Performance of Migrating Birds Optimization Algorithm on Continuous Functions

Ali Fuat Alkaya[1], Ramazan Algin[1], Yusuf Sahin[2], Mustafa Agaoglu[1], and Vural Aksakalli[3]

[1] Marmara University, Department of Computer Engineering, Istanbul, Turkey
[2] Marmara University, Department of Electrical and Electronics Engineering, Istanbul, Turkey
[3] Istanbul Sehir University, Department of Industrial Engineering, Istanbul, Turkey
{falkaya,ysahin,agaoglu}@marmara.edu.tr,
algin.ramazan@gmail.com, aksakalli@sehir.edu.tr

Abstract. In this study, we evaluate the performance of a recently proposed metaheuristic on several well-known functions. The objective of this evaluation is to participate in a competition where several metaheuristics are compared. The metaheuristic we exploit is the recently proposed migrating birds optimization (MBO) algorithm. Our contribution in this study is to develop a novel neighbor generating function for MBO to be used in multidimensional continuous spaces. After a set of preliminary tests presenting the best performing values of the parameters, the results of computational experiments are given in 2, 10 and 30 dimensions.

Keywords: migrating birds optimization, continuous functions, single objective optimization.

1 Introduction

The MBO algorithm is a newly proposed, population-based neighborhood search technique inspired from the V formation flight of the migrating birds which is proven to be an effective formation in energy minimization. In the analogy, initial solutions correspond to a flock of birds. Likewise the leader bird in the flock, a leader solution is chosen and the rest of the solutions is divided into two parts. Each solution generates a number of neighbor solutions. This number is a determiner value on exploration and it corresponds to the speed of the flock. The higher this value, the more detailed the flock explores its surroundings.

The algorithm starts with a number of initial solutions corresponding to birds in a V formation. Starting with the first solution (corresponding to the leader bird) and progressing on the lines towards the tales, each solution is tried to be improved by its neighbor solutions. If any of the neighbor solutions is better, the current solution is replaced by that one. There is also a benefit mechanism for the solutions (birds) from the solutions in front of them. Here we define the benefit mechanism as sharing the best unused neighbors with the solutions that follow. In other words, a solution evaluates a number of its own neighbors and a number of best neighbors of the previous solution and is replaced by the best of them. Once all solutions are improved (or tried to be improved) by neighbor solutions, this procedure is repeated a number of times (tours) after which the first solution becomes the last, and one of the second solutions becomes the first and another loop starts. The algorithm is terminated after a predetermined number of neighbors are generated. Pseudocode of our MBO is given in Figure 1.

```
1. Generate $n$ initial solutions in a random manner and place them on an hypothetical
   $V$ formation arbitrarily
2. while termination condition is not satisfied
3.        for $m$ times
4.              Try to improve the leading solution by generating and evaluating $k$
                neighbors of it
5.              for each solution $s_i$ in the flock (except leader)
6.                    Try to improve $s_i$ by evaluating ($k$-$x$) neighbors of it and $x$ unused
                      best neighbors from the solution in the front
7.              endfor
8.        endfor
9.        Move the leader solution to the end and forward one of the solutions
          following it to the leader position
10. endwhile 11. return the best solution in the flock
```

Fig. 1. Pseudocode of the MBO

MBO algorithm has four parameters: number of solutions (n), number of tours (m), number of neighbor solutions to be generated from a solution (k) and number of solutions to be shared with the following solution (x). However, due to the inherent design of the algorithm n value has to be equal to or greater than $2 * x + 1$.

This new metaheuristic was proposed by Duman et al. [1]. They applied it to solve quadratic assignment problem instances arising from printed circuit board assembly workshops. Its performance was compared with those of metaheuristics implemented and compared in two previous studies. These metaheuristics are simulated annealing, tabu search, genetic algorithm, scatter search, particle swarm optimization, differential evolution and guided evolutionary simulated annealing. In this comparison, the MBO outperformed the best performed metaheuristic (simulated annealing) in the previous studies by approximately three percent on the average. In addition, MBO was tested with some benchmark problem instances obtained from QAPLIB and in most of the instances it obtained the best known solutions. As a result of these tests, it is concluded that the MBO is a promising metaheuristic and it is a candidate to become one of the highly competitive metaheuristics. Duman and Elikucuk [2] applied MBO to solve fraud detection problem. They also proposed a new version of MBO where a different benefit mechanism is used. They tested the original MBO algorithm and its new version on real data and compared their performance with that of genetic algorithm hybridized with scatter search (GASS). Test results showed that the MBO algorithm and its new version performed significantly better than the GASS algorithm.

