FOR CHAPTER 3
Regional routing MATLAB code

```
%Main on google drive
%Will Armington
%Regional routing for residential food waste collection

clear;
pause(2);
delete *.log %delete the cplex clones to prevent matlab from crashing

addpath('C:\Program   Files\IBM\ILOG\CPLEX_Studio1271\cplex\matlab\x64_win64')
%home computer

pause(2);
%
addpath('C:\Users\wra9936\Documents\MATLAB\CplexMilp12.7.1\cplex\matlab\x64_w
in64')
% %school computer

opts = cplexoptimset('Display','on');

% tic %start timer
%inputs

%input case number for cluster (1) vs region (0)
NH_case = 0;

%select the neighborhoods and # of houses in neighborhoods
NH_Select = [7]; %vector of neighborhood numbers for routing to be done within
HH_Select = [15]; % vector of the number of households within each neighborhood
to be randomly selected to perform routing
NHcount = size(NH_Select,2);

% N = 200; %number of households to randomly sample
D = 1; %Number of depots

%general inputs, comment out if specific inputs are used from spreadsheet file
if NH_case == 1
    F = 0; %number of recycling facilities
    K = 1; %number of collection trucks
else
    F = 1; %number of recycling facilities
    K = 1; %number of collection trucks
end

Q = 2000; %lbs. Capacity of collection trucks if all the same type
q = 1; %lbs. food waste quantity generated by households.
    %REFED average FW generation per person in 2016 was 167 lbs. => 3.2 lbs/week
% g = 35; %mph. average speed between nodes. Need conversion to meters/second
% gfacil = 55; %mph. average speed from neighborhoods to recycling facility
m = 0.5; %Minutes. Pickup time at households
B_lim = 8*60*60; %seconds. Time limit in a work day
```

```
P_lo = 0; %lbs. recycling facility quota
P_hi = 1000; %lbs. recycling facility capacity
C_hr = 100; %operating cost $/hr

%If OD matrix is in minutes, comment out
% avgspd_ms = g*0.44704; %Average travel speed of collection vehicle (mi/hr)
% avgspd_ms_recyc = gfacil*0.44704;
% C_s = C_hr/3600; %convert to $/second;
C_m = C_hr/60; %convert to $/minute;

%Inputs from spreadsheets

%Load network OD matrices
%Neighborhoods with household OD matrices
load('Matlab_Inputs/Pen_ParcSamp_ODtime.mat'); %more than one...3D matrix after
compiling?
%includes total number of neighborhoods with empty cells for neighborhoods that
are not included yet.
%Neighborhoods are in the correct matrix number compared to their number
load('Matlab_Inputs/NH_ODs.mat');
% Centroids of neighborhoods w/ depot
load('Matlab_Inputs/Pen_NBHCent_OD.mat');
%Load parcel descriptor data of entire network
HHdata_orig_t = struct2table(load('Matlab_Inputs/PenfieldParcelData.mat'));

HH_tot = size(HHdata_orig_t.OBJECTID_1,1);
%Load household size, location, other data



NHmax = max(HHdata_orig_t.Neighborhood_Num);
%split households and information into NB vector
for NH = 1:NHmax
    idxNH = HHdata_orig_t.Neighborhood_Num == NH;
    HHref_data{1,NH} = HHdata_orig_t(idxNH,:);
    HHref_data{1,NH}.HH_num(:,1) = 1:size(HHref_data{1,NH},1);
end

%need references to which households belong to which neighborhoods
%external reference sheet? NB numbers (figure), number of HH in each NB, NB
%area, current Community Composting service,




%create a structure of households selected for routing in each
%neighborhood. reference relevant data from main matrix if needed
%for loop
for NH = 1:NHmax
    if any(NH_Select == NH) == 1
        rng default %for reproducability
        idxNH = NH_Select == NH;
        HHpick = sort(randperm(size(HHref_data{1,NH},1),HH_Select(idxNH)));
        HHpick_data{1,NH} = HHpick;
    else
        HHpick = 0;
        HHpick_data{1,NH} = HHpick;
```

```
    end
end

% N_pick = sort(randperm(HH_tot,N));
%                                 Parcel_OD_pick                                  =
ParcOD_TimeCost([1,N_pick,1+HH_tot+1:1+HH_tot+F],[1,N_pick,1+HH_tot+1:1+HH_to
t+F]);
%
% HHdata_pick = HHdata_orig_t(N_pick,:);
% HHdata_pick = table2struct(HHdata_pick,'ToScalar',true);

%somehow plot coordinates of households to gain visual understanding. Not
%necessary but nice


% HHcoords = [HHdata_pick.POINT_X,HHdata_pick.POINT_Y];

%trim cost matrices from neighborhoods for the chosen households
NH_ODs_trim = NH_ODs;
for NH = NH_Select
    tempOD = NH_ODs{NH};
    NH_ODs_trim{1,NH}      =      tempOD(HHpick_data{1,NH},HHpick_data{1,NH});
%???????????/
end


%narrow cost matrix for neighborhoods chosen
Pen_NBHCent_OD_trim                                                             =
Pen_NBHCent_OD([1,NH_Select+1,1+NHmax+F],[1,NH_Select+1,1+NHmax+F]);

%trim reference data to match OD cost matrices
HHref_data_trim = HHref_data;
for NH = 1:NHmax
    if any(NH_Select == NH) == 1
        tempHHref = HHref_data{NH};
        HHref_data_trim{1,NH} = tempHHref(HHpick_data{1,NH},:);
    else
        HHref_data_trim{NH} = 0;
    end
end

%clustering algorithm. Do not use if clusters/neighborhoods are pre-defined
% clust_err = zeros(1,3);
% cluster_data = cell(2,1);
% for clust = 1:10
%     err = [];
%                                         [idxClust,centers,sumd,InD]         =
kmeans(HHcoords,clust,'Distance','cityblock');
%     for n = 1:N
%         err_temp = InD(n,idxClust(n))^2;
%         err = [err;err_temp];
%     end
%     SSE = sum(err);
%     if clust > 1
```

```
%                           clust_err   =   [clust_err;   clust,   SSE,   ((SSE-
clust_err(clust,2))/clust_err(clust,2))*100];
%     else
%         clust_err = [clust_err; clust, SSE, -999];
%     end
%     cluster_data{1,clust} = idxClust;
%     cluster_data{2,clust} = centers;
% end

% figure;
% % scatter(clust_err(2:end,1),clust_err(2:end,2))
% scatter(HHcoords(:,1),HHcoords(:,2),'k*');

% %%%%%%
% CNH = 5; %based on elbow in graph. Manual input. Needed to route neighborhoods
% %%%%%%

% HHclustCount = zeros(1,CNH);
% HHdata_pick.ClusterNo = cluster_data{1,CNH};
% NHcent_data = cluster_data{2,CNH};
% idxClust = cluster_data{1,CNH};
% cn = unique(idxClust);
%
% colorspec = 'mcrgbky'; %only 7 cluster colors unless using a different color
scheme
% figure;
% hold on
% for clust = 1:CNH
% scatter(HHcoords(idxClust==clust,1),...
%     HHcoords(idxClust==clust,2),...
%     sprintf('%s.',colorspec(clust)));
% plot(NHcent_data(clust,1),...
%      NHcent_data(clust,2),...
%      'kx','MarkerSize',15,'LineWidth',3)
% end

% HHclustCount = [cn,histc(idxClust(:),cn)];

NHNetData.NH_Select = NH_Select';
NHNetData.HH_Select = HH_Select';
NHNetData.RouteTime = zeros(NHcount,1);
NHNetData.qgen = zeros(NHcount,1);
NHNetData.Cost = zeros(NHcount,1);

%call individual neighborhood routing function
%[NH_Select, NHcount, HH_Select, NH_ODs_trim,
[NHNetData]...
    = NH_ResCollectRoute_v6...

(NHcount,NH_Select,NHNetData,HHref_data_trim,NH_ODs_trim,Q,q,m,C_hr,B_lim,P_l
o,P_hi);

%create distance matrix from cluster function outputs
% startidx = zeros(N,1);
% endidx = zeros(N,1);
```

```
% startidx(NHNetData.StartHH) = 1;
% endidx(NHNetData.EndHH) = 1;
%
% startidx = [1;startidx;1];
% endidx = [1;endidx;1];

t_matrix = Pen_NBHCent_OD_trim; %/ 1609.34; %convert distance of meters to miles
qvec = NHNetData.qgen; %/2000; %convert lbs of generation to tons
mvec = NHNetData.RouteTime;

%Add depot prime to end of cost matrix
t_matrix(1+NHcount+F+1,:) = t_matrix(1,:);
t_matrix(:,1+NHcount+F+1) = t_matrix(:,1);



% %Add depot prime to end of cost matrix
% if NH_case == 1
%     t_matrix(D+CNH+1,:) = 0;
%     t_matrix(:,D+CNH+1) = 0;
%     t_matrix(D,:) = 0;
%     t_matrix(:,D) = 0;
% else
%     t_matrix(D+CNH+F+1,:) = t_matrix(1,:);
%     t_matrix(:, D+CNH+F+1) = t_matrix(:,1);
% end

%create time matrix for links
% B_matrix(D+1:CNH,D+1:CNH) = t_matrix(D+1:CNH,D+1:CNH) ./ avgspd_ms; %convert
distance matrix to time matrix
% B_matrix(D,:) = t_matrix(D,:) ./ avgspd_ms_recyc;
%    B_matrix(:,D+CNH+1:D+CNH+F+D)    =    t_matrix(:,D+CNH+1:D+CNH+F+D)    ./
avgspd_ms_recyc;
Bm_matrix = t_matrix;

%Add intra-neighborhood collection cost to OD matrix
% j = 1;
% for i = D:NHcount+D
%     for h = 1:NHcount+D
%         if i == h
%             continue
%         end
%         Bm_matrix(h,i) = Bm_matrix(h,i) + mvec(j);
%
%     end
%     j = j + 1;
% end
% Bm_matrix(1:D+CNH,D+1:D+CNH) = B_matrix(1:D+CNH,D+1:D+CNH) + m; %add pickup
time to destination node


C_matrix = Bm_matrix .* C_m; %convert total time matrix to monetary cost


%Cost matrix order should be D, N, F, D'
```

```
%Single decision variable counts
if NH_case == 1
    sDV_y = D+NHcount; %household pickup decisions for each truck
    sDV_B = 1; %Recycling facility drop-off decisions for each truck
    sDV_w = 1; %Volume of waste delivered to recycling facility for each truck
    sDV_v = 0; %total volume waste delivered to recycling facility
else
    sDV_y = D+NHcount; %household pickup decisions for each truck
    sDV_B = 1; %Recycling facility drop-off decisions for each truck
    sDV_w = F; %Volume of waste delivered to recycling facility for each truck
    sDV_v = F; %total volume waste delivered to recycling facility
end

%++++++++++++++++++++
%For matlab 2016a when creating output table because we dont update your school
computers
sz = D*2+NHcount+F;
I = cell(sz);
Yl = cell(sDV_y,1);
Bl = cell(sDV_B,1);
Wl = cell(sDV_w,1);
Vl = cell(sDV_v,1);
%++++++++++++++++++++

%total decision veriable counts
DV_y = sDV_y*K; %household pickup decisions for each truck
DV_B = sDV_B*K; %Time count for truck routes
DV_w = sDV_w*K; %Volume of waste delivered to recycling facility for each truck
DV_v = sDV_v; %total number of volume waste variables delivered to recycling
facility

%flag matrix to remove impossible solutions
flag_mat = ~eye(size(t_matrix)); %no solutions along diagonal
flag_mat(:,1) = 0; %No places go back to depot
flag_mat(D+NHcount+F+1:D+NHcount+F+D,:) = 0; %Depot prime does not go anywhere
flag_mat(D,D+NHcount+D) = 0; %Depot cannot go to depot prime

if NH_case ~= 1
        flag_mat(D+NHcount+1:D+NHcount+F,1:D+NHcount+F)    =    0;    %recycling
facilities only go to depot prime
        flag_mat(1:D,D+NHcount+1:D+NHcount+F)   =   0;   %Depot   cannot   go   to
recycling facilities
        flag_mat(D:D+NHcount,D+NHcount+F+1:D+NHcount+F+D)   =   0;   %households
cannot go to depot prime
end

flag_mat_tr = flag_mat';

%create costing for objectives
C_mat_tr = C_matrix';
sc_x = C_mat_tr(flag_mat_tr);
sDV_x = size(sc_x,1); %create single number of x decision variables ebfore
vehicles are accounted for
c_x = sc_x;
```

```
if K > 1
    for k = 2:K
        c_x = [c_x;sc_x];
    end
end

c_y = zeros(DV_y,1);
c_B = zeros(DV_B,1);
c_w = zeros(DV_w,1);
c_v = zeros(DV_v,1);
obj = [c_x;c_y;c_B;c_w;c_v];

%Decision Veriable count part 2
DV_x = sDV_x*K; %total possible arcs

DV_tot = DV_x+DV_y+DV_B+DV_w+DV_v; %total possible decision variables

%CPLEX settings
% ctype = [];
ctype(1:DV_x) = {'B'};
ctype(DV_x+1:DV_tot) = {'C'};
ctype = char(ctype)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Constraints

%IC2 or EC2 depending on if cluster or region
%only one path per truck from depots to households
IEC2 = [];
IEC2_rhs = ones(K,1); %answers

add_this_rhs = zeros(1, DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs

Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
%%%%%%%
Xtr(1:D,D+1:D+NHcount) = 1; %initialize possible routes for trucks from depot
to households
%%%%%%%
Xtr = Xtr';
Xtr  =  Xtr(flag_mat_tr);  %index  against  flag  matrix  to  remove  impossible
solutions
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for k = 1:K %iterate through the number of trucks
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    temp2 = [];
    temp2 = [temp1, add_this_rhs]; %add unused decision variables
    IEC2 = [IEC2;temp2]; %vertically concatenate constraints for each truck
end

%EC3 if truck leaves depot, it must return to the depot after dropping off
%at recycling facility

EC3 = [];
```

```
EC3_rhs = zeros(K,1);

add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for d = 1:D
    Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
    %%%%%%%
    Xtr(d,D+1:D+NHcount) = 1; %initialize possible routes for trucks from depot
to households
    if NH_case == 1
        Xtr(:,D+NHcount+d) = -1; %initialize possible routes from households to
depot
    else
        Xtr(D+NHcount+1:D+NHcount+F,D+NHcount+F+d) = -1; %initialize possible
routes from recycling facilities to depot
    end
    %%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

    for k = 1:K %iterate through the number of trucks
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [];
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        EC3 = [EC3;temp2]; %vertically concatenate constraints for each truck
    end
end

%EC5 if truck goes to household, it must leave the household
EC4 = [];
EC4_rhs = zeros(K*NHcount,1);

add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for n = 1:NHcount %make parallel
    Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
    %%%%%%%%
    Xtr(1:D+NHcount,D+n) = 1; %Initialize route from depot or household to
household
    if NH_case == 1
        Xtr(D+n,D+1:D+NHcount+D) = -1; %initialize route from household to
household
    else
        Xtr(D+n,D+1:D+NHcount+F) = -1; %initialize route from household to
household or recycling facility
    end
    %%%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr);

    for k = 1:K
```

```
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        EC4 = [EC4;temp2]; %vertically concatenate constraints for each truck
    end
end

%EC5 if vehicle goes to recyling facility, it must leave same recycling
%facility

EC5 = [];
if NH_case == 1
    EC5_rhs = [];
else
    EC5_rhs = zeros(F*K,1);

    add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused
DVs
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

    for f = 1:F
        Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
        %%%%%%%
        Xtr(D+1:D+NHcount,D+NHcount+f) = 1; %initialize possible routes from
households to recycling facilities
        Xtr(D+NHcount+f,D+NHcount+F+1:D+NHcount+F+D) = -1; %initialize routes
from recycling facility to depot
        %%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

        for k = 1:K %iterate through the number of trucks
            temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x
DVs
            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp2 = [temp1, add_this_rhs]; %add unused decision variables
            EC5 = [EC5;temp2]; %vertically concatenate constraints for each
truck
        end
    end
end
%EC6 waste from household must go somewhere. Each house can only go
%somewhere once by one truck. remove for commercial?
EC6 = [];
EC6_rhs = ones(NHcount,1);

add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for i = 1:NHcount %make parallel
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

    Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
    %%%%%%%
```

```
    Xtr(D+i,:) = 1;
    %%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

    for k = 1:K
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    end
    temp2 = [temp1, add_this_rhs]; %add unused decision variables
    EC6 = [EC6;temp2]; %vertically concatenate constraints for each truck
end

%IC7 ensure continuity of tours and subtours using capcaity limitations
IC7 = [];
IC7_rhs = ones(NHcount^2*K,1)*Q; %capacity of truck minus pickup size

% for i = 1:N
% qmat(i,:) = qvec;
% end
% qmat = [qvec;qmat]';
%
% qflag = ~eye(N);
% qflag = logical([ones(1,N);qflag]);
% sIC7_rhs = qmat(qflag');
% IC7_rhs = sIC7_rhs;
% if K >= 2
%     for k = 2:K
%         IC7_rhs = [IC7_rhs;sIC7_rhs];
%     end
% end
%
% IC7_rhs = Q-IC7_rhs;
% IC7_rhs(:) = Q;

add_this_rhs = zeros(1,DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x);
temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y DVs

for h = 1:sDV_y %needs to include depot %make parallel
    for i = D+1:sDV_y
        if h==i
            continue
        end
        Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
        Ytr = zeros(sDV_y,1); %matrix of zeros to fill in
        %%%%%%%%
        Xtr(h,i) = Q + qvec(i-1); %if q is specific to household and qvec is
used
%         Xtr(h,i) = Q + q; %need to figure out how to add proper q for speicifc
houeshold if not homogeneous. will be case in commercial example.
        Ytr(h) = 1;
        Ytr(i) = -1;
        %%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);
```

```
        for k = 1:K
            temp1 = zeros(1,DV_x);
            temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y
DVs

            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';

            temp2  =  [temp1,  temp1b,  add_this_rhs];  %add  unused  decision
variables
            IC7  =  [IC7;temp2]; %vertically concatenate constraints for each
truck
        end
    end
end

%EC8 If truck leaves a household, it must bring waste with it
IC8 = [];

add_this_mid = zeros(1,DV_B); %time variable placeholders
add_this_rhs = zeros(1,DV_v); %matrix of zeros for unused DVs

temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y DVs
temp1c = zeros(1,DV_w);

if NH_case == 1
    F = K;
    IC8_rhs = ones(NHcount*K,1)*Q;
else
    IC8_rhs = ones(F*NHcount*K,1)*Q;
end

for f = 1:F
    for i = D+1:sDV_y %make parallel
        Xtr = zeros(size(t_matrix)); %matrix of zeros to fill in
        Ytr = zeros(sDV_y,1); %matrix of zeros to fill in
        Wtr = zeros(sDV_w,1);
        %%%%%%%%
        Xtr(i,D+NHcount+f) = Q;
        Ytr(i) = 1;
        Wtr(f) = -1;
        %%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);
        for k = 1:K
            temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x
DVs
            temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y
DVs
            temp1c = zeros(1,DV_w);

            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';
            temp1c((k-1)*sDV_w+1:k*sDV_w) = Wtr';
```

```
            temp2 = [temp1, temp1b, add_this_mid, temp1c, add_this_rhs]; %add
unused decision variables
            IC8 = [IC8;temp2]; %vertically concatenate constraints for each
truck
        end
    end
end

if NH_case == 1
    F = 0;
end

%IC9 Only one delivery per truck
IC9 = [];
IC9_rhs = ones(K,1)*Q;

add_this_lhs = zeros(1,DV_x+DV_y+DV_B);
add_this_rhs = zeros(1,DV_v);

Wtr = zeros(sDV_w,1); %matrix of zeros to fill in]
%%%%%%%%%
if NH_case == 1
    Wtr(1:K) = 1; %Dropoff to recycling facilities
else
    Wtr(1:F) = 1; %Dropoff to recycling facilities
end
%%%%%%%%

temp1 = zeros(1,DV_w); %create temp matrix of zeros the size of z DVs

for k = 1:K
    temp1 = zeros(1,DV_w); %create temp matrix of zeros the size of z DVs
    temp1((k-1)*sDV_w+1:k*sDV_w) = Wtr';

    temp2 = [add_this_lhs, temp1, add_this_rhs]; %add unused decision variables
    IC9 = [IC9;temp2]; %vertically concatenate constraints for each truck
end

%EC10 the volume of waste delivered must equal the volume of waste
%generated from all the households

EC10 = [];
EC10_rhs = sum(qvec);

add_this_lhs = zeros(1,DV_x+DV_y+DV_B);
add_this_rhs = zeros(1,DV_v);
%%%%%%%%
if NH_case == 1
    Wtr = ones(K,1);
else
    Wtr = ones(F*K,1);
end
```

```
%%%%%%%%

EC10 = [add_this_lhs, Wtr', add_this_rhs];


%EC11 count the total waste sent to each recycling facility
EC11 = [];
if NH_case == 1
    EC11_rhs = [];
else
    EC11_rhs = zeros(F,1);

    add_this_lhs = zeros(1,DV_x+DV_y+DV_B);

    for f = 1:F
        temp1 = [];
        Vtr = zeros(sDV_v,1);
        Wtr = zeros(sDV_w,1);
        %%%%%%%%
        Wtr(f) = -1;
        Vtr(f) = 1;
        %%%%%%%%
            for k = 1:K
                temp1 = [temp1,Wtr'];
            end
%             temp1((k-1)*sDV_w+1:k*sDV_w) = Wtr';
%             temp1b((k-1)*sDV_v+1:k*sDV_v) = Vtr';

        temp2 = [add_this_lhs, temp1, Vtr']; %add unused decision variables
        EC11 = [EC11;temp2]; %vertically concatenate constraints for each truck
    end
end


%IC12 Amount delivered to recycling facility cannot exceed a maximum
%capacity

IC12 = [];
if NH_case == 1
    IC12_rhs = [];
else
    IC12_rhs = ones(F,1)*P_hi; %Change if capcities of facilities are different

    add_this_lhs = zeros(1,DV_x+DV_y+DV_B+DV_w);

    for f = 1:F
        Vtr = zeros(F,1);
        %%%%%%%
        Vtr(f) = 1;
        %%%%%%%
        temp2 = [add_this_lhs, Vtr'];
        IC12 = [IC12;temp2];
    end
end
```

```
IC13 = [];
if NH_case == 1
    IC13_rhs = [];
else
    IC13_rhs = ones(F,1)*-P_lo; %Change if capcities of facilities are different

    add_this_lhs = zeros(1,DV_x+DV_y+DV_B+DV_w);

    for f = 1:F
        Vtr = zeros(F,1);
        %%%%%%%
        Vtr(f) = -1;
        %%%%%%%
        temp2 = [add_this_lhs, Vtr'];
        IC13 = [IC13;temp2];
    end
end


%IC14 Total time traveled in the path must be less than or equal to time
%allotted for each vehicle

EC14 = [];
EC14_rhs = zeros(K,1);

add_this_lhs = zeros(1,DV_y);
add_this_rhs = zeros(1,DV_w+DV_v);

temp1 = zeros(1,sDV_x); %create temp matrix of zeros the size of x DVs
temp1b = zeros(1,sDV_B); %create temp matrix of zeros the size of x DVs

%%%%%%%
Xtr = Bm_matrix;
Btr = -1;
%%%%%%%
Xtr = Xtr';
Xtr = Xtr(flag_mat_tr);

for k = 1:K %iterate through the number of trucks
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
    temp1b = zeros(1,DV_B); %create temp matrix of zeros the size of x DVs

    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    temp1b((k-1)*sDV_B+1:k*sDV_B) = Btr';

    temp2 = [temp1, add_this_lhs, temp1b, add_this_rhs]; %add unused decision
variables
    EC14 = [EC14;temp2]; %vertically concatenate constraints for each truck
end

%EC15 time traveled (B) for each truck can't be higher than the time limit (a
working day)
```

```
IC15 = [];
IC15_rhs = ones(K,1)*B_lim;

add_this_lhs = zeros(1,DV_x+DV_y);
add_this_rhs = zeros(1,DV_w+DV_v);

Btr = zeros(1,DV_B);

for k = 1:K %iterate through the number of trucks
    Btr = zeros(DV_B,1);

    %%%%%%%
    Btr(k) = 1;
    %%%%%%%

    temp2 = [add_this_lhs, Btr', add_this_rhs]; %add unused decision variables
    IC15 = [IC15;temp2]; %vertically concatenate constraints for each truck
end

% ECtest = zeros(1,DV_tot);
% ECtest_rhs = 2;
% ECtest(54) = 1;

%Build A, b, Aeq, and beq matrices. Also build upper and lower bounds.

