International Journal of Advanced Science and Technology
Vol.121 (2018), pp.1-20
http://dx.doi.org/10.14257/ijast.2018.121.01

Lookback-Guess-Next Optimizer: Feedback-Guided Random Search Technique with Biased Mapping for Solving Unconstrained Optimization Problems

Muhammed Jassem Al-Muhammed

Faculty of Information Technology, American University of Madaba, Madaba, Jordan
m.almuhammed@aum.edu.jo

Abstract

Finding global optima for functions is a very important problem. Although a large number of methods have been proposed for solving this problem, more effective and efficient methods are greatly required. This paper proposes an innovative method that combines different effective techniques for speeding up the convergence to the solution and greatly improving its precision. In particular, the method uses feedback-guided random search technique to identify the promising regions of the domains and uses the biased mapping technique to focus the search on these promising regions, without ignoring the other regions of the domains. Therefore, at any point of time, the domain of each variable is entirely covered with much more emphasis on the promising regions. Experiments with our prototype implementation showed that our method is efficient, effective, and outperformed the state-of-art techniques.

Keywords*: Optimization problem; biased mapping; global optima; effective search regions; local optima*

1. Introduction

Most of the real world problems lend themselves to functions optimization. Unfortunately, most of these functions are difficult to optimize using direct mathematical means because they are non-differentiable, have no derivatives, not continuous, and have multiple local optima. Researchers have proposed different methods to tackle these problems. These methods adopt different techniques to speed up the search and improve the precision of the solution (minimum or maximum). The major techniques include evolutionary algorithms (such as [1–4, 15, 16, 20, 21, 25, 27, 28]) and swarm intelligence algorithms (such as [8–14, 17–20, 30]). Other methods use direct mathematical methods (such as [22]).

In order for the solution method to be effective, it must possess several properties. First, it must be able to find the exact global optimum or at least a so-close approximation to it. That is, it must avoid being caught in local optima. Second, in order to be practical, it must require a minimal amount of resources (especially CPU time). Third, it must be general and not tailored to any specific set of problems. Forth, it must be easy to implement.

This paper proposes a feedback-guided random search method augmented with biased mapping technique. We call our method **ANT-BM**.[2] This method is based on three effective techniques: lookback capability technique, biased mapping technique, and auto-adjusting technique. The lookback technique enables the method to use the intermediate search results from previous search round as a feedback for identifying the promising

Received (July 25, 2018), Review Result (September 6, 2018), Accepted (September 22, 2018)
[2]**ANT-BM**: lookb**A**ck-guess-**N**ext op**T**imizer: … with **B**iased **M**apping…

ISSN: 2005-4238 IJAST

regions for the next search round. (The promising regions are those in which the solution is highly expected.) The biased mapping technique enables the method to dynamically condense the search in the promising regions of the domains. In other words, it redirects more randomly generated candidate solutions to these promising regions. The auto-adjusting technique uses the feedback to automatically adjust the method's search parameters. In particular, the auto-adjusting technique utilizes the feedback to dynamically move the focus of the search to the promising regions. These techniques are augmented with the ability to dynamically reduce the size of the promising regions. The latter property (reduction of the promising regions) causes the search to concentrate on continuously narrowing promising regions, which results in thoroughly searching these regions and speeding up the convergence to the solution.

The paper makes the following contributions. First, it proposes a highly effective and efficient (time/space wise) method for finding the optima for *n*-dimensional functions. Second, it defines biasing coefficients and effective mapping techniques that enable focusing the search on the parts of a domain where the solution is likely to reside without ignoring the other parts of the domain. Third, the method provides techniques for dynamically turning the focus of the search to parts of the domains using feedbacks collected during the search.

We present our contributions as follows. Section 2 introduces some basic terminologies. Section 3 discusses the biased mapping technique. Section 4 presents the stopping conditions and Section 5 presents the algorithmic details of the proposed method. We present our comprehensive analysis for the performance of our method in Section 6. We conclude and give directions for future work in Section 7.

2. Terminologies

Let $f(x_1, x_2, \ldots, x_n)$ be an *n*-dimensional function and $[A_i, B_i]$ be bounded domains of the variables x_i (i=1, 2, …, n). Optimizing a function f means finding an *n*-dimensional point $X^*(x_1^*, x_2^*, \ldots, x_n^*)$ from the domains of the variables such that the function f is in its optimal value (minimum or maximum). We call the point $X^*(x_1^*, x_2^*, \ldots, x_n^*)$ the *best global point* for the function f. We also call each value x_i^* the *best substitution* for the variable x_i. We in addition call the part of the domain in which the best substitution is highly expected the *effective search region* or *promising region*.

3. The Biased Mapping Technique

We introduce in this section the biased mapping technique. We first define the biasing coefficients in subsection 3.1. We then use these biasing coefficients to define our biased mapping in subsection 3.2.

3.1. The Biasing Coefficients

Let $[A_i, B_i]$ be a bounded domain of a variable x_i and Ω_i be the center of this domain. Our proposed algorithm splits the domain of each variable x_i into an effective search region and two marginal search regions. The three regions entirely cover the domain of each variable. Figure 1 shows a bonded domain $[A_i, B_i]$ and the three regions. The effective search regions are designated as effective crawlers. The marginal search regions are designated as left region and right region.[3] The effective crawler, which is centered at some point $C_i \in [A_i, B_i]$ and whose radius is r_i, covers the region of the domain in which the best substitution x_i^* is highly expected. (That is why we call this region effective search region.) The effective crawler provides the method with an effective mechanism to

[3] We designate the effective search region as effective crawler because this region crawls over the interval during the search for the best point for a function *f*.

focus most of the search on the promising regions. The left and right regions cover the other parts of the domain in which the best substitution may exist (but not very expected). These two marginal regions serve as a guard to avoid missing the best substitution if this substitution is not really within the region covered by the effective crawler.

Figure 1 also shows a random number generator that produces uniformly distributed random numbers within the range ($B_i - A_i$). This generator provides the technique with random numbers that serve as raw substitutions for f. (As we discuss next, raw substitutions need further processing before they can be used in the function.)

Since the best point is highly expected in the effective regions, we must ensure a comprehensive search for these regions without of course ignoring the marginal regions. We must therefore redirect (or map) most of the random numbers to the effective regions. This idea is illustrated in Figure 1, where a large number of the random numbers generated in the range "B_i–A_i" (thick line) is redirected to the region of the effective crawler using the biased mapping technique, which will be discussed in great details in the next subsection.