In this study, we exploit MBO to solve problems in continuous environments. The set of functions used are given in [3] which are tried to be minimized on 2, 10 and 30 dimensional spaces. The search space is $[-100, 100]^D$ where D is the dimension. We believe that defining an effective neighboring function is much more important than any other modifications on the MBO. In line with this observation, our contribution in this study is to develop a novel neighbor generating function for MBO to be used in multidimensional continuous spaces.

In the next section we present an effective neighbor generating function designed for MBO. Section three presents experimental setup which includes parameter analysis of the MBO algorithm. Section four gives the results where MBO is run on 30 different functions and various dimensions. Section five gives the conclusive remarks together with some future work.

2 A Novel Neighbor Generating Function

In order to design a well performing MBO algorithm, an effective neighbor generating function is essential. To have a more effective exploration plan in the D dimensional solution space, we used D dimensional spheres (D-spheres for short throughout this paper). A neighbor of a solution can be obtained only within the D-sphere around it. A neighbor of a solution can be at most r units away from the original solution where r is the radius of the D-sphere that surrounds it. To find the radius of a D-sphere, we firstly calculate the volume allocated to it using the following formula.

$$V_D = TV/n \tag{1}$$

where V_D is the volume of a D-sphere and TV is the total volume of the solution space. In order to calculate the radii for the D-spheres in a D dimensional space, the volume of the solution space is divided by n. In this way, we try to make an effective exploration and fair distribution of volume for all birds (solutions) to fly around. When the volume of a D-sphere is calculated, we need to find the radius of the D-sphere. The following inductive formulas give the volumes of D-spheres.

$$V_1 = 2 * r \tag{2}$$

$$V_2 = \pi * r^2 \tag{3}$$

$$V_D = V_{D-2} * 2 * \pi / r^D forD > 2 \tag{4}$$

Once the volume for each sphere is calculated, the radii of each sphere can be easily calculated using Equations(2-4). After calculating the radius of D-sphere, we can develop a method to find a neighbor solution (point) within the sphere using some trigonometry. The distance that how far will the new solution be away from the original solution will be a random number in $[0, r]$ where r is the radius of the sphere.

Additionally, we also need to determine the location (coordinate in each axis) of the point in the D dimensional space. For this, we used the following set of trigonometric formula.

$x_D = l * cos(\theta_{D-1})$
$x_{D-1} = l * sin(\theta_{D-1}) * cos(\theta_{D-2})$
$x_{D-2} = l * sin(\theta_{D-1}) * sin(\theta_{D-2}) * cos(\theta_{D-3})$
...
$x_2 = l * sin(\theta_{D-1}) * sin(\theta_{D-2}) * ... * sin(\theta_2) * cos(\theta_1)$
$x_1 = l * sin(\theta_{D-1}) * sin(\theta_{D-2}) * ... * sin(\theta_2) * sin(\theta_1)$

where l is the distance that how far will the new solution be away from the original solution, x_i is the coordinate of the point in the i^{th} axis and θ_i is the angle between i^{th} and $(i+1)^{th}$ axis. Before using this set of formula θ_i's must be obtained randomly such that $\theta_1 \in [0, 2\pi]$ and $\theta_i \in [0, \pi]$ for $i = 2, \ldots, D-1$. An example for the formulas given above is presented in Figure 2 for $D = 3$.

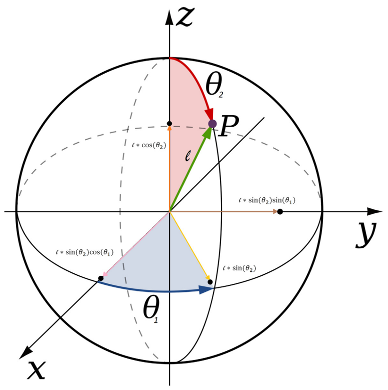

Fig. 2. Representation of a point (solution) and the vectors constituting it in three dimensions

From this setting, one can easily observe that if the number of birds (solutions) is small, then the volume that they are going to explore will be large whereas if the number of birds is large, the volume that they are going to explore will be small. Since we are limited by the number of function evaluation (neighbor generations) due to the competition rules, with a large number of solutions we will be able to explore small number of neighbors in smaller regions whereas with small number of solutions we will be able to explore large number of neighbors in larger regions. Hence, an efficient value for the n parameter must be found for the best performance of the algorithm.

