Sections and Chapters

Gubert Farnsworth

ENME 625: Multi-Displinary Optimization
5/12/2017

Contents

1 Introduction

This is the first section.

2 Unconstrained MOGA Problems

We used this textbook [1]

2.1 ZDT1

The first test problem, denoted as ZDT1 in (insert reference) is shown below.

$$\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x}) = x_1 \\
\text{Minimize} \quad & f_2(\mathbf{x}) = g(x) * h(x) \\
\text{where} \quad & g(x) = 1 + \frac{9}{(n-1)} \sum_{i=2}^{n} x_i \\
& h(x) = 1 - \sqrt{\frac{f_1(x)}{g(x)}} \\
& n = 30 \\
& 0 \leq \mathbf{x} \leq 1
\end{aligned}$$

The true Pareto frontier for this problem occurs when $x_i = 0$ for i = 2,...,30. Figure 1 shows a sample result from both MATLABs built in MOGA and the MOGA developed in this project. The quality metrics chosen to evaluate this problem are Coverage Difference (CD)and Pareto Spread (OS). Ten runs for each algorithm were performed and the mean and standard deviation of each metric are tabulated in Table 1.

Table 1: Quality Metrics for ZDT1

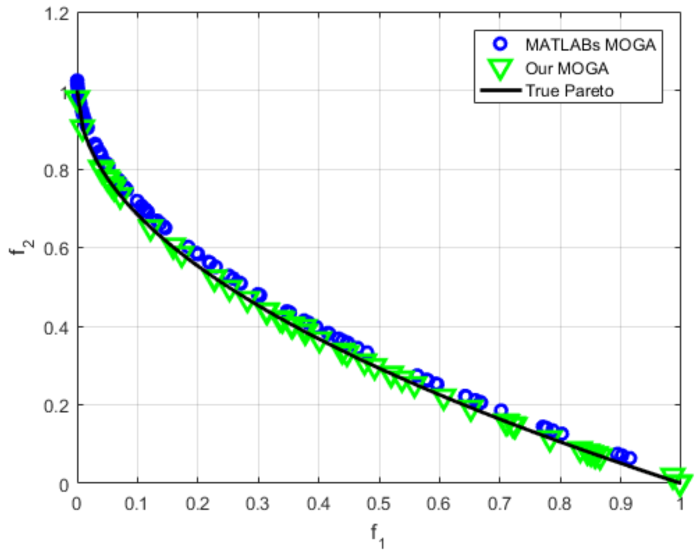
Metric	MATLABs MOGA	Our MOGA
CD	0.3874 (0.0164)	0.3582 (0.0040)
OS	0.9605 (0.1088)	0.9283 (0.0928)

From these metrics, there is certainly a trade-off between MATLABs MOGA and the MOGA developed in this project. The coverage difference of the new MOGA is better in this problem whereas the Pareto spread is improved when using MATLABs MOGA.

2.2 ZDT2

The second test problem, denoted as ZDT2 in (insert reference) is shown below.

Figure 1: Example Pareto Results for ZDT1



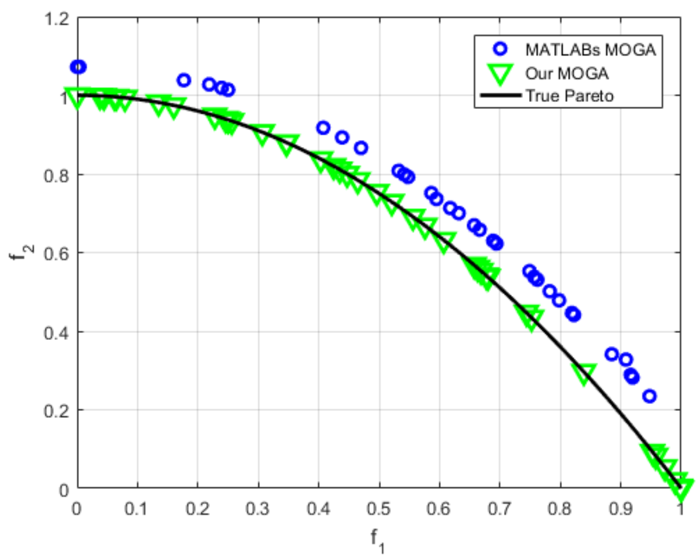
$$
\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x}) = x_1 \\
\text{Minimize} \quad & f_2(\mathbf{x}) = g(x) * h(x) \\
\text{where} \quad & g(x) = 1 + \frac{9}{(n-1)} \sum_{i=2}^{n} x_i \\
& h(x) = 1 - \frac{f_1(x)}{g(x)}^2 \\
& n = 30 \\
& 0 \leq \mathbf{x} \leq 1
\end{aligned}
$$

The true Pareto frontier for this problem, similar to ZDT1, occurs when x_i = 0 for i = 2,...,30. Figure 2 shows a sample result from both MATLABs built in MOGA and the MOGA developed in this project. Table 2 summarizes the mean quality metrics for each algorithm for ten runs.

