International Journal of Hybrid Information Technology
Vol. 4 No. 4, October, 2011

A Harmony Search with Adaptive Pitch Adjustment for Continuous Optimization

Chukiat Worasucheep

Applied Computer Science, Department of Mathematics, Faculty of Science, King Mongkut's University of Technology Thonburi, Bangkok 10140, Thailand
chukiat.wor@kmutt.ac.th

Abstract

The Harmony Search is a relatively new meta-heuristic algorithm for continuous optimization, in which its concept imitates the process of music improvisation. This paper proposes an improved harmony search algorithm called Harmony Search with Adaptive Pitch Adjustment (HSAPA). The adaptive pitch adjustment scheme during a search period is inspired from the velocity clamping in particle swarm optimizations. The pitch adjustment rate is also dynamically adapted. The proposed algorithm is evaluated using thirteen well-known benchmark functions of various characteristics. The experiment is performed using both 30 and 100 dimensions to investigate its performance at both medium and high number of decision variables. The experimental results demonstrate that HSAPA is reliably superior to all prior well-known harmony search variants and a recent widely-accepted variant of Differential Evolution algorithm. Furthermore, performance of HSAPA is shown not to be sensitive to its new parameter, which regulates the size of pitch adjustment.

Keywords: *Harmony Search, Continuous optimization*

1. Introduction

The Harmony Search (HS) is a meta-heuristic algorithm recently developed by Geem *et al.* for continuous optimization [1]. Imitating the process of music improvisation, HS was conceptualized using the musical process of searching for a perfect state of harmony. The harmony in music is analogous to the optimization solution vector. And, the musicians' improvisations of their instruments' pitches are analogous to the local and global search schemes in optimization techniques. Due to its simple structure and effectiveness, HS has been successfully applied to numerous real-world optimization problems [2], particularly in constrained engineering optimization domain [3][4]. However, like most other meta-heuristics, its capabilities are sensitive to parameter setting. Some research works are recently undertaken to improve the performance of HS by tuning its parameters or blending it with other powerful optimization techniques. A few improved variants of HS have been proposed, but some experiments are less rigorous or obtained from low-dimensional problems.

In this work, a new adaptive pitch adjustment scheme is proposed into the harmony improvisation of HS. The performance of the proposed HS is evaluated using thirteen widely used benchmark functions for continuous optimization in 30 and 100 dimensions. The remainder of this article is organized as follows. In Section 2, a brief introduction of harmony search variants will be reviewed. Section 3 describes the proposed HS variant that will be evaluated using benchmark functions in Section 4. Finally, Section 5 concludes this work.

2. Reviews of Harmony Search

2,1 Original Harmony Search

In 2001, Geem *et al.* developed and proposed Harmony Search (HS) meta-heuristic algorithm that was conceptualized using the musical process of searching for a perfect state of harmony [1]. The original HS algorithm consists of a harmony memory (HM) with size HMS. HM stores the candidate solution vectors (of D decision variables), all of which are initialized randomly within the search space as follows.

$$HM_i(d) = LB(d) + (UB(d) - LB(d)) \times rand() \quad \text{for } i = 0 \text{ to HMS} - 1 \text{ and } d = 0 \text{ to } D$$

where LB(d) and UB(d) are the lower and upper bounds in the search space of decision variable d. *Rand*() is a random function returning a number between 0 and 1 with uniform distribution.

Then an improvisation process is repeated as shown in Figure 1 to adjust the pitch in order to search for the perfect harmony, or the optimal solutions. HS has three parameters controlling the improvisation as follows: the harmony consideration rate (HMCR), pitch adjustment rate (PAR), and the pitch adjustment bandwidth (bw). In each improvisation or each iteration, if *rand*() is less than HMCR, then a trial vector is generated with memory consideration, otherwise the trial vector is obtained by a random selection. The trial vector from memory consideration is further adjusted with a small pitch adjustment bandwidth (bw) if another *rand*() returns a number less than PAR. If the obtained trial vector is better than the worst harmony in the HM, replace the worst harmony with the trial vector. This procedure is repeated until a stopping criterion is satisfied.

```
Set parameters: HMCR, HMS, PAR
Initialize HM = {HM_0, HM_1, ..., HM_HMS-1}
i ← 0
while no stopping criterion is met do
    for d = 0 to D – 1 do
        if rand() < HMCR then          // memory consideration
            trial(d) = HM_R(d)         where R = IntRand(0, HMS – 1)
            if rand() < PAR then       // pitch adjustment
                trial(d) = trial(d) ± rand() × bw
            end if
        else                           // random selection
            trial(d) = LB(d) + (UB(d) – LB(d)) × rand()
        end if
    end for
    if fitness(trial) is better than finess(HM_worst) then
        replace HM_worst with trial
    end if
    i = i + 1
end while
```

Figure 1. The Original Harmony Search Algorithm.

HMCR, ranging from 0 to 1, controls the balance of exploration and exploitation. A low HMCR makes the HS behave toward a random search, whereas a high HMCR tends to choose a vector from HM. Generally, the value of HMCR is preset as a constant close to 1.0 to perform more improvisation. Pitch adjustment rate (PAR) determines the chance that the chosen harmony is further adjusted with a small distance bandwidth (*bw*). This adjustment is analogous to the local search with a small step size.

