APPENDIX

Simplex Method

```
function [soln obj status]=SimplexMethod(A, b, c, B_bar)
% The function SimplexMethod uses the simplex method to
% solve an LP in standard form:
%       min c' * x
%  s.t. A * x == b
%           x >= 0
%
% Inputs:
% c = n*1 vector of objective coefficients
% A = m*n matrix with m < n
% b = m*1 vector of RHS coefficients
% B_bar = m*1 vector that contains indices of basic variables
%
% Output:
% soln is an n*1 vector, the final solution of LP
% obj is a number, the final objective value of LP
% status describes the termination condition of the problem as follows:
% 1 - optimal solution
% -3 - unbounded problem
% If the LP is unbounded, soln returns the correspondig extreme direction

soln=[]; obj=[]; status=[];

%% Step 0: Initialization
% Generate an initial basic feasible solution and partition x, A, and c
% so that x=[x_B | x_N], A=[B | N], and c=[c_B | c_N]
indices = [1:length(c)]';
indices(find(ismember(indices, B_bar)==1)) = [];
N_bar = indices;
B = A(:,B_bar);            %basis matrix B
N = A(:,N_bar);            %non-basis matrix N
x_B = B\b;                 %basic variables
x_N = zeros(length(N_bar),1); %non-basic variables
x = [x_B; x_N];            %partition x according to basis
c_B = c(B_bar);            %obj coefficients of basic variables
c_N = c(N_bar);            %obj coefficients of non-basic variables
obj_init = [c_B; c_N]'*x; %initial objective function value
k = 0;

while k >= 0
    %% Step 1: Checking Optimality
    % Compute the reduced costs r_q=c_q-c_B¡¯*B^(-1)*N_q for q in N_bar
    % if r_q >= 0, STOP, current solution is optimal, else go to STEP 2
    pi = B'\c_B;         %solve the system B^T*pi=c_B for simplex multipliers
    r_N = c_N' - pi'*N; %compute reduced cost for non-basic variables
    ratio = find(r_N<0);
    if isempty(ratio)   %if r_q >= 0, then STOP. Optimal
        disp('probelm solved')
        status = 1;
        obj = [c_B; c_N]'*x;
        %indices of x are in increasing order
        indices_temp = [B_bar; N_bar];
        for a = 1:length(c)
            soln(a,1) = x(find(indices_temp==a));
        end
        break
```

```
    else % if r_q < 0, GO TO Step 2
        %% Step 2: Generating Direction Vector
        % Construct d_q=[-B^(-1)*N; e_q].
        % If d_q >= 0, then LP is unbounded, STOP, else go to STEP 3.
        enter_idx = ratio(1);  %choosing entering variable
        e = zeros(length(N_bar),1);
        e(enter_idx) = 1;      %construct vector e_q
        d = -B\(N(:,enter_idx)); %solve Bd=-N_q
        direction = [d; e];    %improved direction d
        d_idx = find(direction < 0);
        if isempty(d_idx)     %if direction > 0, then STOP.(unbounded)
            disp('unbounded problem')
            status=-3;
            indices_temp = [B_bar; N_bar];
            for a = 1:length(c)
                soln(a,1) = direction(find(indices_temp==a));
            end
            break
        else %if d_q < 0, GO TO Step 3
            %% Step 3: Generating Step Length
            % Compute step length by the minimum ratio test. Go to STEP 4.
            step_indices = -x(d_idx)./direction(d_idx);
            step = min(step_indices);
            for i = 1:length(d_idx)
                if step == -x(d_idx(i))/direction(d_idx(i))
                    leave_idx = d_idx(i);
                end
            end
            %% Step 4: Generating Improved Solution
            % Let x(k+1) = x(k) + alpha*d_q. Go to Step 5.
            x_d = x + step*direction;
            %leave_indices = find(x_d(1:length(B_bar))==0);
            %leave_idx = leave_indices(1); %determining leaving variable
            %% Step 5: Updating Basis
            % Generate the new basis B for next iteration,
            % Update c=[c_B|c_N],x=[x_B|x_N],& Aeq=[B|N]. Go to STEP 1.
