Cooperative Coevolution using The Brain Storm Optimization Algorithm

Mohammed El-Abd
Electrical and Computer Engineering Department
American University of Kuwait
Email: melabd@auk.edu.kw

***Abstract*—The Brain Storm Optimization (BSO) algorithm is a powerful optimization approach that has been proposed in the past few years. A number of improvements have been previously proposed for BSO with successful application for real-world problems. In this paper, the implementation of a Cooperative Co-evolutionary BSO (CCBSO) algorithm that is based on the explicit space decomposition approach is investigated. The improved performance of CCBSO is illustrated based on experimental comparisons with other BSO variants on a library of 20 well-known classical benchmark functions.**

I. Introduction

The concept of cooperative algorithms was proposed to have a more efficient search mechanism in comparison to the individual instances of the same algorithm. Cooperation is based on having multiple algorithm instances searching the search space (either serially or in parallel) and exchanging information useful information. This implies that the search space is decomposed between the different algorithms [1], [2]. The decomposition is either *explicit* by dividing the problem variables between the different cooperating instances or *implicit* by having the different instances search different parts of the search space due to different initialization, different parameters settings or both.

The Cooperative Co-evolutionary approach adopted in this work has been investigated before using different algorithms including Genetic Algorithms (GAs) [3], Particle Swarm Optimization (PSO) [4], the Artificial Bee Colony (ABC) [5] and the Bacterial Foraging Optimization Algorithm (BFOA) [6], [7].

The Brain Storm Optimization (BSO) algorithm [8], [9] is the first population-based algorithm in which the population update process is designed to mimic a human activity, namely the brainstorming process. In a brainstorming session, a number of experts of different backgrounds and different abilities sit together in order to come up with ideas to solve a certain problem. Following such a process helps in successfully solving the tackled problem. Disadvantages of BSO include the need to set the number of clusters before hand, the time complexity of the clustering process, the absence of a re-initialization stage, and the fixed schedule for updating the step size. Some of these shortcomings have already been addressed in the literature. However, to the best of our knowledge, BSO has not been adopted in a cooperative environment before although cooperative implementations of other algorithms have demonstrated an excellent performance.

In this paper we adopt BSO as the algorithmic instance in a cooperative co-evolutionary process. The proposed Cooperative Co-evolutionary BSO (CCBSO) is compared against three recent variants of BSO using a suite of 20 well-known benchmark functions for increased problem sizes.

The rest of the paper is organized as follows: Section II gives details about the BSO algorithm. Previous improvements proposed in the literature to improve BSO are covered in Section III. The cooperative co-evolutionary process is explained in Section IV. Section V presents the experimental study and the reached results. Finally, the paper is concluded in Section VI.

II. Brain Storm Optimization

In BSO, each individual solution is referred to as an *idea*. Hence, a population of ideas (solutions) is being updated from one iteration to the next. At the beginning, ideas are randomly initialized in the search domain. In a single iteration, similar ideas are grouped together using k-means clustering. During the clustering process, the best idea in each cluster is saved as the *cluster center*. A new idea is generated through one of four different operations: following a randomly selected cluster center, following a randomly selected idea from a randomly selected cluster, combining two randomly selected cluster centers, or combining two randomly selected ideas from two randomly selected clusters. The generation method used is randomly chosen based on multiple probabilities $p_{one-cluster}$,$p_{one-center}$, and $p_{two-centers}$.

The generated idea is then perturbed using Gaussian random numbers. Finally, the newly generated idea replaces the current one if it has a better fitness. Otherwise, it will be discarded. The algorithm for BSO is shown in Fig. 1.

Note that n is the population size, m is the number of clusters, $N(\mu, 0)$ represents a Gaussian distribution with mean μ and a standard deviation of 0. $rand$ is a uniformly distributed random number between 0 and 1. Finally, ξ is a dynamically updated step-size and k is for changing the slope of the $logsig$ function. For more on BSO, interested readers can refer to [10].

