16

Grey Wolf Optimizer

Ahmed F. Ali
Department of Computer Science
Suez Canal University, Ismaillia, Egypt

Mohamed A. Tawhid
Department of Mathematics and Statistics
Faculty of Science, Thompson Rivers University, Kamloops, Canada

CONTENTS

16.1 Introduction

Grey wolf optimizer (GWO) is a population based swarm intelligence algorithm proposed by Mirjalili et al. in 2014 [4]. The GWO algorithm mimics the social dominant structure of the grey wolves pack. Due to the efficiency of the GWO algorithm, it has been applied in many works such as for CT liver segmentation [1], minimizing potential energy function [6], feature selection [2], [9], solving minimax and integer programming problems [7], solving a global optimization problem [8], solving a flow shop scheduling problem [3], solving an optimal reactive power dispatch problem [5], and casting production scheduling [10]. The rest of the paper is organized as follows. We describe the standard GWO algorithm in Section 16.2. We give and discuss the Matlab and C++ source codes of the GWO algorithm in Sections 16.3–16.4. In Section 16.5, we demonstrate a step by step numerical example of the GWO algorithm. Finally, we outline the conclusion in Section 16.6.

16.2 Original GWO algorithm

In the following subsections, we describe the main concepts of the GWO algorithm and how it works.

16.2.1 Main concepts and inspiration

Grey wolf optimizer (GWO) is a population based swarm intelligence algorithm, which mimics the dominant hierarchy of grey wolves. Grey wolves live in packs, each pack contains 5-12 members. In the pack, there are four levels of members in the dominant hierarchy as follows.

16.2.2 Social hierarchy

The leader of the group is called alpha α which can be male or female. The alpha is the highest member of the hierarchy in the pack and he/she is responsible for hunting, selecting a sleeping place and determining time to walk. The second type of wolves in the pack are the beta β, which help the alpha in their decisions. The third level in the dominant hierarchy is called delta δ which submits to alpha and beta members. The weakest members in the pack are called omega ω, which submit to the members in the prior top levels. The dominant hierarchy levels are shown in Figure 16.1.

16.2.3 Encircling prey

In this subsection, we give a mathematical model of the encircling prey process as follows.

$$\vec{D} = |\vec{C} \cdot \vec{X_p}(t) - \vec{A} \cdot \vec{X}(t)| \tag{16.1}$$

$$\vec{X}(t+1) = \vec{X_p}(t) - \vec{A} \cdot \vec{D} \tag{16.2}$$

where t is the current iteration, $\vec{A}$ and $\vec{C}$ are the coefficient vectors, $\vec{X_p}$ is the prey's position vector, and $\vec{X}$ indicates the grey wolf's position vector.

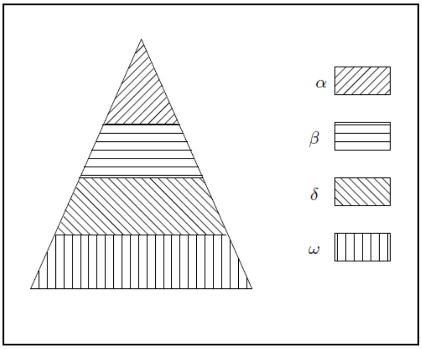

FIGURE 16.1
Social dominant hierarchy of grey wolf.

The vectors $\vec{A}$ and $\vec{C}$ are calculated as follows:

$$\vec{A} = 2\vec{a} \cdot \vec{r_1} \cdot \vec{a} \tag{16.3}$$

$$\vec{C} = 2 \cdot \vec{r_2} \tag{16.4}$$

where components of $\vec{a}$ are linearly decreased from 2 to 0 over the course of iterations and $\vec{r_1}$, $\vec{r_2}$ are random vectors in $[0, 1]$

16.2.4 Hunting process

The alpha guides the beta and delta in the hunting process. In the mathematical model, the alpha represents the overall best solutions, while the beta and delta represent the second and third best solutions in the population. All solutions (wolves) update their positions according to the position of the best first three solutions (alpha, beta and delta). The formula of the mathematical model of the hunting process is shown as follows.