2.1 Improved HS Variants

A few variants of HS algorithms have been proposed since its introduction. Mahdavi *et al*. [5] proposed the Improved Harmony Search (IHS) that dynamically increases PAR and decreases *bw*. The idea of decreasing *bw* is analogous to decreasing the inertia weight in Particle swarm optimization or the learning rate in neural networks. But, as pointed out by Wang and Huang [6], the decreasing PAR is contradictory to a general strategy of continuously increasing local search capability.

Omran & Mahdavi [7] recently enhanced performance of HS by using concept of the 'global best' from Particle swarm optimization [8][9]. In their proposed Global Harmony Search (GHS), the pitch adjustment with *bw* is replaced with the global best harmony, thus beneficially reducing one parameter. However, the experimentation and results have some mistakes, and its reliability is questionable.

Wang & Huang [6] proposed the Self-adaptive Harmony Search, to be called SHS hereafter, with automatically adjusted parameters. Their experiments and results are promising as well as trustworthy with full-factorial design, asymmetric initialization and tested at both 30- and 100-dimensions. Unfortunately, the set of test functions is too small, and the intrinsic strength of algorithm is partially arisen from their low-discrepancy sequences for initialization. More recently, Pan *et al.* [10] proposed an improved global harmony search algorithm whose improvisation scheme attempts to retain good information in the current global best solution. They applied some learning method to self-adapt parameters HMCR and PAR. Distance bandwidth (*bw*) is also dynamically adjusted to favor exploration in the early stages and exploitation during the final stages of the search process.

3. The Proposed Harmony Search

This paper proposes an improved HS variant called *Harmony Search with Adaptive Pitch Adjustment*, or HSAPA for short. A large HMCR value favors of harmony improvisation, thus increasing the convergence rate of the algorithm in most cases, while a small value of HMCR increase the diversity of the harmony memory. The results from my preliminary tests found that a higher HMCR even provides a better performance in general. Therefore, the proposed HSAPA uses HMCR = 0.995.

Parameter PAR is very important in the fine-tuning of optimized solution vectors. Their constant values in traditional HS algorithms prolong the convergence due to the imbalance of global and local search capabilities during the optimization period. Furthermore, simultaneous dynamic HMCR and dynamic PAR can cause a twist of global and local search. This rational confirms the prior discussion of using a constant HMCR and a dynamically decreasing PAR. Based on results of prior experiments, the linearly decrease of PAR from 1.0 to 0.0 during the optimization run outperforms other configurations in most cases.

Parameter *bw* is also very important. The original HS [1] sets it at a constant *bw* = 0.01. On the other hand, IHS [5] uses an exponentially decreasing value, but the beginning and

ending values are constants. GHS [7] replaces this feature with the global best harmony as mentioned in previous section. SHS [6] also replaces *bw* as follows. The new harmony is updated according to the maximal and minimal values in the HM at current iteration from either eq. (1) or eq. (2) with an equal chance.

$$trial(d) + \left[max(HM(d)) - trial(d)\right] \times rand() \quad (1)$$

$$trial(d) - \left[trial(d) - min(HM(d))\right] \times rand() \quad (2)$$

Subscript d denotes the dimension; $max(HM(d))$ and $min(HM(d))$ are the largest and the smallest values of the dth variable in the HM. By this way, the new harmony is guaranteed to stay within search boundary since the updated distances are kept within $min(HM(d))$ and $max(HM(d))$. However, this method eliminates the benefits of the exploration of search space outside current boundary. The equal chance of using eq. (1) or (2) results in a nonuniform distribution of position of the new harmony.

In this work, the pitch adjustment approach is simple and partially inspired from the velocity clamping in classical variants of Particle swarm optimization [8][9]. The pitch adjustment in this work is as follows.

$$trial(d) \pm \lambda \times range(d) \times rand(), \texttt{ where } range(d) = max(HM(d)) - min(HM(d)) \quad (3)$$

$$\texttt{if } trial(d) < LB(d) \texttt{ then } trial(d) = LB(d) \quad (4)$$

$$\texttt{if } trial(d) > UB(d) \texttt{ then } trial(d) = UB(d) \quad (5)$$

```
Set parameters: HMCR, HMS, λ
Initialize HM = {HM_0, HM_1, ..., HM_HMS-1}
i ← 0
while no stopping criterion is met do
    PAR ← 1.0 – i / maxiteration
    for d = 0 to D – 1 do
        calculate max(HM(d)), min(HM(d)), and range(d)
        if rand() < HMCR then          // memory consideration
            trial(d) = HM_R(d)         where R = IntRand(0, HMS – 1)
            if rand() < PAR then       // pitch adjustment
                Pitch adjustment eq. (3)-(5)
            end if
        else                           // random selection
            trial(d) = LB(d) + (UB(d) – LB(d)) × rand()
        end if
    end for
    if fitness(trial) is better than finess(HM_worst) then
        replace HM_worst with trial
    end if
    i = i + 1
end while
```

Figure 2. The Proposed HSAPA Algorithm.

