24

Whale Optimization Algorithm

Ali R. Kashani
Department of Civil Engineering
University of Memphis, Memphis, United States

Charles V. Camp
Department of Civil Engineering
University of Memphis, Memphis, United States

Moein Armanfar
Department of Civil Engineering
Arak University, Arak, Iran

Adam Slowik
Department of Electronics and Computer Science
Koszalin University of Technology, Koszalin, Poland

CONTENTS

24.1 Introduction 323
24.2 Original WOA 324
  24.2.1 Pseudo-code of the WOA 324
  24.2.2 Description of the WOA 325
24.3 Source-code of the WOA in Matlab 326
24.4 Source-code of the WOA in C++ 327
24.5 A step-by-step numerical example of WOA 329
24.6 Conclusions 332
References 333

24.1 Introduction

In 2016, Mirjalili and Lewis [1] developed a swarm-based optimization algorithm called the whale optimization algorithm (WOA). This algorithm numerically models the social behavior and hunting strategies of humpback whales. A WOA explores the solution search space using a bubble-net feeding method. Humpback whales tend to entrap school of krill or small fishes through a circular or '9'-shaped bubble-made cage. To accomplish this task whales use two different tactics: upward-spirals or double-loops. More detail about those tactics can be found in [1, 2]. In general, a WOA uses three main steps: encircling prey; bubble-net attacking method; and search for prey. The encircling the prey method leads all the search agents towards the best-found solution (leader). Next the bubble-net attacking phase (exploitation) simulates the path which whales use to get close to their prey. Based on this strategy, whales move on both circular and spiral-shape paths simultaneously. Finally, in the searching for prey (exploration) phase, randomly selected agents modify the position of the i-th search agent. Successful applications of the WOA for different engineering problems have attracted many in the research community [3-5]. From this research, several variations and improvements on the WOA have been developed such as: a Lévy flight trajectory-based WOA which prevents premature convergence and helps to avoid local optimum solutions [6], an adaptive autoregressive WOA (used for handling a traffic-aware routing in VANET) [7], an improved WOA that uses a dynamic strategy for updating control parameters and applies a quadratic interpolation to the leader that enhanced its ability to handle large scale optimization problem [8], a chaos WOA (applies chaos theory tried to optimize the Elman neural network) [9], and a multi-objective WOA that considers a multi-level threshold for image segmentation [10]. The remainder of this chapter is organized accordingly. In Section 24.2 a step-by-step pseudo-code and the description for the original WOA algorithm are provided, in Sections 24.3 and 24.4 source-codes of the WOA algorithm in both Matlab and C++ are presented, respectively, in Section 24.5 a detailed numerical example is solved by WOA, and finally a conclusion is presented in Section 24.6.

24.2 Original WOA

24.2.1 Pseudo-code of the WOA

Algorithm 23 presents the pseudo-code for the global version of the WOA.

Algorithm 23 Pseudo-code of the original WOA.

1: define the $D-th$ dimensional objective function $OF(.)$
2: define the range of variability for each $j-th$ dimension $\left[X_{i,j}^{min}, X_{i,j}^{max}\right]$
3: determine the WOA algorithm parameter values such as $SearchAgents_no$ – the number of search agents, MI – maximum iteration
4: randomly create positions X_i for $SearchAgents\_no$ number of search agents (each agent is a D-dimensional vector)
5: find the best search agent and call it leader
6: $Iter = 0$
7: **while** termination condition not met (here is reaching MI) **do**

8: **for** each i-th search agent **do**
9: update α value to decrease from 2 to 0 using formula:
10: $\alpha = 2 - Iter \times \left(\frac{2}{MI}\right)$
11: $A = 2\alpha \times rand - \alpha$
12: $C = 2 \times rand$
13: determine p as a random number between 0 and 1
14: **if** p<0.5 **then**
15: **if** $\mid A \mid < 1$ **then**
16: update the position of i-th agent using:
17: $X\left(Iter\right) = X^{*}\left(Iter - 1\right) - A \cdot \mid C \cdot X^{*}\left(Iter - 1\right) - X\left(Iter - 1\right) \mid$
18: **else**
19: randomly select one of the search agents as a leader
20: update the position of the i-th agent using:
21: $X\left(Iter\right) = X_{rand} - A \cdot \mid C \cdot X_{rand} - X\left(Iter - 1\right) \mid$
22: **end if**
23: **else**
24: update the position of i-th agent using:
25: $X\left(Iter\right) = D^{'} \cdot exp\left(bl\right) \cdot \cos\left(2\pi l\right) + X^{*}\left(Iter - 1\right)$
26: **end if**
27: **end for**
28: calculate $OF(.)$
29: update best-found solution
30: **end while**
31: post-processing the results