$$\begin{aligned} \vec{D_\alpha} = &\ |\vec{C_1} \cdot \vec{X_\alpha} - \vec{X}|, \\ \vec{D_\beta} = &\ |\vec{C_2} \cdot \vec{X_\beta} - \vec{X}|, \\ \vec{D_\delta} = &\ |\vec{C_3} \cdot \vec{X_\delta} - \vec{X}|, \end{aligned} \tag{16.5}$$

$$\begin{aligned} \vec{X_1} = &\ \vec{X_\alpha} - \vec{A_1} \cdot (\vec{D_\alpha}), \\ \vec{X_2} = &\ \vec{X_\beta} - \vec{A_2} \cdot (\vec{D_\beta}), \\ \vec{X_3} = &\ \vec{X_\delta} - \vec{A_3} \cdot (\vec{D_\delta}), \end{aligned} \tag{16.6}$$

$$\vec{X}(t+1) = \frac{\vec{X_1} + \vec{X_2} + \vec{X_3}}{3}, \tag{16.7}$$

where $\vec{X_1}$, $\vec{X_2}$ and $\vec{X_3}$ are the first three solutions in the population.

16.2.5 Attacking prey (exploitation)

The hunting process is finished by attacking the prey. This process can be mathematically modeled as follows. When $|A| < 1$, the wolves attack towards the prey, where the vector $\vec{A}$ is a random value in interval $[-2a, 2a]$ and a is decreased from 2 to 0 over the course of iterations.

16.2.6 Search for prey (exploration)

At the beginning of the search each wolf updates its position according to the position of α, β and δ in order to search for prey and converge to attack prey. This process is called diversification or exploration. The mathematical model of the exploration is defined as follows. When $|A| > 1$, the wolves (solutions) are forced to diverge from the prey to find a fitter prey. The vector $\vec{A}$ with random values greater than 1 or less than -1 to force the search agent to diverge from the prey.

16.2.7 Pseudo-code of GWO algorithm

Algorithm 15 Grey wolf optimizer algorithm.

```
Set the initial values of the population size n, parameter a and the maximum number of iterations Max_itr
Set t := 0
for (i = 1 : i ≤ n) do
    Generate an initial population X_i(t) randomly
    Evaluate the fitness function of each search agent (solution) f(X_i)
end for
Assign the values of the first, second and third best solutions X_α, X_β and X_δ, respectively
repeat
    for (i = 1 : i ≤ n) do
        Decrease the parameter a from 2 to 0
        Update the coefficients A and C as shown in Equations (16.3) and (16.4), respectively
        Update each search agent in the population as shown in Equations (16.5), (16.6), (16.7)
        Evaluate the fitness function of each search agent f(X_i)
    end for
    Update the vectors X_α, X_β and X_δ.
    Set t = t + 1
until (t ≥ Max_itr).                    ▷ Termination criteria are satisfied
Produce the best solution X_α
```

16.2.8 Description of the GWO algorithm

- **Parameters initialization.** The standard grey wolf optimizer algorithm starts by initializing the parameters of the population size n, the parameter a and the maximum number of iterations Max_{itr}.

- **Iteration initialization.** Initialize the iteration counter t.

- **Initial population generation.** The initial population n is randomly generated.

- **Solution evaluation.** In the population, each search agent (solution) $\vec{X}_i$ is evaluated by calculating its fitness function $f(\vec{X}_i)$.

- **Assign the overall best three solutions.** The overall best three solutions are assigned which are the alpha α, beta β and delta δ solution $\vec{X}_\alpha$, $\vec{X}_\beta$ and $\vec{X}_\delta$, respectively.

- **Main loop.** The following steps are repeated until the termination criterion is satisfied.

  Solution update. Each search agent (solution) in the population is updated according to the position of the α, β and δ solutions as shown in Equation (16.7).

  Parameter a update. Gradually decrease the parameter a from 2 to 0.

  Coefficients update. The coefficients $\vec{A}$ and $\vec{C}$ are updated as shown in Equations (16.3) and (16.4), respectively.

  Solution evaluation. Each search agent (solution) in the population is evaluated by calculating its fitness function $f(\vec{X}_i)$.

- **Update the overall best three solutions.** The first, second and third best solutions are updated $\vec{X}_\alpha$, $\vec{X}_\beta$ and $\vec{X}_\delta$, respectively.

- **Iteration counter increasing.** The iteration counter is increasing, where $t = t + 1$.

- **Termination criteria satisfied.** All the previous processes are repeated until termination criteria are satisfied.

- **Produce the overall best solution.** The overall best solution $\vec{X}_\alpha$ is produced.

