Mathematical and Computational Applications

MDPI

Article

Hybrid Harmony Search Optimization Algorithm for Continuous Functions

José Alfredo Brambila-Hernández *, Miguel Ángel García-Morales, Héctor Joaquín Fraire-Huacuja, Eduardo Villegas-Huerta and Armando Becerra-del-Ángel

Graduate Program Division, Tecnológico Nacional de México, Instituto Tecnológico de Ciudad Madero, Ciudad Madero 89440, Mexico

* Correspondence: jabrambila@gmail.com

Abstract: This paper proposes a hybrid harmony search algorithm that incorporates a method of reinitializing harmonies memory using a particle swarm optimization algorithm with an improved opposition-based learning method (IOBL) to solve continuous optimization problems. This method allows the algorithm to obtain better results by increasing the search space of the solutions. This approach has been validated by comparing the performance of the proposed algorithm with that of a state-of-the-art harmony search algorithm, solving fifteen standard mathematical functions, and applying the Wilcoxon parametric test at a 5% significance level. The state-of-the-art algorithm uses an opposition-based improvement method (IOBL). Computational experiments show that the proposed algorithm outperforms the state-of-the-art algorithm. In quality, it is better in fourteen of the fifteen instances, and in efficiency is better in seven of fifteen instances.

Keywords: harmony search; improved learning; opposition-based; hybrid algorithm

check for updates

Citation: Brambila-Hernández, J.A.; García-Morales, M.Á.; Fraire-Huacuja, H.J.; Villegas-Huerta, E.; Becerra-del-Ángel, A. Hybrid Harmony Search Optimization Algorithm for Continuous Functions. *Math. Comput. Appl.* **2023**, *28*, 29. https://doi.org/10.3390/mca28020029

Academic Editors: Marcela Quiroz-Castellanos, Daniel Hernández, Leonardo Trujillo and Oliver Schütze

Received: 14 January 2023
Revised: 19 February 2023
Accepted: 21 February 2023
Published: 22 February 2023

Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Currently, society requires maximum benefits at minimum costs. In order to achieve this, optimization techniques are generally used. However, many real-world optimization problems are considered high computational complexity and are called NP-hard. Moreover, these problems have the characteristic that exact solution methods cannot obtain optimal solutions in reasonable times. Using metaheuristic methodologies is considered a good alternative that offers satisfactory solutions for the user in a reasonable time. Harmony search algorithms constitute a metaheuristic methodology for solving continuous optimization problems proposed by Zong Woo Geem [1].

The process of musical improvisation inspires the methodology of harmonic search. In this, a predefined number of musicians try to tune the tone of their instruments until they achieve a pleasant harmony. In nature, harmony is a relationship between several sound waves with different frequencies. Therefore, the quality of improvised harmony is determined by aesthetic estimation. In order to improve aesthetic valuation and find the best harmony, musicians perform multiple rehearsals [2].

Harmony search algorithms are currently considered a competitive alternative to solve a large number of optimization problems that have several advantages over other metaheuristics that are available in the state-of-the-art. For example, they only require adjusting a relatively small number of parameters [3–5].

In ref. [6], a harmony search algorithm using an improved OBL mechanism (IOBL) is proposed. This improved version uses randomness to create a new possible solution and improve the convergence process of such an algorithm. In addition, the IOBL mechanism is used in the upgrade process.

In the literature, we can find improved versions of harmony search that have taken features from the particle swarm optimization (PSO) algorithm [7,8]. Among them, we can find the algorithm proposed by Omran and Mahdavi called global-best harmony

search [9], which modifies the pitch setting within the new harmonies improvisation step and generates a new harmony close to the best harmony memory (HM) harmony. This mechanism is inspired by updating a particle in the PSO algorithm, which is influenced by the best position visited by the same particle and by the position of the best particle in the swarm. For his part, Geem proposed the particle-swarm harmony search [10] algorithm, in which he takes the contribution of Omran and Mahdavi [9] and introduces it as a fourth way of generating a new harmony which they call particle-swarm consideration and conserving the original pitch setting, this within the improvisation step of new harmonies. Both algorithms use the mechanism inspired by the PSO algorithm in the new harmonies improvisation step. In the case of the hybrid harmony search algorithm, a version of the particle swarm optimization algorithm is used to which the IOBL technique (PSO-IOBL) has been applied to regenerate a part of the harmony memory (HM) each time the search process harmonic exceeds the allowed stagnation percentage.

