19

Modified Monarch Butterfly Optimization and Real-life Applications

Pushpendra Singh
Department of Electrical Engineering
Govt. Women Engineering College, Ajmer, India

Nand K. Meena
School of Engineering and Applied Science
Aston University, Birmingham, United Kingdom

Jin Yang
School of Engineering and Applied Science
Aston University, Birmingham, United Kingdom

CONTENTS

19.1 Introduction

Nowadays, the complexity of engineering optimization problems has increased due to involvement of multiple objectives and constraints. Knowing this fact, many researchers are contentiously working to find acceptable solutions of such problems. In this regard, various new optimization problems have been introduced in recent years with better problem solving capabilities such as the squirrel search algorithm (SSA) [1], water cycle algorithm (WCA) [2] etc. However, some of the researchers have upgraded the standard version of well-known optimization techniques and their inspirational improvements are demonstrated on benchmark functions, e.g., improved elephant herding optimization [3], novel monarch butterfly optimization with greedy strategy and self adaptive crossover operation (GCMBO) [4] etc.

The monarch butterfly optimization (MBO) technique is recently proposed by Gai-Ge Wang et al. [5], inspired by the monarch butterfly of the North American region which migrate from region 1 to region 2 and vice-versa, in particular months. The moth flutter upgrades its population by means of two reproduction operators, i.e., the migration operator and butterfly adjusting operators. In [5], the MBO was tested on various benchmark functions to demonstrate the promising problem solving abilities of the method. As identified in [4], the standard version of MBO shows some limitations regarding real-life optimization problems such as worst average and standard fitness values.

In order to overcome such limitations, Gai-Ge Wang et al. proposed a new version of MBO, known as GCMBO. The amendment is done by employing two strategies in the standard version of MBO. First is self-adaptive crossover (SAC), an improved crossover operator, integrated with the butterfly adjusting operator in the standard version of the method. This is proposed to increase the diversity of the monarch flutter during the later searching stage. This SAC operator can also utilize the information of the whole flutter. The second advancement is introduced by incorporating a greedy strategy with migration and butterfly adjusting operators. This operator is designed to accept the monarch individual which shows better fitness than its parents. This operator also helps in accelerating the convergence of the method. Rest condition and operators remain the same as the basic version.

This chapter is organised in five sections. Section 19.1 presents the introduction of this chapter and then the brief description of the basic version of MBO is presented in Section 19.2. In Section 19.3, some suggested improvements in MBO are discussed followed by its implementation on a real-life complex optimization problem of distributed generation (DG) allocations in distribution systems. The conclusions of this chapter are presented in Section 19.7.

19.2 Monarch butterfly optimization

In the standard version of MBO, some sets of rules are defined in order to find the global or near global solution of an optimization problem. The migration of a monarch butterfly flutter is described below by help of some basic rules [5].

- The whole population of the monarch flutter are found in region 1 and 2.
- Every offspring, produced by the butterfly is in region 1 or region 2.
- The size of the monarch flutter will remain unchanged during the optimization process.
- A certain number of fittest butterfly flutter individuals can't be updated by butterfly adjustment operators or migration operators.

19.2.1 Migration operator

The population of the monarch flutter can be bifurcated as, sub-population 1 (P_{n1}) and sub-population 2 (P_{n2}) on the basis of their stay in region 1 and region 2. The (P_{n1}) is equal to the ceiling value of $P_r \times P_n$. Whereas, P_{n2} is expressed as P_n - P_{n1}, where, P_r is the ratio of monarch flutter halting in region 1 and P_n is the absolute number of monarch butterfly population. The migration process is mathematically expressed as

$$z_{j,l}^{i+1} = z_{r_1,l}^{i} \tag{19.1}$$

where, $z_{j,l}^{i+1}$ is the lth element of z_j at the generation level $i+1$ and j is the position of the monarch butterfly. Likewise, $z_{r_1,l}^{i}$ is the lth element of z_{r_1} which is a newly generated position of monarch butterfly r_1 and i is the current stage of generation. Here r_1 is a butterfly randomly selected from the monarch butterflies of sub-population 1. When, $r \leq P_r$, then the component of l in the newly born monarch butterfly is produced by (19.1). The value of r is determined as