III. Previous BSO Improvements

A number of improvements have been proposed in the literature to overcome the previously mentioned shortcomings of BSO and to enhance its performance. The authors in

Algorithm 1 The BSO algorithm

Require: $Max\_Iterations$, n, m, $p_{one-cluster}$, $p_{one-center}$, and $p_{two-centers}$

Randomly initialize n ideas
Evaluate the n ideas
$iter$ = 1
while $iter \leq Max\_Iterations$ **do**
  Cluster n ideas into m clusters
  Rank ideas in each cluster and select cluster centers
  for each idea i **do**
    if $rand < P_{one-cluster}$ **then**
      Probabilistically select a cluster c_r
      if $rand < P_{one-center}$ **then**
        $nidea^i = center_{c_r}$
      else
        Randomly select an idea j in cluster c_r
        $nidea^i = idea^j_{c_r}$
      end if
    else
      Randomly select two clusters c_{r1} and c_{r2}
      Randomly select two ideas c^j_{r1} and c^k_{r2}
      $r = rand$
      if $rand < P_{two-centers}$ **then**
        $nidea^i = r \times center_{c_{r1}} + (1 - r) \times center_{c_{r2}}$
      else
        $nidea^i = r \times idea^j_{c_{r1}} + (1 - r) \times idea^k_{c_{r2}}$
      end if
    end if
    $\xi = rand \times logsig(\frac{0.5 \times Max\_Iterations - Current\_Iteration}{k})$
    $nidea^i = nidea^i + \xi \times N(\mu, 0)$
    if fitness($nidea^i$) > fitness($idea^i$) **then**
      $idea^i$ = $nidea^i$
    end if
  end for
end while
return best idea

[11] proposed using a Simple Grouping Strategy (SGM) to minimize the computational cost of k-means clustering. In addition, they used an idea difference approach to perturb the newly generated ideas instead of Gaussian random numbers. Their approach removed the parameter ξ and replaced it with another parameter p_r that controls injecting the open minded element into the idea creation process.

The authors in [12] proposed to update the ξ parameter using the dynamic range of the ideas in each dimension. Moreover, a total of $3n$ ideas were created in each iteration and then evaluated and replaced in a batch mode. Updating the entire populations was done by randomly grouping the $4n$ ideas (n current ideas and $3n$ new ideas) into n groups of 4 ideas each. Finally, the best idea in each group was copied to the next generation.

The authors in [13] introduced the idea of random grouping to minimize the clustering overhead. They also proposed a new dynamic step-size schedule for updating the parameter ξ as follows:

$$\xi = rand \times e^{1 - \frac{Max\_Iterations}{Max\_Iterations - Current\_Iteration + 1}} \quad (1)$$

To enhance the population diversity of BSO, the authors in [14], [15] proposed partially re-initializing the ideas every a predetermined number of iterations. Multiple experiments were carried to study the effect of changing the number ideas being re-initialized by setting to a fixed number, an increasing number, or a decreasing number. In general, the performance of BSO was improved by their proposed population diversity enhancement strategy.

The authors in [16] introduced a new idea generation mechanism borrowed from differential evolution, referred to as BSODE, while also using the step-size schedule proposed in [13]. In their work, if the generated idea was to follow a randomly selected idea from a randomly selected cluster cr, this operator was modified as follows (where F is the mutation scaling factor while r_1 and r_2 are mutually exclusive integers randomly chosen in the selected cluster):

$$nidea^i = center_{cr} + F \times (idea^{r1}_{cr} - idea^{r2}_{cr}) \quad (2)$$

However, if the generated idea was two follow a combination of two randomly selected ideas j and k from two randomly selected clusters $cr1$ and $cr2$, this operator was modified as follows (where F is defined as above while $Global\_Idea$ is the best idea in the population):

$$nidea^i = Global\_Idea + F \times (idea^j_{cr1} - idea^k_{cr2}) \quad (3)$$

To dynamically set the number of clusters, the authors in [17] proposed the use of Affinity Propagation (AP) clustering to take the clusters structure information into account.

The work in [18] borrowed concepts from the Artificial Bee Colony (ABC) algorithm [19] by starting a re-initialization process when a certain idea has not been improved for a specified number of iterations, $Threshold$. In addition, the idea could be either randomly re-initialized or using a differential approach as follows (where F is defined as above while r_1, r_2 and r_3 are mutually exclusive integers randomly chosen between 1 and n):

$$nidea^i = idea^{r1} + F \times (idea^{r2} - idea^{r3}) \quad (4)$$

The author also modified equation (1) by multiplying it with a parameter α that is proportional to the search space size.

IV. Cooperative Co-evolution

In the Cooperative Co-evolution (CC) framework, the problem being tackled is divided into smaller sub-problems. Each sub-problem consists of a sub-set of the problem variables and is being optimized a different instance of the same algorithm (sub-optimizer). In this scenario, each sub-optimizer will be handling a smaller number of problem variables and will be providing a part of the overall solution problem (*context vector*).

