Random –Guided Search Algorithm for Complex Functions

Muhammed Jassem Al-Muhammed[1]
and
Raed Abu Zitar[2]

Faculty of Information Technology[1,2]
American University of Madaba,
Madaba, Jordan
m.almuhammed@aum.edu.jo
r.abuzitar@aum.edu.jo

Abstract

Optimization is a general goal that has many applications in Engineering, Business, computer science and almost in every operation in life. Devising ways for handling problem optimization is an important yet a challenging task. We look for techniques that are efficient, accurate, and applicable. The search space could have any nature and could have discontinuity or multi-local optima. . In this paper, we address this challenge by offering an algorithm that combines the random search techniques with both an effective mapping and a dynamic adjustment of its search behavior. Our proposed algorithm automatically builds two types of triangles over the unity intervals: principal and marginal. These triangles guide the search within both the effective regions of the search domain that most likely contain the optima and the marginal regions of the search domain that less likely contain the optima. Experiments with our prototype implementation showed that our method can effectively find the global optima for rather complicated mathematical functions chosen from well-known benchmarks and perform better than other algorithms.

Key words: Probability-directed optimization, principal triangle, unconstrained problem optimization, effective search interval

1. Introduction

The performance of many systems can be defined by mathematical functions that capture the important properties of these systems. For instance, the performance of a company depends on its revenue, which is a function of variables such as the amount of the sales. Finding the optima for these functions is vital because it enables for better design decisions for the system captured by these functions. Unfortunately, most of the mathematical functions that model real world systems are difficult to optimize. They are mostly not continuous in their domain, have multiple local optimal values, not differentiable, and so on. This is likely to make the optimization of these functions using analytical means so difficult if not impossible.

Researchers have devised a large number of methods to optimize functions [1][2][10][11]. Some of the methods use the functions' properties [5] while others use heuristic techniques [1][3][9][20]. Examples of latter methods include Tabu Search [13], Simulated Annealing [4], Genetic Algorithms [14][15], Scatter Search [17], and particle swarm optimization [5][7][8][12][16][18]. These methods use random-based techniques and some heuristics to guide the search within the domain toward the optimal values of the functions. Although these methods can be effective in finding the optimum, they are difficult to use and are computationally expensive [2].

The paper proposes an innovative algorithm for finding the global optimal values for functions, which may lack continuity and differentiability and may also have many local

optimums. Our algorithm uses an innovative random search technique that quickly derives the search to the global optimal value of the function. To achieve this, the algorithm builds triangle probability triangle s on the unity interval (i.e. [0, 1]) and employs effective mapping techniques that map random numbers generated from the unity interval to corresponding points in probability triangle s. This mapping quickly directs the search to the global optima. Our algorithm, in addition, can dynamically adjust both the search space and the probability triangle s using gained knowledge during the search; thereby always moving the probability triangle and consequently the mapping to the promising regions of the domains.

The paper makes the following contributions. First, it provides an approach that is fully guided by probability triangle s that identifies and moves to the regions of the domain in which the optima most likely reside without ignoring the other regions in which the optima less likely reside. Second, the algorithm can dynamically adjust its probability triangle s thereby quickly find the optima. Third, the algorithm performs better than other approaches in terms of both approximating the global optima and the execution time.

We present our contributions as follows. Section 2 formalizes the problem for which we define the solution. Section 3 describes our algorithm and Section 4 provides a proof of its convergence to the solution. Section 5 evaluates the performance of algorithm. We conclude and give directions for future work in section 6.

2. Problem Formalization

Let f(x_1, x_2, …, x_n) be a function, where the variables $x_i \in D_i$ (i = 1, 2, …, n) and D_i is a bounded interval [a_i, b_i]. Our objective is to solve the following problem.

(1)

That is, the objective is to find the values  from the domains D_i such that the function f is in its global optimal value. The optimal value for the function f is either the minimum or the maximum value.

We impose no constraints on the function  . It can be linear or non-linear, continuous in its domain or not, have derivatives or not, and have multiple local optimums. This type of functions can be largely found in real world applications. In fact, it is unlikely that the models (functions) of real world applications can result in functions with good properties that make them solvable using analytical means.

3. The Moving Triangles Algorithm

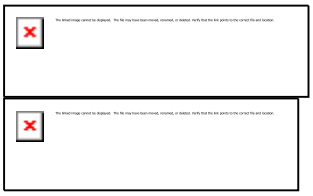
This section describes the fundamental components of the moving triangle algorithm. We specifically discuss the triangle probability triangle s in subsection 3.1 and the mapping between random values generated from the unity interval and the probability triangle s in subsection 3.2. We discuss the convergence conditions in subsection 3.3. Finally, we present the technical details of our algorithm in subsection 3.4.

