ORIGINAL RESEARCH ARTICLE UHD JOURNAL OF SCIENCE AND TECHNOLOGY

Using the Cuckoo Optimization to Initialize the Harmony Memory in Harmony Search Algorithm to Find New Hybrid (CSHS) Algorithm

Fariaa Abdalmajeed Hameed[1,2], Kanar R. Tariq[1,2], Harith Raad Hasan[1,2], Riyam Alaa Johni[3]

1Technical College of Informatics, Sulaimani Polytechnic University (SPU), Sulaimani, Iraq, [2]Department of Software Engineering, Faculty of Engineering and Computer Science, Qaiwan International University (QIU), Sulaymaniyah, Iraq, [3]Department of Computer Science, Kurdistan Technical Institute (KTI), Sulaymaniyah, Iraq.

ABSTRACT

The current paper seeks to present a hybrid harmony search algorithm, using CS and HS. The proposed algorithm takes advantage of using a method to initialize the harmony memory (HM) with the help of the cuckoo search optimization algorithm. The HS algorithm is inspired by the method used by musicians to create and enhance harmony in music. It follows 3 main principles of pitch adjustment (PA), HM consideration (HMC), and random selection (RS). The performance of HS can be affected by its poor accuracy in optimization and speed of convergence, 2 main issues with it. However, to improve the HS algorithm, Cuckoo search can be used as it can do a local search using one parameter only, other than the size of the population, making it work more efficiently. This current method has been tested for its validity and efficiency in performance by being implemented on many international standard optimization problems. The results confirm the fact that the performance of the currently proposed algorithm is better and more efficient in finding solutions compared to HS and other algorithms.

Index Terms: Optimization, Cuckoo Search, Harmony Search Algorithm, Harmony Memory, Hybrid Algorithm

1. INTRODUCTION

Many meta-heuristic search algorithms have been developed over the past decades and have proven useful in solving various optimization problems in technical fields. Such as genetic algorithm-simulated annealing, ant colony optimization, particle swarm optimization, bee algorithm, firefly algorithm [1], bat inspired approach (BIA) [2]. The main reasons metaheuristic algorithms are more popular among researchers than traditional optimization techniques, such as gradient-based techniques are algorithmic simplicity, ease of implementation, and versatility of solutions [3].

Access this article online

DOI:10.21928/uhdjst.v8n1y2024.pp99-107 | E-ISSN: 2521-4217 P-ISSN: 2521-4209

However, when solving some complex optimization problems, such as discrete structure optimization [4], water supply network construction [5], and vehicle routing problems [6]. Conventional optimization algorithms cannot provide reasonable and accurate solutions in a limited time.

In this article, we combine two metaheuristic algorithms, Harmony Search (HS) and Cuckoo Search Optimizer (CS), to produce a hybrid CSHS algorithm. Comparing eight benchmark optimization problems of the proposed algorithm with standard Cuckoo Search CS, Bat Algorithm BA, and standard Harmony Search HS, we find that CSHS performs better on most benchmark optimization problems than other algorithms where HS is prone to local minima. To alleviate this drawback, CS [7] can be used to perform local searches more efficiently.

In the last few years, various meta-heuristic algorithms have been created, and these algorithms have successfully

Corresponding author's e-mail: fariaa.hameed@spu.edu.iq

Received: 05-02-2024 Accepted: 19-03-2024 Published: 07-04-2024

settled problems related to optimization in different technical areas. These algorithms include genetic algorithm simulated annealing, particle swarm optimization, bee algorithm, ant colony optimization, firefly algorithm [1], bat inspired approach (BIA) [2]. There are a few reasons why metaheuristic algorithms are preferred over traditional optimization techniques. The metaheuristic algorithms are simple, and they can be easily applied. Furthermore, they provide comprehensive solutions that can be adapted to many different situations. When it comes to more complicated optimization problems such as vehicle routing problems, metaheuristic algorithms can provide reasonable solutions, while conventional algorithms cannot provide precise and logical solutions in a short time.

This article seeks to take advantage of using both Cuckoo Search and HS Optimizer to create a hybrid CSHS algorithm. Comparing eight benchmark optimization problems of the proposed algorithm with standard CS, standard BA, and standard HS, we find that CSHS performs better on most benchmark optimization problems than other algorithms where HS is prone to local minima. To alleviate this drawback, CS [7] can be used to perform local searches more efficiently.

2. METAHEURISTIC ALGORITHMS

2.1. Harmony Search (HS) Algorithm

The Harmony Search (HS) algorithm was recently introduced by Geem *et al.* It is believed to be a powerful population-based metaheuristic algorithm [8]. This search algorithm is influenced by what musicians do to achieve harmonic state through repeatedly adjusting their instruments' pitch. This algorithm takes the advantage of being simple as it does not need some information like the gradient of the objective function.

