Available online at www.sciencedirect.com

ScienceDirect

Applied Mathematics and Computation 198 (2008) 643–656

APPLIED MATHEMATICS AND COMPUTATION

www.elsevier.com/locate/amc

Global-best harmony search

Mahamed G.H. Omran [a,*], Mehrdad Mahdavi [b]

[a] Department of Computer Science, Gulf University for Science and Technology, Kuwait
[b] Department of Computer Engineering, Sharif University of Technology, Tehran, Iran

Abstract

Harmony search (HS) is a new meta-heuristic optimization method imitating the music improvisation process where musicians improvise their instruments' pitches searching for a perfect state of harmony. A new variant of HS, called *global-best harmony search* (GHS), is proposed in this paper where concepts from swarm intelligence are borrowed to enhance the performance of HS. The performance of the GHS is investigated and compared with HS and a recently developed variation of HS. The experiments conducted show that the GHS generally outperformed the other approaches when applied to ten benchmark problems. The effect of noise on the performance of the three HS variants is investigated and a scalability study is conducted. The effect of the GHS parameters is analyzed. Finally, the three HS variants are compared on several Integer Programming test problems. The results show that the three approaches seem to be an efficient alternative for solving Integer Programming problems.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Harmony search; Meta-heuristics; Evolutionary algorithms; Optimization

1. Introduction

Evolutionary algorithms (EAs) are general-purpose stochastic search methods simulating natural selection and biological evolution. EAs differ from other optimization methods, such as Hill-Climbing [18] and Simulated Annealing [20], in the fact that EAs maintain a population of potential (or candidate) solutions to a problem, and not just one solution.

Generally, all EAs work as follows: a population of individuals is randomly initialized where each individual represents a potential solution to the problem at hand. The quality of each solution is evaluated using a *fitness function*. A selection process is applied during each iteration of an EA in order to form a new population. The selection process is biased toward the fitter individuals to increase their chances of being included in the new population. Individuals are altered using unary transformation (mutation) and higher-order transformation (crossover). This procedure is repeated until convergence is reached. The best solution found is expected to be a *near-optimum* solution.

* Corresponding author.
E-mail addresses: omran.m@gust.edu.kw (M.G.H. Omran), mehrdad.mahdavi@gmail.com (M. Mahdavi).

0096-3003/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2007.09.004

The main evolutionary computation algorithms are:*Genetic Programming* (GP) [12,13], *Evolutionary Programming* (EP) [3], *Evolutionary Strategies* (ES) [1], *Genetic Algorithms* (GA) [9] and *Differential Evolution* (DE) [19].

EAs have been successfully applied to a wide range of optimization problems, for example, image processing, pattern recognition, scheduling, engineering design, amongst others [9].

Recently, a new EA, called harmony search (HS), imitating the improvisation process of musicians was proposed by Greem et al. [6]. The HS has been successfully applied to Many optimization problems [11,7,15,8,16,5].

This paper proposes a new version of HS where concepts from swarm intelligence are used to enhance the performance of HS. The new version is called g*lobal-best harmony search* (GHS). The results of the experiments conducted are shown and compared with the versions of HS proposed by Lee and Geem [16] and a new variant of HS proposed by Mahdavi et al. [17]. Furthermore, the performance of the three approaches when applied to noisy problems is investigated. A scalability study is also conducted. The effect of the GHS parameters is studied. Finally, the three HS versions are used to tackle the Integer Programming problem.

The remainder of the paper is organized as follows: Section 2 provides an overview of HS. IHS is summarized in Section 3. The proposed approach is presented in Section 4. Results of the experiments are presented and discussed in Section 5. In Section 6, the three HS variants are applied to the Integer Programming problem Finally, Section 7 concludes the paper.

2. The harmony search algorithm

Harmony search (HS) [16] is a new meta-heuristic optimization method imitating the music improvisation process where musicians improvise their instruments' pitches searching for a perfect state of harmony. The HS works as follows:

Step 1: Initialize the problem and HS parameters: The optimization problem is defined as Minimize (or maximize) $f(\mathbf{x})$ such that $\mathrm{LB}_i \leqslant x_i \leqslant \mathrm{UB}_i$ Where $f(\mathbf{x})$ is the objective function, $\mathbf{x}$ is a candidate solution consisting of N decision variables (x_i), and LB_i and UB_i are the lower and upper bounds for each decision variable, respectively. In addition, the parameters of the HS are specified in this step. These parameters are the harmony memory size (HMS), harmony memory considering rate (HMCR), pitch adjusting rate (PAR) and the number of improvisations (NI).

Step 2: Initialize the harmony memory: The initial harmony memory is generated from a uniform distribution in the ranges $[\mathrm{LB}_i, \mathrm{UB}_i]$, where $1 \leqslant i \leqslant N$. This is done as follows: $x_i^j = \mathrm{LB}_i + r \times (\mathrm{UB}_i - \mathrm{LB}_i), j = 1, 2, \ldots, \mathrm{HMS}$ where $r \sim U(0, 1)$.