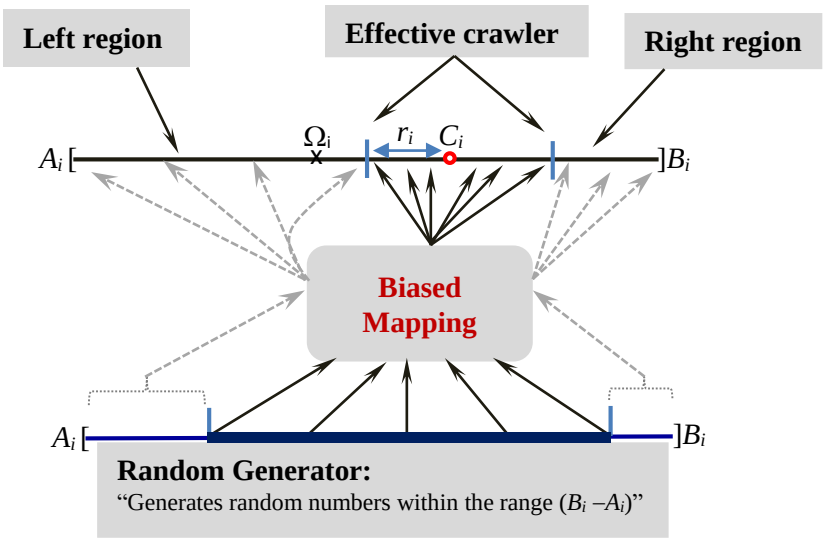

Figure 1. The Effective Crawler and the Right/Left Marginal Regions

To focus the search on the effective search regions, we define the following two values, F_i^{bias} and G_i^{bias}, which we call the *biasing coefficients*.

$$F_i^{bias} = \frac{\alpha + | \beta - r_i |}{2r_i}$$
$$G_i^{bias} = \frac{\beta - | \beta - r_i |}{D_i - 2r_i} \qquad \ldots\ldots\ldots \quad (1)$$

Where $\alpha \geq \beta$, $\alpha + \beta = D_i$, $D_i = B_i - A_i$, and r_i is the radius of the effective search region. The values α and β are the initial weigh that is given to respectively the effective and marginal search regions.

Based on the biasing coefficients, we define the values E_i, L_i, R_i, which we call the amounts of the bias.

$$E_i = 2r_i F_i^{bias}$$
$$L_i = (C_i - r_i - A_i) G_i^{bias} \qquad \ldots\ldots\ldots \quad (2)$$
$$R_i = (B_i - C_i - r_i) G_i^{bias}$$

E_i is the amount of the bias to the effective crawler region. It determines the number of the random numbers, which are mapped to the effective search region. For instance, if the range of the random numbers is 10 and E_i = 6, then roughly 60% of the randomly generated numbers within this range will be mapped to the effective crawler and the other 40% will be mapped to the marginal regions. The values L_i and R_i determine the number of the random numbers that are mapped to the left and right regions respectively. The main goal of the L_i and R_i is to ensure that no portion of a variable's domain is left unsearched. According to the definition of F_i^{bias} and G_i^{bias}, at *least* a range of size α of the domain is redirected to the effective crawler and at *most* a range of size β of the domain is redirected to both L_i and R_i.[4]

Additionally, since both F_i^{bias} and G_i^{bias} depend on the radius of the effective crawler r_i, the amounts of bias E_i, R_i, and L_i vary as the radius r_i changes. That is, they dynamically increase or decrease during the search as r_i changes (decreases or increases).

The effective crawler can move to any point within a variable's domain by moving its center C_i. In addition, crawlers on different domains can be in different positions and have different radiuses. That is, the position and radius of each crawler has *no* relation to the position and radius of the crawlers on the other domains and only depend on the intermediate results of the search.

Before concluding this subsection, we point out two important properties of our technique. First, the ability of moving the effective crawler to different parts of a variable's domain allows our algorithm to dynamically shift the search focus to any part of this domain in which the solution is highly expected. Second, the ability of biasing more random points to a specific part of a variable's domain ensures a comprehensive search for this part without ignoring the other parts of the domain. Both properties ensure not only a full coverage and exhaustive search for the entire domain, but also escape being trapped in local optima because the search is not restricted to only specific regions of a domain (it actually always covers the entire domain although more focus is dedicated to the effective regions).

3.2. The Biased Mapping

The biased mapping plays a vital role in the proposed technique. It largely biases the search toward the effective regions. Its major role is to redirect (map) more randomly generated numbers (or raw substitutions) to a specific region of a variable's domain. In particular, we require our biased mapping to use the amounts of bias (defined in equation 2) as a criterion and redirect more random numbers to the effective search region than to the other two regions. (Recall that the effective search region is more promising.)

Let Ω_i be the center of the bounded domain $[A_i, B_i]$ of the variable x_i and C_i be the center of the region covered by the effective crawler for i =1, 2, …, n. Figure 3 shows our proposed biased mapping.

Generally speaking, the biased mapping redirects a raw random number x_i^{rnd} to only one of the three regions based on the amounts of bias E_i, L_i, and R_i. Therefore, in lines 2 through 6 (similarly lines 8 through 12), we compare the raw random number x_i^{rnd} to the amount of bias and redirect it (x_i^{rnd}) to any of their respective regions only if x_i^{rnd} is less than the corresponding amount of bias. If x_i^{rnd} is less than R_i (line 2), line 3 redirects x_i^{rnd} to the right region (since R_i is the amount of bias to the right region). If, however, x_i^{rnd} is less than L_i+ R_i (line 4), line 5 redirects x_i^{rnd} to the left region. If none of the conditions is true, line 6 redirects x_i^{rnd} to effective search region.

It is clear that if the random generator is fair (uniformly covers the domain $[A_i, B_i]$), more random numbers will be redirected to the effective search region. That is because the amount of bias to the effective search region E_i is always greater than the total amount of bias to both the left and right regions L_i+R_i. Additionally, although less focus on the

[4] We can show with a simple math that $E_i + L_i + R_i = D_i$.

left and right regions, they are nevertheless still covered by random numbers since their respective amounts of bias L_i and R_i are never zero.

The comparison between C_i and Ω_i in line 1 (and implicitly in line 7) is to correctly map between the right and left regions. For instance, if C_i is greater than Ω_i, the right region is smaller than the left region. Therefore, we first try to map the raw random number to right region (the smaller) before considering mapping it to the left region (the lager).

1.	**IF** $C_i > \Omega_i$ **THEN**
2.	**IF** $x_i^{rnd} < R_i$ **THEN**
3.	$x_i^{new} = \frac{x_i^{rnd}}{G_i^{bias}} + C_i + r_i$
4.	**ELSE IF** $x_i^{rnd} < R_i + L_i$ **THEN**
5.	$x_i^{new} = \frac{x_i^{rnd} - R_i}{G_i^{bias}} + A_i$
6	**ELSE** /* $(L_i+R_i) < x_i^{rnd} \le E_i$ */ $x_i^{new} = \frac{x_i^{rnd} - R_i - L_i}{F_i^{bias}} - C_i - r_i$
7	**ELSE**
8	**IF** $x_i^{rnd} < L_i$ **THEN**
9	$x_i^{new} = \frac{x_i^{rnd}}{G_i^{bias}} + C_i + r_i$
10	**ELSE IF** $x_i^{rnd} < R_i + L_i$ **THEN**
11	$x_i^{new} = \frac{x_i^{rnd} - L_i}{G_i^{bias}} + A_i$
12	**ELSE** $x_i^{new} = \frac{x_i^{rnd} - R_i - L_i}{F_i^{bias}} - C_i - r_i$

Figure 3. The Biased Mapping

4. Stopping Conditions

Let $X^{(1)}$ $(x_1^{(1)}, x_2^{(1)}, \ldots, x_n^{(1)})$ and $X^{(2)}$ $(x_1^{(2)}, x_2^{(2)}, \ldots, x_n^{(2)})$ be two points that produced better values $F^{(1)}$ and $F^{(2)}$ for the function f in two consecutive search rounds. We define our stopping conditions as follows.

$$|F^{(1)} - F^{(2)}| < \varepsilon^{(1)}$$
$$|X^{(1)} - X^{(2)}| < \varepsilon^{(2)}$$

Where,

$$|X^{(1)} - X^{(2)}| = \begin{cases} |x_1^{(1)} - x_1^{(2)}| \\ |x_2^{(1)} - x_2^{(2)}| \\ \ldots \\ |x_n^{(1)} - x_n^{(2)}| \end{cases} \quad \ldots\ldots \quad (3)$$

Both $\varepsilon^{(1)}$ and $\varepsilon^{(2)}$ are sufficiently small real numbers (e.g. 1E−16). These conditions mean that if the value of the function and the corresponding points that produced this value do not change for two consecutive search rounds, the search has reached an

equilibrium point. No further enhancement to the value of the function will be achieved. The search must therefore stop because the best value for the function *f* has been reached.

