116E2	2.272616E2	**2.064682E2**
	std	8.713161E−1	8.795501E−1	1.430442E0	7.250767E−1	**3.347494E-1**	1.020947E0	3.98854E0
	best	2.276883E2	2.281216E2	2.194361E2	2.312073E2	2.322532E2	2.243997E2	**1.992195E2**
	worst	2.314512E2	2.319079E2	2.241909E2	2.348308E2	2.341309E2	2.288001E2	**2.164436E2**
f_{12}	mean	1.237169E7	1.22565E7	5.953653E6	1.50547E7	2.336382E7	9.175077E6	**1.071689E6**
	std	6.535212E5	7.105229E5	3.245061E5	9.051791E5	2.826285E6	5.877006E5	**2.442357E5**
	best	1.070055E7	1.061867E7	5.153509E6	1.303044E7	1.597888E7	7.532779E6	**9.172948E5**
	worst	1.388563E7	1.369662E7	6.867481E6	1.695585E7	2.993299E7	1.056158E7	**1.800921E6**
f_{13}	mean	1.067721E12	1.074342E12	2.095703E11	1.477358E12	1.453361E12	7.406881E11	**4.297002E10**
	std	4.848619E10	4.784887E10	**1.253382E10**	8.460166E10	1.166689E11	4.333918E10	2.829466E10
	best	9.556686E11	9.79826E11	1.796329E11	1.3147E12	1.209779E12	6.160608E11	**1.426259E10**
	worst	1.158819E12	1.180059E12	2.312362E11	1.622912E12	1.68357E12	8.363057E11	**1.030212E11**

Table 8
Fitness values for category of $\frac{d}{m}$-group m-nonseparable functions.

Problem	Fitness	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_{14}	mean	1.773788E11	1.78555E11	6.032641E10	2.222764E11	3.121906E11	1.41537E11	**3.105854E9**
	std	6.541579E9	6.426203E9	3.71264E9	1.087088E10	2.045639E10	4.840691E9	**1.555749E9**
	best	1.618672E11	1.636922E11	5.234799E10	1.856703E11	2.785254E11	1.331481E11	**2.314015E9**
	worst	1.904123E11	1.962305E11	6.899652E10	2.412442E11	3.724371E11	1.520298E11	**8.00446E9**
f_{15}	mean	1.94137E4	1.936666E4	1.635699E4	2.086794E4	1.940384E4	1.804918E4	**8.888651E3**
	std	1.812513E2	1.894362E2	2.480304E2	2.402001E2	2.701053E2	**1.780028E2**	8.839402E2
	best	1.895413E4	1.890136E4	1.577224E4	2.029281E4	1.884406E4	1.772453E4	**7.995277E3**
	worst	1.978461E4	1.969912E4	1.708889E4	2.134128E4	2.021599E4	1.844884E4	**1.178759E4**
f_{16}	mean	4.246718E2	4.247268E2	4.128552E2	4.278277E2	4.244649E2	4.210098E2	**3.972158E2**
	std	9.446802E−1	7.237868E−1	1.119355E0	**4.285993E-1**	4.588315E−1	7.277683E−1	2.517957E0
	best	4.224273E2	4.230321E2	4.094639E2	4.268375E2	4.235201E2	4.191447E2	**3.915071E2**
	worst	4.260965E2	4.258929E2	4.15305E2	4.289262E2	4.254258E2	4.224187E2	**4.050413E2**
f_{17}	mean	2.798257E7	2.850259E7	1.020583E7	3.528503E7	6.547E7	2.022883E7	**2.006271E6**
	std	2.032825E6	1.716238E6	6.929692E5	2.556596E6	7.794489E6	9.644589E5	**4.389325E5**
	best	2.182143E7	2.462471E7	8.287396E6	2.963138E7	4.627299E7	1.78153E7	**1.646732E6**
	worst	3.277823E7	3.239803E7	1.16623E7	4.110221E7	7.995987E7	2.198298E7	**3.557665E6**
f_{18}	mean	3.305769E12	3.34092E12	7.395738E11	4.466177E12	3.092095E12	2.397512E12	**1.234188E11**
	std	9.921413E10	9.316633E10	**2.33282E10**	1.636614E11	1.203445E11	7.624818E10	7.247478E10
	best	3.093938E12	3.139185E12	6.869218E11	4.13481E12	2.839936E12	2.169974E12	**7.15E10**
	worst	3.508409E12	3.654493E12	7.950114E11	4.807112E12	3.410251E12	2.527813E12	**3.307134E11**

values, respectively. GSHS and HS methods are two first methods with respect to worst cases of generated fitness values of 51 runs.

Category of $\frac{d}{2m}$-nonseparable test problems includes those nonseparable functions with at most $\frac{d}{2m}$ nonindependent parameters, where $d = \text{NVAR}$ and m is the parameter determining the number of variables in each group which is in fact the degree of separability. During our experiment m was set to 50. The results of experiments for this category is reported in Table 7. This category is constituted by five problems $f_9 - f_{13}$. For f_9 problem, SRHS is the dominant algorithm because it has provided superior performance for mean, standard deviation, best and worst of fitness values. After SRHS, GBHS is the method with the most appropriate performance among other methods. SRHS and GBHS are two first methods for problem f_{10} according to obtained mean, best and worst performances over their generated fitness values. However, HS and GSHS have shown lowest standard deviations for f_{10}, respectively. The results obtained for f_{11} are very similar to those of problem f_{10}. SRHS and GBHS are ranked first and second respectively according to mean, best and worst of fitness values. NGHS and SAHS has had lowest variations for this problem. Results for f_{12} are quite similar to those of f_9. SRHS and GBHS are two first methods with most proper performances on problem f_{12}. The results for test problem f_{13} also verifies superiority of SRHS over other methods. SRHS and GBHS are ranked first and second respectively for f_{13} problem according to their mean, best and worst performances. Just standard deviation of GBHS is better than that of SRHS.

The last category of test problems within *IEEE CEC 2010* is the set of $\frac{d}{m}$-group m-nonseparable functions. Again, SRHS has demonstrated its superiority over other methods as shown in Table 8. SRHS has thoroughly outperformed competitive methods for test problems f_{14} and f_{17}. It also has confirmed its distinction from others according to mean, best and worst performances for test problems f_{15}, f_{16} and f_{18}. However, as we mentioned above, for some test problems SRHS may produce more varying outputs because of its additional random actions.

As depicted by Tables 4–8, SRHS is always the most appealing algorithm according to the mean, best and worst of fitness values. Sometimes, fitness values generated by SRHS are more varying

Table 9
Statistical test results. The p values are reported.

Test Problem	HS	IHS	GBHS	SAHS	NGHS	GSHS
f_1	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_2	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_3	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_4	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_5	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_6	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_7	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_8	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_9	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{10}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{11}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{12}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{13}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{14}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{15}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{16}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{17}	< 0.01	< 0.01	> 0.01	< 0.01	< 0.01	> 0.01
f_{18}	< 0.01	< 0.01	> 0.01	< 0.01	< 0.01	> 0.01
f_{19}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01
f_{20}	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01	< 0.01

than those of other methods because of the greater amount of randomness embedded in SRHS algorithm. After the comparisons we made between the SRHS and other algorithms, it is crucial to answer a key question: how much the comparisons are statistically valid? The best answer that one might suggest is the *statistical testing* of the comparison results. In the literature of computational intelligence, statistical tests have recently become a widespread approach to make sure that the comparisons, made between different methods, are reliable and satisfying. Statistical tests are considered as an important part of the experimental analysis to decide whether one algorithm is actually better than the other or not? In other words, they are necessary to verify that a new proposed method offers significant improvements over existing methods for the underlying problem (Derrac, García, Molina, & Herrera, 2011). There exist two category of statistical tests: parametric and non-parametric. Parametric tests are usually based on assumptions that might be violated in practice. Those assumptions include independence, normality, and homoscedasticity (García, Molina, Lozano, & Herrera, 2009). Non-parametric tests are utilized instead of the parametric ones when those assumptions about the underlying populations are questionable (García et al., 2009).

It is trivial that the fitness values obtained by running the different HS algorithms, can not be assumed to come form a normal distribution. Therefore, to evaluate the statistical significance of the experimental comparisons, we have utilized *Mann–Whitney U Test*, one of the most powerful members of the class of non-parametric tests (McKnight & Najab, 2010). Instead of absolute fitness values, this test considers the rankings of the results obtained form multiple runs of two algorithms. *Mann–Whitney U Test* is a test of null hypothesis against the alternative hypothesis. The null hypothesis is that two samples come from the same population, i.e. both algorithms perform similarly. The alternative hypothesis is that one population tends to have larger values than the other. We have applied left-tailed hypothesis test, where the alternative hypothesis states that the median of SRHS is less than the median of competing method. The results of the conducted *Mann–Whitney U Test* are shown in Table 9. As it can be observed, fitness values of SRHS are totally lower than other methods for all test problems, except for f_{17} and f_{18}. For these two problems, SRHS has performed better than HS, IHS, SAHS and NGHS mwthods. Only, GBHS and GSHS methods have fitness values lower than SRHS over f_{17} and f_{18} problems. In the carried statistical test, a significance level of $\alpha = 0.01$ is considered and results of 51 runs were utilized.

