15

Grasshopper Optimization Algorithm

Szymon Łukasik
Faculty of Physics and Applied Computer Science
AGH University of Science and Technology, Kraków, Poland

CONTENTS

15.1 Introduction

Grasshopper Optimization Algorithm (GOA) is an optimization technique introduced by Saremi, Mirjalili and Lewis in 2017 [9]. It belongs to the class of swarm optimization strategies [5]. The GOA procedure includes both social interaction between ordinary agents (grasshoppers) and the attraction of the best individual. Initial experiments performed by authors demonstrated promising exploration abilities of the algorithm – and they will be further examined in the course of our study.

GOA is reported to implement two components of grasshopper movement strategies. First it is the interaction of grasshoppers which demonstrates itself through slow movements (while in larvae stage) and dynamic motion (while in insect form). The second corresponds to the tendency to move towards the source of food. What is more, deceleration of grasshoppers approaching food and eventually consuming it is also taken into account [12].

The rest of this paper is organized as follows: in Section 15.2 we are providing more detailed description of the algorithm along with the pseudo-code of its implementation. Section 15.3 contains the implementation of the algorithm using the Matlab environment. It is followed by the detailed analysis of the algorithm dynamics – presented on the simple Sphere function benchmark in Section 15.5. Finally in Section 15.6 some final comments on the algorithm's properties are being provided.

15.2 Description of the Grasshopper Optimization Algorithm

This chapter, as others in this monograph, deals with continuous optimization, i.e. the task of finding a value of x – within the feasible search space $S \subset R^D$ – denoted as x^* such as $x^* = \text{argmin}_{x \in S} f(x)$, assuming that the goal is to minimize cost function f. GOA represents a typical population based meta-heuristics [10]. It means that the aforementioned problem is tackled with the population consisting of P agents of the same type. Each agent is represented as a solution vector x_p, $p = 1, ..., P$ and represents exactly one solution in the domain of tested function f[6].

The movement of individual p in iteration k can be presented using the following equation:

$$x_{pd} = c\left(\sum_{q=1,q\neq p}^{P} c\frac{UB_d - LB_d}{2} s(|x_{qd} - x_{pd}|) \frac{x_{qd} - x_{pd}}{dist(x_q, x_p)}\right) + x_d^* \tag{15.1}$$

with $d = 1, 2, ..., D$ representing search space dimensionality. Note that index k was omitted for the sake of readability. The equation contains two components: the first corresponds to the pairwise social interactions between grasshoppers, the second – to the movement attributed to the wind (in the algorithm – in the direction of the best individual). The impact of gravitation – though important for the real grasshopper swarms [4] – is not included in the basic algorithm scheme.

Function s, present in the first factor of 15.1, defines the strength of social forces, and was defined by creators of the algorithm as:

$$s(r) = fe^{\frac{-r}{l}} - e^{-r} \tag{15.2}$$

with default values of $l = 1.5$ and $f = 0.5$.

It divides the space between two considered grasshoppers into three separate zones, as demonstrated in Figure 15.1. Individuals being very close are found in the so called repulsion zone. On the other hand distant grasshoppers are located in the zone of attraction. The zone (or equilibrium state) between them – called the comfort zone – is characterized by the lack of social interactions. The first factor is additionally normalized by the upper and lower bounds of the feasible search space.

In addition to that parameter c – occurring twice in formula (15.1) – is decreased according to the the following equation:

$$c = c_{max} - k\frac{c_{max} - c_{min}}{K_{max}} \tag{15.3}$$

with maximum and minimum values – c_{max}, c_{min} respectively – and K representing maximum number of iterations serving as the algorithm's termination criterion. The first occurrence of c in (15.1) reduces the movements of grasshoppers around the target – balancing between exploration and exploitation of the swarm around the target. It is analogous to the inertia weight present in the Particle Swarm Optimization Algorithm [3]. The whole component $c\frac{UB_d - LB_d}{2}$, as noted in [9], linearly decreases the space that the grasshoppers should explore and exploit.

The pseudo-code for the generic version of the GOA technique is presented in Algorithm 14.