5. The ANT-BM Algorithm

The proposed algorithm iteratively searches for the best point $X^*(x_1^*, x_2^*,\ldots, x_n^*)$ that puts the function *f* in its global optimal value (minimum). It does the search by carrying out many rounds until the stopping conditions (3) hold. Figure 5 shows the algorithmic steps of the proposed algorithm. The algorithm has two parts: initialization process (lines 1–2) and search process (lines 3–17).

In the initialization process, the algorithm sets the centers of the effective crawlers C_i to the centers of the domains Ω_i. In addition, it is fair in the initialization process to consider the entire domain of each variable as a promising region. Thus, the algorithm sets the radius of each crawler r_i to $\Omega_i - \varepsilon$, where ε is a sufficiently small real number (e.g. 1E–30). In this case, each effective crawler covers almost the entire corresponding domain. Finally, the algorithm sets both α and β to half of the domain's length. As the search progresses, however, these initial settings will likely change due to the feedback acquired during the seek for the best point for the function *f*.

```
1.  FOR i=1 to n DO Ci← Ωi , ri← Ωi−ε ENDFOR
2.  fmin ← ∞ /**large value*/
    α=β=(Bi–Ai)/2 /**set alpha and beta to half of the domain size*/
3.  WHILE Stopping conditions do not hold DO
4.      WHILE (ri > δ for all i) DO
5.          compute Ei, Li, Ri /** as specified in formulas (1) and (2)*/
6           FOR i=1 to m DO
7.              FOR j=1 to n DO
8.                  xj^rnd ← RND * (Bj − Aj) /**RND=Random[0,1]*/
                    xj^new ← Biased Mapping(xj^rnd)
                ENDFOR
9.              fv ← f(X^new)
10.             IF fv < fmin THEN
11.                 fmin ← fv
                    X* ← X^new
                ENDIF
            ENDFOR
12.         FOR i = 1 to n DO
                ri ← ri/d
            ENDFOR
13.     ENDWHILE
14.     FOR i = 1 to n DO
15.     Ci ← xi*
16.     Adjust-Radius(Ci)
        ENDFOR
17. ENDWHILE
```

Figure 5. The Algorithmic Steps of ANT-BM Algorithm

The search process carries out several rounds each of which involves iteratively executing the lines 3–17 until the stopping conditions (3) hold. Each round involves iterative search (lines 4–13) and an adjustment for the search's parameters (lines 14–16). The iterative search executes the lines 4–13 until the radiuses of all the effect search regions (effective crawlers) become less than a sufficiently small real number called threshold δ (e.g. δ=1E–300). In any iteration, the algorithm conducts *m* experiments each of which consists of generating *n* random numbers x_j^{rnd} within the range of the variables' domains. Each of the *n* random numbers x_j^{rnd} is then redirected to only one of the three regions using our biased mapping in Figure 3. The result of applying the biased mapping to the random numbers x_j^{rnd} is the substitution $X^{new}(x_1^{new}, x_2^{new}, \ldots, x_n^{new})$ for the function *f*. If the current substitution X^{new} produces a better value for the function *f*, the algorithm keeps both the substitution X^{new} and this better value (lines 10 and 11) in respectively the variables X^* and f_{min}. The value of *f* and the substitution X^{new} constitute fundamental information that will be used as a feedback for adjusting the search parameters for the next round.

After finishing the *m* experiments, the algorithm reduces the radiuses of the effective crawlers r_i by *d* (line 12) and re-calculates the amount of bias, where *d* can be any real number greater than one ("> 1"). The main objective of reducing the radiuses r_i's is to gradually increase the bias toward the effective crawlers, but without ignoring the other two regions. Therefore, the part of the domain covered by the effective crawler is thoroughly searched and the likelihood of finding the global minimum is greatly increased.

After each round, the algorithm makes use of the feedback from the just-finished round to adjust the search parameters for the next round. Adjusting the search parameters is done as follows. First, the algorithm moves the centers of the effective crawlers to the so-far best point X^* because this point has produced a better value of the function (line 15). The objective of changing the center of the effective crawler is to focus the search in the neighborhood of this promising point since it is highly likely that the global best point is located in its neighborhood. Second, it computes the corresponding radiuses r_i for each effective crawler (line 16). Computing each radius r_i is done using the procedure `Adjust-Radius`, which is defined in Figure 6. The logic of the computation is straightforward. If the new computed center of the effective crawler is greater than the center of the domain (Ω_i), the radius is the absolute value of upper limit of the domain (B_i) minus C_i. Otherwise, it is the absolute value of the difference between the new center of the effective crawler C_i and lower limit of the domain (A_i).

```
Adjust-Radius(Ci)
      IF Ci > Ωi THEN ri = |Bi − Ci|
      ELSE ri= |Ci − Ai|;
END Adjust-Radius
```

Figure 6. Computing the Radius r_i of the Effective Crawler

Once the search parameters are adjusted, the algorithm is ready for a new round. Launching a new round depends, however, on whether the stopping conditions hold or not. If these conditions do not hold, a new round is launched using the new centers and radiuses of the effective crawlers. If the stopping conditions actually hold, the search stops. That is because if no enhancement to the value of *f* is achieved for two consecutive stages, the search has reached an equilibrium point; any new round will not produce better results.

6. Experimental Results

We implemented our algorithm using Java programming language. The execution hardware is a laptop dual core processor (1.7GHz) with 2 GB memory. The operating system is windows 7 (32 bits).[5]

We evaluated our algorithm using a large number of hard benchmark functions obtained from [5][6][7][29]. Table 1 shows part of these benchmark functions (54 functions), their dimensions (*N*), their global minimum ($f(x^*)$), and domains of the variables. Figure 7 shows the graphs of a sample of these benchmark functions. In all our evaluations, we set the number of experiments *m* to "5", the radiuses' reduction factor *d* to "1.5", δ=1E−320, and $\varepsilon^{(1),(2)}$ =1E−16. In addition, to better estimate the CPU time, we executed every function 40 times and recorded the minimum, average, and maximum time over all of the 40 runs.[6]

Table 1. A Set of Function Benchmarks (54 different functions)