According to the results obtained in this section, we observed that SRHS has almost outperformed other methods. It seems that the improvements made by SRHS originate form the different manner of this algorithm to consume fitness evaluations. As discussed in the end of Section 5.1, the number of iterations of SRHS is lower than other methods, but the number of total fitness evaluations is the same. In other words, the main difference between SRHS and other competing algorithms is that the SRHS algorithm consumes a part of fitness evaluations in the refinement step of the algorithm, instead of consuming it in the memory update step of the original HS algorithm. Thus, we can deduce that although all of the algorithms consume the same number of fitness evaluations, SRHS has improved the results because it consumes some of the evaluations in the refinement step. Thus, it can be reasonable not to dedicate all fitness evaluations to the memory update step, but consume some of them in the refinement step.

5.3. Computational load comparison

In this section, computational loads of all examined methods are being compared. Similar to analyses in Section 5.2, several runs are drawn and then corresponding statistical results are mentioned. All benchmark and state-of-the-art methods mentioned in previous section, are included in the computational load comparisons.

Computational loads of SRHS algorithm and other methods for separable problems is shown in Table 10. It can be observed that SRHS requires lowest amount of computations among all methods for f_1 problem. Its average, best and worst times are better others. Moreover, it provides lowest variations in the amount of required computations. For f_2 and f_3 problems, although SRHS is not the best method, it has consumed computations comparable to other

Table 10
Computational loads for category of separable functions (mili-seconds).

Problem	Run Time	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_1	mean	1.695445E4	2.625909E4	1.986749E4	1.851654E4	1.331773E4	6.481817E6	**4.326446E3**
	std	6.66962E3	2.631366E3	3.609925E3	4.894613E3	4.316254E3	9.083917E5	**9.444497E2**
	best	9.211E3	2.017E4	1.0647E4	1.051E4	7.711E3	4.795612E6	**2.768E3**
	worst	4.5644E4	3.4724E4	2.9422E4	3.1581E4	2.8018E4	7.714588E6	**6.785E3**
f_2	mean	2.900632E4	3.287093E4	1.971676E4	1.547598E4	**1.200638E4**	6.503569E6	1.522039E4
	std	3.170006E3	5.359571E3	3.578319E3	2.191656E3	**1.127173E3**	9.149771E5	1.395921E3
	best	2.1478E4	2.1634E4	1.3894E4	1.2671E4	**1.1021E4**	4.80241E6	1.2541E4
	worst	3.7855E4	4.492E4	3.551E4	2.2099E4	**1.6279E4**	7.785734E6	2.2859E4
f_3	mean	2.981002E4	3.214479E4	1.966455E4	1.556757E4	**1.220945E4**	6.503817E6	1.637784E4
	std	3.327701E3	4.917218E3	3.377855E3	2.044189E3	**9.578227E2**	9.161697E5	1.353635E3
	best	2.3046E4	2.2998E4	1.4817E4	1.3125E4	**1.1265E4**	4.804166E6	1.3562E4
	worst	3.8101E4	4.3394E4	3.1355E4	2.1667E4	**1.6071E4**	7.787978E6	2.0323E4

Table 11
Computational loads for category of nonseparable functions (mili-seconds).

Problem	Run Time	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_{19}	mean	1.746152E4	2.518688E4	1.853579E4	1.697804E4	1.113271E4	6.495366E6	**4.150364E3**
	std	6.821012E3	3.316894E3	3.672605E3	4.38541E3	2.880915E3	9.188058E5	**8.368369E2**
	best	9.671E3	1.662E4	1.0867E4	1.0763E4	6.961E3	4.804783E6	**2.645E3**
	worst	4.3378E4	3.2242E4	2.6609E4	2.6313E4	1.9205E4	7.793078E6	**5.962E3**
f_{20}	mean	1.853015E4	2.533702E4	1.828277E4	1.677111E4	1.122962E4	6.497555E6	**4.433927E3**
	std	6.230798E3	3.131556E3	4.047405E3	4.325759E3	2.584568E3	9.185541E5	**8.99281E2**
	best	1.0682E4	1.5727E4	1.1479E4	9.425E3	7.624E3	4.806054E6	**2.612E3**
	worst	3.8356E4	3.1456E4	2.6539E4	2.5323E4	1.9088E4	7.778064E6	**6.566E3**

Table 12
Computational loads for category of single-group m-nonseparable functions (mili-seconds).

Problem	Run Time	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_4	mean	1.970938E4	2.561366E4	1.825938E4	1.637771E4	1.174631E4	6.49975E6	**5.018804E3**
	std	7.822E3	3.549734E3	3.793919E3	4.154302E3	2.730834E3	9.117211E5	**7.824338E2**
	best	7.892E3	1.5376E4	1.1674E4	9.632E3	7.3E3	4.808057E6	**3.082E3**
	worst	5.8398E4	3.1433E4	2.4474E4	2.5461E4	1.9468E4	7.7546E6	**6.877E3**
f_5	mean	3.299252E4	3.302935E4	1.960964E4	1.560494E4	**1.276636E4**	6.49869E6	2.063807E4
	std	4.079172E3	4.841534E3	3.383169E3	1.79143E3	**7.331347E2**	9.160995E5	3.969074E3
	best	2.4501E4	2.2027E4	1.3937E4	1.3467E4	**1.1982E4**	4.797098E6	1.3861E4
	worst	4.4954E4	4.6426E4	3.6207E4	2.3252E4	**1.596E4**	7.762668E6	3.2205E4
f_6	mean	3.374573E4	3.003838E4	1.876962E4	1.524082E4	**1.31945E4**	6.499868E6	2.50653E4
	std	4.276494E3	4.491677E3	2.401391E3	1.48812E3	**8.829868E2**	9.144932E5	4.767595E3
	best	2.4713E4	2.2274E4	1.3536E4	1.3531E4	**1.2195E4**	4.79222E6	1.6932E4
	worst	4.4913E4	4.0778E4	2.5024E4	2.2087E4	**1.7675E4**	7.771919E6	3.8216E4
f_7	mean	1.72608E4	2.529168E4	1.854909E4	1.76933E4	1.301783E4	6.490703E6	**4.319796E3**
	std	5.807237E3	2.891334E3	3.675686E3	4.632592E3	3.490261E3	9.172909E5	**8.72041E2**
	best	8.918E3	1.6365E4	1.1385E4	1E4	8.263E3	4.804277E6	**2.43E3**
	worst	3.198E4	3.2444E4	2.4824E4	2.6941E4	2.0696E4	7.767132E6	**6.37E3**
f_8	mean	1.594467E4	2.494884E4	1.872371E4	1.713266E4	1.326785E4	6.494888E6	**4.356393E3**
	std	4.989688E3	2.820985E3	3.725423E3	4.400505E3	4.049112E3	9.078673E5	**9.380378E2**
	best	9.24E3	1.7977E4	1.1252E4	1.0087E4	8.283E3	4.784238E6	**2.803E3**
	worst	2.746E4	3.0692E4	2.8223E4	2.8349E4	2.5247E4	7.768039E6	**6.478E3**

Table 13
Computational loads for category of $\frac{d}{2m}$-group m-nonseparable functions (mili-seconds).

Problem	Run Time	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_9	mean	2.291154E4	2.773177E4	1.890536E4	1.627288E4	1.129011E4	6.502054E6	**7.984625E3**
	std	1.030774E4	4.474114E3	4.382859E3	4.247634E3	2.352336E3	9.127802E5	**1.155503E3**
	best	1.4694E4	1.6938E4	1.1766E4	1.0928E4	7.823E3	4.812782E6	**6.477E3**
	worst	7.1417E4	3.4999E4	2.7682E4	2.8939E4	1.9762E4	7.76098E6	**1.3168E4**
f_{10}	mean	3.727864E4	3.117143E4	1.90933E4	1.58486E4	**1.359215E4**	6.502763E6	2.173451E4
	std	4.692998E3	4.260463E3	2.260732E3	1.003027E3	**5.655488E2**	9.164339E5	1.943042E3
	best	2.562E4	2.2739E4	1.5414E4	1.4722E4	**1.3041E4**	4.790634E6	1.8165E4
	worst	4.9366E4	4.1134E4	2.4889E4	1.8683E4	**1.6284E4**	7.782103E6	2.7082E4
f_{11}	mean	3.819657E4	3.235352E4	1.979552E4	1.628824E4	**1.414178E4**	6.49781E6	2.336198E4
	std	4.343419E3	4.419765E3	2.253974E3	1.235766E3	**7.632709E2**	9.168793E5	2.377567E3
	best	2.9703E4	2.3377E4	1.6193E4	1.5038E4	**1.3485E4**	4.799868E6	1.934E4
	worst	5.0557E4	4.2479E4	2.6418E4	2.0432E4	**1.7334E4**	7.769655E6	2.8853E4
f_{12}	mean	1.593298E4	2.623815E4	1.961086E4	1.906568E4	1.335985E4	6.479292E6	**4.048536E3**
	std	5.685888E3	2.37135E3	3.893976E3	5.118356E3	4.376972E3	9.068452E5	**8.611517E2**
	best	8.65E3	2.0837E4	1.0468E4	1.1357E4	7.729E3	4.792221E6	**2.715E3**
	worst	3.087E4	3.1525E4	2.7846E4	2.9254E4	2.2741E4	7.733156E6	**6.625E3**
f_{13}	mean	1.751255E4	2.610832E4	1.919309E4	1.779827E4	1.292909E4	6.486806E6	**4.317321E3**
	std	5.538975E3	2.465699E3	3.639095E3	4.454552E3	3.723857E3	9.061081E5	**9.512759E2**
	best	9.442E3	1.938E4	1.0365E4	9.573E3	7.562E3	4.811912E6	**2.449E3**
	worst	3.2539E4	3.1569E4	2.5141E4	2.6517E4	2.0593E4	7.737588E6	**6.544E3**

methods. In other words, the order of computational load required by SRHS is the same as best methods for these problems.