$$r = rnd \ * \ \psi \tag{19.2}$$

where, rnd is a random number drawn from a uniform distribution and ψ is a constant considered to be 1.2 for the 12 month period. On other hand, if $r > P_r$ then newborn monarch butterflies will be produced as

$$z_{j,l}^{i+1} = z_{r_2,l}^{i} \tag{19.3}$$

The r_2 is a position of monarch butterfly offspring, randomly obtained from the monarch butterflies of sub-population 2. It is to be noted that by adjusting the value of P_r, the direction of the migration operator can be balanced. For example, if the value of P_r is larger or bigger then more members of

monarch flutter P_{n1} will participate. On other side, if the value of P_r is smaller then more chances will be given to the members of the monarch flutter from sub-population 2, P_{n2}. Therefore, P_{n2}, the sub-population of region 2 is an important factor in producing new monarch butterflies. In this chapter, the value of P_r is considered to be 5/12.

19.2.2 Butterfly adjusting operator

For all the elements in the monarch flutter in k, $rand \leq P_r$ is updated as

$$z_{k,l}^{i+1} = z_{best,l}^{i} \tag{19.4}$$

where, $z_{k,l}^{i+1}$ represents the lth element of z_k in generation $i+1$. Similarly, $z_{best,l}^{i}$ is the lth element of the fittest monarch butterfly z_{best}. However, if $P_r \leq rand$ then updated as follows.

$$z_{k,l}^{i+1} = z_{r_3,l}^{i} \tag{19.5}$$

The $z_{r_3,l}^{i}$ represents lth element of z_{r_3}. For $r_3 \in \{1, 2, ..., P_{n2}\}$. Under this condition, if $rand > BAR$ then it is further updated as

$$z_{k,l}^{i+1} = z_{k,l}^{i+1} + \alpha \ * (dz_l - 0.5) \tag{19.6}$$

where, dz is the Levy flight calculated walk step size of monarch butterfly k, determined as

$$dz = Levy(z_k^t) \tag{19.7}$$

In (19.6), α is the weighting factor calculated as

$$\alpha = S_{max}/i^2 \tag{19.8}$$

where, S_{max} is the maximum walk step taken by a butterfly in a single move step.

19.3 Modified monarch butterfly optimization method

MBO is a robust nature-inspired optimization technique tested, for promising results, on various benchmark systems. However, it may fail to obtain effective results of standard and average fitness value for some benchmark function [4]. To overcome this deficiency, some strategic improvements are incorporated in basic MBO. The improved MBO called as GCMBO is discussed in the following sections.

19.3.1 Modified migration operator

In the standard version of MBO, all new offspring are accepted and transferred to the next generation while in the upgraded version of MBO, a greedy strategy is adopted. In this strategy, the new butterfly individuals with better fitness value than their respective parents are only moved to the next generation; otherwise the new butterflies are rejected and their parents move to the next generation. In this strategy, the position of the jth butterfly updated in (19.1) is again modified as

$$z_{j,new}^{i+1} = \begin{cases} z_{j,l}^{i+1}; & \text{if } f(z_{j,l}^{i+1}) < f(z_{r_1/r_2,l}^{i}) \\ z_{r_1/r_2,l}^{i}; & \text{else} \end{cases} \tag{19.9}$$

where, $z_{j,new}^{i+1}$ is a newly produced offspring of butterfly individual j moving to next generation, i+1.

19.3.2 Modified butterfly adjustment operator

In this modification, the monarch butterfly individual of sub-population 2, P_{n2} is updated as per (19.4)–(19.8). In this modification, an adaptive crossover and greedy strategy operators are discussed. The generated monarch butterfly individuals updated in (19.4)–(19.8) are referred to as z_{k1}^{i+1}. To utilize the complete benefits of monarch flutter information, a crossover operator is implemented in the butterfly adjustment operators, expressed as

$$z_{k2}^{i+1} = z_{k1}^{i+1} \times (1 - C_r) + z_k^i \times C_r \tag{19.10}$$

where, z_{k2}^{i+1} is newly produced butterfly offspring by deploying z_{k1}^{i+1}, z_k^i and crossover rate C_r is calculated as