The main contributions of this work are:

- A new harmony search optimization algorithm to solve continuous functions.
- A new harmony memory reset mechanism applying a particle swarm optimization algorithm to improve the quality of the solutions.
- The incorporation of the improved opposition-based learning technique (IOBL) in the process of reinitializing the memory of harmonies.

2. Basic Harmony Search Algorithm

In a harmony search algorithm (Algorithm 1), harmony is a feasible solution, and each decision variable of harmony is called a note [1]. Several harmonies (HMS) are stored in a harmony memory (HM). Assuming that the goal is to minimize or maximize an aptitude function (f) subject to decision variables x, the problem is defined in Equation (1):

$$\text{Min or Max } f(x_1, x_2, \ldots, x_D) \tag{1}$$

where f is the aptitude function, $X[i] = (I = 1, 2, \ldots, D)$ is the decision variable i, and D denotes the dimensions of the problem.

The harmony search algorithm is generalized; the following steps are obtained:

1. Initialize a new memory of harmonies (HM).
2. Improvise a new harmony.
3. The new harmony must be included or excluded from HM.
4. Steps 2 and 3 must be repeated until the stop criterion is met; when the stop criterion is met, go to step 5.
5. The best harmony stored in HM is returned as an optimal solution.

Algorithm 1. Harmony search

Inputs: *MaxIt*: maximum number of iterations,
HMS: harmonies memory size,
nNew: memory size of new harmonies,
HMCR: Rate of consideration of the memory of harmonies,
PAR: Pitch adjustment rate,
bw: bandwidth,
bw_damp: bandwidth upgrade,
D: number of dimensions,
Cost: calculation of the objective function,
Xnew: new harmony,
u: upper bound,
l: lower bound,
NEW: new set of harmonies,
HM: harmony memory.

Outputs: *bestSol*: best solution found.

Used functions: *random_number* [*0,1*]: generates a random value between 0 and 1.

Algorithm 1. *Cont.*

```
1. Initialize new random harmony memory HM of size HMS
2. Sort HM by Cost
3. bestSol = HM (1)
4. for it = 1 to MaxIt do
5.     for k = 1 to nNEW do
6.         Generate new random harmony Xnew
7.         for j = 1 to D do
8.             r = random_number [0,1]
9.             if r ≤ HMCR then
10.                Randomly select harmony X[i] stored in HM
11.                X[new,j] = X[i,j]
12.            end if
13.            r = random_number [0,1]
14.            if r < PAR then
15.                X[new,j] = X[new,j] + bw × random_number [0,1] × | u[j] – l[j] |
16.            end if
17.        end for
18.        Evaluate(Xnew)
19.        Add Xnew to memory of new harmonies NEW
20.    end for
21.    HM ∪ NEW
22.    Sort HM by Cost
23.    Truncate HM to HMS harmonies
24.    bestSol = HM (1)
25.    bw = bw × bw_damp
26. end for
27. return bestSol
```

3. Method of Improving Opposition-Based Learning (IOBL)

In [6], a new, improved version of the original OBL (IOBL) is presented, which includes randomness in improving the diversity of the solution. Said improvement provides better performance in continuous optimization problems. This improved version updates the harmony search algorithm allowing a better solution space exploitation (Algorithm 2).

This technique starts by generating a random value r [0,1] multiplied by each value stored in the current solution vector $X[D]$. Then, the objective value for each modified element is calculated and evaluated if it is less than the objective value before the modification; if so, the value is updated at the current position of the array. Otherwise, it advances to the next position. This process ends when all the values of the solution vector have been visited.

Algorithm 2. Improving opposition-based learning (IOBL)

Inputs: $X[D] = (x_1, x_2, \ldots, x_D)$: initial solution,
D: number of dimensions,
r: random value.

Outputs: $X[D] = (x_1, x_2, \ldots, x_D)$: improved solution.