Algorithm 14 Grasshopper Optimization Algorithm.

```
1:  $k \leftarrow 1$, $f(x^*(0)) \leftarrow \infty$                                ▷ initialization
2:  for $p = 1$ to $P$ do
3:      $x_p(k) \leftarrow$ Generate_Solution($LB, UB$)
4:  end for
5:                                                                               ▷ find best
6:  for $p = 1$ to $P$ do
7:      $f(x_p(k)) \leftarrow$ Evaluate_quality($x_p(k)$)
8:      if $f(x_p(k)) < f(x^*(k-1))$ then
9:          $x^*(k) \leftarrow x_p(k)$
10:     else
11:         $x^*(k) \leftarrow x^*(k-1)$
12:     end if
13: end for
14: repeat
15:     $c \leftarrow$ Update_c($c_{max}, c_{min}, k, K_{max}$)
16:     for $p = 1$ to $P$ do
17:                                                  ▷ move according to formula (15.1)
18:         $x_p(k) \leftarrow$ Move_Grasshopper($c, UB, LB, x^*(k)$)
19:                                                          ▷ correct if out of bounds
20:         $x_p(k) \leftarrow$ Correct_Solution($x_p(k), UB, LB$)
21:         $f(x_p(k)) \leftarrow$ Evaluate_quality($x_p(k)$)
22:         if $f(x_p(k)) < f(x^*k)$ then
23:             $x^*(k) \leftarrow x_p(k)$, $f(x^*k) \leftarrow f(x_p(k))$
24:         end if
25:     end for
26:     for $p = 1$ to $P$ do
27:         $f(x_p(k+1)) \leftarrow f(x_pk)$, $x_p(k+1) \leftarrow x_p(k)$
28:     end for
```

```
29:     $f(x^*(k+1)) \leftarrow f(x^*k)$, $x^*(k+1) \leftarrow x^*(k)$
30:     $k \leftarrow k+1$
31: until $k < K$ return $f(x^*(k))$, $x^*(k)$
```

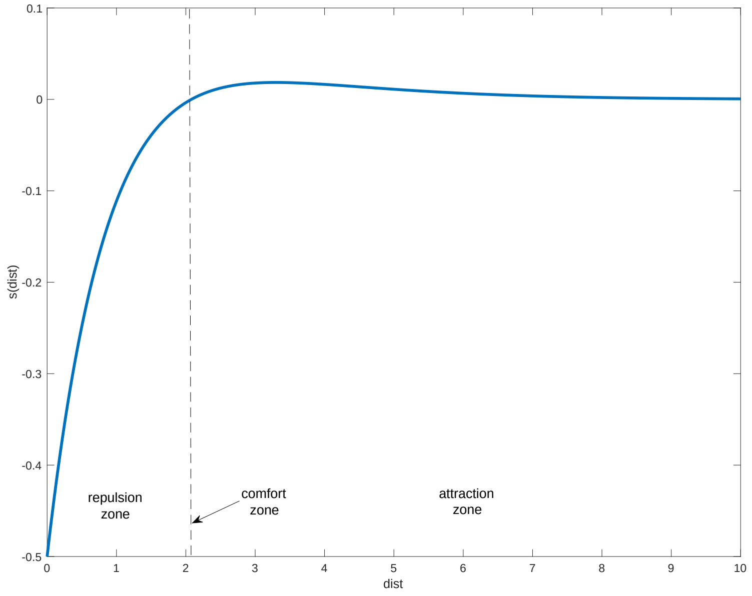

FIGURE 15.1
Function s for $l = 1.5$ and $f = 0.5$ and zones of grasshoppers' movement.

