```
# -*- coding: utf-8 -*-
"""

@author: Celine Nardi
"""

from __future__ import print_function
import pandas as pd
from pandas import ExcelWriter
import networkx as nx
import os
import random as rd
os.chdir('C:/Users/celic/Documents/_UCLouvain/silk road/gurobi')
import gurobipy as gp
from gurobipy import GRB
import csv
from pandas import DataFrame
import math
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
import matplotlib.transforms as mtransforms
scatall= ''
road="road"
maritime="maritime"
rail="rail"
#%%function
def scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,
    fig = plt.figure()
    ax = fig.add_subplot()
    plt.scatter(line1x, line1y, color='green', alpha=0.5, label=labels[2])
    plt.scatter(linenormalx[D], linenormaly[D], color='red', alpha=0.5, label=labels[1])
    plt.scatter(line2x, line2y, color='blue', alpha=0.5, label=labels[0])
    plt.scatter(ut1x, ut1y, color='green', alpha=0.5, marker='x')
    plt.scatter(na1x, na1y, color='green', alpha=0.5, marker='v')
    plt.scatter(utopianorm[D][0],utopianorm[D][1], color='red', alpha=0.5, marker='x')
    plt.scatter(nadirnorma[D][0], nadirnorma[D][1], color='red', alpha=0.5, marker='v')
    plt.scatter(ut2x, ut2y, color='blue', alpha=0.5, marker='x')
    plt.scatter(na2x, na2y, color='blue', alpha=0.5, marker='v')
    plt.plot(line1x, line1y, color='green',linestyle='solid',  alpha=0.5)
    plt.plot(linenormalx[D], linenormaly[D], color='red',linestyle='solid',  alpha=0.5)
    plt.plot(line2x, line2y, color='blue',linestyle='solid',  alpha=0.5)


    plt.title('Pareto Frontier - '+scatall)
    plt.legend()
    plt.xlabel('Total Cost (obj1) [USD]')
    plt.ylabel('Total Time (obj2) [h]')
    #plt.plot([0, 1], [1, 0],color='black',linestyle='dashed',linewidth=1)
    plt.show()
    fig.savefig(scatall+'.png')
def scattersa2(line4x, line4y, ut4x, ut4y,na4x, na4y, line3x, line3y, ut3x, ut3y,na3x, na3y,
    fig = plt.figure()
    ax = fig.add_subplot()
    plt.scatter(line1x, line1y, color='green', alpha=0.5, label=labels[1])
    plt.scatter(linenormalx[D], linenormaly[D], color='red', alpha=0.5, label=labels[0])
    plt.scatter(line2x, line2y, color='blue', alpha=0.5, label=labels[2])
    plt.scatter(ut1x, ut1y, color='green', alpha=0.5, marker='x')
    plt.scatter(na1x, na1y, color='green', alpha=0.5, marker='v')
    plt.scatter(utopianorm[D][0],utopianorm[D][1], color='red', alpha=0.5, marker='x')
```

```
    plt.scatter(nadirnorma[D][0], nadirnorma[D][1], color='red', alpha=0.5, marker='v')
    plt.scatter(ut2x, ut2y, color='blue', alpha=0.5, marker='x')
    plt.scatter(na2x, na2y, color='blue', alpha=0.5, marker='v')

    plt.scatter(line3x, line3y, color='magenta', alpha=0.5, label=labels[3])
    plt.scatter(line4x, line4y, color='orange', alpha=0.5, label=labels[4])
    plt.scatter(ut3x, ut3y, color='magenta', alpha=0.5, marker='x')
    plt.scatter(na3x, na3y, color='magenta', alpha=0.5, marker='v')
    plt.scatter(ut4x, ut4y, color='orange', alpha=0.5, marker='x')
    plt.scatter(na4x, na4y, color='orange', alpha=0.5, marker='v')
    plt.plot(line1x, line1y, color='green',linestyle='solid',  alpha=0.5)
    plt.plot(linenormalx[D], linenormaly[D], color='red',linestyle='solid',  alpha=0.5)
    plt.plot(line2x, line2y, color='blue',linestyle='solid',  alpha=0.5)
    plt.plot(line3x, line3y, color='magenta',linestyle='solid',  alpha=0.5)
    plt.plot(line4x, line4y, color='orange',linestyle='solid',  alpha=0.5)


    plt.title('Pareto Frontier - '+scatall)
    plt.legend()
    plt.xlabel('Total Cost (obj1) [USD]')
    plt.ylabel('Total Time (obj2) [h]')
    #plt.plot([0, 1], [1, 0],color='black',linestyle='dashed',linewidth=1)