Used functions: *random_number* [0,1]: generates a random value between 0 and 1,
objectivefunction(X): calculates the cost of the objective value.

```
1. r = random_number [0,1]
2. While (i < D) do
3.     X[i]′ = X[i] × r
4. end While
5. if (objectivefunction(X′) < objectivefunction(X)) then
6.     X = X′
7. end if
8. return X
```

4. General Structure of the Hybrid Harmony Search Algorithm (HHS-IOBL)

This section describes all the elements that make up the structure of the hybrid algorithm proposed in this research, called the hybrid algorithm of harmony search with improved learning based on the opposition (HHS-IOBL).

4.1. Algorithm Parameters

Table 1 describes the parameters and variables used in the HHS-IOBL and HS-IOBL algorithms.

Table 1. Parameters and variables used.

Parameter	HHS-IOBL	HS-IOBL
Harmony memory size (*HMS*)	5	5
Iterations	100	100
HMR	0.95	0.95
PAR	0.7	0.7
Dimensions	30	30
Dimensions F16, F17	2	2
Percentage in HM replacement (ζ)	0.3	NA
Percentage of stagnation allowed (ς)	0.2	NA
PSO.MaxIt	50	NA
PSO.w	0.7298	NA
PSO.wdamp	0.99	NA
PSO.c1, c2	1.49618, 1.49618	NA

4.2. Particle Swarm Optimization Algorithm with IOBL (PSO-IOBL)

This paper proposes the use of a particle swarm algorithm [7,8] (Algorithm 3) as a method to reinitialize the memory of harmonies. The primary function of this algorithm is to help in the convergence of the harmonic search process. This algorithm also uses the IOBL technique to enable better exploration in the solution space. In steps 1 and 2, an initial population of particles is generated. The best global particle is obtained concerning its objective value. Step 3 calculates the minimum (*velMin*) and maximum (*velMax*) speed with which these particles will move in the solution space. From steps 4 to 13, each particle is updated concerning its position and velocity. Step 14 calculates the objective function of each particle. From steps 15 to 22, the best particle is obtained concerning its objective value and is compared with the global solution; if its value is better than the global solution, it is updated. From steps 23 to 35, the global solution obtained from the previous steps is improved with the IOBL technique, the objective value of this applied improvement is obtained, and it is compared again with the global solution; in case of having a better objective value, the global solution is replaced, and the inertial weight (w) is updated. This process continues until the maximum number of iterations (*MaxIt*) is achieved. Finally, in step 36, the algorithm returns the best overall solution obtained during the process.

4.3. Hybrid Harmony Search Algorithm (HHS-IOBL)

This research paper proposes the hybridization of a harmony search algorithm using a particle swarm algorithm that incorporates the IOBL technique to reinitialize the memory of harmonies in cases where there is local stagnation (the objective value cannot be further optimized) during a certain number of iterations. The mechanism of a reinitialization of harmonies (Algorithm 4) replaces a percentage ζ of the harmonies stored in memory (HM). In such a way that a new solution is generated through the particle swarm algorithm, which receives the memory of harmonies as a parameter to be used as the initial population, the solution is compared to an $HM[i]$ harmony taken randomly from HM if the new solution has a better objective value, then it replaces $HM[i]$ memory. This mechanism is incorporated into the basic harmony search algorithm to transform it into HHS-IOBL (Algorithm 5). In addition, a simple mechanism to determine the stagnation of a solution is also incorporated within the

general iteration cycle. After joining HM with new, ordering, and truncating HM (see steps 21 to 24), it is determined if *HM* (1) is equal to *bestSol* if they are equal, the stagnation variable is increased. Otherwise, this variable is zero (see steps 26 to 30). When $stagnation \geq MaxIt \times \varsigma$ the harmonies reinitialization mechanism is executed, the stagnation variable becomes zero, and *HM* is sorted by the objective value (see steps 31 to 36). This process continues as long as the number of iterations is not exceeded and, in the end, the best solution found *bestSol* returns.

