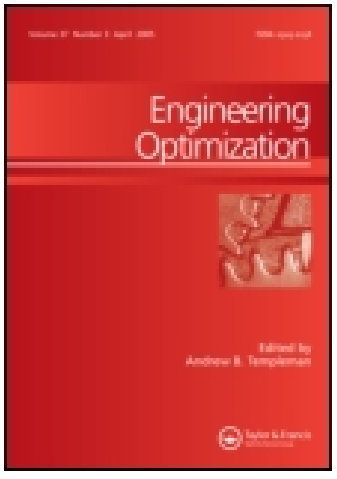

Engineering Optimization
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/geno20

A local-best harmony search algorithm with dynamic subpopulations

Quan-Ke Pan [a], P. N. Suganthan [b], J. J. Liang [c] & M. Fatih Tasgetiren [d]

[a] College of Computer Science, Liaocheng University, Liaocheng, P.R. China

[b] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore

[c] School of Electrical Engineering, Zhengzhou University, Zhengzhou, P.R. China

[d] Department of Industrial Engineering, Yasar University, Bornova, Izmir, Turkey

Published online: 29 Oct 2009.

To cite this article: Quan-Ke Pan, P. N. Suganthan, J. J. Liang & M. Fatih Tasgetiren (2010) A local-best harmony search algorithm with dynamic subpopulations, Engineering Optimization, 42:2, 101-117, DOI: 10.1080/03052150903104366

To link to this article: http://dx.doi.org/10.1080/03052150903104366

Engineering Optimization
Vol. 42, No. 2, February 2010, 101–117

Taylor & Francis
Taylor & Francis Group

A local-best harmony search algorithm with dynamic subpopulations

Quan-Ke Pan[a], P. N. Suganthan[b]*, J.J. Liang[c] and M. Fatih Tasgetiren[d]

[a]*College of Computer Science, Liaocheng University, Liaocheng, P.R. China;* [b]*School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore;* [c]*School of Electrical Engineering, Zhengzhou University, Zhengzhou, P.R. China;* [d]*Department of Industrial Engineering, Yasar University, Bornova, Izmir, Turkey*

(*Received 15 December 2008; final version received 14 May 2009*)

This article presents a local-best harmony search algorithm with dynamic subpopulations (DLHS) for solving the bound-constrained continuous optimization problems. Unlike existing harmony search algorithms, the DLHS algorithm divides the whole harmony memory (HM) into many small-sized sub-HMs and the evolution is performed in each sub-HM independently. To maintain the diversity of the population and to improve the accuracy of the final solution, information exchange among the sub-HMs is achieved by using a periodic regrouping schedule. Furthermore, a novel harmony improvisation scheme is employed to benefit from good information captured in the local best harmony vector. In addition, an adaptive strategy is developed to adjust the parameters to suit the particular problems or the particular phases of search process. Extensive computational simulations and comparisons are carried out by employing a set of 16 benchmark problems from the literature. The computational results show that, overall, the proposed DLHS algorithm is more effective or at least competitive in finding near-optimal solutions compared with state-of-the-art harmony search variants.

Keywords: harmony search; dynamic subpopulations; evolutionary algorithms; continuous optimization

1. Introduction

Optimization has been an active area of research for several decades. As many real-world problems involve optimization with respect to a set of parameters and become increasingly complex, better optimization algorithms are always needed. Without loss of the generality, bound-constrained optimization problems can be formulated as an n-dimensional minimization problem as follows:

$$\text{Min } f(x) \quad st : x(j) \in [LB(j), UB(j)], \quad j = 1, 2, \ldots, n, \tag{1}$$

where $f(x)$ is the objective function, $x = (x(1), x(2), \ldots, x(n))$ is the set of design variables, n is the number of design variables, and $LB(j)$ and $UB(j)$ are the lower and upper bounds, respectively, for the design variable $x(j)$.

*Corresponding author. Email: epnsugan@ntu.edu.sg

ISSN 0305-215X print/ISSN 1029-0273 online

DOI: 10.1080/03052150903104366
http://www.informaworld.com

The harmony search (HS) algorithm is one of the recent evolutionary computation techniques for solving optimization problems (Geem *et al*. 2001). Unlike other evolutionary algorithms, which simulate natural selection and biological evolution, the HS algorithm is conceptualized by using the improvisation process that occurs when a musician searches for a better state of harmony. The harmony in music is analogous to the optimization solution vector, and the musician's improvisation is analogous to local and global search schemes in optimization techniques (Geem *et al*. 2001). In comparison to other meta-heuristics in the literature, the HS algorithm imposes fewer mathematical requirements and can be easily adapted for solving diverse optimization problems (Cheng *et al*. 2007, Cheng *et al*. 2008, Lee and Geem 2005). Furthermore, numerical comparisons have demonstrated that the convergence in the HS algorithm is faster than genetic algorithm (Mahdavi *et al*. 2007, Lee and Geem 2005, Lee *et al*. 2005). Therefore, the HS algorithm has captured much attention and has been applied to solve many practical optimization problems (Cheng *et al*. 2007, Cheng *et al*. 2008, Lee and Geem 2005, Lee *et al*. 2005, Geem 2006, Lee and Geem 2004, Ayvaz 2007, Geem *et al*. 2005, Degertekin 2008, Forsati *et al*. 2008).

Although the HS algorithm is good at identifying good regions in the search space within a reasonable time, it is not so efficient in performing local search in numerical optimization applications (Mahdavi *et al*. 2007). Thus, several novel variants of the HS algorithm were presented to enhance its solution accuracy recently. For example, Mahdavi *et al*. (2007) presented an improved HS (IHS) algorithm by introducing a strategy to dynamically adjust the control parameters. The authors' experiments revealed that the IHS variant could find better solutions than the basic HS algorithm. Later, Omran and Mahdavi (2008) proposed a global best HS algorithm, the GHS algorithm, by taking advantage of the global best solution to enhance the performance of the HS algorithm. It was demonstrated by the authors that the GHS algorithm was not only better than the original HS algorithm, but better than the IHS algorithm as well. More recently, on the basis of the idea that the better harmony vector has a higher selection probability, and that a number of new harmonies are generated in one iteration, Cheng *et al.* (2007, 2008) developed another improved HS algorithm, called the MHS algorithm, which was found by the authors to be more efficient than the basic HS algorithm for slope stability analysis.

In this article, a local-best harmony search algorithm with dynamic subpopulations (DLHS) is presented for solving continuous optimization problems. In the proposed DLHS algorithm, the whole harmony memory (HM) is divided into many small-sized sub-HMs, and the independent evolution is performed in each sub-HM. To keep population diversity, the small-sized sub-HMs are regrouped frequently by using a regrouping schedule to exchange information among the solutions. Furthermore, inspired by the work of Omran and Mahdavi (2008), an improved improvisation scheme is employed to generate candidate solutions so as to effectively take advantage of the information captured in the local best solution. In addition, a novel adaptive parameter-selection strategy is applied to match the parameters and the problems and the particular phases of evolution. Experimental results and comparisons show that the proposed DLHS algorithm generally outperforms the existing HS, IHS, GHS, and MHS algorithms when applied to 16 benchmark global optimization problems.