Step 3: Improvise a new harmony: Generating a new harmony is called *improvisation*. The new harmony vector, $x' = (x'_1, x'_2, \ldots, x'_N)$, is generated using the following rules: memory consideration, pitch adjustment and random selection. The procedure works as follows:

```
for each $i \in [1, N]$ do
  if $U(0,1) \leqslant$ HMCR then /*memory consideration*/
    begin
    $x'_i = x_i^j$,   where $j \sim U(1, \ldots, \mathrm{HMS})$.
    if $U(0,1) \leqslant$ PAR then /*pitch adjustment*/
    begin
      $x'_i = x'_i \pm r \times bw$, where $r \sim U(0,1)$ and bw is an arbitrary distance bandwidth.
    endif
  else/* random selection */
    $x'_i = \mathrm{LB}_i + r \times (\mathrm{UB}_i - \mathrm{LB}_i)$
  endif
done
```

Step 4: Update harmony memory: The generated harmony vector, $x' = (x'_1, x'_2, \ldots, x'_N)$, replaces the worst harmony in the HM, only if its fitness (measured in terms of the objective function) is better than that of the worst harmony.

Step 5: Check the stopping criterion: Terminate when the maximum number of improvisations is reached.

The HMCR and PAR parameters of the HS help the method in searching for globally and locally improved solutions, respectively. PAR and bw have a profound effect on the performance of the HS. Thus, fine tuning these two parameters is very important. From these two parameters, bw is more difficult to tune because it can take any value from $(0, \infty)$.

3. The improved harmony search algorithm

To address the shortcomings of the HS, Mahdavi et al. [17] proposed a new variant of the HS, called the improved harmony search (IHS). The IHS dynamically updates PAR according to the following equation,

$$\mathrm{PAR}(t) = \mathrm{PAR}_{\min} + \frac{(\mathrm{PAR}_{\max} - \mathrm{PAR}_{\min})}{NI} \times t \tag{1}$$

where $\mathrm{PAR}(t)$ is the pitch adjusting rate for generation t, $\mathrm{PAR}_{\min}$ is the minimum adjusting rate, $\mathrm{PAR}_{\max}$ is the maximum adjusting rate and t is the generation number.

In addition, bw is dynamically updated as follows:

$$\mathrm{bw}(t) = \mathrm{bw}_{\max} e^{\left(\frac{\ln\left(\frac{\mathrm{bw}_{\min}}{\mathrm{bw}_{\max}}\right)}{NI} \times t\right)} \tag{2}$$

where $\mathrm{bw}(t)$ is the bandwidth for generation t, $\mathrm{bw}_{\min}$ is the minimum bandwidth and $\mathrm{bw}_{\max}$ is the maximum bandwidth.

A major drawback of the IHS is that the user needs to specify the values for $\mathrm{bw}_{\min}$ and $\mathrm{bw}_{\max}$ which are difficult to guess and problem dependent.

4. The global-best harmony search

Inspired by the concept of swarm intelligence as proposed in Particle Swarm Optimization (PSO) [2,10], a new variation of HS is proposed in this paper. In a global best PSO system, a swarm of individuals (called *particles*) fly through the search space. Each particle represents a candidate solution to the optimization problem. The position of a particle is influenced by the best position visited by itself (i.e. its own experience) and the position of the best particle in the swarm (i.e. the experience of swarm).

The new approach, called global-best harmony search (GHS), modifies the pitch-adjustment step of the HS such that the new harmony can mimic the best harmony in the HM. Thus, replacing the bw parameter altogether and adding a social dimension to the HS. Intuitively, this modification allows the GHS to work efficiently on both continuous and discrete problems.

The GHS has exactly the same steps as the IHS with the exception that Step 3 is modified as follows:

```
for each $i \in [1, N]$ do
  if $U(0,1) \leqslant$ HMCR then /* memory consideration */
    begin
    $x'_i = x^j_i$, where $j \sim U(1, \ldots, HMS)$.
    if $U(0,1) \leqslant \mathrm{PAR}(t)$ then /* pitch adjustment */
    begin
      $x'_i = x^{\mathrm{best}}_k$, where best is the index of the best harmony in the HM and $k \sim U(1, N)$.
    endif
  else /* random selection */
    $x'_i = \mathrm{LB}_i + r \times (\mathrm{UB}_i - \mathrm{LB}_i)$
```

```
        endif
    done
```

4.1. Example

To further understand the HS, IHS and GHS algorithms, the *Rosenbrock* function (defined in Section 5) is considered to show the algorithms behavior in consecutive generations. The number of decision variables is set to 3 with possible values bounds between −600 and 600. Other parameters were set as in defined in Section 5. All of the algorithms start with the same initialization of HM. The solutions vectors in HM are sorted according to the values of objective function. The state of HM in different iterations for the algorithms HS, IHS, and GHS are shown in Tables 1–3 respectively. All of the algorithms improvised a near optimal solution but the result that is obtained by the GHS is better than the results of the other two algorithms.