However, the HS may have slower convergence rates than other optimization techniques, especially in complicated and high-dimensional search spaces. This can have an influence on both the algorithm's performance and efficiency. Fine-tuning factors such as harmony memory (HM) size, pitch adjustment rate, and bandwidth can be difficult to determine and may need extensive trial and error in a number of problem areas.

HS was originally created to solve continuous optimization challenges. Adapting it to address discrete or combinatorial optimization problems may need additional techniques or adjustments. Furthermore, the quality of the initial HM can dramatically impact the algorithm's effectiveness.

In this optimization process, there are 3 factors: harmonic memory (HM), pitch adjustment rate (PAR), and distance bandwidth (BW). Besides these 3 factors, the size of the HM and the number of improvisations (NI) are considered factors. Having been randomly generated, the primary population of harmony vectors is stored in HM. Later, new harmony candidates are generated using all the solutions in the HM. This is done by the memory consideration rule, pitch adjustment rule, and random reinitialization. In the end, the new candidate harmony is compared to the worst harmony vectors to update the HM. If the worst harmonic vector is better than the worst harmonic vector in HM, it is replaced by a new candidate vector. This is repeated until the desired criteria are achieved. Basically, the HM algorithm includes 3 fundamental stages: initialization, harmony vector improvisation, and HM update. All these stages are elaborated on as follows [9]:

Step1. Initialize the optimization problem and algorithm parameters.
Step2. Initialize the HM.
Step3. Improvise a new harmony from the HM.
Step4. Update the HM.
Step5. Repeat Steps 3 and 4 until the termination criterion is satisfied. Figs. 1 and 2 show the Harmony Search flowchart and pseudocode, respectively.

2.2. Cuckoo Search (CS)

The new metaheuristic algorithm called Cuckoo Search (CS) is intended to solve optimization-related problems that align with the cuckoo species' distinctive parasitic behavior, predatory flight behavior, and fruit fly. However, CS, like any algorithm, has restrictions, such as requiring several parameters, such as the cuckoo population size, the chance of egg laying, and the step size for random walks. These parameter choices might have an impact on the algorithm's performance, thus obtaining the best values may necessitate some tinkering. Furthermore, balancing exploration and exploitation is a prevalent problem in optimization algorithms. The basic Cuckoo Search algorithm may not completely use historical data or previous iterations to steer the search. Improved memory and learning techniques may be required to improve the algorithm's performance.

In particular, each egg in CS represents a new solution; the cuckoo's flight determines its walking steps, and the objective is to replace an unsuccessful solution in the nest with an optimal and better one. That is, an egg is in every nest. The technique can be extended to more complicated scenarios in which there are numerous eggs in each nest (and thus numerous solutions). For instance, equation (1) below is used