The rest of the article is organized as follows. In Section 2, the basic HS algorithm is described. In Section 3, the DLHS algorithm is described in detail. The benchmark problems are listed in Section 4. Experimental design and comparisons are presented in Section 5. Finally, Section 6 gives the concluding remarks.

2. The basic HS algorithm

The HS algorithm mimics the improvisation process whereby musicians search for a perfect state of harmony. In the HS algorithm, each solution is called a 'harmony' and represented by an

n-dimension real-valued vector. An initial population of harmony vectors are randomly generated and stored in the HM. Then a new candidate harmony is generated by using a memory consideration rule, a pitch adjustment rule, and a random re-initialization scheme. Finally, the HM is updated by comparing the new candidate harmony and the worst one in the HM. The above process is repeated until a termination criterion is met. The basic HS algorithm consists of three basic phases, namely initialization, improvisation of a harmony vector, and updating the HM. These processes are described below.

2.1. *Initialization*

The initialization of the basic HS algorithm consists of two steps. The first step is to set the control parameters including the harmony memory size (*HMS*), harmony memory consideration rate (*HMCR*), pitch adjustment rate (*PAR*), distance bandwidth (*BW*), and termination criterion. The second step is to fill the HM with *HMS* number of harmony vectors. Let $X_i = \{x_i(1), x_i(2), \ldots, x_i(n)\}$ represents the ith harmony vector, which is randomly generated as follows:

$$x_i(j) = LB(j) + (UB(j) - LB(j)) \times rand() \quad \text{for } j = 1, 2, \ldots, n \text{ and } i = 1, 2, \ldots HMS \quad (2)$$

where $rand()$ is a uniform random function returning a number between 0 and 1. Then, the HM matrix is filled with the *HMS* number of randomly generated harmony vectors.

2.2. *Improvise a new harmony*

Generate a new harmony vector $X_{new} = \{x_{new}(1), x_{new}(2), \cdots, x_{new}(n)\}$ by using the process called improvisation. X_{new} is produced based on three rules: memory consideration; pitch adjustment; and random selection. First of all, if $rand()$ is less than *HMCR*, $x_{new}(j)$ is generated by the memory consideration; otherwise, $x_{new}(j)$ is obtained by a random selection. Secondly, each $x_{new}(j)$ will undergo a pitch adjustment with a probability of *PAR* if it is updated by the memory consideration. The memory consideration, pitch adjustment and random selection are given below in Equations (3), (4), and (5), respectively.

$$x_{new}(j) = x_a(j) \quad \text{where } a \in (1, 2, \ldots, HMS) \quad (3)$$

$$x_{new}(j) = x_{new}(j) \pm rand() \times BW \quad (4)$$

$$x_{new}(j) = LB_j + rand() \times (UB_j - LB_j) \quad (5)$$

2.3. *Update harmony memory*

After X_{new} is generated, the HM will be updated by the survival of the fitter competition between X_{new} and the worst harmony vector X_w in the HM. That is, X_{new} will replace X_w and become a new member of the HM if X_{new} is better than X_W.

2.4. *Computational procedure*

The computational procedure of the basic HS algorithm can be summarized as follows (Geem *et al*. 2001):

Step 1: Set the parameters and initialize the HM.
Step 2: Improvise a new harmony X_{new} as follows:

```
for (j = 1 to n) do
    if (rand() < HMCR) then                          // memory consideration
        x_new (j) = x_a (j) where a ∈ (1, 2, ..., HMS)
        if (rand() < PAR) then                       // pitch adjustment
            x_new (j) = x_new (j) ± rand() × BW
        endif
    else                                             // random selection
        x_new (j) = LB_j + rand() × (UB_j − LB_j)
    endif
endfor
```

Step 3: Update the HM as $X_W = X_{new}$ if $f(X_{new}) < f(X_W)$ (minimization objective)
Step 4: If termination criterion is reached, return the best harmony vector found so far; otherwise go to Step 2.

3. The proposed local best HS (LHS) algorithm

Inspired by the local version of the particle swarm optimization (PSO) algorithm (Blackwell and Branke 2004, Løvbjerg 2001, Liang and Suganthan 2005, Liang *et al*. 2007, Liang *et al*. 2006) and the work of Omran and Mahdavi (2008), a new local-best variant of the HS algorithm with dynamic subpopulations, called the DLHS algorithm, is proposed in this article. In the local PSO algorithm, each individual is called a particle with position and velocity. During the search process, each particle continuously updates its position and velocity according to its personal best, and the best performance achieved so far within its neighbourhood. That is, individual improvement, population cooperation within a local neighbourhood, and population competition direct the swarm towards the best position in the search space. Accordingly, in the proposed DLHS algorithm, the HM is divided into many small-sized sub-HMs, which are regrouped frequently by using a regrouping schedule, and the harmony search is performed in each sub-HM independently. Furthermore, in the proposed DLHS algorithm, a novel harmony improvisation scheme is employed to effectively take advantage of good information from local best harmony vector. An adaptive HS parameter selection method is used to dynamically match the parameter to particular problems and phases of search process.

3.1. *Dynamic subpopulations*

Since the small-sized HM works better than a large one for the HS algorithm (Omran and Mahdavi 2008), and dynamic subpopulation can effectively balance the fast convergence and large diversity (Liang *et al*. 2006), the proposed DLHS algorithm employs multiple small-sized dynamic sub-HMs. More specifically, the whole HM of the DLHS algorithm is first divided into m small-sized sub-HMs randomly. Then each sub-HM uses its own members to search for better region in the search space. Since each sub-HM has a very small population and evolves independently, the algorithm may converge to a local optimum due to the loss of diversity. In order to address this drawback, information is allowed to exchange among the sub-HMs with a frequency R. That is, every R iterations, the whole population is regrouped randomly into different sub-HM configurations and restarts searching by using the new configuration of the small sub-HM. In this way, the information obtained by each sub-HM during the previous regrouping period R is exchanged among the other harmony vectors. Simultaneously, the diversity of each sub-HM is increased. In the final search phase of the evolution, in order to stress the local exploitation, the best three

harmony vectors are selected to form a new single HM. The new HM is employed to perform harmony search until the termination criterion is reached.

For example, suppose that nine harmony vectors are in a HM and divided randomly into three sub-HMs (see Figure 1). In the first phase of the algorithm, these three sub-HMs use their own members to search for better solutions independently. After R iterations, these sub-HMs form a whole HM which is regrouped randomly into three new sub-HMs. Then the new sub-HMs restart their search. This process is continued until the termination criterion of the first stage is satisfied. In the final phase, the best three harmony vectors are chosen to form a new HM. Then the algorithm restart search in this new HM until the certain termination criterion is reached.