The *context vector* is obtained by concatenating a number of components equal to the number of available sub-optimizers. Each component is the best individual provided by the corresponding sub-optimizer. In order to evaluate the fitness of a single individual i in a sub-optimizer j, this individual is inserted in its corresponding position in the *context vector* while filling all the other positions using the best individuals of their respective sub-optimizers. In the basic CC model, each

sub-optimizer will be run for a single iteration. A complete cycle in which all sub-optimizers are invoked once is referred to as an *evolutionary cycle*. The basic CC framework is shown in Algorithm 2.

Algorithm 2 The CC algorithm

Require: $Max\_Function\_Evaluations$
1: Initialize the population
2: $FEVs \leftarrow$ Evaluate the population
3: $Context\_vector = best\_individual$
4: **while** $FEVs \leq Max\_Function\_Evaluations$ **do**
5: **for** each group j **do**
6: Run sub_optimizer$_j$
7: Update $context\_vector$
8: **end for**
9: **end while**
10: **return** $Context\_vector$

In this work, we implement the CC framework using BSO as the underlying sub-optimizer comparing its performance to other BSO variants. We implement and test the basic model in which each sub-component consists only of a single problem variable. In this case, if we have D problem variables, we then have D different sub-optimizers. If we refer to an idea i in a sub-optimizer j as $idea^i_j$, we evaluate this idea using the *context vector* = $(best_1, best_2, ..., best_{j-1}, idea^i_j, best_{j+1}, ..., best_D)$ where $best_k$ is the best idea for sub-optimizer k.

V. Experimental Procedure

A. Parameter Settings

The CCBSO proposed in this work is compared against BSODE [16], RGBSO [13] and IRGBSO [18] using a set of 20 classical benchmark functions shown in Table I.

For all algorithms, the populations size $n = 25$, the number of clusters $m = 5$, $p_{one-cluster} = 0.8$, $p_{one-center} = 0.4$, and $p_{two-centers} = 0.5$. For IRGBSO, $Threshold = 10$, $\alpha = 0.05 \times (UB - LB)$, and F is set to 0.5. Note that LB and UB are the lower and upper bounds of the search space.

Experiments are carried for dimensions $D = 10$, $D = 30$, and $D = 50$ with all algorithms performing a maximum number of function evaluations $10000 \times D$. Results provided are the averages taken over 25 different runs.

The codes for RGBSO and BSODE are obtained via private communication with Z. Cao (the author for [13], [16]). The code for RGBSO is then modified as proposed in [18] to implement IRGBSO and also to implement the CCBSO proposed in this work.

B. Results and Discussion

Results for all algorithms are provided in Tables II, III and IV. Best results are highlighted in bold based on the non-parametric Wilcoxon-ranksum test with a 5% confidence interval.

The results provided clearly illustrate the superior performance of CCBSO across the different problem sizes.

Convergence plots provided in Figures 1 and 2 for different functions and two different problem sizes show a faster speed of convergence for CCBSO in comparison to the other BSO variants.

Following the recommendations given in [20], the performance of CCBSO is assessed further using the Friedman test and the post hoc procedures (Holm and Finner). This is to calculate the ranking of the different algorithms as well as their adjusted p-values. We use the software available at [21] to perform these tests. The results are provided in Table V. Average rankings and p-values are reported for the Friedman test, and the best (lowest) rank is highlighted in bold. For the Holm and Finner procedures, adjusted p-values are reported with significant differences at a 0.05 level of significance. The results confirm the superior performance of the proposed CCBSO algorithm when compared to BSODE, RGBSO and IRGBSO. Note that a similar ranking was obtained in [18] when it was found that IRGBSO is followed by BSODE then RGBSO on a different benchmark library [22].

TABLE V
Results of Friedman, Holm, and Finner Tests

Algorithm	Friedman Ranking	Holm p-value	Finner p-value
RGBSO	3.06	0	0
BSODE	3.03	0	0
IRGBSO	2.25	0.0133	0.0133
CCBSO	**1.67**	-	-

VI. Conclusion

This paper proposed the implementation of a cooperative co-evolutionary algorithm using BSO as the sub-optimizer.

Experiments run on a library of 20 classical benchmark functions confirmed the superior performance of CCBSO when compared against three BSO variants across different problem sizes.

Future research directions involve running more experiments using different benchmark libraries, testing the framework using variable problem decomposition techniques and using different BSO variants as the underlying sub-optimizers.