Benchmark Functions	Benchmark Functions
Alpine 1 (N) $f(x^*) = 0, -10 \le x_i \le 10$	**Booth (2)** $f(x^*) = 0, -10 \le x_i \le 10$
Ackley (N), $f(x^*) = 0, -32.768 \le x_i \le 32.768$	**Rastrigin (N)** $f(x^*) = 0, -5.12 \le x_i \le 5.12$
Sphere (N) $f(x^*) = 0, -100 \le x_i \le 100$	**Beale (2)** $f(x^*) = 0, -4.5 \le x_i \le 4.5$
Exponential(N) $f(x^*) = 1, -1 \le x_i \le 1$	**Bukin (2)** $f(x^*) = 0, -15 \le x_i \le 15$
EASOM (2) $f(x^*) = -1, -100 \le x_i \le 100$	**Cross-In-Tray(2)** $f(x^*) = -2.062, -10 \le x_i \le 10$
Drop-wave (2) $f(x^*) = -1, -5.12 \le x_i \le 5.12$	**Grienback (N)** $f(x^*)=0, -100 \le x_i \le 100$
Holder Table (2) $f(x^*) = -19.20850, -10 \le x_i \le 10$	**Levy (N)** $f(x^*) = 0, -10 \le x_i \le 10$
Schaffer (N), $f(x^*) = 0$ $-10 \le x_i \le 10$	**Schwefle (N)** $f(x^*) = 0, -500 \le x_i \le 500$
Shubert (5), $f(x^*) = -186.7309, -10 \le x_i \le 10$	**Perm (N)** $f(x^*)=0, -N \le x_i \le N$
Rotated Hyper-Ellipsoid (N), $f(x^*) = 0, -65.536 \le x_i \le 65.536$	**Sum of Different Powers (N)**, $f(x^*) = 0, -1 \le x_i \le 1$
Sum Squares (N), f(x*) = 0, $-10 \le x_i \le 10$	**Matyas (2) f(x*)=0, x*(0),** $-10 \le x_i \le 10$
Zakharov (N) f(x*)=0, $-5 \le x_i \le 10$	**Michalewicz (N)**, $f(x^*) = -1.8013, -4.687658, -9.66015$ for respectively N=2, 5, 10, $0 \le x_i \le \pi$
Eggholder (2) $f(x^*) = -959.6407, -512 \le x_i \le 512$	**Goldstein Price (2) f(x*) = 3,** $-2 \le x_i \le 2$
Damavandi (2) $f(x^*)=0$, $x^*(2)$ $0 \le xi \le 14$	**Xin-She Yang's Function No.02 (N)** $f(x^*)=0$, $x^*(0), -2\pi \le xi \le 2\pi$
Xin-She Yang No. 04 (N) $f(x^*) = -1$, $x^*(0), -10 \le xi \le 10$	**CrossLegTable(2)** $f(x^*)=0$, $x^*(0), -10 \le xi \le 10$
DropWave (N), $f(x^*) = -1, -5.12 \le xi \le 5.12$	**RANA Function (N)** $f(x^*) = -511.73, -512 \le xi \le 512$
Xin-She Yang No. 03 (N) $f(x^*)= -1, -20 \le xi \le 20$	**Styblinski-Tang(N)** $f(x^*)= -39.1659*N, -5 \le xi \le 5$
Test Tube Holder (2), f(x*) = –10.8723,	**Pen Holder(2)** $f(x^*)=0.96353483$

[5] The hardware used to test the performance of the other algorithms is more powerful than ours. They mostly used Intel Core i5-3210M processor runs at 2.5 GHz or better (see [16] for instance).

[6] In fact, we tried several values for *d* within the interval (1, 10]. We did a simulation for choosing *d*. The simulation showed that as *d* increases the CPU time **decreases** but the error starts also slightly **building up**. The value "1.5" for *d* is found to be a reasonable compromise between the CPU time the average error. We have *not* tested values for *d* greater than 10. We set *m* to 5 because we observed during simulation that increasing *m* does not improve the precision while increases CPU time.

$-10 \le xi \le 10$	$-11 \le xi \le 11$
Levy 13 (2) f(x*) = 0, $-10 \le xi \le 10$	**Himmelblau (2)** $f(x^*)=0$, $-6 \le xi \le 6$
Salamon (N) f(x*)=0, $-10 \le xi \le 10$	**Trid (N)** $f(x^*)= -N(N-1)(N+4)/6$ $-N^2 \le xi \le N^2$
Adjiman (2) f(x*)= –2.02181 $-1 \le x1 \le 2, -1 \le x2 \le 1$	**Brown(N) f(x*) = 0,** $-1 \le xi \le 4$
Chung Reynolds(N) f(x*)=0, $-100 \le xi \le 100$	**Dixon and Price(N) f(x*)=0 ,** $-10 \le xi \le 10$
Powell Sum(N) f(x*)=0, $-1 \le xi \le 1$	**Giunta (N) f(x*)=0.0644704,** $-1 \le xi \le 1$
Cigar (N) f(x*)=0, $-100 \le xi \le 100$	**Whitley (2) f(x*) =0**, $-10.24 \le xi \le 10.24$
Branin (2) f(x*)=0.397887 $-5 \le x1 \le 10, 0 \le x2 \le 15$	**Trefethen(2) f(x*)=–3.3068686474** $-10 \le xi \le 10$
Egg Crater (N) f(x*) = 0, $-5 \le xi \le 5$	**Crowned Cross (2) f(x*) = 0.0001,** $-10 \le xi \le 10$
Leon (N) f(x*)=0 $-1.2 \le xi \le 1.2$	**Sargan (N) f(x*)=0,** $-100 \le xi \le 100$

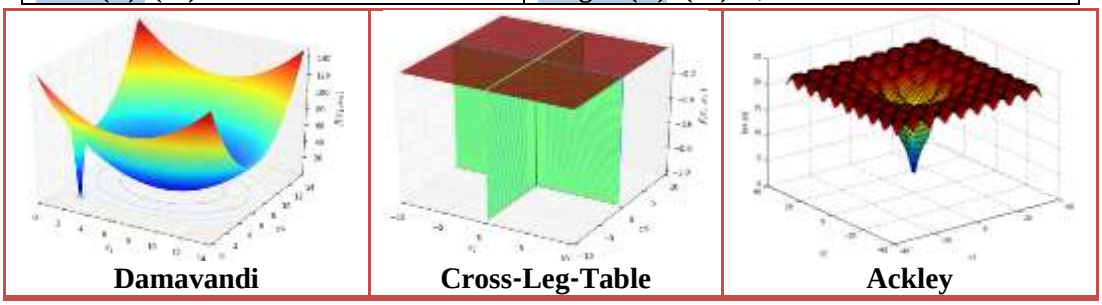

Figure 7. Graphs of a Sample of our Benchmark Functions

6.1. The ANT-BM Algorithm Performance

We start our empirical analysis by showing the performance of our algorithm, **ANT-BM**, using a set of carefully selected hard functions from Table 1. Table 2 shows these functions, their dimensions (*N*), and the actual minimum (True). It additionally shows the performance figures in terms of the average computed minimum and the CPU time (min, average, and max) required for computing this minimum in milliseconds (*ms*).

Referring to Table 2, our algorithm found the exact global minima for most of the functions and very close approximations of the minimum for the rest. (The exact solutions are shaded and **boldfaced**.) Furthermore, the algorithm performed really well in terms of CPU time. Observe that for all cases except for Trid (10) and Brown (10), the algorithm required a maximum time that is less than "0.85" seconds to find the minimum.