Table 1
HM state in different iterations for the Rosenbrock function using the HS algorithm

Rank	x_1	x_2	x_3	$f(x)$
Initial HM				
1	206.909180	241.845703	−102.795410	29.141686
2	−99.938965	332.336426	−397.961426	70.088302
3	−381.262207	−392.358398	−87.634277	77.971790
4	−381.262207	−472.924805	−87.634277	95.243494
5	−470.910645	−54.345703	−430.700684	104.179736
Subsequent HM				
1*	206.909180	−54.337620	−87.634277	13.718178
2	206.909180	241.845703	−102.795410	29.141686
3	−99.938965	332.336426	−397.961426	70.088302
4	−381.262207	−392.358398	−87.634277	77.971790
5	−381.262207	−472.924805	−87.634277	95.243494
HM after 10 iterations				
1	206.909180	−54.337620	−87.634277	13.718178
2	206.909180	−54.341545	−102.795321	15.721512
3	206.909180	241.845703	−102.795410	29.141686
4	206.909180	316.369629	−87.634277	39.325780
5	−438.061523	−54.345703	−102.795436	52.221392
HM after 100 iterations				
1	−15.234375	−54.332356	56.637901	2.787318
2	109.016680	−54.332356	56.643102	5.635628
3	109.020996	−54.338577	56.643102	5.636281
4	109.020996	−54.332356	56.643102	5.636595
5	109.020996	−54.341623	56.637901	5.637259
HM after 500 iterations				
1	−15.243489	−18.162745	−27.026367	0.467774
2	−15.303073	−18.135317	50.484959	1.206920
3	−15.307083	−18.135317	50.488030	1.207263
4	−15.296501	−18.134462	50.484959	1.208392
5	−15.296501	−18.134526	50.484959	1.208400
HM after 5000 iterations				
1	−3.140023	0.000003	−5.433249	0.009857
2	−3.140023	0.000003	−5.433249	0.009857
3	−3.140023	0.000003	−5.433249	0.009857
4	−3.140023	0.000003	−5.433249	0.009857
5	−3.140023	0.000003	−5.433249	0.009857

* New good solution improvised in the first iteration.

Table 2
HM state in different iterations for the Rosenbrock function using the IHS algorithm

Rank	x_1	x_2	x_3	$f(x)$
Initial HM				
1	206.909180	241.845703	−102.795410	29.141686
2	−99.938965	332.336426	−397.961426	70.088302
3	−381.262207	−392.358398	−87.634277	77.971790
4	−381.262207	−472.924805	−87.634277	95.243494
5	−470.910645	−54.345703	−430.700684	104.179736
Subsequent HM				
1	206.909180	241.845703	−102.795410	29.141686
2*	206.909180	−472.924805	−0.439453	67.467811
3	−99.938965	332.336426	−397.961426	70.088302
4	−381.262207	−392.358398	−87.634277	77.971790
5	−470.910645	−54.345703	−430.700684	104.179736
HM after 10 iterations				
1	206.909180	−54.345703	−0.439453	11.786883
2	206.909180	241.845703	−0.439453	26.145702
3	206.909180	241.845703	−0.439453	26.145702
4	206.909180	241.845703	−0.439453	26.145702
5	206.909180	241.845703	−102.795410	29.141686
HM after 100 iterations				
1	11.682129	−54.345703	−0.439453	1.314937
2	11.682129	−54.345703	−0.439453	1.314937
3	11.682129	−54.345703	−0.439453	1.314937
4	11.682129	−54.345703	−0.439453	1.314937
5	11.682129	−54.345703	−0.439453	1.314937
HM after 500 iterations				
1	11.682129	18.127369	−0.439416	0.522051
2	11.682129	18.127369	−0.439429	0.522052
3	11.682129	18.127380	−0.439416	0.522052
4	11.682129	18.127380	−0.439416	0.522052
5	11.682129	18.127380	−0.439416	0.522052
HM after 5000 iterations				
1	0.000184	0.090208	0.087013	0.003297
2	0.000323	0.090182	0.087113	0.003298
3	0.000270	0.090184	0.087107	0.003298
4	0.000361	0.090143	0.087155	0.003298
5	0.000323	0.090166	0.087176	0.003300

* New good solution improvised in the first iteration.

5. Experimental results

This section compares the performance of the global-best harmony search (GHS) with that of the harmony search (HS) and the improved harmony search (IHS) algorithms. For the GHS, HMS = 5, HMCR = 0.9, $\text{PAR}_{\min} = 0.01$ and $\text{PAR}_{\max} = 0.99$. For the HS algorithm, HMS = 5, HMCR = 0.9, PAR = 0.3 and bw = 0.01 (the values of the last three parameters were suggested by Dr. Zong Geem in a private communication). For the IHS algorithm, HMS = 5, HMCR = 0.9, $\text{PAR}_{\min} = 0.01$, $\text{PAR}_{\max} = 0.99$, $\text{bw}_{\min} = 0.0001$ and $\text{bw}_{\max} = 1/(20 \times (\text{UB} - \text{LB}))$. All functions were implemented in 30 dimensions except for the two-dimensional Camel-Back function. Unless otherwise specified, these values were used as defaults for all experiments which use static control parameters. The initial harmony memory was generated from a uniform distribution in the ranges specified below.