$$C_r = 0.8 + 0.2 \times \frac{f(z_k^i) - f(z_{best})}{f(z_{worst}) - f(z_{best})} \tag{19.11}$$

where $f(z_k^i)$ represents the fitness value of butterfly k from sub-population 2 P_{n2}. $f(z_{worst})$ and$f(z_{best})$ are the fitnesses of z_{worst} and z_{best} monarch butterfly individuals. It is observed that the value obtained for C_r, from (19.11) is in the range of [0.2 0.8]. The new butterfly individual is determined by deploying the greedy strategy as

$$z_{k,new}^{i+1} = \begin{cases} z_{k1}^{i+1}; & \text{if } f(z_{k1}^{i+1}) < f(z_{k2}^{i}) \\ z_{k2}^{i+1}; & \text{if } f(z_{k2}^{i+1}) < f(z_{k1}^{i}) \end{cases} \tag{19.12}$$

where, $f(z_{k1}^{i+1})$ and $f(z_{k2}^{i+1})$ are fitnesses of the butterflies z_{k1}^{i+1} and z_{k2}^{i+1} respectively.

19.4 Algorithm of modified MBO

In this section, the algorithm of GCMBO, discussed in previous sections, is point-wise described here. The flow chart of GCMBO is presented in Fig. 19.1.

Modified monarch butterfly optimization

Step-1: Initialize random but feasible population of P_n monarch butterflies and set the values of required parameters, constants and maximum number of iterations, i.e., $S_{Max.}$, BAR, t, etc.

Step-2: calculate the fitness values of each monarch butterfly generated in previous step

Step-3: start the iteration $t = 1$

Step-4: sort all monarch butterfly individuals with respect to their fitness values and then divide these monarch butterflies into two populations, P_{n1} and P_{n2}, for the region 1 and region 2 respectively

Step-5: the sub-population P_{n1} of region 1 generates the new sub-population 1 by using the modified migration operator expressed in (19.9). Similarly, the sub-population P_{n2} of region 2 is updated by the modified adjusting operator succeeded by adaptive crossover and greedy strategy operators in (19.10)–(19.12).

Step-6: a correction algorithm may be applied to correct the infeasible individuals, if any

Step-7: repeat steps 4 to 6 until convergence criteria or maximum number of iterations is reached

Step-8: Print the best solution.

