Online Appendix for the Paper "Sim-heuristic Algorithm for Solving Bi-objective Stochastic Optimization Problems"

Shih-Cheng Horng
schong@cyut.edu.tw

The LWOO approach was implemented in MATLAB R2024a on Windows 11 and executed on a Laptop with an Intel Core i7-8565U at 1.8 GHz and 16 GB of RAM.

(1) LWOO.m

```
N =100; % Number of leaves
MaxIter =500; % Maximum number of iterations
fobj =' simevents_fitness.m ';
LB =[2,2,2,1,1,1,5,5];
UB =[6,6,6,36,36,36,20,20];
Dim=8;
nd =40;    % number of candidate designs
L0=20;    % initial number of simulation replications
add_budget=10; % the additional simulation budget
CB= 3738;    % allowable simulation budgets
KAN();    %Construct the KAN model
function [Former_N_BestPos, Former_N_BestScore] = RLWO(N, MaxIter, LB, UB, Dim, fobj);
function [BestPos, BestScore] = AOCBA(nd, L0, add_budget);
```

(2) RLWO.m

```
%% Refined leaf in wind optimization algorithm %%
function [Former_N_BestPos, Former_N_BestScore] = RLWO(N, MaxIter, LB, UB, Dim, fobj)

 % Refined leaf in wind optimization algorithm
 % N              : Population size
 % MaxIter    : Maximum iterations
 % LB, UB      : Lower and upper bounds
 % Dim           : Problem dimension
 % fobj         : Objective function handle
```

```
%% Parameters setting
C_min=0.001;
C_max=0.8;
P1_min=0.1;
P1_max=0.3;
P2_min=0.05;
P2_max=0.2;
P3_min=0.05;
P3_max=0.2;

%% Initialization
if length(UB)==1
    X = rand(N,Dim).*(UB-LB)+LB;
else
    X = zeros(N,Dim);
    for i=1:Dim
        X(:,i)=rand(N,1).*(UB(i)-LB(i))+LB(i);
    end
end
Fitness = zeros(N,1);
for i=1:N
    Fitness(i)=fobj(X(i,:));
end
[BestScore,idx] = min(Fitness);
BestPos = X(idx,:);

% --- 2. Optimization Loop ---
for t=1:MaxIter
    %% Parameters
    C=C_min+(C_max-C_min)*exp(log(C_min/C_max)* t/ maxIter);
    P1=P1_min+ (P1_max-P1_min) * (1- exp(P1_max/P1_min * t/ maxIter)) ;
    P2=P2_max *exp(-sqrt(P2_max/P2_min)* t/ maxIter);
    P3=P3_min+ (P3_max-P3_min) * exp(1- t/ maxIter) ;

    % Simulates transitioning from stormy exploration to calm exploitation
    for i=1:N
    Xnew = X(i,:);
```

```
if rand < P1
    %% ========================================
    %% Breeze-driven Leaf Strategy (Exploration)
    %% ========================================
    k = randperm(N,2);
    X1 = X(k(1),:);
    X2 = X(k(2),:);
    d = X1 - X2;
    % Linear movement
    Xnew = X(i,:) + C * rand(1,Dim).*d;

    % Spiral movement
    for j=1:Dim
        if rand < P2
            phi = 2*pi*rand;
            spiralStep = C * abs(d(j)) * cos(phi);
            Xnew(j)=Xnew(j)+spiralStep;
        end
    end
else
    %% ========================================
    %% Strong Wind Strategy (Exploitation)
    %% ========================================
    r1 = rand;
    r2 = rand;
    Dbest = abs(r2*BestPos - X(i,:));
    Xnew = BestPos – P3*Dbest;
    % Local perturbation
    j = randi(Dim);
    Xnew(j)=Xnew(j)+0.01*randn;
end

%% Boundary control
Xnew=max(Xnew,LB);
Xnew=min(Xnew,UB);

%% Evaluation
Fnew=fobj(Xnew);
```

```
            if Fnew < Fitness(i)
                X(i,:)=Xnew;
                Fitness(i)=Fnew;
            end

            %% Update global best
            if Fitness(i) < BestScore
                BestScore = Fitness(i);
                BestPos = X(i,:);
            end
        end
    end
    end
```