if NH_case == 1
%     A = [IC7; IC8; IC9; IC15];
%     b = [IC7_rhs; IC8_rhs; IC9_rhs; IC15_rhs];
%
%     Aeq = [IEC2; EC3; EC4; EC6; EC10; EC14];
%     beq = [IEC2_rhs; EC3_rhs; EC4_rhs; EC6_rhs; EC10_rhs; EC14_rhs];

    A = [IC7; IC8; IC9];
    b = [IC7_rhs; IC8_rhs; IC9_rhs];

    Aeq = [IEC2; EC3; EC4; EC5; EC6; EC10; EC14];
    beq = [IEC2_rhs; EC3_rhs; EC4_rhs; EC5_rhs; EC6_rhs; EC10_rhs; EC14_rhs];
else
    A = [IEC2; IC7; IC8; IC9; IC12; IC13; IC15];
    b = [IEC2_rhs; IC7_rhs; IC8_rhs; IC9_rhs; IC12_rhs; IC13_rhs; IC15_rhs];

    Aeq = [EC3; EC4; EC5; EC6; EC10; EC11; EC14];
    beq = [EC3_rhs; EC4_rhs; EC5_rhs; EC6_rhs; EC10_rhs; EC11_rhs; EC14_rhs];
end

lb(1:DV_tot) = 0;
ub(1:DV_x) = 1;
ub(DV_x+1:DV_x+DV_y) = sum(qvec);
ub(DV_x+DV_y+1:DV_x+DV_y+DV_B) = B_lim;
ub(DV_x+DV_y+DV_B+1:DV_x+DV_y+DV_B+DV_w) = Q;
if NH_case ~= 1
    ub(DV_x+DV_y+DV_B+DV_w+1:DV_x+DV_y+DV_B+DV_w+DV_v) = P_hi;
end
```

```
x0 = [];

sostype=[];
sosind=[];
soswt=[];
codetime = toc;

[x,fval,exitflag,output]                                                    =
cplexmilp(obj,A,b,Aeq,beq,sostype,sosind,soswt,lb,ub,ctype,x0,opts);


x
fval
exitflag
output

%%%%%%Output code
% Index the results

%+++++++++++++++++
%For Matlab 2016a because we dont update our school computers

J = I;

if NH_case == 1
    for i = 1:size(I,1) %Matrix already transposed for flag index
        if i <= D
            I(:,i) = cellstr('D');
        elseif i <= D+NHcount
            I(:,i) = cellstr(sprintf('N%u',i-D));
        else
            I(:,i) = cellstr('Dpr');
        end
    end

    for j = 1:size(J,1) %Matrix already transposed for flag index
        if j <= D
            J(j,:) = cellstr('D');
        elseif j <= D+NHcount
            J(j,:) = cellstr(sprintf('N%u',j-D));
        else
            J(j,:) = cellstr('Dpr');
        end
    end

else

    for i = 1:size(I,1) %Matrix already transposed for flag index
        if i <= D
            I(:,i) = cellstr('D');
        elseif i <= D+NHcount
            I(:,i) = cellstr(sprintf('N%u',i-D));
        elseif i <= D+NHcount+F
```

```
            I(:,i) = cellstr(sprintf('F%u',i-(D+NHcount)));
        else
            I(:,i) = cellstr('Dpr');
        end
    end

    for j = 1:size(J,1) %Matrix already transposed for flag index
        if j <= D
            J(j,:) = cellstr('D');
        elseif j <= D+NHcount
            J(j,:) = cellstr(sprintf('N%u',j-D));
        elseif j <= D+NHcount+F
            J(j,:) = cellstr(sprintf('F%u',j-(D+NHcount)));
        else
            J(j,:) = cellstr('Dpr');
        end
    end
end



for i = 1:D+NHcount
    Yl(i) = cellstr(sprintf('y%u',i-1));
end
sY = Yl;

for k = 1:K
    Wl(k) = cellstr(sprintf('w%u',k));
end
sW = Wl;

for bc = 1:DV_B
    Bl(bc) = cellstr(sprintf('B%u',bc));
end
Bl = Bl';

if NH_case ~= 1
    for f = 1:F
    Vl(f) = cellstr(sprintf('v%u',f));
    end
    sV = Vl;
end

NHValueLab = cellstr('InterNH cost ($)');

flag_mat_tr_vehc = flag_mat_tr;
if K >= 2
    for k = 2:K
        flag_mat_tr_vehc = [flag_mat_tr_vehc,flag_mat_tr];
        I = [I,I];
        J = [J,J];
        Yl = [Yl;sY];
%         Wl = [Wl;sW];
    end
end
```

```
[J2, I2] = find(flag_mat_tr_vehc);

%create matrices of results for individual generator cost results table
MatResults_x = zeros(size(flag_mat_tr_vehc));
idx = sub2ind(size(MatResults_x),J2,I2);
MatResults_x(idx) = x(1:DV_x);
MatResults_x_log = MatResults_x > 0.9;

Solution.Departx = I(MatResults_x_log);
Solution.Arrivex = J(MatResults_x_log);
Solution.Answerx = MatResults_x(MatResults_x_log);
Solution.PickupLab = Yl;
Solution.CargoQuant = x(DV_x+1:DV_x+DV_y);
Solution.DropTimeLab = Bl;
Solution.DropTimeQuant = x(DV_x+DV_y+1:DV_x+DV_y+DV_B);
Solution.DropLab = Wl';
Solution.CargoDrop = x(DV_x+DV_y+DV_B+1:DV_x+DV_y+DV_B+DV_w);

if NH_case ~= 1
    Solution.FacilQuantLab = Vl;
    Solution.FacilQuantVal = x(DV_x+DV_y+DV_B+DV_w+1:DV_tot);
end

Solution.NHVLab = NHValueLab;
Solution.Value = fval;
Solution.totVlab = cellstr('Total Cost ($)');
Solution.totValue = fval + sum(NHNetData.Cost);

%     SetupLabs     =     [cellstr('CodeTime(s)');     cellstr('SolverTime(s)');
cellstr('Depots');    cellstr('Households');    cellstr('Collect.    Trucks');
cellstr('HH  Gen.  Quantity');  cellstr('Truck  Capacity');  cellstr('Solution
Labels')];
%   SetupVals   =   [Solution.Value;   num2cell(toc);   num2cell(output.time);
cellstr(sprintf('%u',D)); cellstr(sprintf('%u',N)); cellstr(sprintf('%u',K));
cellstr(sprintf('%u',q));     cellstr(sprintf('%u',Q));     cellstr('Solution
Values')];

if NH_case == 1
    RegSetupLabs    =    [cellstr('CodeTime(s)');    cellstr('SolverTime(s)');
cellstr('Depots');    cellstr('Households');    cellstr('Collect.    Trucks');
cellstr('HH  Gen.  Quantity');  cellstr('Truck  Capacity');  cellstr('Solution
Labels')];
    RegSetupVals      =      [num2cell(codetime);      num2cell(output.time);
cellstr(sprintf('%u',D));                     cellstr(sprintf('%u',NHcount));
cellstr(sprintf('%u',K));  cellstr(sprintf('%u',q)); cellstr(sprintf('%u',Q));
cellstr('Solution Values')];
    RegLabels    =    [RegSetupLabs;    Solution.Departx;    Solution.PickupLab;
Solution.DropTimeLab; Solution.DropLab; Solution.NHVLab];
    RegValues           =           [RegSetupVals;           Solution.Arrivex;
num2cell(Solution.CargoQuant);               num2cell(Solution.DropTimeQuant);
num2cell(Solution.CargoDrop); num2cell(Solution.Value)];
else
    RegSetupLabs    =    [cellstr('CodeTime(s)');    cellstr('SolverTime(s)');
cellstr('Depots'); cellstr('Neighborhoods'); cellstr('Recycling Facilities');
cellstr('Collect.  Trucks');  cellstr('Op.  Cost  ($/hr)');  cellstr('HH  Gen.
Quantity'); cellstr('Truck Capacity'); cellstr('Solution Labels')];
```

```
    RegSetupVals = [num2cell(codetime); num2cell(output.time); num2cell(D);
num2cell(NHcount); num2cell(F); num2cell(K); num2cell(C_hr); num2cell(q);
num2cell(Q); cellstr('Solution Values')];
    RegLabels = [RegSetupLabs; Solution.Departx; Solution.PickupLab;
Solution.DropTimeLab; Solution.DropLab; Solution.FacilQuantLab;
Solution.NHVLab; Solution.totVlab];
    RegValues = [RegSetupVals; Solution.Arrivex;
num2cell(Solution.CargoQuant); num2cell(Solution.DropTimeQuant);
num2cell(Solution.CargoDrop); num2cell(Solution.FacilQuantVal);
num2cell(Solution.Value); num2cell(Solution.totValue)];

end

NHLabels = [cellstr('Neighborhood #'); cellstr('# of Households');
cellstr('Route Time (min)'); cellstr('Food Waste Generation (lbs)');
cellstr('Route Cost ($)')];
NHValues = [num2cell(NHNetData.NH_Select'); num2cell(NHNetData.HH_Select');
num2cell(NHNetData.RouteTime'); num2cell(NHNetData.qgen');
num2cell(NHNetData.Cost')];

emptynum = size(RegLabels,1)-size(NHLabels,1);

emptylabels = cell(emptynum,size(NHLabels,2));
emptyvalues = cell(emptynum,size(NHValues,2));

NHLabels = [NHLabels; emptylabels];
NHValues = [NHValues; emptyvalues];

aaSolutionTable = table(RegLabels, RegValues, NHLabels, NHValues);
```

Neighborhood routing code

```
function [NHNetData]... %other inputs
    = NH_ResCollectRoute_v6...

(NHcount,NH_Select,NHNetData,HHref_data_trim,NH_ODs_trim,Q,q,m,C_hr,B_lim,P_l
o,P_hi)

% addpath('C:\Program Files\IBM\ILOG\CPLEX_Studio1271\cplex\matlab\x64_win64')

opts2 = cplexoptimset('Display','on');

%function for routing individual neighborhoods
%Will Armington
%Residential collection and routing

for NH = NH_Select
delete *.log

clearvars -except NH NHNetData NHcount NHNetData HHdata_pick NH_ODs_trim opts2
Q q m C_hr B_lim P_lo P_hi

 tic %start timer
%inputs

idxNH = NHNetData.NH_Select == NH;

%input case number for cluster vs region
NH_case = 1;

N = NHNetData.HH_Select(idxNH); %number of households
D = 1; %Number of depots

%general inputs, comment out if specific inputs are used from spreadsheet file
if NH_case == 1
    F = 0; %number of recycling facilities
    K = 1; %number of collection trucks
else
    F = 2; %number of recycling facilities
    K = 2; %number of collection trucks
end

% Q = 2000; %lbs. Capacity of collection trucks if all the same type
% q = 1; %lbs. food waste quantity generated by households.
% g = 25; %mph. average speed between nodes. Need conversion to meters/second
% m = 30; %seconds. Pickup time at households
% B_lim = 8*60*60; %seconds. Time limit in a work day
% P_lo = 0; %lbs. recycling facility quota
% P_hi = 1000; %lbs. recycling facility capacity
% C_hr = 100; %operating cost $/hr

% avgspd_ms = g*0.44704; %Average travel speed of collection vehicle (mi/hr)
C_m = C_hr/60; %convert to $/minute;
```

```
qsum = N * q;

% idxHH = idxClust == clust; %index of households in the specific cluster
% idxNet = logical([1;idxHH]);
%
%  qvec  =  HHref_data_trim.EstFWgen_lb_wk(idxHH);  %/2000;  %convert  lbs  of
generation to tons
% B_matrix = NH_ODs_trim{NH}; %/ 1609.34; %convert distance of meters to miles

%Add depot prime to end of cost matrix
if NH_case == 1
    B_matrix = zeros(D+N+1);
    B_matrix(D+1:D+N,D+1:D+N) = NH_ODs_trim{NH};
else
    B_matrix = zeros(D+N+F+1);
%     B_matrix(D+1:D+N,D+1:D+N) = NH_ODs_trim{NH};
%     B_matrix(D+N+F+1,:) = B_matrix(1,:);
%     B_matrix(:, D+N+F+1) = B_matrix(:,1);
end

%create time matrix for links
% B_matrix = B_matrix ./ avgspd_ms; %convert distance matrix to time matrix
Bm_matrix = B_matrix;
Bm_matrix(1:D+N,D+1:D+N)  =  B_matrix(1:D+N,D+1:D+N)  +  m;  %add  pickup  time  to
destination node
C_matrix = Bm_matrix .* C_m; %convert total time matrix to monetary cost


%Cost matrix order should be D, N, F, D'

%Single decision variable counts
if NH_case == 1
    sDV_y = D+N; %household pickup decisions for each truck
    sDV_B = 1; %Recycling facility drop-off decisions for each truck
    sDV_w = 1; %Volume of waste delivered to recycling facility for each truck
    sDV_v = 0; %total volume waste delivered to recycling facility
else
    sDV_y = D+N; %household pickup decisions for each truck
    sDV_B = 1; %Recycling facility drop-off decisions for each truck
    sDV_w = F; %Volume of waste delivered to recycling facility for each truck
    sDV_v = F; %total volume waste delivered to recycling facility
end

%++++++++++++++++++++
%For matlab 2016a when creating output table because we dont update your school
computers
sz = D*2+N+F;
I = cell(sz);
Yl = cell(sDV_y,1);
Bl = cell(sDV_B,1);
Wl = cell(sDV_w,1);
Vl = cell(sDV_v,1);
%++++++++++++++++++++

%total decision veriable counts
```

```
DV_y = sDV_y*K; %household pickup decisions for each truck
DV_B = sDV_B*K; %Time count for truck routes
DV_w = sDV_w*K; %Volume of waste delivered to recycling facility for each truck
DV_v = sDV_v; %total number of volume waste variables delivered to recycling
facility

%flag matrix to remove impossible solutions
flag_mat = ~eye(size(B_matrix)); %no solutions along diagonal
flag_mat(:,1) = 0; %No places go back to depot
flag_mat(D+N+F+1:D+N+F+D,:) = 0; %Depot prime does not go anywhere
flag_mat(D,D+N+D) = 0; %Depot cannot go to depot prime

if NH_case ~= 1
        flag_mat(D+N+1:D+N+F,1:D+N+F) = 0; %recycling facilities only go to
depot prime
        flag_mat(1:D,D+N+1:D+N+F) = 0; %Depot cannot go to recycling facilities
        flag_mat(D:D+N,D+N+F+1:D+N+F+D) = 0; %households cannot go to depot
prime
end

flag_mat_tr = flag_mat';

%create costing for objectives
C_mat_tr = C_matrix';
sc_x = C_mat_tr(flag_mat_tr);
sDV_x = size(sc_x,1); %create single number of x decision variables ebfore
vehicles are accounted for
c_x = sc_x;

if K > 1
    for k = 2:K
        c_x = [c_x;sc_x];
    end
end

c_y = zeros(DV_y,1);
c_B = zeros(DV_B,1);
c_w = zeros(DV_w,1);
c_v = zeros(DV_v,1);
obj = [c_x;c_y;c_B;c_w;c_v];

%Decision Veriable count part 2
DV_x = sDV_x*K; %total possible arcs

DV_tot = DV_x+DV_y+DV_B+DV_w+DV_v; %total possible decision variables

%CPLEX settings
% ctype = [];
ctype(1:DV_x) = {'B'};
ctype(DV_x+1:DV_tot) = {'C'};
ctype = char(ctype)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Constraints
```

```
%IC2 or EC2 depending on if cluster or region
%only one path per truck from depots to households
IEC2 = [];
IEC2_rhs = ones(K,1); %answers

add_this_rhs = zeros(1, DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs

Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
%%%%%%%
Xtr(1:D,D+1:D+N) = 1; %initialize possible routes for trucks from depot to
households
%%%%%%%
Xtr = Xtr';
Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for k = 1:K %iterate through the number of trucks
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    temp2 = [];
    temp2 = [temp1, add_this_rhs]; %add unused decision variables
    IEC2 = [IEC2;temp2]; %vertically concatenate constraints for each truck
end

%EC4 if truck leaves depot, it must return to the depot after dropping off
%at recycling facility

EC3 = [];
EC3_rhs = zeros(K,1);

add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for d = 1:D
    Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
    %%%%%%%
    Xtr(d,D+1:D+N) = 1; %initialize possible routes for trucks from depot to
households
    if NH_case == 1
        Xtr(:,D+N+d) = -1; %initialize possible routes from households to
depotpr
    else
        Xtr(D+N+1:D+N+F,D+N+F+d) = -1; %initialize possible routes from
recycling facilities to depotpr
    end
    %%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

    for k = 1:K %iterate through the number of trucks
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [];
```

```
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        EC3 = [EC3;temp2]; %vertically concatenate constraints for each truck
    end
end

%EC5 if truck goes to household, it must leave the household
EC4 = [];
EC4_rhs = zeros(K*N,1);

add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for n = 1:N %make parallel
    Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
    %%%%%%%%
    Xtr(1:D+N,D+n) = 1; %Initialize route from depot or household to household
    if NH_case == 1
        Xtr(D+n,D+1:D+N+D) = -1; %initialize route from household to household
    else
        Xtr(D+n,D+1:D+N+F) = -1; %initialize route from household to household
or recycling facility
    end
    %%%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr);

    for k = 1:K
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        EC4 = [EC4;temp2]; %vertically concatenate constraints for each truck
    end
end

%EC5 if vehicle goes to recyling facility, it must leave same recycling
%facility

EC5 = [];
if NH_case == 1
    EC5_rhs = [];
else
    EC5_rhs = zeros(F*K,1);

    add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused
DVs
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

    for f = 1:F
        Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
        %%%%%%%
        Xtr(D+1:D+N,D+N+f) = 1; %initialize possible routes from households to
recycling facilities
        Xtr(D+N+f,D+N+F+1:D+N+F+D) = -1; %initialize routes from recycling
facility to depot
        %%%%%%%
```

```
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

        for k = 1:K %iterate through the number of trucks
            temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x
DVs
            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp2 = [temp1, add_this_rhs]; %add unused decision variables
            EC5 = [EC5;temp2]; %vertically concatenate constraints for each
truck
        end
    end
end
%EC6 waste from household must go somewhere. Each house can only go
%somewhere once by one truck.