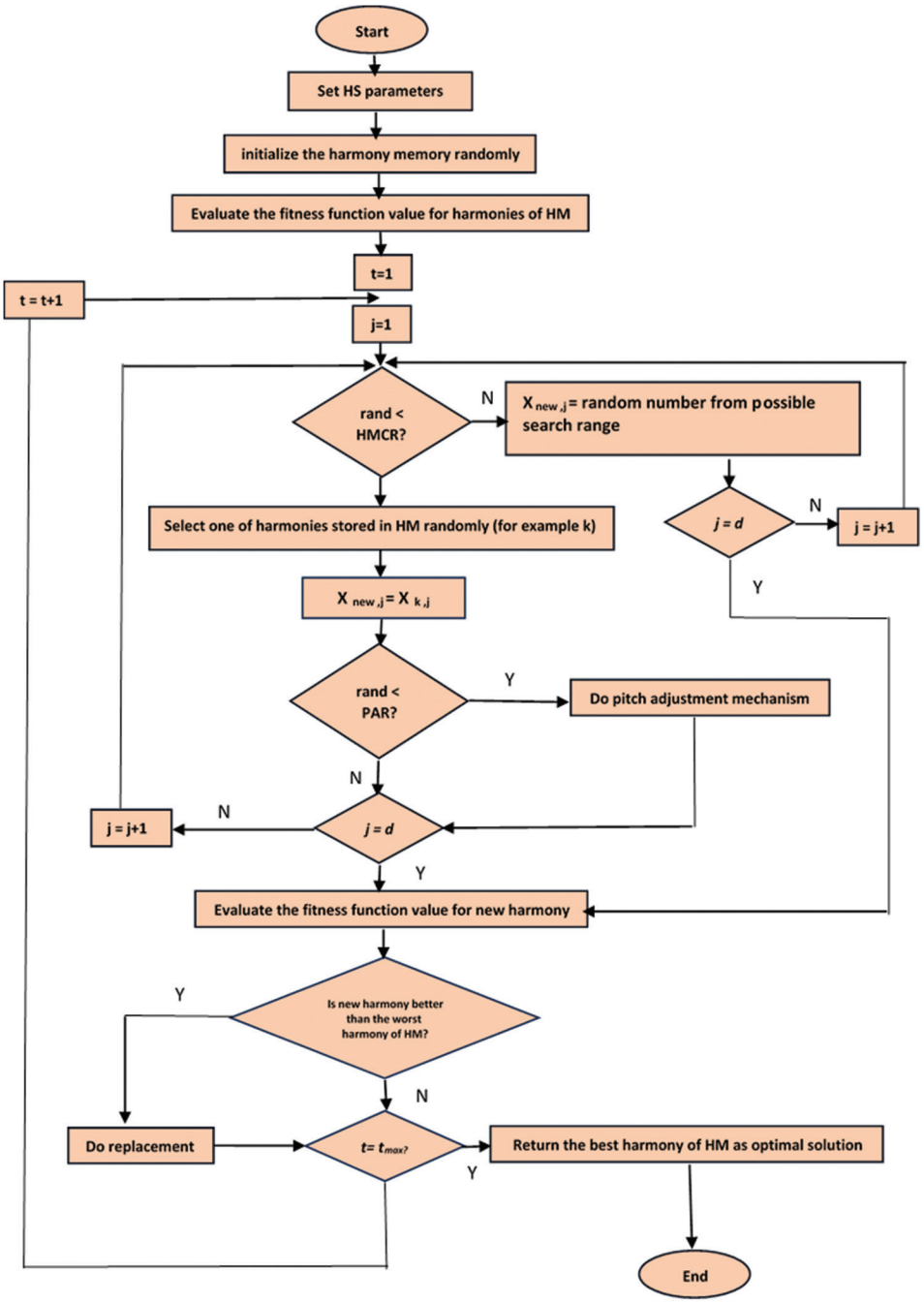

Fig. 1. Flow chart of the harmony search algorithm.

to carry out the Levy flight when creating a new solution x(t+1) for Cuckoo I.

$$\boldsymbol{X}_{\boldsymbol{i}}^{(\boldsymbol{t}+1)} = \boldsymbol{X}_{\boldsymbol{i}}^{(\boldsymbol{t})} + \boldsymbol{\beta} \oplus \boldsymbol{Le}^{\boldsymbol{yy}}(\ \boldsymbol{\lambda}) \qquad (1)$$

The step size associated with the problem scales under examination is represented by $\alpha > 0$ in equation (1) above; in most circumstances, $\alpha = 1$ can be utilized. The product $\oplus$ denotes entry-wise multiplications, whose product is comparable to the PSO; however, because the length of the step is greater in the long run, the random walk-through Levy flight performs better in terms of search step exploration. Equation (2) essentially depicts a random walk that is provided by the Levy flights:

```
Pseudo code of the Harmony Search algorithm (HSA)
Begin;
   Define objective function f(x), x=(x1 ,x2 ,…,xd )^T
   Define Harmony Memory Considering rate (HMCR)
   Define Pitch adjusting rate (PAR) and other parameters
   Generate Harmony Memory with random harmonies
   while (t<max number of iterations)
      while (i<=number of variables)
         if (rand<HMCR),
           Choose a value from HM for the variable i
            if (rand<PAR),
              Adjust the value by adding certain amount
            end if
          else
           Choose a random value
          end if
      end while
       Accept the New Harmony (solution) if better
    end while
    Find the current best solution
end
```

Fig. 2. Pseudocode of the harmony search algorithm.

$$Le^{vy}\left(\lambda\right) \sim u = t^{-\lambda}, \left(1 < \lambda \leq 3\right) \quad (2)$$

The above equation (2) is an infinite variance, having an infinite mean. Figs. 3 and 4 demonstrate the flowchart and the pseudocode of Cuckoo Search.

3. PROPOSED METHOD

Efficiency has turned into an indispensable desired quality in today's world. Therefore, almost all manufacturers strive to gain the highest efficiency possible. As a result, optimization is now valued and wanted more than ever. The ultimate goal is to seek the optimal values of some decision variables to achieve the best solutions to deal with some optimization problems. Instead of using the traditional method of creating and initializing the population of harmony vectors at random and storing them in the HM, the proposed method uses Cuckoo Search, which offers both local and global search to find the best solution. Thus, local solutions are maintained thanks to the harmony search algorithm's HM and pitch adjustment rate. The steps that follow introduce the key steps.

Step 1: Initialize the optimization problem and algorithm parameters.