References

[1] M. El-Abd and M. S. Kamel, "A taxonomy of cooperative search algorithms," in *Proc. 2nd International Workshop on Hybrid Metaheuristics, LNCS*, vol. 3636, 2005, pp. 32–41.

[2] ——, "A taxonomy for cooperative particle swarm optimizers," *International Journal of Computational Intelligence Research*, vol. 4, no. 2, pp. 137–144, 2008.

[3] M. A. Potter and K. A. de Jong, "A cooperative coevolutionary approach to function optimization," in *Proc. 3rd Parallel problem Solving from Nature*, 1994, pp. 249–257.

[4] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.

[5] M. El-Abd, "A cooperative approach to the artificial bee colony algorithm," in *Proc. of IEEE Congress on Evolutionary Computation*, 2010, pp. 1–5.

[6] H. Chen, Y. Zhu, K. Hu, X. He, and B. Niu, "Cooperative approaches to bacterial foraging optimization," in *Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence, LNCS*, Z. W. Geem, Ed. Springer-Verlag:Berlin Heidelberg, 2008, vol. 5227, pp. 541–548.

[7] H. Chen, Y. Zhu, and K. Hu, "Cooperative bacterial foraging optimization," *Discrete Dynamics in Nature and Society*, vol. 2009, pp. 1–17, 2009.

TABLE I
CLASSICAL BENCHMARK FUNCTIONS

Function	Definition	Range
Sphere	$f_1(x) = \sum_{i=1}^{D} x_i^2$	[-100,100]
Griewank	$f_2(x) = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\frac{x_i}{\sqrt{i}} + 1$	[-600, 600]
Ackley	$f_3(x) = 20 + e - 20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) + \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos 2\pi x_i\right)$	[-32.768, 32.768]
Rastrigin	$f_4(x) = \sum_{i=1}^{D}\left[x_i - 10\cos(2\pi x_i) + 10\right]$	[-5.12,5.12]
Rosenbrock	$f_5(x) = \sum_{i=1}^{\frac{D}{2}}(100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2)$	[-2.048,2.048]
Schwefel 2.20	$f_6(x) = -\sum_{i=1}^{D} \mid x_i \mid$	[-100, 100]
Schwefel 2.21	$f_7(x) = \max_{1<i<D} \mid x_i \mid$	[-100, 100]
Schwefel 2.22	$f_8(x) = \sum_{i=1}^{D} \mid x_i \mid + \prod_{i=1}^{D} \mid x_i \mid$	[-100, 100]
Schwefel 2.23	$f_9(x) = \sum_{i=1}^{D} x_i^{10}$	[-10, 10]
Schwefel 2.25	$f_{10}(x) = \sum_{i=2}^{D}((x_i - 1)^2 + (x_1 - x_i^2)^2)$	[0, 10]
Schwefel 2.26	$f_{11}(x) = 418.893 \times D - \frac{1}{D}\sum_{i=1}^{D} x_i \sin(\sqrt{\mid x_i \mid})$	[-500, 500]
Powel-Sum	$f_{12}(x) = \sum_{i=1}^{D} \mid x_i \mid^{i+1}$	[-1, 1]
Levy	$f_{13}(x) = \sin^2(\pi w_1) + \sum_{i=1}^{D-1}(w_i - 1)^2[1 + 10\sin^2(\pi w_i + 1)] + (w_D - 1)^2[1 + \sin^2(2\pi w_D)]$	[-10, 10]
Alpine 1	$f_{14}(x) = \sum_{i=1}^{D} \mid x_i \sin(x_i) + 0.1x_i \mid$	[-10, 10]
Alpine 2	$f_{15}(x) = \prod_{i=1}^{D} \sqrt{x_i}\sin(x_i)$	[0, 10]
Pathological	$f_{16}(x) = \sum_{i=1}^{D-1}(0.5 + \frac{\sin^2(\sqrt{100x_i^2 + x_{i+1}^2}) - 0.5}{1 + 0.001(x_i^2 - 2x_i x_{i+1} + x_{i+1}^2)^2})$	[-100, 100]
Schaffer	$f_{17}(x) = \sum_{i=1}^{D} 0.5 + \frac{\sin^2(\sqrt{x_i^2 + x_{i+1}^2}) - 0.5}{[1 + 0.001(x_i^2 + x_{i+1}^2)]^2}$	[-100, 100]
Step 2	$f_{18}(x) = \sum_{i=1}^{D}(\lfloor x_i + 0.5 \rfloor)^2$	[-100, 100]
Sum Squares	$f_{19}(x) = \sum_{i=1}^{D} i x_i^2$	[-10, 10]
Stretched V-Sine	$f_{20}(x) = \sum_{i=1}^{D-1}(x_i^2 + x_{i+1}^2)^{0.25}[\sin^2(50(x_i^2 + x_{i+1}^2)^{0.1}) + 0.1]$	[-10, 10]

[8] Y. Shi, "Brain storm optimization algorithm," in *Proc. International Conference on Swarm Intelligence*, 2011, pp. 303–309.