    plt.show()
    fig.savefig(scatall+'.png')
def run(name):
    """run a cell
    name= name of the cell"""
    runcell(name, 'C:/Users/celic/Documents/_UCLouvain/silk road/Gurobi/SilkRoad.V.2.0.py')

def getroute(donnees, routes, resnd, hubs={'1':0}, hub=['1', '2']):
    #donnees= countries[D]: list of list of results
    """return dicts proportion of route and mapping of the solutions(key)"""
    routeprop={}
    routeproptot={}
    routenet={}
    listcheck=list(resnd.keys())
    listcheck2=[]
    for names in listcheck:
        listcheck2.append(str(names) +'.xlsx')

    routekey=list(routes.keys())
    for method in donnees: #list method, x[], ... y[]
        for check in listcheck:
            if check+'.xlsx' in method[0]:
                routeprop[method[0]]=[] #TSR, LSR, SC, TC, EUR, rail, road, maritime
                routeprop[method[0]].append(str(method[0]))
                routenet[check]=[]
            #number of TSR

                countTSR=0
                countTC=0
                countLSR=0
                countSC=0
                countEUR=0
                countroutetotal=0

                countrail=0
                countroad=0
                countsea=0
                countmodetotal=0
```

```
                for arcr in routekey:
                    for ligne in method:
                        #print(ligne[0]+ligne[1])
                        if 'x[' in str(ligne[0]+ligne[1]):
                            if str(arcr[0])+','+(arcr[1]) in ligne:
                                countroutetotal=countroutetotal+1
                                countmodetotal=countmodetotal+1
                                #routenet[check].append(arcr)
                                if 'EUR' in routes[arcr]:
                                    countEUR=countEUR+1
                                if 'TSR' in routes[arcr]:
                                    countTSR=countTSR+1
                                if 'LSR' in routes[arcr]:
                                    countLSR=countLSR+1
                                if 'SC' in routes[arcr]:
                                    countSC=countSC+1
                                if 'TC' in routes[arcr]:
                                    countTC=countTC+1
                                if 'rail' in ligne:
                                    countrail=countrail+1
                                    routenet[check].append([arcr,'rail'])
                                if 'road' in ligne:
                                    countroad=countroad+1
                                    routenet[check].append([arcr,'road'])
                                if 'maritime' in ligne:
                                    countsea=countsea+1
                                    routenet[check].append([arcr,'maritime'])
                for ligne in method:
                        if 'y[' in str(ligne[0]+ligne[1]):
                            print(ligne)
                            for hb in hub:
                                if hb in ligne:
                                    print(hb)
                                    hubs[hb]=hubs[hb]+1
                routeprop[method[0]].append(countTSR/countroutetotal)
                routeprop[method[0]].append(countLSR/countroutetotal)
                routeprop[method[0]].append(countSC/countroutetotal)
                routeprop[method[0]].append(countTC/countroutetotal)
                routeprop[method[0]].append(countEUR/countroutetotal)
                routeprop[method[0]].append(countrail/countmodetotal)
                routeprop[method[0]].append(countroad/countmodetotal)