The computational loads for nonseparable functions are depicted in Table 11. AS discussed in Section 5.2, SRHS consistently provided best fitness values for nonseparable functions that are most challenging optimization problems. Furthermore, SRHS presents consistent superiority with respect to required computational for such a difficult problems.

Statistics of computational costs for the category of m-nonseparable functions are depicted in Table 12. It can be observed that SRHS is the best method for f_4, f_7 and f_8 problems with respect to average, best and worst cases of required computations. Again, it has provided lowest fluctuations. It is trivial to

Table 14
Computational loads for category of $\frac{d}{m}$-group m-nonseparable functions (mili-seconds).

Problem	Run Time	Method						
		HS	IHS	GBHS	SAHS	NGHS	GSHS	SRHS
f_{14}	mean	2.225358E4	3.028752E4	1.954404E4	1.600764E4	**1.118938E4**	6.506324E6	1.15366E4
	std	2.528511E3	5.92198E3	4.445116E3	3.166552E3	1.559215E3	9.149901E5	**7.344268E2**
	best	1.5615E4	1.8375E4	1.2504E4	1.1795E4	**9.207E3**	4.807624E6	9.89E3
	worst	2.8929E4	4.2293E4	3.3387E4	2.7021E4	1.7279E4	7.761582E6	**1.3924E4**
f_{15}	mean	4.32418E4	3.3142E4	1.987093E4	1.70134E4	**1.490936E4**	6.493286E6	2.5848E4
	std	5.298815E3	4.089029E3	1.904873E3	9.760518E2	**5.863115E2**	9.142487E5	3.146202E3
	best	3.3612E4	2.4427E4	1.5941E4	1.6137E4	**1.4344E4**	4.799193E6	1.9942E4
	worst	5.6757E4	4.6121E4	2.3877E4	2.0663E4	**1.7462E4**	7.76243E6	3.542E4
f_{16}	mean	4.428309E4	3.254073E4	2.028145E4	1.74552E4	**1.562564E4**	6.491103E6	2.953705E4
	std	5.483624E3	4.509074E3	2.506614E3	9.608317E2	**8.152033E2**	9.156324E5	3.235904E3
	best	3.0614E4	2.5257E4	1.6336E4	1.6648E4	**1.4766E4**	4.798841E6	2.4454E4
	worst	6.069E4	4.6204E4	2.8493E4	2.202E4	**1.8647E4**	7.752653E6	3.8371E4
f_{17}	mean	1.639202E4	2.543442E4	1.890123E4	1.795316E4	1.235646E4	6.489287E6	**3.948618E3**
	std	7.671491E3	2.727992E3	3.604134E3	4.631779E3	3.412466E3	9.101542E5	**7.91349E2**
	best	8.472E3	1.8324E4	1.1395E4	1.1074E4	7.453E3	4.769783E6	**2.514E3**
	worst	5.7649E4	3.2939E4	2.5746E4	2.6718E4	1.9507E4	7.784118E6	**6.046E3**
f_{18}	mean	1.836179E4	2.569289E4	1.876738E4	1.740393E4	1.206554E4	6.492538E6	**4.247815E3**
	std	7.290767E3	2.962118E3	3.998128E3	4.392171E3	3.19059E3	9.150563E5	**8.768872E2**
	best	9.944E3	1.8037E4	1.1122E4	1.0193E4	7.442E3	4.79819E6	**2.45E3**
	worst	5.0132E4	3.423E4	3.0638E4	2.5767E4	2.162E4	7.78604E6	**6.225E3**

see that SRHS has proved its superiority with a considerable gap with others for these problems. For f_5 and f_6 problems, although SRHS is not the best, its performance is comparable to NGHS that consumes lowest computations.

Category of $\frac{d}{2m}$-nonseparable test problems was analyzed with the settings similar to Section 5.2, i.e. $d = \text{NVAR}$ and $m = 50$ being the number of variables in each group which is in fact the degree of separability. As shown in Table 13, SRHS is again the superior method for f_9, f_{12} and f_{13} problems. For other two problems, it is comparable to NGHS because computations for both of them are within the same order.

The last category of test problems for analysis is $\frac{d}{m}$-group m-nonseparable (Table 14). SRHS has demonstrated better computational cost over other methods for f_{17} and f_{18} problems. Although NGHS is the best method for f_{15} and f_{16} problems, SRHS is still comparable to it. For f_{14} problem average cost required by NGHS is better and its worst case computation is lower. However, standard deviation and best case performance of SRHS are more appropriate.

Based on experimental results depicted in Tables 10–14, SRHS requires lower computations in average for many problems of *IEEE CEC 2010* suite. Its best case and worst cases along with standard deviations were better than other for those problems. However, there exist some problems that SRHS is not the best among others. In such situation, experimental results proved that its required computation is comparable to other methods such that they require same order of computations.

5.4. Sensitivity analysis

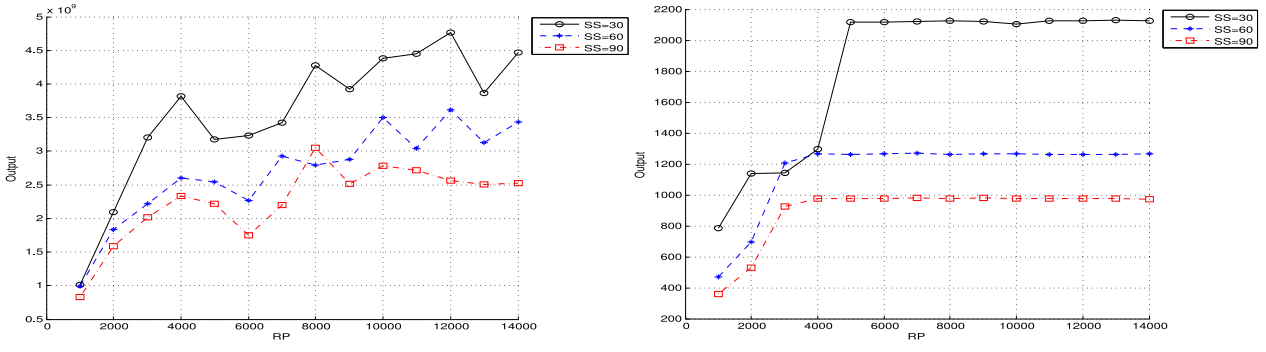
When a heuristic is applied onto the evolutionary algorithms, it is difficult to decide on control parameters of the method. Sensitivity analysis on any EA is the study of how the uncertainty in its output can be apportioned to different parameters of the algorithm. In fact, a generic sensitivity analysis measures the interdependencies of the parameters from the EA method and their influence on its outputs (Hernández-Riveros & Villada-Cano, 2012). In this section, we have demonstrated sensitivity of SRHS algorithm to its parameters.

As explained in Section 5.2, SRHS has made considerable improvements to the original HS algorithm. It has also outperformed state-of-the-art methods for most of the problems in *IEEE CEC 2010* test suite. The improvements made by SRHS algorithm come from its proposed *population refinement* step that was integrated into the original HS algorithm. The *population refinement* step introduced two new parameters: RP and SS. Now, the main question is how each parameter influences the output of SRHS algorithm? As explained in Section 4, SRHS picked the best harmony as the representative of existing knowledge and experience accumulated in HM. Best harmony is utilized to refine current harmonies and improve their quality. In fact, decision variables of the best harmony are exploited to refine other harmonies such that their unsuitable decision variables are pruned. In other words, *population refinement* is an effort to compensate unpleasant decision variables of the HM members. In order to study the sensitivity of SRHS algortihm, we proceed as follows:

- Total number of iterations are fixed to 15,000.
- Every experiment is repeated for 51 runs and the outputs are averaged.
- The range of values for RP parameter is {1000, 2000, ⋯, 14, 000}.
- The range of values for SS parameter is {10, 20, ⋯, 100}.
- When analyzing the RP parameter, SS parameter will be fixed.
- When analyzing the SS parameter, RP parameter will be fixed.

