APPENDIX

Mobility study based on the analysis of Bluetooth data in the city of Brisbane

Authors: María Sarrá Paloma and Ana Valero Serratosa

Supervisor: Mehmet Yildirimoglu

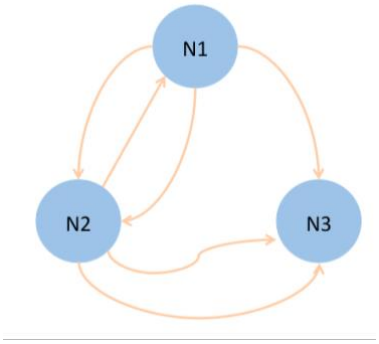
Co-supervisor: Tania Santos

Academic year: 2018-2019

Barcelona School of industrial Engineering

Master of industrial Engineering

1. Example of how the OD matrix dictionary is built



With N1, N2 and N3 being the nodes, each arrow representing one trip in the database, and its origin and destination matching the arrow ends (fig.TTT), the resulting dictionary would be the following:

OD_matrix = {N1: {N1: 0, N2: 3, N3: 1}, N2: {N1: 3, N2: 0, N3: 2}, N3: {N1: 1, N2: 2, N3: 0}}

As observed, it is not being specified in the dictionary if the trip is one way or the other. This is because this information is not relevant for the model proposed.

2. Effect of design parameters in routes obtained

2.1. Trajectory of the routes obtained regarding the population size

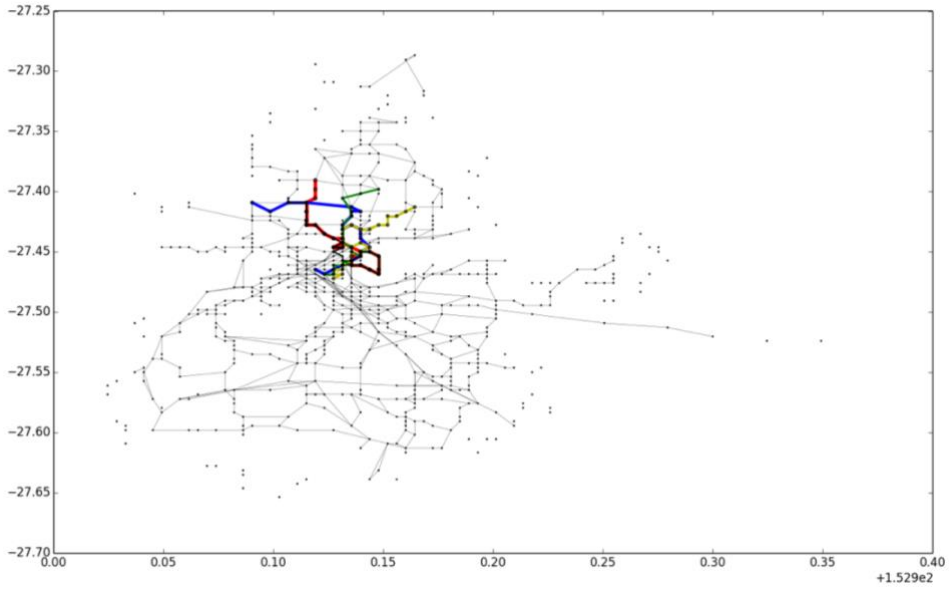


Figure 46 Trajectory of the routes running genetic algorithm with a population size of 100

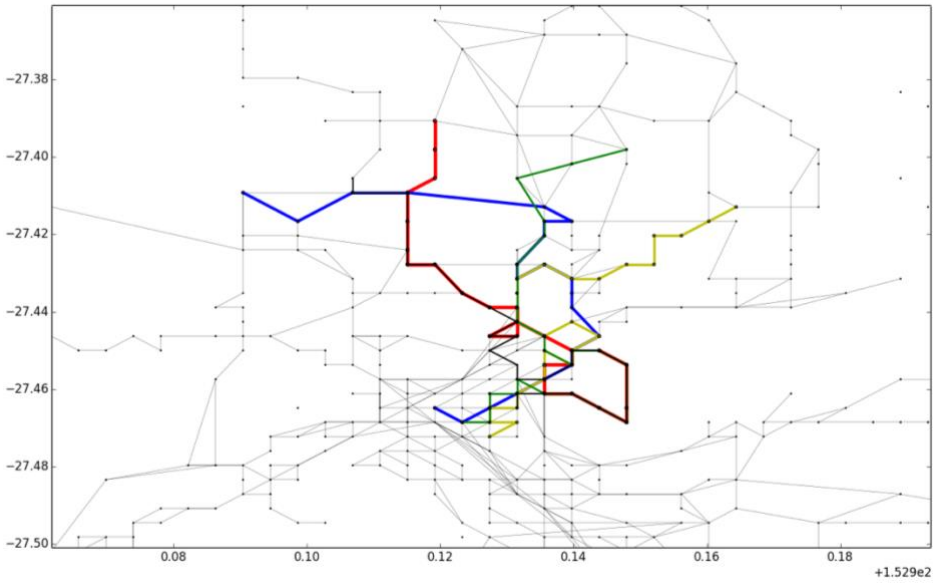


Figure 47 Zoomed trajectory of the routes running genetic algorithm with a population size of 100

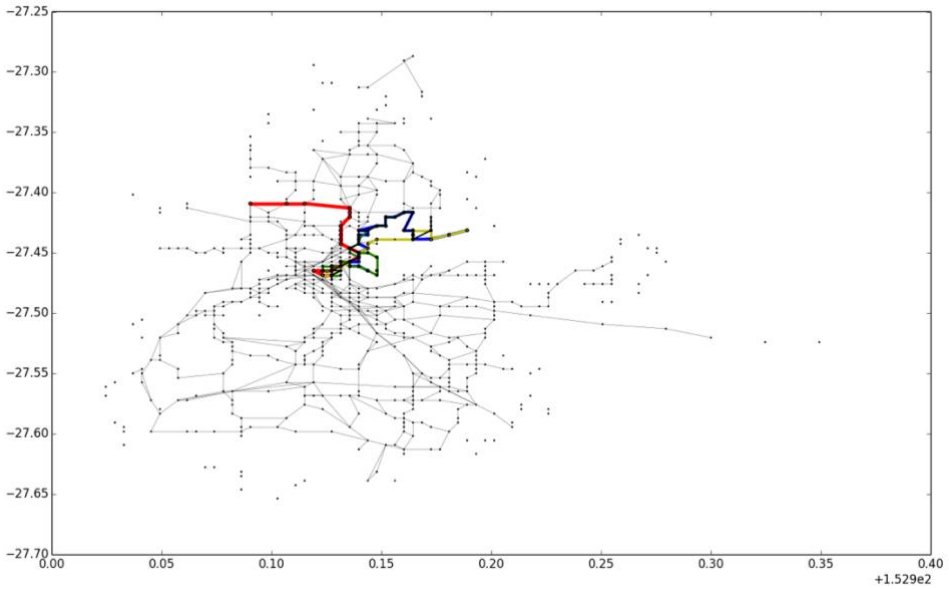


Figure 48 Trajectory of the routes running genetic algorithm with a population size of 1000