3.2. *New scheme for improvising a harmony*

In the basic HS algorithm, a new harmony vector is generated by considering all of the existing vectors in the HM with equal probability. This will decrease the search efficacy, since the best harmony vector often carries better information than others during the evolution process, and the search space around it could be the most promising region. It is obvious that the best harmony vector should be given higher probability to produce offspring, as was done in Omran and Mahdavi (2008). Thus, in the proposed DLHS algorithm, advantage is taken of the local-best harmony vector

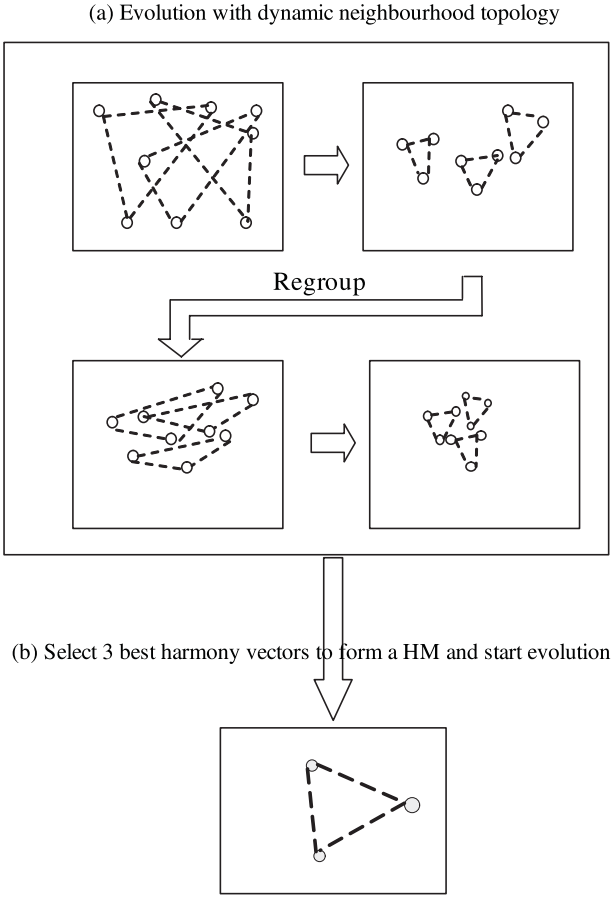

Figure 1. DLHS search.

X_{lB} to produce a new harmony vector X_{new} in each sub-HM, and the memory consideration phase is correspondingly modified as follows:

$$x_{new}(j) = x_{lB}(j) \quad j = 1, 2, \ldots, n. \tag{6}$$

In addition, in the pitch adjustment rule, Equation (4) in HS is replaced by Equation (7), in order to avoid getting trapped in a locally optimal solution.

$$x_{new}(j) = x_a(j) \pm rand() \times BW \quad \text{where } a \in (1, 2, \ldots, HMS) \tag{7}$$

The new improvisation scheme can be given as follows:

```
for (j = 1 to n) do
    if (rand() < HMCR) then                    // memory consideration
      x_new (j) = x_lB (j)
      if (rand() < PAR) then                   // pitch adjustment
        x_new (j) = x_a (j) ± rand() × BW
      endif
    else                                       // random selection
      x_new (j) = LB_j + rand() × (UB_j − LB_j)
    endif
endfor
```

3.3. *Parameter setting for DLHS*

In the DLHS algorithm, four parameters *HMS*, *HMCR*, *PAR*, and *BW* are closely related to the problem being solved and the phase of the search process which may be either exploration or exploitation. It is obvious that a good set of parameters can enhance the algorithm's ability to search for the global optimum or near optimum region with a high convergence rate. In this article, *HMS* is kept as a user-specified value so as to deal with different problems with different dimensions. The other three parameters are either self-learning or dynamically adapted with respect to the favorable evolution of the search process, as presented below.

3.3.1. *Self-adaptation of HMCR and PAR*

HMCR is the probability of choosing one value from the historic values stored in the sub-HM. A large *HMCR* value is in favour of local search thereby increasing the convergence rate of the algorithm, while a small *HMCR* value increases the diversity of the harmony memory. By extensive simulations, Omran and Mahdavi (2008) suggested that it was generally better to use a large value for *HMCR* (*i.e.* $\geq$0.9). *PAR* is the adjustment rate for the pitch chosen from X_{lB}. A small *PAR* value favours passing the information of X_{lB} to next generation, hence enhancing the local exploitation ability of the algorithm around X_{lB} whereas a large *PAR* value helps the new harmony vector select its dimensions from the sub-HM, thus enlarging the search area to global exploration. Since local exploitation and global exploration are always twisted together in the search process, it is difficult to fix the values for *HMCR* and *PAR* when solving a collection of problems. Therefore, in this article a novel adaptive mechanism is presented for determining the best values of *HMCR* and *PAR* by the learning mechanism so as to match the global exploration and local exploitation during the evolution process. The self-adaptive strategy is presented as follows.

In the first place, a parameter set list (PSL) with specified length is generated by filling the parameter sets consisting of *HMCR* and *PAR*, where these parameter sets are randomly generated

with a uniform distribution in the ranges $HMCR \in [0.9, 1]$ and $PAR \in [0, 1]$. Then the DLHS algorithm is started with the parameter set from the PSL. In each generation, a parameter set is taken out from the PSL and used to improvise a new harmony vector. If the improvised harmony vector successfully replaces the worst member in its sub-HM, the associated parameter set will enter into a winning parameter set list (WPSL). Once the PSL is empty, it is refilled by randomly selecting parameter sets from the WPSL (75%) and the randomly generated parameter sets (25%). That is to say, 75% of the elements of the PSL come from the WPSL by random selection, while the remaining 25% are generated with a uniform distribution in their ranges. If the WPSL is empty (this may happen when the search is performed near an optimum with negligible population diversity), the last PSL is used again. The above process is repeated until the termination criterion is reached. As a result, the proper parameter set of *HMCR* and *PAR* values can be gradually learned to suit the particular problem and the particular phase of search process. Note that the WPSL is reset to zero once the PSL is refilled, to avoid any inappropriate long-term accumulation effects. In our experiments, the length of PSL is set as 200, and the influence on the performance of the DLHS algorithm due to small variations of the PSL length is found to be insignificant.