EC6 = [];
EC6_rhs = ones(N,1);

add_this_rhs = zeros(1,DV_y+DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for i = 1:N %make parallel
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

    Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
    %%%%%%%
    Xtr(D+i,:) = 1;
    %%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

    for k = 1:K
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    end
    temp2 = [temp1, add_this_rhs]; %add unused decision variables
    EC6 = [EC6;temp2]; %vertically concatenate constraints for each truck
end

%IC7 ensure continuity of tours and subtours using capcaity limitations
IC7 = [];
IC7_rhs = ones(N^2*K,1)*Q; %capacity of truck minus pickup size

% for i = 1:N
% qmat(i,:) = qvec;
% end
% qmat = [qvec;qmat]';
%
% qflag = ~eye(N);
% qflag = logical([ones(1,N);qflag]);
% sIC7_rhs = qmat(qflag');
% IC7_rhs = sIC7_rhs;
% if K >= 2
%     for k = 2:K
```

```
%         IC7_rhs = [IC7_rhs;sIC7_rhs];
%     end
% end
%
% IC7_rhs = Q-IC7_rhs;
% IC7_rhs(:) = Q;

add_this_rhs = zeros(1,DV_B+DV_w+DV_v); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x);
temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y DVs

for h = 1:sDV_y %needs to include depot %make parallel
    for i = D+1:sDV_y
        if h==i
            continue
        end
        Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
        Ytr = zeros(sDV_y,1); %matrix of zeros to fill in
        %%%%%%%%
        Xtr(h,i) = Q + q; %if q is universal
%         Xtr(h,i) = Q + qvec(i-1); %if q is specific to household and qvec is
used
        Ytr(h) = 1;
        Ytr(i) = -1;
        %%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);
        for k = 1:K
            temp1 = zeros(1,DV_x);
            temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y
DVs

            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';

            temp2  =  [temp1,  temp1b,  add_this_rhs];  %add  unused  decision
variables
            IC7  =  [IC7;temp2];  %vertically  concatenate  constraints  for  each
truck
        end
    end
end

%EC8 If truck leaves a household, it must bring waste with it
IC8 = [];

add_this_mid = zeros(1,DV_B); %time variable placeholders
add_this_rhs = zeros(1,DV_v); %matrix of zeros for unused DVs

temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y DVs
temp1c = zeros(1,DV_w);

if NH_case == 1
    F = K;
```

```
    IC8_rhs = ones(N*K,1)*Q;
else
    IC8_rhs = ones(F*N*K,1)*Q;
end

for f = 1:F
    for i = D+1:sDV_y %make parallel
        Xtr = zeros(size(B_matrix)); %matrix of zeros to fill in
        Ytr = zeros(sDV_y,1); %matrix of zeros to fill in
        Wtr = zeros(sDV_w,1);
        %%%%%%%%
        Xtr(i,D+N+f) = Q;
        Ytr(i) = 1;
        Wtr(f) = -1;
        %%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);
        for k = 1:K
            temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x
DVs
            temp1b = zeros(1,DV_y); %create temp matrix of zeros the size of y
DVs
            temp1c = zeros(1,DV_w);

            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';
            temp1c((k-1)*sDV_w+1:k*sDV_w) = Wtr';

            temp2 = [temp1, temp1b, add_this_mid, temp1c, add_this_rhs]; %add
unused decision variables
            IC8 = [IC8;temp2]; %vertically concatenate constraints for each
truck
        end
    end
end

if NH_case == 1
    F = 0;
end

%IC9 Only one delivery per truck
IC9 = [];
IC9_rhs = ones(K,1)*Q;

add_this_lhs = zeros(1,DV_x+DV_y+DV_B);
add_this_rhs = zeros(1,DV_v);

Wtr = zeros(sDV_w,1); %matrix of zeros to fill in]
%%%%%%%%%
if NH_case == 1
    Wtr(1:K) = 1; %Dropoff to recycling facilities
else
    Wtr(1:F) = 1; %Dropoff to recycling facilities
end
%%%%%%%%
```

```
temp1 = zeros(1,DV_w); %create temp matrix of zeros the size of z DVs

for k = 1:K
    temp1 = zeros(1,DV_w); %create temp matrix of zeros the size of z DVs
    temp1((k-1)*sDV_w+1:k*sDV_w) = Wtr';

    temp2 = [add_this_lhs, temp1, add_this_rhs]; %add unused decision variables
    IC9 = [IC9;temp2]; %vertically concatenate constraints for each truck
end

%EC10 the volume of waste delivered must equal the volume of waste
%generated from all the households

EC10 = [];
EC10_rhs = qsum; %if q is used for every house
% EC10_rhs = sum(qvec); % if qvec used

add_this_lhs = zeros(1,DV_x+DV_y+DV_B);
add_this_rhs = zeros(1,DV_v);
%%%%%%%%
if NH_case == 1
    Wtr = ones(K,1);
else
    Wtr = ones(F*K,1);
end

%%%%%%%%

EC10 = [add_this_lhs, Wtr', add_this_rhs];


%EC11 count the total waste sent to each recycling facility
EC11 = [];
if NH_case == 1
    EC11_rhs = [];
else
    EC11_rhs = zeros(F,1);

    add_this_lhs = zeros(1,DV_x+DV_y+DV_B);

    for f = 1:F
        temp1 = [];
        Vtr = zeros(sDV_v,1);
        Wtr = zeros(sDV_w,1);
        %%%%%%%%
        Wtr(f) = -1;
        Vtr(f) = 1;
        %%%%%%%%
            for k = 1:K
                temp1 = [temp1,Wtr'];
            end
%             temp1((k-1)*sDV_w+1:k*sDV_w) = Wtr';
%             temp1b((k-1)*sDV_v+1:k*sDV_v) = Vtr';
```

```
        temp2 = [add_this_lhs, temp1, Vtr']; %add unused decision variables
        EC11 = [EC11;temp2]; %vertically concatenate constraints for each truck
    end
end


%IC12 Amount delivered to recycling facility cannot exceed a maximum
%capacity

IC12 = [];
if NH_case == 1
    IC12_rhs = [];
else
    IC12_rhs = ones(F,1)*P_hi; %Change if capcities of facilities are different

    add_this_lhs = zeros(1,DV_x+DV_y+DV_B+DV_w);

    for f = 1:F
        Vtr = zeros(F,1);
        %%%%%%%
        Vtr(f) = 1;
        %%%%%%%
        temp2 = [add_this_lhs, Vtr'];
        IC12 = [IC12;temp2];
    end
end


IC13 = [];
if NH_case == 1
    IC13_rhs = [];
else
    IC13_rhs = ones(F,1)*-P_lo; %Change if capcities of facilities are different

    add_this_lhs = zeros(1,DV_x+DV_y+DV_B+DV_w);

    for f = 1:F
        Vtr = zeros(F,1);
        %%%%%%%
        Vtr(f) = -1;
        %%%%%%%
        temp2 = [add_this_lhs, Vtr'];
        IC13 = [IC13;temp2];
    end
end


%IC14 Total time traveled in the path must be less than or equal to time
%allotted for each vehicle

EC14 = [];
EC14_rhs = zeros(K,1);
```

```
add_this_lhs = zeros(1,DV_y);
add_this_rhs = zeros(1,DV_w+DV_v);

temp1 = zeros(1,sDV_x); %create temp matrix of zeros the size of x DVs
temp1b = zeros(1,sDV_B); %create temp matrix of zeros the size of x DVs

%%%%%%%
Xtr = Bm_matrix;
Btr = -1;
%%%%%%%
Xtr = Xtr';
Xtr = Xtr(flag_mat_tr);

for k = 1:K %iterate through the number of trucks
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
    temp1b = zeros(1,DV_B); %create temp matrix of zeros the size of x DVs

    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    temp1b((k-1)*sDV_B+1:k*sDV_B) = Btr';

    temp2 = [temp1, add_this_lhs, temp1b, add_this_rhs]; %add unused decision
variables
    EC14 = [EC14;temp2]; %vertically concatenate constraints for each truck
end

%EC15 time traveled (B) for each truck can't be higher than the time limit (a
working day)
IC15 = [];
IC15_rhs = ones(K,1)*B_lim;

add_this_lhs = zeros(1,DV_x+DV_y);
add_this_rhs = zeros(1,DV_w+DV_v);

Btr = zeros(1,DV_B);

for k = 1:K %iterate through the number of trucks
    Btr = zeros(DV_B,1);

    %%%%%%%
    Btr(k) = 1;
    %%%%%%%

    temp2 = [add_this_lhs, Btr', add_this_rhs]; %add unused decision variables
    IC15 = [IC15;temp2]; %vertically concatenate constraints for each truck
end

% ECtest = zeros(1,DV_tot);
% ECtest_rhs = 2;
% ECtest(54) = 1;

%Build A, b, Aeq, and beq matrices. Also build upper and lower bounds.

if NH_case == 1
%     A = [IC7; IC8; IC9; IC15];
```

```
%     b = [IC7_rhs; IC8_rhs; IC9_rhs; IC15_rhs];
%
%     Aeq = [IEC2; EC3; EC4; EC6; EC10; EC14];
%     beq = [IEC2_rhs; EC3_rhs; EC4_rhs; EC6_rhs; EC10_rhs; EC14_rhs];

    A = [IC7; IC8; IC9];
    b = [IC7_rhs; IC8_rhs; IC9_rhs];

    Aeq = [IEC2; EC3; EC4; EC5; EC6; EC10; EC14];
    beq = [IEC2_rhs; EC3_rhs; EC4_rhs; EC5_rhs; EC6_rhs; EC10_rhs; EC14_rhs];
else
    A = [IEC2; IC7; IC8; IC9; IC12; IC13; IC15];
    b = [IEC2_rhs; IC7_rhs; IC8_rhs; IC9_rhs; IC12_rhs; IC13_rhs; IC15_rhs];

    Aeq = [EC3; EC4; EC5; EC6; EC10; EC11; EC14];
    beq = [EC3_rhs; EC4_rhs; EC5_rhs; EC6_rhs; EC10_rhs; EC11_rhs; EC14_rhs];
end

lb(1:DV_tot) = 0;
ub(1:DV_x) = 1;
ub(DV_x+1:DV_x+DV_y) = qsum; %if q is used for every house
% ub(DV_x+1:DV_x+DV_y) = sum(qvec); %if qvec is used
ub(DV_x+DV_y+1:DV_x+DV_y+DV_B) = B_lim;
ub(DV_x+DV_y+DV_B+1:DV_x+DV_y+DV_B+DV_w) = Q;
if NH_case ~= 1
    ub(DV_x+DV_y+DV_B+DV_w+1:DV_x+DV_y+DV_B+DV_w+DV_v) = P_hi;
end

x0 = [];

sostype=[];
sosind=[];
soswt=[];
codetime = toc;

[x,fval,exitflag,output]                                                    =
cplexmilp(obj,A,b,Aeq,beq,sostype,sosind,soswt,lb,ub,ctype,x0,opts2);

% x
% fval
% exitflag
% output

%%%%%%Output code
% Index the results

%+++++++++++++++++
%For Matlab 2016a because we dont update our school computers

J = I;

if NH_case == 1
    for i = 1:size(I,1) %Matrix already transposed for flag index
        if i <= D
```

```
            I(:,i) = cellstr('D');
        elseif i <= D+N
            I(:,i) = cellstr(sprintf('N%u',i-D));
        else
            I(:,i) = cellstr('Dpr');
        end
    end

    for j = 1:size(J,1) %Matrix already transposed for flag index
        if j <= D
            J(j,:) = cellstr('D');
        elseif j <= D+N
            J(j,:) = cellstr(sprintf('N%u',j-D));
        else
            J(j,:) = cellstr('Dpr');
        end
    end

else

    for i = 1:size(I,1) %Matrix already transposed for flag index
        if i <= D
            I(:,i) = cellstr('D');
        elseif i <= D+N
            I(:,i) = cellstr(sprintf('N%u',i-D));
        elseif i <= D+N+F
            I(:,i) = cellstr(sprintf('F%u',i-(D+N)));
        else
            I(:,i) = cellstr('Dpr');
        end
    end

    for j = 1:size(J,1) %Matrix already transposed for flag index
        if j <= D
            J(j,:) = cellstr('D');
        elseif j <= D+N
            J(j,:) = cellstr(sprintf('N%u',j-D));
        elseif j <= D+N+F
            J(j,:) = cellstr(sprintf('F%u',j-(D+N)));
        else
            J(j,:) = cellstr('Dpr');
        end
    end
end



for i = 1:D+N
    Yl(i) = cellstr(sprintf('y%u',i-1));
end
sY = Yl;

for k = 1:K
    Wl(k) = cellstr(sprintf('w%u',k));
end
```

```
sW = Wl;

for bc = 1:DV_B
    Bl(bc) = cellstr(sprintf('B%u',bc));
end
Bl = Bl';

if NH_case ~= 1
    for f = 1:F
    Vl(f) = cellstr(sprintf('v%u',f));
    end
    sV = Vl;
end

ValueLab = cellstr('Total Value');

flag_mat_tr_vehc = flag_mat_tr;
if K >= 2
    for k = 2:K
        flag_mat_tr_vehc = [flag_mat_tr_vehc,flag_mat_tr];
        I = [I,I];
        J = [J,J];
        Yl = [Yl;sY];
        Wl = [Wl;sW];
    end
end
[J2, I2] = find(flag_mat_tr_vehc);

%create matrices of results for individual generator cost results table
MatResults_x = zeros(size(flag_mat_tr_vehc));
idx = sub2ind(size(MatResults_x),J2,I2);
MatResults_x(idx) = x(1:DV_x);
MatResults_x_log = MatResults_x > 0.9;

Solution.Departx = I(MatResults_x_log);
Solution.Arrivex = J(MatResults_x_log);
Solution.Answerx = MatResults_x(MatResults_x_log);
Solution.PickupLab = Yl;
Solution.CargoQuant = x(DV_x+1:DV_x+DV_y);
Solution.DropTimeLab = Bl;
Solution.DropTimeQuant = x(DV_x+DV_y+1:DV_x+DV_y+DV_B);
Solution.DropLab = Wl;
Solution.CargoDrop = x(DV_x+DV_y+DV_B+1:DV_x+DV_y+DV_B+DV_w);

if NH_case ~= 1
    Solution.FacilQuantLab = Vl;
    Solution.FacilQuantVal = x(DV_x+DV_y+DV_B+DV_w+1:DV_tot);
end

Solution.VLab = ValueLab;
Solution.Value = fval;

% if NH == 1
%           SetupLabs   =   [cellstr('CodeTime(s)');   cellstr('SolverTime(s)');
cellstr('Depots');    cellstr('Households');    cellstr('Collect.    Trucks');
```

```
cellstr('Avg. HH Gen. Quantity'); cellstr('Truck Capacity'); cellstr('Solution
Labels')];
%               SetupVals   =   [num2cell(codetime);   num2cell(output.time);
cellstr(sprintf('%u',D)); cellstr(sprintf('%u',N)); cellstr(sprintf('%u',K));
cellstr(sprintf('%u',mean(qvec))); cellstr(sprintf('%u',Q)); cellstr('Solution
Values')];
%            Labels   =   [SetupLabs;   Solution.Departx;   Solution.PickupLab;
Solution.DropTimeLab; Solution.DropLab; Solution.VLab];
%       Values = [SetupVals; Solution.Arrivex; num2cell(Solution.CargoQuant);
num2cell(Solution.DropTimeQuant);               num2cell(Solution.CargoDrop);
num2cell(Solution.Value)];
% else
%           SetupLabs   =   [cellstr('CodeTime(s)');   cellstr('SolverTime(s)');
cellstr('Depots');   cellstr('Households');   cellstr('Recycling   Facilities');
cellstr('Collect.   Trucks');   cellstr('HH   Gen.   Quantity');   cellstr('Truck
Capacity'); cellstr('Solution Labels')];
%               SetupVals   =   [num2cell(codetime);   num2cell(output.time);
cellstr(sprintf('%u',D)); cellstr(sprintf('%u',N)); cellstr(sprintf('%u',F));
cellstr(sprintf('%u',K));                   cellstr(sprintf('%u',mean(qvec)));
cellstr(sprintf('%u',Q)); cellstr('Solution Values')];
%            Labels   =   [SetupLabs;   Solution.Departx;   Solution.PickupLab;
Solution.DropTimeLab;         Solution.DropLab;         Solution.FacilQuantLab;
Solution.VLab];
%       Values = [SetupVals; Solution.Arrivex; num2cell(Solution.CargoQuant);
num2cell(Solution.DropTimeQuant);               num2cell(Solution.CargoDrop);
num2cell(Solution.FacilQuantVal); num2cell(Solution.Value)];
% end
%
% SolutionTable = table(Labels, Values);
% [sHH,col] = find(MatResults_x(:,1));
% startHH = sHH-D;
% [row,eHH] = find(MatResults_x(end,:));
% endHH = eHH-D;

% idxHHNet = find(idxHH);

NHNetData.RouteTime(idxNH) = Solution.DropTimeQuant;
NHNetData.qgen(idxNH) = Solution.CargoDrop;
NHNetData.Cost(idxNH) = Solution.Value;
% NHNetData.StartHH(clust) = idxHHNet(startHH);
% NHNetData.EndHH(clust) = idxHHNet(endHH);

end
```

FOR CHAPTER 4
Agent Based Model Code

```
'''
run the model version 3
'''

import numpy as np
import pandas as pd
#import numpy as np
from model_v3 import WasteCollection_V3
import os


#transfer_time = 16.5 #minutes/tonne to collect food waste
#initial_trucks = 2
#load_limit = 4 #tonnes
#truck_time_limit = 100 #minutes

if os.path.isfile("ABMSolution.xlsx"):
    os.remove("ABMSolution.xlsx")

#input_filename = 'D:/INFEWS/EcoFWCollect/EcoCollectionModels/input_data_MC_20gen.xlsx'
input_filename = 'D:/INFEWS/EcoFWCollect/EcoCollectionModels/input_data_homogen_100gen.xlsx'

truck_info = pd.read_excel(input_filename, sheet_name = 'truck_info')
ant_model_info = pd.read_excel(input_filename, sheet_name = 'ant_model_info')

#initial_trucks = truck_info.loc['Variables','Trucks']
#load_limit = truck_info.loc['Variables','LoadLimit'] #tonnes
#travel_time_limit = truck_info.loc['Variables','TimeLimit'] #minutes
#tour_limit = truck_info.loc['Variables','TourLimit']
#transfer_time = truck_info.loc['Variables','TransferTime'] #minutes/tonne to collect food waste
#operating_cost = truck_info.loc['Variables','OperatingCost'] #cost in $/min
#service_fee = truck_info.loc['Variables','ServiceFee'] #charge to generators to collect 1 tonne of FW

system_options = pd.read_excel(input_filename, sheet_name = 'SystemOptions')
MILPsol_mat = pd.read_excel(input_filename, sheet_name = 'MILPsol_uniqueID', header = None)

FWgen_rate = pd.read_excel(input_filename, sheet_name = 'FWgen_rate')
initial_inventory = pd.read_excel(input_filename, sheet_name = 'initial_inventory')
if system_options.loc['Options','Distance']:
```

```
    time_matrix = pd.read_excel(input_filename, sheet_name = 'network_dist')
elif system_options.loc['Options','Time']:
    time_matrix = pd.read_excel(input_filename, sheet_name = 'network_time')
network_locs = pd.read_excel(input_filename, sheet_name = 'network_locs')

initial_generators = FWgen_rate.shape[1]
initial_treatments = initial_inventory.shape[1] - initial_generators

generation_percept_template = pd.read_excel(input_filename, sheet_name = 'generation_percept')
history_template = pd.read_excel(input_filename, sheet_name = 'history_template')

pickup_mem_init = [] #initialize a history of collection from geenrators not tied to a route
list(pickup_mem_init)

for i in range(initial_generators):
    colname = history_template.columns[i]
    pickup_mem_init.append(np.array(history_template[colname]))

randomseed = system_options.loc["Options","RandomSeed"]

#initial_inventory = np.array([0,0,0,0,0])
#FWgen_rate = np.array([5,3,1,4])
#time_matrix = np.array([[0,10],[10,0]])
#network = initial_generators + initial_treatments

modelV3 = WasteCollection_V3(truck_info, ant_model_info, initial_generators,
initial_treatments, initial_inventory, FWgen_rate, time_matrix, network_locs,
generation_percept_template, history_template, pickup_mem_init, system_options,
MILPsol_mat, randomseed)
#
#ra = modelV2.counter_list[0]
#sd = modelV2.counter_list[1]
#g = modelV2.generator_list[0]
#treat = modelV2.treatment_list[0]
truck1 = modelV3.truck_list[0]
#truck2 = modelV2.truck_list[1]
```

```
'''
Version 3 of the model
'''

import pandas as pd
import copy
import time
import numpy as np
from mesa import Model
import matplotlib.pyplot as plt

#import matplotlib.animation as animation
#from mesa.space import NetworkGrid
from mesa.time import BaseScheduler
#from mesa.datacollection import DataCollector

from agents_v3 import Truck, Generator, Treatment, Sundial, ResetAgent
#from network import Network


class WasteCollection_V3(Model):

    verbose = False #Print-monitoring

    def __init__(self, truck_info, ant_model_info, initial_generators, initial_treatments, initial_inventory, FWgen_rate, time_matrix, network_locs, generation_percept_template, history_template, pickup_mem_init, system_options, MILPsol_mat, randomseed):

        #initialize system options from excel sheet
        self.system_options = system_options

        #Set parameters
        self.initial_generators = initial_generators
        self.initial_treatments = initial_treatments
        self.initial_inventory = initial_inventory
        self.time_matrix = time_matrix
        self.network_locs = network_locs
        self.current_day = 1
        self.FWgen_rate= FWgen_rate
        self.maxFW = sum(self.FWgen_rate.iloc[0,:])
        self.totalcollected = 0

        #use truck infor to set truck information
        self.initial_trucks = truck_info.loc['Variables','Trucks']
        self.load_limit = truck_info.loc['Variables','LoadLimit'] * 1000 #convert tons to kg
```

```
        if self.system_options.loc['Options','Distance']:
            self.travel_time_limit = truck_info.loc['Variables','DistLimit'] #minutes
            self.fixtransfer_time = 0 #Input in minutes per stop
            self.operating_cost = truck_info.loc['Variables','OperatingCostkm'] / 1000 #input in $/hr convert to cost in $/min
            self.pheromone_cost_init = ant_model_info.loc['Param','initPheromone_dist_cost']
        elif self.system_options.loc['Options','Time']:
            self.travel_time_limit = truck_info.loc['Variables','TimeLimit'] #minutes
            self.fixtransfer_time = truck_info.loc['Variables','FixTransferTime'] #Input in minutes per stop
            self.operating_cost = truck_info.loc['Variables','OperatingCosthr'] / 60 #input in $/hr convert to cost in $/min
            self.pheromone_cost_init = ant_model_info.loc['Param','initPheromone_time_cost']

        self.tour_limit = truck_info.loc['Variables','TourLimit']
        self.vartransfer_time = truck_info.loc['Variables','VarTransferTime'] /1000 #input in minutes/tonne, convert to minutes/kg
        self.service_fee = truck_info.loc['Variables','ServiceFee'] / 1000 #charge to generators to collect 1 tonne of FW converted to kg
        self.exploit_prob_update = truck_info.loc['Variables','ExploitProbUpdate']
        self.exploit_prob_limit = truck_info.loc['Variables','ExploitProbLimit']
        self.exploit_prob_day = 10
#         self.truck_time_limit = truck_time_limit

        self.pheromone_profit_init = ant_model_info.loc['Param','initPheromone_time_profit']
        self.alpha_param = ant_model_info.loc['Param','Alpha']
        self.beta_param = ant_model_info.loc['Param','Beta']
        self.gamma_param = ant_model_info.loc['Param','Gamma']
#         self.pheromone_type = ant_model_info.loc['Param','Type'] #profit, revenue, or cost

        self.objective_hist = np.zeros(1)
        self.compare_objective = 0

        self.reset_value_costnum = 5000

        if self.system_options.loc['Options','ProfitModel']:
            self.current_objective = 0
            self.best_objective = 0
            self.objective_type = 'Profit Model'
        elif self.system_options.loc['Options','CostModel']:
            self.current_objective = 0
            self.best_objective = self.reset_value_costnum
            self.objective_type = 'Cost Model'

        if self.system_options.loc['Options','PartCollect_system']:
```

```
            self.collection_constraint = 'Partial Collect'
        elif self.system_options.loc['Options','FullCollect_system']:
            self.collection_constraint = 'Collect All'


        #initialize a scheduler
        self.schedule = BaseScheduler(self)
        self.counter_list = []
        self.treatment_list = []
        self.generator_list = []
        self.truck_list = []
        self.all_list = []

        #initialize a base perception of generators
        self.generation_percept_init = copy.copy(generation_percept_template)
        generator_dayvisit_mem_init = [0] * self.initial_generators

        self.MILPsol_bestrouteIDs = MILPsol_mat.values.tolist()
        self.MILPday = 25000

        '''
        Create starting distributions by cheesing the averageing process
        '''
        self.pop_FWG_sample_u = int(self.FWgen_rate.loc['Mean'].mean(axis = 0))
        self.pop_FWG_sample_o = int(self.FWgen_rate.loc['Std'].mean(axis = 0) * 2)

        self.random_seed = randomseed

#       self.random_seed = 1
#       self.random_seed = 12
#       self.random_seed = 123
#       self.random_seed = 1234
#       self.random_seed = 12345
#       self.random_seed = 123456
#       self.random_seed = 1234567
#       self.random_seed = 12345678
#       self.random_seed = 123456789
#       self.random_seed = 1234567890

        np.random.seed(self.random_seed) #random seed for reproduceability

        #create and schedule agent for resetting information at the beginning of the day
        for i in range(1):
            breed = 'reset_agent'
            indiv_id = i + 1
            unique_id = breed + ' ' + str(indiv_id)
```

```
            reset_agent = ResetAgent(unique_id, self)
            self.schedule.add(reset_agent)
            self.counter_list.append(reset_agent)



        #Create Treatment facilities
        for i in range(self.initial_treatments):
            breed = 'treatment'
            indiv_id = i + 1
            unique_id = breed + ' ' + str(indiv_id)
            node_id = i
            inventory = initial_inventory.loc['Inventory'][i]
            treatment = Treatment(unique_id, self, node_id, breed, indiv_id, inventory)
#             self.network.place_agent(treatment, node_id)
            self.schedule.add(treatment)
            self.treatment_list.append(treatment)
            self.all_list.append(treatment)

        #Create Generation facilities
        for i in range(self.initial_generators):
            breed = 'generator'
            indiv_id = i + 1
            unique_id = breed + ' ' + str(indiv_id)
            node_id = self.initial_treatments + i
            inventory = initial_inventory.loc['Inventory'][self.initial_treatments + i]
            pickup_request = 0
            FWgen_rate_u = FWgen_rate.loc['Mean'][i]
            FWgen_rate_o = FWgen_rate.loc['Std'][i]
            day = 0
            generator = Generator(unique_id, self, node_id, breed, indiv_id, inventory,
pickup_request, FWgen_rate_u, FWgen_rate_o, day)

            self.generation_percept_init.loc['Mean', unique_id] = self.pop_FWG_sample_u
            self.generation_percept_init.loc['Std', unique_id] = self.pop_FWG_sample_o

            self.schedule.add(generator)
            self.generator_list.append(generator)
            self.all_list.append(generator)

        #Create Trucks
        for i in range(self.initial_trucks):
            breed = 'truck'
            indiv_id = i + 1
            unique_id = breed + ' ' + str(indiv_id)
            home_id = 0 #temporary. need to match with treatment facilities
```

```
            pos_id = home_id
            pos_desc = 'treatment'
            #Switch to use Bayes updating or not. If switched off in options file, then distribution chosen from is same as true generation.