15.3 Source-code of GOA in Matlab

Listing 15.1 provides the source-code for the GOA technique in MATLAB. The code uses additional parameters to set up the algorithm's initialization. They include population size, maximum number of iterations, bounds for the search space and its dimensionality, and finally – the objective function. The location of the minimum found (denoted as *X_ best*) along with the corresponding value of the objective function (*fit_ best*) are being returned.

```

% P - population size
% k_max - max. iterations
% LB, UB - lower & upper bound of the search space
% N - search space dimensionality
% fobj - objective function

if size(UB,1)==1
    UB=ones(N,1)*UB;
    LB=ones(N,1)*LB;
end

% original implementation assumes even number of variables
% if odd number is used additional "dummy" variable is created
odd=false;
if (rem(N,2)~=0)
    N = N+1;
    UB = [UB; 100];
    LB = [LB; -100];
    odd=true;
end

% initialize the population of grasshoppers
X=zeros(P,N);
for i=1:P
    X(i,:)=[rand(N,1).*(UB-LB)+LB]';
endfit_X = zeros(1,P);

% default parameter values
cMax=1;
cMin=0.00004;

% calculate the fitness of initial population
for i=1:P
  if odd == true
      fit_X(1,i)=fobj(X(i,1:end-1));
  else
      fit_X(1,i)=fobj(X(i,:));
  end
end

% sort and find the best
[sorted_fit_X,sorted_index]=sort(fit_X);
    for i=1:P
    sorted_X(i,:)=X(sorted_index(i),:);
end

X_best=sorted_X(1,:);
fit_best=sorted_fit_X(1);

% Main loop
iter=1;
while iter<k_max
    c=cMax-iter*((cMax-cMin)/k_max);
    for i=1:P
        temp= X';
        for k=1:2:N
          S_i=zeros(2,1);
          for j=1:P
            if i~=j
              % calculate the 2D distance between two grasshoppers
              Dist=pdist2(temp(k:k+1,j)', temp(k:k+1,i)');
              % get xj-xi/dij
              r_ij_vec=(temp(k:k+1,j)-temp(k:k+1,i))/(Dist+eps);
              % get |xjd - xid|
              xj_xi=2+rem(Dist,2);
              s_ij=((UB(k:k+1) - LB(k:k+1))*c/2)*S_func(xj_xi).*
              r_ij_vec;
              S_i=S_i+s_ij;
            end
          end
          S_i_total(k:k+1, :) = S_i;
          end
          X_new = c * S_i_total'+ (X_best); % new grasshopper position
          X_temp(i,:)=X_new';
     end
     X=X_temp;
     for i=1:P
          % check for solutions out of bounds and fix
          out_U=X(i,:)>UB';
          out_L=X(i,:)<LB';
          X(i,:)=(X(i,:).*(~(out_U+out_L)))+UB'.*out_U+LB'.*out_L;
          % and get fitness
          if odd == true
            fit_X(1,i)=fobj(X(i,1:end-1));
          else
            fit_X(1,i)=fobj(X(i,:));
          end
          % Update the best solution (if needed)
          if fit_X(1,i)<fit_best
            X_best=X(i,:);
            fit_best=fit_X(1,i);
          end
     end
     iter = iter + 1;
end

if (odd==true)
     X_best = X_best(1:N-1);
end

function o=S_func(r)
     f=0.5;
     l=1.5;
     o=f*exp(-r/l)-exp(-r);
end
```

Listing 15.1
Source-code of the GOA technique in Matlab. This code represents only a minor modification of the original implementation by the algorithm co-author Seyedali Mirjalili.

The objective function which will be used in the following tests is known as the Sphere function. It is frequently used as an elementary unimodal benchmark function [2] and is described by the following formula:

$$f(x) = \sum_{i=1}^{N} x_i^2 \qquad \text{where } -5.12 \leqslant x_i \leqslant 5.12. \tag{15.4}$$

Its minimum $f(x^*) = 0$ is located at $x^* = [0, 0, ..., 0]^N$. It can be coded using MATLAB as described in the Listing 15.2.

```
function y = sphere(x)
  y=sum(x.^2);
end
```

Listing 15.2
Definition of the Sphere objective function in Matlab.

