```
print("STARTING FIRE ENGINE COMPANY LOCATION PROGRAM..")
import math
print "INSIDE GUROBI OPTIMIZATION PROG..."
import random
global MINS
global SPEED
global DIST_THRESHOLD
global BETA
global MIN_TIME
global MAX_TIME
global MIN_TIME2
global MAX_TIME2
global SPEED2
global DETOUR_INDEX
global tot_locs
global CRIT_PROB
global CRIT_DIST
global ENSEMBLE_CODE
global GUROBI_OPT
global PUMPER_COST
global LADDER_COST
global PUMPER_CREW_COST
global LADDER_CREW_COST
global GEN_POP
global CRIT_PROB2
global NUM_ECS
global NUM_ENG_TYPES
global NUM_CROSSOVERS
global NUM_MUTATIONS
global MUTATION_PROB
global GENE_MUTATION_PROB
global COST_INDEX_LT
global PROP_INFEAS_LT
global DO_GA
global EQUITY_SPREAD
global RADIUS
global DO_PERTURB

random.seed(999)
SCALE_FIRES = 100
EPSILON = 0.01
mesh_width = 6.0
NUM_SCENARIOS = 0
DETOUR_INDEX = 1.42
CRIT_DIST = 2.0
Perc_List_Increase = 1.0
GUROBI_OPT = 1
PUMPER_COST = 350000
PUMPER_CREW_COST = 150000
LADDER_COST = 900000
LADDER_CREW_COST = 200000
MIN_TIME2 = 8
MAX_TIME2 = 18
GEN_POP = 100
# CRIT_PROB2 is the probability with which a LADDER truck is placed at each engine company location
CRIT_PROB2 = 0.5
NUM_ENG_TYPES = 2
NUM_CROSSOVERS = 50
```

```
NUM_MUTATIONS = 50
#mutation_prob is the prob of selecting a gene for mutation :he whole gene is selected with this prob
MUTATION_PROB = 0.1
#gene_mutation_prob is the probability of changing the configuration for a specific engine company
within a gene
GENE_MUTATION_PROB = 0.1
COST_INDEX_LT = 200
# NOTE THE PROPORTION OF INFEASIBLE FIRES HAS TO BE EXPRESSED AS A PERCENTAGE, 10% NOT 0.1
PROP_INFEAS_LT = 10
EQUITY_SPREAD = 5.0
DO_GA = 0
RADIUS = 1.0
DO_PERTURB = 0


# If ensemble code is 1, aggregate bases on fire scenarios
# If ensemble code is 2, aggregate based on facility locations
ENSEMBLE_CODE = 0

def getKey(item):
	return(item[5])

def Sort_Master_Train_Loc_List(Master_Train_Loc_List):
	MTL_sorted = sorted(Master_Train_Loc_List, key=getKey, reverse = True)
	return(MTL_sorted)

def get_node_perc_covered(G, yet_to_cover):
	totnodes = 0.0
	covered_nodes = 0.0
```

```
        for n in G.nodes():
                totnodes = totnodes + 1
                if(yet_to_cover[n] == 0):
                        covered_nodes = covered_nodes + 1

        perc_nodes_covered = float(covered_nodes*100)/float(totnodes)
        return(perc_nodes_covered)

def get_pop_perc_covered(G, yet_to_cover):
        totpop = 0.0
        covered_pop = 0.0
        for n in G.nodes():
                totpop = totpop + G.node[n]['node_pop']
                if(yet_to_cover[n] == 0):
                        covered_pop = covered_pop + G.node[n]['node_pop']

        perc_pop_covered = float(covered_pop*100)/float(totpop)
        return(perc_pop_covered)

def get_num_engines(G):
        ncount = 0
        for n in G.nodes():
                if(G.node[n]['EngineLoc'] == 1):
                        ncount = ncount + 1

        return(ncount)

def Get_Equity_Stats(G, cover):
        tot_pop = 0
```

```
    tot_nodes = 0

    for n in G.nodes():
        tot_pop = tot_pop + G.node[n]['node_pop']
        tot_nodes = tot_nodes + 1

    print("tot_pop = ", tot_pop)
    print("tot_nodes = ", tot_nodes)

    for n in G.nodes():
        if(G.node[n]['EngineLoc'] == 1):
            covered_pop = 0
            covered_nodes = 0
            for n2 in G.nodes():
                if(cover[n][n2] == 1):
                    covered_pop = covered_pop + G.node[n2]['node_pop']
                    covered_nodes = covered_nodes + 1

            G.node[n]['covered_pop_perc'] = (float(covered_pop)/float(tot_pop))*100.0
            G.node[n]['covered_nodes_perc'] =
(float(covered_nodes)/float(tot_nodes))*100.0

    max_pop_perc = 0
    min_pop_perc = 0
    iter = 0
    for n in G.nodes():
        if(G.node[n]['EngineLoc'] == 1):
            iter = iter + 1
            if(iter == 1):
                max_pop_perc = G.node[n]['covered_pop_perc']
                min_pop_perc = G.node[n]['covered_pop_perc']
            elif(iter > 1):
                if(G.node[n]['covered_pop_perc'] > max_pop_perc):
                    max_pop_perc = G.node[n]['covered_pop_perc']
                if(G.node[n]['covered_pop_perc'] < min_pop_perc):
                    min_pop_perc = G.node[n]['covered_pop_perc']

    print("ENGINE COMPANY EQUITY REPORT FOR POPULATION COVERAGE MODEL...")
    for n in G.nodes():
        if(G.node[n]['EngineLoc'] == 1):
            print("Company ID = ", int(n), G.node[n]['Xcor'], G.node[n]['Ycor'],"Covered POP % =", G.node[n]['covered_pop_perc'])

    print("MAX POPULATION % COVERAGE for any Engine Company = ", max_pop_perc)
    print("MIN POPULATION % COVERAGE for any Engine Company = ", min_pop_perc)



def get_next_location(G, SP_Lengths, cover, yet_to_cover, HEUR):
    pop_wt_array = [0 for i in range(nnodes+1)]
    node_wt_array = [0 for i in range(nnodes+1)]
    composite_wt_array = [0 for i in range(nnodes+1)]

    for n1 in G.nodes():
        pop_wt_array[n1] = 0
        node_wt_array[n1] = 0
        composite_wt_array[n1] = 0
        for n2 in G.nodes():
            if((cover[n1][n2] == 1) and (yet_to_cover[n2] == 1)):
                pop_wt_array[n1] = pop_wt_array[n1] + G.node[n2]['node_pop']
                node_wt_array[n1] = node_wt_array[n1] + 1

        composite_wt_array[n1] = (node_wt_array[n1])*(pop_wt_array[n1])
    best_node = -1
    best_pop_wt = 0
    best_node_wt = 0
    best_composite_wt = 0
    for n in G.nodes():
        if (G.node[n]['EngineLoc'] == -1):
            if((HEUR == 1) and (pop_wt_array[n] > best_pop_wt)):
                best_pop_wt = pop_wt_array[n]
                best_node = n
            elif((HEUR == 2) and (node_wt_array[n] > best_node_wt)):
                best_node_wt = node_wt_array[n]
                best_node = n
            elif((HEUR == 3) and (composite_wt_array[n] > best_composite_wt)):
                best_composite_wt = composite_wt_array[n]
                best_node = n
    if(best_node == -1):
        for n in G.nodes():
            if ((best_pop_wt == 0)and (yet_to_cover[n] == 1)):
                print('SCENARIO WHERE 0 POP NODE IS BEST POP..')
                best_node = n

    return(best_node)

def get_num_to_cover(G, yet_to_cover):
    num_to_cover = 0
    for n in G.nodes():
        index = int(n)
        if(yet_to_cover[index] == 1):
            num_to_cover = num_to_cover + 1

    return(num_to_cover)

def get_fro_node(G, e):
    for n in G.nodes():
        if (int(n) == int(e[0])):
            return(n)
    return(-1)

def get_to_node(G, e):
    for n in G.nodes():
        if (int(n) == int(e[1])):
            return(n)
    return(-1)

def get_poisson_prob(MU,i):
    prob = (math.exp(-MU))
    for j in range(i):
        try:
            prob = prob*(MU/(j+1))
        except:
            print ("ERROR IN GET_POISSON_PROB..")