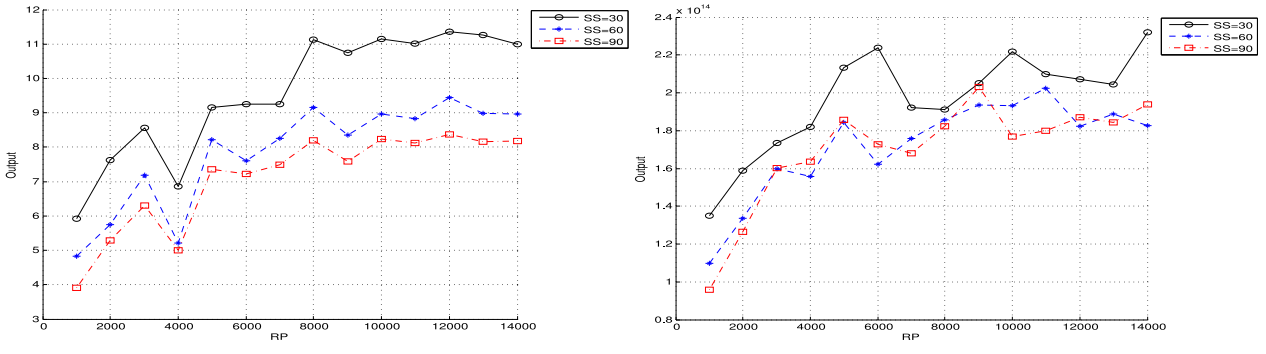
5.4.1. Effect of RP parameter on SRHS algorithm

First, the effect of RP parameter is analyzed. As discussed above, RP determines the iterations in which the refinement of SRHS occurs. As the RP parameter is increased, the distance between refinement iterations gets longer and the total number of refinement actions is reduced. Reduction in the number of refinement operations makes the SRHS algorithm to consume less fitness evaluations as indicated by Eq. (11). This equation similarly yields that higher values of RP make reduction in total number of fitness evaluations and makes SRHS to produce less desired fitness values. In order to examine whether the experimental results confirm this hypothesis about the effect of RP parameter on SRHS, the RP is gradually increased and the output fitnesses are recorded and shown in Fig. 1. As it can be observed, the higher the values of RP, the lower the quality of outputs for all 20 test problems. The analysis for each test problems includes three curves corresponding to three values assigned to SS parameter, i.e. {30, 60, 90}. As shown in Fig. 1, the suitability of outputs



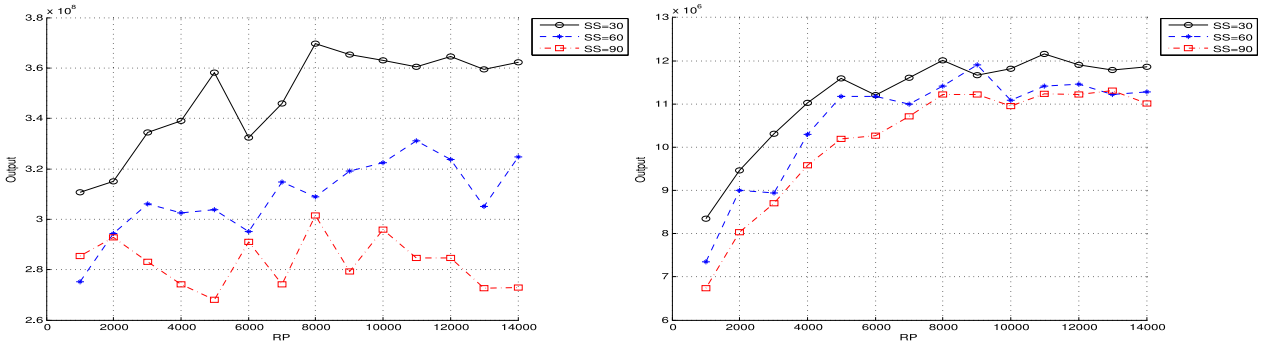
(a) f_1

(b) f_2



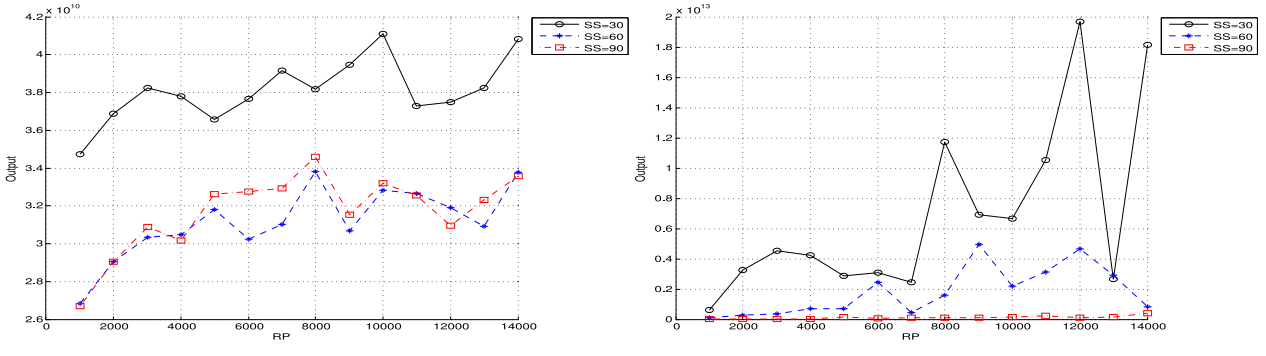
(c) f_3

(d) f_4



(e) f_5

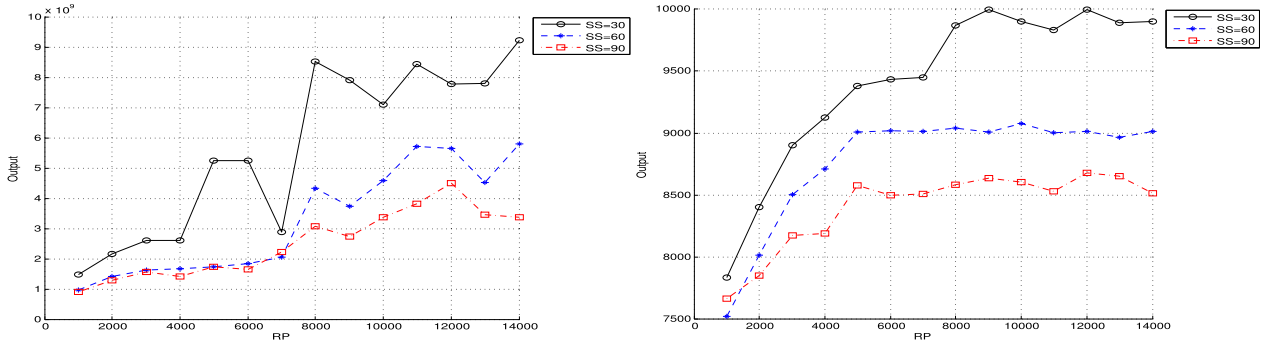
(f) f_6



(g) f_7

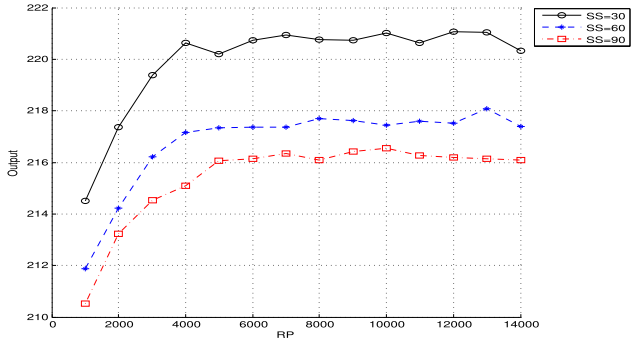
(h) f_8

Fig. 1. Sensitivity of SRHS algorithm to RP parameter.