16.3 Source-code of GWO algorithm in Matlab

In this section, we present the source code of the used fitness function which we need to minimize by using the GWO algorithm as shown in Listing 16.1.

The fitness function is shown in Equation 16.8. The function evaluates each solution in population X. Also, we present the source codes of the main GWO algorithm in Matlab [4] as shown in Listing 16.2.

$$f(X_i) = \sum_{j=1}^{D} X_{i,j}^2 \qquad \text{where } -10 \leqslant X_{i,j} \leqslant 10 \tag{16.8}$$

```
function [out]=fun(X)
[x,y]=size(X);
out=zeros(x,1); for i=1:x
    for j=1:y
    out(i,1)=out(i,1)+X(i,j)^2;
    end
end

```

Listing 16.1
Definition of objective function $fun(.)$ in Matlab.

```

% initialize alpha, beta, and delta_pos
Alpha_pos=zeros(1,dim);
Alpha_score=inf; %change this to -inf for maximization problems

Beta_pos=zeros(1,dim); Beta_score=inf; %change this to -inf for maximization problems
Delta_pos=zeros(1,dim); Delta_score=inf; %change this to -inf for maximization problems
%Initialize the positions of search agents
Positions=initialization(SearchAgents_no,dim,ub,lb);
Convergence_curve=zeros(1,Max_iter); l=0;
% Loop counter
% Main loop
while l<Max_iter
for i=1:size(Positions,1)
%Calculate objective function for each search agent fitness=fobj(Positions(i,:));
% Update Alpha, Beta, and Delta
     if fitness<Alpha_score
         Alpha_score=fitness; % Update alpha
         Alpha_pos=Positions(i,:);
     end
     if fitness>Alpha_score && fitness<Beta_score Beta_score=fitness; % Update beta
         Beta_pos=Positions(i,:);
     end
     if fitness>Alpha_score && fitness>Beta_score && fitness<Delta_score
         Delta_score=fitness; % Update delta Delta_pos=Positions(i,:);
     end
end
a=2-l*((2)/Max_iter); % a decreases linearly from 2 to 0
% Update the Position of search agents including omegas
for i=1:size(Positions,1)
 for j=1:size(Positions,2)
  r1=rand(); % r1 is a random number in [0,1]
  r2=rand(); % r2 is a random number in [0,1]
  A1=2*a*r1-a;
  C1=2*r2;
  D_alpha=abs(C1*Alpha_pos(j)-Positions(i,j));
  X1=Alpha_pos(j)-A1*D_alpha;
  r1=rand();
```

```
    r2=rand();
    A2=2*a*r1-a;
    C2=2*r2;
    D_beta=abs(C2*Beta_pos(j)-Positions(i,j));
    X2=Beta_pos(j)-A2*D_beta;
    r1=rand();
    r2=rand();
    A3=2*a*r1-a;
    C3=2*r2;
    D_delta=abs(C3*Delta_pos(j)-Positions(i,j));
    X3=Delta_pos(j)-A3*D_delta;
    Positions(i,j)=(X1+X2+X3)/3;
    end % 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;
 end
 l=l+1;
Convergence_curve(l)=Alpha_score;
end
```

Listing 16.2
The main code for the grey wolf optimization algorithm $GWO(.)$ in Matlab.

The rest of the matlab code is presented in https://www.mathworks.com/matlabcentral/fileexchange/ 44974-grey-wolf-optimizer-gwo

16.4 Source-code of GWO algorithm in C++

In this section, we present the C++ code of the tested objective function and the GWO algorithm as shown in Listings 16.3–16.4.

```
class fun : public Problem
{ public:
fun(unsigned int dimension) : Problem(dimension) { }
double eval(const std::vector<double>& solution)
{
     double sum = 0.0;
     for (int i = 0; i < solution.size(); ++i)
       {
         sum += solution[i] * solution[i];
       }
return sum;
}
```