24.2.2 Description of the WOA

A step-by-step procedure describing the fundamentals of the WOA is listed in Algorithm 23. In the first step, the optimization problem is defined in the form of a D-dimensional objective function $OF(.)$. Next, boundary constraints are defined for the i-th agent of the j-th variable as $\left[X_{i,j}^{min}, X_{i,j}^{max}\right]$ where $j = \{1, ..., numberofdecisionvariables\}$. The necessary parameters of WOA algorithm have been set in the next step such as: number of search agents (SA_no) and maximum number of iterations (MI). In the fourth step, a randomly generated population of SA_{no} search agent has been initialized. A leader has been selected considering two different attitudes: the best-found solution to provide exploitation, and a randomly selected solution to guarantee the exploration. Then, the search agents explore the solution space around the leader agent. In step 5, the first attitude, using the best-found solution as a leader, has been tackled. Then, the current iteration counter is reset to zero to start the main loop of WOA algorithm. For each i-th search agent an iterative loop updates its position. In step 10, the value of α has been updated using a reducing trend. Next, two coefficient vectors, A and C, are updated in steps 11 and 12, respectively. Humpback whales get close to their prey following a shrinking circle (step 16 and 19) and spiral-shaped path (step 22), simultaneously. WOA will select one of those paths using a probability of 50%. In this way, a randomly generated p with value of lower than 50% leads the algorithm toward using a shrinking circle (as shown in step 14). Now, we check the absolute value of previously determined A in step 15. If this condition is met with values of less than a unit, the best found solution will be utilized as a leader to update the position via a shrinking circle. Otherwise, an agent is selected randomly and the position of the i-th agent is updated in steps 18 and 19, respectively. In step 22, the position of the i-th agent is updated following a spiral-shaped path in case the probability p is more than or equal to 50%. In step 25, we estimate the $OF(.)$ for all the search agents. Finally, the global best solution is updated. Like any other algorithm the best found solution and its relevant vector are proposed as the final result when the termination criteria are satisfied.

24.3 Source-code of the WOA in Matlab

Listing 24.1 shows the source-code for the objective function defined in the WOA. In the objective function $OF(.)$, the input parameters are all search agents (SA). The result of $OF(.)$ function is an N-dimensional column vector with the objective function values for each search agent SA_i for the whole SA population. Equation (24.1) gives the objective function for this example. Listing 24.2 shows the Matlab source-code for the WOA.

$$OF\left(SA_i\right) = \sum_{j=1}^{D} SA_{i,j}^2; where - 5.12 < SA_{i,j} < 5.12 \tag{24.1}$$

```
function [output]=OF(SA)
[x,y]=size(SA);
output=zeros(x,1);
for i=1:x
    for j=1:y
    output(i,1)=output(i,1)+SA(i,j)^2;
    end
end
```