            #To make perfect information complete, also check option for Mean SAmpling in options input

            MILPsol_routeID = self.MILPsol_bestrouteIDs[i]

            if self.system_options.loc['Options','Bayes']:
                generation_percept_init = self.generation_percept_init
            else:
                generation_percept_init = self.FWgen_rate

            truck = Truck(unique_id, self, pos_id, pos_desc, indiv_id, breed, home_id, generation_percept_init, generator_dayvisit_mem_init, pickup_mem_init, history_template, MILPsol_routeID)
            self.schedule.add(truck)
            self.truck_list.append(truck)
            self.all_list.append(truck)

        #Create Sundail agent to track time
        for i in range(1):
            breed = 'sundial'
            indiv_id = i + 1
            unique_id = breed + ' ' + str(indiv_id)
            sundial = Sundial(unique_id, self)
            self.schedule.add(sundial)
            self.counter_list.append(sundial)


        self.running = True



    def step(self):
        self.schedule.step()


    def run_model(self, step_count):
        '''
        for a single truck with a single tour, run for multiples of generators + 1 for complete days
        '''

        start_time = time.time()
```

```
        day_count = step_count

        for i in range(step_count):
            self.step()

        elapsed_time = time.time() - start_time


        x = []
        y = []
        for i in range(len(self.objective_hist)):
            x = np.append(x, i + 1)
            y = np.append(y, self.objective_hist[i])


        tPlot, axes = plt.subplots(
            nrows=2, ncols=1, sharex=False, sharey=False)

        tPlot.subplots_adjust(hspace=0.4)

#           tPlot.suptitle('Alpha = ' + str(self.alpha_param) + '    Beta = ' + str(self.beta_param) + '
Gamma = ' + str(self.gamma_param) + '    Pheromone Initial = ' + str(self.pheromone_init),
fontsize=12)

        axes[0].plot(x,y)
        axes[0].set(title = 'Title Here',
                    xlabel='Days',
                    ylabel=self.objective_type + ' Objective Value')

        if self.objective_type == 'Profit Model':
            init_pheromone_val = self.pheromone_profit_init
            obj_type = "Pr"
        elif self.objective_type == 'Cost Model':
            init_pheromone_val = self.pheromone_cost_init
            obj_type = "Co"

        if self.system_options.loc['Options','PartCollect_system']:
            coll_type = "Pa"
        elif self.system_options.loc['Options','FullCollect_system']:
            coll_type = "Al"

        textstr = '\n'.join((
                'Truck 1 best route = %s' % self.truck_list[0].best_route_ids,
#               'Truck 2 best route = %s' % self.truck_list[1].best_route_ids,
                'Best Objective Value = %0.1f' % self.best_objective,
                'Best Day = %i' % self.best_day,
```

```
            r'$\alpha = %s$' % self.alpha_param,
            r'$\beta = %s$' % self.beta_param,
            r'$\gamma = %s$' % self.gamma_param,
            'Pheromone init. = %s' % init_pheromone_val
            ))

        axes[1].axis('off')
        axes[1].text(0, 1, textstr, verticalalignment='top')

        self.t_loc = self.network_locs.loc[['treatment 1']]
        self.g_loc = self.network_locs.loc['generator 1':]

        if self.system_options.loc['Options','Distance']:
            OC = 'Operating Cost ($/km)'
            OC_val = self.operating_cost * 1000

        elif self.system_options.loc['Options','Time']:
            OC = 'Operating Cost ($/hr)'
            OC_val = self.operating_cost * 60

        '''Create filename ending'''
        '''+ 'rs' + str(self.random_seed) + '_' + str(obj_type)+ str(coll_type) + "_" + str(OC_val) + "c"+
str(self.service_fee*1000) + "r_" + str(day_count) + "D_EP" + str(self.exploit_prob_limit) + "_A"
+ str(self.alpha_param) + "_B" + str(self.beta_param) + "_G" + str(self.gamma_param) +
"_Pherinit" + str(round(init_pheromone_val,4))'''

        MILPday = str(self.MILPday)
        rs = str(self.random_seed)
        gens = str(self.initial_generators)
        objcon = str(obj_type)+ str(coll_type)
        cost = str(OC_val)
        rev = str(self.service_fee*1000)
        trials = str(day_count)
        EP = str(self.exploit_prob_limit)
        alph = str(self.alpha_param)
        bet = str(self.beta_param)
        gam = str(self.gamma_param)
        ph_ini =  str(round(init_pheromone_val,5))


        rs = rs.replace(".",",")
        gens = gens.replace(".",",")
        objcon = objcon.replace(".",",")
        cost = cost.replace(".",",")
        rev = rev.replace(".",",")
        trials = trials.replace(".",",")
```

```
        EP = EP.replace(".",",")
        alph = alph.replace(".",",")
        bet = bet.replace(".",",")
        gam = gam.replace(".",",")
        ph_ini = ph_ini.replace(".",",")

        filepath = "D:/INFEWS/EcoFWCollect/EcoCollectionModels/ABM_ACO_v3/Results/"

        if self.system_options.loc['Options','InsertMILP']:
            fileend = gens + "gens_" + 'MILPinsert' + MILPday + '_rs' + rs + '_' +  objcon + "_c"+ cost
+ "_r" + rev + "_" + trials + "trials_EP" + EP + "_A" + alph + "_B" + bet + "_G" + gam +
"_Pherinit" + ph_ini
        else:
            fileend = gens + "gens_" + 'rs' + rs + '_' +  objcon + "_c"+ cost + "_r" + rev + "_" + trials
+ "trials_EP" + EP + "_A" + alph + "_B" + bet + "_G" + gam + "_Pherinit" + ph_ini


        tPlot.savefig(filepath + "ABMSummary_MC_" + fileend + ".png")


#       routemap, axes2 = plt.subplots(nrows=1, ncols=1)
        network_locs = copy.copy(self.network_locs)

        ax = self.t_loc.plot('xloc','yloc', kind = 'scatter', s=50, color = 'b', marker = '^')

        minx = min(network_locs.xloc)
        miny = min(network_locs.yloc)
        maxx = max(network_locs.xloc)
        maxy = max(network_locs.yloc)

        ax.set_xlim([minx - 0.02,maxx + 0.02])
        ax.set_ylim([miny - 0.02,maxy + 0.02])

        self.g_loc.plot('xloc','yloc', kind = 'scatter', color = 'r', s=50, ax = ax)

        for i, txt in enumerate(network_locs.labels):
            ax.annotate(txt, (network_locs.xloc.iat[i]+0.002,network_locs.yloc.iat[i]+0.006))

#       colorlist = ['k','b','r']

        for truck in self.truck_list:
            br = truck.best_route
            br.insert(0,self.treatment_list[0])
            for j in range(len(br)-1):
                ax.annotate(",
```

```
                    xy=(network_locs.loc[br[j+1].unique_id,
'xloc'],network_locs.loc[br[j+1].unique_id, 'yloc']), xycoords='data',
                    xytext=(network_locs.loc[br[j].unique_id,
'xloc'],network_locs.loc[br[j].unique_id, 'yloc']), textcoords ='data',
                    arrowprops=dict(connectionstyle="arc3", facecolor = 'k', width = 0.3, headwidth
= 5, headlength = 5),
                    )

    ax.figure.savefig(filepath + "ABMFigure_MC_" + fileend + ".png")

    '''
    Format at export results to excel sheet to make similar to matlab output
    '''

    totalcollected = 0

    for k in range(self.initial_trucks):
        truck = self.truck_list[k-1]

        if k == 0:
            self.truck_values = ['treatment1']
            self.truck_labels = ['truck'+ str(k+1)+ ' route']
        for i in range(len(truck.best_route_ids)):
            self.truck_values.append(truck.best_route_ids[i])
            self.truck_labels.append('truck'+ str(k+1)+ ' route')
        for i in range(len(truck.best_route_pickups)):
            self.truck_values.append(truck.best_route_pickups[i])
            self.truck_labels.append('truck'+ str(k+1)+ ' pickups (kg)')
            if truck.best_route_pickups[i] > 0:
                self.totalcollected += truck.best_route_pickups[i]


        if self.system_options.loc['Options','Distance']:
            self.truck_labels.append(['truck'+ str(k+1)+ ' Dist (km)'])
            self.truck_values.append(round(truck.best_operating_time/1000,4))
        elif self.system_options.loc['Options','Time']:
            self.truck_values.append(round(truck.best_operating_time/60,4))
            self.truck_labels.append(['truck'+ str(k+1)+ ' Time (hr)'])

        self.truck_values.append(round(truck.best_revenue,4))
        self.truck_labels.append(['truck'+ str(k+1)+ ' Revenue ($)'])
        self.truck_values.append(round(truck.best_cost,4))
        self.truck_labels.append(['truck'+ str(k+1)+ ' Cost ($)'])
        profit = truck.best_revenue - truck.best_cost
        self.truck_values.append(round(profit,4))
        self.truck_labels.append(['truck'+ str(k+1)+ ' Profit ($)'])
```

```
        self.truck_values.append(self.totalcollected / 1000)
        self.truck_labels.append(['Total FW Collected (t)'])
        self.truck_values.append(round(self.best_revenue,4))
        self.truck_labels.append(['Model Revenue ($)'])
        self.truck_values.append(round(self.best_cost,4))
        self.truck_labels.append(['Model Cost ($)'])
        self.truck_values.append(round(self.best_profit,4))
        self.truck_labels.append(['Model Profit ($)'])

        self.truck_values.append(round(self.best_objective,4))
        self.truck_labels.append(['Model Objective ($)'])



        self.model_labels = ['RunTime (s)','Random Seed','Model Type','Collection Constraint','Seed Value','Exploit Probability','Alpha Param Val','Beta Param Val','Gamma Param Val','Initial Pheromone Value','Depots','Intermediate Drop-offs','Generators','Collection Trucks',\
                             'Truck Capacity (t)','Total Generation (t)','Gen. Collect Time (min)','Service Fee ($/t)',OC,'Days Run','Best Solution Day']

        self.model_labels.extend(self.truck_labels)

        self.model_values = [elapsed_time,self.random_seed,self.objective_type,self.collection_constraint,self.random_seed,self.exploit_prob_limit,self.alpha_param,self.beta_param,self.gamma_param,init_pheromone_val,self.initial_treatments,self.tour_limit-1,self.initial_generators,self.initial_trucks,\

self.load_limit/1000,self.maxFW/1000,self.fixtransfer_time,self.service_fee*1000,OC_val,self.current_day - 1,self.best_day]

        self.model_values.extend(self.truck_values)

        solutiontable = pd.DataFrame(self.model_values, index = self.model_labels, columns = ['ModelValues'])
#       print(str(OC_val))
        solutiontable.to_excel(filepath + "ABMSolution_MC_" + fileend + ".xlsx")

        plt.close('all')
```

```
'''
Version 3 of agents
'''

import pandas as pd
import copy
import numpy as np
from mesa import Agent
#from mymodel import WasteCollection #circular
from truck_behavior_v3 import TruckMove_V3
from sundial_funcs_v3 import TimeKeeper_V3

class Truck(TruckMove_V3):
    '''
    some trucks
    '''

    def __init__(self, unique_id, model, pos_id, pos_desc, indiv_id, breed, home_id, generation_percept_init, generator_dayvisit_mem_init, pickup_mem_init, history_template, MILPsol_routeID):
        super().__init__(unique_id, model, pos_id, pos_desc)

        #initialize starting attributes of truck agent
        self.unique = unique_id
        self.pos_id - pos_id
        self.pos_desc = pos_desc
        self.indiv_id = indiv_id
        self.breed = breed
        self.home_id = home_id
        self.load = 0
        self.active = 1
        self.tours = 0
        self.path_choice = []


        #initialize starting memory of truck agents
        self.generation_percept = generation_percept_init.copy() #trucks perception of generators upon first dailty visit that will be updated as information becomes available to them
        self.generation_percept_plus = generation_percept_init.copy() ##rucks perception of generators upon second or more dailty visits that will be updated as information becomes available to them
        self.pickup_mem = pickup_mem_init.copy() # record of how much FW was collected from the specific generator upon first daily visit
        self.pickup_mem_plus = pickup_mem_init.copy() #record of how much FW was collected from specific generator upon second or more daily visit
```

```
        self.generator_dayvisit_mem = generator_dayvisit_mem_init.copy() #list that records how many times in a day that the truck visits a generator. resets each day
        self.generators_total_visits = generator_dayvisit_mem_init.copy() #record overall how many times generator was visted by a truck
        self.generators_total_visits_plus = generator_dayvisit_mem_init.copy()

        #initialize route histories and counters
        self.route_step = 0
        self.route_hist = ['N/A']
        self.route_hist_ids = ['N/A']
        self.route_pickup_hist = ['N/A'] #pickup history coupled with routes

        #initialize pheromone matrix for ant colony optimization
        self.pheromone_matrix = self.model.time_matrix.copy()
        if self.model.system_options.loc['Options','ProfitModel']:
            self.pheromone_matrix.iloc[:,:] = self.model.pheromone_profit_init
        elif self.model.system_options.loc['Options','CostModel']:
            self.pheromone_matrix.iloc[:,:] = self.model.pheromone_cost_init

        #initialize probability that truck will choose to exploit best path rather than explore
        if self.model.system_options.loc['Options','FWGND']:
            self.exploit_prob = 0
        elif self.model.system_options.loc['Options','ACO']:
            self.exploit_prob = copy.copy(self.model.exploit_prob_limit)

#       self.exploit_prob = 0

        self.exploit_prob_hist = []
        self.pathchoice1 = []
        self.pathchoice2 = []

        #set pairwise improvement counter
        self.current_pairwise_improvement = 0
        self.best_pairwise_improvement = 1

        #initialize current route memory and counters
        self.current_route = [] #list of generator and treatment objects
        self.current_route_ids = []
        self.current_route_pickups = [] #list of how much FW was collected from each generator in order
        self.current_route_revenues = []
        self.current_route_costs = []
        self.current_route_effs = []
        self.current_operating_time = 0

        #initialize best route information
```

```
        self.compare_route = []

        self.best_route = []
        self.best_route_ids = []
        self.best_route_pickups = []
        self.best_route_revenues = []
        self.best_route_costs = []
        self.best_route_effs = []
        self.best_day = 0
        self.best_operating_time = 0

        #initialize current cost, revenue, profit, and objectives based on objective type
        self.current_revenue = 0
        self.current_cost = 0
        self.current_profit = 0

        self.best_revenue = 0
        self.best_cost = 0
        self.best_profit = 0

        self.current_objective = 0
        self.best_objective = self.model.reset_value_costnum

#       self.current_finaleff = 0
#       self.best_finaleff = 0


        self.objective_hist = np.zeros(1)
        self.objective_mem = 0 #what is this for?


        self.MILPsol_route = []
        for tag in MILPsol_routeID:
#           print(tag)
            agent = [agent for agent in self.model.all_list if agent.unique_id == tag]
            self.MILPsol_route.extend(agent)

    def step(self):
        if self.model.system_options.loc['Options','FWGND'] \
        or self.model.system_options.loc['Options','RandomChoice']:
            if self.active: #check to see if truck is active
                possible_targets = self.SurveyGenerators() #get a list of possible target generator (Expand later)
                if self.model.system_options.loc['Options','TimeConstraint']: #time constraint
                    possible_targets = self.ReduceTargets_TimeLimit(possible_targets)
```

```
            if len(possible_targets) == 0: #if there are no generator targets, go home
                target = self.SelectHomeTarget()

            elif len(possible_targets) > 0: #if there are generator targets
                if self.model.system_options.loc['Options','FullCollect_gen']: #if trucks can only load
all inventory at a location
                    possible_targets = self.ReduceTargets_LoadLimit(possible_targets)

                if self.load >= self.model.load_limit: #if trucks are at load capacity
                    possible_targets = []

                if len(possible_targets) > 0:
                    #check possible targets
                    ptids = []
                    for i in range(len(possible_targets)):
                        ptids.append(possible_targets[i].unique_id)
                    print(ptids)

                    if self.model.system_options.loc['Options','RandomChoice']:
                        target = self.SelectGeneratorTargetRandom(possible_targets) #select a targer

                    elif self.model.system_options.loc['Options','FWGND']:
                        target = self.SelectGeneratorTargetFWG(possible_targets)

                elif len(possible_targets) == 0: #if there are no generator targets, go home
                    target = self.SelectHomeTarget()

            travel_time = self.MoveToTarget(target) #move the truck to the target

#             print(self.unique_id + ' moved to ' + target.unique_id) #print which truck moved where

            if self.pos_desc == 'generator':
                self.LoadFoodWaste(target, travel_time)

            elif self.pos_desc == 'treatment':
                self.UnloadFoodWaste(target, travel_time)

            self.StatusCheck()

#             if self.active == 0\

            #if pairwise improvement is needed:
            #do some stuff

        elif self.model.system_options.loc['Options','ACO']:
            while self.active == 1:
```

```
            if self.active: #check to see if truck is active
                possible_targets = self.SurveyGenerators() #get a list of possible target generator (Expand later)

                #reduce the number of targets to ones closest to the truck's current position
#                 possible_targets = self.ReduceTargets_Proximity(possible_targets)

                if self.model.system_options.loc['Options','TimeConstraint']: #time constraint
                    possible_targets = self.ReduceTargets_TimeLimit(possible_targets)

                if len(possible_targets) == 0: #if there are no generator targets, go home
                    target = self.SelectHomeTarget()

                elif len(possible_targets) > 0: #if there are generator targets
                    if self.model.system_options.loc['Options','FullCollect_gen']: #if trucks can only load all inventory at a location
                        possible_targets = self.ReduceTargets_LoadLimit(possible_targets)

                    #reduce targets if they have already been visited in the route
                    possible_targets = self.ReduceTargets_VisitNum(possible_targets)

                    if self.load >= self.model.load_limit: #if trucks are at load capacity
                        possible_targets = []

                    if len(possible_targets) > 0:
#                         target = self.BaseAntColonySelection(possible_targets)
                        target = self.ImprovedAntColonySelection(possible_targets)

                    elif len(possible_targets) == 0: #if there are no generator targets, go home
                        target = self.SelectHomeTarget()

                travel_time = self.MoveToTarget(target) #move the truck to the target

#                 print(self.unique_id + ' moved to ' + target.unique_id) #print which truck moved where

                if self.pos_desc == 'generator':
                    self.LoadFoodWaste(target, travel_time)

                elif self.pos_desc == 'treatment':
                    self.UnloadFoodWaste(target, travel_time)

                self.StatusCheck()
                #if pairwise improvement is needed:
                #do some stuff
```

```
class Generator(Agent):

    def __init__(self, unique_id, model, node_id, breed, indiv_id, inventory, pickup_request, FWgen_rate_u, FWgen_rate_o, day):
#         super().__init__(unique_id, model)
        self.unique_id = unique_id
        self.model = model
        self.breed = breed
        self.indiv_id = indiv_id
        self.node_id = node_id
        self.node_desc = self.breed
        self.inventory = inventory
        self.pickup_request = pickup_request
        self.FWgen_rate_u = FWgen_rate_u
        self.FWgen_rate_o = FWgen_rate_o
        self.day = day

    def step(self):
        if self.day < self.model.current_day:

            quantity = copy.copy(self.FWgen_rate_u)

            if self.model.system_options.loc['Options','Reset'] == 1:
                self.inventory = quantity
            elif self.model.system_options.loc['Options','Reset'] == 0:
                self.inventory += quantity

            if self.inventory > 0:
                self.pickup_request = 1
            self.day += 1
            '''Print inventory if needed'''
#             print(self.unique_id + ' inventory: ' + str(self.inventory) + ' kg FW')


class Treatment(Agent):

    def __init__(self, unique_id, model, node_id, breed, indiv_id, inventory):
        super().__init__(unique_id, model)
        self.unique_id = unique_id
        self.model = model
        self.breed = breed
        self.indiv_id = indiv_id
        self.node_id = node_id
        self.node_desc = self.breed
        self.inventory = inventory
```

```
    def step(self):
        '''Print inventory if needed'''
#         print(self.unique_id + ' inventory: ' + str(self.inventory) + ' kg FW')


class Sundial(TimeKeeper_V3):

    def __init__(self, unique_id, model):
#         super().__init__(unique_id, model)
        self.unique_id = unique_id
        self.model = model

    def step(self):
        active_trucks = [truck for truck in self.model.truck_list if truck.active == 1] #check to see if
any trucks are active
        if len(active_trucks) == 0:

            for i in range(len(self.model.truck_list)):
                truck = self.model.truck_list[i]
                if truck.exploit_prob >= self.model.exploit_prob_limit:
                    truck.exploit_prob = self.model.exploit_prob_limit

            if self.model.system_options.loc['Options','FWGND']:
                self.CompareObjectives()
            elif self.model.system_options.loc['Options','ACO']:
#                 self.BasePheromoneUpdate()
                self.ImprovedPheromoneUpdate()

            self.model.current_day += 1 #move onto the next day to reintialize collection


class ResetAgent(TimeKeeper_V3):
    def __init__(self, unique_id, model):
        self.unique_id = unique_id
        self.model = model
        self.day = 0

    def step(self):
        if self.day < self.model.current_day:
            self.NewDayReset()
            self.day += 1
            print('New Day: ' + str(self.model.current_day)) #what day is it?