                routeprop[method[0]].append(countsea/countmodetotal)
                for ligne in method:
                    if 'y[' in str(ligne[0]+ligne[1]):
                            print(ligne)
                            for hb in hub:
                                if hb in ligne:
                                    print(hb)
                                    hubs[hb]=hubs[hb]+1



    return routeprop, routenet, hubs

def draw(routenetwork, name):
    """routenetwork= dic solution: mapping
    save mapping into a file"""
    for net in routenetwork.keys():
        edgecolor=[]
```

```
        edgelist={}
        datanet=routenetwork[net]
        network=nx.Graph()
        for row in datanet:
            #print(row)
            #
            if row[1]=='road':
                network.add_edge(row[0][0],row[0][1], color='red')
                """print('road')
                edgelist[row]='red'
                edgelist.append(row)
                edgecolor.append('red')"""
            if row[1]=='rail':
                network.add_edge(row[0][0],row[0][1], color='green')
                #edgelist.append(row)
                #edgecolor.append('green')
            if row[1]=='maritime':
                network.add_edge(row[0][0],row[0][1], color='blue')
                #edgelist.append(row)
                #edgecolor.append('blue')
        #plt.figure(figsize=(10,5))
        #ax = fig.add_subplot()
        pos = nx.spring_layout(network)
        fig=plt.figure(figsize=(2,2))
        edges = network.edges()
        colors = [network[u][w]['color'] for u,w in edges]
        #nx.draw_networkx_nodes(network, pos,node_color='orange', node_size=200, alpha=0.5,
        #nx.draw_networkx_edges(network, pos, edges=edges, edge_color=colors, width=4)
        #nx.draw_networkx_labels(G)
        nx.draw(network,pos,node_color='orange', node_size=200, edges=edges, edge_color=colo
        #plt.title('graph '+net)
        #plt.tight_layout()
        #plt.subplots_adjust(top=0.88)
        """plt.text(1,1, 'rail', fontsize = 20, color = "green")
        plt.text(1, 0.9, 'road', fontsize = 20, color = "red")
        plt.text(1, 0.8, 'sea', fontsize = 20, color = "blue")"""
        fig.savefig("graph"+name+''+net+ ".png", dpi=1000)
        plt.show()

def setupscattergeneral(destinations):
    #for general: to scatter sensitivity analysis
    linenormalx={}
    linenormaly={}
    utopianorm={}
    nadirnorma={}
    #for general: to scatter countries
    COUNTCOOX={}
    COUNTCOOY={}
    UTOPIA={}
    NADIR={}
    #for every run
    coorxdic={}
    coorydic={}
    countries={}
    for D in destinations:
        countries[D]=[]
    restest={} #all results objectives
    nadir={}
    utopia={}
    return restest, nadir, utopia, countries, linenormalx,linenormaly, utopianorm, nadirnorm
```

```
def createnet(filenet, filecorridor):
    network=nx.Graph()
    A=[]
    N=[]
    Mij={}#transportation modes of arc ij
    c={}#cost of transportation mode m on arc a
    d={} #distance of transportation mode m on arc a
    v={} #speed of transportation mode m on arc a
    countreader=0
    with open(filenet, 'r') as file:
        reader = csv.reader(file, delimiter= '\t')
        for row in reader:
            #print(row)
            if countreader >=1:
                network.add_edge(row[0], row[1])
                if (row[0], row[1]) not in A:
                    A.append((row[0], row[1]))
                Mij[(row[0],row[1])]=[]
                c[row[0],row[1],row[2]]=float(row[3].replace(',','.'))
                d[row[0],row[1],row[2]]=float(row[4].replace(',','.'))
                v[row[0],row[1],row[2]]=float(row[6].replace(',','.'))