            exit()
    return(prob)
```

```
def sim_num_fires(G, n):
    MU = G.node[n]['mu_fires']
    max_fires = G.node[n]['max_fires']
    rand_no = random.random()
    cum_prob = 0
    for i in range(max_fires + 1):
        cur_prob = get_poisson_prob(MU,i)
        old_cum_prob = cum_prob
        cum_prob = cur_prob + cum_prob
        if((old_cum_prob < rand_no) and (rand_no <= cum_prob)):
            return(i)

    return(max_fires)




def get_max_num_fires(G, n, EPSILON):
    for n1 in G.nodes():
        if (int(n1) == int(n)):
            MU = G.node[n]['mu_fires']
            if MU < EPSILON:
                return 0

            if(MU > 1):
                i = int(MU)
            elif(MU < 1):
                i = 1
            prob = 1
            while prob > (EPSILON):
                i = i + 1
                prob = get_poisson_prob(MU,i)


            return(i)


def get_fire_radius(G,n):
    max_radius = 0
    for e in G.edges():
        n1 = get_fro_node(G, e)
        n2 = get_to_node(G, e)
        if(n1 == n):
            if(G.edge[n1][n2]['weight']> max_radius):
                max_radius = G.edge[n1][n2]['weight']

    max_radius = max_radius/2.0
    return(max_radius)

def get_total_locs(G):
        tot_locs = 0
        for n in G.nodes():
            tot_locs = tot_locs + G.node[n]['NECLocs']

        return(tot_locs)
```

```
def get_cover_prob(travel_time):
        if(travel_time <= MIN_TIME):
            prob = 1.0
        elif(travel_time <= MAX_TIME):
            x = float(travel_time - MIN_TIME)/float(MAX_TIME - MIN_TIME)
            prob = 1.0 - x
        elif(travel_time > MAX_TIME):
            prob = 0.0

        return(prob)

def get_travel_dist(curfx,curfy, curx1, cury1):
        travel_dist = (curfx - curx1)*(curfx - curx1)
        travel_dist = travel_dist + (curfy - cury1)*(curfy - cury1)
        travel_dist = pow((travel_dist),0.5)
        travel_dist = DETOUR_INDEX*travel_dist
        return(travel_dist)

def get_num_fires_yet_to_cover(sc_yet_to_cover):
    tot = 0
    for i in range(len(sc_yet_to_cover)):
        tot = tot + sc_yet_to_cover[i]

    return(tot)

def set_flag(tot, sc_yet_to_cover):
    max_scenarios = len(sc_yet_to_cover) - 1
    thresh_num = (1.0 - BETA)*max_scenarios
    # print("tot uncovered scenarios = ", tot)
    if(tot <= thresh_num):
        return(1)
    else:
        return(0)

def Update_SC_Yet_To_Cover(G, f_s, n_locs, sc_cover, sc_yet_to_cover):
    nlocs_local = 0
    for n in G.nodes():
        ECList = G.node[n]['ECList']
        for  k in range(len(ECList)):
            curLocList = ECList[k]
            nlocs_local = nlocs_local + 1
            if(nlocs_local == n_locs):
                for j in range(len(f_s)):
                    if(sc_cover[n_locs][j+1] == 1):
                        sc_yet_to_cover[j+1] = 0
                return(1)
    return(-1)

def get_facility(G, n_locs):
    nlocs_local = 0
    for n in G.nodes():
        ECList = G.node[n]['ECList']
        for  k in range(len(ECList)):
            curLocList = ECList[k]
            nlocs_local = nlocs_local + 1
            if(nlocs_local == n_locs):
                curNode = int(n)
                curf_ID = curLocList[0]
```

```
                        curfx = curLocList[1]

                        curfy = curLocList[2]

                        curLoc = [n_locs, curNode, curf_ID, curfx, curfy]

                        return(curLoc)


            print("ERROR IN get_facility()..No Match found..")

            curLoc = [-1,-1,-1,-1,-1]

            return(curLoc)




def get_index_for_next_location(G, f_s, sc_cover, sc_yet_to_cover):

            n_locs = 0

            max_tot_cov = 0

            max_index = -1


            for n in G.nodes():

                        # print("At node ", int(n))

                        ECList = G.node[n]['ECList']

                        NumEC = G.node[n]['NECLocs']


                        for  k in range(len(ECList)):

                                    curLocList = ECList[k]

                                    n_locs = n_locs + 1

                                    curf_ID = curLocList[0]

                                    curfx = curLocList[1]

                                    curfy = curLocList[2]

                                    tot_cov = 0


                                    for j in range(len(f_s)):
```

```
                if((sc_cover[n_locs][j+1] == 1)and(sc_yet_to_cover[j+1] == 1)):
                    tot_cov = tot_cov + 1

            if(tot_cov > max_tot_cov):
                max_tot_cov = tot_cov
                max_index = n_locs

    return(max_index)

def Make_Master_Train_Loc_List(Train_Loc_List, Train_List, ENSEMBLE_CODE):
    curLocList = []
    for i in range(len(Train_Loc_List)):
        Loc_List = Train_Loc_List[i]
        for j in range(len(Loc_List)):
            curLoc = Loc_List[j]
            curID = curLoc[0]
            curNode = curLoc[1]
            cur_w_node = curLoc[2]
            curx = curLoc[3]
            cury = curLoc[4]
            cur_weight = 0
            List_Element = [curID, curNode, cur_w_node, curx, cury, cur_weight]
            curLocList.append(List_Element)

    if(ENSEMBLE_CODE == 1):
        for i in range(len(curLocList)):
            curLoc1 = curLocList[i]
            curx1 = curLoc1[3]
            cury1 = curLoc1[4]
            cur_weight = 0
            for j in range(len(Train_List)):
                f_s = Train_List[j]
                for k in range(len(f_s)):
                    curLoc2 = f_s[k]
                    curx2 = curLoc2[3]
                    cury2 = curLoc2[4]
                    dist = 0
                    dist = (curx1-curx2)*(curx1-curx2)
                    dist = dist + (cury1-cury2)*(cury1-cury2)
                    dist = math.sqrt(dist)
                    if(dist <= CRIT_DIST):
                        cur_weight = cur_weight + 1
                    else:
                        cur_weight = cur_weight + math.exp(-dist)

            curLoc1[5] = cur_weight
    elif(ENSEMBLE_CODE == 2):
        for i in range(len(curLocList)):
            curLoc1 = curLocList[i]
            curx1 = curLoc1[3]
            cury1 = curLoc1[4]
            cur_weight = 0
            for j in range(len(curLocList)):
                curLoc2 = curLocList[j]
                curx2 = curLoc2[3]
                cury2 = curLoc2[4]
                dist = 0
                dist = (curx1-curx2)*(curx1-curx2)
```

```
                dist = dist + (cury1-cury2)*(cury1-cury2)
                dist = math.sqrt(dist)
                if(dist <= CRIT_DIST):
                    cur_weight = cur_weight + 1
                else:
                    cur_weight = cur_weight + math.exp(-dist)

            curLoc1[5] = cur_weight



    return(curLocList)


def Make_Full_Train_List(Train_List):
    f_s_list = []
    for i in range(len(Train_List)):
        f_s = Train_List[i]
        for j in range(len(f_s)):
            f_s_list.append(f_s[j])

    return(f_s_list)


def get_best_fac(MTL_List_sorted, Full_Train_List, sc_cover_code, fac_code):
    best_fac = -1
    best_sc_cov = 0

    for i in range(len(MTL_List_sorted)):
        if(fac_code[i] == -1):
            cur_fac_index  = i
            cur_fac = MTL_List_sorted[i]
            curfx = cur_fac[3]

            curfy = cur_fac[4]

            num_sc_cov = 0

            for j in range(len(Full_Train_List)):

                if(sc_cover_code[j] == 1):

                    f_s = Full_Train_List[j]

                    curx1 = f_s[3]

                    cury1 = f_s[4]

                    travel_dist = 0

                    travel_dist = get_travel_dist(curfx,curfy, curx1, cury1)

                    travel_time = (travel_dist*60)/SPEED2

                    cover_prob = get_cover_prob(travel_time)

                    if(cover_prob >= CRIT_PROB):

                        num_sc_cov = num_sc_cov + 1

                        # instead of a raw count of number of covered fires
above, can make this a weight based on ENSEMBLE algorithm.


            if(num_sc_cov > best_sc_cov):

                best_sc_cov = num_sc_cov

                best_fac = cur_fac_index


    return(best_fac)




def get_sc_perc_cov(sc_cover_code):

        num = float(len(sc_cover_code))

        tot = 0

        for i in range(len(sc_cover_code)):

            tot = tot + sc_cover_code[i]
        frac_covered = 1.0 - float(tot)/num
        return(frac_covered)

def get_pithy_train_list(MTL_List_sorted, Full_Train_List, MAX_LOCS):
    Pithy_Train_List = []
    num_in_list = 0
    sc_cover_code = [1 for i in range(len(Full_Train_List))]
    sc_perc_cov = get_sc_perc_cov(sc_cover_code)

    fac_code = [-1 for i in range(len(MTL_List_sorted))]

    while(sc_perc_cov < BETA and num_in_list < MAX_LOCS):
        best_fac = -1
        best_fac = get_best_fac(MTL_List_sorted, Full_Train_List, sc_cover_code, fac_code)
        for i in range(len(MTL_List_sorted)):
            if(best_fac == i):
                cur_fac = MTL_List_sorted[i]
                fac_code[i] = 1
                Pithy_Train_List.append(cur_fac)
                curfx = cur_fac[3]
                curfy = cur_fac[4]
                num_in_list = num_in_list + 1
                for j in range(len(Full_Train_List)):
                    f_s = Full_Train_List[j]
                    curx1 = f_s[3]
                    cury1 = f_s[4]
                    travel_dist = 0
                    travel_dist = get_travel_dist(curfx,curfy, curx1, cury1)
                    travel_time = (travel_dist*60)/SPEED2
                    cover_prob = get_cover_prob(travel_time)
                    if(cover_prob >= CRIT_PROB):
                        sc_cover_code[j] = 0



        sc_perc_cov = get_sc_perc_cov(sc_cover_code)
    return(Pithy_Train_List)