Listing 24.1
Definition of objective function $OF(.)$ in Matlab.

```
% parameter setting for WOA algorithm
SearchAgents_no=6; Max_iter=1; dim=5;
% Bound constraint definition
lb=-5.12*ones(1,dim); ub=5.12*ones(1,dim);
% initialize position vector and score for the leader
Leader_pos=zeros(1,dim); Leader_score=inf;
%Initialize the positions of search agents
```

```
for i=1:dim
    Positions(:,i)=rand(SearchAgents_no,1).*(ub(i)-lb(i))+lb(i);
end
t=0;% Loop counter
% Main loop
while t<Max_iter
    for i=1:size(Positions,1)
% Return back the search agents that go beyond the boundaries of the search space
        Flag4ub=Positions(i,:)>ub; Flag4lb=Positions(i,:)<lb; Positions(i,:)=(Positions(i,:).*(~(Flag4ub+Flag4lb)))+ub.*Flag4ub+lb.*Flag4lb;
% Calculate objective function for each search agent
        fitness=fobj(Positions(i,:));
% Update the leader
        if fitness<Leader_score
            Leader_score=fitness;
            Leader_pos=Positions(i,:);
        end
    end
% a decreases linearly from 2 to 0 (Step 10 in Algorithm 1)
    a=2-t*((2)/Max_iter);
% a2 linearly decreases from -1 to - b2 to calculate l in step 22
    a2=-1+t*((-1)/Max_iter);
% Update the Position of search agents
    for i=1:size(Positions,1)
% Step 11 in Algorithm 1
        A=2*a*rand()-a;
% Step 12 in Algorithm 1
        C=2*rand();
        b=1;            % parameter used in step 22 in Algorithm 1
        l=(a2-1)*rand+1;% parameter used in step 22 in Algorithm 1
        p = rand();     % define p as a random number between 0 and 1 (step 13 in Algorithm 1)
        for j=1:size(Positions,2)
% Shrinking encircling mechanism
            if p<0.5
                if abs(A)>=1
% select one of the search agents as a leader randomly
                    rand_leader_index=floor(SearchAgents_no*rand()+1);
                    X_rand=Positions(rand_leader_index,:);
% Update i-th agent based on step 19 in Algorithm 1
                    D_X_rand=abs(C*X_rand(j)-Positions(i,j));
                    Positions(i,j)=X_rand(j)-A*D_X_rand;
                elseif abs(A)<1
% Update i-th agent based on the best solution as a leader (step 16 in Algorithm 1)
                    D_Leader=abs(C*Leader_pos(j)-Positions(i,j));
                    Positions(i,j)=Leader_pos(j)-A*D_Leader;
                end
            elseif p>=0.5
% Spiral updating position(step 22 in Algorithm 1)
                distance2Leader=abs(Leader_pos(j)-Positions(i,j)); % distance between the whale and prey
                Positions(i,j)=distance2Leader*exp(b.*l).*cos(l.*2*pi)+Leader_pos(j);
            end
        end
    end
    t=t+1;
    Convergence_curve(t)=Leader_score;
    [t Leader_score];
end
```

Listing 24.2
Source-code of the WOA in Matlab.

24.4 Source-code of the WOA in C++

The C++ source-code for the objective function and the WOA are presented in Listing 24.3 and Listing 24.4, respectively.

```
#include <iostream>
using namespace std;
/* Function Definitions */
double OF(double SA[], int size_array)
{
  double output;
  int j;
  output = 0.0;
  for (j = 0; j < size_array; j++) {
    output += SA[j] * SA[j];
  }
  return output;
}
```