(3) simevents_fitness.m

```
    function [throughput] =simevents_fitness(x)
        % x is a vector of design variables

        % 1. Map decision variables to SimEvents block parameters
        machine= x(1:3);
        task= x(4:6);
        bufferSize = x(7:8);

        % 2. Open/load production system model
        modelName = 'ProductionSystemModel';
        load_system(modelName);

        % 3. Set parameters in the workspace or directly on blocks
        set_param([modelName, '/Entity Queue'], 'Capacity', num2str(bufferSize));
        set_param([modelName, '/Entity Server'], 'ServiceTime', num2str(processingTime));

        % 4. Simulate the model and capture Data Inspector/Scope logs
        simOut = sim(modelName, 'SrcWorkspace', 'Current', 'SaveState', 'on', 'StateSaveName',
    'xout');

        % 5. Extract system metrics
        throughputObj = simOut.logsout.find('Name', 'SystemThroughput');
        throughput = max(throughputObj.Values.Data);
```

```
    % Clean up workspace to prevent memory leaks during optimization loops
    bdclose(modelName);
end
```

(4) KAN.m

```
function KAN()
    % Clear workspace and command window
    clear; clc; close all;

    %%1. Load Training Data
    load training;
    numSamples = 300;
    idx = randperm(numSamples);
    trainIdx = idx(1:floor(numSamples));

    X_train = X(trainIdx, :);
    Y_train = Y(trainIdx, :);

    %% 2. Network Configuration
    % Network architecture layout: [Input_Dim, Hidden_Dim, Output_Dim]
    layers = [8, 17, 1];
    polyDegree = 3;             % Degree of Chebyshev polynomial expansions
    learningRate = 0.01;
    epochs = 150;

    %% 3. Initialize KAN Parameters
    % For each layer pair, parameters are dimensions: [Out_Dim, In_Dim, Params]
    % Base weights (w_base) and Polynomial coefficients (w_poly) live on edges
    W1_base = randn(layers(2), layers(1)) * 0.1;
    W1_poly = randn(layers(2), layers(1), polyDegree + 1) * 0.1;
    W2_base = randn(layers(3), layers(2)) * 0.1;
    W2_poly = randn(layers(3), layers(2), polyDegree + 1) * 0.1;

    %% 4. Training Loop (Stochastic Gradient Descent)
    lossHistory = zeros(epochs, 1);
    for epoch = 1:epochs
        epochLoss = 0;
```

```
                % Process data sample by sample (Stochastic approach)
        for i = 1:size(X_train, 1)
            x_sample = X_train(i, :)'; % Column vector [In_Dim x 1]
            y_target = Y_train(i, :);      % Scalar target

            %% --- Forward Pass ---
            % --- Layer 1 ---
            [phi1, dphi1_dbase, dphi1_dpoly] = evaluate_kan_layer(x_sample, W1_base,
W1_poly, polyDegree);
            out1 = sum(phi1, 2); % Sum activations flowing into hidden nodes

            % --- Layer 2 ---
            [phi2, dphi2_dbase, dphi2_dpoly] = evaluate_kan_layer(out1, W2_base, W2_poly,
polyDegree);
            y_pred = sum(phi2, 2); % Final network prediction

            %% --- Loss and Backpropagation ---
            err = y_pred - y_target;
            epochLoss = epochLoss + (err^2);

            % Gradient at output node
            dL_dy = 2 * err;

            % Gradients for Layer 2 parameters
            dW2_base_grad = zeros(size(W2_base));
            dW2_poly_grad = zeros(size(W2_poly));
            dL_dout1 = zeros(size(out1)); % Error backpropagated to hidden layer

            for j = 1:layers(3)             % Output nodes
                for k = 1:layers(2)      % Hidden nodes
                    dW2_base_grad(j,k) = dL_dy * dphi2_dbase(j,k);
                    dW2_poly_grad(j,k,:) = dL_dy * squeeze(dphi2_dpoly(j,k,:));

                    % Chain rule component passing through edge activation derivative
                    % Here we approximate derivative w.r.t edge input node using base term
                    dL_dout1(k) = dL_dout1(k) + dL_dy * W2_base(j,k);
                end
            end
```

```
                % Gradients for Layer 1 parameters
                dW1_base_grad = zeros(size(W1_base));
                dW1_poly_grad = zeros(size(W1_poly));

                for k = 1:layers(2)             % Hidden nodes
                    for m = 1:layers(1)      % Input nodes
                        dW1_base_grad(k,m) = dL_dout1(k) * dphi1_dbase(k,m);
                        dW1_poly_grad(k,m,:) = dL_dout1(k) * squeeze(dphi1_dpoly(k,m,:));
                    end
                end

                %% --- Update Weights (Gradient Descent) ---
                W2_base = W2_base - learningRate * dW2_base_grad;
                W2_poly = W2_poly - learningRate * dW2_poly_grad;
                W1_base = W1_base - learningRate * dW1_base_grad;
                W1_poly = W1_poly - learningRate * dW1_poly_grad;
            end

            lossHistory(epoch) = epochLoss / size(X_train, 1);
            if mod(epoch, 25) == 0 || epoch == 1
                fprintf('Epoch %d/%d - Mean Squared Error: %.5f\n', epoch, epochs,
lossHistory(epoch));
            end
        end
    end
```