            B_bar_temp = B_bar;
            N_bar_temp = N_bar;
            x_B = x_d(1:length(B_bar));
            x_N = x_d(length(B_bar)+1:end);
            x_B_temp = x_d(1:length(B_bar));
            x_N_temp = x_d(length(B_bar)+1:end);
            %exchange the entering and leaving variables in B_bar
            B_bar(leave_idx) = N_bar_temp(enter_idx);
            N_bar(enter_idx) = B_bar_temp(leave_idx);
            x_B(leave_idx) = x_N_temp(enter_idx);
            x_N(enter_idx) = x_B_temp(leave_idx);
            x = [x_B; x_N];           %update x = [x_B | x_N]
            B = A(:,B_bar);           %update basis B
            N = A(:,N_bar);           %update non-basis N
            c_B = c(B_bar);           %update c_B
            c_N = c(N_bar);           %update c_N
            obj_init = [c_B; c_N]'*x; %update objective value
            k = k+1; %GO TO Step 1
        end
    end
end
```

Dantzig-Wolfe Decomposition

```
function [soln, fval, flag] = dw_dec(master, sub, num_sub)
% This function implements the Dantzig-Wolfe Decomposition Algorithm
% This function is for farming problem only
% With slight modification, this function should apply to other problems
% This function works for both BOUNDED and UNBOUNDED subproblems
% This function uses SimplexMethod as its LP solver for subproblems
% Inputs:
% master: parameters of the master problem
% sub: parameters of the subproblems
% num_sub: number of subproblems
% num_sub = 3 in the farming problem
% Outputs:
% soln: solution vector if the LP is bounded
% fval: objective value if the LP is bounded
% flag: status of termination
% flag == 1: optimal solution found
% flag == -3: problem is unbounded
%% Step 0: Initialization
    soln = {};                      % solution
    fval = 0;                       % objective value
    flag = 0;                       % exit flag
    flag_optimal = 0;               % optimal flag
    flag_unbounded = 0;             % unbounded flag
    epsilon = 1e-16;                % termination threshold
    B = eye(4);                     % basis matrix of restricted MP
    b = [500;ones(num_sub,1)];      % RHS vector of restricted MP
    sol.c{1} = [0];                 % objective coefficients
    sol.x{1} = [500];               % extreme points/rays
    for i = 2:num_sub+1
        sol.c{i} = sub.c{i-1};
        sol.x{i} = sub.v0{i-1};
    end
    sol.p = [0;1;2;3];              % problem indices of extreme points

    f_B = zeros(num_sub+1,1);       % f_B
    x_simp = {};                    % solution from SimplexMethod
    fval_simp = {};                 % objective value from SimplexMethod
    flag_simp = {};             % exitflag from SimplexMethod

    % DW-Decomposition Main loop
    while ((1-flag_optimal) && (1-flag_unbounded))
        unbnd_sub = 0; % The flag indicating the existence of an unbounded
subproblem
        %% Step 1: Simplex Multiplier Generation
        for i = 1:num_sub+1
            f_B(i) = sol.c{i}'*sol.x{i};
        end
        x_B = B\b;                  % x_B
        pie = B'\f_B;               % pie
        pie_1 = pie(1);             % Recall pie_1 and pie_2 in pie

        %% Step 2: Optimality Check
        % Use my SimplexMethod instead of linprog
        enter_var = -1;             % entering variable
        for i = 1:num_sub
            A_sub = sub.A{i};       % A matrix of a subproblem
            b_sub = sub.b{i};       % b vector of a subproblem
            c_sub = (sub.c{i}' - pie_1*master.L{i})'; % c vector of a
subproblem
            basis_sub = sub.basis{i}';    % basis indices of a subproblem
```

```
[x_simp{i},fval_simp{i},flag_simp{i}]=SimplexMethod(A_sub,b_sub,c_sub,basis_sub
);
            sol.x_temp{i} = x_simp{i};    % temporary extreme points
            if flag_simp{i} == -3
                enter_var = i+1;
                r_N = [0;-1;-1;-1];       % reduced costs
                unbnd_sub = 1;
                break;
            end
        end

        if enter_var == -1
            r_N = [0;fval_simp{1}-pie(2);fval_simp{2}-pie(3);fval_simp{3}-
pie(4)];
            enter_var = find(r_N==min(r_N));
        end

        if sum(r_N<-1e-3==1)==0    % if r_N >= 0, stop, optimal solution found
            disp('Optimal Solution Found !')