3 Experimental Setup

The experiments are run on an HP Z820 workstation with Intel Xeon E5 processor at 3.0 GHz with 128 GB RAM running Windows 7. The MBO algorithm is implemented in Java language. The stopping criterion for the MBO algorithm is a given number of function evaluations which correspond to number of neighbors generated. Specifically the allowed number of function evaluations is 10000*D.

Table 1. Statistics of 51 runs on 30 different functions when D=2

Function ID	*min*	*max*	*avg*	*med*	*std*
1	40.82	19721.55	1951.86	1113.82	2964.70
2	2.48	4532.26	408.61	98.97	751.68
3	1.67	1.67	1.67	1.67	0.00
4	15.22	4166.54	488.89	227.25	704.77
5	0.17	1.20	0.57	0.59	0.24
6	0.03	3.58	0.95	0.82	0.80
7	0.00	0.01	0.00	0.00	0.00
8	0.91	6.44	3.99	4.17	1.17
9	0.07	0.43	0.26	0.26	0.10
10	0.02	0.39	0.20	0.21	0.10
11	0.01	1.10	0.35	0.23	0.36
12	0.08	1.28	0.55	0.55	0.30
13	0.02	0.08	0.02	0.02	0.01
14	0.23	1.21	0.73	0.75	0.21
15	0.28	2.62	0.98	0.81	0.54
16	0.01	0.15	0.07	0.06	0.04
17	0.01	0.82	0.16	0.12	0.15
18	-861.55	0.00	-840.06	-837.83	5.82
19	0.01	0.28	0.09	0.09	0.06
20	0.39	7.27	2.85	2.97	1.45
21	0.02	1.38	0.51	0.47	0.32
22	0.03	1.44	0.34	0.21	0.34
23	0.32	10.86	7.77	8.34	2.01
24	17.73	7069.62	551.74	261.12	1056.58
25	2.13	2390.53	355.57	138.30	535.51
26	1.33	12.67	2.75	2.03	1.92
27	-41721.46	0.00	-38311.70	-38587.87	2274.27
28	-1.57	0.00	-0.88	-0.87	0.26
29	8.47	17.97	12.75	12.35	2.77
30	0.05	0.84	0.33	0.33	0.16

In order to reveal the best performing parameter values of the MBO on the continuous functions, we run a set of extensive computational experiments. These preliminary tests are conducted on six functions selected out of 30 given in [3]. Best performing values for the parameters are as follows: $n = 5001, k = 3, m = 1, x = 1$.

4 Results

In this section we provide the results of the MBO algorithm on the aforementioned continuous benchmark functions. One of the results to be delivered as a rule of the competition is the $T1$ and $T2$ values. $T1$ is the average run time of five runs of the following piece of MATLAB code in our environment.

Table 2. Statistics of 51 runs on 30 different functions when D=10

Function ID	*min*	*max*	*avg*	*med*	*std*
1	20323203.08	109661239.94	61390076.47	63430416.62	18862528.62
2	240.72	1998.77	801.74	755.39	358.08
3	192.16	304.70	247.67	248.40	21.37
4	354.30	2204.80	1001.13	934.47	468.90
5	1.09	1.64	1.35	1.36	0.12
6	477.57	5023.73	1975.59	1786.44	1048.17
7	0.17	1.53	0.84	0.83	0.25
8	7.04	11.55	9.73	9.74	1.01
9	3.22	5.43	4.75	4.91	0.51
10	1.25	2.12	1.72	1.75	0.22
11	11.01	35.33	27.20	28.07	5.30
12	0.66	2.42	1.54	1.60	0.37
13	1.54	2.58	2.13	2.13	0.25
14	5.30	13.38	8.59	8.56	2.05
15	33.46	214.90	120.04	122.06	42.16
16	1.69	3.46	2.71	2.75	0.43
17	578.73	4028.84	1831.87	1830.40	681.23
18	-3470.64	0.00	-2717.47	-2617.14	376.96
19	1.97	8.05	4.56	4.36	1.30
20	5.90	13.83	9.96	9.91	1.73
21	5.10	19.54	13.41	13.80	3.44
22	38.72	213.23	122.00	126.23	38.49
23	29.19	43.87	38.97	39.58	2.96
24	395.74	1997.54	980.50	935.94	390.40
25	217.89	3921.61	1169.83	1048.91	596.34
26	838.83	6476.15	2421.21	2310.05	1154.72
27	-19655.46	0.00	-13727.18	-13385.63	2203.31
28	14.54	15.72	15.14	15.17	0.29
29	21.37	21.38	21.37	21.37	0.00
30	1.08	1.35	1.22	1.23	0.07