In each iteration, the parameter *bw* in original HS is replaced with $\lambda \times range_d$, making the pitch adjustment in eq. (3) dynamic. The stochastic oscillation in each dimension is restricted to the current range of harmony position and is regulated by the parameter λ. If the value of λ

is too high, the pitch may move erratically going beyond a good solution. But if it is too small, then the target may not be reached. Equations (4) and (5) are necessary to keep the harmony within the search boundary. The experimental results of using 13 widely accepted benchmark functions in Section 4 demonstrate that the performance is not sensitive to λ and its suggested value is between 0.4 and 0.5. Figure 2 outlines the proposed HSAPA algorithm as described above, where *maxiteration* equals to the maximum number of iterations allowed to improvise or optimize. HMS is set at 50 instead of just 5 in most former variants, as suggested from the full-factorial tests in SHS [6].

4. Performance Evaluation

This section describes the experimentation to evaluate the performance of proposed HSAPA and analyzes the results. Thirteen nonlinear optimization functions widely used in literature [11][12] are selected to evaluate the HSAPA. Table 1 illustrates descriptions and the search ranges, for each dimension, of all the 13 benchmark functions. They are minimization functions with optimal values of 0 and having different characteristics. The first 6 functions are unimodal while the remaining 7 functions are more difficult multimodal. Since HSAPA introduces a new parameter λ that could affect the performance, the effects of parameter λ are investigated by varying from 0.2 to 0.8 with a step size of 0.1. The results are compared against original HM, IHM, GHS, and SHS, described early, as well as Opposition-based Differential Evolution (ODE) [13], a state-of-the-art evolutionary algorithm. ODE is a recent variant of Differential Evolution (DE) [14] that was enhanced by utilizing the opposition-based learning in population initialization, generation jumping, and local improvement. The ODE's performance was comprehensively proven using a large set of complex benchmark functions [13].

Recall that SHS [6] was proposed with the low-discrepancy sequences for initialization. From my own prior experiments, this special initialization has improved the results to some extent. However, it has not been equipped into all tested algorithms, and the goal of this experiment is to focus on the evolution design. Therefore, I decide not to test SHS with such low-discrepancy initialization. For a fair comparison, I instead use a common uniform random initialization for every algorithm.

4.2 Experimental Setting

Two sets of experiments are conducted on each test function in 30 and 100 dimensions. For each function, 50 runs with independent seed numbers for random number generator are performed for each algorithm. The average results are presented in Table 2 and Table 3 for the case of 30 and 100 dimensions, respectively. Table 4 and Table 5 summarize the results by ranking all algorithms for each function in 30 and 100 dimensions, respectively.

According to the no free lunch theorem [15], for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class. Therefore, the major criterion for performance comparison in this work is the *mean of rank* (μ_R) achieved for each algorithm. The μ_R is computed from the average ranks of that algorithm for all 13 functions. The mean of rank for unimodal functions and that for multimodal functions (denoted as $\mu_{R,Uni}$ and $\mu_{R,Multi}$, respectively) are also computed, correspondingly.

4.3 Experimental Results and Analysis

The results in Table 2 and Table 3 indicate that the IHS and GHS fail to achieve the best results in nearly all cases. The overall μ_R summaries (in the last lines) of Table 4 and Table 5 also show that both algorithms finish in the worst ranks of all algorithms tested in both 30 and 100 dimensions. A careful investigation on the μ_R lines in both tables suggests that the μ_R's of the original HS are superior to both IHS and GHS, which are quite comparable. Obviously, HSAPA (λ=0.4 and 0.5) have the best convergence over all algorithms in many functions, e.g. $f01$, $f02$, $f10$, $f11$, and $f13$ in both 30 and 100 dimensions.

For a more comprehensible analysis on all 13 test functions, this subsection will group them into four categories based upon their dimension and modality:

1. *30D unimodal functions*: ODE and HSAPA with λ = 0.4 and 0.5 perform very well in 30D unimodal functions and share the first rank in this category. SHS follows as the 8th rank and can beat HSAPA with only λ = 0.2, which is the worst one of all λ tested.
2. *30D multimodal functions*: SHS's $\mu_{R,Multi}$ surpasses ODE with a small margin (6.5 vs. 7.0), but both algorithms are defeated by HSAPA with various λ values (0.3 to 0.6). HSAPA with λ = 0.4, 0.5, and 0.6 are ranked the top three with $\mu_{R,Multi}$ ranging from 3.17 to 4.33, far better than SHS and ODE. Table 2 also shows that only HSAPA with λ = 0.4, 0.5 and 0.6 can achieve the optimal value of 0 for $f11$-Griewank function in every run (mean = s.d. = 0).
3. *100D unimodal functions*: The top three $\mu_{R,Uni}$ belong to HSAPA with λ = 0.4 and 0.5 and ODE, respectively.
4. *100D multimodal functions*: The top three $\mu_{R,Multi}$ belong to HSAPA with λ = 0.4, 0.5 and 0.3, respectively.