An example is given as follows. Suppose that a PSL with length equal to five is randomly generated (Figure 2(a)). In the first iteration, the first parameter set, *i.e.* $HMCR = 0.91$ and $PAR = 0.5$, is taken out (Figure 2(b)) and used to improvise a new harmony X_{new}. Given that X_{new} is better than the worst harmony X_w in the current sub-HM. It will replace X_w and become a member of the sub-HM in the next iteration. Accordingly, the parameter set $HMCR = 0.91$ and $PAR = 0.5$ will enter into WPSL (Figure 2(c)). In the second iteration, the second parameter set $HMCR = 0.98$ and $PAR = 0.4$ is taken out (Figure 2(d)) and used to yield another X_{new}. Suppose that the current X_{new} is worse than the current X_w, the parameter set $HMCR = 0.98$ and $PAR = 0.4$ will be discarded. Repeat the above process until all the parameter sets are taken out and PSL is empty (Figures 2(f) and 2(g)).

The refilling process for the PSL then starts. A uniform random number between 0 and 1 is generated firstly. Suppose that the random number is less than 0.75, then randomly select a parameter set from WPSL (e.g. $HMCR = 0.95$, $PAR = 0.6$) and put it in the first position of PSL (Figure 3(a)). Secondly, another uniform random number is generated. Suppose that it is greater than 0.75, then *HMCR* is randomly produced between 0.9 and 1, *PAR* between 0 and 1, and put them into second position of PSL (Figure 3(b)). Repeat the above process until the PSL is filled, and WPSL is reset empty (Figures 3(c) and 3(d)).

3.3.2. *Dynamically changing BW*

The parameter *BW* is a distance bandwidth for the continuous design variable. A large *BW* value is in favour of the algorithm performing the global search, while a small *BW* value is appropriate for fine-search around the best solution vectors. To well balance the exploration and exploitation of the proposed DLHS algorithm, the *BW* value decreases dynamically with increasing function evaluations (*FE*s) as follows:

$$BW(t) = \begin{cases} BW_{\max} - \dfrac{BW_{\max} - BW_{\min}}{\max\_FEs} \times 2FEs & if\ FEs < \max\_FEs/2 \\ BW_{\min} & \text{otherwise} \end{cases} \tag{8}$$

where $BW_{\max}$ and $BW_{\min}$ are the maximum and minimum distance bandwidths, respectively. *Max_FEs* is the maximum function evaluations.

The DLHS algorithm does not require a precise setting of specific values to critical parameters *HMCR*, *PAR*, and *BW* in accordance with problem's characteristic and complexity. These parameters are either self-adapted by a learning mechanism or dynamically adjusted with increasing

PSL		
Index	*HMCR*	*PAR*
1	0.91	0.5
2	0.98	0.4
3	0.95	0.6
4	0.92	0.2
5	0.99	0.8

a) A PSL with length equal to 5

PSL		
Index	*HMCR*	*PAR*
1		
2	0.98	0.4
3	0.95	0.6
4	0.92	0.2
5	0.99	0.8

b) A PSL after first iteration

WPSL		
Index	*HMCR*	*PAR*
1	0.91	0.5

c) The WPSL after first iteration

PSL		
Index	*HMCR*	*PAR*
1		
2		
3	0.95	0.6
4	0.92	0.2
5	0.99	0.8

d) The PSL after second iteration

WPSL		
Index	*HMCR*	*PAR*
1	0.91	0.5

e) The WPSL after second iteration

PSL		
Index	*HMCR*	*PAR*
1		
2		
3		
4		
5		

f) The PSL after fifth iteration

WPSL		
Index	*HMCR*	*PAR*
1	0.91	0.5
2	0.95	0.6
3	0.92	0.2

g) The WPSL after fifth iteration

Figure 2. The process for taking out parameter sets from PSL.

generation count. Therefore, the DLHS algorithm can demonstrate consistently good performance on problems with different properties.

3.4. *Computational procedure of DLHS*

On the basis of the above dynamic subpopulation concept, new scheme for improvising a harmony, and parameter-setting method, the computational procedure of the proposed DLHS algorithm can be given as follows:

Step 1: Set *HMS*, $BW_{\max}$, $BW_{\min}$, m, R and *Max_FE*s.
Step 2: Initialize HM and PSL.
Step 3: Perform Steps 4 to 6 until the *FE*s is equal to $0.9 \times$ *Max_FE*s.
Step 4: Randomly divide HM into m sub-HMs with the same size.
Step 5: For each sub-HM perform the following Steps 5.1 to 5.4.
Step 5.1: Select a set of *HMCR* and *PAR* from PSL, and determine *BW* according to Equation (8).

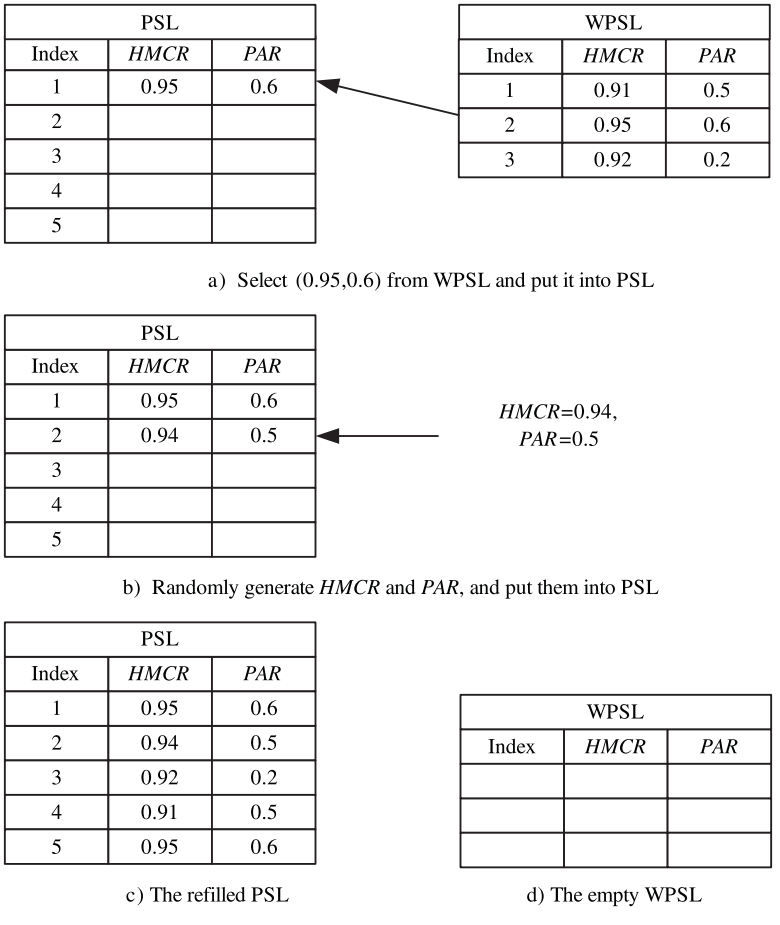

Figure 3. The process for refilling PSL.