Table 3. Statistics of 51 runs on 30 different functions when D=30

Function ID	*min*	*max*	*avg*	*med*	*std*
1	131323505.00	371642428.18	272954269.37	278704299.47	49176717.62
2	370.11	1232.52	730.82	729.31	191.71
3	19702.97	36266.85	28845.16	29096.35	4050.11
4	397.31	2204.13	920.60	891.43	351.70
5	1.04	1.23	1.10	1.10	0.04
6	1607.06	12377.82	7458.20	7420.84	2176.52
7	1.80	3.42	2.80	2.80	0.39
8	10.20	11.89	11.05	11.11	0.46
9	13.70	17.27	15.64	15.67	0.88
10	2.97	4.82	3.96	4.06	0.42
11	81.94	140.23	114.50	117.65	13.35
12	2.77	4.47	3.74	3.74	0.39
13	8.00	10.33	9.18	9.27	0.58
14	6.30	11.09	8.99	9.18	1.14
15	194.15	371.56	298.56	311.88	42.88
16	5.71	8.75	7.47	7.60	0.72
17	35345.91	77582.40	54386.92	55674.38	10061.42
18	-5215.18	0.00	-4565.75	-4527.92	252.22
19	8.43	17.05	11.84	11.86	1.89
20	5.71	9.16	7.86	8.05	0.70
21	20.15	38.43	30.42	30.40	4.19
22	189.36	418.69	316.94	322.05	59.04
23	100.05	116.60	108.54	108.75	3.99
24	603.02	1419.04	951.32	938.95	199.66
25	11240.74	29044.32	19293.43	19268.09	4740.83
26	1980.60	11722.33	7516.65	8055.23	2296.92
27	-6756.60	0.00	-5080.52	-5009.40	753.36
28	54.70	55.72	55.18	55.18	0.28
29	21.61	21.62	21.61	21.61	0.00
30	1.38	1.47	1.43	1.44	0.02

```
for i=1:300000
    evaluate(9,rand(30,1)*200-100);
end
```

$T2$ is the average run time of five runs of the function 9 on D=30 in our environment.

According to our experimental work $T1$, $T2$ and $(T2\text{-}T1)/T1$ are as follows:

- $T1$=29.965
- $T2$=73.369
- $(T2\text{-}T1)/T1$=1.448

Table 1, 2 and 3 present the statistics when D=2, 10 and 30, respectively. The major observation among the tables is that in higher dimensions the performance of the MBO algorithm gets worse. This is an expected result because the search space grows much faster than the allowed number of function evaluations.

5 Conclusion

In this study we applied migrating birds optimization algorithm to 30 different functions on continuous domain. Our contribution in this study is to develop an effective novel neighbor generation function for MBO. The tests are conducted on 2, 10 and 30 dimensions. Results present that even though MBO is a recently proposed algorithm it is also promising for problems in continuous domain.

References

1. Duman, E., Uysal, M., Alkaya, A.F.: Migrating Birds Optimization: A New Metaheuristic Approach and Its Performance on Quadratic Assignment Problem. Information Sciences 217, 65–77 (2012)
2. Duman, E., Elikucuk, I.: Solving Credit Card Fraud Detection Problem by the New Metaheuristics Migrating Birds Optimization. In: Rojas, I., Joya, G., Cabestany, J. (eds.) IWANN 2013, Part II. LNCS, vol. 7903, pp. 62–71. Springer, Heidelberg (2013)
3. Website of Fifth International Conference on Swarm Intelligence, `http://www.ic-si.org/competition`