Table 2. ANT-BM's Performance in Terms of Solution Precision and CPU Time

Function (N)	Minimum		CPU Time (ms)		
	True	Average Computed	Min	Average	Max
Trid (2)	**–2**	**–2**	28	31	37
Trid (5)	**–30**	**–30**	249	266	280
Trid (10)	**–210**	**–210**	9,981	10,001	10,657
Adjiman (2)	–2.02181	–2.02180999	502	531	541
Brown (2)	0	1E–20	67	78	92
Brown (5)	0	3E–12	498	506	517
Brown (10)	0	6E–16	4,977	4,995	5,023
Chung Reynolds (2)	**0**	**0**	44	51	59
Chung Reynolds (5)	**0**	**0**	74	77	85

Chung Reynolds (10)	**0**	**0**	86	87	87
Dixon and Price (2)	0	1E−13	25	31	38
Dixon and Price (5)	0	2E−10	700	701	704
Powell Sum (2)	**0**	**0**	55	63	76
Powell Sum (5)	**0**	**0**	88	92	99
Powell Sum 10)	**0**	**0**	117	128	135
Giunta (2)	**0.064470**	**0.064470**	92	94	97
Salomon (2)	**0**	**0**	61	78	89
Salomon (5)	**0**	**0**	114	125	142
Salomon (10)	**0**	**0**	134	139	151
Cigar (2)	**0**	**0**	50	53	57
Cigar (5)	**0**	**0**	108	111	121
Cigar (10)	**0**	**0**	125	133	143
Whitley (2)	**0**	**0**	91	91	91
RANA (2)	**-511.73**	**-511.7328**	9	11	18
Goldstein Price (2)	**3**	**3**	47	62	74
Booth (2)	**0**	**0**	71	95	110
Easom (2)	**-1**	**-1**	91	124	131
Bukin06 (2)	0	4.7E−8	90	118	130
Cross-In-Tray (2)	**-2.06261**	**-2.06261**	8	9	15
Egg Holder (2)	-959.6407	959.6406	125	356	444
Damavandi (2)	0	1E−14	484	495	500
CrossLegTable (2)	**-1**	**-1**	78	83	95
Schaffer2 (2)	**0**	**0**	3	3	3
Shubert (5)	**-186.7309**	**-186.7309**	8	12	19
Matyas (2)	**0**	**0**	62	67	78
Test Tube Holder (2)	**-10.8723**	**-10.8723**	82	126	187
Pen Holder (2)	**-0.963534**	**-0.963534**	35	42	53
Levy 13 (2)	0	7.6E−22	134	245	368
Himmelblau (2)	0	6.2E−31	96	125	145
Beale (2)	0	1.02E−29	119	278	375
Holder Table (2)	**-19.20850**	**-19.20850**	25	28	32
Branin (2)	**0.397887**	**0.397887**	57	63	75
Trefethen (2)	**-3.3036868**	**-3.306868**	556	755	812
Egg Crater (2)	**0**	**0**	36	43	57
Crowned Cross(2)	**0.0001**	**0.0001**	14	16	17

Table 3 shows the performance of our algorithm ANT-BM, but for higher dimensional functions. The performance is represented in terms of true and average computed global minimum and CPU time (minimum, average, and maximum). (Again the exact solutions are **Boldfaced** and shaded.)

Table 3. ANT-BM's Performance for Higher Dimensional Functions

Benchmark		Global-Minimum		CPU-Time (milliseconds)		
Function	(N)	True	Average Computed	Min	Average	Max
Alpine 1	2	**0**	**0**	62	67	77
	10	**0**	**0**	136	141	157
	100	**0**	**0**	310	321	344
	300	**0**	**0**	544	666	702
	1000	**0**	**0**	1,587	1,802	2,111
Ackley	2	**0**	**0**	10	17	29
	10	**0**	**0**	26	32	41

	100	**0**	**0**	91	97	102
	300	**0**	**0**	215	219	223
	1000	**0**	**0**	398	485	513
Sphere	2	**0**	**0**	30	30	30
	10	**0**	**0**	52	58	62
	100	**0**	**0**	101	102	103
	300	**0**	**0**	205	268	305
	1000	**0**	**0**	483	496	508
Rastrigin	2	**0**	**0**	5	6	6
	10	**0**	**0**	9	11	11
	100	**0**	**0**	20	21	23
	300	**0**	**0**	41	42	43
	1000	**0**	**0**	104	112	119
Exponential	2	**-1**	**-1**	4	5	5
	10	**-1**	**-1**	7	9	15
	100	**-1**	**-1**	17	18	18
	300	**-1**	**-1**	30	31	33
	1000	**-1**	**-1**	76	77	77
Griewank	2	**0**	**0**	6	6	6
	10	**0**	**0**	11	13	13
	100	**0**	**0**	24	26	31
	300	**0**	**0**	49	50	50
	1000	**0**	**0**	134	135	135
Levy	2	0	1.5E−32	396	421	442
	10	0	1.4E−31	456	501	513
	100	0	1.5E−32	500	516	870
	300	**0**	**0**	1,351	1,452	1,604
	1000	0	1.5E−32	4,060	4,098	4,319
Schwefel	2	0	1E−12	7	8	9
	10	0	2E−6	121	134	162
Perm	2	0	2.7E−29	188	207	221
	10	0	6E−8	68,322	98,012	102,095
Rotated Hyper-Ellipsoid	2	**0**	**0**	80	94	121
	10	**0**	**0**	265	266	266
	100	**0**	**0**	888	950	996
	300	**0**	**0**	1,675	1,700	1,861
	1000	**0**	**0**	4,670	5,078	5,433
Sum of Different Powers	2	**0**	**0**	53	61	72
	10	**0**	**0**	200	200	200
	100	**0**	**0**	1,033	1,123	1,369
	300	**0**	**0**	2,567	2,755	2,809
	1000	**0**	**0**	3,987	4,659	5,001
Sums of Squares	2	**0**	**0**	37	39	46
	10	**0**	**0**	68	71	77
	100	**0**	**0**	207	222	235
	300	**0**	**0**	411	419	432
	1000	**0**	**0**	1,249	1,433	1,497
Zakharov	2	**0**	**0**	58	63	76
	10	**0**	**0**	94	111	125
	100	**0**	**0**	215	219	235
	1000	**0**	**0**	499	540	632

Michalewicz	2	**-1.8013**	**-1.8013**	113	135	147
	5	**-4.6876**	**-4.6876**	2,001	2,041	2,106
	10	**-9.66015**	**-9.66015**	4,964	5,112	5,555
Xin-She Yang 02	2	0	2.7E−320	87	111	115
	10	0	1.3E−319	145	172	191
	100	0	1.4E−318	343	359	366
	300	0	4E−318	703	718	719
	1000	0	1.3E−317	1,209	1,356	1,704
Xin-She Yang 03	2	**-1**	**-1**	3	11	16
	10	**-1**	**-1**	12	18	22
	100	**-1**	**-1**	46	47	47
	300	**-1**	**-1**	78	83	94
	1000	**-1**	**-1**	218	230	235
Xin-She Yang 04	2	**-1**	**-1**	3	16	18
	10	**-1**	**-1**	15	17	31
	100	**-1**	**-1**	31	42	49
	300	**-1**	**-1**	78	78	78
	1000	**-1**	**-1**	188	199	203
Drop-Wave	2	**-1**	**-1**	15	16	16
	10	**-1**	**-1**	15	16	16
	100	**-1**	**-1**	32	44	49
	300	**-1**	**-1**	47	53	62
	1000	**-1**	**-1**	125	134	141
Styblinski-Tang	2	**-78.332**	**-78.332**	9	15	16
	10	**-391.661**	**-391.6616**	93	94	94
	100	**-3916.16**	**-3916.616**	4,912	5,281	5,305
Leon	2	**0**	**0**	191	223	276
	25	**0**	**0**	311	467	901
	100	0	1.97E−33	1,987	2,243	2,617
Sargan	2	**0**	**0**	31	32	34
	50	**0**	**0**	102	116	124
	100	**0**	**0**	213	235	240
Cigar	25	**0**	**0**	95	119	121
	100	**0**	**0**	196	197	197
	1000	**0**	**0**	1,314	1,335	1,352
Egg Crater	30	**0**	**0**	178	189	206
	100	**0**	**0**	472	484	488
	1000	**0**	**0**	1,764	1,998	2,021

Generally speaking, the performance numbers in Tables 2 and 3 indicate high effectiveness of our algorithm. For instance, the algorithm found exact minimums for `CrossLegTable` and `XinSheYang03` functions and a very close approximation of the minimum for `Damavandi` function. The hardness index for these three functions is so high according to [8]. In fact, as reported in [8], only 0.25%, 0.83%, and 1.08% of the optimization algorithms have succeeded in finding reasonable approximations for the global minimum of respectively `Damavandi`, `CrossLegTable`, and `XinSheYang03`. Additionally, the CPU time is very reasonable. For instance, the average CPU time for the majority of the functions is less than "0.5" seconds (500 ms).