[9] ——, "An optimization algorithm based on brainstorming process," *International Journal of Swarm Intelligence Research*, vol. 2, no. 4,

TABLE II
RESULTS OF 10D

Bench. Func.	BSODE		RGBSO		IRGBSO		CCBSO	
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
1	5.49e-18	1.40e-17	2.06e-18	5.22e-18	7.20e-18	1.01e-17	**5.49e-19**	**1.77e-18**
2	1.05e-01	4.42e-02	2.09e+01	7.17e+00	**4.17e-02**	**2.98e-02**	**4.36e-02**	**2.53e-02**
3	2.64e-09	2.53e-09	8.25e-10	8.58e-10	1.13e-09	1.46e-09	**6.87e-11**	**6.44e-11**
4	6.81e+00	3.09e+00	4.54e+00	1.91e+00	5.33e+00	2.93e+00	**1.59e-14**	**1.55e-14**
5	3.50e+00	8.88e-01	3.29e+00	7.96e-01	**1.48e-01**	**5.19e-02**	5.90e-01	1.33e+00
6	8.86e-08	1.99e-07	4.74e-09	5.42e-09	1.13e-08	1.33e-08	**5.43e-10**	**5.63e-10**
7	1.61e-07	3.23e-07	**6.35e-09**	**6.17e-09**	1.76e-08	2.08e-08	2.44e-07	1.95e-07
8	1.10e+01	2.55e+01	2.28e+02	1.41e+02	1.10e-08	1.31e-08	**6.92e-10**	**7.67e-10**
9	1.04e-72	5.20e-72	1.44e-82	4.74e-82	1.28e-87	5.98e-87	**1.84e-97**	**9.20e-97**
10	9.89e-18	1.91e-17	1.82e-18	3.35e-18	1.09e-16	1.31e-16	**2.69e-21**	**6.60e-21**
11	1.54e+03	3.74e+02	1.86e+03	3.48e+02	1.05e+03	2.53e+02	**1.11e+02**	**1.09e+02**
12	1.83e-09	2.67e-09	9.19e-10	1.20e-09	4.72e-10	4.20e-10	**7.66e-25**	**3.81e-24**
13	1.26e-12	4.20e-12	3.16e+00	2.87e+00	1.45e+00	1.92e+00	**4.48e-22**	**1.07e-21**
14	6.78e-05	2.49e-04	4.88e-03	4.43e-03	6.57e-04	1.50e-03	**1.40e-11**	**4.16e-11**
15	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
16	1.21e+00	3.34e-01	1.81e+00	4.65e-01	9.52e-01	5.17e-01	**5.95e-01**	**3.56e-01**
17	1.94e+00	6.50e-01	3.51e+00	3.39e-01	1.41e+00	5.50e-01	**1.76e-01**	**2.01e-01**
18	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
19	1.13e-13	2.91e-13	2.95e-15	6.46e-15	3.52e-16	5.33e-16	**1.48e-20**	**3.30e-20**
20	1.03e+00	2.00e-01	1.28e+00	4.50e-01	**3.68e-01**	**3.33e-01**	7.26e-01	6.68e-01