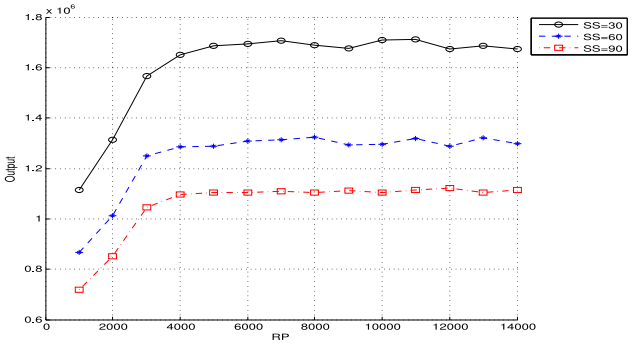


(a) f_9

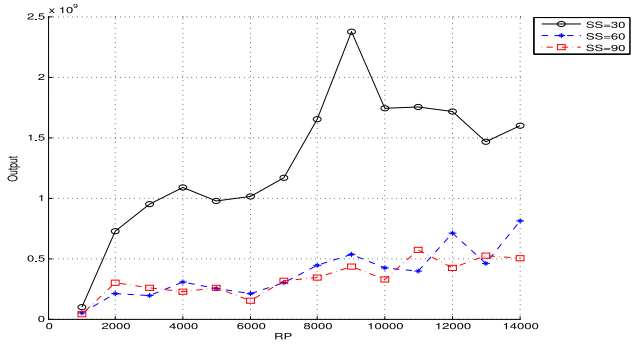
(b) f_{10}



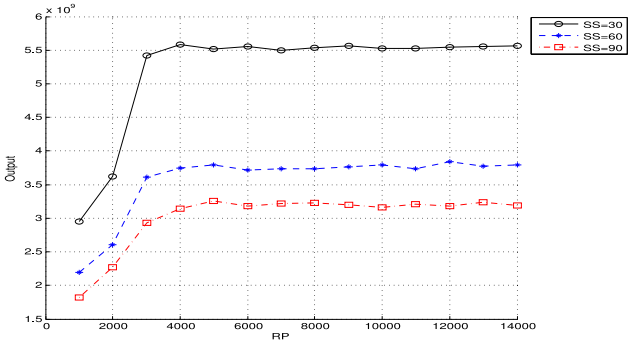
(c) f_{11}



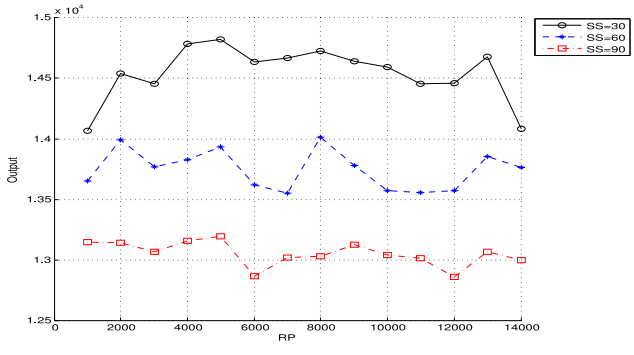
(d) f_{12}



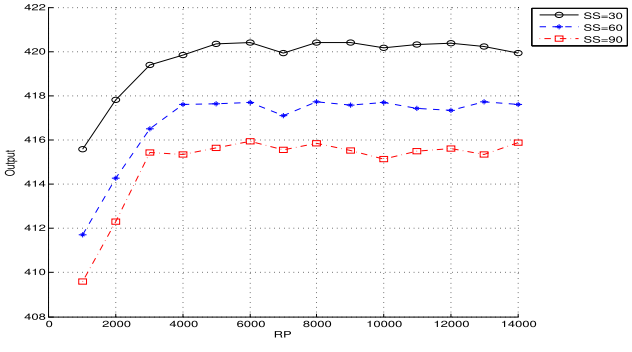
(e) f_{13}



(f) f_{14}



(g) f_{15}



(h) f_{16}

Fig. 2. Sensitivity of SRHS algorithm to RP parameter (cont.).

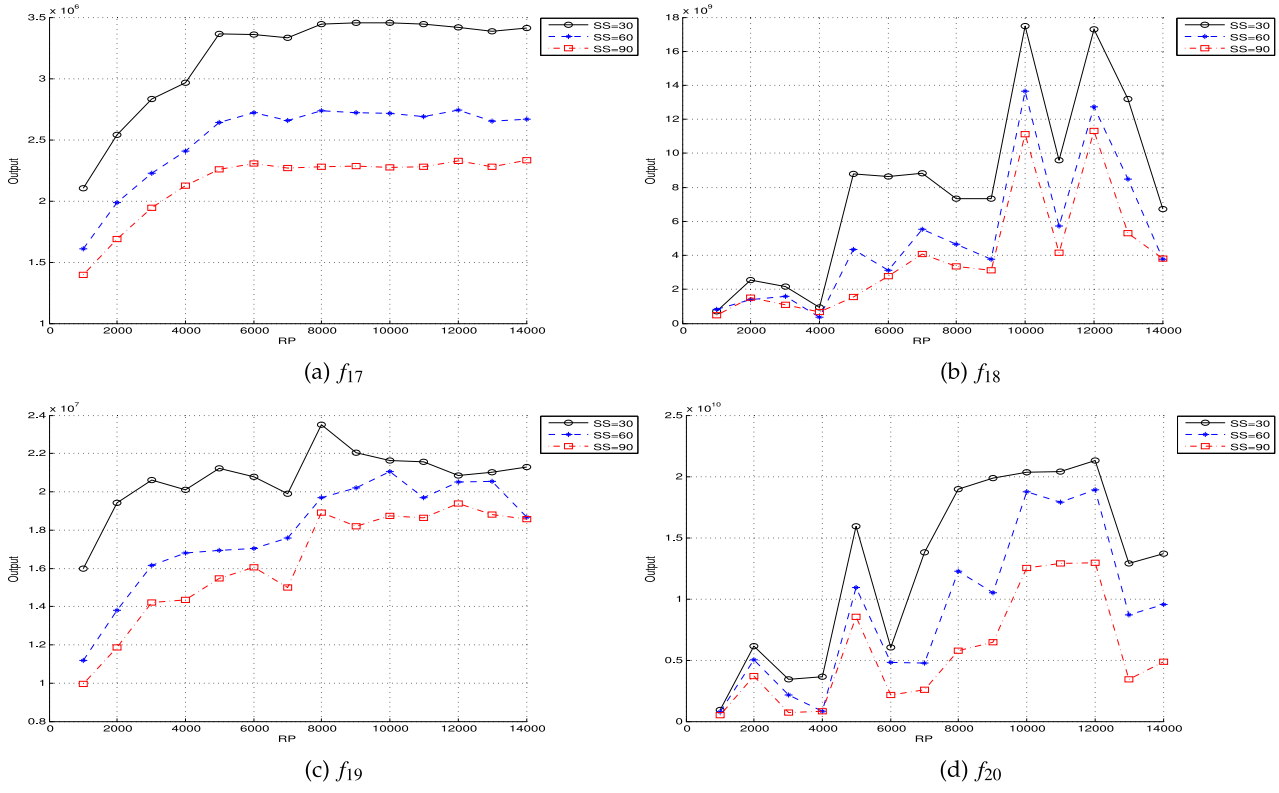


Fig. 3. Sensitivity of SRHS algorithm to RP parameter (cont.).

of SRHS reduces monotonically as RP increases (Note that lower output values correspond to higher fitnesses) (Figs. 2 and 3).

5.4.2. Effect of SS parameter on SRHS algorithm

Next, we analyze the effect of SS parameter. As discussed in Section 4, SS parameter enables SRHS algorithm to select some decision variables of X^{best} to be consumed in the refinement process. The relation between the value assigned to SS parameter and complexity of SRHS algorithm was depicted by Eq. (11). This equation states that higher values of SS increases the total number of fitness evaluation consumed by SRHS algorithm. This is logically reasonable because SS determines the number of decision variables that are selected form X^{best} and compared to variables of HM members in the refinement step. Higher the value of SS, more decision variables of X^{best} are employed in the comparisons made in refinement. Larger number of comparisons yields more fitness evaluations consumed by SRHS algorithm. With additional consumed fitness evaluations, SRHS algorithm is obviously expected to produce better fitness values. To verify this hypothesis about effects of SS, the outputs generated by SRHS algorithm are recorded for a range of values of SS parameter and the resulting curves are shown in Fig. 4. It can be easily observed that the results (with some fluctuations) verify the expected contribution of SS parameter in SRHS algorithm (Figs. 5 and 6).

5.5. Discussion

SRHS is a variant of harmony search algorithm that integrates selection and refinement steps into the memory update procedure of the original work. Selection plays a main role in evolution processes affected by fitness values, such that it provides population diversity and reduces the chance of premature convergence. Selection-based HS algorithms are trying to achieve this purpose. Selection procedure of SRHS was inspired by such an objective. However, the selection step itself might not be able to make a large amount of improvements. This can be justified by the results obtained from GSHS algorithm. One of the main problem with all previous HS algorithms, both selective and non-selective, is the fact that all of them are making decisions just based on fitness values. In fact, a harmony is assessed based on its fitness, in a way that it will be ignored if the fitness values are low. However, the behavior of SRHS algorithm is more similar to musicians because decisions are made based on each decision variable of the harmony. These modifications allowed the SRHS algorithm to behave more naturally and reduce the premature convergence. As depicted by experimental analyses in Section 5.2, with equal number of fitness evaluations, SRHS algorithm has been consistently superior to other methods for almost all problems of *IEEE CEC 2010* test suite. Despite to improvements in fitting power, computational cost of SRHS algorithm is in the same order of best compared method. For some test problems SRHS and for some others NGHS required lowest computations.