The following functions have been used to compare the performance of the different methods. These benchmark functions provide a balance of unimodal and multimodal functions.

Table 3
HM state in different iterations for the Rosenbrock function using the GHS

Rank	x_1	x_2	x_3	$f(x)$
Initial HM				
1	206.909180	241.845703	−102.795410	29.141686
2	−99.938965	332.336426	−397.961426	70.088302
3	−381.262207	−392.358398	−87.634277	77.971790
4	−381.262207	−472.924805	−87.634277	95.243494
5	−470.910645	−54.345703	−430.700684	104.179736
Subsequent HM				
1*	191.381836	−54.345703	−87.634277	13.497674
2	206.909180	241.845703	−102.795410	29.141686
3	−99.938965	332.336426	−397.961426	70.088302
4	−381.262207	−392.358398	−87.634277	77.971790
5	−470.910645	−54.345703	−430.700684	104.179736
HM after 10 iterations				
1	206.909180	−54.345703	−87.634277	13.721652
2	206.909180	−54.345703	−87.634277	13.721652
3	206.909180	136.926270	−102.795410	18.311658
4	206.909180	136.926270	−87.634277	19.032912
5	206.909180	136.926270	−87.634277	19.032912
HM after 100 iterations				
1	−18.859863	24.719238	−45.629883	1.692257
2	−18.859863	45.593262	−45.629883	1.890251
3	−18.859863	45.593262	−45.629883	1.890251
4	−18.859863	45.593262	−45.629883	1.890251
5	−18.859863	45.593262	−45.629883	1.890251
HM after 500 iterations				
1	−18.859863	1.171875	22.082520	0.546616
2	−18.859863	1.171875	22.082520	0.546616
3	−18.859863	1.171875	22.082520	0.546616
4	−18.859863	1.171875	22.082520	0.546616
5	−18.859863	1.171875	22.082520	0.546616
HM after 5000 iterations				
1	0.036621	−0.073242	−0.036621	0.002235
2	0.036621	−0.073242	−0.036621	0.002235
3	0.036621	−0.073242	−0.036621	0.002235
4	0.036621	−0.073242	−0.036621	0.002235
5	0.036621	−0.073242	−0.036621	0.002235

* New good solution improvised in the first iteration.

For each of these functions, the goal is to find the global minimizer, formally defined as
Given f: $\Re^{N_d} \to \Re$
find $x^* \in \Re^{N_d}$ such that $f(x^*) \leqslant f(x), \forall x \in \Re^{N_d}$
The following functions were used:

A. *Sphere* function, defined as

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

where $x^* = \mathbf{0}$ and $f(x^*) = 0$ for $-100 \leqslant x_i \leqslant 100$.

B. *Schwefel*'s Problem 2.22 [21], defined as

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

where $x^*=\mathbf{0}$ and $f(x^*)=0$ for $-10\leqslant x_i\leqslant 10$.

C. *Step* function, defined as

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

where $x^*=\mathbf{0}$ and $f(x^*)=0$ for $-100\leqslant x_i\leqslant 100$.

D. *Rosenbrock* function, defined as

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

where $x^*=(1,1,\ldots,1)$ and $f(x^*)=0$ for $-30\leqslant x_i\leqslant 30$.

E. *Rotated hyper-ellipsoid* function, defined as

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

where $x^*=\mathbf{0}$ and $f(x^*)=0$ for $-100\leqslant x_i\leqslant 100$.

F. *Generalized Swefel*'s Problem 2.26 [21], defined as

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

where $x^*=(420.9687,\ldots,420.9687)$ and $f(x^*)=-12569.5$ for $-500\leqslant x_i\leqslant 500$.

G. *Rastrigin* function, defined as

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

where $x^*=\mathbf{0}$ and $f(x^*)=0$ for $-5.12\leqslant x_i\leqslant 5.12$.

H. *Ackle*y's function, defined as

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

where $x^*=\mathbf{0}$ and $f(x^*)=0$ for $-32\leqslant x_i\leqslant 32$.

I. *Griewank* function, defined as

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

where $x^*=\mathbf{0}$ and $f(x^*)=0$ for $-600\leqslant x_i\leqslant 600$.

J. *Six-Hump Camel-Back* function, defined as

$$f(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4,$$

where $x^*=(-0.08983,0.7126),(-0.08983,0.7126)$ and $f(x^*)=-1.0316285$ for $-5\leqslant x_i\leqslant 5$.

Sphere, Schwefel's Problem 2.22, Rosenbrock and rotated hyper-ellipsoid are unimodal, while the Step function is a discontinuous unimodal function. Schwefel's Problem 2.26, Rastrigin, Ackley and Griewank are difficult multimodal functions where the number of local optima increases exponentially with the problem dimension. The Camel-Back function is a low-dimensional function with only a few local optima.