TABLE III
RESULTS OF 30D

Bench. Func.	BSODE		RGBSO		IRGBSO		CCBSO	
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
1	3.05e-17	6.57e-17	**2.23e-18**	**2.90e-18**	2.32e-17	2.59e-17	**2.47e-18**	**5.35e-18**
2	9.16e-03	8.87e-03	1.24e-02	1.30e-02	**5.02e-03**	**8.07e-03**	3.15e-02	3.22e-02
3	2.43e-09	1.74e-09	9.91e-10	5.39e-10	8.55e-10	3.58e-10	**2.43e-10**	**1.76e-10**
4	3.69e+01	7.89e+00	3.70e+01	8.86e+00	3.96e+01	1.12e+01	**4.41e-02**	**2.20e-01**
5	2.52e+01	1.03e+00	2.54e+01	5.90e-01	2.00e+01	9.08e-01	**1.67e+00**	**4.52e+00**
6	1.72e-05	2.48e-05	2.34e-07	2.95e-07	5.72e-07	8.92e-07	**4.40e-09**	**4.42e-09**
7	9.74e-05	1.14e-04	**7.00e-07**	**5.95e-07**	1.47e-06	9.69e-07	5.22e-05	2.16e-05
8	2.38e+02	1.81e+02	7.62e+02	1.64e+02	3.62e-07	4.51e-07	**5.23e-09**	**3.28e-09**
9	2.04e-61	4.16e-61	8.72e-70	4.31e-69	2.24e-73	1.12e-72	**4.70e-98**	**1.71e-97**
10	3.25e-17	4.69e-17	8.29e-18	1.12e-17	1.28e-15	1.22e-15	**3.19e-20**	**7.64e-20**
11	5.39e+03	5.86e+02	5.35e+03	7.24e+02	3.12e+03	3.38e+02	**3.59e+02**	**1.83e+02**
12	3.81e-09	3.45e-09	8.54e-10	6.66e-10	9.12e-10	4.09e-10	**1.02e-25**	**4.16e-25**
13	1.82e-02	9.09e-02	1.91e+01	7.05e+00	1.00e+01	5.84e+00	**8.51e-21**	**1.61e-20**
14	1.71e-02	1.29e-02	6.20e-02	3.38e-02	5.79e-03	7.40e-03	**6.15e-11**	**5.25e-11**
15	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
16	6.47e+00	1.23e+00	6.99e+00	1.04e+00	6.05e+00	1.94e+00	**1.98e+00**	**6.14e-01**
17	8.78e+00	8.53e-01	1.20e+01	1.07e+00	8.49e+00	8.09e-01	**7.60e-01**	**4.41e-01**
18	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
19	6.89e-07	1.05e-06	2.90e-08	5.39e-08	9.21e-10	1.69e-09	**1.48e-19**	**2.44e-19**
20	4.43e+00	8.64e-01	4.70e+00	9.92e-01	**1.53e+00**	**4.33e-01**	3.97e+00	1.30e+00

pp. 35–62, 2011.

[10] S. Cheng, Q. Qin, J. Chen, and Y. Shi, "Brain storm optimization algorithm: A review," *Artificial Intelligence Review*, pp. 1–14, 2016.

[11] Z. Zhan, J. Zhang, Y. Shi, and H. Liu, "A modified brain storm optimization algorithm," in *Proc. of IEEE Congress on Evolutionary Computation*, 2012, pp. 1969–1976.

[12] D. Zhou, Y. Shi, and S. Cheng, "Brain storm optimization algorithm with modified step-size and individual generation," in *Proc. International Conference on Swarm Intelligence*, 2012, pp. 243–252.

[13] Z. Cao, Y. Shi, X. Rong, B. Liu, Z. Du, and B. Yang, "Random grouping brain storm optimization algorithm with a new dynamically changing step size," in *Proc. International Conference on Swarm Intelligence*, 2015, pp. 387–364.

[14] S. Cheng, Y. Shi, Q. Qin, Q. Zhang, and R. Bai, "Population diversity maintenance in brain storm optimizatio algorithm," *Journal of Artifical Intellgence and Soft Computing Research*, vol. 4, no. 2, pp. 83–97, 2014.

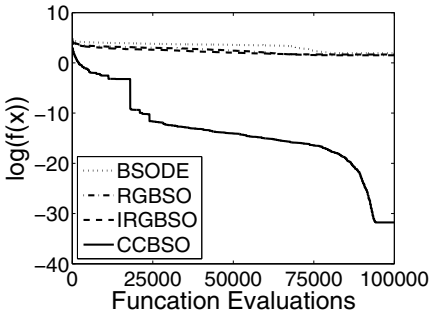
[15] S. Cheng, Y. Shi, Q. Qin, T. O. Ting, and R. Bai, "Maintaining population diversity in brain storm optimization algorithm," in *Proc. of IEEE Congress on Evolutionary Computation*, 2014, pp. 3230–3237.

[16] Z. Cao, X. Hei, L. Wang, Y. Shi, and X. Rong, "An improved brain storm optimization with differential evolution strategy for applications of ANNs," *Mathematical Problems in Engineering*, vol. 2015, pp. 1–18, 2015.