6. Conclusion and future work

This paper presented *Selective Refining Harmony Search* algorithm which incorporates the *population refinement* step into the original HS algorithm. This step is aimed to refine all harmonies based on the current knowledge and experience that exist in the HM. Best harmony memory was considered as an appropriate choice to represent such knowledge and experience and its decision variables were employed to improve decision variables of other harmonies. Refinement step does not occur at every iteration in order to avoid much computational complexity and fast convergence to local minimums. Furthermore, *population refinement* was controlled by two parameters RP and SS. SRHS was compared to 6 state-of-the-art HS variants for 20 test problems of *IEEE CEC 2010* suit. The obtained results explained that SRHS method outperforms almost all other competitors with statistically significant

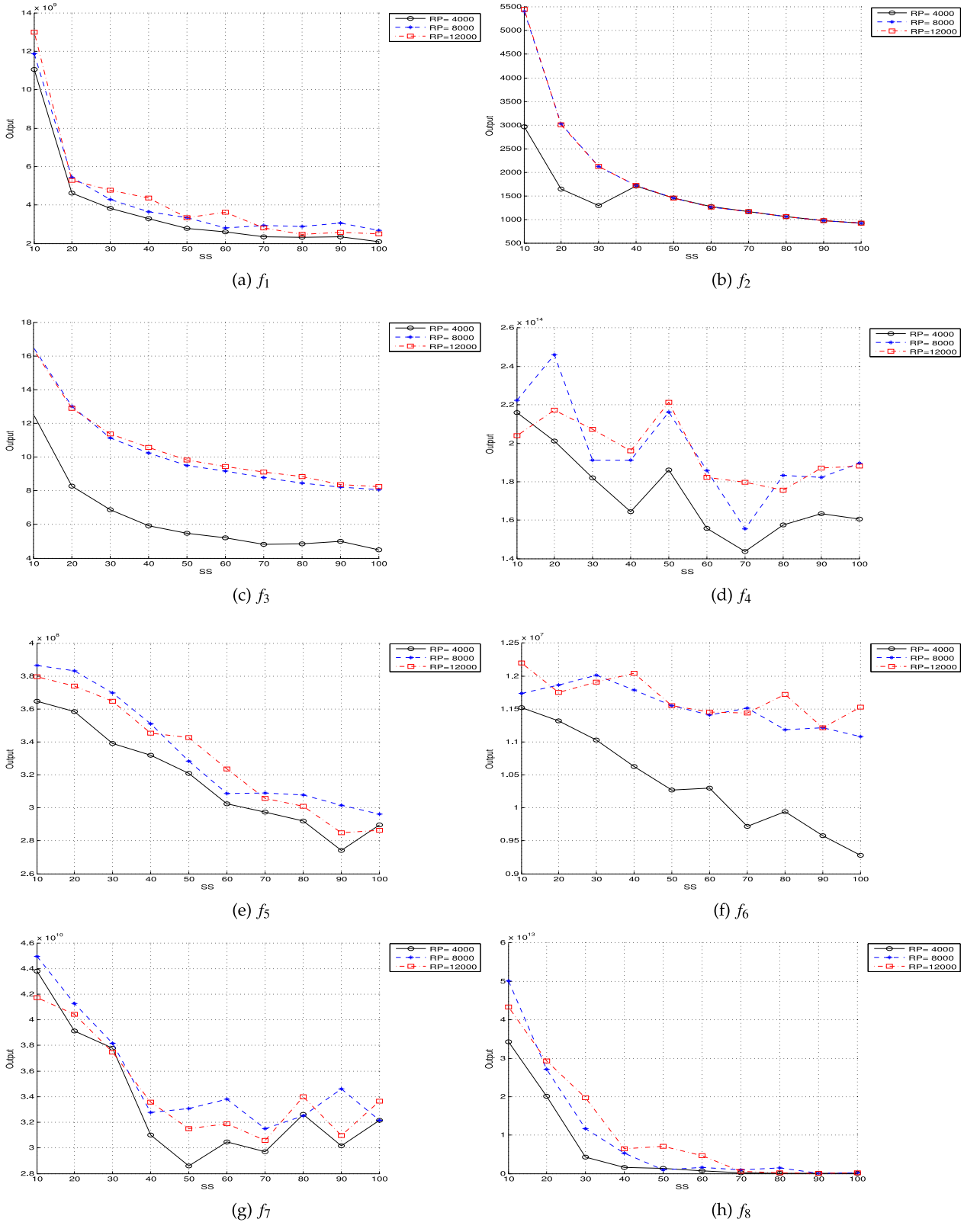
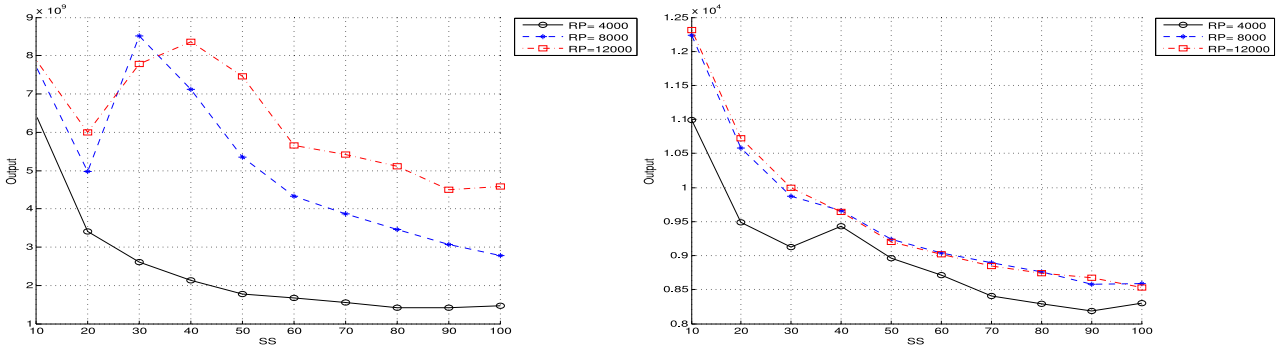
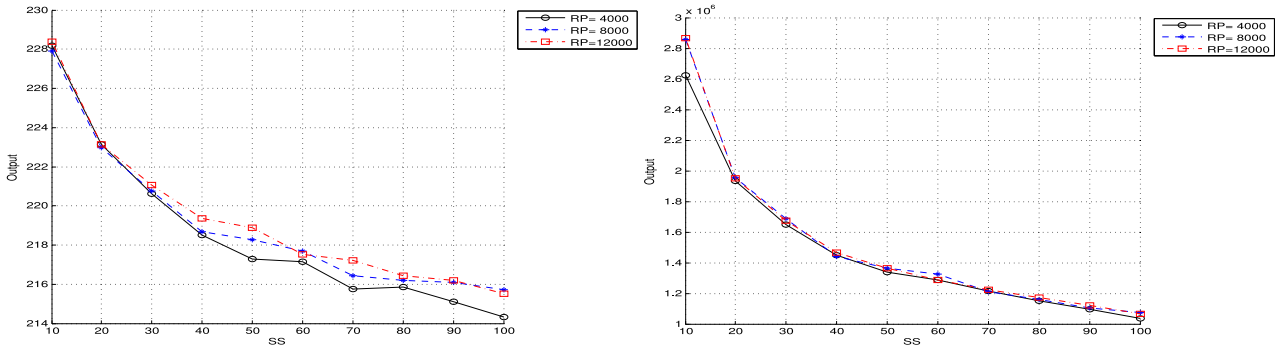


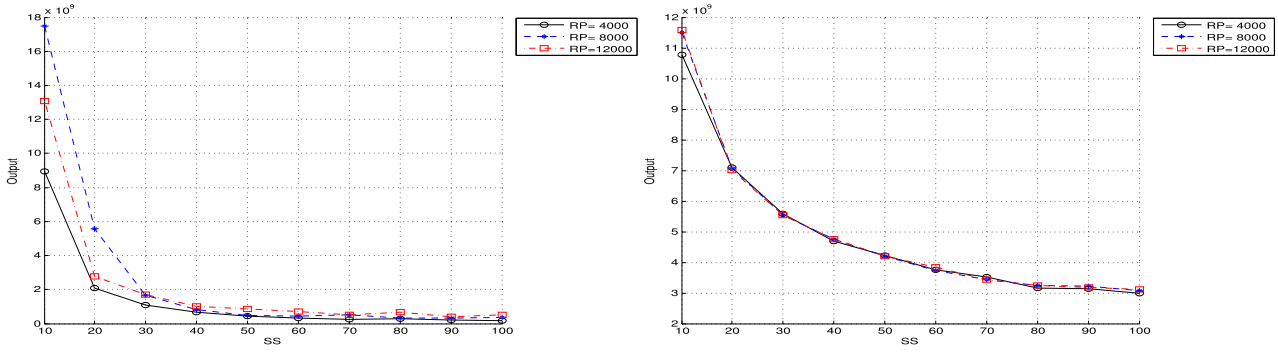
Fig. 4. Sensitivity of SRHS algorithm to SS parameter.



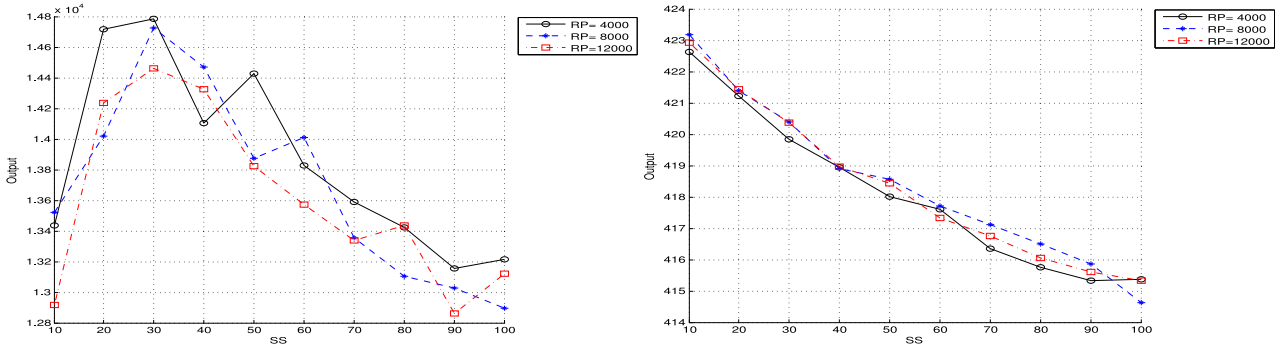
(a) f_9 (b) f_{10}



(c) f_{11} (d) f_{12}



(e) f_{13} (f) f_{14}



(g) f_{15} (h) f_{16}

Fig. 5. Sensitivity of SRHS algorithm to SS parameter (cont.).