(5) AOCBA.m

```
    function [BestPos, BestScore] = AOCBA(nd, L0, add_budget, CB)
    %s_mean[i]: sample mean of design i, i=1,...,nd
    %s_var[i]: sample variance of design i, i=1,...,nd
    %nd: the number of candidate designs
    %n0[i]: initial number of simulation replications of design i, i=1,...nd
    %add_budget; the additional simulation budget
    %an[i]: additional number of simulation replications assigned to design i, i=1,...,nd
    %CB: the allowable simulation budgets
    % ------------------defining variables and initial setting----------
```

```
n0=L0*one((1,nd);
morerun = zeros(1,nd);
t_s_mean = zeros(1,nd);
ratio = ones(1,nd);
%--------------------------------------------------------------------
for i=1:nd,
  t_s_mean(i)=s_mean(i);
end

t_budget=add_budget;
for i=1:nd,
  t_budget=t_budget+n(i);
end

b=best(t_s_mean,nd); % call function "best"
s=second_best(t_s_mean,nd,b); % call function "second_best"
ratio(s)=1.0;

for i = 1:nd
  if(i~=s && i~=b)
  temp =(t_s_mean(b)-t_s_mean(s))./(t_s_mean(b)-t_s_mean(i));
  ratio(i)=temp.*temp.*s_var(i)./s_var(s);
  end
end
% calculate ratio of Ni/Ns

temp=0;
for i = 1:nd
  if(i~=b)
  temp = temp + ratio(i).*ratio(i)./s_var(i);
end
end
ratio(b)=sqrt(s_var(b).*temp); % calculate NB
for i = 1:nd
  t1_budget=t_budget;
end
while (more_alloc)
  more_alloc=0;
```

```
  ratio_s=0.0;
  for i = 1:nd,
  if(morerun(i))
  ratio_s = ratio_s + ratio(i);
  end
  end
  for i = 1:nd,
  if(morerun(i))
  an(i) = t1_budget./ratio_s.*ratio(i);
  if(an(i)<n(i))
  an(i)=n(i);
  morerun(i)=0;
  more_alloc=1;
  end
  end
  if(more_alloc)
  t1_budget=t_budget;
  for j = 1:nd
  if(~morerun(j))
  t1_budget = t1_budget-an(j);
  end
  end
  end
  end
end %end of WHILE
t1_budget=an(0);
for i = 1:nd
  t1_budget = t1_budget+an(i);
end
an(b) = an(b)+t_budget-t1_budget; % give the difference to design b
for i = 1:nd
  an(i)= an(i)-n(i);
end
```