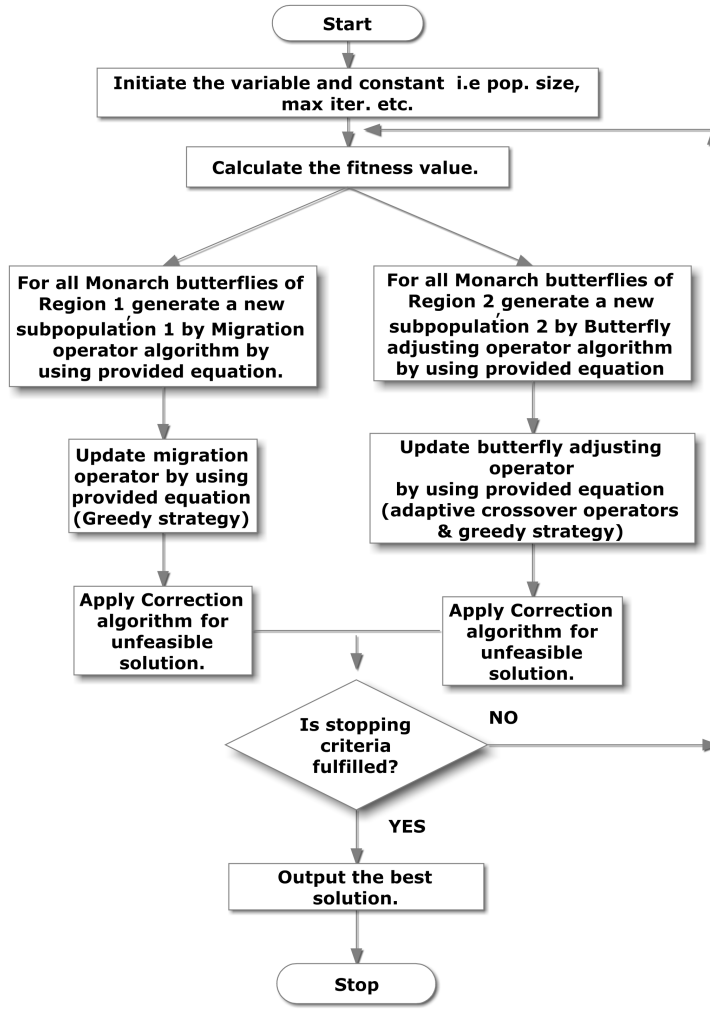

FIGURE 19.1
Flow chart of monarch butterfly optimization with greedy strategy and self adaptive crossover operation.

19.5 Matlab source-code of GCMBO

For basic understanding of GCMBO, the Matlab source-code is presented in Listing 19.1.

```
LB=[];UB=[]; %set upper & lower limits of each variable
iter=0;Max_iter=10;     %initial & maximum number of iterations
Pn = 8 ;      % population size
nFlutr =length(UB);  % number of variables
nBF1=ceil(Pn*5/12); % butterfly of region1 & 2
nBF2=Pn - nBF1;
Lnd1=zeros(nBF1,nFlutr); %flutter at region 1 & 2
Lnd2=zeros(nBF2,nFlutr);
%=================MSO Const=============================
Stepmax=1.0; partt=5/12; period=1.5;
Keep=2; partition=5/12; BAR=partition;
for j=1:Pn % initialization %rand variables
for k= 1:nFlutr
x(j,k)=LB(k)+rand*(UB(k)-LB(k));
end
x(j,:)=Correction_Algo(x(j,:),LB,UB); % Apply Correction
[fitness(j)]=OF(x(j,:)); %Fitness_Function(x(j,:));
end
best_fit=min(fitness); worst_fit=max(fitness); nn_best=find(min(
    fitness)==fitness);
nn_worst=find(max(fitness)==fitness); x_best=x(nn_best(1),:);
xx=x_best; x_worst=x(nn_worst(1),:);
while iter <=Max_iter
iter = iter + 1;COMB_POP=[x, fitness'];
x_srt=sortrows(COMB_POP(:,nFlutr));
x_keep = x_srt(1:Keep,:); % Elitism Strategy
x_srt((Pn-1):Pn,:)=x_keep; x=COMB_POP(:,1:nFlutr);
fitness=(COMB_POP(:,nFlutr+1))';
for pii=1:Pn %%Divide the whole population into two parts
if pii<= nBF1
pop1(pii,:)=x(pii,:);
else
pop2(pii,:)=x(pii,:);
end
end
for j=1:nBF1 %migration operator
for k=1:nFlutr
rr1=rand*1.2;
if rr1<=partition
rr2=round(nBF1*rand + 0.5);
Lnd1(j,k)=pop1(rr2,k);
else
rr3=round(nBF2*rand + 0.5);
Lnd1(j,k)=pop2(rr3,k);
end
end
x_x(2,:)=Lnd1(j,:); %greedy strategy
for ryn=1:2
x_x(ryn,:)=Correction_Algo(x_x(ryn,:), LB, UB);
arn(ryn,:)=OF(x_x(ryn,:)); %Fitness_Function(x_new(j,:));
end
x_x(2,:);arn(2,:);
if arn(1)<arn(2)
Lnd1(j,:)=x_x(1,:);
else
Lnd1(j,:)=x_x(2,:);
end
end
for j=nBF2 %butterfly adjustment operator%%%%%%
```

```
scale=Stepmax/(iter)^2; stepsize=ceil(exprnd(2*Max_iter,1,1)); deltaX=
    LevyFlight(stepsize,nFlutr);
for k=1:nFlutr
if (rand>=partition)
Lnd2(j,k)=x_best(1,1);
else
rr4=round(nBF2*rand + 0.5);        Lnd2(j,ci)=pop2(rr4,k);
if (rand> BAR)
Lnd2(j,k)=Lnd2(j,k)+ scale*(deltaX(k)-0.5);
end
end
end
%%%%%%%%% Self adaptive crossover operator %%%%%%%%%%
Lnd2(j,:)=Correction_Algo(Lnd2(j,:), LB, UB);
x_L(1,:)=OF(Lnd2(j,:));    x_B(1,:) = OF(x_best(1,:));
x_W(1,:)=OF(x_worst(1,:));  %worst & best
C_rate=0.8+0.2.*((x_L(1,:) - x_B(1,:))/(x_W(1,:) - x_B(1,:)));
Cr= rand(nFlutr,1) < C_rate;%%%Cross Rate%%%
new_Lnd2=Lnd2(j,k).*(1-Cr') + (x(1,:).*Cr');
x_x_L1(1,:)=Lnd2(j,:);x_x_L2(1,:)=new_Lnd2(1,:);
x_x_L1(1,:)=Correction_Algo(x_x_L1(1,:), LB, UB);
x_x_L2(1,:)=Correction_Algo(x_x_L2(1,:), LB, UB);
fx_x_L1(1,:)=OF(x_x_L1(1,:)); fx_x_L2(1,:) = OF(x_x_L2(1,:));
if fx_x_L1(1,:) <fx_x_L2(1,:)  %%%%%%%%greedy strategy%%%%%%%%%%
Lnd2(j,:)=x_x_L1(1,:);
else
Lnd2(j,:)=x_x_L2(1,:);
end
end
x_new=[Lnd1;Lnd2];        % fitness calculation
for j=1:Pn
x_new(j,:)=Correction_Algo(x_new(j,:), LB, UB);
[fitness(j)]=OF(x_new(j,:));
end
best_fit1=min(fitness); nn_best=find(min(fitness)==fitness);
x_best1=x(nn_best(1),:);
if best_fit1<best_fit
best_fit=best_fit1;     x_best=x_best1;
end
best_fit; best=x_best; x=x_new;
m(iter)=mean(fitness);b(iter)=best_fit;%storing mean& iter
end
best_fit; %Best Fitness
best=x_best; %Best sting obtain
```