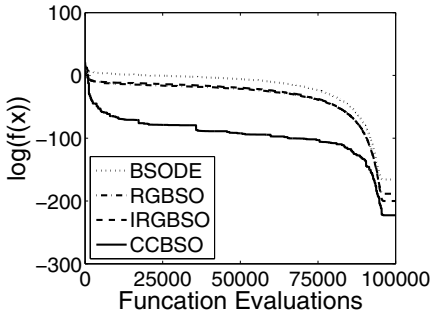
[17] J. Chen, S. Cheng, Y. Chen, Y. Xie, and Y. Shi, "Enhanced brain storm optimization algorithm for wireless sensor networks deployment," in *Proc. International Conference on Swarm Intelligence*, 2015, pp. 373–

TABLE IV
RESULTS OF 50D

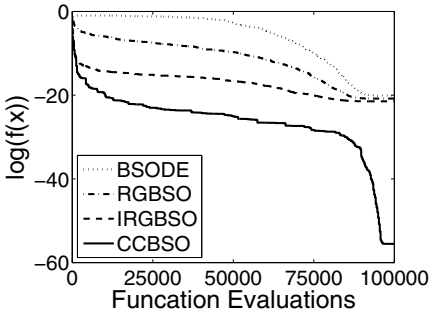
Bench. Func.	BSODE		RGBSO		IRGBSO		CCBSO	
	Mean	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**	**Mean**	**Std.**
1	3.18e-17	4.02e-17	4.92e-17	5.97e-17	2.94e-17	2.52e-17	**6.83e-18**	**9.42e-18**
2	1.97e-03	3.63e-03	1.18e-03	3.27e-03	**5.92e-04**	**2.05e-03**	3.79e-02	4.12e-02
3	3.68e-09	2.79e-09	3.20e-09	1.79e-09	1.17e-09	5.17e-10	**5.14e-10**	**4.61e-10**
4	6.64e+01	1.12e+01	6.34e+01	1.62e+01	8.36e+01	1.76e+01	**5.56e-02**	**2.11e-01**
5	4.56e+01	1.30e+00	4.62e+01	1.36e+00	4.07e+01	9.59e-01	**1.50e+01**	**2.94e+01**
6	1.39e-03	3.89e-03	4.53e-03	1.13e-02	7.26e-06	1.05e-05	**1.45e-08**	**1.02e-08**
7	2.17e-03	1.25e-03	2.81e-03	1.89e-03	1.63e-05	1.08e-05	**2.92e-04**	**8.46e-05**
8	5.92e+02	2.88e+02	6.41e+02	2.27e+02	9.98e-06	1.39e-05	**1.60e-08**	**8.64e-09**
9	1.06e-54	3.36e-54	2.78e-54	1.18e-53	3.56e-71	1.69e-70	**4.46e-95**	**2.20e-94**
10	9.29e-17	9.02e-17	9.00e-17	8.98e-17	3.15e-15	2.62e-15	**5.92e-20**	**1.61e-19**
11	9.45e+03	1.07e+03	9.43e+03	9.10e+02	4.76e+03	5.72e+02	**5.43e+02**	**2.40e+02**
12	2.85e-09	2.23e-09	2.30e-09	1.79e-09	6.84e-10	2.86e-10	**2.38e-26**	**9.99e-26**
13	7.63e-02	1.69e-01	7.24e-02	1.59e-01	1.29e+01	6.67e+00	**6.89e-20**	**2.91e-19**
14	8.92e-02	6.53e-02	9.55e-02	6.73e-02	1.20e-02	1.47e-02	**2.00e-10**	**1.51e-10**
15	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
16	1.30e+01	1.84e+00	1.36e+01	1.44e+00	1.15e+01	2.92e+00	**3.59e+00**	**8.16e-01**
17	1.56e+01	1.36e+00	1.58e+01	1.48e+00	1.65e+01	1.13e+00	**1.21e+00**	**6.43e-01**
18	8.00e-02	2.77e-01	2.40e-01	4.36e-01	**0.00e+00**	**0.00e+00**	**0.00e+00**	**0.00e+00**
19	3.27e-04	5.26e-04	2.97e-04	4.20e-04	1.36e-06	1.71e-06	**1.37e-18**	**1.54e-18**
20	7.85e+00	1.17e+00	8.56e+00	1.16e+00	**4.84e+00**	**1.41e+00**	6.36e+00	1.56e+00