Listing 16.3
Definition of objective function $fun(.)$ and the main file in C++.

```
#include "grey_wolf_optimizer.hpp"
#include "benchmark_functions.hpp"
#include "optimization_utils.hpp"
#include <limits>
#include <iostream>
#include <exception>
#include <cmath>
```

```cpp
solution grey_wolf_optimizer(function f, calculation_type calc_type_a, calculation_type calc_type_c,
int max_number_of_evaluations, int number_of_agents, double left_bound, double right_bound, int dimension)
{
std::vector<double> alpha_pos(dimension, 0.);
double alpha_score = std::numeric_limits<double>::infinity();
std::vector<double> beta_pos(dimension, 0.);
double beta_score = std::numeric_limits<double>::infinity();
std::vector<double> delta_pos(dimension, 0.);
double delta_score = std::numeric_limits<double>::infinity();
auto positions = get_initial_positions(left_bound, right_bound, dimension, number_of_agents);
solution s{};
int iteration{0};
const int max_number_of_iterations{max_number_of_evaluations / number_of_agents};
while (iteration++ < max_number_of_iterations)
  {
   clip_positions(positions, left_bound, right_bound);

   for (auto &agent : positions)
    {
      double fitness = objective_function(f, agent);

        if (fitness < alpha_score)
          {
            alpha_score = fitness;
            alpha_pos = agent;
           }

        if (fitness > alpha_score and fitness < beta_score)
          {
            beta_score = fitness;
            beta_pos = agent;
          }

       if (fitness > alpha_score and fitness > beta_score and fitness < delta_score)
          {
           delta_score = fitness;
           delta_pos = agent;
          }
   }
      double a = calculate_a(calc_type_a, iteration, max_number_of_iterations);

      for (auto &agent : positions)
        for (auto j = 0u; j < agent.size(); ++j)
          {
           // alpha
           double r1 = get_random(0., 1.);
           double r2 = get_random(0., 1.);

           double A1 = 2. * a * r1 - a;
           double C1 = calculate_c(calc_type_c, r2, iteration, max_number_of_iterations);
           const double D_alpha = std::abs(C1 * alpha_pos[j] - agent[j]);
           const double X1 = alpha_pos[j] - A1 * D_alpha;

           // beta
           r1 = get_random(0., 1.);
           r2 = get_random(0., 1.);
           double A2 = 2. * a * r1 - a;
           double C2 = calculate_c(calc_type_c, r2, iteration, max_number_of_iterations);
```

```
            const double D_beta = std::abs(C2 * beta_pos[j] - agent[j]);
            const double X2 = beta_pos[j] - A2 * D_beta;

            // delta
            r1 = get_random(0., 1.);
            r2 = get_random(0., 1.);
            double A3 = 2. * a * r1 - a;
            double C3 = calculate_c(calc_type_c, r2, iteration,
    max_number_of_iterations);
            const double D_delta = std::abs(C3 * delta_pos[j] - agent[j
    ]);
            const double X3 = delta_pos[j] - A3 * D_delta;
            agent[j] = (X1 + X2 + X3) / 3.;
          }
          s.convergence.push_back(alpha_score);
          s.best = alpha_score;
    }
        return s;
  }
```

Listing 16.4
SSA header file in C++.

The rest of the grey wolf optimization C++ code is represented in https://github.com/czeslavo/gwo/blob/master/optimization/grey_wolf_optimizer.cpp

16.5 Step-by-step numerical example of GWO algorithm

In this section, we present the GWO algorithm when it applies to minimize the objective function in Equation 16.8, at dimension $D = 5$. In step one, the algorithm starts by setting the initial values of the population size $n = 6$, parameter $a = 2$ and maximum number of iterations $Max_{itr} = 100$. In step two, we initialize the iteration counter where $t = 0$. In step three, we start the loop to create the initial population in the GWO algorithm. In step four, we initialize the population. Each agent (solution) in the population is a vector with five variables ($D = 5$). The population is represented as follows

$\vec{X_1} = \{6.2945, -4.4300, 9.1433, 5.8441, 3.5747\}$
$\vec{X_2} = \{8.1158, 0.9376, -0.2925, 9.1898, 5.1548\}$
$\vec{X_3} = \{-7.4603, 9.1501, 6.0056, 3.1148, 4.8626\}$
$\vec{X_4} = \{8.2675, 9.2978, -7.1623, -9.2858, -2.1555\}$
$\vec{X_5} = \{2.6472, -6.8477, -1.5648, 6.9826, 3.1096\}$
$\vec{X_6} = \{-8.0492, 9.4119, 8.3147, 8.6799, -6.5763\}$