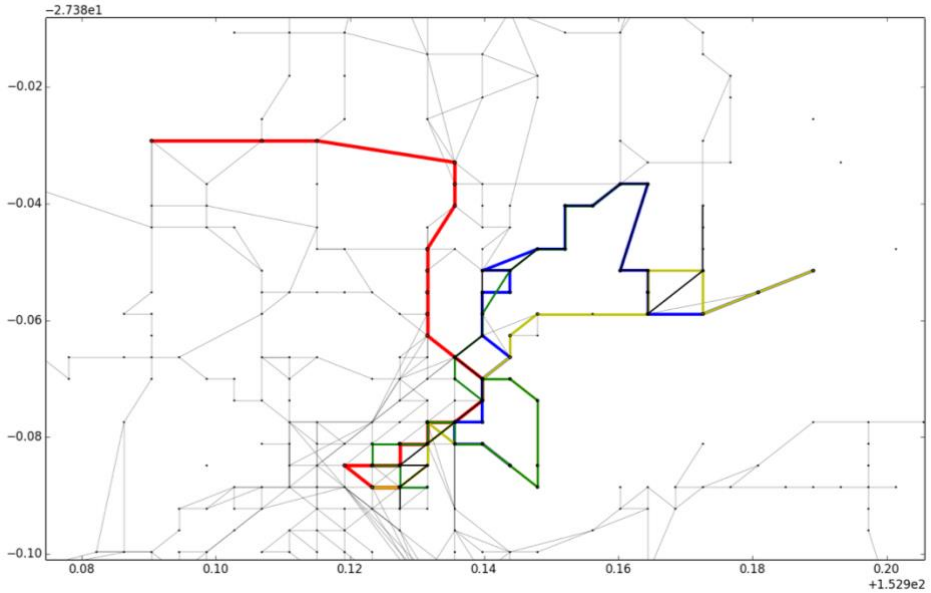


Figure 19 Zoomed trajectory of the routes running genetic algorithm with a population size of 1000

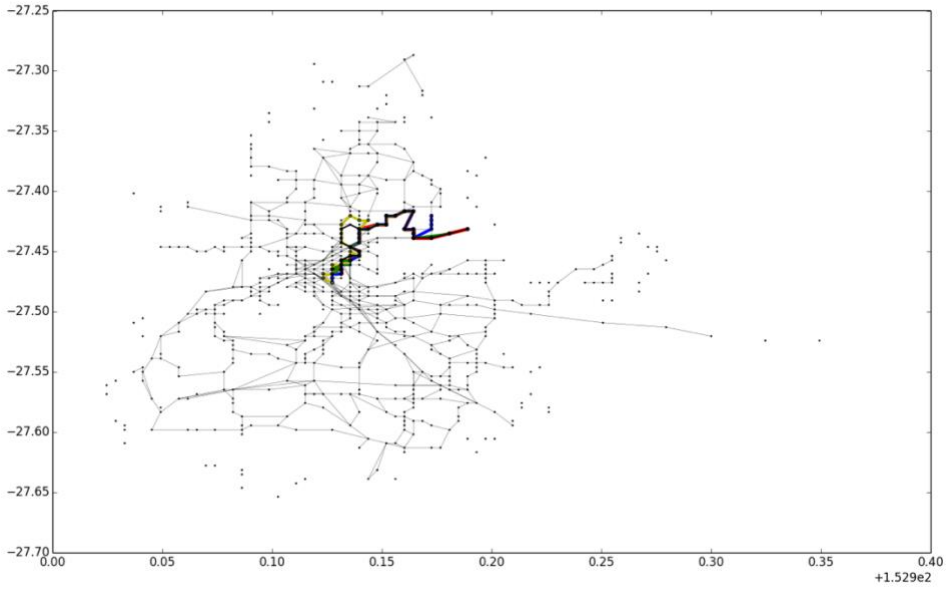


Figure 20 Trajectory of the routes running genetic algorithm with a population size of 3000

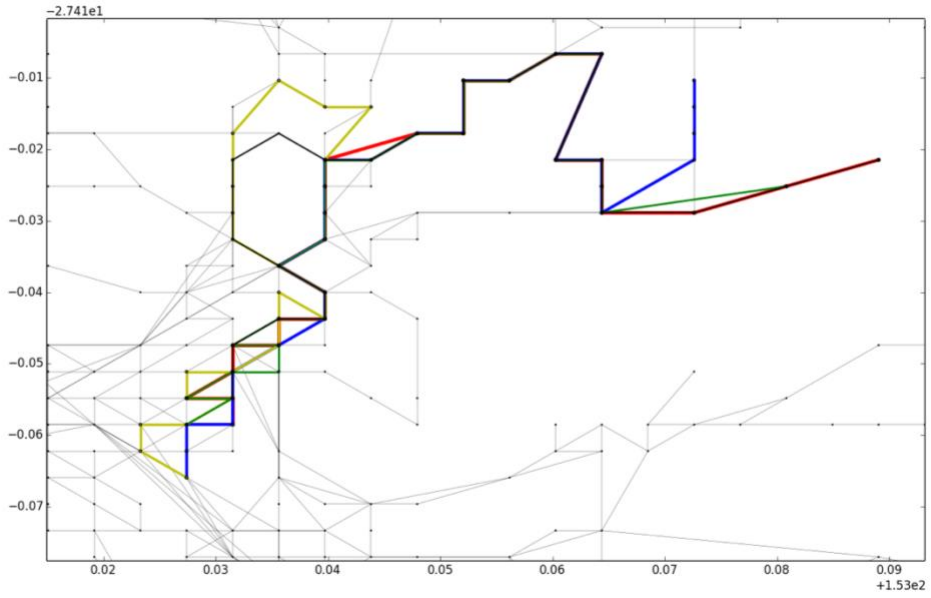


Figure 51 Zoomed trajectory of the routes running genetic algorithm with a population size of 3000