Algorithm 3. Particle swarm optimization with IOBL (PSO-IOBL)

Inputs: *MaxIt*: Maximum number of iterations,
nPop: Population size,
w: Inertia weight,
wdamp: Inertia weight damping ratio,
c1: Personal learning coefficient,
c2: Global learning coefficient,
D: Number of dimensions,
varMin: The minimum value that a decision variable can take,
varMax: The maximum value that a decision variable can take,
HM: Harmony memory.

Outputs: *GlobalBest*: best solution found.

Used Functions: *InitializePopulation* (*HM*): generates an initial population,
random_number [*0,1*]: generates a random value between 0 and 1,
getBestGlobal (*particles*): obtain the best global particle,
Max (*particles*[*i*].*position*[*j*], *varMin*):obtain the maximum of two values,
Min (*particles*[*i*].*position*[*j*], *varMax*):obtain the minimum of two values,
IOBL (*particle*): applies the technique of improving opposition- based learning,
Evalute (*particles*[*i*]): calculate the objective function of the particles.

1: *particles* = *InitializePopulation* (*HM*)
2: *GlobalBest* = *getBestGlobal* (*particles*)
3: calculate *velMin* and *velMax*
4: for *it* = 1 to *MaxIt* do
5: for *i* = 1 to *nPop* do
6: for *j* = 1 to *D* do
7: *particles*[*i*] = $w \times particles_i$.*velocity*[*j*] + *c1* × *random_number* [*0,1*] × (*particles*[*i*].*bestPosition*[*j*] − *particles*[*i*].*position*[*j*]) + *c2* × *random_number* [*0,1*] × (*GlobalBest.position*[*j*] − *particles*[*i*].*position*[*j*]);
8: *particles*[*i*].*velocity*[*j*] = *Max* (*particles*[*i*].*velocity*[*j*], *velMin*);
9: *particles*[*i*].*velocity*[*j*] = *Min* (*particles*[*i*].*velocity*[*j*], *velMax*);
10: *particles*[*i*].*position*[*j*] = *particles*[*i*].*posicion* + *particles*[*i*].*velocity*[*j*];
11: *particles*[*i*].*position*[*j*] = *Max* (*particles*[*i*].*position*[*j*], *varMin*);
12: *particles*[*i*].*position*[*j*] = *Min* (*particles*[*i*].*position*[*j*], *varMax*);
13: end for
14: *Evalute* (*particles*[*i*])
15: if *particles*[*i*].*cost* < *particles*[*i*].*bestCost* then
16: *particles*[*i*].*bestPosition* = *particles*[*i*].*position*
17: *particles*[*i*].*bestCost* = *particles*[*i*].*cost*
18: if *particles*[*i*].*bestCost* < *GlobalBest.cost* then
19: *GlobalBest* = *particles*[*i*]
20: end if
21: end if
22: end for
23: for *i* = 1 to *nPop* do
24: *primeParticle* = *IOBL* (*particle*)
25: end for
26: *Evaluate* (*primeParticle*)
27: if *primeParticle.cost* < *particle*[*i*].*bestCost* then
28: *particulas*[*i*] = *primeParticle*
29: if *particles*[*i*].*cost* < *GlobalBes.cost* then
30: *GlobalBest* = *particles*[*i*]
31: end if
32: end if
33: end for
34: *w* = *w* × *wdamp*
35: end for
36: return *GlobalBest*

Algorithm 4. Reinitializing Harmony Memory

Inputs: ζ: Percentage of population replacement,
HM: Harmony memory,
HMS: Harmony memory size.

Outputs: *HM*: new harmony memory.

Used Functions: *random_integer* [1, *HMS*]: generates a random value between 1 and HMS,
PSO_IOBL(*HM*): calculates a new solution using the PSO algorithm.

1. $numRegen = \zeta \times HMS$
2. for $i = 1$ to $numRegen$ do
3. $index = random\_integer$ [1, *HMS*]
4. $newSolution = PSO\_IOBL(HM)$
5. if $newSolution.cost < HM[indice].cost$ then
6. $HM[indice] = newSolution$
7. end if
8. end for

The previous process is shown in Figure 1, where the interaction of HS and PSO-IOBL can be observed more generally.