            flag_optimal = 1;
            flag = 1;
            % output objective value
            fval = f_B' * (B\b);
            display(fval)
            % output varaibles
            soln{1}=zeros(length(sub.c{1}),1);
            soln{2}=zeros(length(sub.c{2}),1);
            soln{3}=zeros(length(sub.c{3}),1);
            x_B_opt=B\b;
            for i = 1:num_sub+1
                prob = sol.p(i);
                soln{prob} = soln{prob} + x_B_opt(i)*sol.x{i};
            end
            solution = [soln{1};soln{2};soln{3}];
            disp(solution)
            break;
        end

        %% Step 3: Column Generation
        a_bar=zeros(num_sub+1,1);          % a_bar
        enter_prob = sol.p(enter_var);     % subproblem index of entering var
        if enter_prob>0
            if unbnd_sub == 0
                a_bar(enter_prob+1)=1;
            else
                a_bar(enter_prob+1)=0;
            end
            a_bar(1)=master.L{enter_prob}*sol.x_temp{enter_prob};
        end
        if enter_prob==0
           a_bar(1)=1;
        end

        %% Step 4: Descent Direction Generation
        d = B\(-a_bar);                    % descent direction
        if sum(d<-epsilon==1)==0           % if # of all d>>0, stop, problem
unbounded
            flag_unbounded=1;
            flag = -3;
            disp('Problem Unbounded !')
            break;
        end
```

```
        %% Step 5: Step Length Generation
        leave_cand = find(d<-epsilon);     % leaving variable candidates
        ratio=-x_B(leave_cand)./d(leave_cand);
        alpha=min(ratio);                  % alpha: min ratio test
        leave_cand = leave_cand(find(ratio==alpha));
        leave_var = leave_cand(1);         % leaving variable

        %% Step 7: Basis Update (Step 6 is done in another way)
        B(:,leave_var)=a_bar;
        if enter_prob>0
           f_B(leave_var) = sub.c{enter_prob}'*sol.x_temp{enter_prob};
           sol.x{leave_var} = sol.x_temp{enter_prob};
           sol.c{leave_var} = sub.c{enter_prob};
           sol.p(leave_var) = enter_prob;
        end

        if enter_prob==0
           f_B(leave_var)=B\b(leave_var);
           sol.c{leave_var} = [0];
           sol.p(leave_var) = 0;
        end
    end
end
```

Problem Generation

```
%% Problem Instances Generation
clear all;
num_s = 20;      % number of scenarios
num_i = 5;        % for each scenario, run a number of instances
num_sub = 3;      % number of subproblems, set to 3
timelimit = 600; % time limit is 600 seconds = 10 minutes

%% Randomize Nonzero Parameters
para_1 = [];
para_2 = [];
para_3 = [];
for i = 1:num_s
    para_1(i) = 2+randi(100)/100;
    para_2(i) = 2+randi(200)/100;
    para_3(i) = -10-randi(200)/10;
end

% para_1 = [3,2.5,2];
% para_2 = [3.6,3,2.4];
% para_3 = [-24,-20,-16];

%% LP Instance with Unbounded Subproblems
% linking constraints
L1 = [1,zeros(1,num_s*3)];
L2 = [1,zeros(1,num_s*3)];
L3 = [1,zeros(1,num_s*4)];

% master problem parameters
master_u.b = 500;
master_u.L{1} = L1;
master_u.L{2} = L2;
master_u.L{3} = L3;

% objective coefficient vector: c
c1 = [150];
c2 = [230];