Table 2: Quality Metrics for ZDT2

Metric	MATLABs MOGA	Our MOGA
CD	0.7832 (0.0821)	0.6971 (0.0094)
OS	0.8781 (0.0946)	1.0086 (0.0278)

Figure 2: Example Pareto Results for ZDT2



2.3 ZDT3

The third test problem, denoted as ZDT3 in (insert reference) is shown below.

$$
\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x}) = x_1 \\
\text{Minimize} \quad & f_2(\mathbf{x}) = g(x) * h(x) \\
\text{where} \quad & g(x) = 1 + \frac{9}{(n-1)} \sum_{i=2}^{n} x_i \\
& h(x) = 1 - \sqrt{\frac{f_1(x)}{g(x)}} - \frac{f_1(x)}{g(x)} \sin(10\pi f_1) \\
& n = 30 \\
& 0 \leq \mathbf{x} \leq 1
\end{aligned}
$$

The true Pareto frontier for this problem again occurs when $x_i = 0$ for i = 2,...,30. Figure 3 shows a sample result from both MATLABs built in MOGA and the MOGA developed in this project. Table 3 summarizes the mean quality metrics for each algorithm for ten runs.

Overall, MATLABs MOGA outperforms the MOGA developed in this project in both coverage difference and Pareto spread. A paired t-test shows that the difference in the means for coverage difference is statistically significant (p¡0.05), while the difference is not statistically significant in Pareto Spread (p=0.21).

Figure 3: Example Pareto Results for ZDT3

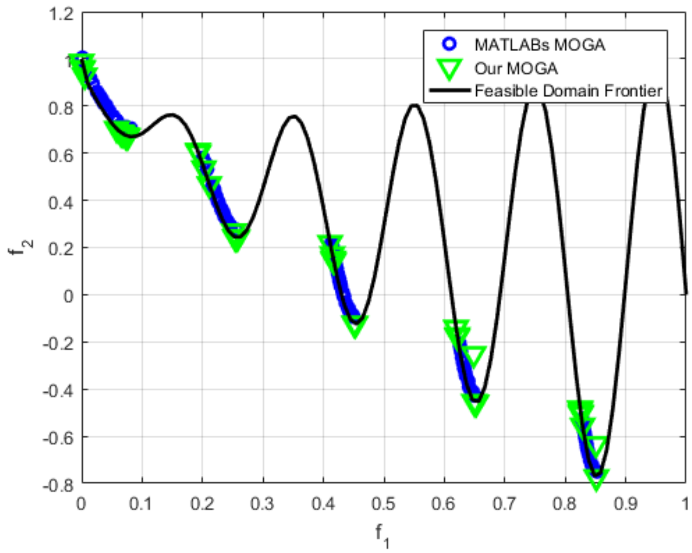


Table 3: Quality Metrics for ZDT3

Metric	MATLABs MOGA	Our MOGA
CD	0.7407 (0.0050)	0.7554 (0.0031)
OS	0.8565 (0.0098)	0.8489 (0.0180)

2.4 OSY

This test problem, denoted as OSY in (insert reference) is shown below.

$$
\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x}) = -(25(x_1-2)^2 + (x_2-2)^2 + (x_3-1)^2 + (x_4-4)^2 + (x_5-1)^2) \\
\text{Minimize} \quad & f_2(\mathbf{x}) = \sum_{i=1}^{6} x_i^2 \\
\text{Subject to} \quad & g_1(x) = 1 - \frac{x_1+x_2}{2} \leq 0 \\
& g_2(x) = \frac{x_1+x_2}{6} - 1 \leq 0 \\
& g_3(x) = \frac{x_2-x_1}{2} - 1 \leq 0 \\
& g_4(x) = \frac{x_1-3x_2}{2} - 1 \leq 0 \\
& g_5(x) = \frac{(x_3-3)^2+x_4}{4} - 1 \leq 0 \\
& g_6(x) = 1 - \frac{(x_5-3)^2+x_6}{4} \leq 0 \\
& \quad 0 \leq x_1, x_2, x_6 \leq 10 \\
& \quad 1 \leq x_3, x_5 \leq 5 \\
& \quad 0 \leq x_4 \leq 6
\end{aligned}
$$

The true Pareto frontier for this problem again occurs when $x_i = 0$ for i = 2,...,30. Figure **??** shows a sample result from both MATLABs built in MOGA and the MOGA developed in this project. Table 4 summarizes the mean quality metrics for each algorithm for ten runs.

Table 4: Quality Metrics for OSY

Metric	MATLABs MOGA	Our MOGA
CD		
OS		

3 Constrained MOGA Problems

This is a reference to Azarms constraint paper [2].