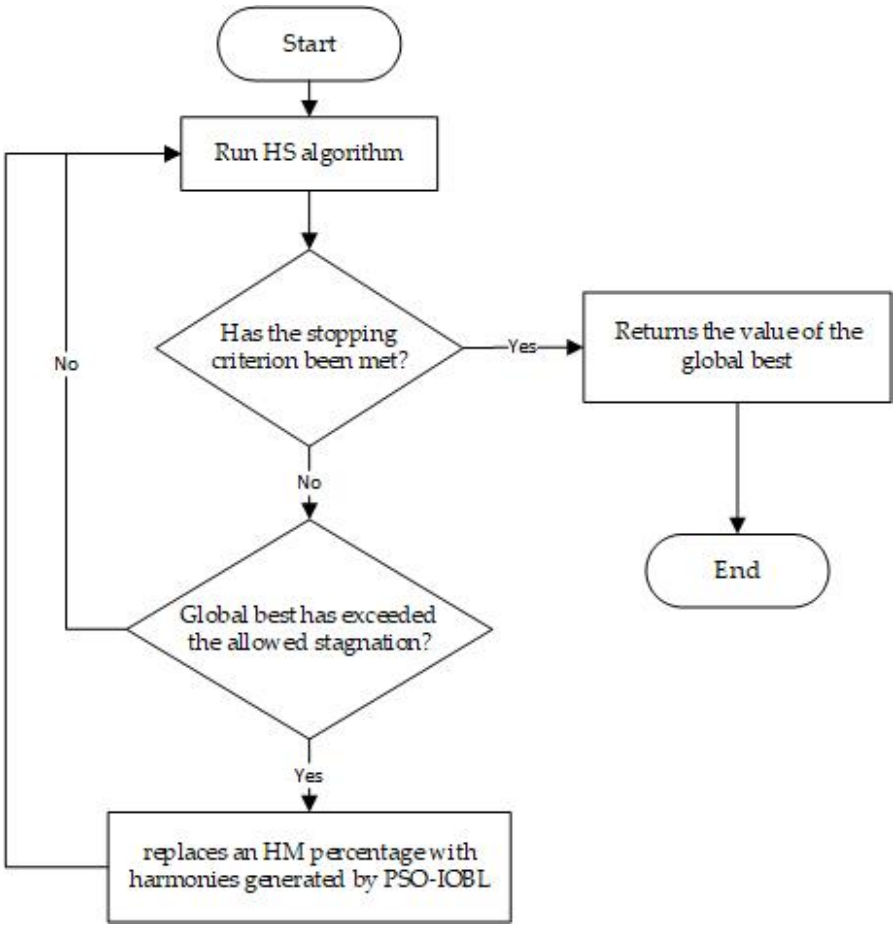

Figure 1. General scheme of the HHS algorithm.

Algorithm 5. Hybrid Harmony Search—IOBL

Inputs: *MaxIt*: maximum number of iterations,
HMS: harmony memory size,
nNew: memory size of new harmonies,
HMCR: Rate of consideration of the memory of harmonies,
PAR: Pitch adjustment rate,
bw: bandwidth,
bw_damp: bandwidth upgrade,
D: number of dimensions,
ζ: Percentage of HM replacement,
ς: percentage of stagnation allowed.

Outputs: *bestSol*: best solution found.

Used functions: *Cost*: *random_number* [*0,1*]*:* generates a random value between 0 and 1.

1. Initialize new random harmony memory *HM* of size *HMS*
2. *stagnation* = 0
3. Sort *HM* by Cost
4. *bestSolt* = *HM* (1)
5. for *it* = 1 to *MaxIt* do
6. for *k* = 1 to *nNEW* do
7. Generate new random harmony *xnew*
8. for *j* = 1 to *D* do
9. *r* = *random_number* [*0,1*]
10. if $r \leq HMCR$ then
11. Randomly select harmony *X*[*i*]stored in *HM*
12. *X*[*new,j*] = *X*[*i,j*]
13. end if
14. *r* = *random_number* [*0,1*]
15. if $r < PAR$ then
16. $X[new,j] = X[new,j] + bw \times random\_number\ [0,1] \times |u[j] - l[j]|$
17. end if
18. end for
19. *Evaluate* (*Xnew*)
20. Add *Xnew*to memory of new harmonies *NEW*
21. end for
22. $HM \cup NEW$
23. Sort *HM* by cost
24. Truncate *HM* to *HMS* harmonies
25. if *it* > 1 then
26. if *HM*(1) = *bestSol* then
27. *stagnation* = *stagnation* + 1
28. else
29. *stagnation* = 0
30. end if
31. if *stagnation* > *MaxIt* $\times$ ς then
32. *reinitializingHarmonyMemory* (*HM*)
33. end if
34. end if
35. *bestSol* = *HM*(1)
36. *bw* = *bw* $\times$ *bw_damp*
37. end for
38. return *bestSol*