Step 2: Using CS to Initialize the HM.

Step 3: Improvise a new harmony from the HM.

Step 4: Update the HM.

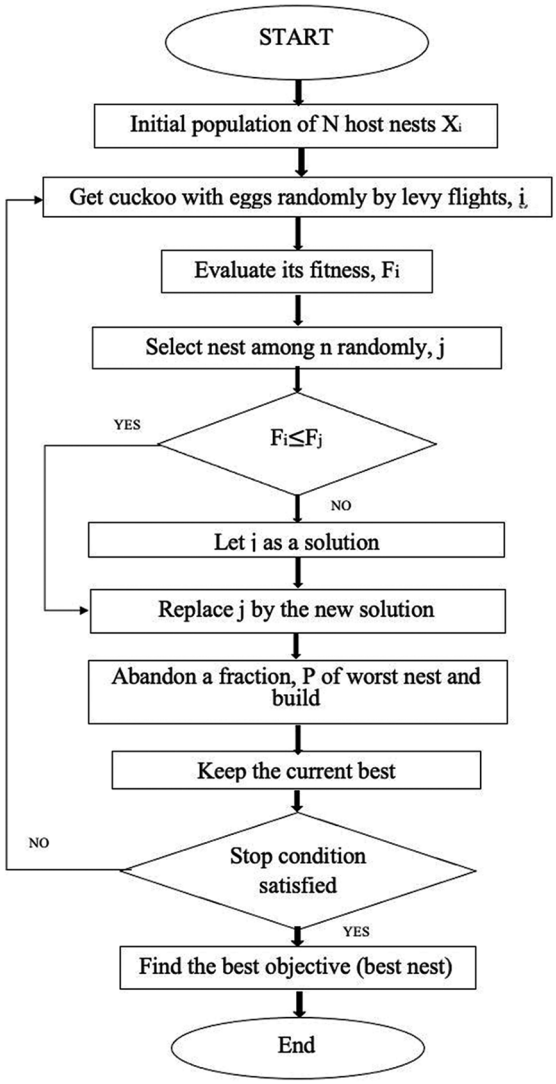

Fig. 3. Cuckoo algorithm flowchart [10].

Step 5: Repeat Steps 3 and 4 until the termination criterion is satisfied.

Figs. 5 and 6 show the CSHS flowchart and pseudocode.

4. EXPERIMENTAL SETUP AND RESULT

4.1. Experimental Setup

In this part, the studies that were carried out to determine and evaluate the performance of the proposed hybrid harmony algorithm and its variations are described for several analytical benchmark functions. Where it is applied to 8 minimization and maximization standard benchmark functions [11] as detailed in Table 1, Dimension (n) and Range are the feasible

bound of function, respectively. To have a balanced outcome, all the possible methods are used in a computer equipped with a Windows 10 OS, an Intel(R) Core (TM) i5-8350U CPU @ 1.70GHz 1.90 GHz, and Python used as a programming language.

```
Objective function f(X), X = (f(x1, x2, ..., xd)^T
Generate initial population of n host nests Xi (i=1, 2, ..., n)
While t < Max_itertions do
    Get a cuckoo randomly by Levy flights
    Evaluate its quality/ fitness Fi
    Choose a nest among n (say, j) randomly
    If Fi > Fj then
        replace j by the new solution;
    End If
    A fraction (Pa) of worse nests are abandoned and
    new ones are built;
    Keep the best solutions
    Rank the solutions and find the current best
End While
Postprocess results and visualization
```

Fig. 4. Pseudocode of the Cuckoo search algorithm

TABLE 1: Numerical benchmark functions

Test function	Dimension (n)	Range
F1	30	[−100, 100]
F2	3	[−5.12, 5.12]
F3	2	[−32.768, 32.768]
F4	10	[−30,30]
F5	2	[−500, 500]
F6	2	[−100, 100]
F7	3	[−5.12, 5.12]
F8	2	[−1.28, 1.28]

1. SpShpere (n variables) (F1):

$$SP_n(x) = \sum_{i=1}^{n} x_i^2 \tag{3}$$

2. Rastrigin's Function (F2):

$$f(x) = 10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{4}$$

3. Ackley's Function (F3):

$$f(x) = -a * \exp\left(-b * \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right)$$

$$-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1) \tag{5}$$

Where:
a= 20
b = 0.2
c = 2*π

4. Rn Rosen Brock (F4):

$$f(x) = \sum_{i=1}^{n-1}\left[100(x_{i+1} - x_i^2)^2 + (1 + x_i)^2\right] \tag{6}$$

5. Schwefel's function (F5)

$$f(x) = \sum_{i=1}^{n}\left[-x_i \sin\left(\sqrt{|x_i|}\right)\right]. \tag{7}$$