In summary, HSAPA with λ = 0.4 and 0.5 are superior to all other tested algorithms in terms of μ_R for both unimodal and multimodal function sets in both 30 and 100 dimensions. In addition, the performance of HSAPA is not very sensitive to the parameter λ, especially with its value ranging from 0.3 to 0.6.

5. Conclusion

This paper proposes the HSAPA algorithm which is an improved harmony search in which both rate and distance of pitch adjustment are dynamic. The performance of HSAPA is evaluated using 13 widely accepted non-linear benchmark functions with various characteristics. The experiment, conducted in both 30 and 100 dimensions, confirms that HSAPA reliably outperforms all prior well-known harmony search algorithms. Its performance is also superior to ODE [13], a recent state-of-the-art evolutionary algorithm. In addition, the performance of HSAPA is not sensitive to its parameter λ, which is introduced to control the dynamics of pitch adjustment.

References

[1] Z.W. Geem, J.H. Kim, and G.V. Loganathan, "A New Heuristic Optimization Algorithm: Harmony Search", Simulations, vol. 76, 2001, pp. 60–68.

[2] Z.W. Geem, Ed., Music-Inspired Harmony Search Algorithm: Theory and Applications. New York: Springer-Verlag, 2009, series Studies in Computational Intelligence Series.

[3] K.S. Lee and Z.W. Geem, "A New Meta-Heuristic Algorithm for Continuous Engineering Optimization: Harmony Search Theory and Practice", Computer Methods in Applied Mechanics and Engineering, vol. 194, 36–38, 2005, pp. 3902–3933.

[4] A. Vasebi, M. Fesanghary, and S.M.T. Bathaeea, "Combined Heat and Power Economic Dispatch by Harmony Search algorithm," Int. J. Elect. Power Energy Syst., vol. 29, no. 10, 2007, pp. 713–719.

[5] M. Mahdavi, M. Fesanghary, E. Damangir, "An Improved Harmony Search Algorithm for Solving Optimization Problems", Applied Mathematics and Computation, vol. 188, 2007, pp. 1567–1579.

[6] C.-M. Wang, Y.-F. Huang, "Self-Adaptive Harmony Search Algorithm for Optimization", Expert Syst. Appl. vol. 37, 2010, pp. 2826–2837.

[7] M. G. Omran, M. Mahdavi, "Global-Best Harmony Search", Applied Mathematics and Computation, vol. 198(2), 2008, pp. 643–656.

[8] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence. Morgan Kaufmann, San Francisco, 2001.

[9] Y. Shi, R. Eberhart, "A Modified Particle Swarm Optimizer", in Proc. IEEE World Congr. Comput. Intell. 1998, pp. 69–73.

[10] Q.-K. Pan, P.N. Suganthan, M.F. Tasgetiren, J.J. Liang, "A Self-Adaptive Global Best Harmony Search Algorithm for Continuous Optimization Problems", Applied Math. and Comp., vol. 216, 2010, pp. 830–848.

[11] X. Yao, Y. Liu, G. Lin, "Evolutionary Programming Made Faster", IEEE Trans. Evol. Comput., vol. 3, no. 2, 1999, pp. 82–102.

[12] J. Brest, S. Greiner, B. Borko, M. Marjan, Ž. Vilgem, "Self-Adapting Control Parameters in Differential Evolution: A comparative study on numerical benchmark problems", IEEE Trans. Evol. Comp., vol. 10, no. 6, 2006, pp. 646–657.

[13] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, "Opposition-Based Differential Evolution", IEEE Trans. Evol. Comput., vol. 12, no. 1, 2008, pp. 64–79.

[14] R. Storn, K. Price, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces", J. of Global Optimization, vol. 11, 1997, pp. 341–359.

[15] D.H. Wolpert, W.G. Macready, "No free lunch theorems for optimization", IEEE Trans Evol. Comput. vol. 1, no. 1, 1997, pp. 67–82.

Table 1. Benchmark Functions.