Listing 19.1
Source-code of GCMBO in Matlab.

19.6 Application of GCMBO for optimal allocation of distributed generations

19.6.1 Problem statement

The optimal allocation of distributed generations (DGs) in a distribution system is found to be a complex mixed-integer, non-linear, and non-convex optimization problem [3], which therefore needs an effective optimization method. In this chapter, a benchmark 33-bus distribution system is considered [6]. The

objective is to determine the optimal sites (nodes) and sizes (DG capacities) of 3 DGs for minimum real power loss, P_{Loss} of 33-bus distribution system. The expression of real power loss is expressed as

$$P_{Loss} = \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_{ij}(P_i P_j + Q_i Q_j) + \beta_{ij}(Q_i P_j - P_i Q_j) \tag{19.13}$$

$$\alpha_{ij} = \frac{r_{ij} \cos(\delta_i - \delta_j)}{V_i V_j} \tag{19.14}$$

$$\beta_{ij} = \frac{r_{ij} \sin(\delta_i - \delta_j)}{V_i V_j} \tag{19.15}$$

s. t.

1. *Nodal power balance constraints*

$$P_i = P_{G_i} - P_{D_i} = V_i \sum_{j=1}^{N} V_j Y_{ij} \cos(\phi_{ij} + \delta_j - \delta_i) \quad \forall\, i \tag{19.16}$$

$$Q_i = Q_{G_i} - Q_{D_i} = -V_i \sum_{j=1}^{N} V_j Y_{ij} \sin(\phi_{ij} + \delta_j - \delta_i) \quad \forall\, i \tag{19.17}$$