            else:
                countreader=1
    for key in c.keys():
        if key[2] not in Mij[(key[0],key[1])]:

            Mij[(key[0],key[1])].append(key[2])
    N= list(network.nodes())

    #route list
    route={}
    for arc in A:
        route[arc]=[]
    countreader=0
    with open(filecorridor, 'r') as file:
        reader = csv.reader(file, delimiter= '\t')
        for row in reader:
            #print(row)
            if countreader >=1:
                if row[7] not in route[(row[0], row[1])]:
                    route[(row[0], row[1])].append(row[7])
                    #print(len(row[7]))
                if len(row[8])>=2:
                    if row[8] not in route[(row[0], row[1])]:
                        route[(row[0], row[1])].append(row[8])
                        #print(len(row[8]))
                if len(row[9])>=2:
                    if row[9] not in route[(row[0], row[1])]:
                        route[(row[0], row[1])].append(row[9])
                        #print(len(row[9]))
            else:
                countreader=1
    M=['maritime', 'rail', 'road']
    return M, route,network, A, N, Mij, c, d, v

def createset(G, Node, Arc, Mode):
    I={} #conjoint nodes
    tranship={} #reanshipment nodes
    MI={} #set of transportation modes link i and conjoint nodes
```

```
    for n in Node:
        I[n]=list(G.neighbors(n))
        MI[n]=[]
        transhipp[n]=[]
    for a in Arc:
        n1=a[0]
        n2=a[1]
        for t in Mode[a]:
            transhipp[n1].append(t)
            transhipp[n2].append(t)
            if t not in MI[n1]:
                MI[n1].append(t)
            if t not in MI[n2]:
                MI[n2].append(t)
    return I, transhipp, MI
def createsetnode(fileN):
    C={} #transhipping cost from m1 to m2 at node i
    T={} #transhipping time from m1 to m2 at node i
    Nt={} #transhipping capacity from m1 to m2 at node i
    countreader=0
    with open(fileN, 'r') as file:
        reader = csv.reader(file, delimiter= '\t')
        for row in reader:
            #print(row)
            if countreader >=1:
                C[row[0],row[1],row[2]]=float(row[3].replace(',','.'))
                T[row[0],row[1],row[2]]=float(row[4].replace(',','.'))
                Nt[row[0],row[1],row[2]]=2
                C[row[0],row[2],row[1]]=float(row[3].replace(',','.'))
                T[row[0],row[2],row[1]]=float(row[4].replace(',','.'))
                Nt[row[0],row[2],row[1]]=2
            else:
                countreader=1
    return C, T, Nt
def parameter(transit, consignment, capacity):
    nc = consignment # Number of containers (measured by TEU) carrying the consignment of go
    transit = transit #transit period

    nct={} #capacity of transportation mode m on arc a
    for key2 in c.keys():
        nct[key2]=capacity
    return nc, transit, nct
def exportcsv(MI):
    """allow to create filenode"""
    l1=[]
    l2=[]
    l3=[]
    for mi in MI.keys():
        if len(MI[mi])==1:
            l1.append(mi)
            l2.append(MI[mi][0])
            l3.append(MI[mi][0])
        if len(MI[mi])==2:
            l1.append(mi)
            l2.append(MI[mi][0])
            l3.append(MI[mi][1])
            l1.append(mi)
            l2.append(MI[mi][1])
            l3.append(MI[mi][1])
            l1.append(mi)
```

```
                l2.append(MI[mi][0])
                l3.append(MI[mi][0])
            if len(MI[mi])==3:
                l1.append(mi)
                l2.append(MI[mi][0])
                l3.append(MI[mi][1])
                l1.append(mi)
                l2.append(MI[mi][1])
                l3.append(MI[mi][2])
                l1.append(mi)
                l2.append(MI[mi][0])
                l3.append(MI[mi][2])
                l1.append(mi)
                l2.append(MI[mi][1])
                l3.append(MI[mi][1])
                l1.append(mi)