3.1. The Triangle Probability Triangle

Let  be a function, where $x_i \in [a_i, b_i] \subset \Re$ (i = 1, 2, …,n). The most fundamental part of the moving triangles algorithm is the definition of the probability triangle s. In our approach, the algorithm dynamically creates triangle probability triangle s at the unity interval (i.e. on the interval [0, 1]) for each of the function's variables x_i (i = 1, 2, …, n). Two types of triangle probability triangle s are created at the unity interval [0, 1] as shown in Figure 1(a). The first type is centered at the middle point of the unity interval 0.5 and whose radius (half base of the triangle) is q_i (i = 1, 2, …, n). We call these probability triangles the *principal triangles*. We also call the parts of the unity intervals on which the principal triangle

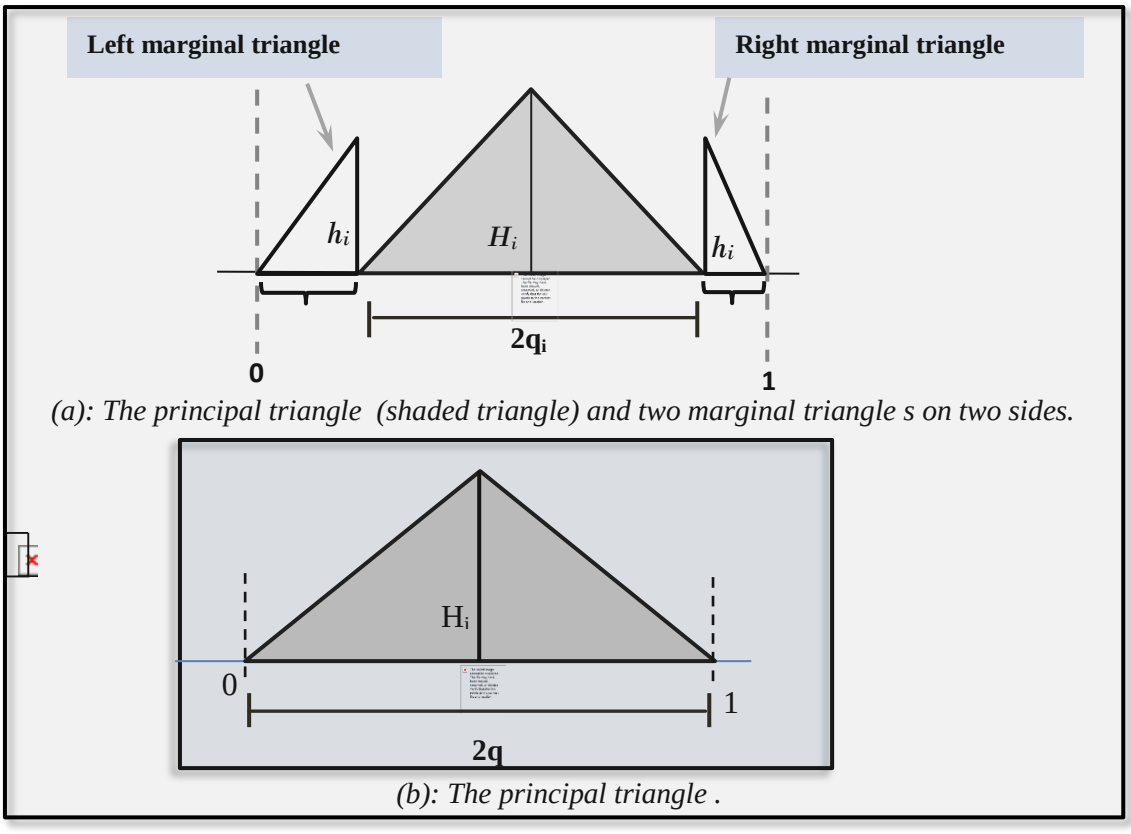
s are built the *effective search intervals*.[1] The second type is built on the parts of the unity intervals that are not covered by the principal triangle (1-2q_i). We call these probability triangles the *marginal triangles* and we call the parts of the intervals on which they are built the *marginal search intervals*.[2]

The principal and marginal triangles are fully defined by their heights and bases. The principal triangle is isosceles or right-angled triangle whose area is q_i * H_i, where q_i is half of the base of the triangle and H_i is its height. Likewise, the marginal triangle is right-angled triangle whose area is (1-2q_i) * h_i, where h_i the height of the triangle and (1-2q_i) is base length. The heights of the principal triangle and the marginal triangle s H_i and h_i are given by the following formulas.