15.4 Source-code of GOA in C++

Listing 15.3 provides the C++ implementation of GOA technique. For the linear algebra the Armadillo library was used [1]. It saves the user from the additional time overhead needed to implement vectorized operations. It allows us also to use automatic multi-threading.

```

double Dist, c, cMax, cMin, xj_xi ;
int i, iter, j, k, odd ;
mat X, X_new, X_temp, fit_X, r_ij_vec, s_ij, sorted_X, temp ;
rowvec out_L, out_U ;
vec S_i, S_i_total, sorted_fit_X, sorted_index ;
i
f (UB.n_rows==1)
{
  UB = arma::ones<mat>(N, 1)*UB ;
  LB = arma::ones<mat>(N, 1)*LB ;
}

X = arma::zeros<mat>(P, N) ;
sorted_X = arma::zeros<mat>(P, N) ;
X_temp = arma::zeros<mat>(P, N) ;
S_i_total = arma::zeros<vec>(N) ;
c = double(arma::as_scalar(arma::zeros<rowvec>(1))) ;

for (i=1; i<=P; i++)
{
  X.row(i-1) = arma::trans(arma::randu<mat>(N, 1)%(UB-LB)+LB) ;
}

fit_X = arma::zeros<mat>(1, P) ;
cMax = double(1) ;
cMin = 0.00004 ;

for (i=1; i<=P; i++)
{
  fit_X(0, i-1) = fobj(X.row(i-1)) ;
}

[sorted_fit_X, sorted_index] = sort(fit_X) ;

for (i=1; i<=P; i++)
{
  sorted_X.row(i-1) = X.row((uword) sorted_index(i-1)-1) ;
}

X_best = sorted_X.row(0) ;
fit_best = sorted_fit_X(0) ;
iter = 1 ;
k = 1 ;
while (iter<k_max)
{
  c = cMax-iter*((cMax-cMin)/k_max) ;
  for (i=1; i<=P; i++)
  {
    temp = arma::trans(X) ;
    for (k=1; k<=N; k+=2)
    {
      S_i = arma::zeros<vec>(2) ;
      for (j=1; j<=P; j++)
      {
        if (i!=j)
        {
          Dist = pdist2(arma::trans(temp(arma::span(k-1, k), m2cpp::
          span<uvec>(j-1, j-1))), arma::trans(temp(arma::span(k-1, k),
          m2cpp::span<uvec>(i-1, i-1)))) ;
          r_ij_vec = (temp(arma::span(k-1, k), m2cpp::span<uvec>(j-1,
          j-1))-temp(arma::span(k-1, k), m2cpp::span<uvec>(i-1, i-1)))
          /(Dist+datum::eps) ;
          xj_xi = 2+rem(Dist, 2) ;
          s_ij = ((UB(arma::span(k-1, k))-LB(arma::span(k-1, k)))*
          c/2.0)*S_func(xj_xi)%r_ij_vec ;
          S_i = S_i+s_ij ;
        }
      }
      S_i_total(arma::span(k-1, k)-1, m2cpp::span<uvec>(0, S_i_total.
      n_cols-1)-1) = S_i ;
    }
    X_new = c*S_i_total ;
    X_temp.row(i-1) = arma::trans(X_new) ;
  }

  X = X_temp ;
  for (i=1; i<=P; i++)
  {
    out_U = X.row(i-1)>arma::trans(UB) ;
    out_L = X.row(i-1)<arma::trans(LB) ;
    X.row(i-1) = (X.row(i-1)%(!(out_U+out_L)))+arma::trans(UB)%out_U+
    arma::trans(LB)%out_L ;
    fit_X(0, i-1) = fobj(X.row(i-1)) ;
  }

  if (fit_X(0, i-1)<fit_best)
  {
    X_best = X.row(i-1) ;
    fit_best = fit_X(0, i-1) ;
  }
}
iter = iter+1 ;

double fobj(rowvec x)
{
  double o ;
  o = double(arma::as_scalar(arma::sum(arma::square(x)))) ;
  return o ;
}
```

Listing 15.3
Source-code of the GOA technique in C++.

The following Section contains a numerical example of GOA run on a two-dimensional instance of the Sphere function minimization problem.