TABLE 2: Summarize results for all benchmarks

Benchmark	n		HS	CS	BA	CSHS
F1	30	max	4.8265	78	8.365	0.74979
		min	0.0123	0.0000462	0.6124	0.66048
F2	3	max	1.45501	0	1.2301	12.0736
		min	0	0	0	0.01733
F3	2	max	0.650521303	6.91E-01	0. 521303	2.6489
		min	6.1303E-19	6.50E-19	6.103E-15	0.025
F4	10	max	−0.418 e005	−0.418 e005	−0.418 e005	−0.409 e005
		min	−0.418 e005	−0.358 e005	−0.418 e005	−0.398 e005
F5	2	max	−837.965	−595.2761	−887.900	−429.42
		min	−837.9657	−710.327	−337.9657	−837.946
F6	2	max	1.7147	0.0298	1.847	0.96028
		min	0.2	0.00064	0.2	0.018245
F7	3	max	0.97989	1.7147	0.93389	1.2358
		min	0.000511	0.2	0.000511	0.0099
F8	2	max	−0.802−1	−0.265	−0.692	−0.0002
		min		−0.38	−1	−0.231

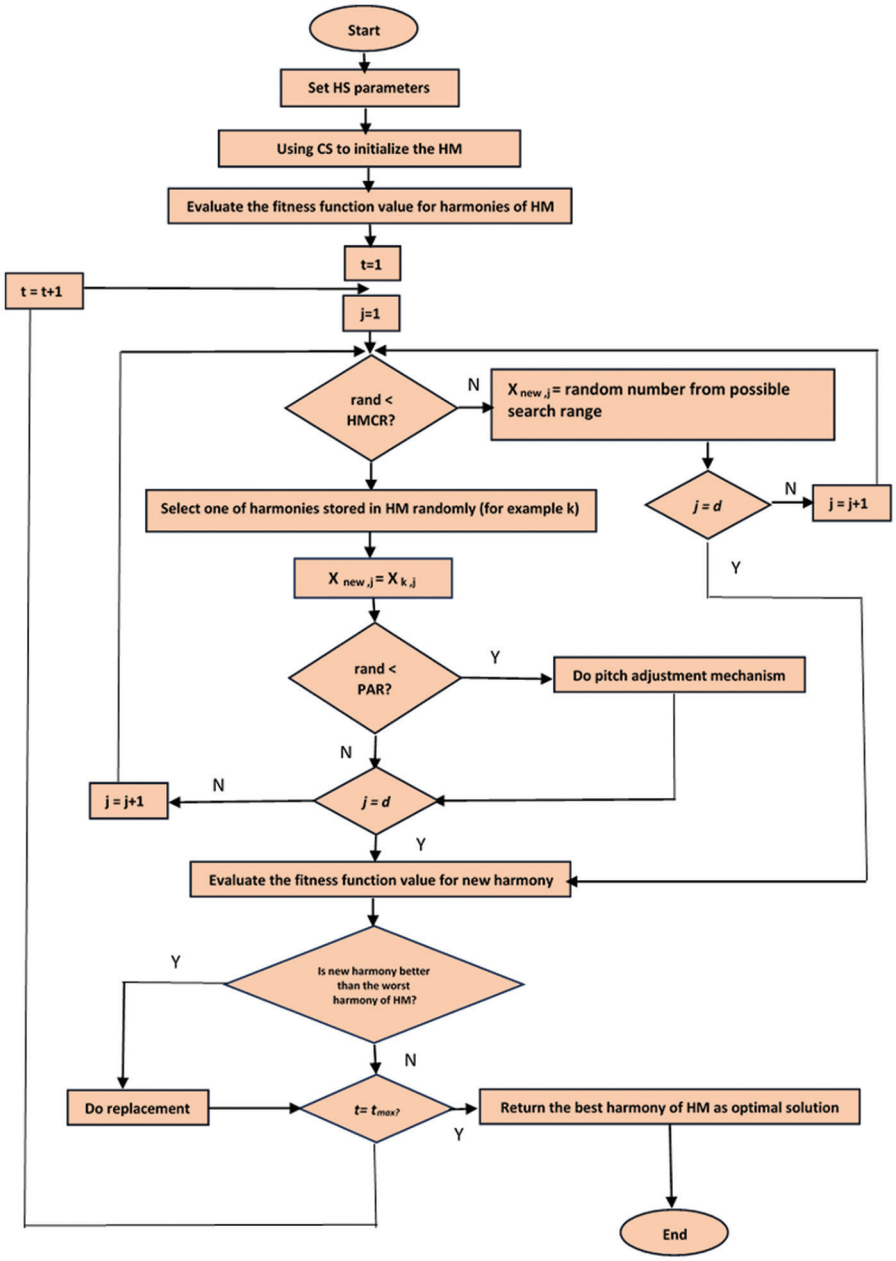

Fig. 5. CSHS flowchart.