2.2. Trajectory of the routes obtained regarding the mutation rate

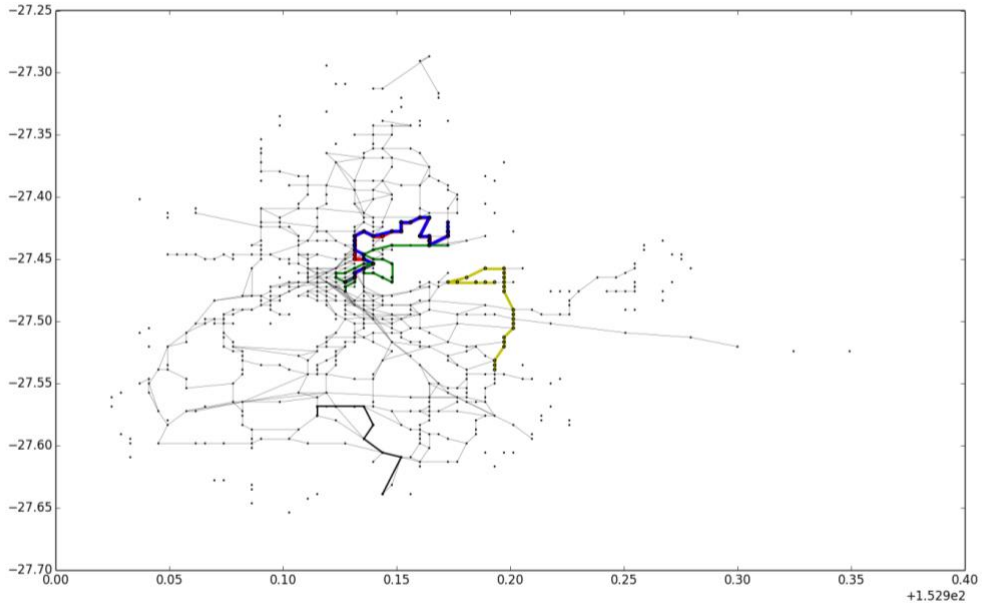


Figure 52 Trajectory of the routes running genetic algorithm with a mutation rate of 0,2

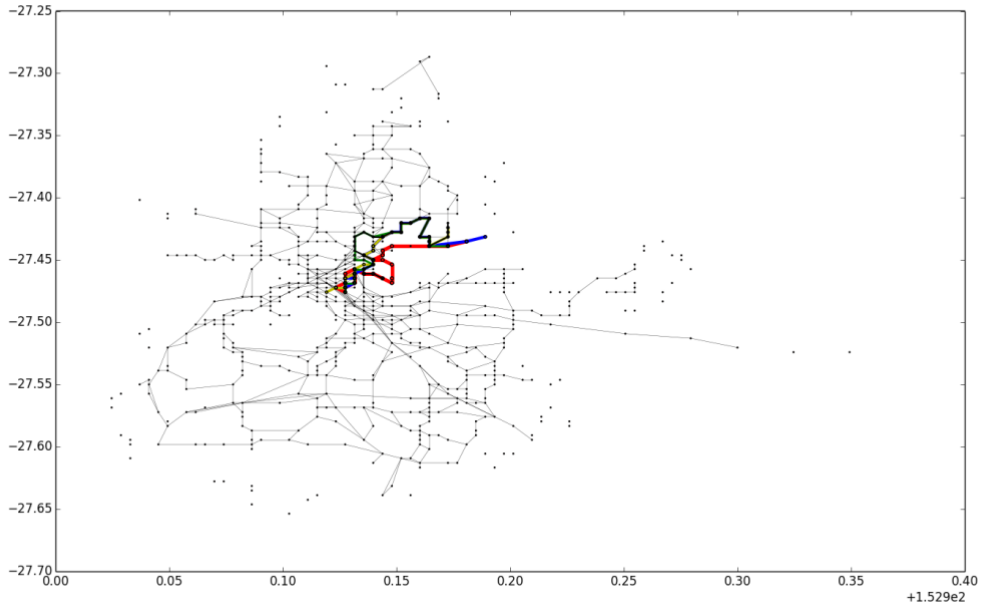


Figure 53 Trajectory of the routes running genetic algorithm with a mutation rate of 0,6

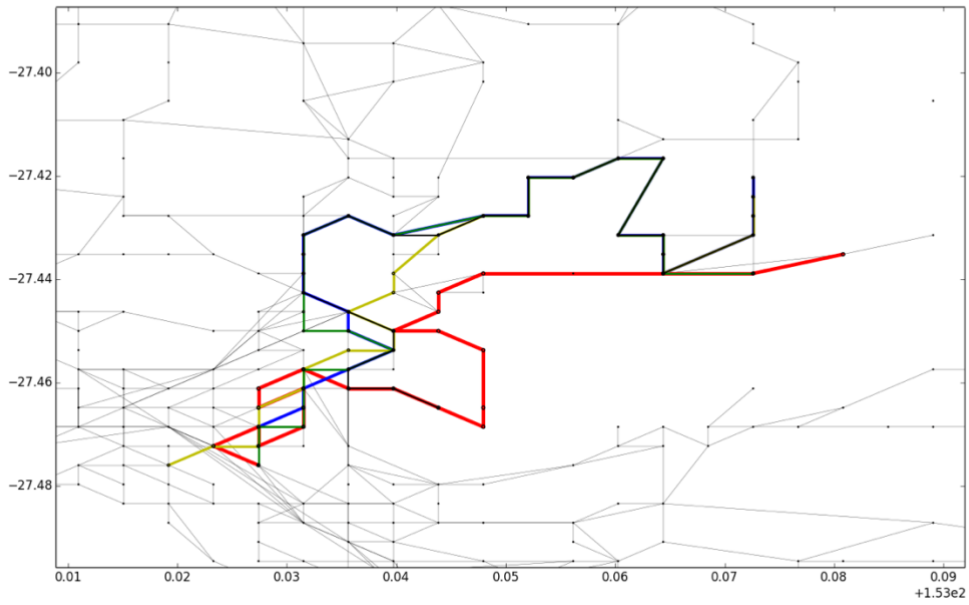


Figure 54 Zoomed trajectory of the routes running genetic algorithm with a mutation rate of 0,6