(2)

Based on the definitions of H_i and h_i above, the area of the principal triangle is at least 0.75 while the area of the marginal triangle s is at most 0.25. Furthermore, the sum of the areas of principal and marginal triangle s always equal to 1 regardless of the changes to the quantities q_i's.[3]



(a): The principal triangle (shaded triangle) and two marginal triangle s on two sides.

(b): The principal triangle .

Figure 1: The triangles that are built on the unity interval [0, 1].

[1] We call these intervals the effective search intervals because-as we will see later-the probability that they have the optimal value for the function is large.

[2] There are at most two marginal triangle s. We also call it marginal because the probability that they have the optimal value is much lower than that of principal triangle .

[3] We point out that the numbers 0.75 and 0.25 that appear in (2) are adjusted experimentally after training our algorithm on a large collection of functions.

It is important to note that according to definition of the heights in (2), the area of the principal triangle gradually grows to become larger than 0.75 as the radius q_i gradually shrinks. On the contrary, the area of the marginal triangle s gradually shrinks to become smaller than 0.25 as q_i gradually shrinks. In addition, when q_i equals 0.5, there will be only the principal triangle , which then covers the entire unity interval as shown in Figure 1 (b).

3.2. Mapping from Unity interval to Principal / Marginal Triangle s

The second fundamental concept in our algorithm is the mapping between the unity interval and the triangle probability triangle s. The important property of the mapping is that it must depend on the areas of the triangle s (i.e. is proportional to the area of the triangle s). More precisely, we require our mapping to map more points from the unity interval to the probability triangle with the larger area.

The easiest case in the mapping is when the radius of the probability triangle equal 0.5. That is because the principal triangle covers the entire unity interval as it clearly appears in Figure 1(b). Thus, there is a perfect correspondence between the points of the unity interval and those that belong to the base of the principal triangle. As a result, the mapping is straightforward: each value $\gamma_i \in [0, 1]$ is mapped to itself.

When, however, we have both the principal and marginal triangle s, mapping points $\in [0, 1]$ to one of these triangles becomes a little bit tricky as this mapping becomes proportional to the area of each triangle. Let $L_i = (C_i - q_i) * h_i$ and $R_i = (1\text{-}C_i - q_i) * h_i$ be the areas of the left and the right marginal triangle s respectively (Figure 1(a)). We map a point $\in [0, 1]$ to the principal or to the marginal triangle s using the logic in Figure 3.

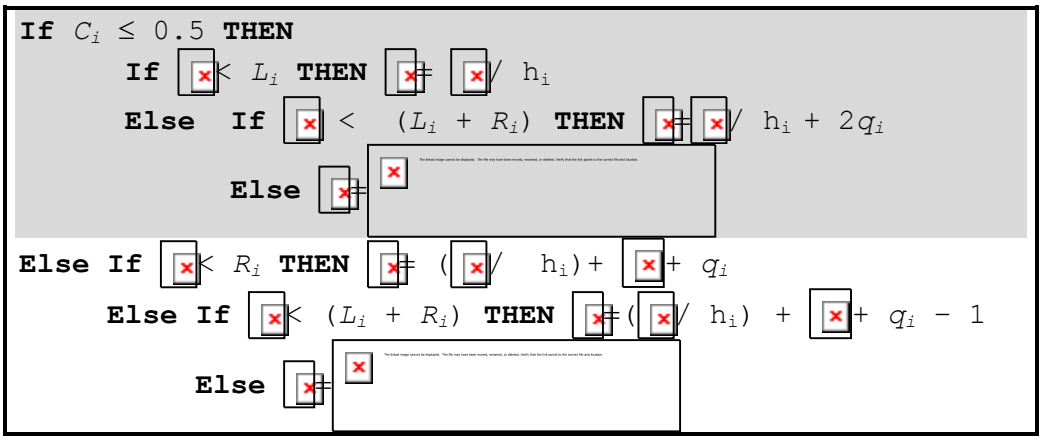


Figure 3: The mapping logic from the unity interval [0, 1] to the principal/marginal triangle s.

Broadly speaking, the mapping distinguishes between two cases based on the location of the center C_i of the principal triangle within the unity interval. In particular, when the center of the principal triangle is less than or equal to 0.5 we have (the shaded part). When the center is greater than 0.5, we have the (un-shaded part). In all cases we check whether the point is less than the area of the left marginal triangle (L_i) or less than the area of the right marginal triangle (R_i).

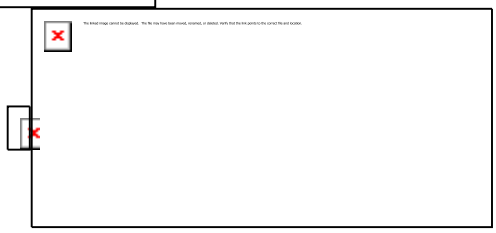
3.3 The Halting Conditions

Let and be the values of the variables x_1, x_2, …, x_n in two consecutive rounds i and i+1 and for which the function f was in the best optimal value. Suppose also that and are the optimal values for f in the rounds i and i+1

respectively. We define the convergence conditions to the optimal value of a function f as follows.