Name	Function	Search Range
1. Sphere	$f01(x)=\sum_{i=1}^{D} x_i^2$	$-100 < x_i < 100$
2. Schwefel function 2.22	$f02(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D} x_i$	$-10 < x_i < 10$
3. Schwefel function 1.20	$f03(x)=\sum_{i=1}^{D}(\sum_{j=1}^{i} x_j)^2$	$-100 < x_i < 100$
4. Schwefel function 2.21	$f04(x)=max\lvert x_i\rvert, 1\le i\le D$	$-100 < x_i < 100$
5. Rosenbrock's valley	$f05(x)=\sum_{i=1}^{D-1}\left[100(x_{i+1}-x_i^2)^2+(1-x_i)^2\right]$	$-30 < x_i < 30$
6. Step	$f06(x)=\sum_{i=1}^{D}\left(\lfloor x_i+0.5\rfloor\right)^2$	$-100 < x_i < 100$
7. Quartic function with noise	$f07(x)=rand[0,1)+\sum_{i=1}^{D} i\cdot x_i^4$	$-1.28 < x_i < 1.28$
8. Schwefel function 2.26	$f08(x)=418.98289\cdot D+\sum_{i=1}^{D}-x_i\cdot\sin\sqrt{\lvert x_i\rvert}$	$-500 < x_i < 500$
9. Rastrigin	$f09(x)=\sum_{i=1}^{D}\left[x_i^2-10\cos(2\pi x_i)+10\right]$	$-5.12 < x_i < 5.12$
10. Ackley's path	$f10(x)=-20\cdot\exp\left(-0.2\cdot\sqrt{\frac{1}{D}\cdot\sum_{i=1}^{D} x_i^2}\right)-\exp\left(\frac{1}{D}\cdot\sum_{i=1}^{D}\cos(2\pi x_i)\right)+20+e$	$-320 < x_i < 32$
11. Griewank	$f11(x)=\frac{1}{4000}\sum_{i=1}^{D} x_i^2-\prod_{i=1}^{D}\cos(\frac{x_i}{\sqrt{i}})+1$	$-600 < x_i < 600$
12. Penalized function 1	$f12(x)=\frac{\pi}{D}\left\{10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(x_i-1)^2\cdot\left[1+10\sin^2(\pi y_{i+1}+1)\right]+(y_D-1)^2\right\}$ $+\sum_{i=1}^{D} u(x_i,10,100,4),\ y_i=1+(x_i+1)/4,\ u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a,\\ 0, & -a\le x_i\le a,\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	$-50 < x_i < 50$
13. Penalized function 2	$f13(x)=0.1\left\{\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+(x_D-1)\left[1+\sin^2(2\pi x_D)\right]\right\}$ $+\sum_{i=1}^{D} u(x_i,5,100,4)$	$-50 < x_i < 50$

Table 2. Mean and standard deviation (in parenthesis) of the benchmark function optimization results in **30** dimensions. The results were averaged over 50 runs. The best (lowest) mean and standard deviations achieved are highlighted with red bold font.