3.1 TNK

This test problem, denoted as TNK in (insert reference) is shown below with the true Pareto frontier shown in Figure 4.

$$
\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x}) = x_1 \\
\text{Minimize} \quad & f_2(\mathbf{x}) = x_2 \\
\text{Subject to} \quad & g_1(x) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(\frac{x_1}{x_2})) \leq 0 \\
& g_2(x) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 - 0.5 \leq 0 \\
& \quad 0 \leq x_1, x_2 \leq \pi
\end{aligned}
$$

Figure 4: True Pareto Frontier for TNK

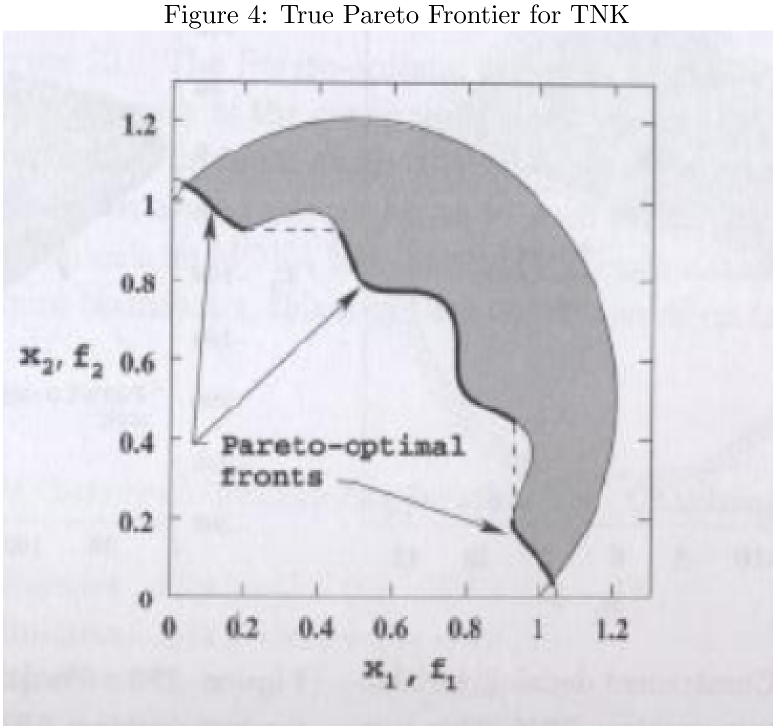


Figure 5 shows a sample result from both MATLABs built in MOGA and the MOGA developed in this project. Table 5 summarizes the mean quality metrics for each algorithm for ten runs.

Comparing the estimated Pareto frontiers to the true Pareto frontier, is is clear that our MOGA outperforms MATLAB's MOGA. The Pareto spread in our MOGA is significantly higher however the coverage difference is higher in MATLABs MOGA (p¡0.05).

Figure 5: Example Pareto Results for TNK

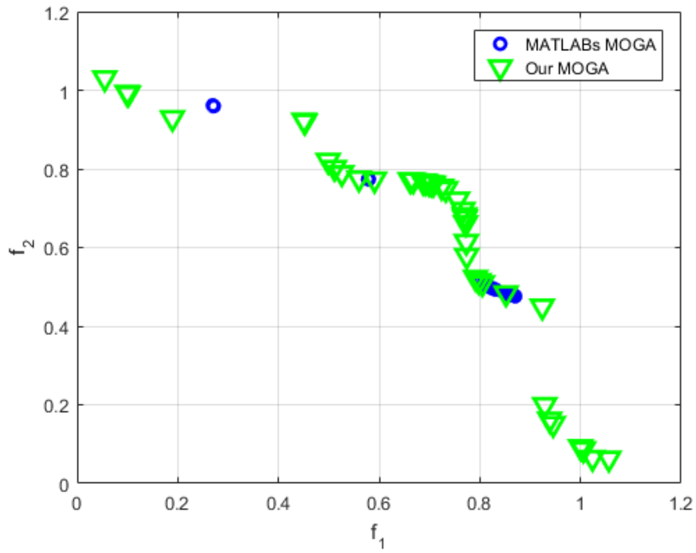


Table 5: Quality Metrics for TNK

Metric	MATLABs MOGA	Our MOGA
CD	0.8581 (0.0480)	0.7792 (0.0036)
OS	0.4177 (0.3216)	0.9763 (0.0178)

3.2 CTP

This test problem, denoted as CTP in (insert reference) is shown below and the true Pareto frontier is shown in Figure 6.