(3)

The amount ε is sufficiently small real number and depends on the required accuracy of the solution. The amount  is defined as follows.



3. 4. The algorithm

The moving triangles algorithm is iterative. It searches for the global optimal value of a function $f(x_1, x_2, \ldots, x_n)$ by performing a number of rounds until the convergence conditions (3) hold. During a round r (r = 1, 2, …), the algorithm conducts many experiments each of which consists of k steps. In each step, it generates n random numbers  in the interval [0, 1] using the computer built-in random generator and maps them to base of the triangle probability triangle s (principal or marginal) using the logic in Figure (3). The main objective of this mapping to the triangle is to direct and therefore focus the search into these parts of the intervals that most likely contain the values of the variables x_i for which the function is in the global optimal value; thereby expediting the convergence to the solution. Specifically, the algorithm uses the principal triangle to focus most of the search in the effective search intervals since these parts of the intervals are most likely to contain the global optimal value. Furthermore, it uses the marginal triangle s to cover the parts of the intervals (the marginal search intervals) that are less likely to contain the optimal value; thereby avoiding missing the optimal value or getting trapped in local optima.

In any subsequent round r, the algorithm uses the information gained in previous round r-1 to adjust the parameters of the principal and the marginal triangle s. Particularly, the centers of the principal triangle C_i are moved to the values that produced the best optimal value for the function in the previous round. The rationale is that: the solution is most likely resides in the vicinity of these values.

Figure 4 shows the technical steps of the moving triangles algorithm. The algorithm starts the search by having the principal triangle s cover the entire unity interval [0, 1]. The centers C_i of the principal triangle s are therefore at the centers of the unity intervals (i.e. at the point 0.5) and the radiuses q_i's of the principal triangle s equal to 0.5.

The algorithm then generates n random numbers $\psi_i \in [0, 1]$ for each variable x_i and maps them to the base of one of the triangles using the logic in Figure (3). The mapping yields the random numbers γ_i (i =1, 2, …, n), which belong to the bases of the triangle triangles (principal or marginal). Because the principal triangle has a larger area, more of random numbers ψ_i will be mapped to it and less points are mapped to the left and right triangle s.

To plug the random numbers γ_i in the function's variables, these numbers must be first mapped to the actual domains of function's variables. Mapping γ_i to the actual domains ($[a_i, b_i]$) of the variables is straightforward and is done using the following transformation in (4)

 (4)

The algorithm computes the value of f at this point  to get the value F. If the value F is better than the previous  the algorithm keeps this value along with other

information. Specifically, it keeps: [image], [image], and [image]. The algorithm repeats this process *m* times before going to another experiment.

After performing *m* steps, the algorithm reduces the radiuses of the principal triangle s and therefore enlarges their areas using the formula [image], where d >1.[4] Performing another experiment is subjected to the values q_i. If the value of $q_i < \varepsilon$ for all *i*, the algorithm halts the round *r*. If, however, $q_i < \varepsilon$ does not hold for some *i*, the algorithm initiate another experiment.

```
FOR i← 1 to n DO
   Ci = 0.5
   qi = 0.5
[image] /** initially the optimal value is set to a large value */
REPEAT
    j = 1
    WHILE (qi > ε for all i) DO
        FOR k← 1 to m DO /** m steps in each experiment */
              FOR i← 1 to n DO
                  Generate a random number ψi ∈ [0, 1]
                  γi = MAP (ψi) using the logic in Figure 3
                  [image]
              F = f([image]) /** compute the function f at ([image]) */
              IF F is better than [image]
                  [image] = F
                  [image]
                  [image]
                  [image]
          END FOR (K)
          FOR i← 1 to n DO
             qi = qi/d /**reduce the radius qi by d*/
             Compute Hi and hi from Formula(2)
        END WHILE (qi > ε)
        FOR i← 1 to n DO
             Ci = γi
             IF Ci <=0.5 qi = Ci
             ELSE qi = 1- Ci
        j= j + 1
UNTIL Halting Conditions (Figure 3) hold.
```

Initially, the centers of the principal triangle s are at the center of the unity interval [0, 1].

Transform the random numbers to the actual variable intervals [a_i, b_i]

Keep the so-far best optimal value in round *j* along with the values of its variables and the random numbers that produced this value.

Change the centers of the principal triangle to the new points that has given the best value in round j

Figure 4: the technical steps of the Moving Triangles Algorithm.