```

```
'''
truck behavior modules version 3
'''

import numpy as np
import copy
import pandas as pd
import random
import itertools
from itertools import permutations
from operator import itemgetter, attrgetter


#from mymodel import WasteCollection
from mesa import Agent

class TruckMove_V3(Agent):

    def __init__(self, unique_id, model, pos_id, pos_desc):
        '''
        Class implementing driving behavior of trucks
        '''

        super().__init__(unique_id, model)
        self.pos_id = pos_id
        self.pos_desc = pos_desc


    def SurveyGenerators(self):
        '''
        Survey the network for all potential generator targets.
        '''
        avail_gens = [generator for generator in self.model.generator_list if generator.pickup_request
== 1]
#       truck_positions = [truck.pos_id for truck in WasteCollection.schedule ]
        possible_targets = avail_gens #change later to suit needs
        return possible_targets


    def ReduceTargets_Proximity(self, possible_targets):

        pt_unsorted = copy.copy(possible_targets)
        #define the number of generators to limit search
        proxi_lim = 100 #limit number of generators to hard number
#       proxi_lim = self.model.initial_generators * 0.25 #limit number of generators to ratio of total
network
```

```
#       pt_idnums = []
#       pt_idnums = [generator.indiv_id for generator in pt_unsorted]

#       self.pt = possible_targets
        if len(pt_unsorted) > proxi_lim:
            arcs_cost = []
            arcs_cost = [self.model.time_matrix.iloc[self.pos_id, generator.node_id] for generator in pt_unsorted]
#           print(arcs_cost)
            idx_list = list(range(len(possible_targets)))
            idx_list_sorted = sorted(idx_list, key = lambda e: arcs_cost[e])
#           print(idx_list_sorted)

            idx_list_sortshort = idx_list_sorted[:proxi_lim]
            possible_targets = [None] * proxi_lim
            for x in idx_list_sortshort:
                gen = pt_unsorted[x]
                possible_targets[idx_list_sortshort.index(x)] = gen

        return possible_targets
#           possible_targets = possible_targets_sorted[:proxi_lim-1]
#           print(possible_targets)

    def ReduceTargets_TimeLimit(self, possible_targets):
        '''
        Reduce possible targets further given criteria
        '''
        self.pts = possible_targets
        if self.model.system_options.loc['Options','VariableTransferTime']:
            possible_targets = [generator for generator in possible_targets if self.current_current_operating_time + self.model.time_matrix.iloc[self.pos_id, generator.node_id] + generator.inventory * self.model.vartransfer_time + self.model.time_matrix.iloc[generator.node_id, self.home_id] <= self.model.travel_time_limit] #make sure that total travel time will be within limits
        elif self.model.system_options.loc['Options','FixedTransferTime']:
            possible_targets = [generator for generator in possible_targets if self.current_operating_time + self.model.time_matrix.iloc[self.pos_id, generator.node_id] + self.model.fixtransfer_time + self.model.time_matrix.iloc[generator.node_id, self.home_id] <= self.model.travel_time_limit] #make sure that total travel time will be within limits
        return possible_targets


    def ReduceTargets_LoadLimit(self, possible_targets):

        possible_targets = [generator for generator in possible_targets if self.load + generator.inventory <= self.model.load_limit]
```

```
        return possible_targets


    def ReduceTargets_VisitNum(self,possible_targets):
        '''
        remove locations in possible targets if truck already visited them once
        '''

        pts_idnums = []
        pts_idnums = [generator.indiv_id for generator in possible_targets]

        for i in range(len(self.generator_dayvisit_mem)):
            if self.generator_dayvisit_mem[i] >= 1:
                remove_id = i+1
                if remove_id in pts_idnums:
                    remove_idx = pts_idnums.index(remove_id)
                    del possible_targets[remove_idx]
                    del pts_idnums[remove_idx]

        return possible_targets

    def SelectGeneratorTargetRandom(self, possible_targets):
        '''
        Select a generator target given some criteria
        '''
        target = random.choice(possible_targets)
        return target


    def SelectGeneratorTargetFWG(self, possible_targets):
        '''
        Select generator targets based on distribution of perceptions. accounts for first instance of visit and second+ instances of visits in same day.
        Visit information is individual per truck. If truck 1 visits a location first, then truck 2 visits that location, it counts as the first visit for truck 2.
        Truck 2's perception will be skewed because truck 1 decided to go to the location first. This is intended and allows for "territory" building
        '''

        #choose between taking the best path option for the step or choosing a new option based on the strength of the best path
        temp = np.random.choice(['exploit','explore'], p = [self.exploit_prob, 1 - self.exploit_prob])
        pathchoice = temp.item(0)

        self.pathchoice1.append(pathchoice)
        print(self.route_step)
```

```
        if self.model.current_day > 1 and pathchoice == 'exploit': #control for problems in best path 1) if the next step is not in the best path, 2) truck has already visted the generator already, 3) generator is not in its list of possible targets(likely because another truck has already gone there)
            print('looking for problems')
            if self.route_step >= len(self.best_route):
                pathchoice = 'explore'
            elif self.best_route[self.route_step] in self.current_route:
                pathchoice = 'explore'
            elif self.best_route[self.route_step] not in possible_targets:
                pathchoice = 'explore'

        #find another location for this bit of code
#         if self.path_prob > 0.89 \
#         and self.model.current_day % 2 == 0:
#             pathchoice = 'compare'

        self.pathchoice2.append(pathchoice)
        print(pathchoice)

        if pathchoice == 'exploit': #choose the best path result for the tour. Should be controlled for during the first day
            target = self.best_route[self.route_step]

        elif pathchoice == 'compare':
            target = self.compare_route[self.route_step]

        elif pathchoice == 'explore': #survey the network for a new option to travel to

            quantity_samples = []
            service_rev = np.zeros(len(possible_targets))
            operate_cost = np.zeros(len(possible_targets))
            profit_values = np.zeros(len(possible_targets))

            for i in range(len(possible_targets)):
                name = possible_targets[i].unique_id #index perceptions by name
                #pick perceptions of generation from possible targets, type of random choice
                if self.generator_dayvisit_mem[i] >= 1: #account for visiting generators multiple times during the day
                    quantity_pick = np.random.normal(self.generation_percept_plus.loc['Mean',name], self.generation_percept_plus.loc['Std',name], 1) #select a value from percieved distribution
                    quantity_pick = quantity_pick.item(0)
                    if quantity_pick < 0: # bring quantity pick up to zero if it is less than zero
                        quantity_pick = 0
                else:
                    #pick generation from distribution
```

```
            quantity_pick = np.random.normal(self.generation_percept.loc['Mean',name], self.generation_percept.loc['Std',name], 1) #select a value from percieved distribution
            quantity_pick = quantity_pick.item(0)
            #pick generation from expected value
            if quantity_pick < 0: # bring quantity pick up to zero if it is less than zero
               quantity_pick = 0

         quantity_samples = np.append(quantity_samples, quantity_pick)
         service_rev[i] = quantity_pick * self.model.service_fee

         if self.model.system_options.loc['Options','FuturePathscan']:
            if self.model.system_options.loc['Options','VariableTransferTime']:
               #considers traveling home with each decision
               operate_cost[i] = round((self.model.time_matrix.iloc[self.pos_id, possible_targets[i].node_id] + quantity_pick * self.model.vartransfer_time + self.model.time_matrix.iloc[possible_targets[i].node_id, self.home_id]) * self.model.operating_cost, 1)
            elif self.model.system_options.loc['Options','FixedTransferTime']:
               operate_cost[i] = round((self.model.time_matrix.iloc[self.pos_id, possible_targets[i].node_id] + self.model.fixtransfer_time + self.model.time_matrix.iloc[possible_targets[i].node_id, self.home_id]) * self.model.operating_cost, 1)

         elif self.model.system_options.loc['Options','ImmediatePathscan']:
         #Does not consider traveling home with each decision
            if self.model.system_options.loc['Options','VariableTransferTime']:
               operate_cost[i] = round((self.model.time_matrix.iloc[self.pos_id, possible_targets[i].node_id] + quantity_pick * self.model.vartransfer_time) * self.model.operating_cost, 1)
               #premptively calculate the operting cost for going to a specific location based on travel times and percieved loading time
            elif self.model.system_options.loc['Options','FixedTransferTime']:
               operate_cost[i] = round((self.model.time_matrix.iloc[self.pos_id, possible_targets[i].node_id] + self.model.fixtransfer_time) * self.model.operating_cost, 1)

         #compute profits for individual locations
         profit_values[i] = service_rev[i] - operate_cost[i]

#           targets_samples = np.append(targets_samples, profit_values[i]) #append all percieved values into an array

      #add treatment center at end of targets_samples to allow truck to cut loses and go home
      if self.route_step >= 1:
         possible_targets.append(self.model.treatment_list[0])
         travel_cost_treatment = round(self.model.time_matrix.iloc[self.pos_id, self.home_id] * self.model.operating_cost, 1)
```

```
            profit_values = np.append(profit_values, travel_cost_treatment * -1)
            operate_cost = np.append(operate_cost, travel_cost_treatment)

#           print(service_rev)
#           print(operate_cost)
#           print(profit_values)

          if self.model.system_options.loc['Options','ProfitModel']:
            target_idx = np.argmax(profit_values) #pick largest value in array as target

          elif self.model.system_options.loc['Options','CostModel']:
            target_idx = np.argmin(operate_cost) #pick least expensive value in array as target
#
#           print(possible_targets)
#           print(target_idx)

          target = possible_targets[target_idx]

      self.path_choice.append(pathchoice)
      return target


#   def BaseAntColonySelection(self, possible_targets):
#
#       operate_cost = np.zeros(len(possible_targets))
#       tops = [None] * len(possible_targets)
#       probability_list = [None] * len(possible_targets)
#
#               pathchoice = np.random.choice(['exploit','explore'], p = [self.exploit_prob, 1 -
self.exploit_prob])
#
#
#       print(pathchoice)
#
#       for j in range(len(possible_targets)):
#           #assume fixed transfer time, no case switching
#           if self.model.system_options.loc['Options','FuturePathscan']:
#                                   operate_cost[j] = round((self.model.time_matrix.iloc[self.pos_id,
possible_targets[j].node_id] + self.model.fixtransfer_time +
self.model.time_matrix.iloc[possible_targets[j].node_id, self.home_id]) *
self.model.operating_cost, 2)
#           elif self.model.system_options.loc['Options','ImmediatePathscan']:
#                                   operate_cost[j] = round((self.model.time_matrix.iloc[self.pos_id,
possible_targets[j].node_id] + self.model.fixtransfer_time) * self.model.operating_cost, 2)
#
#       for j in range(len(possible_targets)):
```

```
#                   tops[j] = self.pheromone_matrix.iloc[self.pos_id, possible_targets[j].node_id] *
((1/operate_cost[j])**self.model.beta_param)
#
#       if pathchoice == 'exploit':
#           target_idx = np.argmax(tops) #pick largest value in array
#           target = possible_targets[target_idx]
#
#       elif pathchoice == 'explore':
#           bottom = sum(tops)
#           for j in range(len(possible_targets)):
#               #create probability distribution for choosing the next target
#               probability_list[j] = tops[j]/bottom
#
#            target = np.random.choice(possible_targets, p = probability_list) #pick target based on
probability
#
#       return target


   def ImprovedAntColonySelection(self, possible_targets):

       quantity_samples = []
       service_rev = np.zeros(len(possible_targets))
       operate_cost = np.zeros(len(possible_targets))
       tops = [None] * len(possible_targets)
       probability_list = [None] * len(possible_targets)
       N = [None]* len(possible_targets)


       pathchoice = np.random.choice(['exploit','explore'], p = [self.exploit_prob, 1 -
self.exploit_prob])


#       print(pathchoice)
       #if the model is a profit model, need to perceive what will be collected from the generator
       if self.model.system_options.loc['Options','ProfitModel']:
           for i in range(len(possible_targets)):
               if self.model.system_options.loc['Options','DistSample']:
                   name = possible_targets[i].unique_id #index perceptions by name
                   #pick perceptions of generation from possible targets, type of random choice
                   if self.generator_dayvisit_mem[i] >= 1: #account for visiting generators multiple
times during the day
                       quantity_pick                                                              =
np.random.normal(self.generation_percept_plus.loc['Mean',name],
self.generation_percept_plus.loc['Std',name], 1) #select a value from percieved distribution
                       quantity_pick = quantity_pick.item(0)
```

```
                if quantity_pick < 0: # bring quantity pick up to zero if it is less than zero
                    quantity_pick = 0.001
                else:
                    #pick generation from distribution
                    quantity_pick = np.random.normal(self.generation_percept.loc['Mean',name], self.generation_percept.loc['Std',name], 1) #select a value from percieved distribution
                    quantity_pick = quantity_pick.item(0)
                    #pick generation from expected value
                    if quantity_pick < 0: # bring quantity pick up to a small number if it is less than zero
                        quantity_pick = 0.001

            elif self.model.system_options.loc['Options','MeanSample']:     #default method. don't introfduce too much variaility
                name = possible_targets[i].unique_id
                #select FW generation from mean or expected single value
#                 quantity_pick = possible_targets[i].FWgen_rate_u
                quantity_pick = self.generation_percept_plus.loc['Mean',name]

            quantity_samples = np.append(quantity_samples, quantity_pick)
            service_rev[i] = quantity_pick * self.model.service_fee

        for j in range(len(possible_targets)):
            #assume fixed transfer time, no case switching
            if self.model.system_options.loc['Options','FuturePathscan']:
                operate_cost[j] = (self.model.time_matrix.iloc[self.pos_id, possible_targets[j].node_id] + self.model.fixtransfer_time + self.model.time_matrix.iloc[possible_targets[j].node_id, self.home_id]) * self.model.operating_cost
            elif self.model.system_options.loc['Options','ImmediatePathscan']:
                operate_cost[j] = (self.model.time_matrix.iloc[self.pos_id, possible_targets[j].node_id] + self.model.fixtransfer_time) * self.model.operating_cost

            #figure out a way to append treatment facility as an option

            '"come back to this!!!!!!!!!!!!!"'
        if self.model.system_options.loc['Options','PartCollect_system']:
            possible_targets.append(self.model.treatment_list[0])
            travel_cost_treatment = (self.model.time_matrix.iloc[self.pos_id, self.home_id] + self.model.fixtransfer_time) * self.model.operating_cost
#             profit_values = np.append(profit_values, self.load - travel_cost_treatment)

#             if self.current_revenue == 0:
#                 revenue = 1
#             else:
#                 revenue = self.current_revenue
```

```
            #add treatment facility option to lineup
            service_rev = np.append(service_rev, 0)
#               service_rev = np.append(service_rev, 1)
            operate_cost = np.append(operate_cost, travel_cost_treatment)

            tops.append([None])
            N.append([None])
            probability_list.append([None])

        for j in range(len(possible_targets)):
            #Define N for determining parameters
            if self.model.system_options.loc['Options','ProfitModel']:
                denom = (self.model.maxFW * self.model.service_fee) + operate_cost[j] - service_rev[j]
#                 print(possible_targets[j].unique_id)
#                 print(str(denom))
                N[j] = 1/denom #N = inverse of distance divided by quantity at pickup location

            elif self.model.system_options.loc['Options','CostModel']:
                denom = operate_cost[j]
                if denom <= 0:
                    denom = 10000
                N[j] = 1/denom # N = inverse of cost (distance) between locations


            tops[j] = self.pheromone_matrix.iloc[self.pos_id, possible_targets[j].node_id] * (N[j]**self.model.beta_param)

#         print(N)
#         print(possible_targets)

        if pathchoice == 'exploit':
            target_idx = np.argmax(tops) #pick largest value in array
            target = possible_targets[target_idx]

        elif pathchoice == 'explore':
            bottom = sum(tops)
            for j in range(len(possible_targets)):
                #create probability distribution for choosing the next target
                probability_list[j] = tops[j]/bottom

            target = np.random.choice(possible_targets, p = probability_list) #pick target based on probability

#         print(target.unique_id)
        '''Insertion of best route found by MILP'''
```

```
        if self.model.current_day == self.model.MILPday:
            if self.model.system_options.loc['Options','InsertMILP']:
                target = self.MILPsol_route[self.route_step]

        return target

#    def PairWiseImprovement_Best(self):
#
#
#          #go through all permutations of generator nodes in route to determine a more efficient routing method
#        self.best_route_gens = self.best_route[0:-1] #extract generators only from route to shuffle
#        treatment = self.best_route[-1] #extract treatment facility from route
#
#        print('tick1')
#
#        best_route_genperms_tuple = []
#        for p in itertools.permutations(self.best_route_gens):
#            if p[0].node_id < p[-1].node_id:
#                best_route_genperms_tuple.append(p)
#
#        print('tick2')
#
#        best_route_perms = []
#            for i in range(len(best_route_genperms_tuple)): #convert the tuple back into a list to manipulate and add on treatment to end
#            temp_route = best_route_genperms_tuple[i]
#            temp_route = list(temp_route)
#            temp_route.append(treatment)
#            temp_route.insert(0, treatment)
#            best_route_perms.append(temp_route)
#
#        print('tick2.1')
#
#        perm_route_profits = []
#        perm_route_costs = []
#
#        temp_rev = self.best_revenue
#        tm = self.model.time_matrix
#        ftt = self.model.fixtransfer_time
#        moc = self.model.operating_cost
#
#        def recost_route(i, route):
#            temp_cost = 0
#            temp_profit = 0
#            arc_values_list = []
```

```
#            for j in range(len(route) - 1):
#                print('tick2.' + str(i) + '.' + str(j))
#                arc_values_list.append((tm.iloc[route[j].node_id, route[j + 1].node_id] + ftt) * moc)
#            temp_cost = sum(arc_values_list)
#            temp_profit = temp_rev - temp_cost
#            return temp_cost, temp_profit
#
#        for i in range(len(best_route_perms)):
#            route = best_route_perms[i]
#            temp_cost, temp_profit = recost_route(i, route)
#            perm_route_profits.append(temp_profit)
#            perm_route_costs.append(temp_cost)
#
#        print('tick2.2')
#
#        if self.model.system_options.loc['Options','ProfitModel']:
#          best_obj_idx = np.argmax(perm_route_profits) #pick best permutation of route depending
on objective type
#        if self.model.system_options.loc['Options','CostModel']:
#            best_obj_idx = np.argmin(perm_route_costs)
#
#        print('tick3')
#
#
#        #compare orders of best route and index them to make changes to pathways
#
#        self.previous_best_route = copy.copy(self.best_route)
#        self.previous_best_route_ids = copy.copy(self.best_route_ids)
#        self.new_best_route = best_route_perms[best_obj_idx]
#        #remove treatment facility from front of list
#        del(self.new_best_route[0])
#
#
#        self.pbr_idx = []
#        self.nbr_idx = []
#        for i in range(len(self.previous_best_route)):
#            print(str(i))
#            self.gen = self.previous_best_route[i]
#            self.pbr_idx.append(self.previous_best_route.index(self.gen))
#            self.nbr_idx.append(self.new_best_route.index(self.gen))
#
#        self.best_route = self.new_best_route
#        self.best_objective = perm_route_profits[best_obj_idx]
#        self.best_cost = perm_route_costs[best_obj_idx]
#
##        self.best_route_ids = [item.unique_id for item in self.best_route]
```

```
#
#       br_len = len(self.previous_best_route)
#       temp_brID = [None]* br_len
#       temp_brP = [None]* br_len
#       temp_brR = [None]* br_len
#       temp_brC = [None]* br_len
#       temp_brE = [None]* br_len
#
#       print('tick4')
#
#       for i in range(len(self.pbr_idx)):
#           j =  self.nbr_idx[i]
#           temp_brID[i] = self.best_route_ids[j]
#           temp_brP[i] = self.best_route_pickups[j]
#           temp_brR[i] = self.best_route_revenues[j]
#           temp_brC[i] = self.best_route_costs[j]
#           temp_brE[i] = self.best_route_effs[j]
#
#       self.best_route_ids = temp_brID
#       self.best_route_pickups = temp_brP
#       self.best_route_revenues = temp_brR
#       self.best_route_costs = temp_brC
#       self.best_route_effs = temp_brE
#
#       self.best_pairwise_improvement = 1
#
#       print('tick5')
#
#   def PairWiseImprovement_Current(self):
#
#
#          #go through all permutations of generator nodes in route to determine a more efficient routing method
#       current_route_gens = self.current_route[0:-1] #extract generators only from route to shuffle
#       treatment = self.current_route[-1] #extract treatment facility from route
#
#       current_route_genperms_tuple = list(permutations(current_route_gens))
#
#       current_route_perms = []
#          for i in range(len(current_route_genperms_tuple)): #convert the tuple back into a list to manipulate and add on treatment to end
#           temp_route = current_route_genperms_tuple[i]
#           temp_route = list(temp_route)
#           temp_route.append(treatment)
#           current_route_perms.append(temp_route)
#
```

```
#       perm_route_profits = []
#       perm_route_costs = []
#
#       for i in range(len(current_route_perms)): #calculate the cost of route permutations
#           perm_i = current_route_perms[i]
#           temp_rev = self.current_revenue
#           temp_cost = 0
#           temp_profit = 0
#           for j in range(len(perm_i) - 1):
#               temp_cost += (self.model.time_matrix.iloc[perm_i[j].node_id, perm_i[j + 1].node_id]
+ self.model.fixtransfer_time) * self.model.operating_cost
#
#           temp_profit = temp_rev - temp_cost
#
#          perm_route_profits = np.append(perm_route_profits, temp_profit) #put all objectives into
a list to evaluate
#           perm_route_costs = np.append(perm_route_costs, temp_cost)
#
#       if self.model.system_options.loc['Options','ProfitModel']:
#          best_obj_idx = np.argmax(perm_route_profits) #pick best permutation of route depending
on objective type
#
#       if self.model.system_options.loc['Options','CostModel']:
#           best_obj_idx = np.argmin(perm_route_costs)
#
#       #compare orders of best route and index them to make changes to pathways
#       self.previous_current_route = copy.copy(self.current_route)
#       self.previous_current_route_ids = copy.copy(self.current_route_ids)
#       new_current_route = current_route_perms[best_obj_idx]
#
#       self.pcr_idx = []
#       self.ncr_idx = []
#       for i in range(len(self.previous_current_route)):
#           gen = self.previous_current_route[i]
#           self.pcr_idx.append(self.previous_current_route.index(gen))
#           self.ncr_idx.append(new_current_route.index(gen))
#
#       self.current_route = new_current_route
#       self.current_objective = perm_route_profits[best_obj_idx]
#       self.current_cost = perm_route_costs[best_obj_idx]
#
##       self.best_route_ids = [item.unique_id for item in self.best_route]
#
#       br_len = len(self.previous_current_route)
#       temp_brID = [None]* br_len
#       temp_brP = [None]* br_len
```

```
#       temp_brR = [None]* br_len
#       temp_brC = [None]* br_len
#       temp_brE = [None]* br_len
#
#
#       for i in range(len(self.pcr_idx)):
#           j =  self.ncr_idx[i]
#           temp_brID[i] = self.current_route_ids[j]
#           temp_brP[i] = self.current_route_pickups[j]
#           temp_brR[i] = self.current_route_revenues[j]
#           temp_brC[i] = self.current_route_costs[j]
#           temp_brE[i] = self.current_route_effs[j]
#
#       self.current_route_ids = temp_brID
#       self.current_route_pickups = temp_brP
#       self.current_route_revenues = temp_brR
#       self.current_route_costs = temp_brC
#       self.current_route_effs = temp_brE
#
#       self.current_pairwise_improvement = 1


    def SelectHomeTarget(self):
        '''
        Set next target to home; the treatment facility wher ethe truck started
        '''
        home = [treatment for treatment in self.model.treatment_list if treatment.node_id ==
self.home_id]
        target = home[0]
        return target


    def MoveToTarget(self, target):
        '''
        Move to the target
        '''
        travel_time = round(self.model.time_matrix.iloc[self.pos_id, target.node_id],4)

        self.current_operating_time += travel_time
        self.current_cost = self.current_operating_time * self.model.operating_cost

        self.pos_id = target.node_id
        self.pos_desc = target.node_desc
        self.current_route.append(target)
        self.current_route_ids.append(target.unique_id)
```

```
        self.route_step += 1
        return(travel_time)

    def LoadFoodWaste(self, target, travel_time):
        '''
        Load FW into truck. Update truck memory to reflect what it can collect next time it comes to the generator
        '''

        target_id = target.indiv_id - 1


        if self.generator_dayvisit_mem[target_id] >= 1:
            self.pickup_mem_plus[target_id] = np.append(self.pickup_mem_plus[target_id], target.inventory)
        else:
            self.pickup_mem[target_id] = np.append(self.pickup_mem[target_id], target.inventory)

        '''
        initialize section to update perceptions of generation
        '''

        known_stdv = self.model.FWgen_rate.loc['Std', target.unique_id]

        if self.generator_dayvisit_mem[target_id] >= 1:
            mu_prior = self.generation_percept_plus.loc['Mean', target.unique_id]
            stdv_prior =  self.generation_percept_plus.loc['Std', target.unique_id]

        else:
            mu_prior = self.generation_percept.loc['Mean', target.unique_id]
            stdv_prior =  self.generation_percept.loc['Std', target.unique_id]

        #update truck memory and perception here

        mu_post = ((known_stdv**2 * mu_prior) + (stdv_prior**2 * target.inventory))/(stdv_prior**2 + known_stdv**2)
        stdv_post = np.sqrt((stdv_prior**2 * known_stdv**2)/(stdv_prior**2 + known_stdv**2))


        if self.generator_dayvisit_mem[target_id] >= 1:
            self.generation_percept_plus.loc['Mean',target.unique_id] = round(mu_post, 5)
            self.generation_percept_plus.loc['Std',target.unique_id] = round(stdv_post, 5)
        else:
            self.generation_percept.loc['Mean',target.unique_id] = round(mu_post, 5)
            self.generation_percept.loc['Std',target.unique_id] = round(stdv_post, 5)
```

```
        '''
        End update section
        '''

        '''
        Initialize to not update perceptions
        '''
        #do nothing!