30D	HSAPA-0.2	HSAPA-0.3	HSAPA-0.4	HSAPA-0.5	HSAPA-0.6	HSAPA-0.7	HSAPA-0.8	ODE	SHS	IHS	GHS	HS
*f*01	2.264E-01	1.389E-14	**1.384E-41**	3.068E-40	3.505E-35	6.281E-24	1.807E-20	6.691E-15	1.813E-09	1.345E+00	1.336E+00	5.144E-04
Sphere	(4.247E-01)	(5.381E-14)	**(5.243E-41)**	(1.187E-39)	(9.862E-35)	(2.432E-23)	(6.818E-20)	(7.002E-15)	(1.282E-08)	(5.383E-01)	(4.935E-01)	(3.475E-03)
*f*02	2.503E-03	1.678E-17	**5.535E-27**	1.859E-26	7.147E-22	1.825E-17	9.672E-14	1.403E-04	9.096E-05	4.557E-01	4.459E-01	9.755E-03
Schwefel222	(5.502E-03)	(6.480E-17)	**(2.144E-26)**	(6.237E-26)	(9.854E-22)	(3.708E-17)	(1.491E-13)	(7.012E-05)	(4.808E-04)	(6.960E-02)	(7.000E-02)	(9.604E-04)
*f*03	7.733E+02	2.471E+02	9.284E+01	2.199E+02	4.970E+02	1.273E+03	2.563E+03	**1.018E+01**	2.131E+04	3.808E+03	3.813E+03	3.663E+03
Schwefel120	(3.594E+02)	(2.057E+02)	(3.489E+01)	(9.847E+01)	(2.178E+02)	(5.465E+02)	(1.292E+03)	**(7.158E+00)**	(6.824E+03)	(1.239E+03)	(1.124E+03)	(1.250E+03)
*f*04	4.410E+00	1.671E+00	2.483E-01	1.002E-01	3.474E-02	3.189E-02	1.111E-01	**8.632E-03**	1.211E-02	3.916E+00	3.875E+00	3.778E+00
Schwefel221	(6.605E-01)	(7.756E-01)	(2.377E-01)	(1.955E-01)	(3.918E-02)	(3.616E-02)	(1.477E-01)	**(4.727E-02)**	(3.753E-02)	(6.989E-01)	(5.872E-01)	(7.019E-01)
*f*05	2.014E+02	9.558E+01	4.745E+01	2.711E+01	2.696E+01	2.688E+01	3.118E+01	**2.580E+01**	3.754E+01	1.659E+02	1.773E+02	7.640E+01
Rosenbrock	(9.924E+01)	(4.729E+01)	(2.998E+01)	(3.281E-01)	(2.066E-01)	(2.917E-01)	(1.605E+01)	**(1.022E+00)**	(2.341E+01)	(4.394E+01)	(7.738E+01)	(7.011E+01)
*f*06	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	6.000E-02	8.000E-02	2.000E-02
Step	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	(2.399E-01)	(2.740E-01)	(1.414E-01)
*f*07	4.921E-03	**2.018E-03**	2.425E-03	3.374E-03	4.325E-03	4.660E-03	6.390E-03	3.317E-03	8.499E-03	4.348E-02	4.963E-02	3.754E-02
QuarticNois	(2.952E-03)	**(3.263E-04)**	(5.486E-04)	(9.355E-04)	(9.662E-04)	(9.393E-04)	(1.354E-03)	(1.104E-03)	(1.952E-03)	(1.233E-02)	(2.118E-02)	(1.284E-02)
*f*08	1.208E-01	**5.012E-02**	2.725E-01	9.171E+00	1.458E+01	2.344E+01	2.895E+01	7.360E+03	1.114E+00	4.096E+00	4.366E+00	1.878E+00
Schwefel226	(1.430E-01)	**(1.258E-01)**	(4.616E-01)	(1.128E+01)	(2.980E+01)	(4.573E+01)	(4.489E+01)	(2.623E+02)	(1.829E+00)	(1.469E+00)	(1.942E+00)	(1.258E+00)
*f*09	9.154E-02	3.134E-01	1.478E+00	1.838E+00	1.584E+00	2.304E+00	1.902E+00	1.029E+02	1.893E+00	6.998E-01	6.626E-01	**4.269E-02**
Rastrigin	(2.567E-01)	(5.786E-01)	(1.223E+00)	(1.446E+00)	(1.003E+00)	(1.718E+00)	(8.953E-01)	(1.758E+01)	(1.066E+00)	(2.413E-01)	(2.334E-01)	**(1.968E-01)**
*f*10	6.981E-02	2.672E-06	**3.109E-15**	**3.109E-15**	3.819E-15	1.874E-14	1.191E-11	6.013E-08	3.125E-05	4.863E-01	4.668E-01	2.937E-03
Ackley	(9.128E-02)	(1.030E-05)	**(0)**	**(0)**	(1.471E-15)	(5.271E-14)	(1.874E-11)	(4.707E-08)	(2.113E-04)	(1.200E-01)	(1.182E-01)	(3.165E-04)
*f*11	6.399E-02	1.834E-07	**0**	**0**	**0**	4.931E-04	1.314E-03	4.105E-04	4.669E-03	9.052E-01	9.226E-01	5.007E-01
Griewank	(4.770E-02)	(7.101E-07)	**(0)**	**(0)**	**(0)**	(1.910E-03)	(3.592E-03)	(2.249E-03)	(1.417E-02)	(1.076E-01)	(1.025E-01)	(2.023E-01)
*f*12	1.565E-01	1.191E-01	1.191E-01	9.680E-02	1.190E-01	1.339E-01	1.320E-01	**7.437E-03**	9.617E-02	1.260E-01	1.233E-01	1.321E-01
Penalized1	(5.668E-02)	(5.101E-02)	(6.624E-02)	(5.739E-02)	(5.108E-02)	(4.617E-02)	(4.278E-02)	**(2.830E-02)**	(5.982E-02)	(5.350E-02)	(5.604E-02)	(6.244E-02)
*f*13	7.463E-04	1.278E-07	1.399E-32	**1.358E-32**	1.366E-32	5.601E-26	1.420E-25	1.735E-17	2.239E-07	1.226E-03	9.292E-04	2.210E-04
Penalized2	(2.835E-03)	(4.951E-07)	(7.796E-34)	**(3.183E-34)**	(4.337E-34)	(2.169E-25)	(4.379E-25)	(6.712E-17)	(1.584E-06)	(1.187E-03)	(8.038E-04)	(1.554E-03)

Table 3. Mean and standard deviation (in parenthesis) of the benchmark function optimization results in **100** dimensions. The results were averaged over 50 runs. The best (lowest) mean and standard deviations achieved are highlighted with red bold font.