15.5 Step-by-step numerical example of GOA

As was already mentioned an illustrative example Grasshopper Optimization Algorithm will be used here to solve the minimization task realized for a simple two-dimensional variant of Sphere function (eq. 15.5):

$$f(x = [x_1, x_2]) = x_1^2 + x_2^2 \tag{15.5}$$

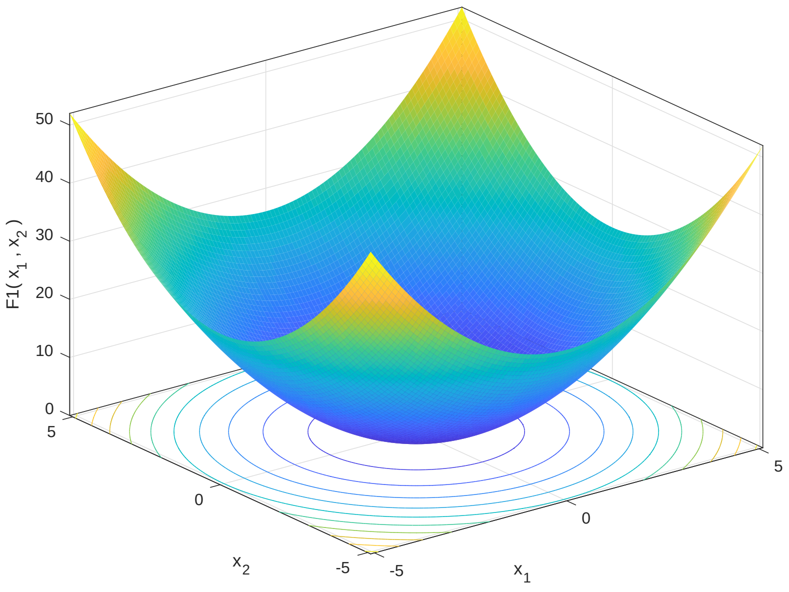

FIGURE 15.2
Two-dimensional Sphere function used in the experiments.

Function shape was demonstrated in Figure 15.2. The algorithm was executed with a population of $P = 10$ individuals. The maximum number of iterations k_{max} was set to 20. Default values of parameters $c_{max} = 1$ and $c_{min} = 0.00004$ were also employed, even though for some problems they were already demonstrated not to be the optimal ones [12].

The algorithm is starting with random dislocation of agents – obtained placement of population within the search space was demonstrated in Figure 15.3. It was obtained using a uniform random number generator with bounded output. Other methods could also be used [7].

In the subsequent iterations members of the swarm are being moved according to (15.1). The trajectory of the first grasshopper, initially located at [-4.24, -2.43] in the course of the algorithm's 20 iterations was demonstrated in Figure 15.4.

In the first iteration this swarm member is moved towards the rest of the population as was demonstrated in Table 15.1. It reports both the location of swarm members, their fitness, value of c, distance between the first grasshopper to the selected grasshopper p and contribution of each member p to the new location $[0.38; 1.07]$ of the first grasshopper. In other words "New location" for grasshopper i gives a position of grasshopper 1 caused by the

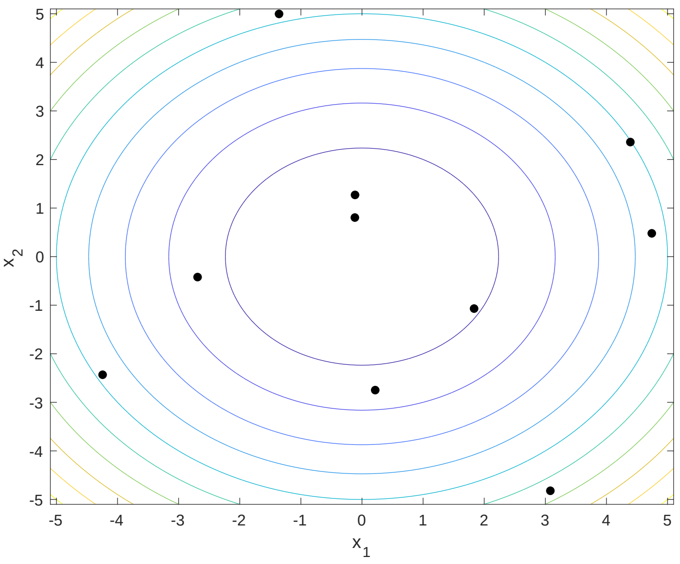

FIGURE 15.3
Initial configuration of the population.