6.2. *Comparative Performance Analysis*

To further analyze the performance of our algorithm, **ANT-BM,** we compare it with many state-of-art algorithms. We call the state of art algorithms with which we compare: *reference algorithms*.

We first compare ANT-BM with PSO-3P [16], ELPSO [18], PSO-EO [20], NABC [21], and MSFLA-EO [19]. The authors in [17] also provided performance results for a large number of algorithms including ABC, MSFLA, GA, LX-PM, SFLA, and others (see [17] for more information) and they showed that their algorithm (PSO-3P) performed better than these algorithms.

Tables 4 and 5 show the functions, their dimensions, and the used algorithms (our algorithm ANT-BM is shaded for simplifying the comparison). The two tables additionally show the performance figures in terms of the average computed minimum and the CPU time. (Entries with asterisks "*" represent missing values—not reported in the original papers.)

As could be clearly seen in the two tables, our algorithm outperformed PSO-3P and the other algorithms in terms of either the solution precision or the CPU time or both. Consider, for instance, **Ackley** function in Table 4. The global minimum for this function is "0" (zero). Our algorithm found the exact minimum "0" for all the dimensions while PSO-3P found only approximated minimums with an average error ranging from *x*E−05 to *x*E−16 (where *x* is some value). The MSFLA-EO algorithm found the exact global minimum only for Ackley function with 30 variables. The authors of MSFAL-EO, however, reported only the best obtained minimum not the average minimum. In addition, it is unclear how the MSFLA-EO's performance will play out for functions with higher dimensions (say, higher than 50 variables).

If we consider the CPU times, we find that PSO-3P is faster than all of the other reference algorithms. However, our algorithm, ANT-BM, is not only faster than PSO-3P (and by default than the others), but also it requires significantly much less CPU time.

Table 4. The Performance of Our Algorithm ANT-BM versus PSO-3P and Others

Function	N	Algorithm	Minimum (average)	CPU Time (ms)
Ackley	10	ANT-BM	**0**	**32**
		PSO-3P	7.56E−05	655
	30	ANT-BM	0	43
		PSO-3P	1.40E−05	558
		ELPSO	0.2279	*
		MSFLA-EO	0	*
		PSO-EO	9.5E−04	260
		PSO	2.58	*
		SFLA	2.4	1,830
	50	ANT-BM	**0**	**66**
		PSO-3P	7.99E−15	632
	100	ANT-BM	**0**	**97**
		PSO-3P	8.88E − 16	1,040
		ELPSO	0.2255	*
	1000	ANT-BM	**0**	**485**
		PSO-3P	4.52E−05	555
Griewank	10	ANT-BM	0	**13**
		PSO-3P	0	338
	30	ANT-BM	0	**18**
		PSO-3P	1.61E−5	860
		ELPSO	2.27E−04	*
		MSFLA-EO	0	210
		ABC	4.52E−09	*
		NABC	1.13E−16	*

		PSO-EO	0	230
		PSO	0	520
	50	ANT-BM	0	**22**
		PSO-3P	0	646
		MSFLA-EO	0	2,060
	100	ANT-BM	0	**26**
		PSO-3P	0	589
	1000	ANT-BM	0	**135**
		PSO-3P	0	617

Table 5. The Performance of Our Algorithm versus PSO-3P and Other Algorithms

Function	N	Algorithm	Minimum (average)	CPU Time (ms)
Rastrigin	10	ANT-BM	0	**11**
		PSO-3P	0	1,101
	30	ANT-BM	0	**18**
		PSO-3P	0	584
		ELPSO	8.6403	*
		ABC	2.9E−09	*
		NABC	0	*
		PSO-EO	0	610
	50	ANT-BM	0	**19**
		PSO-3P	1.63E−07	482
		MSFLA-EO	0	9,410
	100	ANT-BM	0	**21**
		PSO-3P	8.48E−05	595
		ELPSO	5.5402	*
	1000	ANT-BM	0	**112**
		PSO-3P	1.28E−04	400
Sphere	10	ANT-BM	0	**58**
		PSO-3P	4.75E−08	569
	30	ANT-BM	**0**	**68**
		PSO-3P	1.51E–08	560
		ELPSO	5.244E–08	*
		ABC	3.75E–10	*
		NABC	4.75E–16	*
	50	ANT-BM	**0**	**76**
		PSO-3P	3.03E−08	539
	100	ANT-BM	**0**	**102**
		PSO-3P	1.27E–07	569
		ELPSO	6.035E–05	*
	1000	ANT-BM	**0**	**496**
		PSO-3P	3.64E−08	599
Zakharov	10	ANT-BM	**0**	**111**
		PSO-3P	2.33E−05	603
	30	ANT-BM	**0**	**125**
		PSO-3P	8.79E–05	643
		ELPSO	0.0003	*
	50	ANT-BM	**0**	**170**
		PSO-3P	2.11E−05	783
	100	ANT-BM	**0**	**219**

		PSO-3P	8.14E–02	944
		ELPSO	2.5016	*
	1000	ANT-BM	0	**540**
		PSO-3P	0	1,268

We secondly compare our algorithm with EOCSO [14]. The authors reported the performance numbers for their algorithm (EOCSO) and additionally compared EOCSO with other effective algorithms. The performance numbers in [14] showed that EOCSO is superior to the other algorithms which EOCSO is compared with. The reported evaluation in [14] includes functions with just 30 variables and only one function with 4 variables.

Table 6 shows the benchmark functions, their dimensions, the performance numbers of our algorithm ANT-BM, EOCSO, and other algorithms reported in [14]. In general terms, our algorithm performed better than the reference algorithms. In terms of the solution, our algorithm was able to find exact minimums for most of the functions. Our algorithm in addition, found better approximations of the minimum for the rest of the functions. For instance, our algorithm found the exact minimum for the Sphere function (with dimension 30) while EOCSO produced an approximation of 1.19E–201. Consider as another example "Schwefel 2.22" function (dimension 30) whose global minimum equals to "0". Our algorithm found an approximated minimum of 1.61E–299 while the approximated minimum by EOCSO is 6.6E–116. No need to mention the other algorithms in Table 6 since their performance is inferior to EOCSO and by default to our algorithm.

Although [14] did not report time performance, we recorded the CPU time required by our algorithm to produce the solutions.