100D	HSAPA-0.2	HSAPA-0.3	HSAPA-0.4	HSAPA-0.5	HSAPA-0.6	HSAPA-0.7	HSAPA-0.8	ODE	SHS	IHS	GHS	HS
*f*01	2.378E-01	3.292E-04	**3.717E-23**	3.969E-17	1.366E-14	1.556E-09	1.332E-05	1.851E-06	6.159E-05	2.078E+00	1.853E+00	3.200E-04
Sphere	(1.357E-01)	(1.251E-03)	**(1.386E-22)**	(1.524E-16)	(3.525E-14)	(4.750E-09)	(4.749E-05)	(2.166E-06)	(2.426E-04)	(5.005E-01)	(5.078E-01)	(2.056E-05)
*f*02	4.022E-02	1.130E-09	**4.891E-17**	9.209E-11	1.392E-08	4.792E-08	4.444E-06	1.493E-02	2.656E-02	9.118E-01	9.036E-01	1.140E-01
Schwefel222	(2.251E-02)	(2.700E-09)	**(1.873E-16)**	(3.534E-10)	(4.458E-08)	(6.808E-08)	(6.085E-06)	(5.120E-03)	(1.867E-02)	(1.067E-01)	(7.981E-02)	(4.423E-03)
*f*03	8.818E+04	1.681E+05	2.393E+05	3.025E+05	3.647E+05	4.282E+05	4.272E+05	**1.981E+04**	4.109E+05	5.086E+04	4.125E+04	5.125E+04
Schwefel120	(3.560E+04)	(2.856E+04)	(3.263E+04)	(3.765E+04)	(3.549E+04)	(3.597E+04)	(5.391E+04)	**(8.218E+03)**	(3.801E+04)	(2.006E+04)	(4.759E+03)	(1.671E+04)
*f*04	5.443E+00	4.907E+00	6.594E+00	8.655E+00	1.128E+01	1.278E+01	1.585E+01	**1.725E-01**	3.617E-01	6.751E+00	6.640E+00	6.737E+00
Schwefel221	(3.900E-01)	(6.568E-01)	(8.671E-01)	(1.267E+00)	(9.556E-01)	(1.440E+00)	(1.387E+00)	**(3.470E-01)**	(1.201E-01)	(6.617E-01)	(5.300E-01)	(5.921E-01)
*f*05	4.467E+02	1.691E+02	1.174E+02	1.132E+02	1.279E+02	1.528E+02	1.664E+02	1.036E+02	**1.023E+02**	4.442E+02	4.307E+02	2.444E+02
Rosenbrock	(6.056E+01)	(3.470E+01)	(3.086E+01)	(2.913E+01)	(3.070E+01)	(6.964E+01)	(5.352E+01)	(2.101E+01)	**(1.651E+01)**	(6.188E+01)	(6.647E+01)	(1.512E+02)
*f*06	**0**	**0**	**0**	**0**	**0**	**0**	**0**	4.667E-01	**0**	**0**	3.333E-02	**0**
Step	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	**(0)**	(1.655E+00)	**(0)**	**(0)**	(1.826E-01)	**(0)**
*f*07	2.618E-02	**4.761E-03**	6.222E-03	9.349E-03	1.295E-02	1.767E-02	2.407E-02	8.689E-03	2.177E-02	1.239E-01	1.358E-01	7.389E-02
QuarticNois	(1.202E-02)	**(7.925E-04)**	(5.769E-04)	(1.300E-03)	(2.450E-03)	(2.405E-03)	(2.977E-03)	(2.997E-03)	(2.418E-03)	(1.992E-02)	(2.790E-02)	(1.038E-02)
*f*08	1.866E+00	4.070E+00	9.091E+00	1.788E+01	4.223E+01	8.539E+01	1.093E+02	3.243E+04	1.459E+01	5.642E+00	5.368E+00	**1.042E+00**
Schwefel226	(1.348E+00)	(2.558E+00)	(5.098E+00)	(7.811E+00)	(2.594E+01)	(5.214E+01)	(6.499E+01)	(5.508E+02)	(7.253E+00)	(1.103E+00)	(1.023E+00)	**(4.727E-01)**
*f*09	1.575E-01	1.485E+00	5.231E+00	1.144E+01	1.435E+01	1.871E+01	2.229E+01	6.009E+02	2.040E+01	8.936E-01	9.561E-01	**5.659E-02**
Rastrigin	(9.360E-02)	(8.498E-01)	(2.233E+00)	(2.492E+00)	(2.965E+00)	(2.930E+00)	(3.061E+00)	(6.880E+01)	(3.107E+00)	(1.745E-01)	(2.156E-01)	**(4.606E-03)**
*f*10	4.812E-02	1.023E-04	**3.760E-12**	1.202E-10	4.240E-08	1.019E-06	1.885E-03	6.533E-04	2.315E-03	2.672E-01	2.729E-01	6.979E-03
Ackley	(1.944E-02)	(2.515E-04)	**(1.102E-11)**	(3.798E-10)	(8.753E-08)	(1.713E-06)	(4.292E-03)	(9.166E-04)	(2.578E-03)	(3.191E-02)	(3.807E-02)	(2.405E-04)
*f*11	1.932E-01	5.392E-04	**0**	**0**	9.325E-13	3.237E-10	2.974E-08	5.558E-03	3.399E-05	7.481E-01	7.317E-01	2.891E-01
Griewank	(8.688E-02)	(1.901E-03)	**(0)**	**(0)**	(2.773E-12)	(8.362E-10)	(5.942E-08)	(1.753E-02)	(7.100E-05)	(1.137E-01)	(1.200E-01)	(9.715E-02)
*f*12	2.606E-01	2.827E-01	2.529E-01	2.677E-01	2.981E-01	2.830E-01	3.683E-01	**1.570E-01**	3.343E-01	3.428E-01	3.761E-01	4.128E-01
Penalized1	(1.089E-01)	(1.105E-01)	(1.072E-01)	(1.099E-01)	(1.165E-01)	(1.104E-01)	(1.325E-01)	**(1.095E-01)**	(1.057E-01)	(7.133E-02)	(1.075E-01)	(1.213E-01)
*f*13	6.170E-05	6.849E-08	**3.564E-23**	3.338E-21	2.393E-17	2.711E-08	4.749E-11	1.831E-03	2.488E-08	7.525E-04	1.061E-03	2.801E-05
Penalized2	(2.281E-05)	(2.421E-07)	**(1.380E-22)**	(1.283E-20)	(5.678E-17)	(1.050E-07)	(1.015E-10)	(8.204E-03)	(5.621E-08)	(5.006E-04)	(1.011E-03)	(1.398E-06)

Table 4. Rank of the benchmark function optimization results in 30 dimensions. The results were categorized into unimodal function set and multimodal function set of which the means and summarized ranks are shown. The last two lines summarize for all 13 functions.