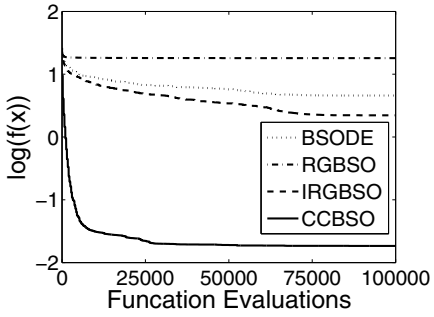
(a) f4



(b) f9



(c) f12



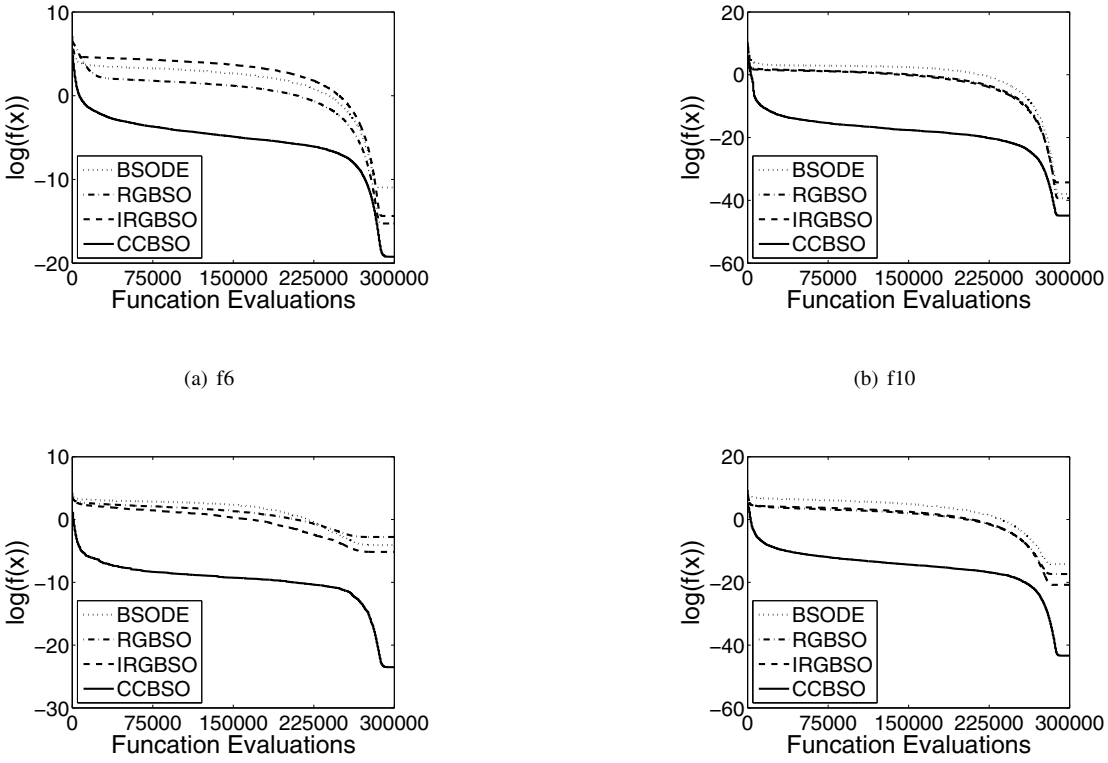
(d) f17

Fig. 1. Convergence behavior of all the algorithms for a sample of the classical benchmark functions; 10 dimensions.

381.

[18] M. El-Abd, "Brain storm optimization algorithm with re-initialized ideas and modified step size," in *Accepted in IEEE Congress on Evolutionary Computation*.

[19] B. Basturk and D. Karaboga, "An artificial bee colony (ABC) algorithm for numeric function optimization," in *Proceedings of the IEEE Swarm*



(a) f6

(b) f10

(c) f14

(d) f19

Fig. 2. Convergence behavior of all the algorithms for a sample of the classical benchmark functions; 30 dimensions.

Intelligence Symposium, 2006.

[20] J. Derrac, S. Garcia, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[21] "Keel software," http://sci2s.ugr.es/keel/download.php, 2012.

[22] B. Y. Qu, J. J. Liang, Z. Y. Wang, Q. Chen, and P. N. Suganthan, "Novel benchmark functions for continuous multimodal optimization with comparative results," *Swarm and Evolutionary Computation*, vol. 26, pp. 23–34, 2016.