6. Step Function (F6)

$$f(x) = \sum_{i=1}^{n} (x_i + 0.5)^2 . \quad (8)$$

7. Generalized Restringing's Function (F7)

$$f(x) = \sum_{i=1}^{n} [x_i^2 - 10 \cos(2\pi x_i) + 10] \quad (9)$$

8. Quadric (F8)

$$f(x) = \sum_{i=1}^{n} i x_i^4 + random(0,1) \quad (10)$$

4.2. Experimental Result

The performance of CSHS on 8 benchmark functions is compared to 3 other approaches called BA, CS [12], HS [13], [14] to verify its performance. The results of the

TABLE 3: The average of the best value on F2 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F2	2000	0.633	1.45501	0.161	0.8930	0.124	0.141	0.311	0.281
	3000	0.786	0.00106	0.7729	0.095	0.188	0.188	0.138	0.359
	4000	0.161	0	0. 143	0.261	0.218	0.219	0.2106	0.421
	5000	0.775	0	0.5626	0.318	0.218	0.218	0.205	0.483

TABLE 4: The average of the best value on F3 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F3	2000	6.50503E-1	6.50303E-1	4.418 e005	0.67566	0.171	0.171	0.141	0.312
	3000	6.51303E-19	6.3303E-1	3.418 e005	0.3746	0.187	0.218	0.188	0.39
	4000	6.591303E-19	6.91303E-1	4.418 e005	0.0280	0.234	0.218	0.219	0.452
	5000	6.1303E-19	6.34903E-1	6.418 e005	0.025	0.266	0.266	0.218	0.468

TABLE 5: The average of the best value on F4 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F4	2000	−0.418 e005	−0.411 e005	−0.413 e005	−0.412 e005	0.156	0.071	0.155	0.112
	3000	−0.418 e005	−0.362 e005	−0.315 e005	−0.415 e005	0.171	0.203	0.218	0.359
	4000	−0.418 e005	−0.362 e005	−0.417 e005	−0.415 e005	0.171	0.203	0.218	0.359
	5000	−0.418 e005	−0.362 e005	−0.615 e005	−0.415 e005	0.249	0.218	0.266	0.421

TABLE 6: The average of the best value on F5 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F5	2000	−837.9657745	−595.2761	−395.761	−837.246	0.156	0.171	0.124	0.312
	3000	−837.965	−684.648	−954.648	−837.842	0.187	0.172	0.188	0.375
	4000	−837.965	−623.720	−603.70	−837.884	0.218	0.219	0.218	0.405
	5000	−837.965	−699.914	−749.14	−837.798	0.266	0.234	0.218	0.437

TABLE 7: The average of the best value on F6 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F6	2000	0.20	0.0055	0.84	0.2662	0.141	0.187	0.134	0.214
	3000	0.37	0.0298	0.9052	0.0692	0.156	0.188	0.103	0.172
	4000	0.84	0.00064	0.75	0.9052	0.187	0.172	0.298	0.374
	5000	0.75	0.00155	0.0298	0.4888	0.203	0.218	0.234	0.39

TABLE 8: The average of the best value on F7 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F7	2000	0.131	0.20	0.122	0.6351	0.172	0.156	0.311	0.228
	3000	0.142	0.39	0.192	0.00990	0.187	0.203	0.138	0.043
	4000	0.187	0.54	0.387	0.01149	0.218	0.218	0.2106	0.39
	5000	0.203	0.95	0.293	0.0526	0.25	0.219	0.205	0.468

TABLE 9: The average of the best value on F8 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F8	2000	−1	−0.299	−0.262	−0.153	0.0452	0.03355	0.228	0.078
	3000	−1	−0.362	−0.310	−0.231	0.06865	0.0484	0.033	0.1162
	4000	−0.802	−0.340	−0.680	−0.0002	0.0905	0.0655	0.039	0.1498
	5000	−0.802	−0.380	−0.802	−0.0003	0.11235	0.07955	0.468	0.18565

TABLE 10: The average of the best value on F1 benchmark function with the iterations 2000,3000,4000,5000

Function	Iteration	HS	CS	BA	CSHS	Time CS	Time HS	Time BA	Time CSHS
F1	2000	3.727	0.87	0.5626	0.71727	0.11	0.078	0.1638	0.56005
	3000	0.5626	008	0.29	0.69163	0.1638	0.11465	0.2106	0.8502
	4000	2.9317	0.29	0.69143	0.68829	0.2106	0.1529	0.0708	1.0975
	5000	1.3834	078	0.5626	0.69143	0.26205	0.1903	0.31465	1.40325