$$
\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x}) = x_1 \\
\text{Minimize} \quad & f_2(\mathbf{x}) = g(x)(1 - \sqrt{\frac{f_1(x)}{g(x)}} \\
\text{Subject to} \quad & g_1(x) = a|\sin(b\pi(\sin(\theta)(f_2(x) - e) + \cos(\theta) f_1(x))^c)|^d \\
& \quad - \cos(\theta)(f_2(x) - e) - \sin(\theta) f_1(x) \leq 0 \\
\text{where} \quad & \theta = -0.2\pi, a = 0.2, b = 10, c = 1, d = 6, e = 1 \\
& g(x) = |1 + (\sum_{i=2}^{10} x_i)^{0.25}| \\
& 0 \leq x_1 \leq 1 \\
& -5 \leq x_i \leq 5, i = 2, ..., 10
\end{aligned}
$$

Figure 6: True Pareto Frontier for CTP

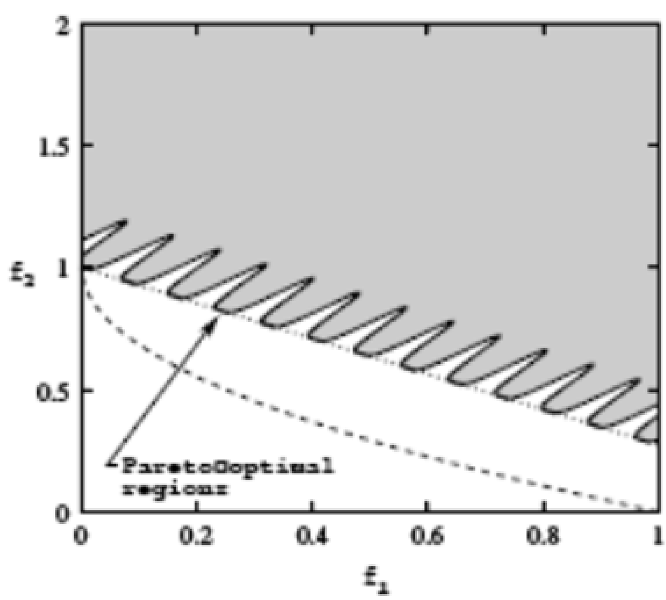


Figure 7 shows a sample result from both MATLABs built in MOGA and the MOGA developed in this project. Table 6 summarizes the mean quality metrics for each algorithm for ten runs [1].

Figure 7: Example Pareto Results for CTP

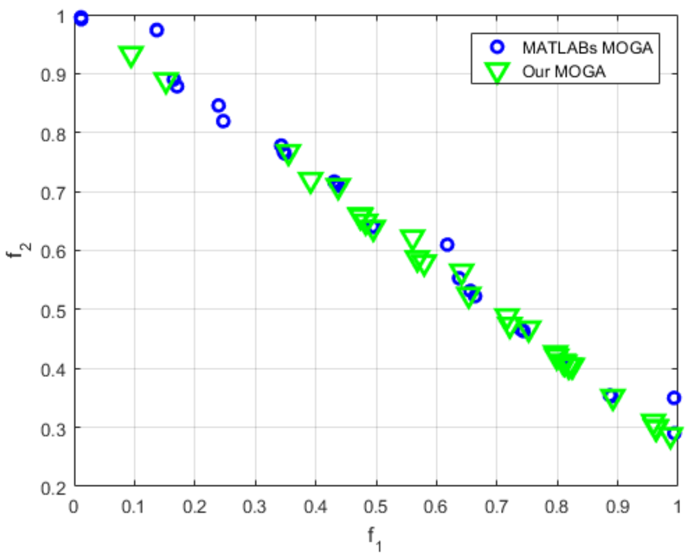


Table 6: Quality Metrics for CTP

Metric	MATLABs MOGA	Our MOGA
CD	0.6802 (0.0067)	0.6802 (0.0092)
OS	0.7901 (0.0734)	0.5959 (0.1324)

The mean values for coverage difference are exactly the same for both algorithms. In fact a paired t-test also shows the means are not statistically difference(p¿0.05). For Pareto spread, MATLAB's MOGA performs the highest.

4 Flight Planning Problem (FPP)

Now let us consider the final problem where we try to minimize the total flight time from the start location $(0, 12.5)$ to the finish location $(40, 12.5)$. In this problem there is a know wind velocity field which changes the total velocity of the aircraft with respect to the ground. At the same time, we want to maximize the distance of the flight path from some group of some exclusion zones. This also introduces a constraint that requires the flight path not intersect the exclusion zone. Each exclusion zone is approximated by a circle with known centers. The radius of each exclusion zone is known to $\pm 2m$. As a result this problem can be approached as a bi-objective optimization with a robust feasibility constraint.