Launching the round *r* + 1 depends on the convergence conditions. If they hold, the algorithm ends the search and prints the results. Otherwise, the algorithm changes the centers

[4] The value of *d* can be theoretically any real number greater than 1. However, as *d* becomes larger the algorithm converges faster, but the resulting optimal value may suffer. While experimenting with our prototype we found that 1< d ≤ 2 is good enough. We have not tested for larger value of *d*.

of the principal triangle s C_i's to the point  that has resulted in the best optimal value in round r and calculates the corresponding radiuses q_i 's using the following logic.

```
FOR i = 1 to n DO
    
    IF Ci ≤ 0.5 THEN qi = Ci
    ELSE    qi = 1 - Ci
```

Observe, because of the logic above, the triangles (principal and marginal) move within the unity intervals. As such, the mapping in Figure 3 condenses the points in the new parts of the intervals on which the principal triangle s are built.

4. Performance Analysis

We analyze in this section the time complexity of our algorithm. We first find its upper bound time complexity. We then present and discuss the results of our experiments conducted using many functions obtained from benchmarks.

4.1. The Time Upper Bound

Referring to Figure 4, the algorithm has three nested loops that define the major computations. The first loop (**REPEAT** loop) iterates until the halting conditions hold. Although the number of iterations may vary depending on the function, it is finite and bounded by some integer *n*. The second loop (**WHILE** loop) iterates until the radiuses of the principal triangle s become less than some pre-specified threshold. Again, the exact number of iterations may vary depending on the length of the interval and the reduction factor, but it is, nevertheless, bounded by some integer *k*. The third loop (**FOR** loop) iterates a fixed number of iterations *m*.

The other computations such as generating random numbers using the build-in random generator and calculating the parameters of the probability triangle s require constant time and therefore we ignore them. Additionally, we ignore also the time required for calling the function and evaluate it since this time depends on the implementation platform.

Due to the fact that these three loops are nested, we can represent the time complexity of the algorithm in terms of the number iterations as $O(m*n*k)$. In other words, the time complexity of the algorithm is cubic. According to our observations during the experimental study, the integers n and k are much smaller than m. In fact, m may equal few tens (only 50 in our experiments) while n and k are much smaller. Based on this observation, we can claim that the upper bound of the actual time complexity of the algorithm is $O(m^3)$.

4.2. Empirical Analysis

We conducted many experiments using our prototype implementation. We implemented our prototype using Java programming language. The execution platform was a laptop with core 2 Dou Processor (1.7 GHz) and 2 GB main memory. The operating system was windows 7 (32 bits).

We tested our algorithm using functions from the benchmarks [1][19]. These benchmarks serve dual objectives. First, we want reference functions with known global optimums so that we can validate the effectiveness of our algorithm in finding the global optima. That is, we want to measure the quality of approximating the global optima. Second, we want functions whose global optima are difficult to locate in order to show the true performance of the algorithm. These benchmarks serve both objectives because they have challenging functions; each one has only one global optimum that is surrounded by too many local optimums. Figure 5 shows the graphs of a sample of these functions. As the graphs clearly illustrate, finding the global optima for these functions is extremely hard since these global optima have too many local optimums in their neighborhood.

Table 1 shows the first set of functions that we used in the testing. These functions have the same different global minima. We grouped the functions according to their dimensions. Note we ignore the differences in the difficulty of each function in a group. We then used our algorithm to find the optimum for every function in a specific group. Because the search is random, but nevertheless directed, the execution time required to find the global optimum for a function may slightly vary from execution to another. To count for this variation in the execution time and obtain a better estimation, we executed the algorithm 30 times for each function. We calculated the average time of the thirty executions and also saved the minimum and maximum time. We performed the same procedure for all the functions in a specific group. We finally calculated the average time for finding the optima for all the functions in a specific group and saved the minimum and the maximum time. We also recorded the global optima as the worst approximation of the global optima for all the functions in the group. (The worst approximation is the calculated global optimum value that has the largest error. For instance if 1E−18 and 1E−20 two approximations of the global optimum 0, 1E−18 is the worst approximation.)