        '''
        end of section
        '''

        if self.model.system_options.loc['Options','FullCollect_gen']: #transfer full amount of generator inventory
            transfer = copy.copy(target.inventory)
        elif self.model.system_options.loc['Options','PartCollect_gen']: #transfer only what truck can take
            avail_space = self.model.load_limit - self.load
            if target.inventory <= avail_space:
                transfer = target.inventory
            elif target.inventory > avail_space:
                transfer = avail_space

        self.load += transfer #transfer inventory to truck
        self.current_route_pickups.append(transfer)
        target.inventory -= transfer # remove inventory from generator
        if target.inventory <= 0: #check to see if any FW remaining
            target.pickup_request = 0 #if no FW remaining, set pickup request to 0

        if self.model.system_options.loc['Options','VariableTransferTime']:
            self.current_operating_time += round(transfer * self.model.vartransfer_time,4)
            self.current_cost += round(transfer * self.model.vartransfer_time * self.model.operating_cost, 4)
        elif self.model.system_options.loc['Options','FixedTransferTime']:
            self.current_operating_time += round(self.model.fixtransfer_time,4)
            self.current_cost = self.current_operating_time * self.model.operating_cost

        self.current_revenue += round(transfer * self.model.service_fee, 4)
        self.current_route_costs.append(round((self.model.fixtransfer_time + travel_time) * self.model.operating_cost, 4))
        self.current_route_revenues.append(round(transfer * self.model.service_fee, 4))
        self.current_route_effs.append(round((transfer * self.model.service_fee)/((self.model.fixtransfer_time + travel_time) * self.model.operating_cost), 4))
```

```
        if self.generator_dayvisit_mem[target_id] >=1:
            self.generators_total_visits_plus[target_id] += 1
        else:
            self.generators_total_visits[target_id] += 1

        self.generator_dayvisit_mem[target_id] += 1 #update visiting memory

        '''Print outputs if needed'''
#         print(self.unique_id + ' loaded ' + str(transfer) + ' kg FW from ' + target.unique_id)
#         print(self.unique_id + ' load: ' + str(self.load) + ' kg FW')
#         print(self.unique_id + ' travel time: ' + str(self.current_operating_time) + ' minutes')


    def UnloadFoodWaste(self, target, travel_time):

#         if self.load > 0:
        target.inventory += self.load #transfer load to treatment plant
        dropped = self.load
        self.load -= self.load #remove load

        if dropped > 0:
            self.current_route_costs.append(round((self.model.fixtransfer_time + travel_time) *
self.model.operating_cost, 2))

            if self.model.system_options.loc['Options','VariableTransferTime']:
                self.current_operating_time += round(dropped * self.model.vartransfer_time,4)
                self.current_cost += round(dropped * self.model.vartransfer_time *
self.model.operating_cost, 4)
            elif self.model.system_options.loc['Options','FixedTransferTime']:
                self.current_operating_time += round(self.model.fixtransfer_time,0)
                self.current_cost = self.current_operating_time * self.model.operating_cost

        self.current_route_pickups.append(dropped*-1)
        self.current_route_revenues.append(0)
        self.current_route_effs.append(0)
        self.tours += 1
#
        '''Print outputs if needed'''
#         print(self.unique_id + ' unloaded ' + str(dropped) + ' kg FW at ' + target.unique_id)
#         print(self.unique_id + ' load: ' + str(self.load) + ' kg FW')
#         print(target.unique_id + ' inventory: ' + str(target.inventory) + ' kg FW')
#             print(self.unique_id + ' travel time for day ' + str(self.model.current_day) + ': ' +
str(self.current_operating_time) + ' minutes')


    def StatusCheck(self): #tally up objectives and tours and stuff at the end of the day
```

```
        if self.tours >= self.model.tour_limit:
            '''
            THIS IS VERY IMPORTANT TO DECIDE WHAT THE OBJECTIVE TYPE WILL BE
            '''
            self.current_profit = self.current_cost - self.current_revenue



            if self.model.system_options.loc['Options','ProfitModel']: #tally the objective for the tour
                self.current_objective =  copy.copy(self.current_profit)
#                 self.model.current_objective += self.current_objective #update the global objective

            elif self.model.system_options.loc['Options','CostModel']:
                self.current_objective = copy.copy(self.current_profit)
#                 self.model.current_objective += self.current_objective #update the global objective

            if self.model.current_day == 1:
                self.objective_hist =  copy.copy(self.current_objective)
                self.route_hist = copy.copy([self.current_route])
                self.route_hist_ids = copy.copy([self.current_route_ids])
                self.route_pickup_hist = copy.copy([self.current_route_pickups])

            else:
                self.objective_hist = np.append(self.objective_hist, self.current_objective)
                self.route_hist.append(self.current_route)
                self.route_hist_ids.append(self.current_route_ids)
                self.route_pickup_hist.append(self.current_route_pickups)

            self.active = 0 #deactivate self
#             print(self.model.current_day)
```

```
'''
Sundial and Reset Functions V3
'''

import numpy as np
import copy
import pandas as pd
import random

#from mymodel import WasteCollection
from mesa import Agent

class TimeKeeper_V3(Agent):

    def __init__(self, unique_id, model):
        '''
        Class implementing driving behavior of trucks
        '''

        super().__init__(unique_id, model)


    def ImprovedPheromoneUpdate(self):

        #Assume cost model
        #redundant if done in truck sttus check
        to_add_obj = np.zeros(self.model.initial_trucks)
        to_add_rev = np.zeros(self.model.initial_trucks)
        to_add_cost = np.zeros(self.model.initial_trucks)
        to_add_prof = np.zeros(self.model.initial_trucks)
        for i in range(len(self.model.truck_list)): #cycle through trucks to create a list of current objectives for trucks
            truck = self.model.truck_list[i]
            to_add_obj[i] = truck.current_objective
            to_add_rev[i] = truck.current_revenue
            to_add_cost[i] = truck.current_cost
            to_add_prof[i] = truck.current_profit

        #add together individual truck objectives for full model objective
        temp_objective = np.sum(to_add_obj)
        temp_rev = np.sum(to_add_rev)
        temp_cost = np.sum(to_add_cost)
        temp_prof = np.sum(to_add_prof)

        self.model.current_objective = copy.copy(temp_objective)
```

```
        self.model.current_revenue = copy.copy(temp_rev)
        self.model.current_cost = copy.copy(temp_cost)
        self.model.current_profit = copy.copy(temp_prof)

        if self.model.current_day == 1: # fill in everything if this is the first time initializing a route
            self.model.best_objective = copy.copy(self.model.current_objective)
            self.model.best_revenue = copy.copy(self.model.current_revenue)
            self.model.best_cost = copy.copy(self.model.current_cost)
            self.model.best_profit = copy.copy(self.model.current_profit)

            self.model.objective_hist = copy.copy(self.model.current_objective)
            for i in range(len(self.model.truck_list)): #cycle through trucks
                truck = self.model.truck_list[i]
                truck.best_objective = copy.copy(truck.current_objective)
                truck.best_revenue = copy.copy(truck.current_revenue)
                truck.best_cost = copy.copy(truck.current_cost)
                truck.best_profit = copy.copy(truck.current_profit)
                truck.best_route = truck.current_route.copy()
                truck.best_route_ids = truck.current_route_ids.copy()
                truck.best_route_pickups = truck.current_route_pickups.copy()
                truck.best_route_revenues = truck.current_route_revenues.copy()
                truck.best_route_costs = truck.current_route_costs.copy()
                truck.best_route_effs = truck.current_route_effs.copy()
                truck.best_day = copy.copy(self.model.current_day)
                truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter for best
route
                truck.best_operating_time = copy.copy(truck.current_operating_time)
                self.model.best_day = copy.copy(self.model.current_day)

                #add treatment facility to beginning of route to correctly update pheromones
                current_route = copy.copy(truck.current_route)
                current_route.insert(0,current_route[-1])
                #update pheromone levels



                if self.model.system_options.loc['Options','ProfitModel']:
                    #scale profit denominator so remain positive
                    denom = ((self.model.maxFW/self.model.initial_trucks) * self.model.service_fee) +
truck.best_cost - truck.best_revenue
                    N = 1/denom #N = inverse of distance divided by quantity at pickup location

                elif self.model.system_options.loc['Options','CostModel']:
                        denom = truck.best_cost
                        if truck.best_cost <= 0:
                            denom = 10000
```

```
                N = 1/denom # N = inverse of cost (distance) between locations 1

            for j in range(len(current_route) - 1):
                pt_link = truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j + 1].node_id]
                truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j + 1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
#                 truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j + 1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * 1/truck.best_cost


        else:
            #add current objective to history
            self.model.objective_hist = np.append(self.model.objective_hist, copy.copy(self.model.current_objective))
#
#           if self.model.system_options.loc['Options','ProfitModel']:
#           #update best objective as appropriate
#               if self.model.best_objective < self.model.current_objective:
#                   self.model.best_objective = copy.copy(self.model.current_objective)
#                   self.model.best_revenue = copy.copy(self.model.current_revenue)
#                   self.model.best_cost = copy.copy(self.model.current_cost)
#                   self.model.best_profit = copy.copy(self.model.current_profit)
#                   for i in range(len(self.model.truck_list)): #cycle through trucks
#                       truck = self.model.truck_list[i]
#                       truck.best_objective = copy.copy(truck.current_objective)
#                       truck.best_revenue = copy.copy(truck.current_revenue)
#                       truck.best_cost = copy.copy(truck.current_cost)
#                       truck.best_profit = copy.copy(truck.current_profit)
#                       truck.best_route = truck.current_route.copy()
#                       truck.best_route_ids = truck.current_route_ids.copy()
#                       truck.best_route_pickups = truck.current_route_pickups.copy()
#                       truck.best_route_revenues = truck.current_route_revenues.copy()
#                       truck.best_route_costs = truck.current_route_costs.copy()
#                       truck.best_route_effs = truck.current_route_effs.copy()
#                       truck.best_day = copy.copy(self.model.current_day)
#                         truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter for best route
#                       truck.best_operating_time = copy.copy(truck.current_operating_time)
#                       self.model.best_day = copy.copy(self.model.current_day)
#
#                       #add treatment facility to beginning of route to correctly update pheromones
#                       best_route = copy.copy(truck.best_route)
#                       best_route.insert(0,best_route[-1])
#
##                         if truck.best_revenue == 0:
```

```
##                         best_revenue = 1
##                     else:
##                         best_revenue = copy.copy(truck.best_revenue)
#
#
##                     if truck.current_cost > 0:
#                       #scale denominator according to network revenue
#                           denom = (self.model.maxFW * self.model.service_fee) + truck.best_cost -
truck.best_revenue
#                   N = 1/denom #N = inverse of distance divided by quantity at pickup location
#
#                   for j in range(len(best_route) - 1): #update the pheromones on the newfound best
route
#                       pt_link = truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j +
1].node_id]
#                       truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id]
= (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
##                      truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id]
= (1 - self.model.alpha_param) * pt_link + N
#
#
#
#           elif self.model.best_objective >= self.model.current_objective:
#               for i in range(len(self.model.truck_list)): #cycle through trucks
#                   truck = self.model.truck_list[i]
#
#                   #add treatment facility to beginning of route to correctly update pheromones
#                   current_route = copy.copy(truck.current_route)
#                   current_route.insert(0,current_route[-1])
#                   best_route = copy.copy(truck.best_route)
#                   best_route.insert(0,best_route[-1])
#
##                     if truck.best_revenue == 0:
##                         best_revenue = 1
##                     else:
##                         best_revenue = copy.copy(truck.best_revenue)
#
##                     if truck.current_cost > 0:
#                       #Scale denominator to be positive based on maximum revenue
#                           denom = (self.model.maxFW * self.model.service_fee) + truck.best_cost -
truck.best_revenue
#                   N = 1/denom  #N = inverse of distance divided by quantity at pickup location
#
#                  for j in range(len(best_route) - 1): #update pheromones of best route used by elitist
ants
```

```
#                         pt_link = truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j +
1].node_id]
#                       truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id]
= (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
##                      truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id]
= (1 - self.model.alpha_param) * pt_link + N
#
#
#                    #update pheromones on current route
#                   for j in range(len(current_route) - 1): #update pheromones of current route traveled
by truck this day
#                     pt_link = truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j
+ 1].node_id]
##                            truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
#                           truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.gamma_param *
self.model.alpha_param * N
##                          truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id] = (1 - self.model.alpha_param) * pt_link + N
#

#           elif self.model.system_options.loc['Options','CostModel']:
          #update best objective as appropriate
          if self.model.best_objective > self.model.current_objective:
             self.model.best_objective = copy.copy(self.model.current_objective)
             self.model.best_revenue = copy.copy(self.model.current_revenue)
             self.model.best_cost = copy.copy(self.model.current_cost)
             self.model.best_profit = copy.copy(self.model.current_profit)

             for i in range(len(self.model.truck_list)): #cycle through trucks
                truck = self.model.truck_list[i]

#                 truck.exploit_prob = 0
#                 truck.exploit_prob_hist.append(truck.exploit_prob)

                truck.best_objective = copy.copy(truck.current_objective)
                truck.best_revenue = copy.copy(truck.current_revenue)
                truck.best_cost = copy.copy(truck.current_cost)
                truck.best_profit = copy.copy(truck.current_profit)
                truck.best_route = truck.current_route.copy()
                truck.best_route_ids = truck.current_route_ids.copy()
                truck.best_route_pickups = truck.current_route_pickups.copy()
                truck.best_route_revenues = truck.current_route_revenues.copy()
                truck.best_route_costs = truck.current_route_costs.copy()
                truck.best_route_effs = truck.current_route_effs.copy()
```

```
                truck.best_day = copy.copy(self.model.current_day)
                truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter for best route
                truck.best_operating_time = copy.copy(truck.current_operating_time)
                self.model.best_day = copy.copy(self.model.current_day)

                #add treatment facility to beginning of route to correctly update pheromones
                best_route = copy.copy(truck.best_route)
                best_route.insert(0, best_route[-1])



                if self.model.system_options.loc['Options','ProfitModel']:
                    #scale profit denominator so remain positive
                    denom = ((self.model.maxFW/self.model.initial_trucks) * self.model.service_fee) + truck.best_cost - truck.best_revenue
                    N = 1/denom #N = inverse of distance divided by quantity at pickup location

                elif self.model.system_options.loc['Options','CostModel']:
                    denom = truck.best_cost
                    if truck.best_cost <= 0:
                        denom = 10000
                    N = 1/denom # N = inverse of cost (distance) between locations 2

                for j in range(len(best_route) - 1): #update the pheromones on the newfound best route
                    pt_link = truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id]
                    truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
#                     truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id] = (1 - self.model.alpha_param) * pt_link + N

            elif self.model.best_objective <= self.model.current_objective:
                for i in range(len(self.model.truck_list)): #cycle through trucks
                    truck = self.model.truck_list[i]
#                     truck.exploit_prob += copy.copy(self.model.exploit_prob_update)
#                     if truck.exploit_prob > self.model.exploit_prob_limit:
#                         truck.exploit_prob = self.model.exploit_prob_limit
#                     truck.exploit_prob_hist.append(truck.exploit_prob)
                    #add treatment facility to beginning of route to correctly update pheromones
                    current_route = copy.copy(truck.current_route)
                    current_route.insert(0,current_route[-1])
                    best_route = copy.copy(truck.best_route)
                    best_route.insert(0,best_route[-1])
```

```
            if self.model.system_options.loc['Options','ProfitModel']:
                #scale profit denominator so remain positive
                denom = ((self.model.maxFW/self.model.initial_trucks) * self.model.service_fee)
+ truck.best_cost - truck.best_revenue
                N = 1/denom #N = inverse of distance divided by quantity at pickup location

            elif self.model.system_options.loc['Options','CostModel']:
                denom = truck.best_cost
                if truck.best_cost <= 0:
                    denom = 10000
                N = 1/denom # N = inverse of cost (distance) between locations 3

            for j in range(len(best_route) - 1): #update pheromones of best route used by elitist
ants
                pt_link = truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j +
1].node_id]
                truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id] =
(1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
   #                 truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id] =
(1 - self.model.alpha_param) * pt_link + N

            #update pheromones on current route
            for j in range(len(current_route) - 1): #update pheromones of current route traveled
by truck this day
                pt_link = truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id]
   #                     truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * N
                truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.gamma_param *
self.model.alpha_param * N
   #                     truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.gamma_param * N


    def NewDayReset(self):

        self.model.current_objective = 0
        self.model.treatment_list[0].inventory = 0

        for i in range(len(self.model.truck_list)): #cycle through trucks
            truck = self.model.truck_list[i]

            truck.active = 1 #reactivate trucks
            truck.current_operating_time = 0 #reset travel times for truck
            truck.tours = 0 #reset tours
```

```
            truck.route_step = 0

            if self.model.system_options.loc['Options','ProfitModel']:
                truck.current_cost = 0
                truck.current_objective = 0 #reset objectives for the day

            elif self.model.system_options.loc['Options','CostModel']:
                truck.current_cost = 0
                truck.current_objective = 0 #reset objectives for the day

            truck.current_revenue = 0
            truck.current_profit = 0

            truck.current_route = [] #reset route
            truck.current_route_ids = []
            truck.current_route_pickups = []
            truck.compare_route = []
            truck.current_route_revenues = []
            truck.current_route_costs = []
            truck.current_route_effs = []
            truck.current_pairwise_improvement = 0
#           truck.current_route_finaleff = 0
            for j in range(len(self.model.generator_list)):
                truck.generator_dayvisit_mem[j] = 0









#    def CompareObjectives(self):
#
#        #unecessary if truck status check updates model objective as model runs
#        to_add_obj = np.zeros(self.model.initial_trucks)
#        to_add_rev = np.zeros(self.model.initial_trucks)
#        to_add_cost = np.zeros(self.model.initial_trucks)
#        to_add_prof = np.zeros(self.model.initial_trucks)
#          for i in range(len(self.model.truck_list)): #cycle through trucks to create a list of current
objectives for trucks