$$\begin{aligned}
\text{Minimize} \quad & f_1(\mathbf{x},\mathbf{y}) = flightTime(\mathbf{x},\mathbf{y}) \\
\text{Maximize} \quad & f_2(\mathbf{x},\mathbf{y}) = -\min_{i,j} \sqrt{(\mathbf{x}_i - \mathbf{xc}_j)^2 + (\mathbf{y}_i - \mathbf{yc}_j)^2} \\
\text{Subject to} \quad & \mathbf{g}_j(\mathbf{x},\mathbf{y}) = -\min_{j} \left[\mathbf{r}_j + p_0 + \Delta p - \sqrt{(\mathbf{x}_i - \mathbf{xc}_j)^2 + (\mathbf{y}_i - \mathbf{yc}_j)^2}\right] \le 0 \\
\text{where} \quad & 0 \le x \le 50, i = 2, ..., 10 \\
& 0 \le y \le 25, i = 2, ..., 10 \\
& p_0 = 0 \\
& \Delta p \in [0, 2]
\end{aligned}$$

The function $flightTime(\mathbf{x},\mathbf{y})$ is a black-box. The values $\mathbf{xc}_j$ and $\mathbf{yc}_j$ specify the exclusion zone centers while $\mathbf{r}_j$ specify the radii. A solution to this problem is shown in figure 8. In this figure there are two exclusion zones.

Figure 8: True Pareto Frontier for CTP

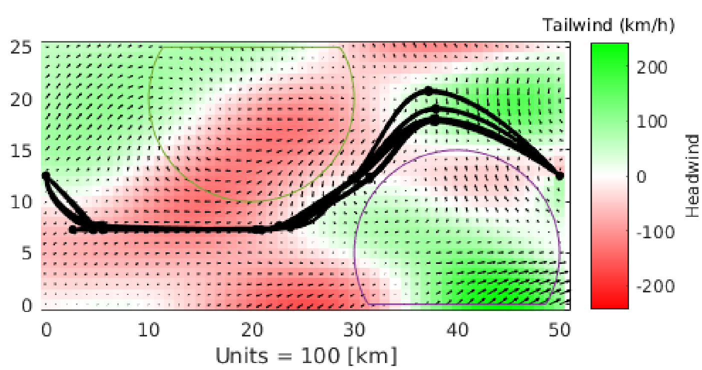
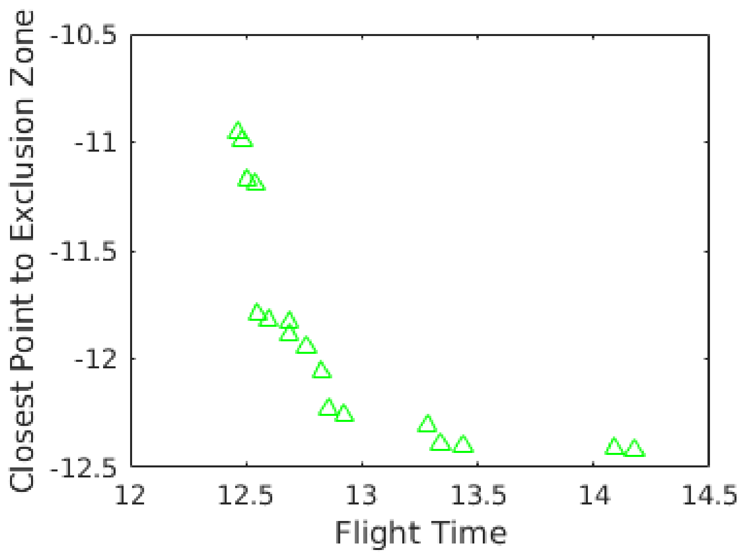


Figure 9: True Pareto Frontier for CTP



References

[1] K. Deb, "Multi-objective optimization using evolutionary algorithms, 2001," *Chicheter, John-Wiley.*, 2001.

[2] A. Kurpati, S. Azarm, and J. Wu, "Constraint handling improvements for multiobjective genetic algorithms," *Structural and Multidisciplinary Optimization*, vol. 23, no. 3, pp. 204–213, 2002.