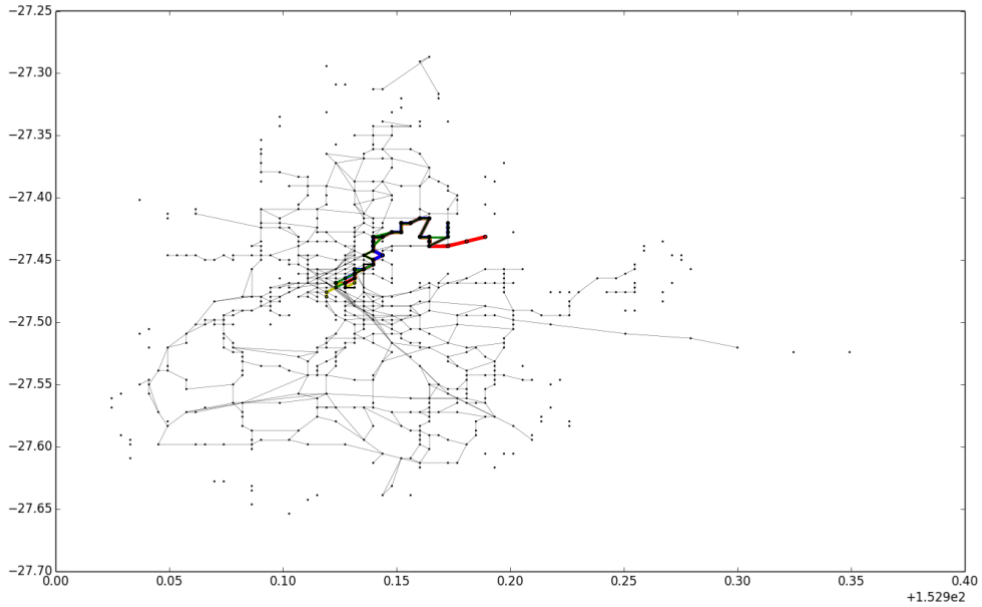


Figure 55 Trajectory of the routes running genetic algorithm with a mutation rate of 1

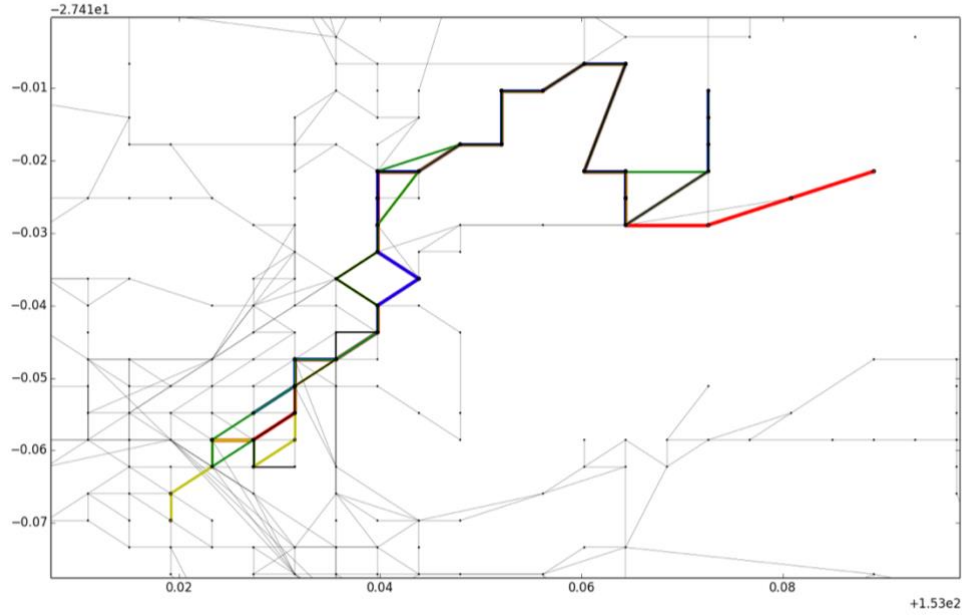


Figure 56 Zoomed trajectory of the routes running genetic algorithm with a mutation rate of 0,6

2.3. Trajectory of the routes obtained regarding the selected candidates

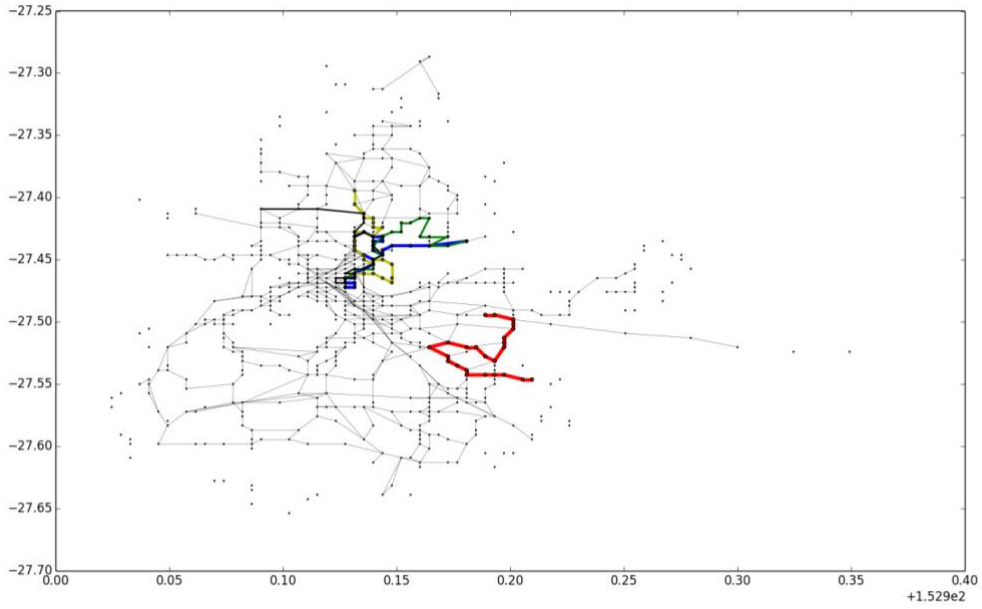


Figure 57 Trajectory of the routes running genetic algorithm with a 25 selected candidates

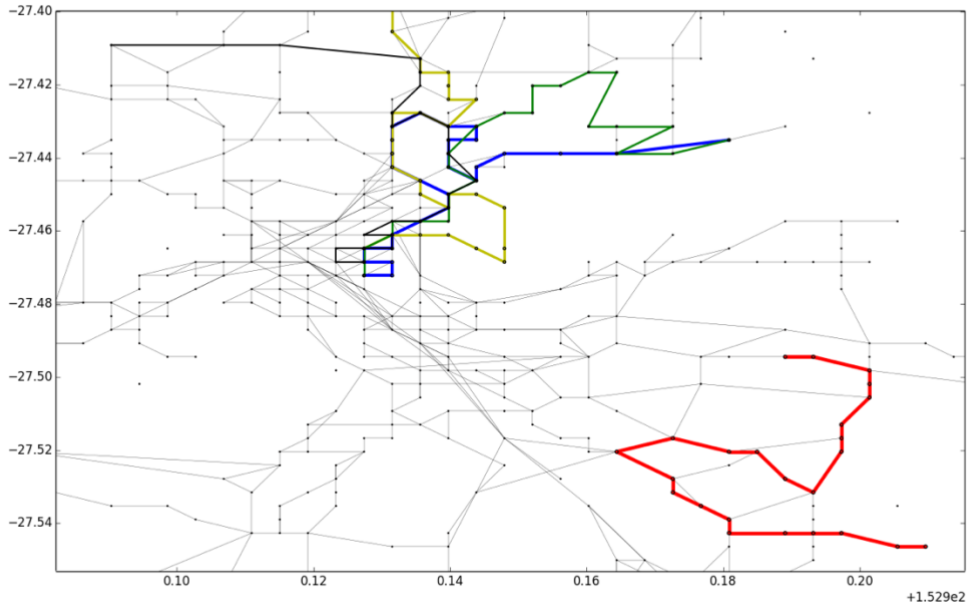


Figure 58 Zoomed trajectory of the routes running genetic algorithm with 25 selected candidates