In step five, we evaluate each agent (solution) in the population by calculating its objective function $OF(.)$ in Equation 16.8. For agent $X_1 = \{6.2945, -4.4300, 9.1433, 5.8441, 3.5747\}$, the objective function of it is calculated as the following $OF(\vec{X_1}) = (6.2945)^2 + (-4.4300)^2 + (9.1433)^2 + (5.8441)^2 + (3.5747)^2 = 189.7788$. The objective function of each agent in the population is presented as follows.

$$OF(\vec{X_1}) = 189.7788$$
$$OF(\vec{X_2}) = 177.8568$$
$$OF(\vec{X_3}) = 208.7953$$
$$OF(\vec{X_4}) = 296.9700$$
$$OF(\vec{X_5}) = 114.7735$$
$$OF(\vec{X_6}) = 341.0943.$$

In step seven, the overall best three solutions are assigned according to their objective functions. The overall best solution is $\vec{X_\alpha}$, where $\vec{X_\alpha} = \{2.6472, -6.8477, -1.5648, 6.9826, 3.1096\}$. The second overall best solution is $\vec{X_\beta}$, where $\vec{X_\beta} = \{8.1158, 0.9376, -0.2925, 9.1898, 5.1548\}$.
The third overall best solution in the population is the $\vec{X_\delta}$, where $\vec{X_\delta} = \{6.2945, -4.4300, 9.1433, 5.8441, 3.5747\}$.

In step eight, we start the main loop of the GWO algorithm. In steps nine and ten, for each agent in the population, the coefficient a is initialized, where $a = 2$. In step eleven, for the first three solutions in the population, the coefficients $\vec{A}$ and $\vec{C}$ are updated as shown in Equations 16.3–16.4. The values of vectors $\vec{A}$ and $\vec{C}$ for the first three agents (solutions) in the population are presented as follows.

$$\vec{A_1} = \{0.8242, 0.7793, 1.0621, 0.8375, -1.5240\}$$
$$\vec{C_1} = \{0.0637, 0.6342, 1.5904, 1.5094, 0.9967\}$$
$$\vec{A_2} = \{-0.8923, 1.8009, -1.2525, -0.8959, 1.8390\}$$
$$\vec{C_2} = \{0.0923, 0.0689, 0.9795, 1.3594, 0.6808\}$$
$$\vec{A_3} = \{-1.6115, -0.2450, -0.2177, 0.6204, 0.3411\}$$
$$\vec{C_3} = \{1.6469, 0.7631, 1.2926, 0.3252, 0.4476\}$$

In step twelve, we update the solutions in the population as shown in Equations 16.5–16.7 based on the updating of the first three solutions in the population as shown below

$$\begin{aligned}
\vec{D_\alpha} = &|\{0.0637, 0.6342, 1.5904, 1.5094, 0.9967\} \\
&\cdot\{2.6472, -6.8477, -1.5648, 6.9826, 3.1096\} \\
&-\{6.2945, -4.4300, 9.1433, 5.8441, 3.5747\}| \\
&= \{6.1259, 0.0872, 11.6320, 4.6952, 0.4753\}. \\
\vec{D_\beta} = &|\{0.0923, 0.0689, 0.9795, 1.3594, 0.6808\} \\
&\cdot\{8.1158, 0.9376, -0.2925, 9.1898, 5.1548\} \\
&-\{8.1158, 0.9376, -0.2925, 9.1898, 5.1548\}| \\
&= \{6.9834, 5.0604, 3.2607, 1.6099, 0.2644\}. \\
\vec{D_\delta} = &|\{1.6469, 0.7631, 1.2926, 0.3252, 0.4476\} \\
&\cdot\{6.2945, -4.4300, 9.1433, 5.8441, 3.5747\} \\
&-\{-7.4603, 9.1501, 6.0056, 3.1148, 4.8626\} \\
&= \{7.3214, 11.5253, 18.4015, 8.8641, 4.5395\}. \\
\vec{X_1} = &\{2.6472, -6.8477, -1.5648, 6.9826, 3.1096\} \\
&-\{0.8242, 0.7793, 1.0621, 0.8375, -1.5240\} \\
&= \{-2.4017, -6.9157, -13.9187, 3.0506, 3.8339\}.
\end{aligned}$$