Listing 24.3
Definition of objective function $OF(.)$ in C++.

```
/* Include files */
#include <iostream>
#include <stdlib.h>
#include <math.h>
using namespace std;
/* Function Definitions */
WOA(int SearchAgents_no, int Max_iter, int dim, double lb[], double ub
    [])
{
double leader_pos[dim]; double leader_score; double fitness;
double Positions[SearchAgents_no][dim]; double fitness[1][
    SearchAgents_no]; double convergence[1][SearchAgents_no];
for (int i = 0; i < SearchAgents_no; i++) {
  leader_pos[i] = { 0 }; leader_score = {1.79769e+308 };   /* change
    this to -inf for maximization problems */
  srand(time(0));
  for (int j = 0; j < dim; j++) {
    double  r = (rand() % 10000) / 10000;
    Positions[i][j] = (ub[j] - lb[j])*r + lb[j];
  }
}
double t = 0; /*  Loop counter */
double Convergence_curve[1][Max_iter];
 /*  Main loop */
  while (t < Max_iter) {
    for (int i = 0; i < SearchAgents_no; i++) {
      /*  Return back the search agents that go beyond the boundaries
      of the search space */
        for (int j = 0; j < dim; j++) {
          if (Positions[i][j] > ub[j]) Positions[i][j] = ub[j];
          if (Positions[i][j] < lb[j]) Positions[i][j] = lb[j];
        }
        fitness = OF(Positions[i], dim);
        if (fitness < leader_score) {
          leader_score = fitness;
          for (int k = 0; k < dim; k++) leader_pos[k] = fitness[k];
        }

    }
    a = 2.0 - t*((2.0) / Max_iter);   /* a decreases linearly fron 2 to
     0 */
```

```
    a2 = -1.0 + t*((-1.0) / Max_iter); /* a2 linearly decreases from
    -1 to -2 */
    /*  Update the Position of search agents  */
    for (i = 0; i < (int)SearchAgents_no; i++) {
      srand(time(0));
      double r1 = (rand() % 10000) / 10000;
      double r2 = (rand() % 10000) / 10000;
      double A = 2.0 * a * r1 - a;
      double C = 2.0 * r2;
      double l = (a2 - 1.0) * ((rand() % 10000) / 10000) + 1.0;
      double p = (rand() % 10000) / 10000;
      for (int j = 0; j < dim; j++) {
        if (p < 0.5) {
          if (abs(A) >= 1) {
            int rand_leader_index = floor(searchAgents_no*((rand() %
    10000) / 10000) + 1);
            for (int k = 0; K < dim; k++) double X_rand = Positions[
    rand_leader_index][k];
            double D_X_rand = abs(C*X_rand[j] - Positions[i][j]);
            Positions[i][j] = X_rand[j] - A*D_X_rand;
          }
          else if (abs(A) < 1) {
            double D_leader = abs(C*leader_pos[j] - Positions[i][j]);
            Positions[i][j] = leader_pos[j] - A*D_leader;
          }
        }
        else if (p >= 0.5) {
          double distance2LEader = abs(leader_pos[j] - Positions[i][j
    ]);
          Positions[i][j] = distance2leader*exp(b*l)*cos(2 * l* M_PI)
    + leader_pos[j];
        }
      }
    }
    t++;
    Convergence_curve[1][t] = leader_score;}
```

Listing 24.4
Source-code of the WOA in C++.

24.5 A step-by-step numerical example of WOA

In this section, a detailed computational process is provided for the WOA using the objective function defined in Equation 24.1. In this optimization example, there are 5 design variables and 6 search agents. In the first step, the position of the leader is set to be zero and its objective value is set to an initial value (infinity for a minimization problem). The leader search agent is updated by the best solution found in the population. In the second step, an initial population is developed by randomly generating values for the design variables within the permitted solution domain. In this example, the initial population is:

$SA_1 = \{-0.0597, 2.0352, 4.1444, -3.2469, 4.9017\}$
$SA_2 = \{2.8575, -3.0944, 1.1250, -2.6631, 2.1780\}$

$SA_3 = \{2.2020, -4.8073, 1.2049, 3.9579, 0.0048\}$
$SA_4 = \{4.1341, 2.4993, 3.6807, -4.8264, -0.2961\}$
$SA_5 = \{4.0030, 0.0002, 3.1282, -0.1034, -4.5095\}$
$SA_6 = \{-1.6982, -0.2056, 0.7856, -3.4004, 1.8634\}$

In the third step, the main loop of the algorithm starts and continues until termination criteria are satisfied. Within this loop, the objective values for the population of search agents are evaluated as:

$OF(SA_1) = 55.8899$
$OF(SA_2) = 30.8422$
$OF(SA_3) = 45.0751$
$OF(SA_4) = 60.2664$
$OF(SA_5) = 46.1564$
$OF(SA_6) = 18.5784$

In the next step, the position and objective function value of the leader search agent is updated based on the best-found solution. In this example, the new leader is SA_6:

$Leader\_position = \{-1.6982, -0.2056, 0.7856, -3.4004, 1.8634\}$
$Leader\_score = 18.5784$

Now, values of α (step 10 in Algorithm 23) and α_2 (for calculating l in line 36 of Listing 24.4) are updated on a linearly decreasing pattern from 2 to 0 and from -1 to -2, respectively.