#            to_add_obj[i] = self.model.truck_list[i].current_objective
#
#
#        #add together individual truck objectives for full model objective
```

```
#       temp_objective = np.sum(to_add)
#       self.model.current_objective = copy.copy(temp_objective)
#
#       if self.model.current_day == 1: # fill in everything if this is the first time initializing a route
#          self.model.best_objective = copy.copy(self.model.current_objective)
#          self.model.objective_hist = copy.copy(self.model.current_objective)
#          for i in range(len(self.model.truck_list)): #cycle through trucks
#             truck = self.model.truck_list[i]
#             truck.best_objective = copy.copy(truck.current_objective)
#             truck.best_route = truck.current_route.copy()
#             truck.best_route_ids = truck.current_route_ids.copy()
#             truck.best_route_pickups = truck.current_route_pickups.copy()
#             truck.best_route_revenues = truck.current_route_revenues.copy()
#             truck.best_route_costs = truck.current_route_costs.copy()
#             truck.best_route_effs = truck.current_route_effs.copy()
#             truck.best_cost = copy.copy(truck.current_cost)
#             truck.best_revenue = copy.copy(truck.current_revenue)
#             truck.best_profit = copy.copy(truck.current_profit)
#             truck.best_day = copy.copy(self.model.current_day)
#            truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter for best
route
#
#       else:
#          #add current objective to history
#                                  self.model.objective_hist   =   np.append(self.model.objective_hist,
copy.copy(self.model.current_objective))
#
#          if self.model.system_options.loc['Options','ProfitModel']:
#
#             #update best objective for the profit model
#             if self.model.best_objective < self.model.current_objective:
#                self.model.best_objective = copy.copy(self.model.current_objective)
#                for i in range(len(self.model.truck_list)): #cycle through trucks
#                   truck = self.model.truck_list[i]
#                   truck.best_objective = copy.copy(truck.current_objective)
#                   truck.best_revenue = copy.copy(truck.current_revenue)
#                   truck.best_cost = copy.copy(truck.current_cost)
#                   truck.best_profit = copy.copy(truck.current_profit)
#                   truck.best_route = truck.current_route.copy()
#                   truck.exploit_prob = copy.copy(self.model.exploit_prob_update)
#                   truck.best_route_ids = truck.current_route_ids.copy()
#                   truck.best_route_pickups = truck.current_route_pickups.copy()
#                   truck.best_route_revenues = truck.current_route_revenues.copy()
#                   truck.best_route_costs = truck.current_route_costs.copy()
#                   truck.best_route_effs = truck.current_route_effs.copy()
#                   truck.best_day = copy.copy(self.model.current_day)
```

```
#                     truck.exploit_prob_hist.append(truck.exploit_prob)
#                       truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter
for best route
#
#               elif self.model.best_objective >= self.model.current_objective:
#                   for i in range(len(self.model.truck_list)): #cycle through trucks
#                       truck = self.model.truck_list[i]
#                       if self.model.system_options.loc['Options','FWGND']:
#                           truck.exploit_prob += copy.copy(self.model.exploit_prob_update)
#                           if truck.exploit_prob >= self.model.exploit_prob_limit:
#                            truck.exploit_prob = self.model.exploit_prob_limit #set cutoff so that pathprob
does not go over 1
#                       truck.exploit_prob_hist.append(truck.exploit_prob)
#
#           elif self.model.system_options.loc['Options','CostModel']:
#
#               #update best objective for the cost model
#               if self.model.best_objective >= self.model.current_objective:
#                   self.model.best_objective = copy.copy(self.model.current_objective)
#                   for i in range(len(self.model.truck_list)): #cycle through trucks
#                       truck = self.model.truck_list[i]
#                       truck.best_objective = copy.copy(truck.current_objective)
#                       truck.best_revenue = copy.copy(truck.current_revenue)
#                       truck.best_cost = copy.copy(truck.current_cost)
#                       truck.best_profit = copy.copy(truck.current_profit)
#                       truck.best_route = truck.current_route.copy()
#                       truck.exploit_prob = copy.copy(self.model.exploit_prob_update)
#                       truck.best_route_ids = truck.current_route_ids.copy()
#                       truck.best_route_pickups = truck.current_route_pickups.copy()
#                       truck.best_route_revenues = truck.current_route_revenues.copy()
#                       truck.best_route_costs = truck.current_route_costs.copy()
#                       truck.best_day = copy.copy(self.model.current_day)
#                       truck.best_route_effs = truck.current_route_effs.copy()
#                         truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter
for best route
#
#
#               elif self.model.best_objective <= self.model.current_objective:
#                   for i in range(len(self.model.truck_list)): #cycle through trucks
#                       truck = self.model.truck_list[i]
#                       if self.model.system_options.loc['Options','FWGND']:
#                           truck.exploit_prob += copy.copy(self.model.exploit_prob_update)
#                           if truck.exploit_prob >= self.model.exploit_exploit_limit:
#                                   truck.exploit_prob = self.model.exploit_exploit_limit #set cutoff so that
pathprob does not go over 1
#
```

```
#
#
#   def BasePheromoneUpdate(self):
#
#       #Assume cost model
#       #redundant if done in truck sttus check
#       to_add = np.zeros(self.model.initial_trucks)
#         for i in range(len(self.model.truck_list)): #cycle through trucks to create a list of current
objectives for trucks
#           truck = self.model.truck_list[i]
#           to_add[i] = truck.current_objective
#
#       #add together individual truck objectives for full model objective
#       temp_objective = np.sum(to_add)
#       self.model.current_objective = copy.copy(temp_objective)
#
#       if self.model.current_day == 1: # fill in everything if this is the first time initializing a route
#           self.model.best_objective = copy.copy(self.model.current_objective)
#           self.model.objective_hist = copy.copy(self.model.current_objective)
#           for i in range(len(self.model.truck_list)): #cycle through trucks
#               truck = self.model.truck_list[i]
#               truck.best_objective = copy.copy(truck.current_objective)
#               truck.best_revenue = copy.copy(truck.current_revenue)
#               truck.best_cost = copy.copy(truck.current_cost)
#               truck.best_profit = copy.copy(truck.current_profit)
#               truck.best_route = truck.current_route.copy()
#               truck.best_route_ids = truck.current_route_ids.copy()
#               truck.best_route_pickups = truck.current_route_pickups.copy()
#               truck.best_route_revenues = truck.current_route_revenues.copy()
#               truck.best_route_costs = truck.current_route_costs.copy()
#               truck.best_route_effs = truck.current_route_effs.copy()
#               truck.best_day = copy.copy(self.model.current_day)
#              truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter for best
route
#               self.model.best_day = copy.copy(self.model.current_day)
#
#               #add treatment facility to beginning of route to correctly update pheromones
#               current_route = copy.copy(truck.current_route)
#               current_route.insert(0,current_route[-1])
#               #update pheromone levels
#               for j in range(len(current_route) - 1):
#                   pt_link = truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j +
1].node_id]
#                 truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j + 1].node_id]
= (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * (1/truck.best_cost)
#
```

```
#       else:
#           #add current objective to history
#                                   self.model.objective_hist   =   np.append(self.model.objective_hist,
copy.copy(self.model.current_objective))
#
#       #base ACO model operates on cost only
#           #update best objective as appropriate
#           if self.model.best_objective > self.model.current_objective:
#               self.model.best_objective = copy.copy(self.model.current_objective)
#               for i in range(len(self.model.truck_list)): #cycle through trucks
#                   truck = self.model.truck_list[i]
#                   truck.best_objective = copy.copy(truck.current_objective)
#                   truck.best_revenue = copy.copy(truck.current_revenue)
#                   truck.best_cost = copy.copy(truck.current_cost)
#                   truck.best_profit = copy.copy(truck.current_profit)
#                   truck.best_route = truck.current_route.copy()
#                   truck.best_route_ids = truck.current_route_ids.copy()
#                   truck.best_route_pickups = truck.current_route_pickups.copy()
#                   truck.best_route_revenues = truck.current_route_revenues.copy()
#                   truck.best_route_costs = truck.current_route_costs.copy()
#                   truck.best_route_effs = truck.current_route_effs.copy()
#                   truck.best_day = copy.copy(self.model.current_day)
#                    truck.best_pairwise_improvement = 0 #reset the pairwise improvement counter for
best route
#                   self.model.best_day = copy.copy(self.model.current_day)
#
#                   #add treatment facility to beginning of route to correctly update pheromones
#                   best_route = copy.copy(truck.best_route)
#                   best_route.insert(0,best_route[-1])
#
#                     for j in range(len(best_route) - 1): #update the pheromones on the newfound best
route
#                           pt_link = truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j +
1].node_id]
#                       truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id] =
(1 - self.model.alpha_param) * pt_link + self.model.alpha_param * (1/truck.best_cost)
#
#           elif self.model.best_objective <= self.model.current_objective:
#               for i in range(len(self.model.truck_list)): #cycle through trucks
#                   truck = self.model.truck_list[i]
#
#                   #add treatment facility to beginning of route to correctly update pheromones
#                   current_route = copy.copy(truck.current_route)
#                   current_route.insert(0,current_route[-1])
#                   best_route = copy.copy(truck.best_route)
#                   best_route.insert(0,best_route[-1])
```

```
#
#                 for j in range(len(best_route) - 1): #update pheromones of best route used by elitist ants
#                         pt_link = truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id]
#                     truck.pheromone_matrix.iloc[best_route[j].node_id, best_route[j + 1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * (1/truck.best_cost)
#                 #update pheromones on current route
#                 for j in range(len(current_route) - 1): #update pheromones of current route traveled by truck this day
#                     pt_link = truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j + 1].node_id]
#                 truck.pheromone_matrix.iloc[current_route[j].node_id, current_route[j + 1].node_id] = (1 - self.model.alpha_param) * pt_link + self.model.alpha_param * (1/truck.best_cost)
```

MATLAB benchmark code main function

```
%Main on google drive
%Will Armington
%Residential collection and routing

clear;
delete '*.xlsx'
delete '*.fig'
delete '*.png'
addpath('C:\Program Files\IBM\ILOG\CPLEX_Studio128\cplex\matlab\x64_win64')

tic %start timer

%Current formulation: trucks can only visit each node in network once
%Must designate maximum number of times a truck can drop FW off at intermediate
facility

%inputs

filename                                                                    =
"D:\INFEWS\EcoFWCollect\EcoCollectionModels\input_data_MC_6gen.xlsx";

SystemOptions                                                               =
readtable(filename,'Sheet','SystemOptions','ReadRowNames',true);
TruckInfo = readtable(filename,'Sheet','truck_info','ReadRowNames',true);
GeneratorInfo = readtable(filename,'Sheet','FWgen_rate','ReadRowNames',true);
NetworkLocs = readtable(filename,'Sheet','network_locs','ReadRowNames',true);

if SystemOptions.Distance(1) == 1
    Network = readtable(filename,'Sheet','network_dist','ReadRowNames',true);
elseif SystemOptions.Time(1) == 1
    Network = readtable(filename,'Sheet','network_time','ReadRowNames',true);
end
%Test input options for errors????

N = size(GeneratorInfo,2); %Number of generators
D = size(Network,1) - N; %Number of Depots
Dos = TruckInfo.TourLimit(1) - 1;% Number of drop offs a truck can make.
U = D * Dos; %Number of intermediate facilities to initialize
K = TruckInfo.Trucks(1); %Number of trucks to use

%import model type
if SystemOptions.ProfitModel(1) == 1
    model = 'profit';
elseif SystemOptions.CostModel(1) == 1
    model = 'cost';
end

if SystemOptions.ProfitModel(1) + SystemOptions.CostModel(1) ~= 1
    error('ERROR: Pick one model type')
end

%import transfer type
```

```
if SystemOptions.FixedTransferTime(1) == 1
    transfer = 'fixed'; %I dont think this works so not reccomended
elseif SystemOptions.FixedTransferTime(1) ~= 1
    error('ERROR: Model only works for fixed transfer time')
end

%import collection contraints
if SystemOptions.FullCollect_system(1) == 1
    collection = 'full';
elseif SystemOptions.PartCollect_system(1) == 1
    collection = 'partial';
end

if  SystemOptions.FullCollect_system(1)  +  SystemOptions.PartCollect_system(1)
~= 1
    error('ERROR: Pick one collection type')
end

if SystemOptions.Distance(1) == 1
    time_lim = TruckInfo.DistLimit(1);
elseif SystemOptions.Time(1) == 1
    time_lim = TruckInfo.TimeLimit(1);
end

%import other model factors
%import load limit. Specified in tons in sheet, converted to kg
if SystemOptions.Distance(1) == 1
    C_main = TruckInfo.OperatingCostkm(1);
elseif SystemOptions.Time(1) == 1
    C_main = TruckInfo.OperatingCosthr(1);
end

Q = TruckInfo.LoadLimit(1) * 1000;

service_fee = TruckInfo.ServiceFee(1);
TipFee = TruckInfo.TipFee(1);

qvec = GeneratorInfo{1,:}; %kgs. matrix of generated food waste at commercial
generators
qvec(N+1:N+U) = -Q; %make the maximum amount a truck can drop off at an
intermediate station equal to its capacity

distmatrix = Network{:,:}; %import network arcs

%Transfer Case
switch transfer
    case 'variable'
        transfertime  =  xlsread('input_data_test.xlsx',  'truck_info','E2');
%minutes per tonne FW
        m = (q ./ 1000) .* transfertime; %minutes. vector of pickup times at
commercial generators
    case 'fixed'
        if SystemOptions.Distance(1) == 1
            m = 0;
        elseif SystemOptions.Time(1) == 1
```

```
            transfertime = TruckInfo.FixTransferTime(1); %minutes per stop
            m = transfertime;
        end
end

%import node locations
xloc = NetworkLocs.xloc';
yloc = NetworkLocs.yloc';

%PLot Imported network
[figure1]...
    = PlotNetwork... %only works on Matlab 2017a due to intro of string matrices
    (D, N, xloc, yloc);

hgsave('Network.fig');
hold on

%%%%%%%%%%%%%%%%%%% End of user interface
if SystemOptions.Distance(1) == 1
    C_s = C_main/1000; %Convert $/km to $/m
elseif SystemOptions.Time(1) == 1
    C_s = C_main/60; %convert to $/minute;
end

revenue_fee = service_fee / 1000; %convert service fee to $/kg

%Add intermediate depot and depot prime to end of cost matrix

trav_matrix = distmatrix;

for d = 1:D
    for u = 1:Dos
        trav_matrix(D+N+((d-1)*Dos + u),:) = trav_matrix(d,:);
        trav_matrix(:,D+N+((d-1)*Dos + u)) = trav_matrix(:,d);
        xloc(D+N+((d-1)*Dos + u)) = xloc(d);
        yloc(D+N+((d-1)*Dos + u)) = yloc(d);
    end
    trav_matrix(:,D+N+U+d) = trav_matrix(:,d);
    xloc(D+N+U+d) = xloc(d);
    yloc(D+N+U+d) = yloc(d);
end
trav_matrix(:,1:D) = 0;
trav_matrix(D+N+U+1:D+N+U+D,:) = 0;
trav_matrix(D+N+1:D+N+U+D,D+N+1:D+N+U+D) = 0;
trav_matrix(1:D,D+N+1:D+N+U+D) = 0;

%initialize pickup time included matrix
% OpTime_matrix = trav_matrix;


OpTime_matrix = trav_matrix + m; %add pickup time to destination node


C_matrix = OpTime_matrix .* C_s; %Turn the time matrix into a cost matrix
```

```
%Cost matrix order should be D, N, D'

%Single decision variable counts

sDV_y = N+U; %tracks load in truck at each household
sDV_w = 1; %Volume of waste delivered to recycling facility for each truck
sDV_z = D+N+U; %keep track of waste collected from each facility individually

%for tracking and labeling only. Does not show up in the actual formulation
sDV_time = 1; %track amount of time truck operates
sDV_rev = 1; %track revenue for individual trucks
sDV_cost = 1; %track cost for individual trucks

%++++++++++++++++++++
%For matlab 2016a when creating output table
sz = D*2+N;
I = cell(sz);
Y_lab = cell(sDV_y,1);
Z_lab = cell(sDV_z, 1);
W_lab = cell(sDV_w,1);
T_lab = cell(sDV_time,1);
Rev_lab = cell(sDV_rev,1);
Cost_lab = cell(sDV_cost,1);
%++++++++++++++++++++

%total decision veriable counts
DV_y = sDV_y*K; %Pickup decision of truck at each generator or
% intermediate facility
DV_z = sDV_z*K; %Tracks load of truck just before leaving each generator
DV_w = sDV_w*K; %Volume of waste delivered to recycling facility for each truck

%flag matrix to remove impossible solutions
flag_mat = ~eye(size(trav_matrix)); %no solutions along diagonal
flag_mat(:,1:D) = 0; %No places go back to depot
flag_mat(D+N+U+1:D+N+U+D,:) = 0; %Depot prime does not go anywhere
flag_mat(1:D,D+N+1:D+N+U+D) = 0; %Depot cannot go to depot prime
flag_mat(D+N+1:D+N+U+D,D+N+1:D+N+U+D) = 0; %truck cannot go from intermediate
depot to another depot

flag_mat_tr = flag_mat';

%create costing for objectives
Cx_mat_tr = C_matrix';
sc_x = Cx_mat_tr(flag_mat_tr);
sDV_x = size(sc_x,1); %create single number of x decision variables before
vehicles are accounted for
c_x = sc_x;

%Decision Veriable count part 2
DV_x = sDV_x*K; %total possible arcs
% DV_y = sDV_y*K; %total possible arcs

%append cost matrix to itself for each truck
if K > 1
```

```
    for k = 2:K
        c_x = [c_x;sc_x];
%         c_y = [c_y;sc_y];
    end
end

%objectives given to z to keep load as small as possible. tends to balloon
%past where it needs to because there is no upper constraint (impossible to
%implement?)

switch model
    case 'cost'
        obj_type = 'Co';
        %minimize cost
        c_y = zeros(DV_y,1);

    case 'profit'
        obj_type = 'Pr';
        %maximize profit
        c_y = ones(DV_y,1) .* -revenue_fee;

end

%Z variable not included in formulation but given a very small coefficient
%to keep variable outputs in line...
c_z = zeros(DV_z, 1);
c_w = zeros(DV_w,1);
%Change out objectives at intermediates
c_y(N+1:N+U) = 0;

if U > 0
    c_z(D+N+1:D+N+U) = -.000000001;
end

% c_time = zeros(DV_time,1);

obj = [c_x;c_y;c_z;c_w];

DV_tot = DV_x+DV_y+DV_z+DV_w; %total possible decision variables

%CPLEX settings
% ctype = [];
ctype(1:DV_x) = {'B'};
ctype(DV_x+1:DV_tot) = {'C'};
ctype = char(ctype)';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Constraints

%IC3 Truck can only leave depot and generators once per truck
IC3 = [];
IC3_rhs = ones((D+N+U)*K,1); %answers

add_this_rhs = zeros(1,DV_y+DV_z+DV_w); %matrix of zeros for unused DVs
```

```
for i = 1:D+N+U
    Xtr = zeros(size(OpTime_matrix)); %matrix of zeros to fill in
    %%%%%%%
    Xtr(i,D+1:D+N+U+D) = 1; %initialize possible routes for trucks from depot
to households
    %%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions
    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

    for k = 1:K %iterate through the number of trucks
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        IC3 = [IC3;temp2]; %vertically concatenate constraints for each truck
    end
end


%IC4 Truck cannot visit intermediate drop-off unless it leaves depot for tour
IC4 = [];
IC4_rhs = zeros(K,1);
Bignum = Q; %set a large number

add_this_rhs = zeros(1,DV_y+DV_z+DV_w); %matrix of zeros for unused DVs

for k = 1:K
    Xtr = zeros(size(OpTime_matrix)); %matrix of zeros to fill in
    %%%%%%%%
    Xtr(D+N+1:D+N+U,D+1:D+N) = 1; %Initialize route from depot or household to
household
    Xtr(1:D,D+1:D+N) = -Bignum; %initialize route from household to household
    %%%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr);

    temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    temp2 = [temp1, add_this_rhs]; %add unused decision variables
    IC4 = [IC4;temp2]; %vertically concatenate constraints for each truck

end


%EC5 if truck goes to generator or intermediate facility, it must leave the
%generator or intermediate facility. solutions constrained to only feasible
%ones
EC5 = [];
EC5_rhs = zeros((N+U)*K,1);

add_this_rhs = zeros(1,DV_y+DV_z+DV_w); %matrix of zeros for unused DVs
```

```
for n = 1:N+U
    Xtr = zeros(size(OpTime_matrix)); %matrix of zeros to fill in
    %%%%%%%%
    Xtr(1:D+N+U,D+n) = 1; %Initialize route from depot or household to household
    Xtr(D+n,D+1:D+N+U+D) = -1; %initialize route from household to household
    %%%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr);

    for k = 1:K
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        EC5 = [EC5;temp2]; %vertically concatenate constraints for each truck
    end
end


%EC6 if truck leaves depot, it must return to the depot from generator

EC6 = [];
EC6_rhs = zeros(K,1);

add_this_rhs = zeros(1,DV_y+DV_z+DV_w); %matrix of zeros for unused DVs
temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs

for d = 1:D
    Xtr = zeros(size(OpTime_matrix)); %matrix of zeros to fill in
    %%%%%%%
    Xtr(d,D+1:D+N) = 1; %initialize possible routes for trucks from depot to
households
    Xtr(:,D+N+U+d) = -1; %initialize possible routes from households to depotpr
    %%%%%%%
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr); %index against flag matrix to remove impossible
solutions

    for k = 1:K %iterate through the number of trucks
        temp1 = zeros(1,DV_x); %create temp matrix of zeros the size of x DVs
        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp2 = [];
        temp2 = [temp1, add_this_rhs]; %add unused decision variables
        EC6 = [EC6;temp2]; %vertically concatenate constraints for each truck
    end
end


%IC7 can only pick something up from generator if it goes to generator.