Table 6. The Performance of Our Algorithm versus EOCSO and Others

Function[16]	N	Algorithm	Minimum		CPU Time (ms)
			True	Computed	
Sphere	30	ANT-BM	0	**0**	68
		EOCSO	0	1.19E–201	*
		CSO	0	1.19E–55	*
		DE	0	3.1E–12	*
		PSO	0	6.78	*
		BA	0	9.4E–6	*
		ACO	0	9.9E–8	*
		FPA	0	6.3E–6	*
Schwefel 2.22	10	ANT-BM	0	**0**	64
	30	ANT-BM	0	4.5E–322	106
		EOCSO	0	6.6E–116	*
		CSO	0	1.91E–43	*
		DE	0	3.04E–8	*
		PSO	0	6.59	*
		BA	0	0.012	*
		ACO	0	4.94E–5	*
		FPA	0	1.97E–7	*
	100	ANT-BM	0	5.56E–299	125
	1000	ANT-BM	0	5.7E–298	574
Schwefel 1.2	10	ANT-BM	0	0	37
	30	ANT-BM	0	**0**	51
		EOCSO	0	4.49E–51	*
		CSO	0	7E+1	*
		DE	0	15255.14	*

		PSO	0	4.03E+2	*
		BA	0	2.27E–5	*
		ACO	0	11951.1	*
		FPA	0	2.52E–2	*
	100	ANT-BM	0	0	119
	1000	ANT-BM	0	0	4,343
Rosenbrock	10	ANT-BM	0	0	120
	30	ANT-BM	0	**1.3E–149**	24,621
		EOCSO	0	9.53E–9	*
		CSO	0	26.5	*
		DE	0	25.83	*
		PSO	0	343.6	*
		BA	0	0.196	*
		ACO	0	17.14	*
		FPA	0	2.12E–6	*
	100	ANT-BM	0	0	134,100
	1000	ANT-BM	0	4.45E–207	361,045
Rastrigin	30	ANT-BM	0	0	18
		EOCSO	0	0	*
		CSO	0	0	*
		DE	0	47.02	*
		PSO	0	71.56	*
		BA	0	30.84	*
		ACO	0	108.47	*
		FPA	0	15.92	*
Schwefel 2.26	10	ANT-BM	-4189.829	-4189.82887	1,036
	30	ANT-BM	-12569.487	**-12569.48661**	14,739
		EOCSO	-12569.487	-12474.618	*
		CSO	-12569.487	-9133.94	*
		DE	-12569.487	-12546.71	*
		PSO	-12569.487	-7963.94	*
		BA	-12569.487	-8502.89	*
		ACO	-12569.487	-6090.03	*
		FPA	-12569.487	-10694.82	*
Ackley	30	ANT-BM	0	**0**	43
		EOCSO	0	4.4E–15	*
		CSO	0	4.4E–15	*
		DE	0	4.5E–7	*
		PSO	0	4.91	*
		BA	0	11.60	*
		ACO	0	6.5E–5	*
		FPA	0	2.013	*
Kowalik	4	ANT-BM	0.0003074861	**0.0003074861**	588
		EOCSO	0.0003074861	0.0003074860	*
		CSO	0.0003074861	0.000330135	*
		DE	0.0003074861	0.000368256	*
		PSO	0.0003074861	0.000307498	*
		BA	0.0003074861	0.000307834	*
		ACO	0.0003074861	0.001325092	*
		FPA	0.0003074861	0.000307486	*

We thirdly compare our algorithm with WOA [28], FMA [23], COA [24], ALC-PSO [26], and DE [25]. The authors in [24] reported performance numbers for functions with only 30 variables. We confined the comparisons to functions with only 30 variables.

Table 7 shows the performance of our algorithm (ANT-BM) and the other algorithms. The table shows also the time requirements for our algorithm (ANT-BM) only since no reported CPU time for the rest. As the performance numbers show, our algorithm performed better than all the other algorithms FMA, COA, ALC-PS and DE especially for hard functions **F1**, **F2**, and **F5**. For the other functions, our algorithm's performance was equivalent to the others except for "Schwefel 2.22" function, where the algorithms FMA, COA, and DE actually performed slightly better than ours.

Table 7. The Performance of ANT-BM versus Other Algorithms

Function	N	Algorithm	Minimum		Time (ms
			True	Computed	
Sphere	30	ANT-BM	0	0	68
		FMA	0	0	*
		COA	0	0	*
		DE	0	0	*
		ALC-PSO	0	1.13E-172	*
		WOA	0	1.41E-30	*
F1= $\sum_{i=1}^{n} ix^4 + Random(0,1)$ -1.28≤ xi≤1.28	30	ANT-BM	0	**4.93E–8**	1,854
		FMA	0	0.364	*
		COA	0	3.44E–5	*
		DE	0	7.7E–7	*
		WOA	0	0.001425	*
Sum of different powers	30	ANT-BM	0	0	409
		FMA	0	0	*
Schwefel 2.22	30	ANT-BM	0	4.5E–322	106
		FMA	0	0	*
		COA	0	0	*
		DE	0	0	*
		ALC-PSO	0	1.2E–98	*
		WOA	0	1.06E–21	*
Rosenbrock	30	ANT-BM	0	1.3E–149	24,621
		FMA	0	0	*
		COA	0	4.12E–4	*
		ALC-PSO	0	3.7E–7	*
		DE	0	0	*
Griewank	30	ANT-BM	0	0	18
		FMA	0	0	*
		COA	0	0	*
		ALC-PSO	0	0	*
		DE	0	0	*
		WOA	0	0.0000289	*
F2= $\sum_{i=1}^{N/4} (x_{4i-3}+10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i-2})^2 + (x_{4i-2} - x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4$	30	ANT-BM	0	**0**	433
		FMA	0	7.58E–10	*
F3= $\sum_{i=1}^{N/4} (y_i^2 - 10\cos(2\pi y_i) + 10)$	30	ANT-BM	0	0	187
		FMA	0	0.252	*

$y_i = \begin{cases} x_i; \text{ if } \lvert x_i \rvert < 0.5 \\ Round(2x_i)/2; \text{ otherwise} \end{cases}$ $-5 \le x_i \le 5$		ALC-PSO	0	0	*
F4= $\sum_{i=1}^{n} ix^4$, $-1.28 \le x_i \le 1.28$	30	ANT-BM	0	0	821
		FMA	0	0	*
F5=$4x_1^2 - 2.1x_1^4 + 1/3x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$ $-5 \le x_i \le 5$	2	ANT-BM	-1.03163	**-1.03163**	200
		FMA	-1.03163	-1.03164	*
		COA	-1.03163	-1.0316	*
		WOA	-1.03163	-1.03163	*
Rastrigin	30	ANT-BM	0	0	18
		FMA	0	0	*
		ALC-PSO	0	7.1E−15	*
		DE	0	0	*
Ackley	30	ANT-BM	0	0	43
		FMA	0	0	*
		COA	0	8.88E−16	*
		ALC-PSO	0	1.69E−15	*
		DE	0	4.44E−16	*
		WOA	0	7.4043	
Sum of Squares	30	ANT-BM	0	0	72
		FMA	0	0	*

7. Conclusions and Future Work

This paper proposed an innovative algorithm for optimization problem. The algorithm is based on guided-random search technique augmented with biased mapping. The random search is guided by feedbacks. The feedbacks help the algorithm effectively identify the promising regions for the next round and dynamically adjust the search parameters. The biased mapping provides the algorithm with an effective mechanism that not only maps randomly generated candidate solutions to these identified promising regions of the variables' domain but also dynamically controls the amount of these candidate solutions mapped to these regions. The biased mapping has yet another feature: it does not ignore the less promising regions (or marginal regions).