$\alpha = 2$ (line 25 in Listing 24.4); α_2=-1 (line 28 in Listing 24.4).

In the next step, three possible movements can be considered for all the search agents via the $'for'$ loop (line 38 in Listing 24.4). Within this loop, in each iteration two different random numbers are produced between 0 and 1 to calculate A and C as follows:

$r_1 = 0.0424 \to A = 2\alpha \cdot r_1 - \alpha$
$A = \{-1.8303, 0.0744, -0.1848, -1.4673, 1.2435, -0.3328\}$

$r_1 = 0.0714 \to A = 2 \cdot r_2$
$C = \{0.1429, 1.9459, 0.8648, 0.3468, 0.1209, 1.3137\}$

Next, determine the spiral updating position l, a random number in $[-1, 1]$.

$l = (\alpha_2 - 1) \times rand + 1$
$l = \{-0.0433, -0.2980, -0.6506, 0.2181, 0.2015, -0.2559\}$

An important fact about humpback whale behaviors is the way of they approach their prey either by moving in a shrinking circle or spiral path. The WOA decides between these motions using a threshold probability of 50%. To this end, a random number p is generated between 0 and 1 (step 13 in Algorithm 23).

$p = \{0.0967, 0.8003, 0.0835, 0.8314, 0.5269, 0.2920\}$

In this example, it can be seen that for the first, third, and sixth search agents the shrinking encircling mechanism is used and the others are updated by the spiral updating position method. For the first search agent, the p value is less than 0.5 and $|A| > 1$. Hence, the shrinking circle method based on the randomly selected leader is used to update its position. To this end, $rand\_leader\_index$ in line 43 of Listing 24.4 proposes the 5-th search agent as a leader. In this case, the position of the search agent is updated using:

$X\,(t+1) = X_{leader} - A \cdot D$ where $D = \mid C \cdot X_{leader} - X\,(t) \mid$

In this example, these values are:

$D = \{0.6317, 2.0351, 3.6974, 2.7829, 4.9440\}$
$SA_1 = \{5.1592, 3.7251, 9.8954, 1.8466, 8.7528\}$

For the second search agent, the value of p is 0.8003 and is greater than 0.5. Therefore, the spiral updating position strategy is selected for the particle's movement. For this agent, the new position is obtained using:

$X\,() = D^{'} \cdot e^{bl} \cdot cos\,(2\pi l) + X^{*}\,(t)$

The updated position is computed as:

$D^{'} = \{4.5557, 2.8888, 0.3394, 0.7373, 0.3146\}$
$SA_2 = \{-2.7023, -0.8424, 0.7108, -3.5629, 1.7940\}$

For the third search agent, $p = 0.0835$ and $|A| < 1$, so the shrinking encircling mechanism is utilized for updating the position. As a result, the third search agent uses the following information for updating its position:

$D = \{3.6705, 4.6295, 0.5255, 6.8985, 1.6066\}$
$SA_3 = \{-1.0198, 0.6500, 0.8827, -2.1255, 2.1603\}$

The p value for the fourth search agent is more than 0.5, therefore, a spiral updating strategy is uses with the following result:

$D^{'} = \{5.8323, 2.7049, 2.8951, 1.4260, 2.1594\}$
$SA_4 = \{-0.2551, 0.4637, 1.5020, -3.0476, 2.3977\}$

For the fifth search agent, $p = 0.5269$ and the spiral updating position strategy is used with the following results:

$D' = \{5.7012, 0.2058, 2.3426, 3.2970, 6.3729\}$
$SA_5 = \{0.3949, -0.1300, 1.6457, -2.1900, 4.2031\}$

For the last search agent, $p = 0.2920$ and $|A| < 1$, so the shrinking encircling approach is used by the best solution leader to update the individual's position:

$D = \{0.5327, 0.0645, 0.2465, 1.0668, 0.5846\}$
$SA_6 = \{-1.5209, -0.1841, 0.8677, -3.0454, 2.0579\}$

Next, the boundary conditions are checked and every violated individual is pushed back into the permitted solution domain.