2. *Voltage limits constraints*

$$V^{Min} \leq V_i \leq V^{Max} \qquad \forall\, i \tag{19.18}$$

3. *Feeder capacity limits constraint*

$$0 \leq I_{ij} \leq I_{ij}^{Max} \qquad \forall\, i, j \tag{19.19}$$

4. *Maximum penetration limit of each bus*

$$0 \leq P_{DG_i} \leq P_{DG}^{Max} \qquad \forall\, i \tag{19.20}$$

5. *Maximum DG penetration limits of the system*

$$0 \leq \sum_{i=1}^{N} \sigma_i P_{DG_i} \leq P^{Peak} \qquad \forall\, i \tag{19.21}$$

where, P_i, Q_i, P_{G_i}, Q_{G_i}, P_{D_i}, Q_{D_i}, σ_i, V_i, and δ_i are representing the real power injection, reactive power injection, real power generation, reactive power generation, real power load, reactive power load, binary decision variable of DG installation, magnitude of node voltage and voltage angles of node i respectively. Furthermore, r_{ij}, ϕ_{ij}, Y_{ij}, I_{ij}, I_{ij}^{Max}, N, P_{DG}^{Max}, P^{Peak} denote the branch resistance, impedance angle, Y-Bus element, feeder current, maximum allowed current limit of the branch connected between node i and j, and number of system nodes, allowed DG capacity limit at a node, and in the system respectively.

19.6.2 Optimization framework for optimal DG allocation

In this section, we discuss the implementation of GCMBO on optimal DG allocation problems. As stated in the problem statement, we need to determine the sites and sizes of three DGs in 33-bus distribution system. Therefore, the number of variables used in a butterfly individual will be 6, i.e., 3 sites + 3 sizes. The lower and upper limits of the first three variables will be [1 33]; whereas, it will be [0 2000] for the remaining 3 variables, if $P_{DG}^{Max} = 2000$kW and $P^{Peak} = 6000$kW. Fig. 19.2 presents the contour plot of power loss of a 33-bus distribution system with respect to nodes and DG sizes up to 2000kW.

It may be observed that DG placement is a non-convex and non-linear optimization problem which has multiple solutions. The solution variables are mixed integers as nodes are integer whereas, DG sizes are non-integers. Now, the DG allocation problem discussed in the previous section is solved by using GCMBO. The Matlab source-code of objective function $OF(.)$, i.e., power loss minimization, is presented in Listing 19.2.

The optimal nodes and sizes obtained are

DG-1: 811 kW at node number 13

DG-2: 1487 kW at node number 24

DG-3: 1001 kW at node number 30

The minimum values of the power loss obtained is, $P_{Loss} = 75.7282$ kW.

The convergence characteristics of MBO and GCMBO are compared and presented in Fig. 19.3. The figure shows that the suggested improvements significantly improve the solution searching abilities of the method. The GCMBO continuously searches solutions in each iteration. The other performance parameters of the MBO and GCMBO algorithms, such as best fitness, worst fitness, mean fitness and standard deviation, for 50 independent trials, are also compared and then presented in Table 19.1. The table shows that GCMBO outperformed in the global optimal solution search.

```
function [OBJ]=OF(X)
X=floor(X);
% System data
Linedata=[]; Busdata=[];
d=1; %slack bus
Niter=4; % No. of iterations
Base_Kv=12.66; % Base KV
Base_MVA=100; % 100MVA Base;
Z_Base=(((Base_Kv)^2)/(Base_MVA)); fb=Linedata(:,1);
tb=Linedata(:,2); N=max(max(fb),max(tb));
Nb=length(fb); R=Linedata(:,3); X=Linedata(:,4);
Z_Line=complex(R,X)/Z_Base; PL=Busdata(:,7);
QL=Busdata(:,8); SL=((complex(PL,QL))/(1000*Base_MVA));
PG=Busdata(:,5); QG=Busdata(:,6);
%Find terminal and internal buses
Terminalbuses=zeros(N,1); Intermediatebuses=zeros(N,1);
for i=1:N-1
Terminalbuses(i,1)=0; Intermediatebuses(i,1)=0;
end
do=0;in=0;
for i=1:Nb
for j=1:Nb
if (i~=j)&&(tb(i)==fb(j))
do=do+1;
```