Step 5.2: Improvise a new harmony X_{new} as follows:

```
for (j = 1 to n) do
    if (rand() < HMCR) then                    // memory consideration
      x_new (j) = x_lB (j)
      if (rand() < PAR) then                   // pitch adjustment
        x_new (j) = x_a (j) ± rand() × BW
      endif
    else                                       // random selection
      x_new (j) = LB_j + rand() × (UB_j − LB_j)
    endif
endfor
```

Step 5.3: Update the sub-HM and record the current *HMCR* and *PAR* into WPSL, if X_{new} is better than the worst harmony vector X_w in the sub-HM.
Step 5.4: If PSL is empty, refill it as explained in Section 3.3.
Step 6: If 90% of the total *FE*s (function evaluations) are exhausted, go to Step 7. If the regrouping period R is reached, go to Step 4. Otherwise go to Step 5.

Step 7: Select the best 3 vectors from HM and form a new single HM. Repeat the Steps 7.1 to 7.3 until termination criterion is met.
Step 7.1: Randomly select a set of *HMCR* and *PAR* from PSL, and determine *BW* according to Equation (8).
Step 7.2: Improvise a new harmony X_{new} as follows:

```
for (j = 1    to n) do
    if (rand() < HMCR) then                    // memory consideration
      x_new (j) = x_B (j)
      if (rand() < PAR) then                   // pitch adjustment
        x_new (j) = x_a (j) ± rand() × BW
      endif
    else                                       // random selection
      x_new (j) = LB_j + rand() × (UB_j − LB_j)
    endif
endfor
```

Step 7.3: Update the new HM if X_{new} is better than the worst harmony vector X_w in the current new HM.
Step 8: Output the best harmony vector found so far.

4. Benchmark functions

This section lists the global minimization benchmark functions used to evaluate the proposed algorithm with other variants of the HS algorithms. These benchmark functions provide a balance of unimodal and multimodal functions, taken from evolutionary computation literature (Yao *et al*. 1999, Suganthan *et al*. 2005). The following functions are used:

A Sphere function, defined as

$$f(x) = \sum_{i=1}^{n} x^2(i),$$

where global optimum $x^* = 0$ and $f(x^*) = 0$ for $-100 \le x(i) \le 100$.

B Schwefel's problem 2.22, defined as

$$f(x) = \sum_{i=1}^{n} |x(i)| + \prod_{i=1}^{n} |x(i)|,$$

Where global optimum $x^* = 0$ and $f(x^*) = 0$ for $-10 \le x(i) \le 10$.

C Rosenbrock function, defined as

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

where global optimum $x^* = (1, 1, \ldots 1)$ and $f(x^*) = 0$ for $-30 \le x(i) \le 30$.

D Step function, defined as

$$f(x) = \sum_{i=1}^{n} (\lfloor x(i) + 0.5 \rfloor)^2,$$

where global optimum $x^* = 0$ and $f(x^*) = 0$ for $-100 \le x(i) \le 100$.

E Rotated hyper-ellipsoid function, defined as

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x(j) \right)^2,$$

where global optimum $x^* = 0$ and $f(x^*) = 0$ for $-100 \leq x(i) \leq 100$.

F Generalized Schwefel problem 2.26, defined as

$$f(x) = -\sum_{i=1}^{n} \left(x(i) \sin(\sqrt{|x(i)|}) \right),$$

where global optimum $x^* = (420.9687, 420.9687, \ldots 420.9687)$ and $f(x^*) = -12569.5$ for $-500 \leq x(i) \leq 500$.

G Rastrigin function, defined as

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

where global optimum $x^* = 0$ and $f(x^*) = 0$ for $-5.12 \leq x(i) \leq 5.12$.

H Ackley's function, defined as

$$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{n} x^2(i)} \right) - \exp\left(\frac{1}{30}\sum_{i=1}^{n} \cos(2\pi x(i)) \right) + 20 + e,$$

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

I Griewank function, defined as

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

where global optimum $x^* = 0$ and $f(x^*) = 0$ for $-600 \leq x(i) \leq 600$.

J Six-hump camel-back function, defined as

$$f(x) = 4x^2(1) - 2.1x^4(1) + \frac{1}{3}x^6(1) + x(1) \times x(2) - 4x^2(2) + 4x^4(2),$$

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

K Shifted Sphere function

$$f(x) = \sum_{i=1}^{n} z^2(i) + f\_bias_1,$$

where $z = x - o$; $o = \{o(1), o(2) \ldots, o(n)\}$ is the shifted global optimum; global optimum $x^* = o$ and $f(x^*) = f\_bias_1 = -450$ for $-100 \leq x(i) \leq 100$.

L Shifted Schwefel problem 1.2

$$f(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} z(j) \right)^2 + f\_bias_2,$$

where $z = x - o$; $o = \{o(1), o(2) \ldots, o(n)\}$ is the shifted global optimum; global optimum $x^* = o$ and $f(x^*) = f\_bias_2 = -450$ for $-100 \leq x(i) \leq 100$.

M Shifted Rosenbrock function, defined as

$$f(x) = \sum_{i=1}^{n-1}(100(z(i+1) - x^2(i))^2 + (z(i) - 1)^2) + f\_bias_6,$$

where $z = x - o + 1$; $o = \{o(1), o(2) \ldots, o(n)\}$ is the shifted global optimum; global optimum $x^* = o$ and $f(x^*) = f\_bias_6 = 390$ for $-100 \leq x(i) \leq 100$.

N Shifted Rastrigin function, defined as

$$f(x) = \sum_{i=1}^{n}(z^2(i) - 10\cos(2\pi z(i)) + 10) + f\_bias9,$$

where $z = x - o$; $o = \{o(1), o(2) \ldots, o(n)\}$ is the shifted global optimum; global optimum $x^* = o$ and $f(x^*) = f\_bias_9 = -330$ for $-5 \leq x(i) \leq 5$.

O Shifted Rotated High Conditional Elliptic Function, defined as

$$f(x) = \sum_{i=1}^{n}(10^6)^{(i-1)/(n-1)}z^2(i) + f\_bias3$$

where $z = (x - o) \times M$, and $o = \{o(1), o(2) \ldots, o(n)\}$ is the shifted global optimum, M is liner transformation matrix; global optimum $x^* = o$ and $f(x^*) = f\_bias3 = -450$ for $-100 \leq x(i) \leq 100$.

P Shifted Rotated Griewank's Function without Bounds, defined as

$$f(x) = \sum_{i=1}^{n}\frac{z^2(i)}{4000} - \prod_{i=1}^{n}\cos\left(\frac{z(i)}{i}\right) + 1 + f\_bias7$$

where $z = (x - o) \times M$, and $o = \{o(1), o(2) \ldots, o(n)\}$ is the shifted global optimum, M is liner transformation matrix; global optimum $x^* = o$ and $f(x^*) = f\_bias7 = -180$ for $-100 \leq x(i) \leq 100$.