$Valid\_SA_1 = \{5.1200, 3.7251, 5.1200, 1.8466, 5.1200\}$
$Valid\_SA_2 = \{-2.7023, -0.8424, 0.7108, -3.5629, 1.7940\}$
$Valid\_SA_3 = \{-1.0198, 0.6500, 0.8827, -2.1255, 2.1603\}$
$Valid\_SA_4 = \{-0.2551, 0.4637, 1.5020, -3.0476, 2.3977\}$
$Valid\_SA_5 = \{0.3949, -0.1300, 1.6457, -2.1900, 4.2031\}$
$Valid\_SA_6 = \{-1.5209, -0.1841, 0.8677, -3.0454, 2.0579\}$

Next, the objective function values are computed for each search agent as:

$OF(Valid\_SA_1) = 95.9295$
$OF(Valid\_SA_2) = 24.4300$
$OF(Valid\_SA_3) = 11.4263$
$OF(Valid\_SA_4) = 17.5729$
$OF(Valid\_SA_5) = 25.3433$
$OF(Valid\_SA_6) = 25.3433$

In the final step, the leader's objective function value and position are updated based on the best-found solution.

$Leader\_position = \{-1.0198, 0.6500, 0.8827, -2.1255, 2.1603\}$
$Leader\_score = 11.4263$

24.6 Conclusions

In this chapter, the fundamentals of a WOA are described and examined. In support of this effort, pseudo-code for a WOA is presented to help to demonstrate how the algorithm works. Source-codes for a WOA are provided in both Matlab and C++ to help readers comprehend the fundamentals of the WOA. Finally, a step-by-step numerical example analysis is presented to help explain the mechanism of the WOA.

References

1. S. Mirjalili, A. Lewis, A. "The whale optimization algorithm". *Advances in Engineering Software*, vol. 95, pp. 51-67, 2016.
2. J.A. Goldbogen, A.S. Friedlaender, J. Calambokidis, M.F. Mckenna, M. Simon, D.P. Nowacek. "Integrative approaches to the study of baleen whale diving behavior, feeding performance, and foraging ecology". *BioScience*, vol. 63(2), pp. 90-100, 2013.
3. H.M. Hasanien. "Performance improvement of photovoltaic power systems using an optimal control strategy based on whale optimization algorithm". *Electric Power Systems Research*, vol. 157, pp. 168-176, 2018.
4. M. Mafarja, S. Mirjalili. "Whale optimization approaches for wrapper feature selection". *Applied Soft Computing*, vol. 62, pp. 441-453, 2018.
5. K. ben oualid Medani, S. Sayah, A. Bekrar. "Whale optimization algorithm based optimal reactive power dispatch: A case study of the Algerian power system". *Electric Power Systems Research*, vol. 163, pp. 696-705, 2018.
6. Y. Ling, Y. Zhou, Q. Luo. "Lévy Flight Trajectory-Based Whale Optimization Algorithm for Global Optimization". *IEEE Access*, vol. 5(99), pp. 6168-6186, 2017.
7. D. Rewadkar, D. Doye. "Adaptive-ARW: Adaptive autoregressive whale optimization algorithm for traffic-aware routing in urban VANET". *International Journal of Computer Sciences and Engineering*, vol. 6(2), pp. 303-312, 2018.
8. Y. Sun, X. Wang, Y. Chen, Z. Liu. "A modified whale optimization algorithm for large-scale global optimization problems". *Expert Systems with Applications*, vol. 114, pp. 563-577, 2018.
9. W.Z. Sun, J.S. Wang. "Elman Neural Network Soft-Sensor Model of Conversion Velocity in Polymerization Process Optimized by Chaos Whale Optimization Algorithm". *IEEE Access*, vol. 5, pp. 13062-13076, 2017.
10. M.A. El Aziz, A.A. Ewees, A.E. Hassanien, M. Mudhsh, S. Xiong. "Multi-objective Whale Optimization Algorithm for Multilevel Thresholding Segmentation". *Advances in Soft Computing and Machine Learning in Image Processing*, Springer, pp. 23-39, 2018.