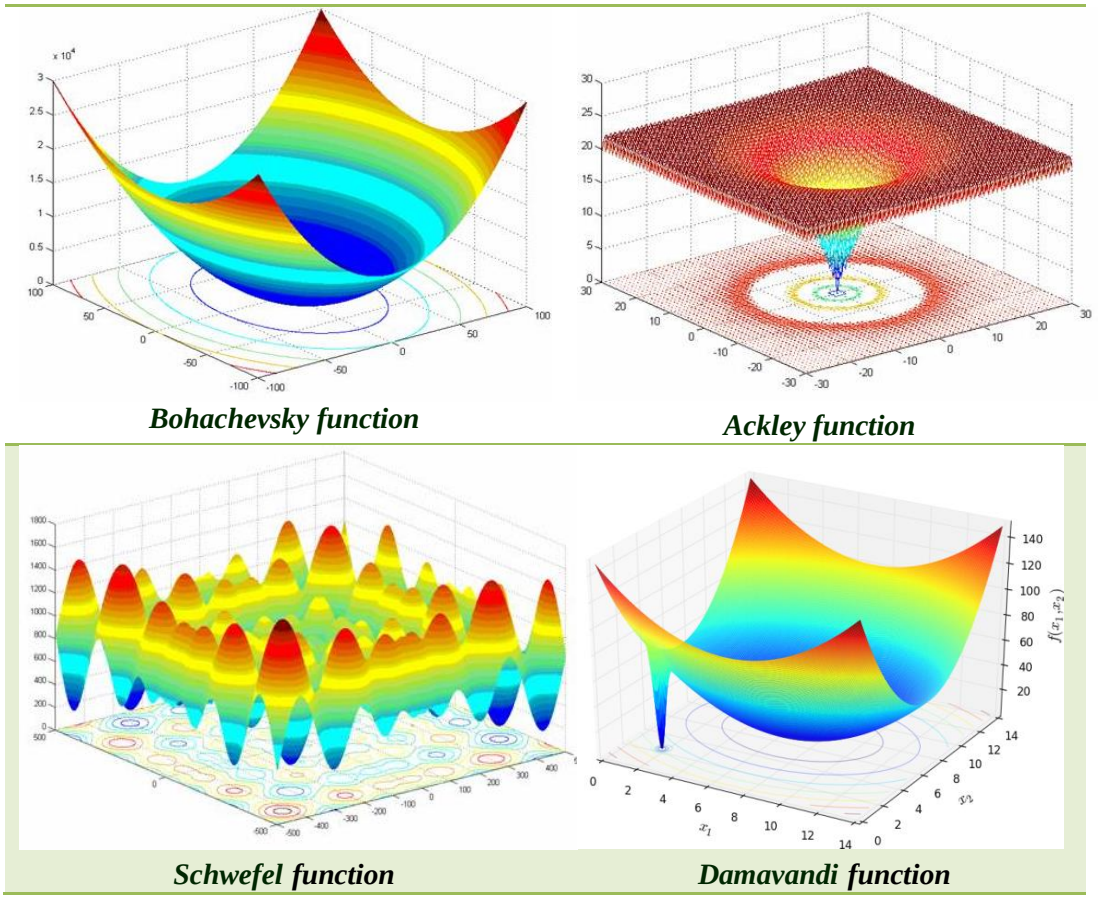


Figure 5: Examples of the graphs of some of our testing functions.

Table 1: The set of functions used for our experiments.

Function	Dimension	domain	Global Minima
Zakharov	1	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Schwefel	1	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$
Sum squares	1	[-10, 10]	$x^* = (0), f(x^*) = 0$
Beale	2	[-4.5, 4.5]	$x^* = (3, 0.5), f(x^*) = 0$
Zakharov	2	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Bohachecsky 1	2	[-100, 100]	$x^* = (0), f_1(x^*) = 0$

Booth	2	[-10, 10]	$x^* = (1, 3), f(x^*) = 0$
Easom	2	[-100, 100]	$x^* = (\pi), f(x^*) = -1$
Sum squares	2	[-10, 10]	$x^* = (0), f(x^*) = 0$
Matyas	2	[-10, 10]	$x^* = (0), f(x^*) = 0$
Goldstein & Price	2	[-2, 2]	$x^* = (0, 1), f(x^*) = 3$
Cross-in-Tray	2	[-10, 10]	$x^*=(1.3494,-1.3494), f(x^*)= -2.0626$
Drop Wave	2	[-5.12, 5.12]	$x^*(0), f(x^*)=-1$
Rastrigin	3	[-5.12, 5.12]	$x^* = (0,), f(x^*) = 0$
Zakharov	3	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Sum squares	3	[-10, 10]	$x^* = (0), f(x^*) = 0$
Schwefel	3	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$
Colville	4	[-10, 10]	$x^* = (1), f(x^*) = 0$
Rastrigin	4	[-5.12, 5.12]	$x^* = (0, 0, 0,0), f(x^*) = 0$
Zakharov	4	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Schwefel	4	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$
Rosenbrock	4	[-5, 10]	$x^* = (1), f(x^*) = o$
Rastrigin	5	[-5.12, 5.12]	$x^* = (0), f(x^*) = 0$
Zakharov	5	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Rosenbrock	5	[-5, 10]	$x^* = (0), f(x^*) = 0$
Schwefel	5	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$
Zakharov	6	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Rastrigin	6	[-5.12, 5.12]	$x^* = (0), f(x^*) = 0$
Rosenbrock	6	[-5, 10]	$x^* = (1), f(x^*) = o$
Schwefel	6	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$
Rastrigin	7	[-5.12, 5.12]	$x^* = (0), f(x^*) = 0$
Zakharov	7	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Rosenbrock	7	[-5, 10]	$x^* = (1), f(x^*) = o$
Schwefel	7	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$
Rastrigin	8	[-5.12, 5.12]	$x^* = (0), f(x^*) = 0$
Rosenbrock	8	[-5, 10]	$x^* = (1), f(x^*) = o$
Zakharov	8	[-5, 10]	$x^* = (o), Z_n(x^*) = o$
Schwefel	8	[-500, 500]	$x^* = (420.9687), f(x^*) = 0$