# BELOW JUST SORTS MTL_LIST.
#def get_pithy_list(MTL_List_sorted, MAX_LOCS):
#   Pithy_Train_List = []
#   num_in_list = 0
#   for i in range(len(MTL_List_sorted)):
#       curLoc = MTL_List_sorted[i]
#       num_in_list = num_in_list + 1
#       if(num_in_list <= MAX_LOCS):
#           Pithy_Train_List.append(curLoc)
#
#   return(Pithy_Train_List)


def get_average_stats_for_test_scenario(Pithy_Train_List, f_s):
    num_scen = len(f_s)
    ytc_sc = [1 for i in range(num_scen + 1)]
    ytc_sc[0] = 0
    max_response_time = 0
    min_response_time = 999999
    sum_resp_times = 0
```

```
d_stats = [0,0,0,0,0,0,0]

for i in range(len(f_s)):

        curList = f_s[i]

        curx1 = curList[3]

        cury1 = curList[4]

        max_time_for_cur_sc = 99999

        for k in range(len(Pithy_Train_List)):

                curLoc = Pithy_Train_List[k]

                curfx = curLoc[3]

                curfy = curLoc[4]

                travel_dist = 0

                travel_dist = get_travel_dist(curfx,curfy, curx1, cury1)

                travel_time = (travel_dist*60)/SPEED2

                if(travel_time < max_time_for_cur_sc):

                        max_time_for_cur_sc = travel_time

                cover_prob = get_cover_prob(travel_time)

                if(cover_prob >= CRIT_PROB):

                        ytc_sc[i+1] = 0

        if(max_time_for_cur_sc > max_response_time):

                max_response_time = max_time_for_cur_sc


        if(max_time_for_cur_sc < min_response_time):

                min_response_time = max_time_for_cur_sc


        sum_resp_times = sum_resp_times + max_time_for_cur_sc


num_to_cover = 0

for kk in range(num_scen):
    num_to_cover = num_to_cover + ytc_sc[kk]

avg_resp_time = float(sum_resp_times)/float(num_scen)
beta_est = 1 - (float(num_to_cover))/float(num_scen)
d_stats[0] = beta_est
d_stats[1] = max_response_time
d_stats[2] = avg_resp_time
d_stats[3] = min_response_time

max_sc_cover = 0
min_sc_cover = 999999
ytc_sc2 = [1 for i in range(num_scen + 1)]
ytc_sc2[0] = 0
cum_sc_cover = 0
n_engs = 0
for k in range(len(Pithy_Train_List)):
    n_engs = n_engs + 1
    curLoc = Pithy_Train_List[k]
    curfx = curLoc[3]
    curfy = curLoc[4]
    n_sc_cover = 0
    for i in range(len(f_s)):
        curList = f_s[i]
        curx1 = curList[3]
        cury1 = curList[4]
        travel_dist = 0
        travel_dist = get_travel_dist(curfx,curfy, curx1, cury1)
        travel_time = (travel_dist*60)/SPEED2
        cover_prob = get_cover_prob(travel_time)
```

```
            if(cover_prob >= CRIT_PROB):
                n_sc_cover = n_sc_cover + 1
                if(ytc_sc2[i] == 1):
                    ytc_sc2[i] = 0
                    cum_sc_cover = cum_sc_cover + 1

        cum_sc_perc = float((cum_sc_cover*100)/float(num_scen))
        print("Engine company ",n_engs ," covers ", cum_sc_perc, "of fires..")
        if(n_sc_cover > max_sc_cover):
            max_sc_cover = n_sc_cover
        if(n_sc_cover < min_sc_cover):
            min_sc_cover = n_sc_cover
    d_stats[4] = float(max_sc_cover*100)/float(num_scen)
    d_stats[5] = float(min_sc_cover*100)/float(num_scen)

    return(d_stats)

def Compute_Average_Stats_For_Test_Scenarios(test_stats_list):
        d_avg = [0,0,0,0,0,0,0]
        d_tot = [0,0,0,0,0,0,0]

        for i in range(len(test_stats_list)):
            curStat = test_stats_list[i]
            d_tot[0] = d_tot[0] + curStat[0]
            d_tot[1] = d_tot[1] + curStat[1]
            d_tot[2] = d_tot[2] + curStat[2]
            d_tot[3] = d_tot[3] + curStat[3]
            d_tot[4] = d_tot[4] + curStat[4]
            d_tot[5] = d_tot[5] + curStat[5]
        d_avg[0] = d_tot[0]/float(len(test_stats_list))
        d_avg[1] = d_tot[1]/float(len(test_stats_list))
        d_avg[2] = d_tot[2]/float(len(test_stats_list))
        d_avg[3] = d_tot[3]/float(len(test_stats_list))
        d_avg[4] = (d_tot[4])/float(len(test_stats_list))
        d_avg[5] = (d_tot[5])/float(len(test_stats_list))

        return(d_avg)

def Perturb_Loc_List(Loc_List):

    for i in range(len(Loc_List)):
        curLoc = Loc_List[i]
        nloc = curLoc[0]
        nodeId = curLoc[1]
        n_w_node = curLoc[2]
        curx = curLoc[3]
        cury = curLoc[4]
        rand1 = random.random()
        pert_angle = 360*rand1
        delta_x = RADIUS*math.cos(pert_angle)
        delta_y = RADIUS*math.sin(pert_angle)
        old_curx = curx
        old_cury = cury
        curLoc[3] = curx + delta_x
        curLoc[4] = cury + delta_y
        curLoc.append(old_curx)
        curLoc.append(old_cury)
    return(Loc_List)


def Print_Loc_List(Loc_List):
    print("Printing Location List of Length = ", len(Loc_List))
    f = open('Locations_EC.txt','w')
    for i in range(len(Loc_List)):
        curLoc = Loc_List[i]
        nloc = curLoc[0]
        nodeId = curLoc[1]
        n_w_node = curLoc[2]
        curx = curLoc[3]
        cury = curLoc[4]
        str1 = str(nloc) + ",  " + str(curx) + ",  " + str(cury) + "\n"
        f.write(str1)
        print(nloc, nodeId, n_w_node, curx, cury)



    f.close()


def Get_Set_Covering_Matrix_Gurobi(G, f_s):
    num_scen = len(f_s)
    print("tot_POTENTIAL_Engine_Co_locs = ", tot_locs)
    print("num_fires_in_scenario = ", num_scen)

    # NOTE ORDER BELOW CAREFULLY. COLUMNS j DECLARED FIRST...
    # For the Gurobi A-matrix, rows = fires, cols = engine locs
    amat = [[0 for j in range(tot_locs + 1)] for i in range(num_scen + 1)]
```

```
for ii in range((num_scen + 1)):
        for jj in range((tot_locs + 1)):
                amat[ii][jj] = 0

n_locs = 0

for n in G.nodes():
        # print("At node ", int(n))
        ECList = G.node[n]['ECList']
        NumEC = G.node[n]['NECLocs']

        for  k in range(len(ECList)):
                curLocList = ECList[k]
                n_locs = n_locs + 1
                curf_ID = curLocList[0]
                curfx = curLocList[1]
                curfy = curLocList[2]

                for j in range(len(f_s)):
                        curList = f_s[j]
                        sc_id = curList[0]
                        sc_node = curList[1]
                        sc_node_id = curList[2]
                        curx1 = curList[3]
                        cury1 = curList[4]
                        travel_dist = 0
                        travel_dist = get_travel_dist(curfx,curfy, curx1, cury1)
                        travel_time = (travel_dist*60)/SPEED2
                        cover_prob = get_cover_prob(travel_time)
                if(cover_prob >= CRIT_PROB):
                    amat[j+1][n_locs] = 1


    return(amat)


def Write_MPS_File_For_Gurobi(G, f_s):
    try:
        cmd1 = 'delete gurobimps.txt'
        os.system(cmd1)
    except:
        print('No file with name gurobimps.txt,so continuing...')