c3 = [260];
for i = 1:num_s
    c1 = [c1;238/num_s;-170/num_s];
    c2 = [c2;210/num_s;-150/num_s];
    c3 = [c3;-36/num_s;-10/num_s];
end
c1 = [c1;zeros(num_s,1)];
c2 = [c2;zeros(num_s,1)];
c3 = [c3;zeros(2*num_s,1)];

% RHS vector: b
b1 = [200*ones(num_s,1)];
b2 = [240*ones(num_s,1)];
b3 = [zeros(num_s,1);6000*ones(num_s,1)];

% constraint matrix of subproblems: A
A1 = zeros(num_s,num_s*3+1);
A2 = zeros(num_s,num_s*3+1);
A3 = zeros(2*num_s,num_s*4+1);
A1(:,1) = para_1';
A2(:,1) = para_2';
A3(:,1) = [para_3';zeros(num_s,1)];
for i = 1:num_s
    A1(i,i*2:i*2+1) = [1,-1];
    A2(i,i*2:i*2+1) = [1,-1];
    A3(i,i*2:i*2+1) = [1,1];
```

```
end
A1(:,num_s*2+2:end) = -1*eye(num_s);
A2(:,num_s*2+2:end) = -1*eye(num_s);
for i = 1:num_s
    A3(num_s+i,2*i) = 1;
end
A3(:,num_s*2+2:end) = eye(num_s*2);

% basic variables
basic_v1 = [];
basic_v2 = [];
basic_v3 = [];
for i = 1:num_s
    basic_v1(i) = 2*i;
    basic_v2(i) = 2*i;
    basic_v3(i) = 2*i+1;
    basic_v3(num_s+i) = num_s*3+1+i;
end

% initial extreme points
v1_init = zeros(length(c1),1);
B1 = A1(:,basic_v1);
v1_init_t = B1\b1;
v1_init(basic_v1) = v1_init_t;

v2_init = zeros(length(c2),1);
B2 = A2(:,basic_v2);
v2_init_t = B2\b2;
v2_init(basic_v2) = v2_init_t;

v3_init = zeros(length(c3),1);
B3 = A3(:,basic_v3);
v3_init_t = B3\b3;
v3_init(basic_v3) = v3_init_t;

% master problem parameters
sub_u.c{1} = c1;
sub_u.c{2} = c2;
sub_u.c{3} = c3;
sub_u.b{1} = b1;
sub_u.b{2} = b2;
sub_u.b{3} = b3;
sub_u.A{1} = A1;
sub_u.A{2} = A2;
sub_u.A{3} = A3;
sub_u.basis{1} = basic_v1;
sub_u.basis{2} = basic_v2;
sub_u.basis{3} = basic_v3;
sub_u.v0{1} = v1_init;
sub_u.v0{2} = v2_init;
sub_u.v0{3} = v3_init;

%% Generate STD Form for SimplexMethod
% need one more slack variable
c_ = [c1;c2;c3;0];
b_ = [500;b1;b2;b3];
A_sub = blkdiag(A1,A2,A3);
A_link = [L1,L2,L3];
A_ = [A_link;A_sub];
slack_link = zeros(length(b_),1);
slack_link(1) = 1;
A_ = [A_,slack_link];
B_bar = [];
for i = 1:num_s
```

```
    B_bar = [B_bar;2*i];
end
for i = 1:num_s
    B_bar = [B_bar;1+3*num_s+2*i];
end
for i = 1:num_s*2
    B_bar = [B_bar;2+6*num_s+2*num_s+1+i];
end
B_bar = [B_bar;length(c_)];

%% Directly Use Gurobi to Solve the LP
model.A = sparse(A_);
model.obj = c_';
model.rhs = b_;
model.sense = '=';
model.vtype = 'C';
model.modelsense = 'min';
params.outputflag = 0;
params.Method = -1;
t1 = clock;
result = gurobi(model, params);
t2 = clock;
t(7) = etime(t2,t1)
disp(result)
fval_g = result.objval
x1 = result.x

%% Directly Use linprog (interior-point-legacy) to Solve the LP
options = optimoptions('linprog','Algorithm','interior-point-legacy');
t1 = clock;
[soln_lp1,fval_lp1,flag_lp1,output_lp1]=linprog(c_,[],[],A_,b_,zeros(length(c_)
,1),[],options);
t2 = clock;
t(4) = etime(t2,t1)

%% Directly Use linprog (interior-point) to Solve the LP
options = optimoptions('linprog','Algorithm','interior-point');
t1 = clock;
[soln_lp2,fval_lp2,flag_lp2,output_lp2]=linprog(c_,[],[],A_,b_,zeros(length(c_)
,1),[],options);
t2 = clock;
t(5) = etime(t2,t1)

%% Directly Use linprog (dual-simplex) to Solve the LP
options = optimoptions('linprog','Algorithm','dual-simplex');
t1 = clock;