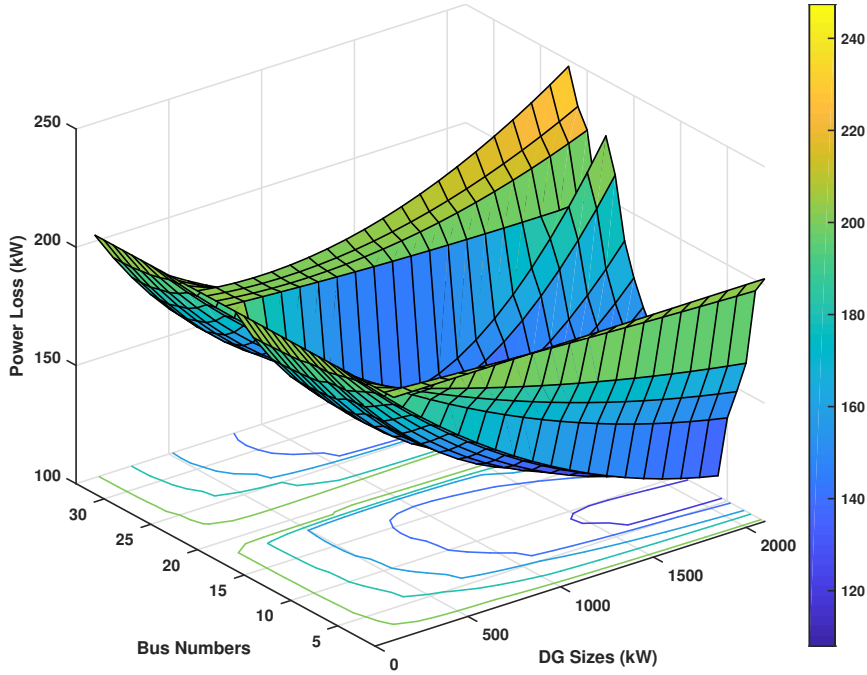

FIGURE 19.2
Contour plot of power loss with respect to nodes and DG sizes.

```
end
end
if do==0
in=in+1;
Terminalbuses(in,1)=tb(i);
else
in=in+1;
Intermediatebuses(in,1)=tb(i);
end
do=0;
end
T1=Terminalbuses; In=Intermediatebuses;
T2=find(T1); In2=flipud(find(In));
PG=zeros(N,1); QG=zeros(N,1);
%DG placement from X
for opj=1:length(X)/2
PG(X(opj))=PG(X(opj))+X(opj+length(X)/2);
end
SG=((complex(PG,QG))/(1000*Base_MVA)); Sinj=SG - SL;
V=Busdata(:,3); Ang=Busdata(:,4);
%iteration starts.....................
for iter=1:Niter
% Calculation of current injection Matrix
I_injection1=zeros(N,N);
for i=1:N
I_injection1(i,i)=conj((Sinj(i))/(V(i)));%Diagonal elements of
    injection matrix
end
%Current Calculations in Terminal Buses.
```

```
I_injection2=zeros(N,N);
for i=1:length(T2)
I_injection2(fb(T2(i)),tb(T2(i)))=-I_injection1(tb(T2(i)),tb(T2(i)));% Current Calculations from Sub Terminal to Terminal Buses.
end
% Current Calculations between Intermediate & Terminal Buses.
for j=1:length(In2)
I_injection2(fb(In2(j)),tb(In2(j)))=-I_injection1(tb(In2(j)),tb(In2(j)))+sum(I_injection2(tb(In2(j)),:));
I_injection2(tb(In2(j)),fb(In2(j)))=-I_injection2(fb(In2(j)),tb(In2(j)));
end
for i=1:length(T2)
I_injection2(tb(T2(i)),fb(T2(i)))=-I_injection2(fb(T2(i)),tb(T2(i)));% Current Calculations from Sub Terminal to Terminal Buses.
end
%voltage update using ohms law.
V(d)=1.0;
for k=1:Nb
V(tb(k))=(V(fb(k)))-Z_Line(k)*(I_injection2(fb(k),tb(k)));
end
V_mag=abs(V);
end
%Branch Power Flows in the system.
S_Flow=zeros(N,N);
%Power Loss calculation in the system..
S_Loss = zeros(Nb,1);
for m = 1:Nb
S_Flow(fb(m),tb(m)) = (V(fb(m))*conj(I_injection2(fb(m),tb(m))))*1000;
S_Flow(tb(m),fb(m)) = -(V(tb(m))*conj(I_injection2(fb(m),tb(m))))*1000;
S_Loss(m) = S_Flow(fb(m),tb(m)) + S_Flow(tb(m),fb(m));
end
S_Loss=S_Loss*Base_MVA;
P_Loss = real(S_Loss);
OBJ=sum(P_Loss);
```

Listing 19.2
Source-code of objective function OF() for power loss minimization.