    mps_code = 0
    max_uncovered = int((1-BETA)*len(f_s))
    print('max_uncovered = ', max_uncovered, ' max_fires = ', len(f_s))
    try:
        # Get set covering matrix Amat
        amat = Get_Set_Covering_Matrix_Gurobi(G, f_s)
        nrows = len(f_s)
        ncols = tot_locs
        f2 = open('gurobimps.txt','w')
        f2.write('NAME PROB_SET_COVER ' + '\n')
        f2.write('ROWS ' + '\n')
        f2.write('  N COST  ' + '\n')
        for i in range(nrows):
            ii = i + 1
            cur_str = 'R' + str(ii)
            f2.write('  G ' + cur_str + '  ' + '\n')
        cur_str = 'RBETA'
```

```
f2.write('  L ' + cur_str + '  ' + '\n')


f2.write('COLUMNS ' + '\n')
for j in range(ncols):
    jj = j + 1
    cur_var = 'X' + str(jj)
    f2.write('  ' + cur_var + '  ' + 'COST' + '  +1' + '  ' + '\n')
    for i in range(nrows):
        ii = i + 1
        if(amat[ii][jj] == 1):
            cur_row = 'R' + str(ii)
            f2.write('  ' + cur_var + '  ' + cur_row + '  +1' + '  ' + '\n')


for i in range(nrows):
    ii = i + 1
    cur_var = 'S' + str(ii)
    cur_row = 'R' + str(ii)
    f2.write('  ' + cur_var + '  ' + cur_row + '  +1' + '  ' + '\n')
    cur_row = 'RBETA'
    f2.write('  ' + cur_var + '  ' + cur_row + '  +1' + '  ' + '\n')


f2.write('RHS  ' + '\n')
cur_row = 'RBETA'
f2.write('  rhs' + '  ' + cur_row + '   ' + str(max_uncovered) + '  ' + '\n')
for i in range(nrows):
    ii = i + 1
    cur_row = 'R' + str(ii)
    f2.write('  rhs' + '  ' + cur_row + '  +1' + '  ' + '\n')
```

```
        f2.write('BOUNDS   ' + '\n')
        for j in range(ncols):
            jj = j + 1
            cur_var = 'X' + str(jj)
            f2.write('  BV BOUND  ' + cur_var + '  ' + '  +1' + '  ' + '\n')
        for i in range(nrows):
            ii = i + 1
            cur_var = 'S' + str(ii)
            f2.write('  BV BOUND  ' + cur_var + '  ' + '  +1' + '  ' + '\n')

        f2.write('ENDATA  ' + '\n')
        mps_code = 1
    except:
        mps_code = -1

    return(mps_code)

def Build_Facility_List_From_Gurobi(G, Y1):
    fac_list = [[0, 0, 0, 0, 0]]
    n_locs = 0
    num_y_vars = len(Y1)
    #print('num_y_vars inside Build_Facility_List_From_Gurobi = ', num_y_vars)
    for i in range(len(Y1)):
        if(Y1[i] > 0):
            print(i, Y1[i], 'found a facility in Y array..')
    for n in G.nodes():
        # print("At node ", int(n))
        ECList = G.node[n]['ECList']
        NumEC = G.node[n]['NECLocs']
        for  k in range(len(ECList)):
            n_locs = n_locs + 1
            ii = n_locs - 1
            if(Y1[ii] > 0):
                #print('FOUND A FACILITY...')
                curLoc = get_facility(G, n_locs)
                fac_list.append(curLoc)
    fac_list.pop(0)
    k = len(fac_list)
    print('GUROBI IDENTIFIED ', k, ' FIRE ENGINE COMPANY LOCATIONS...')
    return(fac_list)


def Get_Gurobi_Locations_For_Scenario(G, f_s):
    import sys
    if "C:\gurobi652\win32\python27\lib" not in sys.path:
        sys.path.append("C:\gurobi652\win32\python27\lib")
    from gurobipy import *
    print ("INSIDE Get_Gurobi_Locations_For_Scenario FUNCTION...")

    max_uncovered = int((1-BETA)*len(f_s))
    print('max_uncovered = ', max_uncovered, ' max_fires = ', len(f_s))
    try:
        # Get set covering matrix Amat
        amat = Get_Set_Covering_Matrix_Gurobi(G, f_s)
        nrows = len(f_s)
        ncols = tot_locs
```

```
num_y_vars = tot_locs
num_s_vars = len(f_s)
# Create a new model
m = Model("mip1")
Y = [0]*num_y_vars
S = [0]*num_s_vars
# Create variables
for i in range(num_y_vars):
        cur_str = 'Y' + str(i+1)
        Y[i] = m.addVar(vtype=GRB.BINARY, name =cur_str)

for i in range(num_s_vars):
        cur_str = 'S' + str(i+1)
        S[i] = m.addVar(vtype=GRB.BINARY, name =cur_str)

# Integrate new variables
m.update()

# Set objective
cur_obj = 0
for i in range(num_y_vars):
        cur_obj = cur_obj + Y[i]
m.setObjective(cur_obj, GRB.MINIMIZE)

# Add set covering constraints:
for j in range(len(f_s)):
        jj = j + 1
        cur_str = 'C' + str(jj)
        cur_lhs = 0
```

```
    for i in range(num_y_vars):
        if(amat[jj][i+1] == 1):
            cur_lhs = cur_lhs + Y[i]

    cur_lhs = cur_lhs + S[j]
    m.addConstr(cur_lhs >= 1, cur_str)

# Add final constraint:
cur_lhs = 0
for j in range(len(f_s)):
    cur_lhs = cur_lhs + S[j]
m.addConstr(cur_lhs <= max_uncovered, "RBETA")

m.optimize()

for v in m.getVars():
    if(v.x > 0):
        print('%s %g' % (v.varName, v.x))

print('Obj: %g' % m.objVal)

print("SUCCESFUL GUROBI COMPLETION, YAY!!")
Y1 = [0]*num_y_vars
for v in m.getVars():
    if(v.x > 0):
        cur_str = v.varName
        if(cur_str[0] == 'S'):
            continue
        elif(cur_str[0] == 'Y'):
```

```
                    cur_index = int(cur_str[1:])
                    Y1[cur_index - 1] = 1
                    print('Y of ' , cur_index-1, ' is 1')
                    print('%s %g' % (v.varName, v.x))
        #for i in range(num_y_vars):
        #   if(Y[i] > 0):
        #       print("Y variable ", i, "chosen by Gurobi")

        #Time_pass = int(input("Time_pass :"))
        fac_list = Build_Facility_List_From_Gurobi(G, Y1)
        return(fac_list)
    except GurobiError:
        print('Encountered a Gurobi error')
        sys.exit('aa! Errors!')

    except AttributeError:
        print('Encountered an attribute error')
        sys.exit('aa! Errors!')

def Get_Equitable_Gurobi_Locations_For_Scenario(G, f_s):
    import sys
    if "C:\gurobi652\win32\python27\lib" not in sys.path:
        sys.path.append("C:\gurobi652\win32\python27\lib")
    from gurobipy import *
    print ("INSIDE Get_Equitable_Gurobi_Locations_For_Scenario FUNCTION...")

    min_covered = int((BETA)*len(f_s))
    print('min_covered = ', min_covered, 'OUT OF max_fires = ', len(f_s))
    max_uncovered = int((1-BETA)*len(f_s))
```

```
print('max_uncovered = ', max_uncovered, ' max_fires = ', len(f_s))

try:
    # Get set covering matrix Amat
    amat = Get_Set_Covering_Matrix_Gurobi(G, f_s)
    #nrows = len(f_s)
    ncols = tot_locs

    num_y_vars = tot_locs
    num_x_vars = len(f_s)*tot_locs
    num_s_vars = len(f_s)
    num_scen = len(f_s)
    eq_const = float(EQUITY_SPREAD)*float(len(f_s))/100.0
    eq_const = int(eq_const)

    print('eq_const = ', eq_const)

    # Create a new model
    m = Model("mip1")
    Y = [0]*num_y_vars
    S = [0]*num_s_vars
    X = [[0 for j in range(tot_locs + 1)] for i in range(num_scen + 1)]

    # Create variables
    for i in range(num_y_vars):
        cur_str = 'Y' + str(i+1)
        Y[i] = m.addVar(vtype=GRB.BINARY, name =cur_str)

    for i in range(num_s_vars):
```

```
    cur_str = 'S' + str(i+1)
    S[i] = m.addVar(vtype=GRB.BINARY, name =cur_str)


for i in range(len(f_s)):
    for j in range(tot_locs):
        if(amat[i+1][j+1] == 1):
            cur_str = 'X' + '_' + str(i+1) + '_' + str(j+1)
            X[i][j] = m.addVar(vtype=GRB.BINARY, name =cur_str)
# ADD MAXI AND MINI


cur_str = 'maxi'
maxi = m.addVar(vtype=GRB.INTEGER, name =cur_str)


cur_str = 'mini'
mini = m.addVar(vtype=GRB.INTEGER, name =cur_str)


# Integrate new variables
m.update()
print('FINISHED MAKING VARIABLES...')
# Set objective
cur_obj = 0
for i in range(num_y_vars):
    cur_obj = cur_obj + Y[i]
m.setObjective(cur_obj, GRB.MINIMIZE)
print('FINISHED MAKING OBJECTIVE...')
# Add set covering constraints:
for j in range(num_y_vars):
    jj = j + 1
    cur_lhs = 0
```

```
            cur_lhs1 = 0
            cur_lhs2 = 0
            cur_lhs3 = 0

            for k in range(len(f_s)):
                kk = k + 1
                if(amat[kk][jj] == 1):
                    cur_lhs = cur_lhs + X[k][j]

            #print('jj = ', jj)
            cur_lhs1 = cur_lhs - (num_scen*Y[j])
            cur_str = 'C' + str(jj)
            m.addConstr(cur_lhs1 <= 0, cur_str)
            cur_lhs2 = cur_lhs - mini
            cur_str = 'MINI' + str(jj)
            m.addConstr(cur_lhs2 >= 0, cur_str)
            cur_lhs3 = cur_lhs - maxi
            cur_str = 'MAXI' + str(jj)
            m.addConstr(cur_lhs3 <= 0, cur_str)
            #print('Finished constraint  ', jj)
        print('FINISHED MAXI/MINI CONSTRAINTS and Xij <= |I|yj...')