5 Appendix

Matlab Code

```
%function [optX,optF]=MasterCode(prob,nChrome,nRun,alpha_
    ,sigma_,epsilon_,save_figure,use_matlabs_moga)
nargin=0;

% load .mat file
current_dir = pwd;
%file_name = 'results_and_params.mat';
if(contains(current_dir,'/ENME625_Optimization')) %linux
    or mac
    path_prefix = [current_dir, current_dir(1)];
```

```
elseif(contains(current_dir,'/ENME625_Optimization')) % windows
    path_prefix = [current_dir, '\'];
else
    fprintf('not running from the correct directory')
    return
end
file_name = [path_prefix,'results_and_params.mat'];
results_and_params = load(file_name);
results_and_params = results_and_params.results_and_params;

global alpha sigma epsilon

if(nargin < 1)
    prompt = 'Which Test Problem Do You Want To Run? \n 1 - ZDT1\n 2 - ZDT2 \n 3 - ZDT3 \n 4 - OSY \n 5 - TNK \n 6 - CTP \n 7 - Robust TNK \n';
    prob = input(prompt);
end
if nargin <2
    prompt2 = 'How Many Chromosomes? Suggest 20-30 ';
    nChrome = input(prompt2);
end
if nargin <3
    prompt3 = 'How Many Runs? Suggest >40: ';
    nRun = input(prompt3);
end
if nargin < 4
    prompt4 = 'What value for alpha? ';
    alpha = input(prompt4);
else
    alpha = alpha_;
end
if nargin < 5
    prompt5 = 'What value for sigma? (Nominal 0.158) ';
    sigma = input(prompt5);
else
    sigma = sigma_;
end
if nargin < 6
    prompt6 = 'What value for epsilon? (Nominal 0.22) ';
    epsilon = input(prompt6);
else
    epsilon = epsilon_;
end
```

```
if nargin <7
    prompt7 = 'Autosave figures [ 1 or 0 ]?';
    save_figure=input(prompt7);
end
if nargin <8
    prompt8 = 'Use Matlabs MOGA [ 1 or 0 ]?';
    use_matlabs_moga=input(prompt8);
end

 problem = results_and_params{prob,1};

% ZD-func is our problem function
switch prob
    case 1
        problem_function = @(X) ZDT1(X);
        nvar = 30; LB = zeros(1,nvar); UB = ones(1,nvar);
        problem_constraints = [];
    case 2
        problem_function = @(X) ZDT2(X);
        nvar = 30; LB = zeros(1,nvar); UB = ones(1,nvar);
        problem_constraints = [];
    case 3
        problem_function = @(X) ZDT3(X);
        nvar = 30; LB = zeros(1,nvar); UB = ones(1,nvar);
        problem_constraints = [];
    case 4
        problem_function = @(X) OSY(X);
        nvar = 6; LB = [0,0,1,0,1,0]; UB =
            [10,10,5,6,5,10];
        problem_constraints = @OSY_constraints; % Only
            used in matlab test
    case 5
        problem_function = @(X) TNK(X);
        nvar = 2; LB = [0,0]; UB=[pi,pi];
        problem_constraints = @TNK_constraints; % Only
            used in matlab test

    case 6
        problem_function = @(X) CTP(X);
        nvar = 10; LB = -5*ones(1,10); UB = 5*ones(1,10);
             LB(1,1) = 0; UB(1,1) = 1;
        problem_constraints = @CTP_constraints; % Only
            used in matlab test
    case 7
        DP=1;
        problem_function = @(X,DP) TNK_Robust(X,DP);
```

```
            nvar = 2; LB = [0,0]; UB=[pi,pi];
            robust_fitness = @(X,DP) TNK_NEGCN2(X,DP);
        otherwise
            problem_function = @(X) 0;
end

A = []; b = []; Aeq = []; beq = [];
if use_matlabs_moga ==1
    % Modify options setting
    options = optimoptions('gamultiobj');
    options = optimoptions(options,'PopulationSize', nRun);
    options = optimoptions(options,'CrossoverFcn', @crossoverscattered);
    options = optimoptions(options,'Display', 'final');
    options = optimoptions(options,'PlotFcn', { @gaplotpareto });
    options = optimoptions(options,'ParetoFraction', 0.9);
    indexat = @(expr, index) expr(index);
    problem_function = @(X) indexat(problem_function(X), 1:2);
    [~,optF] = gamultiobj(problem_function,nvar,[],[],[],[],LB,UB,problem_constraints,options);
    %problem.prob = prob; problem.nChrome = nChrome; problem.nRun = nRun;
    problem.matlab_optF = optF;
    results_and_params{prob,1} = problem;
    save(file_name,'results_and_params')
    return
end

Pareto = [];
options = optimoptions(@ga,'PopulationSize',nChrome,'UseVectorized',true,'CrossoverFraction', 0.90);
optF =[];
for gen = 1:nRun
    Obj_fcn = @(X) fitFCN5(X,problem_function);
    if(prob==7);Obj_fcn = @(X) fitFCN5(X,problem_function,robust_fitness);end
    [X,~,~,~] = ga(Obj_fcn,nvar,A,b,Aeq,beq,LB,UB,[],options);
    if(prob==7)
        [optF(gen,:)] = problem_function(X,[0,0]);
    else
        [optF(gen,:)] = problem_function(X);
```

```
    end
    optX(gen,:) = X;
end
nfunc = 2; % Making this static because it will not change in this project

P = paretoset(optF(:,1:nfunc));
m = 1;
for k = 1:length(P)
    if P(k) == 1
        Pareto(m,:) = optF(k,1:2); m = m+1;
    end
end

% figure
hold on;
if isempty(problem)==false
    if(isfield(problem,'matlab_optF'))
        ml_optF = problem.matlab_optF;
        plot(ml_optF(:,1),ml_optF(:,2),'b*')
    end
end

if(isempty(Pareto) == false)
    plot(Pareto(:,1),Pareto(:,2),'gv','LineWidth',2,'MarkerSize',10)
end
 plot(optF(:,1),optF(:,2),'r*','LineWidth',2)
hold on; grid on;
xlabel('f_1'); ylabel('f_2')

handle = gcf;
if save_figure == 1
    %Save the figures
    dir_val = pwd;
    saveFigure(handle,[dir_val,dir_val(1),num2str(prob),'_',num2str(nChrome,'%03.0f'),'_',num2str(nRun,'%04.0f')]);
    print([path_prefix,num2str(prob),'_',num2str(nChrome,'%03.0f'),'_',num2str(nRun,'%04.0f'),'.png'],'-dpng');

    %Save the .mat file
    problem.prob = prob; problem.nChrome = nChrome; problem.nRun = nRun;
    problem.alpha = alpha; problem.sigma = sigma; problem
```

```
        .epsilon = epsilon;
    problem.optF = optF; problem.Pareto= Pareto;
    results_and_params{prob,1} = problem;
    save(file_name,'results_and_params')
end
%save(['ZDT',num2str(prob),'_Nchr',num2str(nChrome),'run',num2str(nRun),'alp',num2str(alpha,2),'epsi',num2str(epsilon,3),'sig',num2str(sigma,3)])
```

```
function [ fit ] = fitFCN5(X, ZD_func, robust_constraint_fitness)
%NSGA algorithm. Use Approach 1 for sorting
global alpha sigma epsilon

cheat = 1;

if nargin < 3
    robust_constraint_fitness = [];
end

%Check is this is a robust problem
delta_P = [];
if isempty(robust_constraint_fitness)
    func = ZD_func(X);
else%Evaluates feasible solutions with uncertaintly applied in problems with
%uncertainy
    if cheat ==1
        delta_P = [0.5708*ones(length(X),1),-1*ones(length(X),1)];
    else
        [M,~] = size(X);
        for k = 1:M
            options = optimoptions(@ga,'PopulationSize',10,'UseVectorized',true);
            lb = [-2,-2]; ub = [2,2];
            fitnessfn = @(DP) -TNK_NEGCN2(X(k,:),DP);
            [d_p,~] = ga(fitnessfn,2,[],[],[],[],lb,ub,[],options);
            delta_P = [delta_P ; d_p];
        end
    end
    func = ZD_func(X,delta_P);
end