                l2.append(MI[mi][0])
                l3.append(MI[mi][0])
                l1.append(mi)
                l2.append(MI[mi][2])
                l3.append(MI[mi][2])
        df = DataFrame({'node': l1, 'm1': l2, 'm2':l3})
        df.to_excel('test11.xlsx', sheet_name='sheet1', index=False)

    def singleobj(name, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest, Tr, destinati

        WE={}
        WE['1']=[1,0] #100% weight on obj1
        WE['0']=[0,1] #0% weight on obj1
        result={}

        for ww in WE.keys():
            try:
             # Create a new model
                m = gp.Model(name)
                #set variables
                x = m.addVars(A,M, vtype=GRB.BINARY, name="x") #=1 if goods uses arc a with m
                y = m.addVars(N,M,M, vtype=GRB.BINARY, name="y") #=1 if good transships at i fr

                m.update()

                #set objective

                #Multiobjective: change weight to optimize one only (1,0), change priority to o
                we=WE[ww]
                objname= ww

                m.setObjectiveN((gp.quicksum(x[i,j,m]*c[i,j,m] for (i, j) in A for m in Mij[(i,
                m.setObjectiveN((gp.quicksum(d[i,j,m]/v[i,j,m]*x[i,j,m] for (i, j) in A for m in

                #addconstraints
                m.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mij[h,
                m.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mij[h,
                m.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mij[h,
                m.addConstrs((gp.quicksum(x[i, j, m] for m in Mij[i,j]) <= 1 for i, j in A), "4"
                m.addConstrs((gp.quicksum(y[i, k, l] for k in MI[i] for l in MI[i])<= 1 for i in
                m.addConstrs((gp.quicksum(y[i,k,l] for k in MI[i])==gp.quicksum(x[i,j,l] for j
                m.addConstrs((gp.quicksum(x[h,i,k] for h in I[i] if (h,i) in A if k in Mij[h,i]
                m.addConstr((gp.quicksum((d[i,j,m]/v[i,j,m])*x[i,j,m] for (i,j) in A for m in M
                m.addConstrs((x[i,j,m]*nc <= nct[i,j,m] for (i, j) in A for m in Mij[(i,j)]), "
```

```
        m.addConstrs((y[i,k,l]*nc <= Nt[i,k,l] for i in Tr for k in MI[i] for l in MI[i]
        #11 binary set in addVar
        #12 binary set in addVar
        m.addConstrs((y[i,k,l]==0 for i in Ori_Dest for k in MI[i] for l in MI[i]), "13"


        #minimize
        m.ModelSense=1 #he default 1 value indicates that the objective is to minimize t
        #optimise
        m.optimize()

        m.printStats()

        #help(GRB.Attr)
        for v1 in m.getVars():
            #print('%s %g' % (v1.varName, v1.x))
            result[v1.varName]=v1.x


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

    except gp.GurobiError as e:
        print('Error code ' + str(e.errno) + ': ' + str(e))

    except AttributeError:
        print('Encountered an attribute error')

    namefile= '_res_' + D + '_'+ objname + '.xlsx'
    RESXL=[]
    RESXL.append(namefile)

    objCF=m.getObjective(0)
    objCV=objCF.getValue()
    print(str(objCV)+" u")
    objTF=m.getObjective(1)
    objTV=objTF.getValue()
    print(str(objTV)+" h")
    print("result")
    for i in list(result.keys()):
        if result[i]==1:
            print(i)
            RESXL.append(i)


    #df = DataFrame( {'res': RESXL})
    #df.to_excel(namefile, sheet_name='sheet1', index=False)
    countries[D].append(RESXL)
    nametest= 'res_' +D +'_' + objname

    #store results
    restest[nametest]=[objCV, objTV]


nameD1 = 'res_' +D +'_0'
nameD2= 'res_' +D +'_1'
nadir[D]=[max(restest[nameD1][0], restest[nameD2][0]),max(restest[nameD1][1], restest[na
utopia[D]=[min(restest[nameD1][0], restest[nameD2][0]),min(restest[nameD1][1], restest[r
objCVn= (restest[nameD1][0]-utopia[D][0])/(nadir[D][0]-utopia[D][0])
objTVn= (restest[nameD1][1]-utopia[D][1])/(nadir[D][1]-utopia[D][1])