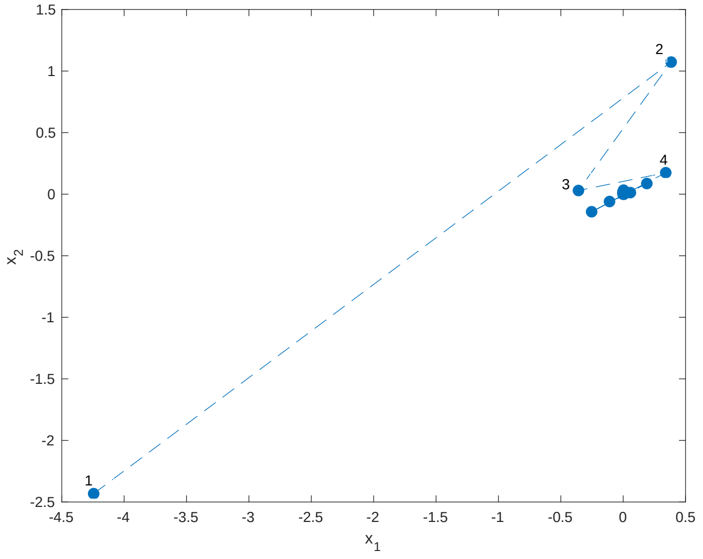

FIGURE 15.4
Trajectory of the first grasshopper.

TABLE 15.1
Location of swarm members in the first iteration and their impact on the new location of grasshopper no. 1 (with initial location [-4.24, -2.43]).

p	x_{p1}	x_{p2}	Fitness	c	$dist(x_1, x_p)$	$x_1 - x_p$	New location
1	-4.24	-2.43	23.93	0.95	0	[0,0]	—
2	3.08	-4.82	32.74		7.71	[-7.33, 2.39]	[0.08; 0.02]
3	4.39	2.36	24.84		9.87	[-8.64, -4.79]	[0.07; 0.04]
4	0.12	0.80	0.66		5.24	[-4.13, -3.23]	[0.07; 0.05]
5	-2.69	0.42	7.41		2.54	[-1.55, -2.01]	[0.04; 0.05]
6	4.74	0.48	22.72		9.45	[-8.98, -2.91]	[0.08; 0.03]
7	0.21	-2.75	7.60		4.47	[-4.46, 0.31]	[0.06; -0.01]
8	0.11	1.27	1.63		5.55	[-4.13, -3.70]	[0.07; 0.06]
9	1.83	-1.07	4.51		6.23	[-6.08, -1.36]	[0.03; 0.01]
10	-1.36	4.99	26.81		7.97	[-2.89, -7.42]	[0.03; 0.08]
					Sum:		[0.53; 0.28]