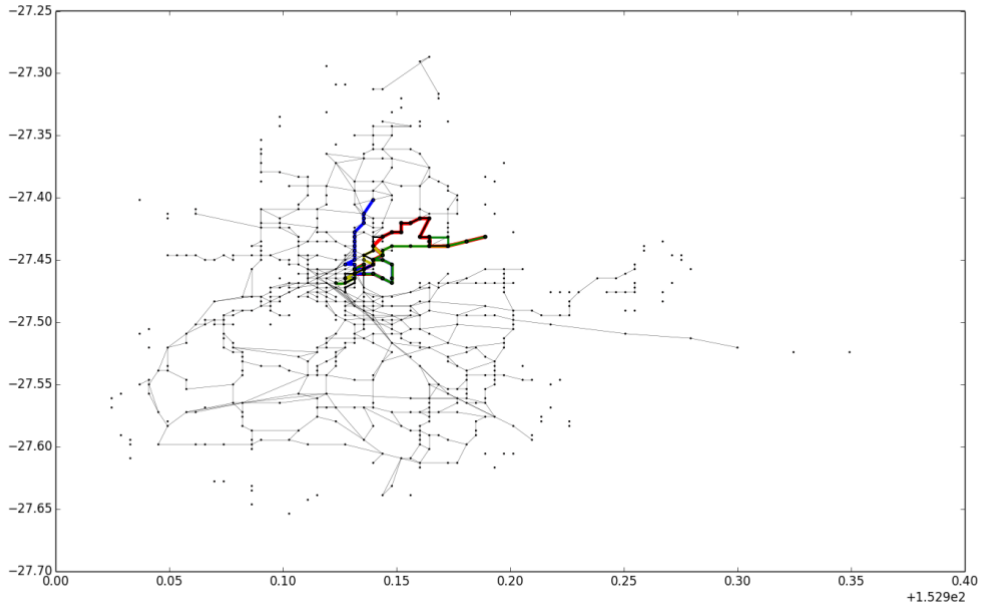


Figure 59 Trajectory of the routes running genetic algorithm with 75 selected candidates

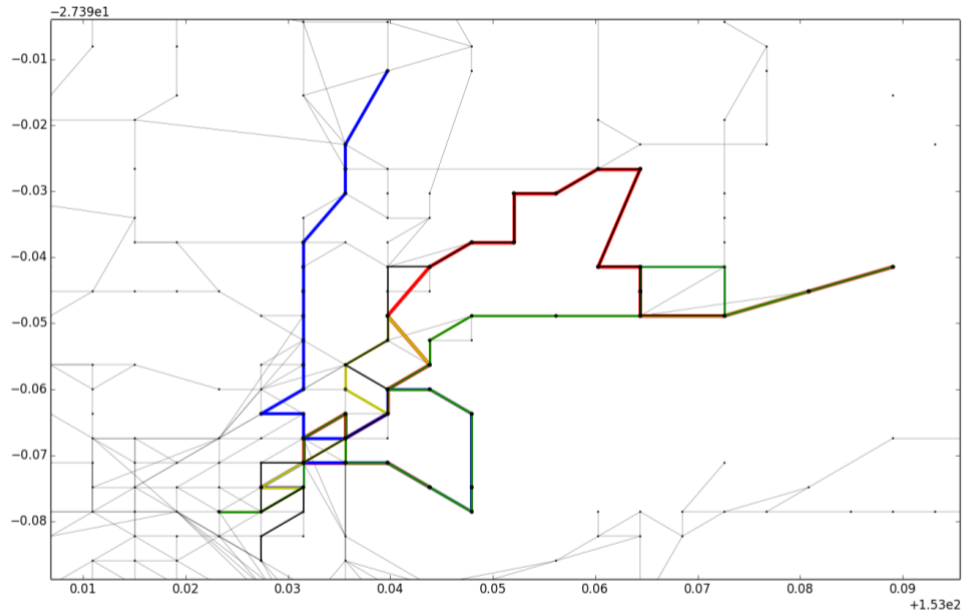


Figure 60 Zoomed trajectory of the routes running genetic algorithm with 75 selected candidates

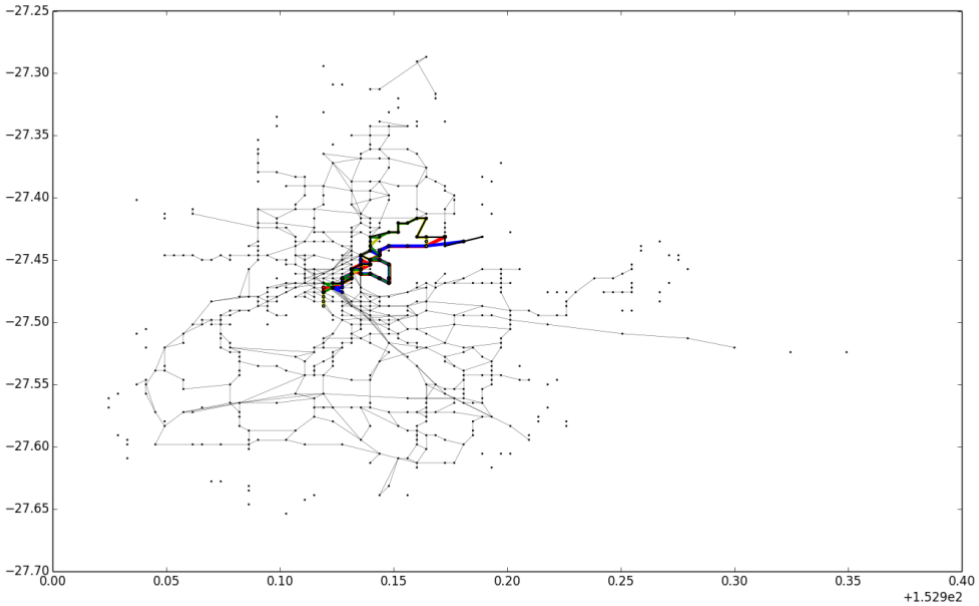


Figure 61 Trajectory of the routes running genetic algorithm with 125 selected candidates