The results reported in this section are averages and standard deviations over 30 simulations. Each simulation was allowed to run for 50,000 evaluations of the objective function. The statistically significant best solutions have been shown in bold (using the z-test with $\alpha = 0.05$).

Table 4 summarizes the results obtained by applying the three approaches to the benchmark functions. The results show that the GHS outperformed HS and IHS in all the functions except for the Rotated-hyper ellipsoid function (where there was no statistical significant difference between the three methods) and the Camel-Back function (where HS and IHS performed better than the GHS). For the Camel-Back function, the GHS performed relatively worse than HS and IHS because the dimension of the problem is too low (i.e. 2). In case of low-dimensionality (i.e. small number of decision variables), the GHS' pitch-adjustment step suffers. However, when the dimensionality increases, the GHS takes the lead and outperforms the other methods.

(1) Effect of noise on performance
This section investigates the effect of noise on the performance of the three approaches. The noisy versions of the benchmark functions are defined as

$$f_{\text{Noisy}}(x) = f(x) + N(0,1),$$

where $N(0,1)$ represents the normal distribution with zero mean and standard deviation of one.
Table 5 summarizes the results obtained for the noisy problems for the benchmark functions. In general, noise adversely affects the performance of all the examined strategies. Table 5, however, shows that the GHS retained its position as the best performer when applied to the benchmark functions even in the presence of noise. The exceptions are the noisy Rotated hyper-ellipsoid function where the HS performed better than the GHS and the noisy Camel-Back function where HS and IHS outperformed the GHS.

Table 4
Mean and standard deviation (±SD) of the benchmark function optimization results

	HS	IHS	GHS
Sphere	0.000187 (0.000032)	0.000712 (0.000644)	**0.000010 (0.000022)**
Schwefel Problem 2.22	0.171524 (0.072851)	1.097325 (0.181253)	**0.072815 (0.114464)**
Rosenbrock	340.297100 (266.691353)	624.323216 (559.847363)	**49.669203 (59.161192)**
Step	4.233333 (3.029668)	3.333333 (2.195956)	**0(0)**
Rotated hyper-ellipsoid	4297.816457 (1362.148438)	4313.653320 (1062.106222)	5146.176259 (6348.792556)
Schwefel Problem 2.26	−12539.237786 (11.960017)	−12534.968625 (10.400177)	**−12569.458343 (0.050361)**
Rastrigin	1.390625 (0.824244)	3.499144 (1.182907)	**0.008629 (0.015277)**
Ackley	1.130004 (0.407044)	1.893394 (0.314610)	**0.020909 (0.021686)**
Griewank	1.119266 (0.041207)	1.120992 (0.040887)	**0.102407 (0.175640)**
Camel-Back	**−1.031628 (0.000000)**	**−1.031628 (0.000000)**	−1.031600 (0.000018)

Table 5
Mean and standard deviation (±SD) of the noisy benchmark function optimization results

	HS	IHS	GHS
Sphere	0.071377 (0.169067)	0.099251 (0.222471)	**0.001031** **(0.001213)**
Schwefel Problem 2.22	6.654909 (1.652772)	6.723778 (1.315942)	**0.043880** **(0.183376)**
Rosenbrock	534.883721 (643.935556)	504.485368 (252.597506)	**65.166415** **(64.582174)**
Step	10.054982 (4.526308)	7.452871 (3.914539)	**0.000989** **(0.000911)**
Rotated hyper-ellipsoid	**4596.546169** **(1282.763999)**	4412.324728 (1511.752121)	5574.541228 (6247.474005)
Schwefel Problem 2.26	−12532.687992 (10.749470)	−12530.442804 (13.111425)	−**12567.97599** **(6.541833)**
Rastrigin	11.530824 (3.719364)	14.840561 (2.901781)	**0.019196** **(0.055387)**
Ackley	16.048601 (1.418158)	16.068966 (0.905531)	**7.970873** **(7.450370)**
Griewank	3.549717 (1.135888)	3.054119 (1.518994)	**0.012079** **(0.019128)**
Camel-Back	−**4.759560** **(0.532310)**	−**4.867695** **(0.270855)**	−5.039987 (0.380634)

(2) Scalability study
When the dimension of the functions increases from 30 to 100, the performance of the different methods degraded as shown in Table 6. The results show that the GHS is still the best performer. In general, the IHS performed comparably to the HS.