Pseudocode of the HSCS
Begin;
Define objective function $f(x)$, $x=(x_1, x_2.....,x_d)^T$
Define Harmony Memory Considering rate(HMCR)
Define Pitch adjusting rate (PAR)and other parameters.
Generate Harmony Memory with the Cuckoo Search algorithm
Begin;
Generate initial population of n host nests X_i (i =1,2,…….,n)
While t < Max_itertions do
Get a cuckoo randomly by Levy flights
Evaluate it's quality/fitness F_i
Choose a nest among n *(say, j)* randomly
If $F_i > F_j$ **then**
replace j with the new solution;
End if
A fraction *(Pa)* of worse nests are abandoned and new ones are built;
Keep the best solutions
Rank the solutions and find the current best
End while
Postprocess results and visualization
//End generation of the Harmony Memory
while (t< max number of iterations)
while (i<=number of variables)
if (*rand*< HMCR)
Choose a value from HM for the variable i
if (*rand* <PAR),
Adjust the value by adding a certain amount
end if
else choose a random value
end if
end while
Accept the new harmony 9solution) if better
end while
Find the current best solution
end

Fig. 6. Pseudocode of the HSCS algorithm.

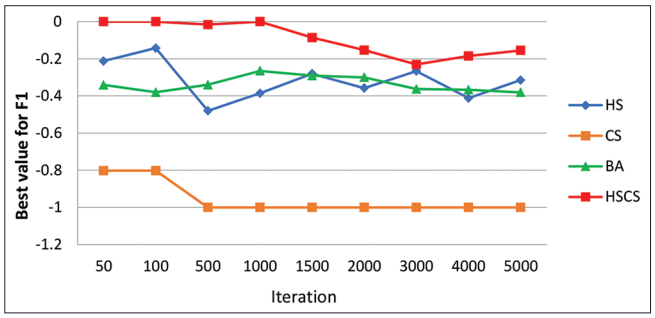

Fig. 7. The Performance of HS, CS, BA, HSCS on test function F1.

benchmark functions used in this experiment are illustrated in Table 2. Here, n shows the function dimension and min and max represent minimum value and maximum value, respectively.

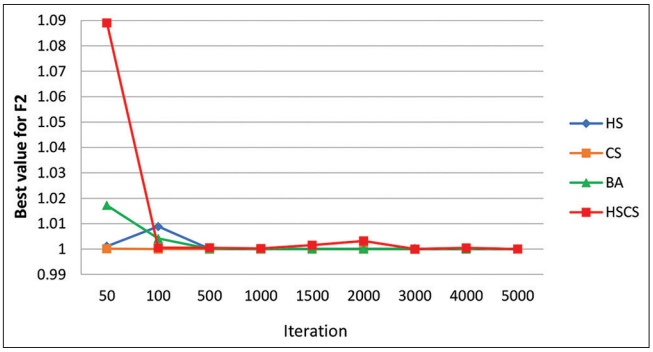

Fig. 8. The Performance of HS, CS, BA, HSCS on test function F2.

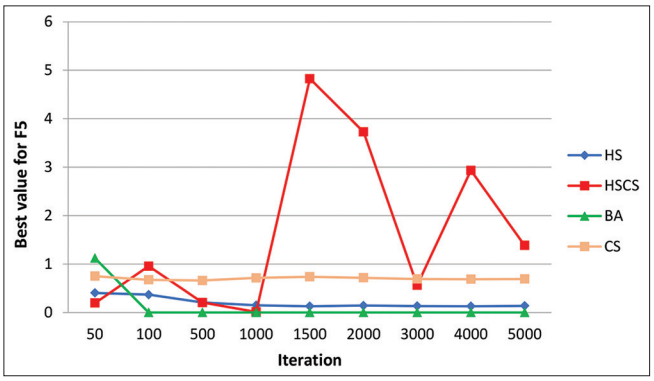

Fig. 9. The Performance of HS, CS, BA, HSCS on test function F5.

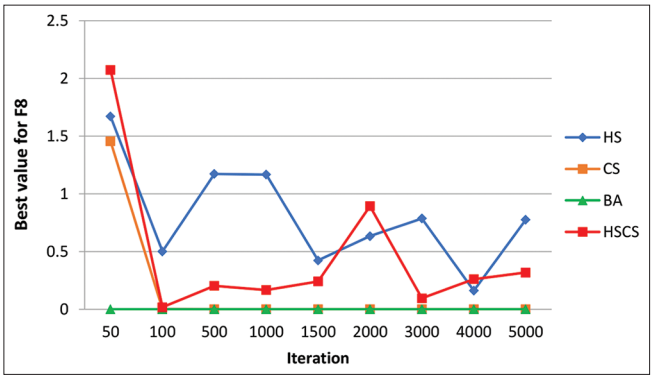

Fig. 10. The Performance of HS, CS, BA, HSCS on test function F8.