Among the above 16 benchmark problems, the Sphere function, Schwefel Problem 2.22, Step function, Rotated Hyper-Ellipsoid function, Shifted Rotated High Conditioned Elliptic function, Shifted Sphere function, and Shifted Schwefel Problem 1.2 are unimodal. Step function is discontinuous. Rosenbrock function, Schwefel Problem 2.26, Rastrigin function, Schwefel Problem 2.26, Ackley function, Griewank function, Shifted Rosenbrock function, Shifted Rastrigin function, and Shifted Rotated Griewank function are difficult multimodal problems where the number of local optima increases with the problem dimension. Six-hump camel-back function is a low-dimensional function with only a few local optima.

5. Computational results

5.1. *Experimental setup*

To test the performance of the proposed DLHS algorithm, an extensive experimental evaluation and comparison with the HS, IHS, GHS, and MHS algorithms are provided based on the 16 benchmark optimization problems listed in Section 4 with 30 and 50 dimensions (except Six-hump camel-back function with 2 dimensions). All the compared algorithms are coded in Visual C++ and execute the experiment on a Pentium® PIV 3.0 GHz PC with 512 MB memory. For the

Table 1. Average error (AE) values and standard deviations (SD) generated by the compared algorithms ($n = 30$).

	IHS		GHS		HS		MHS		DLHS	
Problem	AE	SD	AE	SD	AE	SD	AE	SD	AE	SD
A	4.716702e−007	1.308007e−007	1.172420e−002	1.807095e−002	7.235628e+000	3.236447e+000	2.762490e−005	8.937370e−006	1.299296e−009	2.766409e−009
B	9.558302e−003	2.385677e−002	3.812779e−002	2.882198e−002	1.035849e−001	5.389395e−002	1.258202e−002	1.886592e−003	1.234472e−004	2.268207e−004
C	2.332179e+002	2.579212e+002	5.527813e+001	5.546510e+001	4.020729e+002	6.191397e+002	1.319807e+002	1.642222e+002	2.283165e+002	2.507772e+002
D	4.666667e−001	8.995529e−001	0.000000e+000	0.000000e+000	3.333333e+000	2.073367e+000	3.000000e−001	5.349831e−001	1.333333e+000	2.770949e+000
E	4.155316e+003	1.089887e+003	6.253290e+003	7.456851e+003	4.433246e+003	1.046275e+003	5.483608e+003	2.205458e+003	9.028620e+002	4.663480e+002
F	1.652893e−001	4.949998e−001	6.526251e−002	9.360273e−002	2.764240e+001	1.260249e+001	1.342464e−002	6.963275e−005	6.785688e−003	6.907049e−003
G	1.970091e+000	1.251774e+000	4.973614e−003	8.458331e−003	8.587395e−001	7.556476e−001	7.157104e−002	2.520678e−001	1.862979e+000	1.339693e+000
H	6.663751e−001	5.412766e−001	2.429043e−002	2.061595e−002	9.914932e−001	3.405301e−001	6.025625e−003	1.947375e−003	1.909532e+000	6.838100e−001
I	1.000725e+000	1.719085e−003	1.000122e+000	1.709927e−004	1.087766e+000	3.225293e−002	1.000006e+000	3.006508e−005	1.000000e+000	1.166585e−006
J	4.651142e−008	1.572752e−012	4.157365e−005	6.907631e−005	4.657144e−008	6.934321e−011	5.774731e−008	1.354077e−008	4.651022e−008	1.531212e−013
K	4.629052e−007	1.274560e−007	1.803211e+003	3.617633e+002	6.446807e+000	2.777075e+000	2.700876e−005	1.086084e−005	2.443522e−007	1.331816e−006
L	4.068391e+003	1.734997e+003	1.889050e+004	4.537944e+003	3.888179e+003	1.115259e+003	6.860953e+003	2.426299e+003	2.843568e+003	1.766875e+003
M	1.730733e+003	2.950501e+003	3.504655e+007	2.213643e+007	3.400700e+003	3.271574e+003	2.513942e+004	1.581945e+004	3.779258e+003	4.838043e+003
N	1.777448e+000	8.080342e−001	6.672805e+001	9.356209e+000	8.710275e−001	8.086818e−001	7.892778e+000	2.219472e+000	1.578081e+000	1.499639e+000
O	1.466480e+007	6.682298e+006	6.830344e+007	2.549566e+007	1.500038e+007	4.455539e+006	1.848607e+007	5.310656e+006	3.194379e+006	1.720226e+006
P	3.385125e+003	8.982705e+001	4.075064e+003	2.126846e+001	4.079914e+003	5.733071e+000	3.895360e+003	1.169755e+002	9.676603e+002	2.213311e+002

DLHS algorithm, the parameters are set as $HMS = 9$, $BW_{\max} = (UB(j) - LB(j))/200$, $BW_{\min} = 0.0001$, $R = 50$, and $m = 3$. The parameters for the HS, IHS, and GHS algorithms are fixed the same as those in Omran and Mahdavi (2008): (a) for the HS algorithm, $HMS = 5$, $HMCR = 0.9$, $PAR = 0.3$ and $BW = 0.01$; (b) for the IHS algorithm, $HMS = 5$, $HMCR = 0.9$, $BW_{\max} = (UB(j) - LB(j))/20$, $BW_{\min} = 0.0001$, $PAR_{\min} = 0.01$, and $PAR_{\max} = 0.99$; and (c) for the GHS algorithm, $HMS = 5$, $HMCR = 0.9$, $PAR_{\min} = 0.01$, and $PAR_{\max} = 0.99$. The parameters for the MHS algorithm are set as those in Cheng *et al*. (2007): $HMS = 2n$, $HMCR = 0.98$, $PAR = 0.1$, $Nhm = 0.1 \times HMS$, $N_{m1} = 500$, and $N_{m2} = 200$. In the experiments, each problem is run for 30 independent replications and each replication is allowed to run 50,000 evaluations of the objective function ($Max\_FES = 50,000$) when solving 30-dimensional problems and 100,000 ($Max\_FES = 100,000$) when solving their 50-dimensional counterparts. The average error (AE) and standard deviations (SD) over these 30 replications of the 16 benchmark functions are calculated as the statistics for the performance measures.

5.2. *Results for the 30-dimensional problems*

Table 1 presents the AE and SD over these 30 runs of the compared HS algorithms on the 16 test functions with dimension equal to 30 (except for the two-dimensional six-hump camel-back function). In order to determine whether the results obtained by the DLHS algorithm are statistically different from those by the HS, IHS, GHS, and MHS algorithms, the two-sided paired t-tests with the 5% significance level are conducted for the DLHS against IHS, DLHS against GHS, DLHS against HS, and DLHS against MHS algorithms. The h values reported in the Table 2 are the results of t-tests. An h value equal to 1 or -1 indicates that results obtained by the former algorithm is significantly better or worse than those by the later one, while h value equal to zero implies that the results by the two compared algorithms are not statistically different.