Constrained
%to maximum generation amount

IC7 = [];
IC7_rhs = zeros((N)*K,1);

add_this_rhs = zeros(1,DV_z+DV_w);
```

```
% for i = 1:sDV_z

% for i = 1:D+N+U
    for j = D+1:D+N
%         if flag_mat(i,j) == 0
%             continue
%         end
        Xtr = zeros(size(OpTime_matrix));
        Ytr = zeros(sDV_y,1);

        %%%%%%%%%%%%%%%%%%
        Xtr(:,j) = qvec(j-D)*-1;
        Ytr(j-D) = 1;
        %%%%%%%%%%%%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);

        for k = 1:K
            temp1a = zeros(1,DV_x);
            temp1b = zeros(1,DV_y);

            temp1a((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';

            temp2 = [temp1a, temp1b, add_this_rhs]; %add unused decision
variables
            IC7 = [IC7;temp2]; %vertically concatenate constraints for each
truck
        end
    end
% end

%IC8 Inverse of IC7 to account for drop-offs at intermediate facilities

IC8 = [];
IC8_rhs = zeros(U*K,1);

add_this_rhs = zeros(1,DV_z+DV_w);

% for i = 1:sDV_z

% for i = D+N+1:D+N+U
    for j = D+N+1:D+N+U
        Xtr = zeros(size(OpTime_matrix));
        Ytr = zeros(sDV_y,1);

        %%%%%%%%%%%%%%%%%%
        Xtr(:,j) = qvec(j-D);
        Ytr(j-D) = -1;
        %%%%%%%%%%%%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);
```

```
        for k = 1:K
            temp1a = zeros(1,DV_x);
            temp1b = zeros(1,DV_y);

            temp1a((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';

            temp2  =  [temp1a,  temp1b,  add_this_rhs];  %add  unused  decision
variables
            IC8  =  [IC8;temp2];  %vertically  concatenate  constraints  for  each
truck
        end
    end
% end



%IC9 Total load collected at generators across links and truck cannot exceed
generator generation rate


IC9 = [];
IC9_rhs = ones(N,1);

for q = 1:N
    IC9_rhs(q,1) = qvec(q);
end

add_this_lhs = zeros(1,DV_x);
add_this_rhs = zeros(1,DV_z+DV_w);

for j = 1:N
    Ytr = zeros(sDV_y,1);
    %%%%%%%%%%%%%%%
    Ytr(j,:) = 1;
    %%%%%%%%

    temp1  = [];
    for k = 1:K
       temp1  = [temp1,Ytr'];
    end
    IC9 = [IC9;add_this_lhs, temp1, add_this_rhs];
end


%IC10 in ub and lb

%IC11 load and tour continuity constraint. make sure z and y function
%together
IC11 = [];
IC11_rhs = ones(sum(sum(flag_mat(1:D+N+U,D+1:D+N+U) == 1))*K,1)*Bignum;

add_this_rhs = zeros(1,DV_w); %matrix of zeros for unused DVs
```

```
for i = 1:D+N+U
    for j = D:D+N+U
        if flag_mat(i,j) == 0
            continue
        end
        Xtr = zeros(size(OpTime_matrix)); %matrix of zeros to fill in
        Ytr = zeros(sDV_y,1);
        Ztr = zeros(sDV_z,1);

        %%%%%%%%%%%%%
        Xtr(i,j) = Bignum;
        Ytr(j-D) = 1;
        Ztr(i) = 1;
        Ztr(j) = -1;
        %%%%%%%%%%%%%
        Xtr = Xtr';
        Xtr = Xtr(flag_mat_tr);

        for k = 1:K
            temp1 = zeros(1,DV_x);
            temp1b = zeros(1,DV_y);
            temp1c = zeros(1,DV_z); %create temp matrix of zeros the size of y
DVs

            temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
            temp1b((k-1)*sDV_y+1:k*sDV_y) = Ytr';
            temp1c((k-1)*sDV_z+1:k*sDV_z) = Ztr';

            temp2 = [temp1, temp1b, temp1c, add_this_rhs]; %add unused decision
variables
            IC11 = [IC11;temp2]; %vertically concatenate constraints for each
truck
        end
    end
end

%EC12 in upper and lower bounds

%IC13 total load leaving any location can only exist if truck is leaving
%it. otherwise it is 0

IC13 = [];
IC13_rhs = zeros((D+N)*K,1);

add_this_lhs = zeros(1,DV_y);
add_this_rhs = zeros(1,DV_w); %matrix of zeros for unused DVs

for j = D:D+N
    Xtr = zeros(size(OpTime_matrix)); %matrix of zeros to fill in
    Ztr = zeros(sDV_z,1); %matrix of zeros to fill in
    %%%%%%%%
    Xtr(1:D+N+U,j) = -Q; %need to figure out how to add proper q for speicifc
houeshold if not homogeneous. will be case in commercial example.
    Ztr(j) = 1;
    %%%%%%%%
```

```
    Xtr = Xtr';
    Xtr = Xtr(flag_mat_tr);
    for k = 1:K
        temp1 = zeros(1,DV_x);
        temp1b = zeros(1,DV_z); %create temp matrix of zeros the size of y DVs

        temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
        temp1b((k-1)*sDV_z+1:k*sDV_z) = Ztr';

        temp2 = [temp1, add_this_lhs, temp1b, add_this_rhs]; %add unused
decision variables
        IC13 = [IC13;temp2]; %vertically concatenate constraints for each truck
    end
end


%EC14 sum of goods picked up equals cargo dropped deposited over depots and
%intermediate facilities
EC14 = [];
EC14_rhs = zeros(K,1);

add_this_lhs = zeros(1,DV_x);
add_this_mid = zeros(1,DV_z);

Ytr = zeros(sDV_y,1);

%%%%%%%%%%
Ytr(1:N) = 1;
Wtr = ones(sDV_w,1)*-1;
%%%%%%%%%%


for k = 1:K
    temp1 = zeros(1,DV_y);
    temp1b = zeros(1,DV_w);

    temp1((k-1)*sDV_y+1:k*sDV_y) = Ytr';
    temp1b((k-1)*sDV_w+1:k*sDV_w) = Wtr';

    temp2 = [add_this_lhs, temp1, add_this_mid, temp1b];
    EC14 = [EC14;temp2];
end

%IC15 total load delivered cannot exeed the capacity of truck multiplied by
%the number of times it leaves a depot or dropoff facility

IC15 = [];
IC15_rhs = zeros(K,1);

add_this_mid = zeros(1,DV_y+DV_z);

Xtr = zeros(size(OpTime_matrix));

%%%%%%%%%%%%%
```

```
Xtr([1:D,D+N+1:D+N+U],:) = -Q;
Wtr = ones(sDV_w,1);
%%%%%%%%%%%%%%

Xtr = Xtr';
Xtr = Xtr(flag_mat_tr);


for k = 1:K
    temp1 = zeros(1,DV_x);
    temp1b = zeros(1,DV_w);

    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';
    temp1b((k-1)*sDV_w+1:k*sDV_w) = Wtr';

    temp2 = [temp1, add_this_mid, temp1b];
    IC15 = [IC15;temp2];
end

%IC16 Truck cannot travel more than time constraint allows

IC16 = [];
IC16_rhs = ones(K,1) * time_lim;

add_this_rhs = zeros(1,DV_y+DV_z+DV_w);

Xtr = OpTime_matrix;
Xtr = Xtr';
Xtr = Xtr(flag_mat_tr);

for k = 1:K
    temp1 = zeros(1,DV_x);

    temp1((k-1)*sDV_x+1:k*sDV_x) = Xtr';

    temp2 = [temp1, add_this_rhs];
    IC16 = [IC16;temp2];
end


%IEC17AB total load in trucks cannot exceed total generation in system

IEC17 = [];
IEC17_rhs = sum(qvec(1:N));

add_this_lhs = zeros(1,DV_x+DV_y+DV_z);
Wtr = ones(1,K*D);

IEC17 = [add_this_lhs, Wtr];

%End Constraints

% Put all constraints together
```

```
%constraints 8.9 and 11 are put in the upper and lower bound matricies
switch collection
    case 'full'
        coll_type = 'Al';

        A = [IC3; IC4; IC7; IC8; IC9; IC11; IC13; IC15];
        b = [IC3_rhs; IC4_rhs; IC7_rhs; IC8_rhs; IC9_rhs; IC11_rhs; IC13_rhs;
IC15_rhs];

        Aeq = [EC5; EC6; EC14; IEC17];
        beq = [EC5_rhs; EC6_rhs; EC14_rhs; IEC17_rhs];
    case 'partial'
        coll_type = 'Pa';

        A = [IC3; IC4; IC7; IC8; IC9; IC11; IC13; IC15; IEC17];
        b = [IC3_rhs; IC4_rhs; IC7_rhs; IC8_rhs; IC9_rhs; IC11_rhs; IC13_rhs;
IC15_rhs; IEC17_rhs];

        Aeq = [EC5; EC6; EC14];
        beq = [EC5_rhs; EC6_rhs; EC14_rhs];
end

% lb(1:DV_tot) = 0;
%Create inital lower and upper bounds
lb_x = []; lb_x_temp = zeros(size(OpTime_matrix));
lb_y = []; lb_y_temp = zeros(sDV_y,1);
lb_z = []; lb_z_temp = zeros(sDV_z,1);
lb_w = []; lb_w_temp = zeros(sDV_w,1);

ub_x = []; ub_x_temp = ones(size(OpTime_matrix));
ub_y = []; ub_y_temp = ones(sDV_y,1) .* Q;
ub_z = []; ub_z_temp = ones(sDV_z,1) .* Q;
ub_w = []; ub_w_temp = ones(sDV_w,1) .* sum(qvec(1:N));

%Assign special values to bounds if necessary

%IC10
lb_y_temp(N+1:N+U) = -Q;
ub_y_temp(N+1:N+U) = 0;
%EC12
ub_z_temp([1:D,D+N+1:D+N+U]) = 0;

%Reformat X decision variables into vectors

lb_x_temp = lb_x_temp';
ub_x_temp = ub_x_temp';

lb_x_temp = lb_x_temp(flag_mat_tr);
ub_x_temp = ub_x_temp(flag_mat_tr);

%Build lower bound and upper bounds
for k = 1:K
    lb_x = [lb_x,lb_x_temp']; ub_x = [ub_x,ub_x_temp'];
```

```
    lb_y = [lb_y,lb_y_temp']; ub_y = [ub_y,ub_y_temp'];
    lb_z = [lb_z,lb_z_temp']; ub_z = [ub_z,ub_z_temp'];
    lb_w = [lb_w,lb_w_temp']; ub_w = [ub_w,ub_w_temp'];
end

lb = [lb_x, lb_y, lb_z, lb_w];
ub = [ub_x, ub_y, ub_z, ub_w];

opts = cplexoptimset('Display','on');
x0 = [];

sostype=[];
sosind=[];
soswt=[];
codetime = toc;

[solution,fval,exitflag,output]                                                    =
cplexmilp(obj,A,b,Aeq,beq,sostype,sosind,soswt,lb,ub,ctype,x0,opts);

codensolvertime = toc;

solution = round(solution);
fval
exitflag
output

%%%%%%Output code
% Index the results

%crate solution label key
Xlabels = []; Ylabels = []; Zlabels = []; Wlabels = []; Tlabels = [];
Xlab_temp = strings(size(OpTime_matrix));
Ylab_temp = strings(sDV_y,1);
Zlab_temp = strings(sDV_z,1);
Wlab_temp = strings(sDV_w,1);
%Count for each truck: Operating Time, Cost, Revenue
Tlab_temp = strings(3,1);

for k = 1:K

    Xlab_I = Xlab_temp;
    Xlab_J = Xlab_temp;
    Ylab_I = Ylab_temp;
    Zlab_I = Zlab_temp;
    Wlab_I = Wlab_temp;
    Tlab_I = Tlab_temp;



    if SystemOptions.Distance(1) == 1
        Tlab_I(1) = "K" + string(k) + ": Distance (km)";

    elseif SystemOptions.Time(1) == 1
        Tlab_I(1) = "K" + string(k) + ": Time (hr)";
```

```
    end

    Tlab_I(2) = "K" + string(k) + ": Revenue ($)";
    Tlab_I(3) = "K" + string(k) + ": Cost ($)";

    for d = 1:D
        Xlab_I(d,:) = "K" + string(k) + ": D" + string(d);
        Xlab_J(:,d) = "-D" + string(d);
        Zlab_I(d) = "K" + string(k) + ": (Z)D" + string(d);

        %depot prime
        Xlab_I(D+N+U+d,:) = "K" + string(k) + ": D" + string(d) + "'";
        Xlab_J(:,D+N+U+d) = "-D" + string(d) + "'";
        Wlab_I(d) = "K" + string(k) + ": (W)D" + string(d) + "'";

        for u = 1:Dos
            Xlab_I(D+N+((d-1)*Dos + u),:) = "K" + string(k) + ": U" + string(d)
+ "." + string(u);
            Xlab_J(:,D+N+((d-1)*Dos + u)) = "-U" + string(d) + "." + string(u);
            Ylab_I(N+((d-1)*Dos + u)) = "K" + string(k) + ": (Y)U" + string(d)
+ "." + string(u);
            Zlab_I(D+N+((d-1)*Dos + u),:) = "K" + string(k) + ": (Z)U" +
string(d) + "." + string(u);
        end
    end
    for n = 1:N
        Xlab_I(D+n,:) = "K" + string(k) + ": N" + string(n);
        Xlab_J(:,D+n) = "-N" + string(n);
        Ylab_I(n) = "K" + string(k) + ": (Y)N" + string(n);
        Zlab_I(D+n) = "K" + string(k) + ": (Z)N" + string(n);
    end


    Xlab_temp2 = Xlab_I + Xlab_J;
    Xlab_temp3 = Xlab_temp2';
    Xlab_temp3 = Xlab_temp3(flag_mat_tr);

    Xlabels = [Xlabels;Xlab_temp3];
    Ylabels = [Ylabels;Ylab_I];
    Zlabels = [Zlabels;Zlab_I];
    Wlabels = [Wlabels;Wlab_I];
    Tlabels = [Tlabels;Tlab_I];

end

Tvalues = [];
totalrevenue = 0;
totalcost = 0;

OpTime_mat_tr = OpTime_matrix';
OpTime_vec = OpTime_mat_tr(flag_mat_tr);
Cx_vec = Cx_mat_tr(flag_mat_tr);
```

```
%graph network solution
colormat = ['k','b'];

from_i = zeros(size(OpTime_matrix));
to_j = from_i;

for i = 1:D+N+U+D
    from_i(i,:) = i;
end

for j = 1:D+N+U+D
    to_j(:,j) = j;
end

from_i = from_i';
to_j = to_j';

from_i_vec = from_i(flag_mat_tr);
to_j_vec = to_j(flag_mat_tr);

x_sol = solution(1:DV_x);
x_sol_idx = logical(x_sol);
% x_sol_short_temp = x_sol(x_sol_idx);
% Xlabels_short_temp = Xlabels(x_sol_idx);

Xlabels_short = strings;
x_sol_short = [];

%put together network graphic solution and tally costs/revenues
for k = 1:K
    %create indices for solution values
    idx_xlow = (k-1)*sDV_x + 1;
    idx_xhigh = k*sDV_x;

    truck_travel_ij = x_sol_idx(idx_xlow:idx_xhigh);

    truck_from_i = from_i_vec(x_sol_idx(idx_xlow:idx_xhigh));
    truck_to_j = to_j_vec(x_sol_idx(idx_xlow:idx_xhigh));

    %rearrange solution to travel order
    row = 1;
%     idx = find(truck_travel_ij,1,'first');
%     i = from_i_vec(idx);
    Xlabels_short_temp = strings;
    x_sol_short_temp = [];
    %write route labels in answer order
    for row = 1:size(truck_from_i,1)
        i = truck_from_i(row);
        j = truck_to_j(row);
        Xlabels_short_temp(row) = Xlab_temp2(i,j);
        x_sol_short_temp(row) = 1;
    end
```

```
    %Follow path loop in labeling
%     while i < D+N+U+1
%         j = to_j_vec(idx);
%         Xlabels_short_temp(row) = Xlab_temp2(i,j);
%         x_sol_short_temp(row) = x_sol(idx);
%
%         row = row + 1;
%         i = j;
%         is = find(from_i_vec == i);
%         js = find(truck_travel_ij == 1);
%         idx = is(ismember(is,js));
%     end


    startpoints = [xloc(truck_from_i)',yloc(truck_from_i)'];
    endpoints = [xloc(truck_to_j)',yloc(truck_to_j)'];
    s = colormat(k);
    width = 1;
    height = 1;
    if size(startpoints,1) >=1
        hn = arrow3(startpoints,endpoints,s,width,height);
    end

    %conditional labeling
    if SystemOptions.Distance(1) == 1
        TruckOpTime = sum(OpTime_vec(truck_travel_ij))/1000;

    elseif SystemOptions.Time(1) == 1
        TruckOpTime = sum(OpTime_vec(truck_travel_ij))/60;

    end

    TruckRev = solution(DV_x+DV_y+DV_z+k)* revenue_fee;
    TruckOpCost = sum(Cx_vec(truck_travel_ij));

    Tvalues = [Tvalues;TruckOpTime;TruckRev;TruckOpCost];

    totalrevenue = totalrevenue + TruckRev;
    totalcost = totalcost + TruckOpCost;

    if k == 1
        Xlabels_short = Xlabels_short_temp';
    elseif k > 1
        Xlabels_short = [Xlabels_short;Xlabels_short_temp'];
    end

    x_sol_short = [x_sol_short;x_sol_short_temp'];
end


legend('Depot','Generator','Path','Location','EastOutside')
```

```
figurename                                                                    =
'LPNetRoute_'+string(obj_type)+string(coll_type)+'_'+string(C_main)+'c'+strin
g(service_fee)+'r_'+string(N)+'gens';

hgsave(figurename+'.fig');
saveas(gcf,figurename+'.png')
% savefig(figure1,'NetworkSolution.png');

profitval = totalrevenue - totalcost;

SolLabels = [Xlabels_short;Ylabels;Zlabels;Wlabels];
if size(x_sol_short,1) < 1
    SolShort = [NaN(1);solution(DV_x+1:DV_tot)];
elseif size(x_sol_short,1) >= 1
    SolShort = [x_sol_short;solution(DV_x+1:DV_tot)];
end

%conditional labeling
if SystemOptions.Distance(1) == 1
    OC = "Operating Cost ($/km)";
    rt = "Distance Limit (m)";

elseif SystemOptions.Time(1) == 1
    OC = "Operating Cost ($/hr)";
    rt = "Time Limit (min)";

end

ModelLabels =   ["PrepTime(s)";
                "SolverTime(s)";
                "CodeTime (s)";
                "Model Type: " + model;
                "Collection Constraint: " + collection;
                "Depots";
                "Intermediate Drop-offs";
                "Generators";
                "Collection Trucks";
                "Truck Capacity (t)";
                "Total Generation (t)";
                "Gen. Collect Time (min)";
                "Service Fee ($/t)";
                OC;
                rt;
                SolLabels;
                Tlabels;
                "Total Profit ($)";
                "Objective Value"];

ModelValues =   [codetime;
                codensolvertime-codetime;
                codensolvertime;
                NaN(1);
                NaN(1);
                D;
                Dos;
```

```
                N;
                K;
                Q/1000;
                sum(qvec(1:N))/1000;
                m;
                revenue_fee*1000;
                C_main;
                time_lim;
                SolShort;
                Tvalues;
                profitval;
                fval];

solutiontable = table(ModelLabels,ModelValues);

tablename                                                                    =
'LPNetRoute_'+string(obj_type)+string(coll_type)+'_'+string(C_main)+'c'+strin
g(service_fee)+'r_'+string(N)+'gens';
writetable(solutiontable, tablename+'.xlsx');
```

Sub code plot network

```
%Plot Figure of points

function [figure1] = PlotNetwork(D, N, xloc, yloc)

    D_lab = [];%Labels for depot
    for d = 1:D
        ds = string(d);
        D_lab = [D_lab, 't' + ds];
    end

    N_lab = []; %labels for generators
    for n = 1:N
        ns = string(n);
        N_lab = [N_lab, 'g' + ns];
    end

    nodeIDs = 1:D+N;
    labels = [D_lab,N_lab];

    figure1                                                                    =
plot(xloc(1:D),yloc(1:D),'b^','MarkerSize',6,'MarkerFaceColor','b');
    hold on

plot(xloc(D+1:D+N),yloc(D+1:D+N),'ro','MarkerSize',6,'MarkerFaceColor','r');
    % Label points
    text(xloc+.002,yloc+.006,labels);

    minx = min(xloc);
    miny = min(yloc);
    maxx = max(xloc);
    maxy = max(yloc);

    xlim([minx-.02 maxx+.02]);ylim([miny-.02 maxy+.02])

    %Make Proxy arrows for legend
    legendarrow = quiver(10,10,1,0,'k');
    xlabel('Longitude')
    ylabel('Latitude')

legend('Depot','Generator','Location','EastOutside')
```

FOR CHAPTER 5

Manure Allocation Script

```
# -*- coding: utf-8 -*-
# ---------------------------------------------------------------------------
# ManureAllocation.py
# Created on: 2019-06-22 10:08:15.00000
#   (generated by ArcGIS/ModelBuilder)
# Description:
# ---------------------------------------------------------------------------

# Import arcpy module
import arcpy
from arcpy import env
from arcpy.sa import *


# Load required toolboxes


#geodatabases
GDB = "SourceSinkAnalysisv2.gdb/"
sGDB = "SSAScratch.gdb/"
# Local variable names:
WNY_CAFO = GDB + "ORLdata/WNY_CAFO"
CropField = Raster(GDB + "Crop_Pdemand_AgDistricts")
WNYboundary = GDB + "BackgroundInfo/WNY_Boundary"

# Set Geoprocessing environments
#env.workspace = GDB
arcpy.env.scratchWorkspace = "D:/INFEWS/SourceSinkAnalysis/SSAScratch.gdb"
arcpy.env.snapRaster = GDB + "Crop_Pdemand_AgDistricts"
```

```
arcpy.env.extent = "105585.22883297 4650944.03513106 373305.22883297 4808384.03513106"
arcpy.env.overwriteOutput = True

for i_py in range(214):
    i = i_py + 1
    print("CAFO " + str(i))
    select = "OBJECTID = " + str(i)
#SelectFeature
    CAFO_i = arcpy.Select_analysis(in_features=WNY_CAFO,out_feature_class=sGDB + "CAFO_select",where_clause=select)
    cursor = arcpy.SearchCursor(CAFO_i)
    #Extract capacity from CAFO data
    for row in cursor:
        capacity = row.getValue("Pavail_kgpy")

    radius_base = 90
    radius_iter = 1
    Pdemand = 0
    #make areas of interest larger until Pdemand is satisfied by manure P
    while Pdemand < capacity:
        radius = radius_base * radius_iter
        print("radius " + str(radius))
        circle = arcpy.Buffer_analysis(in_features=CAFO_i,buffer_distance_or_field=radius,out_feature_class=sGDB + "AOI",dissolve_option="NONE")
        ZoneTable = ZonalStatisticsAsTable(circle,"OBJECTID",CropField,sGDB + "ZoneTable","DATA","SUM")
        cursor2 = arcpy.SearchCursor(ZoneTable)
        for row in cursor2:
            Pdemand = row.getValue("SUM")
        radius_iter += 1
```

```
    #cut out area from boundary to extract crop information
    WNYbound_in = arcpy.Copy_management(in_data=WNYboundary,out_data=sGDB + "WNYbound_in")
    WNYboundary = arcpy.Erase_analysis(WNYbound_in,circle,sGDB + "WNYBoundary_update")
    #extract only crop fields from outside "used" fields
    CropField.save(sGDB + "CropFields_in")
    CropField_in = sGDB + "CropFields_in"
    CropField = ExtractByMask(CropField_in,WNYboundary)
    CropField.save(sGDB + "CropFields_update")
```