This research considers the time component in addition to all the information displayed in Tables 2-10, where different

numbers of iterations have been applied to the optimization methods.

In addition, the most representative convergence curves are provided (Figs. 7-10). The values in the figures represent the average functional optimum, with being the true value. From Fig. 7, we can see that HSCS is better at finding solutions than any other method, especially in F1, F2, F5, and F8.

5. CONCLUSION

The CSHS algorithm outperformed the other three classical optimization algorithms, CS, BA, and HS, in terms of obtaining the best value or minimum values in the majority of the benchmark functions, as can be seen from the tables above and Table 2's results. This is because the proposed algorithm completed local searches more quickly than the other two classical algorithms. The primary advantage of Cuckoo Search is that it requires relatively few parameters – just one, the probability function – and the population size. In comparison to other metaheuristic algorithms, this makes the Cuckoo search algorithm highly straightforward and efficient. It enhances the process by merging HS and CS.

ACKNOWLEDGMENT

The authors would like to acknowledge the Sulaimani Polytechnic University, Qaiwan International University, and Kurdistan Technical Institute for their financial support of this research. Thanks also go to the reviewers of this paper for their constructive comments.

DATA AVAILABILITY

All data included in this study are available on request by contact with the corresponding author.

CONFLICTS OF INTEREST

The authors declare that there are no conflicts of interest regarding the publication of this paper.

REFERENCES

[1] X. S. Yang. "*Nature-inspired Metaheuristic Algorithms*". Luniver Press, United Kingdom, 2010. Available from: https://books.google.iq/books [Last accessed on 2024 Feb 05].

[2] X. S. Yang. A new metaheuristic bat-inspired algorithm. In: "*Nature Inspired Cooperative Strategies for Optimization (NICSO 2010)*". Springer, Berlin, Heidelberg, pp. 65-74, 2010.

[3] X. S. Yang. "Nature-inspired mateheuristic algorithms: Success and new challenges". *Journal of Computer Engineering and Information Technology*, vol. 1, pp. 1-3, 2012.

[4] K. S. Lee, Z. W. Geem, S. Lee and K. Bae. "The harmony search heuristic algorithm for discrete structural optimization". *Engineering Optimization*, vol. 37, no. 7, pp. 663-684, 2005.

[5] Z. W. Geem. "Optimal cost design of water distribution networks using harmony search". *Engineering Optimization*, vol. 38, no. 3, pp. 259-277, 2006.

[6] Z. W. Geem, K. S. Lee and Y. Park. "Application of harmony search to vehicle routing". *American Journal of Applied Sciences*, vol. 2, no. 12, pp. 1552-1557, 2005.

[7] R. G. Babukartik and P. Dhavachelvan. "Hybrid algorithm using the advantage of ACO and Cuckoo search for job scheduling". *International Journal of Information Technology Convergence and Services*, vol. 2, no. 4, pp. 25-34, 2012.

[8] Z. W. Geem, J. H. Kim and G. V. Loganathan. "A new heuristic optimization algorithm: Harmony search". *Simulation*, vol. 76, no. 2, pp. 60-68, 2001.

[9] 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, no. 36, pp. 3902-3933, 2005.

[10] M. Shehab, A. T. Khader and M. A. Al-Betar. "A survey on applications and variants of the cuckoo search algorithm". *Applied Soft Computing*, vol. 61, pp. 1041-1059, 2017.

[11] "*Evolutionary Programming Made Faster*". IEEE Xplore. Available from: https://ieeexplore.ieee.org/abstract/document/771163 [Last accessed on 2024 Feb 04].

[12] X. S. Yang and S. Deb. "*Cuckoo Search Via Lévy Flights*". IEEE Xplore, 2009. Available from: https://ieeexplore.ieee.org/document/5393690 [Last accessed on 2024 Feb 04].

[13] J. H. Kim and Z. W. Geem. "*Harmony Search Algorithm: Proceedings of the 2nd International Conference on Harmony Search Algorithm (ICHSA2015)*". Springer, Berlin, 2015. Available from: https://books.google.iq/books [Last accessed on 2024 Feb 05].

[14] Y. Feng, G. G. Wang and X. Z. Gao. "A novel hybrid Cuckoo search algorithm with global harmony search for 0-1 knapsack problems". *International Journal of Computational Intelligence Systems*, vol. 9, no. 6, pp. 1174-1190, 2016.