[soln_lp3,fval_lp3,flag_lp3,output_lp3]=linprog(c_,[],[],A_,b_,zeros(length(c_)
,1),[],options);
t2 = clock;
t(6) = etime(t2,t1)

%% The Same LP Instance with Bounded Subproblems
% linking constraints
L1 = [1,zeros(1,num_s*3+1)];
L2 = [1,zeros(1,num_s*3+1)];
L3 = [1,zeros(1,num_s*4+1)];

% master problem parameters
master_b.b = 500;
master_b.L{1} = L1;
master_b.L{2} = L2;
master_b.L{3} = L3;

% objective coefficient vector: c
c1 = [150];
```

```
c2 = [230];
c3 = [260];
for i = 1:num_s
    c1 = [c1;238/num_s;-170/num_s];
    c2 = [c2;210/num_s;-150/num_s];
    c3 = [c3;-36/num_s;-10/num_s];
end
c1 = [c1;zeros(num_s+1,1)];
c2 = [c2;zeros(num_s+1,1)];
c3 = [c3;zeros(2*num_s+1,1)];

% RHS vector: b
b1 = [200*ones(num_s,1);500];
b2 = [240*ones(num_s,1);500];
b3 = [zeros(num_s,1);6000*ones(num_s,1);500];

% constraint matrix of subproblems: A
A1 = zeros(num_s+1,num_s*3+2);
A2 = zeros(num_s+1,num_s*3+2);
A3 = zeros(2*num_s+1,num_s*4+2);
A1(1:num_s,1) = para_1';
A2(1:num_s,1) = para_2';
A3(1:2*num_s,1) = [para_3';zeros(num_s,1)];
for i = 1:num_s
    A1(i,i*2:i*2+1) = [1,-1];
    A2(i,i*2:i*2+1) = [1,-1];
    A3(i,i*2:i*2+1) = [1,1];
end
A1(1:num_s,num_s*2+2:end-1) = -1*eye(num_s);
A2(1:num_s,num_s*2+2:end-1) = -1*eye(num_s);
A1(num_s+1,1) = 1;
A1(num_s+1,end) = 1;
A2(num_s+1,1) = 1;
A2(num_s+1,end) = 1;
for i = 1:num_s
    A3(num_s+i,2*i) = 1;
end
A3(1:2*num_s,num_s*2+2:end-1) = eye(num_s*2);
A3(2*num_s+1,1) = 1;
A3(2*num_s+1,end) = 1;

% basic variables
basic_v1 = [];
basic_v2 = [];
basic_v3 = [];
for i = 1:num_s
    basic_v1(i) = 2*i;
    basic_v2(i) = 2*i;
    basic_v3(i) = 2*i+1;
    basic_v3(num_s+i) = num_s*3+1+i;
end
basic_v1 = [basic_v1,num_s*3+2];
basic_v2 = [basic_v2,num_s*3+2];
basic_v3 = [basic_v3,num_s*4+2];

% initial extreme points
v1_init = zeros(length(c1),1);
B1 = A1(:,basic_v1);
v1_init_t = B1\b1;
v1_init(basic_v1) = v1_init_t;

v2_init = zeros(length(c2),1);
B2 = A2(:,basic_v2);
v2_init_t = B2\b2;
```

```
v2_init(basic_v2) = v2_init_t;

v3_init = zeros(length(c3),1);
B3 = A3(:,basic_v3);
v3_init_t = B3\b3;
v3_init(basic_v3) = v3_init_t;

% master problem parameters
sub_b.c{1} = c1;
sub_b.c{2} = c2;
sub_b.c{3} = c3;
sub_b.b{1} = b1;
sub_b.b{2} = b2;
sub_b.b{3} = b3;
sub_b.A{1} = A1;
sub_b.A{2} = A2;
sub_b.A{3} = A3;
sub_b.basis{1} = basic_v1;
sub_b.basis{2} = basic_v2;
sub_b.basis{3} = basic_v3;
sub_b.v0{1} = v1_init;
sub_b.v0{2} = v2_init;
sub_b.v0{3} = v3_init;

%% Directly Use SimplexMethod to Solve the LP
% t1 = clock;
% [soln_s, fval_s, flag_s] = SimplexMethod(A_, b_, c_, B_bar);
% t2 = clock;
% t(1) = etime(t2,t1)

%% Use DW-Decomposition with SimplexMethod as its LP Solver
t1 = clock;
[soln_u, fval_u, flag_u] = dw_dec(master_u, sub_u, num_sub);
t2 = clock;
t(2) = etime(t2,t1)

%% Use DW-Decomposition with SimplexMethod as its LP Solver
t1 = clock;
[soln_b, fval_b, flag_b] = dw_dec(master_b, sub_b, num_sub);
t2 = clock;
t(3) = etime(t2,t1)

t'

% Timer Index:
% 1: Simplex
% 2: DW with Unbounded Subproblems
% 3: DW with Bounded Subproblems
% 4: linprog with interior-point-legacy
% 5: linprog with interior-point
% 6: linprog with dual-simplex
```