```

```matlab
%
% if isempty(existing_points)
%      fit  = sum(ZD_func(X));
%      return
% end

%% Find Dominate Points

% Modify existing code to find Pareto points to find dominant layers

% func = [existing_points;ZD_func(X)];

nfunc = func(1,end-3);
nconstr = func(1,end-2);
g = func(:,nfunc+1:nfunc+nconstr);
nconstr_lin = func(1,end-1);
if nconstr_lin > 0
    fprintf('weird error here')
end
h = func(:,nfunc+nconstr+1:nfunc+nconstr+nconstr_lin);
func = func(:,1:nfunc);

[M,~] = size(X);




if nconstr == 0 && nconstr_lin ==0
    nc_col = nfunc + 3;
    init_fit_col = nfunc + 4;
    sim_col = nfunc+5;
    indecies = [1:M]';
    func = [func,indecies]; %We need to know indecies later so this should save time
    P_temp = func;
    level = 0;
    level_col = nfunc +2;
    func(:,level_col) = 0;
    while ~isempty(P_temp)
        level = level+1;% increment the level value
        if length(P_temp(:,1))== 1
            %If there is only one value left at the end,
```

```
                assign this to a level
            func(P_temp(:,nfunc+1),level_col) = level;
            break
        end
        place = paretoset(P_temp(:,1:nfunc)); % get all the indecies in the lowest layer
        for k = 1:length(place)
            if place(k) == 1
                current_level_indecies = P_temp(k,nfunc+1); %map them from
                func(current_level_indecies, level_col) = level; % assiged from prtp
            end
        end

        P_temp(place, :) = [];
    end

    numLayer = level;

    %Make sure all individuals have a layer number
    flag = 0;
    for k = 1:M
        if func(level_col)==0
            func(k,level_col) = numLayer+1;
            flag = 1;
        end
    end
    if flag == 1, numLayer = numLayer+1; end

    %% Similarity
    %Assess similarity layer-by-layer, assess in objective space.

%     sigma = 0.158;
%     epsilon = 0.1;
%     alpha = 1;
    var_rem = 0;
    F_min = M+epsilon;
    for k = 1:numLayer
        Fitness = []; incl = [];
        incl = find(func(:,level_col)==k); % incl = include
        Fitness = func(incl,1:nfunc);
        var_rem = var_rem+length(Fitness(:,1));
        if(isempty(Fitness) == 0)
```

```
        if length(incl) == 1

            F_int = F_min-epsilon;
            Fit_share = F_int;
            func(incl,sim_col+1) = F_int;
            func(incl,sim_col) = F_int;
            func(incl,nc_col) = 1;
        else
            for m = 1:nfunc
                maxF(m) = max(Fitness(:,m));
                minF(m) = min(Fitness(:,m));
                func(m,init_fit_col) = minF(m);
            end
            d = []; similar = []; sh = [];
            for i = 1:length(incl)
                F_int = F_min-epsilon;
                for j = 1:length(incl)
                    for p = 1:nfunc
                        similar(p) = ((Fitness(i,p)-Fitness(j,p))/(maxF(p)-minF(p)))^2;
                    end
                    d(i,j) = sqrt(sum(similar));
                    if d(i,j)<=sigma
                        sh(i,j) = 1-(d(i,j)/sigma)^alpha;
                    else sh(i,j) = 0;
                    end
                end
                nc(i) = sum(sh(i,:));
                Fit_share(i) = F_int/nc(i);
                func(incl(i),nc_col) = nc(i);
                func(incl(i),sim_col) = Fit_share(i);
                func(incl(i),sim_col+1) = F_int;

            end
        end
        F_min = min(Fit_share);
    end

end

%Since a greater fitness value is a larger number, we use the inverse
fit = -func(:,sim_col);