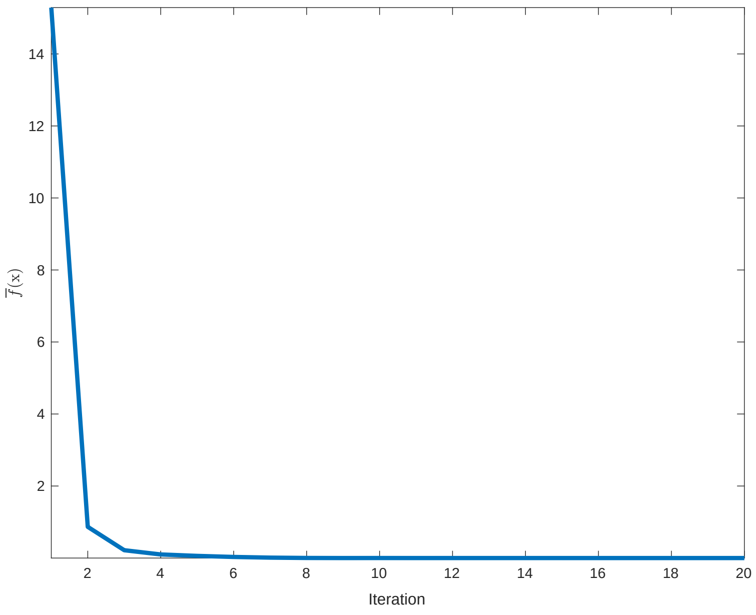

FIGURE 15.5
Mean population fitness during 20 iterations of GOA run for 2-dimensional Sphere function.

influence of grasshopper i). It is obtained by summing all entries in column "New location", scaling it by the value of c and adding the location of the best individual (grasshopper no. 4).

In the course of the algorithm the fitness of the population is rapidly improving. It is presented in Figure 15.5. The figure presents mean population fitness.

The final result of the optimization (with the fitness value of $8.38 \cdot 10^{-9}$) is very close to the global optimum – even though a very small number of individuals and algorithm's iterations were involved.

15.6 Conclusion

This chapter has studied the basic, off-the-shelf variant of the Grasshopper Optimization Algorithm. Besides the algorithm description we have provided a demonstration of its mechanics – using simple problem of minimizing Sphere function as our testbed. It can be seen that the Grasshopper Optimization Algorithm is a very effective optimizer, locating the minimum of the objective function within its first few iteration.

Finally, it is worth while to underline that the algorithm can be conveniently modified to include other factors, e.g. presence of multiple optimization criteria [8]. In addition to that it has already found many interesting applications and modifications to its standard scheme. They are covered in the forthcoming chapter [11].

References

1. Armadilo C++ library for linear algebra & scientific computing, url: http://arma.sourceforge.net/, accessed: June 1st, 2019.
2. N. Awad, M. Ali, J. Liang, B. Qu, and P. Suganthan. "Problem definitions and evaluation criteria for the cec 2017 special session and competition on single objective bound constrained real-parameter numerical optimization", Technical report, Nanyang Technological University Singapore, Nov 2016.
3. J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, and A. Abraham. "Inertia weight strategies in particle swarm optimization", 2011 Third World Congress on Nature and Biologically Inspired Computing, Oct 2011, pp. 633-640.

4. R.F. Chapman and A. Joern. "Biology of Grasshoppers", A Wiley-Interscience publication, Wiley, 1990.
5. A. E. Hassanien and E. Emary. "Swarm Intelligence: Principles, Advances, and Applications", CRC Press, 2018.
6. S. Lukasik and P.A. Kowalski. "Study of flower pollination algorithm for continuous optimization" in P. Angelov et al. (eds Intelligent Systems'2014. Advances in Intelligent Systems and Computing, vol. 322, pp. 451-459, Springer, 2015.
7. H. Maaranen, K. Miettinen, and A. Penttinen. "On initial populations of a genetic algorithm for continuous optimization problems", Journal of Global Optimization, vol. 37(2006), no. 3, 405.
8. S. Z. Mirjalili, S. Mirjalili, S. Saremi, H. Faris, and I. Aljarah. "Grasshopper optimization algorithm for multi-objective optimization problems", Applied Intelligence vol. 48, no. 4, pp. 805-820, 2018.
9. S. Saremi, S. Mirjalili, and A. Lewis. "Grasshopper optimisation algorithm: Theory and application", Advances in Engineering Software, vol. 105, pp. 30-47, 2017.
10. X. S. Yang. "Nature-Inspired Optimization Algorithms", Elsevier, London, 2014.
11. S. Łukasik. "Grasshopper optimization algorithm - modifications and exemplary applications," Swarm Intelligence Algorithms: Modifications and Applications (A. Slowik, ed.), CRC Press, 2020.
12. S. Łukasik, P. A. Kowalski, M. Charytanowicz, and P. Kulczycki. "Data clustering with grasshopper optimization algorithm", 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 71-74, 2017.