        # ADD CONSTRAINTS TO COVER FIRE WITH ONE EC
        for i in range(len(f_s)):
            cur_lhs = 0
            cur_str = 'F' + str(i+1)
            for k in range(num_y_vars):
                kk = k + 1
                if(amat[i+1][kk] == 1):
                    cur_lhs = cur_lhs + X[i][k]
```

```
    cur_lhs = cur_lhs + S[i]
    print('just before cur_str = ', cur_str)
    m.addConstr(cur_lhs >= 1, cur_str)

print('FINISHED ASSIGNMENT CONSTRAINTS...')
# Add final 3 constraints:
# Add final constraint:
cur_lhs = 0
for j in range(len(f_s)):
    cur_lhs = cur_lhs + S[j]
m.addConstr(cur_lhs <= max_uncovered, "RBETA1")

cur_lhs = 0

for i in range(num_y_vars):
    for k in range(len(f_s)):
        kk = k + 1
        if(amat[kk][i+1] == 1):
            cur_lhs = cur_lhs + X[k][i]

m.addConstr(cur_lhs >= min_covered, "RBETA2")

# ADD EQUITY CONSTRAINT
m.addConstr(maxi - mini <= eq_const, "EQUITY")

print('BUILT EQUITY MODEL..NOW OPTIMIZING..')
m.optimize()

for v in m.getVars():
        if(v.x > 0):
            print('%s %g' % (v.varName, v.x))

    print('Obj: %g' % m.objVal)

    print("SUCCESFUL EQUITY GUROBI COMPLETION, YAY!!")
    Y1 = [0]*num_y_vars
    for v in m.getVars():
        if(v.x > 0):
            cur_str = v.varName
            if(cur_str[0] == 'S'):
                continue
            elif(cur_str[0] == 'Y'):
                cur_index = int(cur_str[1:])
                Y1[cur_index - 1] = 1
                print('Y of ' , cur_index-1, ' is 1')
                print('%s %g' % (v.varName, v.x))
    #for i in range(num_y_vars):
    #   if(Y[i] > 0):
    #       print("Y variable ", i, "chosen by Gurobi")

    #Time_pass = int(input("Time_pass :"))
    fac_list = Build_Facility_List_From_Gurobi(G, Y1)
    return(fac_list)
except GurobiError:
    print('Encountered a Gurobi error')
    sys.exit('aa! Errors!')

except AttributeError:
```

```
                print('Encountered an attribute error')
                sys.exit('aa! Errors!')

def Get_Locations_For_Scenario(G, f_s):
        num_scen = len(f_s)
        print("tot_POTENTIAL_Engine_Co_locs = ", tot_locs)
        print("num_fires_in_scenario = ", num_scen)
        # NOTE ORDER BELOW CAREFULLY. COLUMNS j DECLARED FIRST...
        sc_cover = [[0 for j in range(num_scen + 1)] for i in range(tot_locs + 1)]
        sc_yet_to_cover = [1 for i in range(num_scen + 1)]

        for ii in range((tot_locs + 1)):
                for jj in range((num_scen + 1)):
                        # print("ii = ", ii, "jj = ", jj)
                        sc_cover[ii][jj] = 0
                        sc_yet_to_cover[jj] = 1

        sc_yet_to_cover[0] = 0
        T_loc_sc = ()
        n_locs = 0

        for n in G.nodes():
                # print("At node ", int(n))
                ECList = G.node[n]['ECList']
                NumEC = G.node[n]['NECLocs']

                for  k in range(len(ECList)):
                        curLocList = ECList[k]
                        n_locs = n_locs + 1
```

```
            curf_ID = curLocList[0]
            curfx = curLocList[1]
            curfy = curLocList[2]

            for j in range(len(f_s)):
                curList = f_s[j]
                sc_id = curList[0]
                sc_node = curList[1]
                sc_node_id = curList[2]
                curx1 = curList[3]
                cury1 = curList[4]
                travel_dist = 0
                travel_dist = get_travel_dist(curfx,curfy, curx1, cury1)
                travel_time = (travel_dist*60)/SPEED2
                cover_prob = get_cover_prob(travel_time)
                if(cover_prob >= CRIT_PROB):
                    T1 = (n_locs, int(n), curf_ID, curfx, curfy, sc_id, sc_node,
sc_node_id, curx1, cury1)
                    T_loc_sc = T_loc_sc + T1
                    sc_cover[n_locs][j+1] = 1

    tot = get_num_fires_yet_to_cover(sc_yet_to_cover)
    #print("tot = ", tot)
    flag = set_flag(tot, sc_yet_to_cover)
    #print("flag = ", flag)
    #print("LOCATION-SCENARIO TUPLES..= ", len(T_loc_sc))
    #for i in range(len(T_loc_sc)):
        #print(T_loc_sc[i])
    fac_list = [[0, 0, 0, 0, 0]]
    while(flag == 0):
        flag = 1
        n_locs = get_index_for_next_location(G, f_s, sc_cover, sc_yet_to_cover)
        curLoc = get_facility(G, n_locs)
        fac_list.append(curLoc)
        flag2 = -1
        flag2 = Update_SC_Yet_To_Cover(G, f_s, n_locs, sc_cover, sc_yet_to_cover)
        #if(flag2 == 1):
            #print("sc_yet_to_cover has been updated...")
        #else:
            #print("Error in update of sc_yet_to_cover...")
        tot = get_num_fires_yet_to_cover(sc_yet_to_cover)
        #print("Number of Fires YET to be covered = ", tot)
        flag = set_flag(tot, sc_yet_to_cover)
    fac_list.pop(0)
    return(fac_list)


def Get_Mutation_Code(sing_eng_list):
    eng_part = sing_eng_list[1]
    code1 = 0
    if((eng_part[0] == 1) and (eng_part[1] == 0)):
        code1 = 1
    elif((eng_part[0] == 0) and (eng_part[1] == 1)):
        code1 = 2
    elif((eng_part[0] == 1) and (eng_part[1] == 1)):
        code1 = 3
    elif((eng_part[0] == 2) and (eng_part[1] == 0)):
        code1 = 4
    elif((eng_part[0] == 0) and (eng_part[1] == 2)):
        code1 = 5
    code2 = code1
    while (code2 == code1) :
        rand1 = random.random()
        if(rand1 <= 0.2):
            code2 = 1
        elif(rand1 <= 0.4):
            code2 = 2
        elif(rand1 <= 0.6):
            code2 = 3
        elif(rand1 <= 0.8):
            code2 = 4
        elif(rand1 <= 1.0):
            code2 = 5

    return(code2)


def Mutate_Single_Engine_List(sing_eng_list):
    sing_eng_list1 = []
    sing_eng_list1.append(sing_eng_list[0])
    sing_eng_list1.append([0]*NUM_ENG_TYPES)
    code  = Get_Mutation_Code(sing_eng_list)
    eng_part = sing_eng_list1[1]
    if (code == 1):
        eng_part[0] = 1
        eng_part[1] = 0
    elif(code == 2):
        eng_part[0] = 0
        eng_part[1] = 1
    elif(code == 3):
        eng_part[0] = 1
        eng_part[1] = 1
    elif(code == 4):
        eng_part[0] = 2
        eng_part[1] = 0
    elif(code == 5):
        eng_part[0] = 0
        eng_part[1] = 2

    return(sing_eng_list1)

def Perform_Mutations(gen_pop_list):
    for i in range(len(gen_pop_list)):
        rand1 = random.random()
        if(rand1 <= MUTATION_PROB):
            cur_pop_mem = gen_pop_list[i]
            #print('POP MEMBER BEFORE MUTATION ', cur_pop_mem)
            eng_list = cur_pop_mem[1]
            for j in range(len(eng_list)):
                sing_eng_list = eng_list[j]
                #print('BEFORE MUTATION ', eng_list[j])
                rand2 = random.random()
                if(rand2 <= GENE_MUTATION_PROB):
                    sing_eng_list1 = Mutate_Single_Engine_List(sing_eng_list)
                    eng_list[j] = sing_eng_list1
                    #print('AFTER MUTATION ', eng_list[j])
            #print('POP MEMBER AFTER MUTATION ', cur_pop_mem)
            #Time_pass = int(input("Time_pass :"))

    return(gen_pop_list)

def Perform_Crossover(gen_pop_list):

    delta1 = 0.01
    rand1 = random.random()
    rand2 = rand1
    delta = 2*delta1
    x = rand1 - rand2
    while (x < delta):
        rand2 = random.random()
        x = rand1 - rand2
        if x < 0:
            x = abs(x)
        #print('rand1 = ', rand1, 'rand2 = ', rand2)



    #
    if (rand2 < rand1):
        temp = rand1
        rand1 = rand2
        rand2 = temp

    #print('rand1 = ', rand1, 'rand2 = ', rand2)
    pop_size = len(gen_pop_list)
    index1 = -1
    index2 = -1
```

```
llimit = 0.0
ulimit = delta1
cntr = int(1/delta1)
for i in range(cntr):
    if((llimit <= rand1) and (rand1 < ulimit)):
        index1 = i
    elif((llimit <= rand2) and (rand2 < ulimit)):
        index2 = i
    llimit = llimit + delta1
    ulimit = ulimit + delta1

#print('INDEX1 = ', index1, 'INDEX2 = ', index2)
if(index1 == index2):
    print('index1 cannot be equal to index2 for crossover operation...')