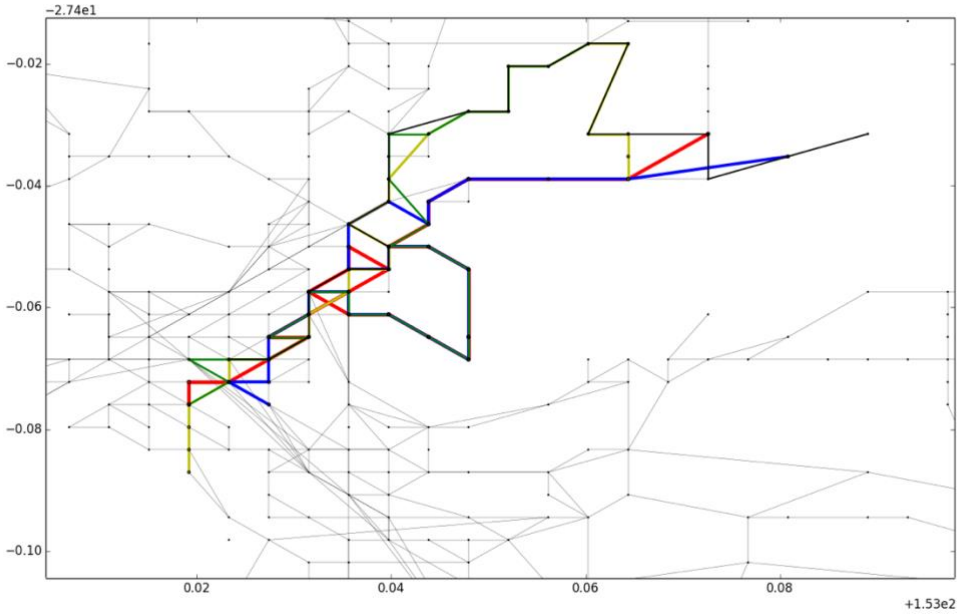


Figure 62 Zoomed trajectory of the routes running genetic algorithm with 75 selected candidates