Table 2 shows the performance numbers in terms of effectiveness (how close the approximation to the global optimum) and the time efficiency. The time efficiency is shown in terms of the average time, the minimum time, and the maximum time. All the times are in milliseconds (*ms*) and rounded to the closest integer.

Referring to Table 2, the numbers show that our algorithm is effective in finding the global optima. It was able to find the exact global optimum for some functions and an excellent approximation of the global optimum for others with maximum error of 10^{-15}.

As the time numbers (in Table 2) show, there is an increase in the execution time as the number of variables (the dimension of each function) increases. We can attribute this time increase to the increase in the search space. Examining the minimum times in Table 2 reveals that sometimes the algorithm is able to find the optimum in much shorter time than the average.

Table 2: The performance numbers. The average time to find the global optimum, minimum and maximum times and the deviations from the actual global optimum.

Function Dimension	Effectiveness	Time Efficiency		
	Maximum deviation from the actual global optimum	Average Time (ms)	Min (ms)	Max (ms)

1	0	11	6	14
2	2E-23	12	6	16
3	6E-42	31	19	38
4	0	45	35	51
5	2E-46	58	49	62
6	1E-46	63	51	68
7	0	69	59	74
8	3E-29	77	66	80

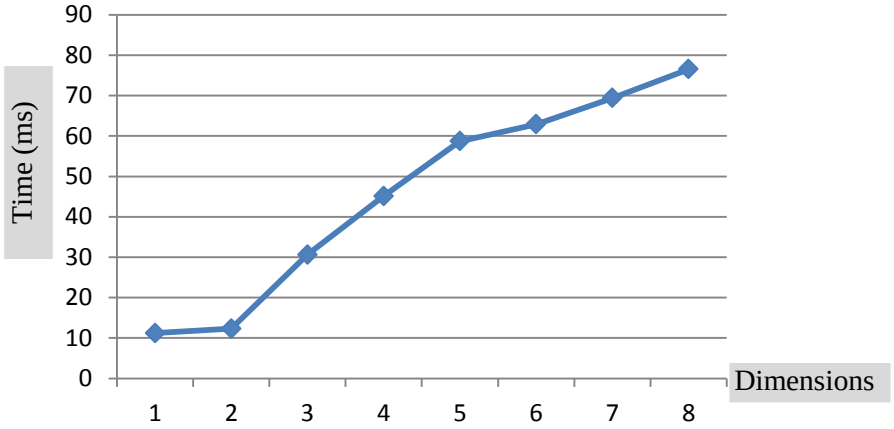


Figure 6: The time increase as a function of the number of variables.

Figure 6 visually graphs the time requirements as a function of the number of variables. Careful examination of the graph shows that there is an increase in time as the number of variables (dimensions) increases. This time increase is nevertheless polynomial. This outcome seems to be consistent with our previous analysis of the upper bound (Big-O) and with our observations drawn from the experiments.

5. Conclusions and Future Work

The paper showed a global efficient search algorithm that implements guided random search. The guided search is necessary to focus the search on the areas where the optimal value is expected and leaving opportunities for the other areas to be explored. Our algorithm is effective in finding global optima, time efficient, and easy to implement. It uses the probability triangle s to create a dynamic coverage for the search space and it is augmented with effective mapping techniques to guide the searching process. The linear mapping of random numbers into different random values constitutes how the random search is guided. Therefore the algorithm quickly directs the search to the parts of the domain that most likely contain the global optima.

The algorithm can dynamically adjust its triangle s' parameters and change the mapping mechanism using the knowledge gained from previous rounds. This dynamism grants our algorithm not only the superiority in quickly finding the global optimal, but also the effectiveness in quickly deriving the search to the promising regions (the regions in which the optima may reside with high probability). Furthermore, our algorithm has a unique feature. At any given time, the searching process covers the effective parts of the domains without ignoring the other parts (of the domains) that less likely contain the global optimal value. Therefore, it is highly unlikely that the algorithm misses the global optima.