restest[nameD1]=[restest[nameD1][0], restest[nameD1][1], objCVn, objTVn]
```

```
    objCVn= (restest[nameD2][0]-utopia[D][0])/(nadir[D][0]-utopia[D][0])
    objTVn= (restest[nameD2][1]-utopia[D][1])/(nadir[D][1]-utopia[D][1])
    restest[nameD2]=[restest[nameD2][0], restest[nameD2][1], objCVn, objTVn]
     #create file with stored results
            #nadir point, utopia point
        #ressingobj[nameD1]=[restest[nameD1][0], restest[nameD1][1], objCVn, objTVn]
        #ressingobj[nameD2]=[restest[nameD2][0], restest[nameD2][1], objCVn, objTVn]
    return restest, countries, nadir, utopia
def nwsa(resnwsa, name, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest, Tr, desti
    result={}
    WE={}
    WE['a']=[0.1,0.9]
    WE['b']=[0.2,0.8]
    WE['c']=[0.3,0.7]
    WE['d']=[0.4,0.6]
    WE['e']=[0.5,0.5]
    WE['f']=[0.6,0.4]
    WE['g']=[0.7,0.3]
    WE['h']=[0.8,0.2]
    WE['i']=[0.9,0.1]
    for wekey in WE.keys():
        try:
         # Create a new model
            mo = gp.Model(name)


            x = mo.addVars(A,M, vtype=GRB.BINARY, name="x") #=1 if goods uses arc a with m
            y = mo.addVars(N,M,M, vtype=GRB.BINARY, name="y") #=1 if good transships at i fr

            mo.update()


            #Multiobjective: change weight to optimize one only (1,0), change priority to op

            we=WE[wekey]
            objname= str(WE[wekey][0])

            #we=[1,0]#min obj1
            #we=[0,1]#min obj2
            #we=[0.5,0.5]#min both
            mo.setObjectiveN((((gp.quicksum(x[i,j,m]*c[i,j,m] for (i, j) in A for m in Mij[(
            mo.setObjectiveN((((gp.quicksum(d[i,j,m]/v[i,j,m]*x[i,j,m] for (i, j) in A for m

            #addconstraints
            mo.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mij[h,
            mo.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mij[h,
            mo.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mij[h,
            mo.addConstrs((gp.quicksum(x[i, j, m] for m in Mij[i,j]) <= 1 for i, j in A), "4
            mo.addConstrs((gp.quicksum(y[i, k, l] for k in MI[i] for l in MI[i])<= 1 for i i
            mo.addConstrs((gp.quicksum(y[i,k,l] for k in MI[i])==gp.quicksum(x[i,j,l] for j
            mo.addConstrs((gp.quicksum(x[h,i,k] for h in I[i] if (h,i) in A if k in Mij[h,i]
            mo.addConstr((gp.quicksum((d[i,j,m]/v[i,j,m])*x[i,j,m] for (i,j) in A for m in M
            mo.addConstrs((x[i,j,m]*nc <= nct[i,j,m] for (i, j) in A for m in Mij[(i,j)]), "
            mo.addConstrs((y[i,k,l]*nc <= Nt[i,k,l] for i in Tr for k in MI[i] for l in MI[i
            #11 binary set in addVar
            #12 binary set in addVar
            mo.addConstrs((y[i,k,l]==0 for i in Ori_Dest for k in MI[i] for l in MI[i]), "13
            #deduct a solution
            mo.addConstr(((gp.quicksum(x[i,j,m]*c[i,j,m] for (i, j) in A for m in Mij[(i,j)]
            mo.addConstr(((gp.quicksum(d[i,j,m]/v[i,j,m]*x[i,j,m] for (i, j) in A for m in M
```

```
            #minimize
            mo.ModelSense=1 #he default 1 value indicates that the objective is to minimize
            #optimise
            mo.optimize()

            mo.printStats()

            #help(GRB.Attr)
            for v1 in mo.getVars():
                #print('%s %g' % (v1.varName, v1.x))
                result[v1.varName]=v1.x


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

        except gp.GurobiError as e:
            print('Error code ' + str(e.errno) + ': ' + str(e))

        except AttributeError:
            print('Encountered an attribute error')

        namefile= '_res_' + D + '_'+ objname + '.xlsx'
        RESXL=[]
        RESXL.append(namefile)

        objCF=mo.getObjective(0)
        objCV=objCF.getValue()
        print(str(objCV)+" u")
        objTF=mo.getObjective(1)
        objTV=objTF.getValue()
        print(str(objTV)+" h")
        print("result")