TABLE 19.1
Performance comparison of MBO and GCMBO for 50 independent trials.

Method	Best fitness	Worst fitness	Mean fitness	Standard deviation	CPU time(s)
MBO	0.0753	0.0880	0.0827	0.0034	7.04
GCMBO	0.0734	0.0793	0.0741	0.0023	7.06

19.7 Conclusion

In this chapter, a new nature inspired meta-heuristic optimization technique is developed. It is inspired by the migration behaviour of monarch butterflies found in the continent of North America. It is found that MBO provides

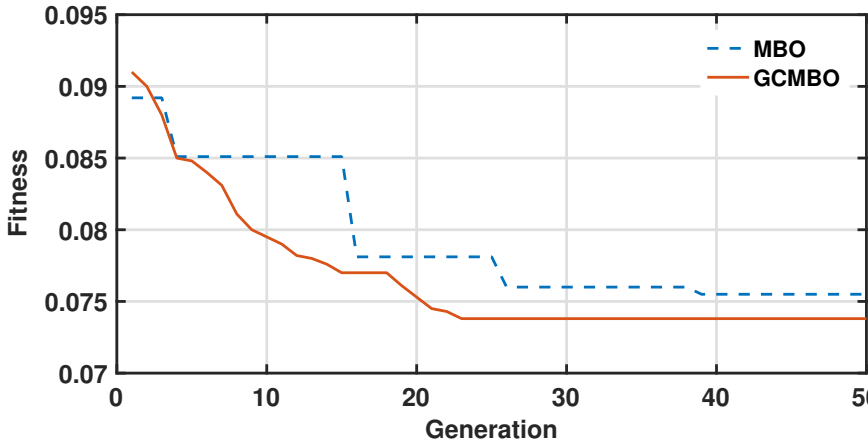

FIGURE 19.3
Best Fitness comparison of MBO & GCMBO.

promising results for various benchmark functions; however, it generates poor standard deviation and worst mean fitness for some bench functions. In order to improve the performance of the standard version of MBO, some modifications are suggested in migration operators, known as GCMBO. In GCMBO, a greedy and self adaptive crossover operator is injected into the migration and butterfly operators. A real-life DG allocation problem of power loss minimization is then solved and it is demonstrated that the greedy operators significantly accelerated the convergence speed and efficiency.

Acknowledgement

This work was supported by the Engineering and Physical Sciences Research Council (EPSRC) of United Kingdom (Reference Nos.: EP/R001456/1 and EP/S001778/1).

References

1. M. Jain, V. Singh and A. Rani. "A novel nature-inspired algorithm for optimization: Squirrel search algorithm". *Swarm and Evolutionary Computation*, vol. 44, 2019, pp.148-175.

2. Eskandar, H., Sadollah, A., Bahreininejad, A. and Hamdi, M. Water cycle algorithm–A novel metaheuristic optimization method for solving constrained engineering optimization problems. *Computers & Structures*, 110, pp.151-166, 2012.
3. N. K. Meena, S. Parashar, A. Swarnkar, N. Gupta and K. R. Niazi. "Improved elephant herding optimization for multiobjective DER accommodation in distribution systems". *IEEE Transactions on Industrial Informatics*, vol. 14(3), 2018, pp.1029-1039.
4. Wang, G.G., Deb, S., Zhao, X. and Cui, Z. A new monarch butterfly optimization with an improved crossover operator. *Operational Research*, 18(3), pp.731-755, 2018.
5. G G. Wang, S. Deb and Z. Cui. "Monarch butterfly optimization". *Neural Comput Appl.*, 2015, pp. 1433-3058. doi:10.1007/s00521-015-1923-y.
6. M. E. Baran and F. F. Wu. "Network reconfiguration in distribution systems for loss reduction and load balancing". *IEEE Transactions on Power Delivery*, vol. 4(2), 1989, pp. 1401-1407.