Table 6
Mean and standard deviation (±SD) of the benchmark function optimization results ($N = 100$)

	HS	IHS	GHS
Sphere	8.683062 (0.775134)	8.840449 (0.762496)	**2.230721** **(0.565271)**
Schwefel Problem 2.22	82.926284 (6.717904)	82.548978 (6.341707)	**19.020813** **(5.093733)**
Rosenbrock	16675172.184717 (3182464.488466)	17277654.059718 (2945544.275052)	**2598652.617273** **(915937.797217)**
Step	20280.200000 (2003.829956)	20827.733333 (2175.284501)	**5219.933333** **(1134.876027)**
Rotated hyper-ellipsoid	**215052.904398** **(28276.375538)**	**213812.584732** **(28305.249583)**	321780.353575 (39589.041160)
Schwefel Problem 2.26	−**33937.364505** **(572.390489)**	−33596.899217 (731.191869)	−40627.345524 (395.457330)
Rastrigin	343.497796 (27.245380)	343.232044 (25.149464)	**80.657677** **(30.368471)**
Ackley	13.857189 (0.284945)	13.801383 (0.530388)	**8.767846** **(0.880066)**
Griewank	195.592577 (24.808359)	204.291518 (19.157177)	**54.252289** **(18.600195)**

(3) Effect of HMCR, HMS and PAR

In this subsection, the effect of HMCR, HMS and PAR on the performance of the GHS is investigated. Table 7 summarizes the results obtained using different values for HMCR. The results show that increasing the HMCR value improves the performance of the GHS for all the functions except for the Camel-Back where the opposite is true. Using a small value for HMCR increases the diversity and, hence, prevents the GHS from convergence (i.e. it results in (inefficient) random search). Thus, it is generally better to use a large value for the HMCR (i.e. $\geqslant 0.9$). However, for problem with very low dimensionality (e.g. Camel-Back), using a small value of HMCR is beneficial since it improves the exploration capability of the GHS.

Table 8 summarizes the results obtained using different values for HMS. Each simulation was allowed to run for 50,000 evaluations of the objective function. The results show that no single choice is superior to the others indicating an independence to the value of the HMS. In general, using a small HM seems to be a good and logical choice with the added advantage of reducing space requirements. Actually, since HM resembles the short-term memory of a musician and since the short-term memory of the human is known to be small, it is logical to use a small HM.

Until now, we have used Eq. (1) to dynamically adjust the GHS' PAR. Herein, the effect of using constant values for PAR on the performance of the GHS is investigated. Table 9 shows the results of using different values for PAR. In general, the results show that no single choice is superior to the others. However, it seams that using a relatively small value of PAR (*i.e.* $\leqslant 0.5$) improves the performance of the GHS. In addition, the GHS using a small constant value for PAR generally performed better than the GHS using Eq. (1).

6. The integer programming problem

Many real-world applications (e.g. production scheduling, resource allocation, VLSI circuit design, etc.) require the variables to be integers. These problems are called Integer Programming problems. Optimization

Table 7
The effect of HMCR

	HMCR = 0.5	HMCR = 0.7	HMCR = 0.9	HMCR = 0.95
Sphere	5.295242 (0.713208)	0.745164 (0.265503)	0.000010 (0.000022)	0.000001 (0.000001)
Schwefel Problem 2.22	36.952315 (3.935531)	9.603271 (2.859507)	0.072815 (0.114464)	0.017029 (0.020551)
Rosenbrock	13090605.360898 (4323029.791459)	376181.410765 (296509.690094)	49.669203 (59.161192)	47.293368 (100.109620)
Step	12222.833333 (1771.547235)	1761.633333 (705.998654)	0(0)	0(0)
Rotated hyper-ellipsoid	30128.835193 (4745.705757)	17933.990434 (5561.440021)	5146.176259 (6348.792556)	3632.546963 (5014.897019)
Schwefel Problem 2.26	−8926.586935 (375.856862)	−11970.385304 (226.448517)	−12569.458343 (0.050361)	−12569.477436 (0.017952)
Rastrigin	153.349949 (16.322363)	41.488430 (12.259544)	0.008629 (0.015277)	0.001606 (0.002871)
Ackley	16.262370 (0.632526)	8.976493 (1.402425)	0.020909 (0.021686)	0.010069 (0.006675)
Griewank	117.145910 (16.780988)	16.522998 (6.666186)	0.136015 (0.228799)	0.011433 (0.027773)
Camel-Back	−1.031625 (0.000003)	−1.031622 (0.000014)	−1.031600 (0.000018)	−1.031608 (0.000015)

Table 8
The effect of HMS

	HMS = 5	HMS = 10	HMS = 20	HMS = 50
Sphere	0.000010 (0.000022)	0.000014 (0.000024)	0.000018 (0.000047)	0.000019 (0.000030)
Schwefel Problem 2.22	0.072815 (0.114464)	0.055379 (0.061338)	0.051270 (0.044451)	0.034180 (0.024898)
Rosenbrock	49.669203 (59.161192)	71.871738 (86.611858)	49.010352 (51.127082)	65.821680 (64.172156)
Step	0(0)	0(0)	0(0)	0(0)
Rotated hyper-ellipsoid	5146.176259 (6348.792556)	2840.550927 (3485.583318)	4211.308058 (5741.749864)	2279.716176 (2969.907898)
Schwefel Problem 2.26	−12569.458343 (0.050361)	−12569.452670 (0.072130)	−12569.40515 (0.143585)	−12569.454553 (0.042580)
Rastrigin	0.008629 (0.015277)	0.004154 (0.006119)	0.011628 (0.018224)	0.015724 (0.029582)
Ackley	0.020909 (0.021686)	0.036708 (0.049579)	0.030355 (0.023022)	0.042470 (0.029947)
Griewank	0.136015 (0.228799)	0.113143 (0.204220)	0.100169 (0.164517)	0.177135 (0.205283)
Camel-Back	−1.031600 (0.000018)	−1.031618 (0.000009)	−1.03147 (0.000160)	−1.031567 (0.000054)