3. Code of the algorithm

```
import csv
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import xlsxwriter
import math
import numpy
import random
import inspyred
import time
import copy
import operator
from inspyred import ec


start = time.time()
global nomfitxer
global data
nomfitxer = '2016-06-01.csv'
data = '2016-06-01'

#_____funcion_distancia_entre_nodos_____
def haversine(lat1,lon1,lat2,lon2):
    rad = math.pi/180
```

```
    dlat = lat2 - lat1

    dlon = lon2 - lon1

    R = 6372.795477598

    a = (math.sin(rad*dlat/2))**2 +
math.cos(rad*lat1)*math.cos(rad*lat2)*(math.sin(rad*dlon/2))**2

    distancia = 2*R*math.asin(math.sqrt(a))

    return distancia


#_____funcion_definir_dicc_iniciales_____

def arregla_csv():

#funcion que devuelve dicc_nodos = dicc con todos los sensores que hay en la base
de datos en la fecha y intervalo horario seleccionado

#dicc_nodos = {'sensor1':(lon,lat),'sensor2':(lon,lat)}

#dicc_enlaces                                                                    =
{'coche1':[sensorA,sensorB,sensorC],'coche2':[sensorC,sensorA,sensorF,sensorR]} -
-> recorrido de cada coche por los sensores

    dicc_nodos = {}

    dicc_enlaces_p = {}

    with open(nomfitxer,'r') as inp:

        for linea in inp:

            linea = linea.strip()

            ID_dispo,dia_hora,dur,ID_sensor,lat,longitud,owner = linea.split(',')

            if ID_dispo == 'deviceid':

                pass

            else:

                dia,hora = dia_hora.split(' ')

                if float(dur) < 500 and int(hora[0:2]) in [6,7,8]:
```

```
                if ID_sensor not in dicc_nodos:
                    dicc_nodos[ID_sensor] = (longitud, lat)
                if ID_dispo in dicc_enlaces_p:
                    if ID_sensor not in dicc_enlaces_p[ID_dispo]:
                        dicc_enlaces_p[ID_dispo].append(ID_sensor)
                if ID_dispo not in dicc_enlaces_p:
                    dicc_enlaces_p[ID_dispo] = [ID_sensor]
            if int(hora[0:2])>8:
                break
    return dicc_nodos, dicc_enlaces_p


global dicc_nodos
global dicc_enlaces_p
dicc_nodos, dicc_enlaces_p = arregla_csv()
global G

def analitza_dicc_enlaces():
#funcion que devuelve d_ini_fin y d_num
#d_ini_fin = {'coche1':(sensor_ini1,sensor_fin1),'coche2': (sensor_ini2,sensor_fin2)}
#d_ini_fin es diccionario con el punto inicial y final de la ruta de cada dispositivo
    lat_min = 90
    lat_max = -90
    long_min = 180
    long_max = -180
```

```
for sensor in dicc_nodos:
    lat, lon = float(dicc_nodos[sensor][1]),float(dicc_nodos[sensor][0])
    if lat>lat_max:
        lat_max = lat
    if lat<lat_min:
        lat_min = lat
    if lon>long_max:
        long_max = lon
    if lon<long_min:
        long_min = lon
inc = (lat_max - lat_min)/(50)
y = numpy.arange(lat_min,lat_max,inc)
lista_frontera = []
for i in y:
    min_aux = 180
    max_aux = -180
    sens_max = ''
    sens_min = ''
    for sensor in dicc_nodos:
        lat, lon = float(dicc_nodos[sensor][1]),float(dicc_nodos[sensor][0])
        if i<=lat and lat <= i+inc :
            if lon>max_aux:
                max_aux = lon
                sens_max = sensor
            if lon<min_aux:
```

```
                min_aux = lon
                sens_min = sensor
        if sens_max != '' and sens_min != '':
            if sens_max not in lista_frontera:
                lista_frontera.append(sens_max)
                if sens_min not in lista_frontera:
                    lista_frontera.append(sens_min)
    dicc_enlaces = {}
    for dispo in dicc_enlaces_p:
        i = 0
        m = ''
        if (dicc_enlaces_p[dispo][0] not in lista_frontera) and (dicc_enlaces_p[dispo][len(dicc_enlaces_p[dispo])-1] not in lista_frontera):
            while i<(len(dicc_enlaces_p[dispo])-1):
                sens1 = dicc_enlaces_p[dispo][i]
                sens2 = dicc_enlaces_p[dispo][i+1]
                lat1,lon1 = float(dicc_nodos[sens1][1]),float(dicc_nodos[sens1][0])
                lat2,lon2 = float(dicc_nodos[sens2][1]),float(dicc_nodos[sens2][0])
                dist = haversine(lat1,lon1,lat2,lon2)
                if dist > 4:
                    m = 'eliminar ruta'
                    break
                i = i+1
            if len(dicc_enlaces_p[dispo])>2:
                if m!= 'eliminar ruta':
                    dicc_enlaces[dispo] = dicc_enlaces_p[dispo]
```

```
    d_num = {}
    d_ini_fin = {}
    for e in dicc_enlaces:
        d_ini_fin[e] = (dicc_enlaces[e][0],dicc_enlaces[e][len(dicc_enlaces[e])-1])
        i = 0
        while i< (len(dicc_enlaces[e])-1):
            if (dicc_enlaces[e][i],dicc_enlaces[e][i+1]) in d_num:
                d_num[(dicc_enlaces[e][i],dicc_enlaces[e][i+1])] =
d_num[(dicc_enlaces[e][i],dicc_enlaces[e][i+1])] + 1
            elif (dicc_enlaces[e][i+1],dicc_enlaces[e][i]) in d_num:
                d_num[(dicc_enlaces[e][i+1],dicc_enlaces[e][i])] =
d_num[(dicc_enlaces[e][i+1],dicc_enlaces[e][i])] + 1
            else:
                d_num[(dicc_enlaces[e][i],dicc_enlaces[e][i+1])] = 1
            i = i + 1
    return dicc_enlaces, d_ini_fin, d_num


#_____________________funcion_agrupacion_de_nodos_____________________


global dicc_grupos
global dicc_coord_grup
global dicc_rel
global dicc_enlaces_grup
global d_ini_fin_grup
global d_num_grup
```

```
def agrup_nodos():

#agrupa los nodos segun proximidad

#dicc_grupos = {'grupo1':['s1','s2','s3'],'grupo2':['s7','s10','s22']} --> contiene los ID de los sensores de cada grupo

#dicc_coord_grup = {'grupo1': (lat,lon),'grupo2':(lat2,lon2)} --> contiene las coordenadas del primer sensor de cada grupo (para tener una referencia)

#dicc_rel = {'sensor1':'grupo1','sensor2':'grupo2'...}

    lat_min = 90

    lat_max = -90

    long_min = 180

    long_max = -180

    for sensor in dicc_nodos:

        lat, lon = float(dicc_nodos[sensor][1]),float(dicc_nodos[sensor][0])

        if lat>lat_max:

            lat_max = lat

        if lat<lat_min:

            lat_min = lat

        if lon>long_max:

            long_max = lon

        if lon<long_min:

            long_min = lon

    lat_aux = lat_min

    long_aux = long_min

    inc_lat = (lat_max - lat_min)/100

    inc_long = (long_max - long_min)/80

    sens_aux = 1
```

```
    dicc_grupos = {}
    dicc_coord_grup = {}
    dicc_rel = {}
    while lat_aux < lat_max:
        while long_aux < long_max:
            for sensor in dicc_nodos:
                lat, lon = float(dicc_nodos[sensor][1]),float(dicc_nodos[sensor][0])
                if lat_aux<=lat and lat<(lat_aux+inc_lat) and long_aux<=lon and
lon<(long_aux+inc_long):
                    if sens_aux in dicc_grupos:
                        dicc_grupos[sens_aux].append(sensor)
                    if sens_aux not in dicc_grupos:
                        dicc_grupos[sens_aux] = [sensor]
                    if sens_aux not in dicc_coord_grup:
                        lat_med = (lat_aux+lat_aux+inc_lat)/2
                        lon_med = (long_aux+long_aux+inc_long)/2
                        dicc_coord_grup[sens_aux] = (lon_med,lat_med)
                    if sensor not in dicc_rel:
                        dicc_rel[sensor] = sens_aux
            long_aux = long_aux + inc_long
            sens_aux = sens_aux + 1
        lat_aux = lat_aux + inc_lat
        long_aux = long_min
        sens_aux = sens_aux + 1
    return dicc_grupos, dicc_coord_grup, dicc_rel
```

```
dicc_grupos, dicc_coord_grup, dicc_rel = agrup_nodos()


def pasar_a_grup():
    dicc_enlaces, d_ini_fin, d_num = analitza_dicc_enlaces()
    dicc_enlaces_grup = {}
    d_ini_fin_grup = {}
    d_num_grup = {}
    for dispo in dicc_enlaces:
        dicc_enlaces_grup[dispo] = []
        i = 0
        while i <= (len(dicc_enlaces[dispo])-1):
            dicc_enlaces_grup[dispo].append(dicc_rel[dicc_enlaces[dispo][i]])
            i = i + 1


    for e in dicc_enlaces_grup:
        d_ini_fin_grup[e] =
(dicc_enlaces_grup[e][0],dicc_enlaces_grup[e][len(dicc_enlaces_grup[e])-1])
        i = 0
        while i< (len(dicc_enlaces_grup[e])-1):