        for i in list(result.keys()):
            if result[i]==1:
                print(i)
                RESXL.append(i)


        df = DataFrame( {'res': RESXL})
        #df.to_excel(namefile, sheet_name='sheet1', index=False)
        countries[D].append(RESXL)
        #store results
        nametest= 'res_' +D +'_' + objname
        objCVreal=(objCV*(nadir[D][0]-utopia[D][0]))+utopia[D][0]
        objTVreal=(objTV*(nadir[D][1]-utopia[D][1]))+utopia[D][1]
        restest[nametest]=[objCVreal, objTVreal, objCV, objTV]
        resnwsa[nametest]=[objCVreal, objTVreal, objCV, objTV]
    return restest, countries, resnwsa

def ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest, Tr, destination,
    result={}
    WE={}
    WE['1']=[1,0] #100% weight on obj1
    WE['0']=[0,1] #0% weight on obj1
    epsilon={}
    epsilon[0]=(nadir[D][0]-utopia[D][0])/number
    epsilon[1]=(nadir[D][1]-utopia[D][1])/number
    intervals={}
    intervals[0]=[]
```

```
intervals[1]=[]
intervals[0].append(utopia[D][0]+epsilon[0])
intervals[1].append(utopia[D][1]+epsilon[1])
for i in range(number-2):
    #print(i)
    intervals[0].append(intervals[0][i]+epsilon[0])
    intervals[1].append(intervals[1][i]+epsilon[1])
for ww in list(WE.keys()):
    #print(ww)
    we=WE[ww]
    objname= 'z_ecm_'+ww
    for interval in list(intervals[we[0]]):
        #print(interval)
        try:
         # Create a new model
            m = gp.Model("ecm")
            #set variables

            x = m.addVars(A,M, vtype=GRB.BINARY, name="x") #=1 if goods uses arc a with
            y = m.addVars(N,M,M, vtype=GRB.BINARY, name="y") #=1 if good transships at

            m.update()


            #Multiobjective: change weight to optimize one only (1,0), change priority

            m.setObjectiveN((gp.quicksum(x[i,j,m]*c[i,j,m] for (i, j) in A for m in Mij
            m.setObjectiveN((gp.quicksum(d[i,j,m]/v[i,j,m]*x[i,j,m] for (i, j) in A for

            #addconstraints
            m.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mi
            m.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mi
            m.addConstrs((gp.quicksum(x[h, i, l]for h in I[i] if (h,i) in A for l in Mi
            m.addConstrs((gp.quicksum(x[i, j, m] for m in Mij[i,j]) <= 1 for i, j in A)
            m.addConstrs((gp.quicksum(y[i, k, l] for k in MI[i] for l in MI[i])<= 1 for
            m.addConstrs((gp.quicksum(y[i,k,l] for k in MI[i])==gp.quicksum(x[i,j,l] for
            m.addConstrs((gp.quicksum(x[h,i,k] for h in I[i] if (h,i) in A if k in Mij[k
            m.addConstr((gp.quicksum((d[i,j,m]/v[i,j,m])*x[i,j,m] for (i,j) in A for m
            m.addConstrs((x[i,j,m]*nc <= nct[i,j,m] for (i, j) in A for m in Mij[(i,j)]
            m.addConstrs((y[i,k,l]*nc <= Nt[i,k,l] for i in Tr for k in MI[i] for l in M
            #11 binary set in addVar
            #12 binary set in addVar
            m.addConstrs((y[i,k,l]==0 for i in Ori_Dest for k in MI[i] for l in MI[i]),
            #nadir
            m.addConstr(((gp.quicksum(x[i,j,m]*c[i,j,m] for (i, j) in A for m in Mij[(i,
            m.addConstr(((gp.quicksum(d[i,j,m]/v[i,j,m]*x[i,j,m] for (i, j) in A for m
            #ECM constraint
            if ww=='0':
                m.addConstr((gp.quicksum(x[i,j,m]*c[i,j,m] for (i, j) in A for m in Mij
            if ww=='1':
                m.addConstr((gp.quicksum(d[i,j,m]/v[i,j,m]*x[i,j,m] for (i, j) in A for
            #minimize
            m.ModelSense=1 #he default 1 value indicates that the objective is to minim
            #optimise
            m.update()
            m.optimize()

            m.printStats()

            #help(GRB.Attr)
```

```
                for v1 in m.getVars():
                    #print('%s %g' % (v1.varName, v1.x))
                    result[v1.varName]=v1.x

                #print('Obj: %g' % m.objVal)