It can be easily seen from Tables 1 and 2 that the proposed DLHS algorithm performs best among the compared algorithms since it produces 13, 15, 12, and 14 significantly better or competitive AE values out of 16 problems than the HS, IHS, GHS and MHS algorithms, respectively. More specifically, compared with the GHS algorithm, the proposed DLHS algorithm yields significantly better AE values for 12 problems but significantly worse for four problems; that is, Rosenbrock function, Step function, Rastrigin function, and Ackley's function. Note that Step function, Rastrigin function, and Ackley's function are separable and can be solved by using n one-dimensionality searches, which are in favor of the GHS algorithm. When compared with the IHS algorithm, the DLHS algorithm yields eight significantly better, seven competitive, and one significantly worse AE values, separately. Whereas, the DLSH algorithm outperforms both the HS and MHS algorithms for eight problems, but is surpassed by HS and MHS algorithms for three and two functions, separately. For the other problems, the differences produced by the compared algorithms are not statistically significant at the 5% significance level. In addition, the convergence characteristics of the best run of each algorithm for each test function further show that overall the proposed DLHS algorithm converges much faster to reach lower levels than those of the HS, IHS, GHS, and MHS algorithms. Hence, it is concluded that the proposed DLHS

Table 2. Paired t-test results for the compared algorithms ($n = 30$).

Problem	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
DLHS, IHS	1	1	0	0	1	0	0	−1	1	1	0	1	0	0	1	1
DLHS,GHS	1	1	−1	−1	1	1	−1	−1	1	1	1	1	1	1	1	1
DLHS,HS	1	1	0	1	1	1	−1	−1	1	1	1	1	0	−1	1	1
DLHS,MHS	1	1	0	0	1	1	−1	−1	0	1	1	1	1	1	1	1

Table 3. Average error (AE) values and standard deviations (SD) generated by the compared algorithms ($n = 50$).

	IHS		GHS		HS		MHS		DLHS	
Problem	AE	SD	AE	SD	AE	SD	AE	SD	AE	SD
A	1.711017e+002	4.355921e+001	3.685046e−001	4.050945e−001	1.983843e+002	4.620486e+001	6.395574e−004	1.459474e−004	6.634742e−010	7.547699e−010
B	3.808752e+000	6.445648e−001	1.691946e−001	1.511031e−001	5.492385e+000	8.073660e−001	8.634210e−002	1.231139e−002	9.558785e−005	5.037300e−005
C	5.706621e+003	2.012067e+003	1.714861e+002	2.065773e+002	6.182869e+003	1.826151e+003	1.045949e+002	3.964892e+001	3.115616e+002	4.475919e+002
D	1.229000e+002	4.128317e+001	0.000000e+000	0.000000e+000	1.408000e+002	3.986996e+001	1.033333e+000	1.351457e+000	6.000000e−001	1.037238e+000
E	2.073968e+004	4.294607e+003	5.114420e+004	1.765592e+004	2.135296e+004	4.805949e+003	1.089425e+004	2.187877e+003	3.734342e+003	1.987639e+003
F	3.436580e−002	3.413408e−002	2.894834e−002	2.236191e−002	2.588805e−002	2.585380e−002	4.238283e−005	4.689524e−005	3.579741e−008	3.380678e−008
G	2.344046e+001	3.594388e+000	1.048578e−001	1.509820e−001	1.894834e+001	2.840074e+000	2.366429e−001	3.413594e−001	3.402879e+000	3.495566e+000
H	4.992222e+000	1.052036e+000	1.164628e−003	1.024043e−003	3.947255e+000	5.108282e−001	1.333747e−006	1.149842e−006	8.404242e−001	1.175427e+000
I	2.746673e+000	3.487459e−001	1.002956e+000	2.962747e−003	2.712054e+000	3.734636e−001	1.000000e+000	0.000000e+000	1.000000e+000	2.537081e−007
J	4.651050e−008	5.614584e−013	1.636176e−005	1.959895e−005	4.651938e−008	9.502959e−012	5.774731e−008	1.354077e−008	4.651014e−008	3.058059e−014
K	1.845193e+002	4.298849e+001	1.057371e+004	1.650988e+003	1.849752e+002	3.954686e+001	6.264417e−004	1.531245e−004	7.836820e−010	8.044724e−010
L	2.199583e+004	4.468100e+003	7.647648e+004	1.191825e+004	2.044531e+004	5.438337e+003	1.394327e+004	2.596852e+003	7.981563e+003	3.285940e+003
M	4.270314e+005	2.072882e+005	7.670155e+008	1.808080e+008	3.812089e+005	2.165696e+005	4.671698e+006	1.851266e+006	3.084812e+003	4.110394e+003
N	2.440209e+001	3.810229e+000	1.882191e+002	2.166235e+001	1.937830e+001	4.037571e+000	5.789840e+001	8.607486e+000	2.572159e+000	2.501262e+000
O	7.378912e+007	1.759401e+007	2.908354e+008	8.911107e+007	8.061911e+007	2.580984e+007	1.070944e+008	3.025740e+007	1.011791e+007	3.089581e+006
P	7.717431e+003	8.341777e+001	8.042574e+003	9.704456e+001	7.998874e+003	3.126196e+001	7.789929e+003	2.757604e−002	1.012136e+003	4.713192e+002

Table 4. Paired t-test results for the compared algorithms ($n = 50$).

Problem	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
DLHS, IHS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DLHS,GHS	1	1	0	−1	1	1	−1	−1	1	1	1	1	1	1	1	1
DLHS,HS	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
DLHS,MHS	1	1	−1	0	1	1	−1	−1	0	1	1	1	1	1	1	1

algorithm is clearly superior to the existing HS, IHS, GHS, and MHS algorithms for solving continuous optimization problems.

5.3. *Results for the 50-dimensional problems*

The experiments conducted in Section 5.2 are repeated on the 50-dimensional problems. The AE and SD generated by the above five HS algorithms over 30 independent runs are reported in Table 3, and the results of two-sided paired t-tests for the DLHS against IHS, DLHS against GHS, DLHS against HS, and DLHS against MHS algorithms are given in Table 4.

From Tables 3 and 4, it can be observed that the algorithms achieve ranking similar to those in the 30-dimensional cases. The proposed DLHS algorithm outperforms all the compared algorithms on 11 problems (*i.e.* problems A, B E, F, J, K, L, M, N, O, and P), and no AE value obtained by the DLHS algorithm is beaten by other algorithms simultaneously. The 50-dimensional problems become more difficult than their 30-dimensional counterparts. Hence, the results are not as good as those in the 30-dimensional cases, although the maximum function evaluations are increased from 50,000 to 100,000. However, it can be easily seen that the proposed DLHS algorithm produces 16, 12, 16, and 11 significantly better results than the HS, IHS, GHS, and MHS algorithms, respectively. This further highlights the fact that the proposed DLHS algorithm is much more efficient and effective than the HS, IHS, GHS, and MHS algorithms in finding better solutions for the unconstrained optimization problems.