            if dicc_enlaces_grup[e][i]==dicc_enlaces_grup[e][i+1]:
                pass
            elif (dicc_enlaces_grup[e][i],dicc_enlaces_grup[e][i+1]) in d_num_grup:
                d_num_grup[(dicc_enlaces_grup[e][i],dicc_enlaces_grup[e][i+1])] =
d_num_grup[(dicc_enlaces_grup[e][i],dicc_enlaces_grup[e][i+1])] + 1
            elif (dicc_enlaces_grup[e][i+1],dicc_enlaces_grup[e][i]) in d_num_grup:
                d_num_grup[(dicc_enlaces_grup[e][i+1],dicc_enlaces_grup[e][i])] =
d_num_grup[(dicc_enlaces_grup[e][i+1],dicc_enlaces_grup[e][i])] + 1
```

```
        else:
            d_num_grup[(dicc_enlaces_grup[e][i],dicc_enlaces_grup[e][i+1])] = 1
        i = i + 1

    return dicc_enlaces_grup, d_ini_fin_grup, d_num_grup

dicc_enlaces_grup, d_ini_fin_grup, d_num_grup = pasar_a_grup()

#_____________________________matriz_OD_____________________

global OD_matrix

def premat_OD():
#funcion intermedia para generar d_preOD
#d_preOD = {'(sensorA,sensorB)':3,'(sensorF,sensorR)':2}
#d_preOD tiene como claves los sensores iniciales y finales que hay en d_ini_fin y como valores el numero de dispositivos que tienen como origen y final esos sensores
    d_preOD = {}
    for dispo in d_ini_fin_grup:
        if d_ini_fin_grup[dispo] not in d_preOD:
            d_preOD[d_ini_fin_grup[dispo]] = 1
        else:
            d_preOD[d_ini_fin_grup[dispo]] = d_preOD[d_ini_fin_grup[dispo]] + 1
    return d_preOD

def mat_OD():
```

```
#funcion que genera la matriz origen destino

#distance_matrix = {'sensorA':{sensorA:0, sensorB:2,sensorC:3}, 'sensorB':{sensorA:2, sensorB:0, sensorC:1}, 'sensorC':{sensorA:3, sensorB:1, sensorC:0}}

#distance_matrix tiene info sobre desde cada sensor cuantos dispositivos se desplazan a cada sensor

    #d_ini_fin, d_num = analitza_dicc_enlaces()

    d_preOD = premat_OD()

    OD_matrix = {}

    for sensor1 in dicc_coord_grup:

        OD_matrix[sensor1] = {}

        for sensor2 in dicc_coord_grup:

            if sensor1 == sensor2:

                OD_matrix[sensor1][sensor2] = 0

            elif ((sensor1,sensor2) in d_preOD) and ((sensor2,sensor1) in d_preOD):

                OD_matrix[sensor1][sensor2] = d_preOD[(sensor1,sensor2)] + d_preOD[(sensor2,sensor1)]

            elif (sensor1,sensor2) in d_preOD:

                OD_matrix[sensor1][sensor2] = d_preOD[(sensor1,sensor2)]

            elif (sensor2,sensor1) in d_preOD:

                OD_matrix[sensor1][sensor2] = d_preOD[(sensor2,sensor1)]

            else:

                OD_matrix[sensor1][sensor2] = 0

    return OD_matrix


OD_matrix = mat_OD()
```

```
#_____________________funcion_genera_grafo_____________________

def graf_grup():

#funcion que genera un grafo con la ruta de cada dispositivo a traves de los distintos grupos de sensores

    G = nx.Graph()

    for group in dicc_coord_grup:

        G.add_node(group, loc = (dicc_coord_grup[group][0],dicc_coord_grup[group][1]))

    for enlace in d_num_grup:

        lat1,lon1,lat2,lon2 = dicc_coord_grup[enlace[0]][1],dicc_coord_grup[enlace[0]][0],dicc_coord_grup[enlace[1]][1],dicc_coord_grup[enlace[1]][0]

        dist = haversine(lat1,lon1,lat2,lon2)

        if dist < 4:

            if d_num_grup[enlace]>1000 :

                G.add_edge(enlace[0], enlace[1], num = d_num_grup[enlace], length = dist,color = 'b')

            if d_num_grup[enlace]>50 and d_num_grup[enlace]<1000 :

                G.add_edge(enlace[0], enlace[1], num = d_num_grup[enlace], length = dist,color = 'r')

    F = G.degree()

    l = []

    for nodo in G.nodes():

        if F[nodo]==1:

            l.append(nodo)

    G.remove_nodes_from(l)

    edges = G.edges()

    colors = [G[u][v]['color'] for u,v in edges]
```

```
    return G


#_____________________funcion_generacion_de_poblacion_inicial__________
G = graf_grup()
def generate_routes(random,args):
    while True:
        try:
            sensor1 = random.choice(list(G.nodes()))
            sensor2 = random.choice(list(G.nodes()))
            ind = nx.dijkstra_path(G,sensor1,sensor2,weight='length')
            length = round(nx.dijkstra_path_length(G,sensor1,sensor2, weight='length'),
2)
            if length>7 and length<9 :
                break
        except:
            pass
    return [ind]



#_____________________funcion_de_evaluacion_____________________________
def evaluate_routes(candidates,args):
    fitness = []
    for route in candidates:
        pathDemand = 0
        route = route[0]
        i=0
```

```
        route_copy = copy.copy(route)
        while i<(len(route_copy)):
            for e in route_copy:
                pathDemand = pathDemand + int(OD_matrix[route_copy[0]][e])
            route_copy.pop(0)
        fitness.append(pathDemand)
    return fitness


#_____________________funcion_de_mutacion_____________________________
def route_mutator(random,candidates,args):
    children = inspyred.ec.variators.default_variation(random,candidates,args)
    offspring = []
    for ind,l in enumerate(children):
        lista,prob = l
        mut_prob = random.random()
        if mut_prob < 0.6:
            while True:
                j = 0
                try:
                    if j>30:
                        break
                    H = G.copy()
                    lista_copy2 = copy.copy(lista)
                    sensor1 = int(random.choice(lista_copy2))
                    pos_sens1 = lista_copy2.index(sensor1)
```

```
l_sin_sens1 = lista_copy2[:pos_sens1]+lista_copy2[(pos_sens1+1):]
sensor2 = int(random.choice(l_sin_sens1))
pos_sens2 = lista_copy2.index(sensor2)
if pos_sens1<pos_sens2:
    part1 = lista_copy2[:pos_sens1]
    part2 = lista_copy2[(pos_sens2):]
    finalsens1 = sensor1
    pos1 = pos_sens1
    initialsens2 = sensor2
    pos2 = pos_sens2
if pos_sens2<pos_sens1:
    part1 = lista_copy2[:pos_sens2]
    part2 = lista_copy2[(pos_sens1):]
    finalsens1 = sensor2
    pos1 = pos_sens2
    initialsens2 = sensor1
    pos2 = pos_sens1
if pos_sens2-pos_sens1==1 or pos_sens2-pos_sens1==-1:
    H.remove_edge(sensor1,sensor2)
nodes_to_remove =
lista_copy2[:pos1]+lista_copy2[(pos1+1):pos2]+lista_copy2[(pos2+1):]
H.remove_nodes_from(nodes_to_remove)
try:
    part1_2 = nx.dijkstra_path(H,finalsens1,initialsens2)
    new_route = part1[:]+part1_2[:]+part2[1:]
    route = nx.DiGraph()
```

```
                    inisens1 = new_route[0]
                    part_med = new_route[1:]
                    route.add_node(inisens1, loc = G.node[inisens1]['loc'])
                    for sens in part_med:
                        route.add_node(sens, loc = G.node[sens]['loc'])
                        route.add_edge(inisens1, sens, num = G[inisens1][sens]['num'], length = G[inisens1][sens]['length'] ,color = G[inisens1][sens]['color'])
                        inisens1 = sens
                    if new_route in children or new_route in candidates:
                        raise NameError('La ruta ya existe')
                    length = route.size('length')
                    if length<15:
                        ra = random.random()
                        offspring.append([new_route,ra])
                        break
                    else:
                        raise NameError('Ruta demasiado larga')
                except:
                    pass
            except:
                j = j+1
                pass
        else:
            pass
    return offspring
```

```
#_____________________Genetic_Algorithm_____________________

rand = random.Random()

rand.seed(int(time.time()))

es = ec.ES(rand)

es.observer =
[inspyred.ec.observers.stats_observer,inspyred.ec.observers.file_observer]

es.selector = inspyred.ec.selectors.fitness_proportionate_selection

es.replacer = inspyred.ec.replacers.plus_replacement

es.variator = route_mutator

es.terminator = inspyred.ec.terminators.generation_termination


final_pop = es.evolve(generator=generate_routes, num_selected=50,
evaluator=evaluate_routes, pop_size=1000, maximize=True, max_generations=20)


final_pop.sort(reverse=True)

print 'final_pop: ' , final_pop

end = time.time()

print 'tiempo: ', (end-start)
```