Table 9
The effect of PAR

	PAR = 0.1	PAR = 0.3	PAR = 0.5	PAR = 0.7	PAR = 0.9
Sphere	0.000004 (0.000012)	0.000005 (0.000016)	0.000004 (0.000006)	0.000011 (0.000017)	0.000010 (0.000015)
Schwefel Problem 2.22	0.056458 (0.043551)	0.047404 (0.032014)	0.046387 (0.029801)	0.046997 (0.039182)	0.068705 (0.056855)
Rosenbrock	37.375855 (36.379778)	34.694156 (40.258490)	48.580795 (56.871491)	41.744516 (49.501602)	38.110374 (37.536347)
Step	0(0)	0(0)	0(0)	0(0)	0(0)
Rotated hyper-ellipsoid	2933.680647 (1677.980140)	2199.433261 (2974.500178)	2662.057366 (3615.948333)	2159.448105 (2360.188117)	4279.831064 (6700.051680)
Schwefel Problem 2.26	−12569.46259 (0.024735)	−12569.42179 (0.108268)	−12569.383768 (0.197043)	−12569.458178 (0.059009)	−12569.42452 (0.235659)
Rastrigin	0.012112 (0.020371)	0.013436 (0.018171)	0.004066 (0.005308)	0.015272 (0.030422)	0.008142 (0.013092)
Ackley	0.023014 (0.021537)	0.022487 (0.014631)	0.024789 (0.017931)	0.018062 (0.018143)	0.027083 (0.024494)
Griewank	0.051157 (0.093015)	0.091768 (0.197166)	0.072257 (0.141671)	0.202081 (0.267918)	0.116950 (0.230767)
Camel-Back	−1.031554 (0.000072)	−1.031611 (0.000012)	−1.031578 (0.000040)	−1.031568 (0.000062)	−1.029316 (0.001898)

Table 10
Mean and standard deviation (±SD) of the integer programming problems results

Function	Method	Mean (SD)
$F_1(N = 5)$	HS	0(0)
	IHS	0(0)
	GHS	0(0)
$F_1(N = 15)$	HS	0(0)
	IHS	0(0)
	GHS	0(0)
$F_1(N = 30)$	HS	0(0)
	IHS	0(0)
	GHS	0(0)
F_2	HS	0(0)
	IHS	0(0)
	GHS	0(0)
F_3	HS	0(0)
	IHS	0(0)
	GHS	0(0)
F_4	HS	−4.866667(0.991071)
	IHS	−4.066667(0.359011)
	GHS	−5(1)
F_5	HS	−3833.12(0)
	IHS	−3833.12(0)
	GHS	−3833.12(0)
$F_6(N = 5)$	HS	0(0)
	IHS	0(0)
	GHS	0(0)

methods developed for real search spaces can be used to solve Integer Programming problems by rounding off the real optimum values to the nearest integers [14].

The unconstrained Integer Programming problem can be defined as

$$\min f(x), x \in S \subseteq Z^{N_d},$$

where Z^{N_d} is an N-dimensional discrete space of integers, and $\boldsymbol{S}$ represents a feasible region that is not necessarily a bounded set. Integer Programming problems encompass both maximization and minimization problems. Any maximization problem can be converted into a minimization problem and *vice versa*. The problems tackled in this section are minimization problems. Therefore, the remainder of the discussion focuses on minimization problems.

Six commonly used Integer programming benchmark problems [14] were chosen to investigate the performance of the HS variants.

Test Problem 1:

$$F_1(x) = \sum_{i=1}^{N_d} |x_i|,$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(x^*) = 0$. This problem was considered in dimensions 5, 15 and 30.

Test Problem 2:

$$F_2(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2,$$

where $\boldsymbol{x}^* = (1, 1)^{\mathrm{T}}$ and $F_2(x^*) = 0$.

Test Problem 3:

$$F_3(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4,$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $F_3(x^*) = 0$.

Test Problem 4:

$$F_4(x) = 2x_1^2 + 3x_2^2 + 4x_1x_2 - 6x_1 - 3x_2,$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $F_4(x^*) = 0$.

Test Problem 5:

$$F_5(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2,$$

where $\boldsymbol{x}^* = (0,1)^{\mathrm{T}}$ and $F_5(x^*) = -3833.12$.

Test Problem 6:

$$F_6(x) = x^{\mathrm{T}}x,$$

where $\boldsymbol{x}^* = \mathbf{0}$ and $f(\boldsymbol{x}^*) = 0$. This problem was considered in dimension 5 as in Laskari et al. [14].