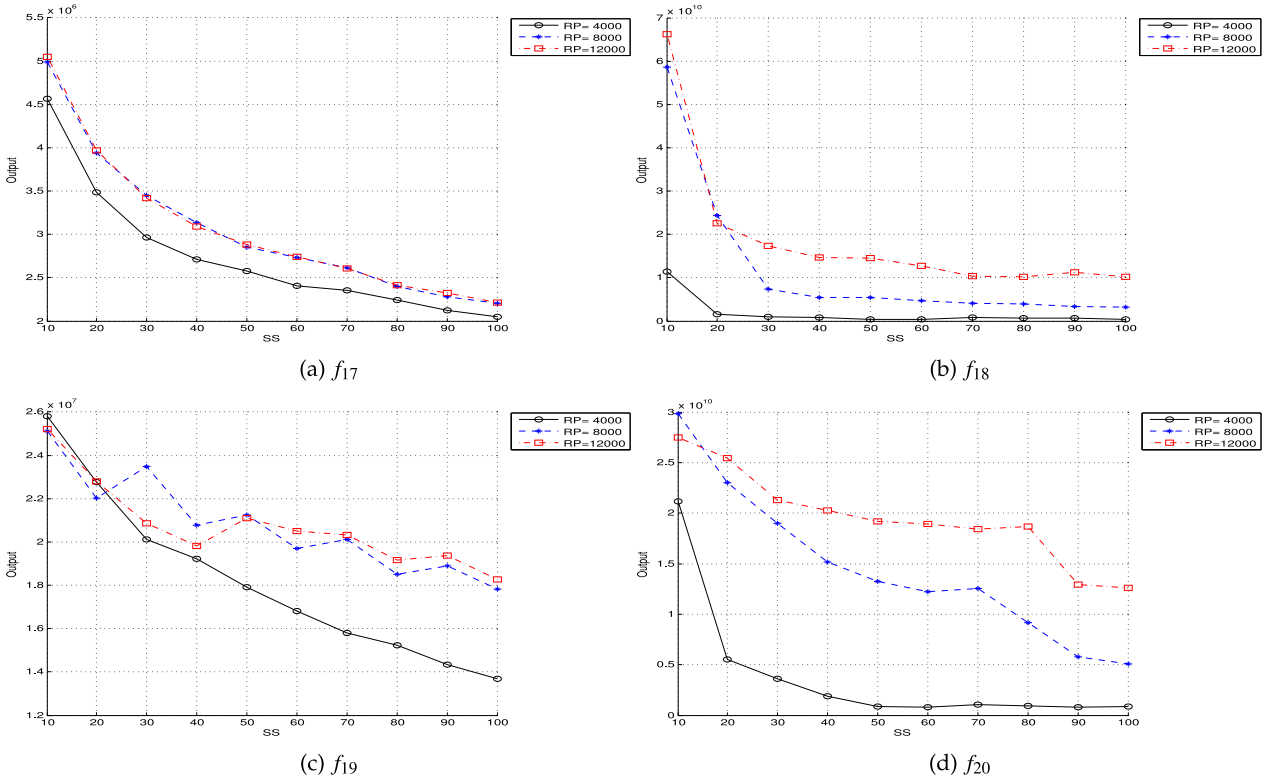


Fig. 6. Sensitivity of SRHS algorithm to SS parameter (cont.).

differences for most of the test problems. To put in a nutshell, experimental results and theoretical analyses proved the advantages and drawbacks of SRHS algorithm to be as follows: (1) With equal number of fitness evaluations, fitting power of SRHS algorithm has been consistently superior to other methods for almost all problems of *IEEE CEC 2010* test suite; (2) Despite to improvements in fitting power, computational cost of SRHS algorithm is in the same order of best compared method. For some test problems SRHS and for some others NGHS required lowest computations; (3) SRHS has integrated selection and refinement steps into the memory update step. This modification allowed the SRHS algorithm to reduce the chance of premature convergence problem; (4) Refinement step of SRHS algorithm extended the selection process from harmony level to decision variable level. In other words, all harmony search algorithms just decide based on the fitness value of the underlying harmony such that those with low fitness are thrown away. However, SRHS has a more natural behavior since it does not ignore all the decision variables of a weak harmony. In future, we will investigate the essence of refinement concept and try to figure out why it helps. As mentioned in the paper, assigning some fitness evaluations to refinement step, instead of memory update step, allowed the SRHS algorithm produce promising results. Another possible future direction might be to study the extent to which that refinement can occur and its effects on performance and convergence of the algorithm.

References

Akkoyunlu, M. C., Engin, O., & Buyukozkan, K. (2015). A harmony search algorithm for hybrid flow shop scheduling with multiprocessor task problems. In *Modeling, simulation, and applied optimization (ICMSAO), 2015 6th international conference on* (pp. 1–3). IEEE.

Al-Betar, M. A., Awadallah, M. A., Khader, A. T., & Abdalkareem, Z. A. (2015). Island-based harmony search for optimization problems. *Expert Systems with Applications, 42*(4), 2026–2035.

Al-Betar, M. A., Doush, I. A., Khader, A. T., & Awadallah, M. A. (2012). Novel selection schemes for harmony search. *Applied Mathematics and Computation, 218*(10), 6095–6117.

Al-Betar, M. A., Khader, A. T., Geem, Z. W., Doush, I. A., & Awadallah, M. A. (2013). An analysis of selection methods in memory consideration for harmony search. *Applied Mathematics and Computation, 219*(22), 10753–10767.

Auger, A., Hansen, N., Mauny, N., Ros, R., & Schoenauer, M. (2007). *Bio-inspired continuous optimization: The coming of age*. Piscataway, NJ, USA.

Bäck, T. (1994). Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Evolutionary computation, 1994. IEEE world congress on computational intelligence., proceedings of the first IEEE conference on* (pp. 57–62). IEEE.

Bäck, T., & Hoffmeister, F. (1991). *Extended selection mechanisms in genetic algorithms*.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. In *Proceedings of an international conference on genetic algorithms and their applications* (pp. 101–111). Hillsdale, New Jersey.

Bellman, R., Bellman, R. E., Bellman, R. E., & Bellman, R. E. (1961). *Adaptive control processes: A guided tour*: 4. Princeton university press Princeton.

Bhattacharyya, M., & Bandyopadhyay, S. (2015). Finding quasi core with simulated stacked neural networks. *Information Sciences, 294*, 1–14.

Bianchi, L., Dorigo, M., Gambardella, L. M., & Gutjahr, W. J. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: An International Journal, 8*(2), 239–287.

Bueno, P. M., Jino, M., & Wong, W. E. (2014). Diversity oriented test data generation using metaheuristic search techniques. *Information Sciences, 259*, 490–509.

Cao, F., & Wang, W. (2011). Harmony search based particle swarm optimisation approach for optimal pid control in electroslag remelting process. *International Journal of Modelling, Identification and Control, 15*(1), 20–27.

Castelli, M., Silva, S., Manzoni, L., & Vanneschi, L. (2014). Geometric selective harmony search. *Information Sciences, 279*, 468–482.

Clerc, M. (2010). *Particle swarm optimization*: 93. John Wiley & Sons.

Črepinšek, M., Liu, S.-H., Mernik, L., & Mernik, M. (2015). Is a comparison of results meaningful from the inexact replications of computational experiments? *Soft Computing*, 1–13.

Črepinšek, M., Liu, S.-H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR), 45*(3), 35.

Črepinšek, M., Liu, S.-H., & Mernik, M. (2014). Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them. *Applied Soft Computing, 19*, 161–170.

Darwin, C. (1859). *On the origins of species by means of natural selection* p. 247. London: Murray.

Das, S., Mukhopadhyay, A., Roy, A., Abraham, A., & Panigrahi, B. K. (2011). Exploratory power of the harmony search algorithm: Analysis and improvements for global numerical optimization. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 41*(1), 89–106.

Davis, L. (1991). *Handbook of genetic algorithms*.

Derrac, J., García, S., Molina, D., & Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation, 1*(1), 3–18.

Devika, K., Jafarian, A., Hassanzadeh, A., & Khodaverdi, R. (2014). Optimizing of bullwhip effect and net stock amplification in three-echelon supply chains using evolutionary multi-objective metaheuristics. *Annals of Operations Research*, 1–31.