5. Computational Experiments

Table 2 shows the fifteen mathematical functions used in this work, their optimal value, and the range of variables used in the experiments performed, the first nine functions were taken from [6], and the last six were taken from [11]. For each algorithm and mathematical function, 30 independent runs were performed. The computer and software configuration is an AMD Ryzen 7 3700U 4-core 8-thread processor and 8 GB of RAM.

Table 2. Mathematical functions used.

Function	Lower Bound	Upper Bound	Global Minimum
Sphere	−100	100	0
Schwefel's 2.22	−10	10	0
Step	−100	100	0
Rosenbrock	−30	30	0
Schwefel's 2.26	−500	500	−12,569.5
Rastrigin	−5.12	5.12	0
Ackleys	−32	32	0
Griewank	−600	600	0
Rotate hyper-ellipsoid	−100	100	0
F4	−100	100	0
F7	−128	128	0
F12	−50	50	0
F13	−50	50	0
F16	−5	5	−1.0316
F17	−5	5	0.398

6. Results

Table 3 shows the results obtained with the proposed hybrid algorithm (HHS-IOBL) and the harmony search algorithm that incorporates the IOBL technique (HS-IOBL). The first column lists the fifteen functions used in the experiments, the second lists the worst, best, and average measurements, and the third and fourth columns show the objective values (OV) and times when the best solution is found (t Best), respectively. The fifth and sixth columns show the objective values (OV) and times when the best solution is found (t Best) of the algorithm against which the comparison is made. A Wilcoxon hypothesis test has been carried out to determine the significance of the results obtained for OV and t best. The seventh and eighth columns show the p-values for OV (p-value 1) and for t best (p-value 2) obtained with the Wilcoxon non-parametric test with 5% significance. The shaded cell in gray represents the lowest objective value (OV) or the shortest time when the best solution is found (t Best), as the case may be. The symbol ↑ represents a significant difference in favor of the reference algorithm (HHS-IOBL). The symbol ↓ represents a significant difference in favor of the algorithm against which it is compared (HS-IOBL). Moreover, the symbol – represents no significant difference in favor of either algorithm.

As can be seen, in quality, the performance of the HHS-IOBL algorithm outperforms the HS-IOBL algorithm by fourteen out of fifteen instances. Only in one case does it have the same performance as the HS-IOBL algorithm. In efficiency, the HHS-IOBL algorithm obtains the following results for the HS-IOBL algorithm: it is better in four instances, in another four, they have the same performance, and in seven instances, the HS-IOBL algorithm is better. Based on these results, it can be affirmed that the HHS-IOBL hybrid algorithm is a competitive alternative to the HS-IOBL algorithm and that incorporating the IOBL technique in the execution process of the particle swarm algorithm generates an improvement in the performance of the algorithm. Even when the HS-IOBL algorithm is better in efficiency, we can affirm that the proposed algorithm reflects a clear superiority in the quality of the results in terms of the objective function.

Table 3. Results obtained with the HHS-IOBL and HS-IOBL algorithms.