    exit()
for i in range(pop_size):
    if(index1 == i):
        pop1 = gen_pop_list[i]
    elif(index2 == i):
        pop2 = gen_pop_list[i]

delta2 = float(1/float(NUM_ECS))
#print('delta2 = ', delta2)
llimit = 0
ulimit = delta2
rand3 = random.random()
index3 = 0
for j in range(NUM_ECS):
    if((llimit <= rand3) and (rand3 < ulimit)):
        index3 = j
        if(index3 == (NUM_ECS - 1)):
            index3 = NUM_ECS - 2
    llimit = llimit + delta2
    ulimit = ulimit + delta2
#print('INDEX3 = ', index3)
#print(pop1, pop2)
cur_pop_mem = []
index4 = len(gen_pop_list) + 1
cur_pop_mem.append(index4)
eng_list = []

for i in range(NUM_ECS):
    ii = i  + 1
    sing_eng_list = []
    sing_eng_list.append(i)
    sing_eng_list.append([0]*NUM_ENG_TYPES)
    if(i <= index3):
        eng_list1 = pop1[1][i]
    elif(i > index3):
        eng_list1 = pop2[1][i]
    for l in range(NUM_ENG_TYPES):
            sing_eng_list[1][l] = eng_list1[1][l]

    eng_list.append(sing_eng_list)
cur_pop_mem.append(eng_list)
cur_pop_mem.append(0)
cur_pop_mem.append(0)
cur_pop_mem.append(0)
```

```
    gen_pop_list.append(cur_pop_mem)
    pop_size = len(gen_pop_list)
    #print(cur_pop_mem)
    #print('pop_size = ', pop_size)
    #Time_pass = int(input("Time_pass :"))
    return(gen_pop_list)




def Update_Engine_Costs(gen_pop_list):
    pcost = PUMPER_COST + PUMPER_CREW_COST
    lcost = LADDER_COST + LADDER_CREW_COST
    pop_size = len(gen_pop_list)
    for i in range(pop_size):
        cur_pop_mem = gen_pop_list[i]
        eng_list = cur_pop_mem[1]
        pump_cost = 0
        ladd_cost = 0
        for kk in range(len(eng_list)):
            cur_eng_loc = eng_list[kk][0]
            pumper_eng = eng_list[kk][1][0]
            ladder_eng = eng_list[kk][1][1]
            pump_cost = pump_cost + (pumper_eng*pcost)
            ladd_cost = ladd_cost + (lcost*ladder_eng)

        cur_pop_mem[3] = pump_cost
        cur_pop_mem[4] = ladd_cost
    return(gen_pop_list)
```

```
def Update_Num_Infeasible_Fires(gen_pop_list, Pithy_Train_List, Test_List):
    pop_size = len(gen_pop_list)
    tot_num_fires = 0
    for j in range(len(Test_List)):
        f_s = Test_List[j]
        tot_num_fires = tot_num_fires + len(f_s)
    print('tot_num_fires = ', tot_num_fires)

    for i in range(pop_size):
        cur_pop_mem = gen_pop_list[i]
        eng_list = cur_pop_mem[1]
        if(len(eng_list) != len(Pithy_Train_List)):
            print('ERROR! ERROR! Lengths of Pithy Train List and Genetic Population Engine List dont match..')
            exit()

        num_infeas_fires = 0
        for j in range(len(Test_List)):
            f_s = Test_List[j]
            for jjj in range(len(f_s)):
                flag1 = 1
                flag2 = 1
                curList = f_s[jjj]
                curx1 = curList[3]
                cury1 = curList[4]
                for kk in range(len(eng_list)):
                    cur_eng_loc = eng_list[kk][0]
                    pumper_eng = eng_list[kk][1][0]
                    ladder_eng = eng_list[kk][1][1]
                    for k in range(len(Pithy_Train_List)):
                        if(k == cur_eng_loc):
                            curLoc = Pithy_Train_List[k]
                            curfx = curLoc[3]
                            curfy = curLoc[4]
                            travel_dist = 0
                            travel_dist = get_travel_dist(curfx,curfy, curx1,
cury1)
                            travel_time = (travel_dist*60)/SPEED2
                            if(travel_time <= MIN_TIME2 and pumper_eng
== 1):
                                flag1 = 0
                            if(travel_time <= MAX_TIME2 and ladder_eng
== 1):
                                flag2 = 0

                if(flag1 == 1 or flag2 == 1):
                    num_infeas_fires = num_infeas_fires + 1

        print(num_infeas_fires, "num_infeas_fires for population member ", cur_pop_mem[0])
        covered_prop = float(num_infeas_fires)*100/float(tot_num_fires)
        cur_pop_mem[2] = covered_prop

    return(gen_pop_list)

def Create_Population_For_GA():

    gen_pop_list = []
```

```
for j in range(GEN_POP):
    jj = j + 1
    cur_pop_mem = []
    cur_pop_mem.append(jj)
    eng_list = []

    for i in range(NUM_ECS):
        ii = i  + 1
        sing_eng_list = []
        sing_eng_list.append(i)
        sing_eng_list.append([0]*NUM_ENG_TYPES)
        for l in range(NUM_ENG_TYPES):
            sing_eng_list[1][l] = 1
            if(l == (NUM_ENG_TYPES - 1)):
                rand_no = random.random()
            if((l == (NUM_ENG_TYPES - 1)) and (rand_no < CRIT_PROB2)):
                sing_eng_list[1][l] = 0
                #print(rand_no, CRIT_PROB)
                #Time_pass = int(input("Time_pass :"))
        eng_list.append(sing_eng_list)
    cur_pop_mem.append(eng_list)
    # create bucket for proportion of infeasible fires
    cur_pop_mem.append(0)
    # create bucket for pumper engine related costs
    cur_pop_mem.append(0)
    # create bucket for ladder engine related costs
    cur_pop_mem.append(0)
    gen_pop_list.append(cur_pop_mem)
return(gen_pop_list)
```

```
def Generate_Scenario_List(G, NUM_SCENARIOS):
    Scenario_List = []
    for k in range(NUM_SCENARIOS):
        print("Generating Scenario ", k + 1)
        f_s = generate_fire_scenario(G)
        Scenario_List.append(f_s)

    print("Length of Generated Scenario List = ", len(Scenario_List))
    return(Scenario_List)

def generate_fire_scenario(G):
    fire_scenario = [[0,0,0,0,0]]
    num_total_fires = 0
    for n in G.nodes():
        num_neigh_fires = 0
        MU = G.node[n]['mu_fires']
        max_fires = G.node[n]['max_fires']
        xmin = G.node[n]['Xcor'] - G.node[n]['radius']
        xmax = G.node[n]['Xcor'] + G.node[n]['radius']
        ymin = G.node[n]['Ycor'] - G.node[n]['radius']
        ymax = G.node[n]['Ycor'] + G.node[n]['radius']
        n_sim_fires = sim_num_fires(G, n)
        #print("MU = ", MU, "sim_fires = ", n_sim_fires,"max_fires = ", max_fires)
        for j in range(n_sim_fires):
            curx = xmin + random.random()*(xmax - xmin)
            cury = ymin + random.random()*(ymax - ymin)
            num_neigh_fires = num_neigh_fires + 1
            num_total_fires = num_total_fires + 1
            cur_fire = [num_total_fires, int(n), num_neigh_fires, curx, cury]
            fire_scenario.append(cur_fire)

    fire_scenario.pop(0)
    return(fire_scenario)


def Set_Engine_Company_Candidate_Locations(G, mesh_width):
    for n in G.nodes():
        nloc = 0
        xmin = G.node[n]['Xcor'] - G.node[n]['radius']
        xmax = G.node[n]['Xcor'] + G.node[n]['radius']
        ymin = G.node[n]['Ycor'] - G.node[n]['radius']
        ymax = G.node[n]['Ycor'] + G.node[n]['radius']
        curx = xmin - mesh_width
        ECLocs = [[0,0,0]]


        while curx < xmax:
            curx = curx + mesh_width
            cury = ymin - mesh_width
            while cury < ymax:
                cury = cury + mesh_width
                nloc = nloc + 1
                curList = [nloc, curx, cury]
                ECLocs.append(curList)

        ECLocs.pop(0)
        G.node[n]['ECList'] = ECLocs
        G.node[n]['NECLocs'] = len(ECLocs)

print("Reading Neighborhood Input data into a graph...")
import networkx as nx
G = nx.read_adjlist("adj-matrix-phillie.txt",create_using = nx.DiGraph(),nodetype = int)
print("Finished creating graph model..")

#print("PRINTING NODES...")
#print(G.nodes())
#print("PRINTING EDGES")
#print(G.edges())
print("READING NEIGHBORHOOD POPULATION DATA AND COORDINATES...")