We tested our algorithm using benchmark functions set. The benchmark functions tested compromise a comprehensive testing suite for any optimization algorithm. The testing results were really promising. These results showed that our method is both (1) effective in finding the exact global minimum (or in some cases a so-close approximation for it) and (2) efficient in terms of CPU time. In fact, as presented in the performance analysis section (subsection 6.1), our algorithm was able to find exact solutions for very hard problems (e.g. *XinSheYang03* function) in a short CPU time. We compared our algorithm with many state-of-art algorithms that adopt different search techniques. Our algorithm performed better than these algorithms in almost all the cases.

We have two directions for future work. First, we want to use more effective random generator than the computer built-in one. Second, we want to extend our algorithm to find the best path that leads to the global minimum rather than the minimum value per se.

References

[1] O. Krause, D. R. Arbonès, and C. Igel , "CMA-ES with Optimal Covariance Update and Storage Complexity", 29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain; **(2016)**, pp. 370–378.
[2] A. Mohamed and H. Sabry, Constrained Optimization Based on Modified Differential Evolution Algorithm, Information Sciences, vol. 194, **(2012)**, pp.171–208.
[3] R. H. Abiyev and M. Tunay, Optimization of High-Dimensional Functions through Hypercube Evaluation, Computational Intelligence and Neuroscience, vol. 2015, **(2015)**. doi:10.1155/2015/967320
[4] J. Zhang, A.C. Sanderson. JADE: Self-Adaptive Differential Evolution with Optional External Archive. IEEE Transactions on Evolutionary Computation, vol. 13, no. 5, **(2009)**, pp. 945–958.
[5] Benchmark Functions. https://www.sfu.ca/~ssurjano/optimization.html, January **(2018)**.
[6] M. Jamil and X.-S. Yang, "A Literature Survey of Benchmark Functions for Global Optimization Problems", International Journal of Mathematical Modeling and Numerical Optimization (IJMMNO), vol. 4, no. 2, **(2013)**, pp. 150–194.
[7] A. Gavana, Global Optimization Benchmarks, http://infinity77.net, January **(2018)**.
[8] E. Cuevas, M. Cienfuegos, D. Zaldívar, and M. Pérez-Cisneros, A Swarm Optimization Algorithm Inspired in the Behavior of the Social-Spider, Expert Systems with Applications, vol. 40, no.16, **(2013)**, pp. 6374-6384.
[9] X. Meng, Y. Liu, X. Gao X., H. Zhang, "A New Bio-inspired Algorithm: Chicken Swarm Optimization", In: Tan Y., Shi Y., Coello C.A.C. (eds) Advances in Swarm Intelligence. ICSI 2014. Lecture Notes in Computer Science, vol. 8794, **(2014)**, pp. 86-94.
[10] G. G. Wang, B. Chang and Z. Zhang, A Multi-Swarm Bat Algorithm for Global Optimization, 2015 IEEE Congress on Evolutionary Computation (CEC), Sendai, **(2015)**, pp. 480-485. doi: 10.1109/CEC.2015.7256928
[11] Y. Zhou, J. Xie, L. Li, and M. Ma, Cloud Model Bat Algorithm, The Scientific World Journal, vol. 2014, **(2014)**. http://dx.doi.org/10.1155/2014/237102
[12] P. W. Tsai, J. S. Pan, B. Y. Laio, and V. Istanda, "Bat Algorithm Inspired Algorithm for Solving Numerical Optimization Problems", Applied Mechanics and Materials, vol. 148, no. 149, **(2012)**, pp. 134-137,.
[13] W. C. Feng, L. Kui, and S. P. Ping, "Hybrid Artificial Bee Colony Algorithm and Particle Swarm Search for Global Optimization", Mathematical Problems in Engineering Journal, vol. 2014 , **(2014)**.
[14] C. Qu, S. Zhao, Y. Fu, and W. He, "Chicken Swarm Optimization Based on Elite Opposition-Based Learning, Mathematical Problems in Engineering, vol. 2017, **(2017)**.
[15] I. A. Etukudo, "Optimal Designs Technique for Solving Unconstrained Optimization Problems with Univariate Quadratic Surfaces", American Journal of Computational and Applied Mathematics, vol. 7, no. 2, **(2017)**, pp. 33–36.
[16] S. Gerardo de-los-Cobos-Silva, M. Á ngel Gutiérrez-Andrade, and et al, "An Efficient Algorithm for Unconstrained Optimization", Journal of Mathematical Problems in Engineering, vol. 2015, **(2015)**.
[17] A. R. Jordehi, "Enhanced Leader PSO (ELPSO): A New PSO Variant for Solving Global Optimization Problems", Applied Soft Computing Journal, vol. 26, **(2015)**, pp. 401–417.
[18] X. Li, J. Luo, M.-R. Chen, and N. Wang, "An Improved Shuffled Frog-Leaping Algorithm with Extremal Optimization for Continuous Optimization", Information Sciences, vol. 192, **(2012)**, pp. 143–151.
[19] M.-R. Chen, X. Li, X. Zhang, and Y.-Z. Lu, "A Novel Particle Swarm Optimizer Hybridized with Extremal Optimization", Applied Soft Computing Journal, vol. 10, no. 2, **(2010)**, pp. 367–373.
[20] Y. Xu, P. Fan, and L. Yuan, "A Simple and Efficient Artificial Bee Colony Algorithm", Mathematical Problems in Engineering, vol. 2013, **(2013)**.
[21] M. Farahani, S. B. Movahhed, S. F. Ghaderi, "A Hybrid Meta-Heuristic Optimization Algorithm based on SFLA", Proceedings of the 2nd International Conference on Engineering Optimization, pp. 1–8, Lisbon, Portugal, Sept. **(2010)**.
[22] S. Chapra and R. Canale, Numerical Methods for Engineers, 7th edition, McGraw-Hill Education, **(2014)**.
[23] A. Ritthipakdee, A. Thammano, N. Premasathian, and D. Jitkongchuen. Firefly Mating Algorithm for Continuous Optimization Problems. Computational Intelligence and Neuroscience Journal, Vol. 2017, **(2017)**. https://doi.org/10.1155/2017/8034573
[24] M. Li, H. Zhao, X. Weng, and T. Han, Cognitive Behavior Optimization Algorithm for Solving Optimization Problems, Applied Soft Computing Journal, vol. 39, **(2016)**, pp. 199–222. https://doi.org/10.1016/j.asoc.2015.11.015
[25] D. Jitkongchuen and A. Thammano, "A Self-Adaptive Differential Evolution Algorithm for Continuous Optimization Problems", Artificial Life and Robotics, vol. 19, no. 2, **(2014)**, pp. 201–208.
[26] W.-N. Chen, J. Zhang, Y. Lin et al., "Particle Swarm Optimization with an Aging Leader and Challengers", IEEE Transactions on Evolutionary Computation, vol. 17, no. 2, **(2013)**, pp. 241–258.
[27] S. Mirjalili, S. M Mirjalili, and A. Hatamlou, "Multi-Verse Optimizer: a Nature-Inspired Algorithm for Global Optimization", Neural Computing and Applications, vol. 27, no. 2, **(2016)**, pp. 495-513.

[28] S. Mirjalili, A. Lewis, the Whale Optimization Algorithm. Advances in Engineering Software, vol. 95, **(2016)**, pp. 51-67.
[29] Well-known benchmark functions. http://benchmarkfcns.xyz/fcns, **(2018)**.
[30] M.N.A. Wahab, S. Nefti-Meziani, and A. Atyabi, "A Comprehensive Review of Swarm Optimization Algorithms", PLOS ONE, vol. 10, no. 5, **(2015)**.