Function	Measure	Hybrid Harmony Search IOBL		Harmony Search IOBL		*p*-Value	*p*-Value
		OV	t Best (ms)	OV	t Best (ms)	(OV)	(t Best)
Sphere	Worst	1.63×10^{-172}	78	1.14×10^{-1}	78		
	Best	0	31	4.46×10^{-6}	16		
	Average	6.0541×10^{-174} ↑	46.53	1.21×10^{-2}	33.86 ↓	0.00001	0.00023
Schwefel's 2.22	Worst	2.82×10^{-64}	78	8.16×10^{-2}	78		
	Best	2.98×10^{-192}	15	1.30×10^{-3}	15		
	Average	9.4×10^{-66} ↑	42.06	2.15×10^{-2}	32.3 ↓	0.00001	0.0024
Step	Worst	0	47	0	79		
	Best	0	12	0	15		
	Average	0 –	17.46 ↑	0	31.26	-	0.00001
Rosenbrock	Worst	2.88×10^{1}	78	2.98×10^{1}	93		
	Best	2.77×10^{1}	31	2.90×10^{1}	15		
	Average	2.84×10^{1} ↑	42.06	2.91×10^{1}	33.86 ↓	0.00001	0.00494
Schwefel's 2.26	Worst	-4.21×10^{3}	93	-4.08×10^{3}	78		
	Best	-6.80×10^{3}	15	-5.39×10^{3}	15		
	Average	-5.13×10^{3} ↑	32.13 –	-4.73×10^{3}	32.36	0.00104	0.2177
Rastrigin	Worst	0	31	4.38×10^{-2}	78		
	Best	0	8	1.48×10^{-6}	16		
	Average	0 ↑	16.6 ↑	5.92×10^{-3}	35.46	0.00001	0.00001
Ackleys	Worst	4.44×10^{-16}	33	8.35×10^{-2}	79		
	Best	4.44×10^{-16}	10	1.79×10^{-4}	31		
	Average	4.44×10^{-16} ↑	18.9 ↑	2.20×10^{-2}	35.46	0.00001	0.00001
Griewank	Worst	1	35	1.000924505	78		
	Best	1	0	1.00000000007363	31		
	Average	1 ↑	18.56 ↑	1.000107049	35.93	0.00001	0.00001
Rotate hyper-ellipsoid	Worst	6.67×10^{-173}	94	1.98×10	78		
	Best	0	31	1.36×10^{-5}	31		
	Average	2.47×10^{-174} ↑	53.96	2.31×10^{-1}	37 ↓	0.0001	0.00001
F4	Worst	1.03×10^{-68}	94	1.45×10^{-1}	78		
	Best	8.81×10^{-199}	16	6.73×10^{-4}	16		
	Average	3.44×10^{-70} ↑	45.9	3.41×10^{-2}	33.6 ↓	0.00001	0.00652
F7	Worst	6.86×10^{-4}	63	2.68×10^{-1}	78		
	Best	1.07×10^{-5}	16	4.65×10^{-3}	31		
	Average	1.38×10^{-4} ↑	35.2 –	7.09×10^{-2}	38.6	0.00001	0.20408
F12	Worst	1.28×10^{-1}	94	1.17×10	94		
	Best	6.63×10^{-3}	31	7.23×10^{-1}	16		
	Average	4.16×10^{-2} ↑	57.5	9.52×10^{-1}	38.6 ↓	0.00001	0.00018
F13	Worst	2.97×10^{1}	141	3.13×10^{1}	78		
	Best	2.30×10^{1}	19	2.13×10^{1}	31		
	Average	2.79×10^{1} ↑	62.36	2.97×10^{1}	40.13 ↓	0.00012	0.01108
F16	Worst	−1.031523778	16	−1.030373845	31		
	Best	−1.031628453	0	−1.031627095	0		
	Average	−1.031623571 ↑	5.5 –	−1.031339296	6.73	0.00001	0.50926
F17	Worst	0.39832392	16	0.401305458	16		
	Best	0.397887358	0	0.397888394	0		
	Average	0.397908609 ↑	4.76	0.398294244	4.33 –	0.00001	0.75656

The shaded cell in gray represents the lowest objective value (OV) or the shortest time when the best solution is found (t Best), as the case may be.