The ability of the algorithm to shrink and expand its search based on guided random tools gives it a great chance to discover more hidden solutions within the multi-dimensional search space. It is not easy to fool this algorithm as evident from the simulations. Although the nature of the search depends a lot on the nature of the random number generator, our random

number generator has been tested to be unbiased mimics real world generated random numbers.

We have two directions for future work. We are extending our approach to tackle constrained optimization problems. We also want to include more effective and time efficient random generators and measure the enhancements in the performance. The algorithm may be developed to search for patterns and only single points. Optimum paths or projections that have many applications could the target of our new research.

References

[1]. Momin Jamil and Xin-She Yang, A Literature Survey of Benchmark Functions for Global Optimization Problems, International Journal of Mathematical Modelling and Numerical Optimization, Vol. 4, No. 2, pp. 150–194 (2013).

[2]. T. Bäck, H. P. Schwefel, An Overview of Evolutionary Algorithm for Parameter Optimization, Evolutionary Computation, vol. 1, no. 1, pp. 1-23, 1993.

[3]. M. Montaz Ali, C. Khompatraporn and Z. B. Zabinsky. A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems, Vol 31m No. 4, pp. 635-672, 2005.

[4]. H. Edwin Romeijn and Robert L. Smith, Simulated annealing for constrained global optimization, Journal of Global Optimization, Vol, 5, No. 2, pp 101–126, 1994.

[5]. S. Chapra and R. Canale, *Numerical Methods for Engineers*, 7th edition, McGraw-Hill Education, 2014.

[6]. S. Gerardo de-los-Cobos-Silva, M. Ángel Gutiérrez-Andrade, and et al, An Efficient Algorithm for Unconstrained Optimization, Journal of Mathematical Problems in Engineering, Vol. 2015, number of pages 17, 2015.

[7]. I. Muhammad, H. Rathiah, and A. K. Noor Elaiza, An Overview of Particle Swarm Optimization Variants, *Procedia Engineering*, vol. 53, pp. 491–496, 2013. View at Google Scholar

[8]. H. Jabeen, Z. Jalil and A.R. Baig, Opposition Based Initialization in Particle Swarm Optimization, in Proceedings of the 1 1th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, New York, NY, USA, pp. 2047-2052, 2009.

[9]. I. Fister, X. S Yang, J. Brest, Jr.I. Fister, On the Randomized Firefly Algorithm. Cuckoo Search and Firefly Algorithm. Springer International Publishing, pp. 27–48, 2014.

[10]. S.L. Tilahun and J.M.T. Ngnotchouye , Firefly Algorithm for Discrete Optimization Problems: A survey, KSCE Journal of Civil Engineering, Vol. 21, No. 2, pp 535–545, February 2017.

[11]. Zhang L, Liu L, Yang X-S, Dai Y, A Novel Hybrid Firefly Algorithm for Global Optimization. PLoS ONE 11(9), 2016. e0163230. doi:10.1371/journal.pone.0163230

[12]. Y. Zhao, W. Zu, and H. Zeng, A Modified Particle Swarm Optimization via Particle Visual Modeling Analysis, Computers & Mathematics with Applications, vol. 57, no. 11-12, pp. 2022–2029, 2009

[13]. F. Glover, Tabu Search, part I, ORSA Journal on Computing, vol. 1, no. 3, pp. 190–206, 1989.

[14]. D.J. Reid, Genetic algorithms in constrained optimization, Vol. 23, no. 5, pp. 87-111, 1996.

[15]. Q. Long, C. Wu, T. Huang, and X. Wang, A genetic algorithm for unconstrained multi-objective optimization, Swarm and Evolutionary Computation, vol. 22, pp. 1-14, 2015.

[16]. J. Y.Wu, Solving unconstrained global optimization problems via hybrid swarm intelligence approaches, Mathematical Problems in Engineering, vol. 2013, 15 pages, 2013.

[17]. F.Glover, M. , Laguna., R. Marti, Fundamentals of Scatter Search and Path Relinking, Control and Cybernetics, Vol. 29, no. 3, pp. 653-684, 2000.

[18]. Y. Zhang, S. Wang, and G. Ji, A Comprehensive Survey on Particle Swarm Optimization Algorithm and Its Applications, Mathematical Problems in Engineering Volume 2015, 2015.

[19]. A. Gavana, Global Optimization Benchmarks, 2017, http://infinity77.net.

[20]. Y. Zheng and Z. Wan, A new variant of the memory gradient method for unconstrained optimization, Optimization Letters, Vol. 6, no. 8, pp 1643–1655, 2012.