# START READING NODE WEIGHTS (= # FIRE INCIDENTS AT NODE)
F1 = open('Neigh-node-pops-v2.txt')
for line in F1:
    line.rstrip()
    parts = line.split('	')
    cur_node = int(parts[0])
    cur_pop = int(parts[1])
    cur_X = float(parts[2])
    cur_Y = float(parts[3])
    cur_fnum = float(parts[4])
    for n in G.nodes():
        if(int(n) == cur_node):
            G.node[n]['node_pop'] = cur_pop
            if(SCALE_FIRES > 0):
                G.node[n]['mu_fires'] = cur_fnum/SCALE_FIRES
            else:
                G.node[n]['mu_fires'] = cur_fnum
```

```
			G.node[n]['Xcor'] = cur_X
			G.node[n]['Ycor'] = cur_Y
			G.node[n]['EngineLoc'] = -1
#print('PRINTING NODE POP, MU_FIRES, XCOR, YCOR, EngineLocIndicator')
#for n in G.nodes():
#	print(n, G.node[n]['node_pop'],G.node[n]['mu_fires'], G.node[n]['Xcor'],G.node[n]['Ycor'], G.node[n]['EngineLoc'])

# END READING NODE WEIGHTS

#CREATE EDGE LENGTHS
for e1 in G.edges():
	#print(e1[0],e1[1])
	i = get_fro_node(G, e1)
	#print(e1[0],e1[1], i)
	j = get_to_node(G, e1)
	#print(e1[0],e1[1], j)
	xdiff = G.node[i]['Xcor'] - G.node[j]['Xcor']
	xdiff2 = xdiff*xdiff
	ydiff = G.node[i]['Ycor'] - G.node[j]['Ycor']
	ydiff2 = ydiff*ydiff
	dist = pow((xdiff2+ydiff2),0.5)
	G.edge[i][j]['weight'] = dist

print("FINISHED CREATING EDGE LENGTHS")

#for e in G.edges():
	#print(e[0],e[1],G.edge[e[0]][e[1]]['weight'])
```

```
print("Calculating Shortest Path Lengths for POPULATION COVERAGE MODEL...")
SP_Lengths = nx.shortest_path_length(G,weight='weight')
print("Finished Calculating Shortest Path Lengths for POPULATION COVERAGE MODEL...")

# Generate Maximum # Fires in a Neighborhood for SIMULATION
# Generate "Neighborhood Radius" for fires in a neighborhood
for n in G.nodes():
    G.node[n]['max_fires'] = get_max_num_fires(G, n, EPSILON)
    G.node[n]['radius'] = get_fire_radius(G,n)

# NOW DETERMINE MIN_X, MAX_X, MIN_Y, MAX_Y, MIN_RADIUS and MAX_RADIUS
ITER = 0
MAX_RADIUS = 0
for n in G.nodes():
    ITER = ITER + 1
    if(ITER == 1):
        MIN_X = G.node[n]['Xcor']
        MIN_Y = G.node[n]['Ycor']
        MAX_X = G.node[n]['Xcor']
        MAX_Y = G.node[n]['Ycor']
        MIN_RADIUS = G.node[n]['radius']
    elif(ITER > 1):
        if(G.node[n]['Xcor'] < MIN_X):
            MIN_X = G.node[n]['Xcor']
        if(G.node[n]['Ycor'] < MIN_Y):
            MIN_Y = G.node[n]['Ycor']
        if(G.node[n]['Xcor'] > MAX_X):
            MAX_X = G.node[n]['Xcor']
        if(G.node[n]['Ycor'] > MAX_Y):
            MAX_Y = G.node[n]['Ycor']


    if(G.node[n]['radius']> MAX_RADIUS):
        MAX_RADIUS = G.node[n]['radius']
    if(G.node[n]['radius']< MIN_RADIUS):
        MIN_RADIUS = G.node[n]['radius']

print("MIN_X, MAX_X, MIN_Y, MAX_Y = ", MIN_X, MAX_X, MIN_Y, MAX_Y)
print("MIN_RADIUS, MAX_RADIUS=", MIN_RADIUS, MAX_RADIUS)
# FINISHED DETERMINING MIN_X, MAX_X, MIN_Y, MAX_Y, MIN_RADIUS and MAX_RADIUS



#for n in G.nodes():
#    print(int(n), "MU =", G.node[n]['mu_fires'],"Max fires=", G.node[n]['max_fires'],"Radius=", G.node[n]['radius'])
#for n1 in G.nodes():
#    for n2 in G.nodes():
#        print(n1,n2,SP_Lengths[n1][n2])
print("\n")
print("For POPULATION COVERAGE MODEL, Enter response time limit in MINUTES...")
MINS = int(input("MINS = Response time limit(4-24 mins recommended):"))
print("Enter fire engine TRAVEL SPEED...")
SPEED = int(input("Fire engine speed:(20mph-heavy-traffic to 40-mph-light traffic:)"))
print("\n")
DIST_THRESHOLD = float(float(SPEED*MINS)/60.0)
# print(DIST_THRESHOLD)

# DEVELOP MATRIX FOR SET COVERING PROBLEM
```

```
nnodes = len(G.nodes())
cover = [[0 for i in range(nnodes+1)] for j in range(nnodes+1)]
yet_to_cover = [1 for i in range(nnodes+1)]
for n1 in G.nodes():
    #print('yet to cover for',n1, yet_to_cover[n1])
    for n2 in G.nodes():
        cover[n1][n2] = 0
        #print(n1,n2,SP_Lengths[n1][n2])
        if(SP_Lengths[n1][n2] <= DIST_THRESHOLD):
            cover[n1][n2] = 1
        #print(n1, n2, cover[n1][n2])

print('For POPULATION COVERAGE MODEL,PLEASE SELECT A HEURISTIC TO SOLVE THE PROBLEM')
print('ENTER 1 IF HEURISTIC GUIDED BY POPULATION')
print('ENTER 2 IF HEURISTIC GUIDED BY # OF TERRITORIES COVERED')
print('ENTER 3 IF A COMBINATION OF POPULATION AND # TERRITORIES TO BE USED')
HEUR = int(input("ENTER HEURISTIC NUMBER VALUE(1-3):"))

print("Beginning Statistics for POPULATION COVERAGE MODEL...")
num_to_cover = get_num_to_cover(G, yet_to_cover)
print('num_to_cover=',num_to_cover)

while (num_to_cover > 0) :
    n = get_next_location(G, SP_Lengths, cover, yet_to_cover, HEUR)
    print('LOCATE NEXT ENGINE COMPANY AT NODE...', n)
    G.node[n]['EngineLoc'] = +1
    ncount = get_num_engines(G)
    print(ncount,' ENGINE COMPANIES CURRENTLY...')
    for n1 in G.nodes():
        if(cover[n][n1] == 1):
            yet_to_cover[n1] = 0
    pop_perc_covered = get_pop_perc_covered(G, yet_to_cover)
    print('POP PERC COVERED = ', pop_perc_covered)
    node_perc_covered = get_node_perc_covered(G, yet_to_cover)
    print('NODE PERC COVERED = ', node_perc_covered)
    num_to_cover = get_num_to_cover(G, yet_to_cover)
    print('num_to_cover=',num_to_cover)

Get_Equity_Stats(G, cover)
ncount = get_num_engines(G)
print(ncount,' ENGINE COMPANIES CURRENTLY...')
print('TIME THRESHOLD, ENGINE SPEED(mph), NUM_ENGINE_LOCS, HEUR_VALUE')
print(MINS, SPEED, ncount, HEUR)
print("Finished Statistics for POPULATION COVERAGE MODEL...")

# BEGIN ROBUST OPTIMIZATION
print("BEGINNING ROBUST OPTIMIZATION...")
print("For ROBUST COVERAGE MODEL, Enter BETA (float in range 0 to 1)")
print("BETA is the % of fires covered with threshold probability p (to be defined)")
BETA = float(input("BETA = :"))

flag = 1
while (flag > 0):
    flag = 0
    print("For ROBUST COVERAGE MODEL, Enter MIN_TIME(4-12 mins recommended)")
    print("Responses within MIN_TIME will cover the incident with p = 1 (certain coverage)")
    MIN_TIME = int(input("MIN_TIME :"))
    print("For ROBUST COVERAGE MODEL, Enter MAX_TIME(10-20 mins recommended)")
    print("Responses after MAX_TIME will NOT cover the incident, coverage p = 0 (NO coverage)")
    MAX_TIME = int(input("MAX_TIME :"))

    if(MIN_TIME >= MAX_TIME):
        print("ERROR: MIN_TIME AS HIGH AS MAX_TIME..Try again..")
        flag = 1

print("\n")
print("RE-CONFIRM fire engine TRAVEL SPEED for the ROBUST OPTIMIZATION MODEL...")