7. Conclusions

In this work, we approach the global optimization of real continuous functions. We propose a hybrid harmony search algorithm that incorporates a method of reinitializing harmonies memory using a particle swarm optimization algorithm with an improved opposition-based learning method (IOBL). This approach has been validated by comparing the performance of the proposed algorithm with the performance of the state-of-the-art harmony search algorithm, solving fifteen standard mathematical functions, and applying the Wilcoxon parametric test at a 5% significance level. The results of the computational experiments show that the proposed algorithm outperforms the state-of-the-art algorithm in quality and efficiency.

The main reason for the remarkable performance of the proposed algorithm is that the incorporation of a harmony memory reset mechanism prevents premature convergence of the hybrid algorithm.

Now, we are working to apply this approach to solve problems of other domains and to include a dynamic parameter tuning process in the algorithm.

Author Contributions: Conceptualization: H.J.F.-H.; methodology: E.V.-H.; research: E.V.-H.; software: J.A.B.-H. and M.Á.G.-M.; formal analysis: H.J.F.-H.; writing, proofreading, and editing: H.J.F.-H., J.A.B.-H., M.Á.G.-M. and A.B.-d.-Á. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The source code can be downloaded from https://github.com/JAlfredoBrambila/HybridHarmonySearch (accessed on 13 January 2023).

Acknowledgments: The authors thank CONACYT for supporting the projects with number A1-S-11012 of the Call for Basic Scientific Research 2017–2018 and project number 12397 of the Support Program for Scientific, Technological and Innovation Activities (PAACTI) in order to participate in the Call 2020-1 Support for Scientific Research Projects, Technological Development and Innovation in Health in the face of the Contingency by COVID-19. Alfredo Brambila, Miguel Angel García Morales, and Eduardo Villegas Huerta thank CONACYT for the support 760308, 731279, and 001818, respectively. Héctor Fraire thanks the Tecnológico Nacional de México for supporting the research project 10362.21-P.

Conflicts of Interest: Authors declare that they have no conflict of interest.

References

1. Geem, Z.W.; Kim, J.H.; Loganathan, G.V. A new heuristic optimization algorithm: Harmony search. *Simulation* **2001**, *76*, 60–68. [CrossRef]
2. Askarzadeh, A.; Esmat, R. Harmony Search Algorithm: Basic Concepts and Engineering Applications. In *Recent Developments in Intelligent Nature-Inspired Computing*; Srikanta, P., Ed.; IGI Global: Hershey, PA, USA, 2017; pp. 1–36.
3. Fesanghary, M.; Mahdavi, M.; Alizadeh, Y. Hybridizing harmony search algorithm with sequential quadratic programming for engineering optimization problems. *Comput. Methods Appl. Mech. Eng.* **2008**, *197*, 3080–3091. [CrossRef]
4. Geem, Z.W.; Sim, K.-B. Parameter-setting-free harmony search algorithm. *Appl. Math. Comput.* **2010**, *217*, 3881–3889. [CrossRef]
5. Vasebi, A.; Fesanghary, M.; Bathaee, S.M.T. Combined heat and power economic dispatch by harmony search algorithm. *Int. J. Electr. Power Energy Syst.* **2007**, *29*, 713–719. [CrossRef]
6. Alomoush, A.A.; Alsewari, A.R.A.; Zamli, K.Z.; Alrosan, A.; Alomoush, W.; Alissa, K. Enhancing three variants of harmony search algorithm for continuous optimization problems. *Int. J. Electr. Comput. Eng.* **2021**, *11*, 2088–8708. [CrossRef]
7. Eberhart, R.; Kennedy, J. A new optimizer using particle swarm theory. In Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, 4–6 October 1995; pp. 39–43.
8. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the IEEE International Joint Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; IEEE Press: New York, NY, USA, 1995; pp. 1942–1948.
9. Omran, M.G.H.; Mahdavi, M. Global-best harmony search. *Appl. Math. Comput.* **2008**, *198*, 643–656. [CrossRef]

10. Geem, Z.W. Particle-swarm harmony search for water network design. *Eng. Optim.* **2009**, *41*, 297–311. [CrossRef]
11. Abualigah, L.; Elaziz, M.A.; Sumari, P.; Geem, Z.W.; Gandomi, A.H. Reptile Search Algorithm (RSA): A nature-inspired meta-heuristic optimizer. *Expert Syst. Appl.* **2022**, *191*, 116158. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.