30D		HSAPA-0.2	HSAPA-0.3	HSAPA-0.4	HSAPA-0.5	HSAPA-0.6	HSAPA-0.7	HSAPA-0.8	ODE	SHS	IHS	GHS	HS
Unimodal	$f01$	10	7	**1**	2	3	4	5	6	8	12	11	9
	$f02$	9	4	**1**	2	3	5	6	8	7	12	11	10
	$f03$	6	4	2	3	5	7	8	**1**	12	10	11	9
	$f04$	12	8	7	5	4	3	6	**1**	2	11	10	9
	$f05$	12	9	7	4	3	2	5	**1**	6	10	11	8
	$f06$	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	11	12	10
	$f07$	7	**1**	2	4	5	6	8	3	9	11	12	10
Unimodal	$\mu_{R,Uni}$	8.14	4.86	**3.00**	**3.00**	3.43	4.00	5.57	**3.00**	6.43	11.00	11.14	9.29
7 functions	Rank	9	6	**1**	**1**	4	5	7	**1**	8	11	12	10
Multimodal	$f08$	2	**1**	3	8	9	10	11	12	4	6	7	5
	$f09$	2	3	6	8	7	11	10	12	9	5	4	**1**
	$f10$	10	7	**1**	**1**	3	4	5	6	8	12	11	9
	$f11$	9	4	**1**	**1**	**1**	6	7	5	8	11	12	10
	$f12$	12	6	5	3	4	11	9	**1**	2	8	7	10
	$f13$	10	7	3	**1**	2	4	5	6	8	12	11	9
Multimodal	$\mu_{R,Multi}$	7.50	4.67	**3.17**	3.67	4.33	7.67	7.83	7.00	6.50	9.00	8.67	7.33
6 functions	Rank	8	4	**1**	2	3	9	10	6	5	12	11	7
Overall	μ_R	7.94	4.86	**2.93**	3.13	3.83	5.53	6.57	4.47	6.56	10.20	10.21	8.55
13 functions	Final rank	9	5	**1**	2	3	6	8	4	7	11	12	10

Table 5. Rank of the benchmark function optimization results in 100 dimensions. The results were categorized into unimodal function set and multimodal function set of which the means and summarized ranks are shown. The last two lines summarize for all 13 functions.

100D		HSAPA-0.2	HSAPA-0.3	HSAPA-0.4	HSAPA-0.5	HSAPA-0.6	HSAPA-0.7	HSAPA-0.8	ODE	SHS	IHS	GHS	HS
Unimodal	$f01$	10	9	**1**	2	3	4	6	5	7	12	11	8
	$f02$	9	3	**1**	2	4	5	6	7	8	12	11	10
	$f03$	5	6	7	8	9	12	11	**1**	10	3	2	4
	$f04$	4	3	5	9	10	11	12	**1**	2	8	6	7
	$f05$	12	8	4	3	5	6	7	2	**1**	11	10	9
	$f06$	**1**	**1**	**1**	**1**	**1**	**1**	**1**	12	**1**	**1**	11	**1**
	$f07$	9	**1**	2	4	5	6	8	3	7	11	12	10
Unimodal	$\mu_{R,Uni}$	7.14	4.43	**3.00**	4.14	5.29	6.43	7.29	4.43	5.14	8.29	9.00	7.00
7 functions	Rank	9	3	**1**	2	6	7	10	3	5	11	12	8
Multimodal	$f08$	2	3	6	8	9	10	11	12	7	5	4	**1**
	$f09$	2	5	6	7	8	9	11	12	10	3	4	**1**
	$f10$	10	5	**1**	2	3	4	7	6	8	11	12	9
	$f11$	9	7	**1**	**1**	3	4	5	8	6	12	11	10
	$f12$	3	5	2	4	7	6	10	**1**	8	9	11	12
	$f13$	9	7	**1**	2	3	6	4	12	5	10	11	8
Multimodal	$\mu_{R,Multi}$	5.83	5.33	**2.83**	4.00	5.50	6.50	8.00	8.50	7.33	8.33	8.83	6.83
6 functions	Rank	5	3	**1**	2	4	6	9	11	8	10	12	7
Overall	μ_R	6.74	4.70	**2.80**	3.94	5.42	6.50	7.75	5.96	6.01	8.49	9.13	7.00
13 functions	Final rank	8	3	**1**	2	4	7	10	5	6	11	12	9

Author

Chukiat Worasucheep is an assistance professor of Applied Computer Science program, Faculty of Science, King Mongkut's University of Technology Thonburi (KMUTT), Thailand. His research interests are computational intelligence, particularly evolutionary computations and particle swarm optimizations, for financial, engineering, and scheduling applications.