For all the above problems, $x_i \in [-100, 100] \subseteq Z$.

The three versions of HS were applied to the above test problems and the results are shown in Table 10. All the parameters were set as in Section 5. The results show that the three approaches performed comparably. The three approaches found the global optimum solution for all the benchmark problems except for F_4. However, when HMCR was set to 0.1, the three methods found the global optimum.

7. Conclusions

This paper proposed a new version of harmony search. The approach modifies the pitch-adjustment step of the HS such that a new harmony is affected by the best harmony in the harmony memory. This modification alleviates the problem of tuning the HS's bw parameter which is difficult to specify *a priori*. In addition, the new modification allows the GHS to work efficiently on both continuous and discrete problems. The approach was tested on ten benchmark functions where it generally outperformed the other approaches. This paper investigated the effect of noise on the performance of the HS variations. Empirical results show that, in general, the GHS provided the best results when applied to high-dimensional problems. Moreover, the effect of the two parameters of GHS (i.e. HMCR and HMS) was investigated. The results show that using a large value for HMCR (e.g. 0.95) generally improves the performance of the GHS except for the case of problems with very low dimensionality where a small value of HMCR is recommended. In addition, using a small value for HMS seems to be a good choice. Replacing the dynamically adjusted PAR with a constant value was also investigated. The experiments show that using a relatively small constant value for PAR seems to improve the performance of the GHS. These observations confirm the observations of Geem [4]. Finally, the performance of HS variants to solve Integer Programming problems was studied. Experimental results using six commonly used benchmark problems show that the three approaches performed comparably.

Acknowledgements

We are grateful to Dr. Zong Geem and Dr. Kang Lee for their valuable help and suggestions.

References

[1] T. Bäck, F. Hoffmeister, H. Schwefel, A Survey of evolution strategies. In: Proceedings of the Fourth International Conference on Genetic Algorithms and their Applications, vols. 2–9, 1991.

[2] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micromachine and Human Science, pp. 39–43, 1995.
[3] L. Fogel, Evolutionary programming in perspective: the top-down view, in: J.M. Zurada, R. MarksII, C. Robinson (Eds.), Computational Intelligence: Imitating Life, IEEE Press, Piscataway, NJ, USA, 1994.
[4] Z. Geem, Optimal design of water distribution networks using harmony search. PhD Thesis, Korea University, Seoul, Korea, 2000.
[5] Z. Geem, Optimal cost design of water distribution networks using harmony search, Engineering Optimization 38 (3) (2006) 259–280.
[6] Z. Geem, J. Kim, G. Loganathan, A new heuristic optimization algorithm: harmony search, Simulation 76 (2) (2001) 60–68.
[7] Z. Geem, J. Kim, G. Loganathan, Harmony search optimization: application to pipe network design, International Journal of Model Simulation 22 (2) (2002) 125–133.
[8] Z. Geem, C. Tseng, Y. Park, Harmony search for generalized orienteering problem: best touring in China, in: Springer Lecture Notes in Computer Science 3412 (2005) 741–750.
[9] D. Goldberg, Genetic Algorithms in Search, Optimization and machine learning, Addison-Wesley, 1989.
[10] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of the IEEE International Joint Conference on Neural Networks, IEEE Press, 1995. pp. 1942–1948.
[11] J. Kim, Z. Geem, E. Kim, Parameter estimation of the nonlinear Muskingum model using harmony search, Journal of American Water Resource Association 37 (5) (2001) 1131–1138.
[12] J. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, Cambridge, MA, 1992.
[13] J. Koza, R. Poli, Genetic programming, in: E. Burke, G. Kendall (Eds.), Introductory Tutorials in Optimization, Decision Support and Search Methodology, Kluwer Press, 2005, pp. 127–164, Chapter 5.
[14] E. Laskari, K. Parsopoulos, M. Vrahatis, Particle swarm optimization for integer programming, in: Proceedings of the 2002 Congress on Evolutionary Computation (2) (2002) 1582–1587.
[15] K. Lee, Z. Geem, A new structural optimization method based on the harmony search algorithm, Computer and Structures 82 (9–10) (2004) 781–798.
[16] K. Lee, Z. Geem, A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice, Computer Methods in Applied Mechanics and Engineering 194 (2005) (2005) 3902–3933.
[17] M. Mahdavi, M. Fesanghary, E. Damangir, An improved harmony search algorithm for solving optimization problems, Applied Mathematics and Computation 188 (2007) (2007) 1567–1579.
[18] Z. Michalewicz, D. Fogel, How to Solve It: Modern heuristics, Springer-Verlag, Berlin, 2000.
[19] R. Storn, K. Price, Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces, Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA, 1995.
[20] P. Van Laarhoven, E. Aarts, Simulated annealing: theory and applications, Kluwer Academic Publishers, 1987.
[21] X. Yao, Y. Liu, G. Lin, Evolutionary programming made faster, IEEE Transactions on Evolutionary Computation 3 (2) (1999) 82–102.