5.4. *Parameter study*

In this section, the 30-dimensional problems are employed to investigate the impact of refilling frequency R, maximum Bandwidth $BW_{\max}$, and minimum bandwidth $BW_{\min}$ on the DLHS algorithm. In the experiment, R varies from 30 to 110 with a step equal to 20; $BW_{\max}$ is set as $BW_{\max} = (UB(j) - LB(j))/100$, $BW_{\max} = (UB(j) - LB(j))/150$, $BW_{\max} = (UB(j) - LB(j))/200$, $BW_{\max} = (UB(j) - LB(j))/250$ or $BW_{\max} = (UB(j) - LB(j))/300$; $BW_{\min}$ is set as 1×10^{-3}, 5×10^{-4}, 1×10^{-4}, 5×10^{-5} or 1×10^{-5}. The computational experiments show that the DLHS algorithm with different RP, $BW_{\max}$, and $BW_{\min}$ values generates very similar results, which suggests that, overall, the DLHS algorithm is less sensitive to the parameters RP value in the range [0.7, 0.9], $BW_{\max}$ between $(UB(j) - LB(j))/300$ and $(UB(j) - LB(j))/100$, and $BW_{\min}$ between 1×10^{-3} and 1×10^{-5}.

6. Conclusions

In this article, a local-best harmony search (DLHS) algorithm is presented with dynamic subpopulation for solving continuous optimization problems. The proposed DLHS algorithm had four novel characteristics: (a) The whole HM was divided into many small-sized sub-HMs, and the search was performed in each sub-HM independently; (b) a regrouping schedule was frequently used to regroup the small-sized sub-HMs for sharing the obtained good information and enlarging

the population diversity; (c) a new harmony improvisation scheme was employed to well utilize good information from local best harmony vector; and (d) a novel adaptive parameter selection strategy was presented to match the parameters to the problems and the particular phases of search process. Numerical experiments based on the benchmark problems showed that the proposed algorithm was, overall, more effective in finding better solutions than the existing HS, IHS, GHS, and MHS algorithms. Future work will generalize the proposed DLHS algorithm to solve combinatorial and discrete optimization problems.

Acknowledgements

The authors wish to acknowledge the financial support offered by the A*Star (Agency for Science, Technology and Research, Singapore) under grant #052 101 0020. This research is also partially supported by National Science Foundation of China under grants 60874075 and 70871065, and Open Research Foundation from State Key Laboratory of Digital Manufacturing Equipment and Technology (Huazhong University of Science and Technology), and Postdoctoral Science Foundation of China under grant 20070410791.

References

Ayvaz, T.M., 2007. Simultaneous determination of aquifer parameters and zone structures with fuzzy c-means clustering and meta-heuristic harmony search algorithm. *Advances in Water Resources*, 30, 2326–2338.

Blackwell, T.M. and Branke, J., 2004. Multi-swarm optimization in dynamic environments. *In*: *LNCS No. 3005: Proceedings of Applications of Evolutionary Computing: EvoWorkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoISAP, EvoMUSART, and EvoSTOC*, Coimbra, Portugal. 489–500.

Cheng, Y.M., Li, L., and Chi, S.C., 2007. Performance studies on six heuristic global optimization methods in the location of critical slip surface. *Computers and Geotechnics*, l34, 462–484.

Cheng, Y.M., Li, L., Lansivaara, T., Chi, S.C., and Sun, Y.J., 2008. An improved harmony search minimization algorithm using different slip surface generation methods for slope stability analysis. *Engineering Optimization*, 40, 95–115.

Degertekin, S.O., 2008. Optimum design of steel frames using harmony search algorithm. *Structural and Multidisciplinary Optimization*, 36 (4), 393–401.

Forsati, R., Haghighat, A.T., and Mahdavi, M., 2008. Harmony search based algorithms for bandwidth-delay-constrained least-cost multicast routing. *Computer Communications*, 31 (10), 2505–2519.

Geem, Z.W., Kim, J.H., and Loganathan, G.V., 2001. A new heuristic optimization algorithm: Harmony search. *Simulations*, 76, 60–68.

Geem, Z.W., 2006. Optimal cost design of water distribution networks using harmony search. *Engineering Optimization*, 38, 259–280.

Geem, Z.W., Lee, K.S., and Park, Y.J., 2005. Application of harmony search to vehicle routing. *American Journal of Applied Sciences*, 2, 1552–1557.

Lee, K.S. and Geem, Z.W., 2005. A new meta-heuristic algorithm for continuous engineering optimization, harmony search theory and practice. *Computing Methods in Applied Mechanical Engineering*, 194, 3902–3933.

Lee, K.S., Geem, Z.W., Lee, S.H., and Bae, K.W., 2005. The harmony search heuristic algorithm for discrete structural optimization. *Engineering Optimization*, 37, 663–684.

Lee, K.S. and Geem, Z.W., 2004. A new structural optimization method based on the harmony search algorithm. *Computers and Structures*, 82, 781–798.

Liang, J.J. and Suganthan, P.N., 2005. Dynamic multi-swarm particle swarm optimizer. *In*: *Proceedings of the IEEE Swarm Intelligence Symposium* (*SIS* 2005), June 2005, 124 – 129.

Liang, J.J., Chan, C.C., Huang, V.L., and Suganthan, P.N., 2007. Improving the performance of FBG sensor networks using dynamic multi-swarm particle swarm optimizer. *Journal of Optoelectronics and Advanced Materials, Rapid Communications*, 1 (8), 373–378.

Liang, J.J., Suganthan, P.N., Chan, C.C., and Huang, V.L., 2006. Wavelength detection in FBG sensor network using tree search dynamic multi-swarm particle swarm optimizer. *IEEE Photonics Technical Letters*, 18 (12),1305–1307.

Løvbjerg M., Rasmussen T.K., and Krink T., 2001. Hybrid particle swarm optimiser with breeding and subpopulations. *In*: *Proceedings of the Genetic and Evolutionary Computation Conference*, (*GECCO* 2001), 469–476.

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

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

Suganthan, P.N., Hansen, N., and Liang, J.J., 2005. *Problem definitions and evaluation criteria for the CEC 2005 Special Session on Real-Parameter Optimization*. Technical Report, Nanyang Technological University, Singapore, May 2005 AND KanGAL Report #2005005, IIT Kanpur, India.

Yao, X., Liu, Y., and Lin, G., 1999. Evolutionary programming made faster. *IEEE Transactions on Evolution Computation*, 3 (2), 82–102.