```

```
%% Constraint Handling
else
     Cmax = 1.2; Cmin = 0.8;  r = 0.8*M;
     CF1 = 0.01;
     CF2 = 0.01;
     rank = zeros(1,M);

     % Assign moderate rank to all feasible solutions
      for k = 1:M
          flag = 0; flag_lin = 0;
          for p = 1:nconstr
               if g(k,p)>0, flag = 1;
               end
               if nconstr_lin ~= 0
                    if h(k,p)~=0, flag_lin = 1;
                    end
               end
          end
          if flag == 0 && flag_lin == 0
               rank(k) = 0.5*M;
          end
      end



      % Collect together feasible population
      feas_pop = []; infeas_pop = []; m = 1;
      for k = 1:M
           if rank(k) ~= 0
                if isempty(h)
                     feas_pop = [feas_pop;func(k,:),g(k,:)];
                else
                     feas_pop = [feas_pop;func(k,:),g(k,:),h(
                         k,:)];
                end
           else
                 if isempty(h)
                      infeas_pop = [infeas_pop;func(k,:),g(k
                          ,:)];
                 else
                      infeas_pop = [infeas_pop;func(k,:),g(k
                          ,:),h(k,:)];
                 end
                 loc(m) = k; m = m+1; %keep track of which
                      solutions were infeasible
           end
```

```
    end

    % Identify noninferior points
    if ~isempty(feas_pop)
        place = paretoset(feas_pop(:,1:nfunc));
        m = 1;
        for k = 1:length(place)
            if place(k) == 1
                Pareto(m,:) = feas_pop(k,:); %Assign noninferior points along with constraint values
                rank(k) = 1; m = m+1;
            end
        end
    end
    % Evaluate rank for infeasible individuals
    g = infeas_pop(:,nfunc+1:nconstr+nfunc);
    h = infeas_pop(:,nconstr+nfunc+1:end);

    for k = 1:length(g(:,1))
        for p = 1:nconstr
            if g(k,p)<=0
                feas_g(k,p) = 0; delta_g(k,p) = 0;
            else
                feas_g(k,p) = g(k,p); delta_g(k,p) = 1;
            end
        end
        if nconstr_lin == 0
            feas_h = zeros(length(g(:,1)),1);
            delta_h = zeros(length(g(:,1)),1);
        else
            for n = 1:nconstr_lin
                feas_h(k,n) = abs(h(k,n));
            end
            if h(k,p)==0, delta_h(k,p) = 0;
            else delta_h(k,p) = 1;
            end
        end
    end
    num1 = sum(feas_g,2)+sum(feas_h,2);
    denom1 = (sum(sum(feas_g))+sum(sum(feas_h)))/M;
    J = nconstr; K = nconstr_lin;
    num2 = (sum(delta_g,2)+sum(delta_h,2));
    denom2 = (J+K);

    factor1 = CF1.*(num1./denom1);
```

```
    factor2 = CF2.*(num2./denom2);



    for k = 1:length(g(:,1))
        if (factor1(k)> mean(factor1)) && (factor2(k) <
            mean(factor2))
            w1 = 0.75; w2 = 0.25;
        elseif (factor1(k) < mean(factor1)) && (factor2(k
            ) > mean(factor2))
            w1 = 0.25; w2 = 0.75;
        else
            w1 = 0.5; w2 = 0.5;
        end
        fit_constr(k) = -((Cmax-(Cmax-Cmin)*(r-1)/(M-1))
            -(w1.*factor1(k)+w2.*factor2(k)));
    end

    for k = 1:length(loc)
        rank(loc(k)) = fit_constr(k);
    end
    fit = rank;


end
end
```