SPEED2 = int(input("Fire engine speed:(20mph-heavy-traffic to 40-mph-light traffic:)"))
print("\n")
print("\n")
print("Enter critical probability threshold for evaluating coverage level...")
print("Enter a number between 0 and 1")
CRIT_PROB = float(input("CRIT_PROB :"))
print("\n")
Set_Engine_Company_Candidate_Locations(G, mesh_width)
tot_locs = get_total_locs(G)
print("\n")
print("TOTAL POTENTIAL Engine Company Locations = tot_locs = ", tot_locs)

for n in G.nodes():
    #print("PRINTING ECLOCS FOR NODE", int(n))
    ECList = G.node[n]['ECList']
    NumEC = G.node[n]['NECLocs']

    for  j in range(len(ECList)):
        curList = ECList[j]
        # print(curList[0], curList[1], curList[2])

NUM_TRAIN = int(input("Enter number of TRAINING scenarios to be generated:"))
NUM_TEST = int(input("Enter number of TESTING scenarios to be generated:"))
print("\n")

Train_List = Generate_Scenario_List(G, NUM_TRAIN)
Test_List = Generate_Scenario_List(G, NUM_TEST)

Train_Loc_List = []
max_train_list_length = 0
for k in range(len(Train_List)):
    print("Training Scenario = ", k + 1)
    f_s = Train_List[k]
    if(GUROBI_OPT == 0):
        Loc_List = Get_Locations_For_Scenario(G, f_s)
    elif(GUROBI_OPT == 1):
        Loc_List = Get_Gurobi_Locations_For_Scenario(G, f_s)
        #Loc_List = Get_Equitable_Gurobi_Locations_For_Scenario(G, f_s)
    # THIS FUNCTION HAS BEEN ADDED TO DO GUROBI OPTIMIZATION AND WRITE THE MPS FILE
    mps_code = Write_MPS_File_For_Gurobi(G, f_s)
    if(mps_code == 1):
        print("MPS FILE WRITTEN...\n")
    elif(mps_code <= 0):
        print('mps_code =', mps_code)
    if(len(Loc_List) > max_train_list_length):
        max_train_list_length = len(Loc_List)
    Print_Loc_List(Loc_List)
    Train_Loc_List.append(Loc_List)
    #for j in range(len(f_s)):
    #   curList = f_s[j]
    #   if(j == (len(f_s) - 1)):
    #       print("Printing Last Training scenario ", j + 1)
    #       print(curList[0], curList[1], curList[2], curList[3], curList[4])

print("Length of Train_Loc_List = ", len(Train_Loc_List))
print("For ENSEMBLE_CODE, enter a 1 if you want to base ensemble on scenarios")
print("For ENSEMBLE_CODE, enter a 2 if you want to base ensemble on chosen facilities")
ENSEMBLE_CODE = int(input("Enter ENSEMBLE_CODE:"))
Master_Train_Loc_List = Make_Master_Train_Loc_List(Train_Loc_List, Train_List, ENSEMBLE_CODE)
MTL_List_sorted = Sort_Master_Train_Loc_List(Master_Train_Loc_List)
#print("Length of Master_Train_Loc_List = ", len(Master_Train_Loc_List))
#Print_Loc_List(Master_Train_Loc_List)
#print("Printing SORTED Master Train Location List..")
#for i in range(len(MTL_List_sorted)):
    #curLoc = MTL_List_sorted[i]
    #print(curLoc[0], curLoc[1], curLoc[2], curLoc[3], curLoc[4], curLoc[5])
Lower_B = max_train_list_length
Upper_B = int(Perc_List_Increase*Lower_B)

#print("Enter MAXIMUM NUMBER of ENGINE COMPANY LOCATIONS to be selected")
#print("You must enter a number in the range ", Lower_B, "to", Upper_B)
#MAX_LOCS = int(input("MAX_LOCS :"))
MAX_LOCS = Upper_B
Full_Train_List = Make_Full_Train_List(Train_List)
#print("Length of FULL_TRAIN_LIST = ", len(Full_Train_List))
```

```
# get_pithy_list simply selects the top lists after sorting.
# This function below has been commented out
#Pithy_Train_List = get_pithy_list(MTL_List_sorted, MAX_LOCS)
# function get_pithy_train_list considers training scenarios...

Pithy_Train_List = get_pithy_train_list(MTL_List_sorted, Full_Train_List, MAX_LOCS)
print("NUMBER OF FINAL ENGINE COMPANY LOCATIONS SELECTED = ", len(Pithy_Train_List))
print("PRINTING COORDINATES OF SELECTED ENGINE COMPANIES...")
Print_Loc_List(Pithy_Train_List)
if(DO_PERTURB == 1):
	Pithy_Train_List = Perturb_Loc_List(Pithy_Train_List)

test_stats_list = []
for k in range(len(Test_List)):
	print("Test Scenario = ", k + 1)
	f_s = Test_List[k]
	test_stats = get_average_stats_for_test_scenario(Pithy_Train_List, f_s)
	print("Estimated BETA for TEST scenario = ", test_stats[0])
	print("MAXIMUM response time for ANY fire = ", test_stats[1])
	print("AVERAGE response time for Test Scenario fires = ", test_stats[2])
	#print("MINIMUM response time for ANY fires = ", test_stats[3])
	print("MAX % of scenarios covered by any engine location = ", test_stats[4])
	print("MIN % of scenarios covered by any engine location = ", test_stats[5])
	test_stats_list.append(test_stats)
	print(" ")
	print(" ")
	#for j in range(len(f_s)):
	#	curList = f_s[j]
	#	if(j == (len(f_s) - 1)):
    #       print("Printing Last Test scenario ", j + 1)
    #       print(curList[0], curList[1], curList[2], curList[3], curList[4])



Average_Test_Stats = Compute_Average_Stats_For_Test_Scenarios(test_stats_list)
print(" ")
print(" ")
print("Average BETA for ALL TEST scenarios = ", Average_Test_Stats[0])
print("Average MAX Response Time for ALL TEST scenarios = ", Average_Test_Stats[1])
print("Average Response Time for ALL TEST scenarios = ", Average_Test_Stats[2])
# print("Average MIN Response Time for ALL TEST scenarios = ", Average_Test_Stats[3])
print("Max % scenarios covered by any engine company = ", Average_Test_Stats[4])
print("MIN % scenarios covered by any engine company = ", Average_Test_Stats[5])

# BEGIN GENETIC ALGORITHM FOR CONSTRAINT SATISFACTION PROBLEM
if(DO_GA == 0):
    print('QUITTING WITHOUT GA...')
    exit()

NUM_ECS = len(Pithy_Train_List)
gen_pop_list = Create_Population_For_GA()
gen_pop_list = Update_Num_Infeasible_Fires(gen_pop_list, Pithy_Train_List, Test_List)
gen_pop_list = Update_Engine_Costs(gen_pop_list)

for i in range(NUM_CROSSOVERS):
    gen_pop_list = Perform_Crossover(gen_pop_list)
    print('CROSSOVER NUMER ', i+1, ' COMPLETE')

for i in range(NUM_MUTATIONS):
    gen_pop_list = Perform_Mutations(gen_pop_list)
    print('MUTATION NUMER ', i+1, ' COMPLETE')


gen_pop_list = Update_Num_Infeasible_Fires(gen_pop_list, Pithy_Train_List, Test_List)
gen_pop_list = Update_Engine_Costs(gen_pop_list)


tot_feas_sols = 0
best_cost_index = 1000
best_infeas_prop = 100
best_cost_pop_mem = 0
best_infeas_pop_mem = 0
f7 = open('Pareto_Sols.txt','w')
for i in range(len(gen_pop_list)):


    cur_pop_mem = gen_pop_list[i]
    pop_mem = cur_pop_mem[0]
    eng_list = cur_pop_mem[1]
    prop_infeas_fires = cur_pop_mem[2]
    pcost = cur_pop_mem[3]
    lcost = cur_pop_mem[4]
    tot_cost = pcost + lcost
    base_cost = NUM_ECS*PUMPER_COST
    cost_index = float((tot_cost - base_cost)*100)/float(base_cost)
    print("prop_infeas_fires = ", prop_infeas_fires, "cost_index = ", cost_index, COST_INDEX_LT,
PROP_INFEAS_LT)
    if((cost_index <= COST_INDEX_LT) and (prop_infeas_fires <= PROP_INFEAS_LT)):
        print('FOUND FEASIBLE POPULATION MEMBER..')
        print('POP_MEMBER = ', i)
        print('cost_index = ', cost_index)
        print('prop_infeas_fires = ', prop_infeas_fires)
        strcost = str(cost_index)
        strinfeas = str(prop_infeas_fires)
        f7.write(strcost + ' ,   ' + strinfeas + '\n')
        tot_feas_sols = tot_feas_sols + 1
        if(cost_index < best_cost_index):
            best_cost_index = cost_index
            best_cost_pop_mem = i
        if(prop_infeas_fires < best_infeas_prop):
            best_infeas_prop = prop_infeas_fires
            best_infeas_pop_mem = i

print('BEST COST SOLUTION = ', best_cost_index)
print('BEST INFEAS SOLUTION = ', best_infeas_prop)
f7.close()




    #kk = len(eng_list)
    #for k in range(kk):
    #   cur_ec = eng_list[k]
    #   eng_com_id = cur_ec[0]
    #   pumper = cur_ec[1][0]
    #   ladder = cur_ec[1][1]
    #   print("POP ", pop_mem, "eng_com = ", eng_com_id, "PUMP = ", pumper, "LAD = ",
ladder)
    #   print("prop_infeas_fires = ", prop_infeas_fires, "pumper cost = ", pcost, "ladder cost = ",
lcost)
```