Dorigo, M., & Birattari, M. (2010). Ant colony optimization. In *Encyclopedia of machine learning* (pp. 36–39). Springer.

Eberhart, R. C., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science: 1* (pp. 39–43). New York, NY.

Fesanghary, M. (2009). Harmony search applications in mechanical, chemical and electrical engineering. In *Music-inspired harmony search algorithm* (pp. 71–86). Springer.

Floudas, C. A., & Pardalos, P. M. (2008). *Encyclopedia of optimization*: 1. Springer Science & Business Media.

Forsati, R., & Mahdavi, M. (2010). Web text mining using harmony search. In *Recent advances in harmony search algorithm* (pp. 51–64). Springer.

Forsati, R., & Shamsfard, M. (2015). Novel harmony search-based algorithms for part-of-speech tagging. *Knowledge and Information Systems, 42*(3), 709–736.

Gandomi, A. H., Yang, X.-S., & Alavi, A. H. (2013). Cuckoo search algorithm: A metaheuristic approach to solve structural optimization problems. *Engineering with Computers, 29*(1), 17–35.

Garcia, M. P., Montiel, O., Castillo, O., Sepúlveda, R., & Melin, P. (2009). Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. *Applied Soft Computing, 9*(3), 1102–1110.

García, S., Molina, D., Lozano, M., & Herrera, F. (2009). A study on the use of non–parametric tests for analyzing the evolutionary algorithms behaviour: A case study on the cec'2005 special session on real parameter optimization. *Journal of Heuristics, 15*(6), 617–644.

Geem, Z. W. (2008). Novel derivative of harmony search algorithm for discrete design variables. *Applied Mathematics and Computation, 199*(1), 223–230.

Geem, Z. W., Kim, J. H., & Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation, 76*(2), 60–68.

Gendreau, M., & Potvin, J.-Y. (2010). *Handbook of metaheuristics*: 2. Springer.

Goldberg, D. E., Korb, B., & Deb, K. (1989). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems, 3*(5), 493–530.

Griewank, A., & Toint, P. L. (1982). Partitioned variable metric updates for large structured optimization problems. *Numerische Mathematik, 39*(1), 119–137.

Guo, P., Wang, J., Gao, X. Z., & Tanskanen, J. (2012). Epileptic eeg signal classification with marching pursuit based on harmony search method. In *Systems, man, and cybernetics (SMC), 2012 IEEE international conference on* (pp. 283–288). IEEE.

Hadwan, M., Ayob, M., Sabar, N. R., & Qu, R. (2013). A harmony search algorithm for nurse rostering problems. *Information Sciences, 233*, 126–140.

Hancock, P. J. (1994). An empirical comparison of selection methods in evolutionary algorithms. In *Evolutionary computing* (pp. 80–94). Springer.

Hernández-Riveros, J.-A., & Villada-Cano, D. (2012). Sensitivity analysis of an autonomous evolutionary algorithm. In *Advances in artificial intelligence–IBERAMIA 2012* (pp. 271–280). Springer.

Ho, J.-H., Shih, H.-C., Liao, B.-Y., & Chu, S.-C. (2012). A ladder diffusion algorithm using ant colony optimization for wireless sensor networks. *Information Sciences, 192*, 204–212.

Holland, J. H. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Karimi, M., Askarzadeh, A., & Rezazadeh, A. (2012). Using tournament selection approach to improve harmony search algorithm for modeling of proton exchange membrane fuel cell. *International Journal of Electrochemical Science, 7*, 6426–6435.

Kennedy, J. (2010). Particle swarm optimization. In *Encyclopedia of machine learning* (pp. 760–766). Springer.

Khooban, M. H., Abadi, D. N. M., Alfi, A., & Siahi, M. (2013). Swarm optimization tuned Mamdani fuzzy controller for diabetes delayed model. *Turkish Journal of Electrical Engineering & Computer Sciences, 21*(Sup 1), 2110–2126.

Kushida, J.-i., Hara, A., Takahama, T., & Kido, A. (2013). Island-based differential evolution with varying subpopulation size. In *Computational intelligence & applications (IWCIA), 2013 IEEE sixth international workshop on* (pp. 119–124). IEEE.

Landa-Torres, I., Manjarres, D., Salcedo-Sanz, S., Del Ser, J., & Gil-Lopez, S. (2013). A multi-objective grouping harmony search algorithm for the optimal distribution of 24-hour medical emergency units. *Expert Systems with Applications, 40*(6), 2343–2349.

Lee, K. S., & Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering, 194*(36), 3902–3933.

Leung, S. C., Zhang, D., Zhou, C., & Wu, T. (2012). A hybrid simulated annealing metaheuristic algorithm for the two-dimensional knapsack packing problem. *Computers & Operations Research, 39*(1), 64–73.

Liu, L., & Zhou, H. (2013). Hybridization of harmony search with variable neighborhood search for restrictive single-machine earliness/tardiness problem. *Information Sciences, 226*, 68–92.

Mahdavi, M., Fesanghary, M., & Damangir, E. (2007). An improved harmony search algorithm for solving optimization problems. *Applied Mathematics and Computation, 188*(2), 1567–1579.

McKnight, P. E., & Najab, J. (2010). Mann–Whitney u test. *Corsini Encyclopedia of Psychology*.

Mernik, M., Liu, S.-H., Karaboga, D., & Črepinšek, M. (2015). On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. *Information Sciences, 291*, 115–127.

Miller, B. L., & Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems, 9*(3), 193–212.

Miller, B. L., & Goldberg, D. E. (1996). Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation, 4*(2), 113–131.

Minetti, G., Leguizamón, G., & Alba, E. (2014). An improved trajectory-based hybrid metaheuristic applied to the noisy dna fragment assembly problem. *Information Sciences, 277*, 273–283.

Moh'd Alia, O., Mandava, R., & Aziz, M. E. (2011). A hybrid harmony search algorithm for mri brain segmentation. *Evolutionary Intelligence, 4*(1), 31–49.

Omran, M. G., & Mahdavi, M. (2008). Global-best harmony search. *Applied Mathematics and Computation, 198*(2), 643–656.

Osman, I. H., & Kelly, J. P. (2012). *Meta-heuristics: Theory and applications*. Springer Science & Business Media.

Ouyang, H.-b., Gao, L.-q., Li, S., & Kong, X.-y. (2015). Improved novel global harmony search with a new relaxation method for reliability optimization problems. *Information Sciences, 305*, 14–55.

Pan, Q.-K., Suganthan, P. N., Tasgetiren, M. F., & Liang, J. J. (2010). A self-adaptive global best harmony search algorithm for continuous optimization problems. *Applied Mathematics and Computation, 216*(3), 830–848.

Rodriguez, F. J., Garcia-Martinez, C., & Lozano, M. (2012). Hybrid metaheuristics based on evolutionary algorithms and simulated annealing: Taxonomy, comparison, and synergy test. *Evolutionary Computation, IEEE Transactions on, 16*(6), 787–800.

dos Santos Coelho, L., & de Andrade Bernert, D. L. (2009). An improved harmony search algorithm for synchronization of discrete-time chaotic systems. *Chaos, Solitons & Fractals, 41*(5), 2526–2532.

Shokri-Ghaleh, H., & Alfi, A. (2014a). A comparison between optimization algorithms applied to synchronization of bilateral teleoperation systems against time delay and modeling uncertainties. *Applied Soft Computing, 24*, 447–456.

Shokri-Ghaleh, H., & Alfi, A. (2014b). Optimal synchronization of teleoperation systems via cuckoo optimization algorithm. *Nonlinear Dynamics, 78*(4), 2359–2376.

Shreem, S. S., Abdullah, S., & Nazri, M. Z. A. (2014). Hybridising harmony search with a Markov blanket for gene selection problems. *Information Sciences, 258*, 108–121.

Spencer, H. (1896). *The principles of biology*: 1. D. Appleton.

Talbi, E.-G. (2009). *Metaheuristics: From design to implementation*: 74. John Wiley & Sons.

Tang, K., Xiaodong Li, P. S., & Zhenyu Yang, T. W. (2010). Benchmark functions for the CEC 2010 special session and competition on large scale global optimization. *Technical Report*. University of Science and Technology of China (USTC).

Turky, A. M., & Abdullah, S. (2014). A multi-population harmony search algorithm with external archive for dynamic optimization problems. *Information Sciences, 272*, 84–95.

Wang, C.-M., & Huang, Y.-F. (2010). Self-adaptive harmony search algorithm for optimization. *Expert Systems with Applications, 37*(4), 2826–2837.

Yang, X.-S. (2010). *Nature-inspired metaheuristic algorithms*. Luniver press.

Yuan, Y., Xu, H., & Yang, J. (2013). A hybrid harmony search algorithm for the flexible job shop scheduling problem. *Applied Soft Computing, 13*(7), 3259–3272.

Zimmer, C., & Emlen, D. J. (2013). *Evolution: Making sense of life*. Roberts Greenwood Village.

Zou, D., Gao, L., Wu, J., Li, S., & Li, Y. (2010). A novel global harmony search algorithm for reliability problems. *Computers & Industrial Engineering, 58*(2), 307–316.