$\vec{X_2} = \{8.1158, 0.9376, -0.2925, 9.1898, 5.1548\}$
$-\{-0.8923, 1.8009, -1.2525, -0.8959, 1.8390\}$
$\cdot\{6.9834, 5.0604, 3.2607, 1.6099, 0.2644\}$
$= \{-1.2289, 0.0369, 10.0896, 4.5571, 4.3765\}$
$\vec{X_3} = \{6.2945, -4.4300, 9.1433, 5.8441, 3.5747\}$
$-\{-1.6115, -0.2450, -0.2177, 0.6204, 0.3411\}$
$\cdot\{7.3214, 11.5253, 18.4015, 8.8641, 4.5395\}$
$= \{20.0658, 12.1217, -3.1571, -14.7850, -3.7038\}$.
The new solutions in the population are generated as shown in Equation 16.7 and we obtain the following new $\vec{X_1} = \{5.4784, 1.7476, -2.3287, -2.3924, 1.5022\}$.
In step thirteen, the objective function of each solution in the new population is calculated and we obtain the following values.
$OF(\vec{X_1}) = 61.2648$
$OF(\vec{X_2}) = 167.8913$
$OF(\vec{X_3}) = 54.6693$
$OF(\vec{X_4}) = 173.1462$
$OF(\vec{X_5}) = 24.3717$
$OF(\vec{X_6}) = 216.7138$
In step fifteen, the vectors $\vec{X_\alpha}, \vec{X_\beta}, \vec{X_\delta}$ are updated to be as the following.
$\vec{X_\alpha} = \{-3.4887, 0.0886, 0.0813, 2.1271, -2.7680\}$
$\vec{X_\beta} = \{4.4180, -3.7046, -4.2665, 1.5438, -0.9172\}$
$\vec{X_\delta} = \{-0.2841, -6.4442, -0.7633, -2.0841, 3.8379\}$
In step sixteen, the iteration counter is increased to be $t = 1$. In step seventeen, the termination criterion is tested and if it not satisfied, we return to step nine; otherwise the algorithm produces the overall best solution $\vec{X_\alpha}$ as shown in step eighteen.

16.6 Conclusion

In this chapter, we present the main steps of the GWO algorithm and how it works. We present the source code in Matlab and C++ language to help the user to implement it on various applications. In order to give a better understanding of the GWO algorithm, we demonstrate a step by step numerical example and we show how it can solve the global optimization problem.

References

1. A.F. Ali, A. Mostafa, G.I, Sayed, M.A. Elfattah and A.E. Hassanien. Nature inspired optimization algorithms for ct liver segmentation. In: Dey N., Bhateja V., Hassanien A. (eds) Medical Imaging in

Clinical Applications. Studies in Computational Intelligence, vol. 651, pp. 431-460, Springer, 2016.

2. E.Emary, H.M. Zawbaa and A.E. Hassanien. Binary grey wolf optimization approaches for feature selection. Neurocomputing, 172, 371-381, 2016.
3. G. M. Komaki and V. Kayvanfar. Grey Wolf Optimizer algorithm for the two-stage assembly flow shop scheduling problem with release time. Journal of Computational Science, 8, 109-120, 2015.
4. S. Mirjalili, S. M. Mirjalili, and A. Lewis, Grey Wolf Optimizer, Advances in Engineering Software, 69 (2014), 46–61.
5. M. H. Sulaiman, Z. Mustaffa, M. R. Mohamed, and O. Aliman. Using the gray wolf optimizer for solving optimal reactive power dispatch problem. Applied Soft Computing, 32, 286-292, 2015.
6. M.A. Tawhid and A.F. Ali. A hybrid grey wolf optimizer and genetic algorithm for minimizing potential energy function. Memetic Computing, 9(4), 347-359, 2017.
7. M.A. Tawhid and A.F. Ali. A simplex grey wolf optimizer for solving integer programming and minimax problems. Numerical Algebra,Control & Optimization, 7(3), 301-323, 2017.
8. M.A. Tawhid and A.F. Ali. Multidirectional grey wolf optimizer algorithm for solving global optimization problems. International Journal of Computational Intelligence and Applications, 1850022, 2018.
9. Q. Tu, X. Chen, and X. Liu. Multi-strategy ensemble grey wolf optimizer and its application to feature selection. Applied Soft Computing, 76, 16-30, 2019.
10. H. Qin, P. Fan, H. Tang, P. Huang, B. Fang and S. Pan. An effective hybrid discrete grey wolf optimizer for the casting production scheduling problem with multi-objective and multi-constraint. Computers & Industrial Engineering, 128, 458-476, 2019.