                namefile= '_res_' + D + '_'+ objname + str(interval) +'.xlsx'
                RESXL=[]
                RESXL.append(namefile)

                objCF=m.getObjective(0)
                objCV=objCF.getValue()
                print(str(objCV)+" u")
                objTF=m.getObjective(1)
                objTV=objTF.getValue()
                print(str(objTV)+" h")
                #print("result")
                for i in list(result.keys()):
                    if result[i]==1:
                        print(i)
                        RESXL.append(i)
                countries[D].append(RESXL)


                nametest= 'res_' +D +'_' + objname +str(interval)
                objCVn= (objCV-utopia[D][0])/(nadir[D][0]-utopia[D][0])
                objTVn=(objTV-utopia[D][1])/(nadir[D][1]-utopia[D][1])
                restest[nametest]=[objCV, objTV, objCVn, objTVn]
                #resecm[nametest]=[objCV, objTV, objCVn, objTVn]
            except gp.GurobiError as e:
                print('Error code ' + str(e.errno) + ': ' + str(e))

            except AttributeError:
                print('Encountered an attribute error')


                df = DataFrame( {'res': RESXL})
                #df.to_excel(namefile, sheet_name='sheet1', index=False)

    return restest, countries
def storeresult(restest,utopia,nadir, D):
    storedname=[]
    storedobj1=[]
    storedobj2=[]
    storedobj1n=[]
    storedobj2n=[]
    for key in list(restest.keys()):
        storedname.append(key)
        storedobj1.append(restest[key][0])
        storedobj2.append(restest[key][1])
        storedobj1n.append(restest[key][2])
        storedobj2n.append(restest[key][3])
    namaut= 'res_'+ D +'__utopia'
    namana= 'res_'+ D +'__nadir'
    storedname.append(namaut)
    storedobj1.append(utopia[D][0])
    storedobj2.append(utopia[D][1])
    storedobj1n.append(0)
    storedobj2n.append(0)
    storedname.append(namana)
    storedobj1.append(nadir[D][0])
```

```
    storedobj2.append(nadir[D][1])
    storedobj1n.append(1)
    storedobj2n.append(1)


    dftest = DataFrame({'name': storedname, 'obj1': storedobj1, 'obj2': storedobj2, 'obj1n'
    #dftest.to_excel(filestorage, sheet_name='sheet1', index=False)
    return dftest
def noduplicates(restest,utopia,nadir):
    resnd=restest.copy() #restest without duplicates
    deleted={}
    for knd1 in list(restest.keys()):
        if knd1 in list(resnd.keys()):
            nd1=resnd[knd1]
            #print(knd1)
            #print('keep')
            for knd2 in list(restest.keys()):
                if knd2 in list(resnd.keys()):
                    nd2=resnd[knd2]
                    #print(knd2)
                    if knd2 != knd1:
                        if abs(nd1[0]-nd2[0])<=1:
                            if abs(nd1[1]-nd2[1])<=1:
                                #print('same')
                                deleted[knd2]=nd2
                                resnd.pop(knd2)
                                #print('deleted')
    return resnd
def dftestnduplicates(resnd, count, countnwsa):
    storedname=[]
    storedobj1=[]
    storedobj2=[]
    storedobj1n=[]
    storedobj2n=[]
    for key in list(resnd.keys()):
        storedname.append(key)
        storedobj1.append(resnd[key][0])
        storedobj2.append(resnd[key][1])
        storedobj1n.append(resnd[key][2])
        storedobj2n.append(resnd[key][3])
    namaut= 'res_'+ D +'__utopia'
    namana= 'res_'+ D +'__nadir'
    storedname.append(namaut)
    storedobj1.append(utopia[D][0])
    storedobj2.append(utopia[D][1])
    storedobj1n.append(0)
    storedobj2n.append(0)
    storedname.append(namana)
    storedobj1.append(nadir[D][0])
    storedobj2.append(nadir[D][1])
    storedobj1n.append(1)
    storedobj2n.append(1)

    C=list(count.values())
    C2=list(countnwsa.values())
    C.append(0)
    C2.append(0)
    C.append(0)
    C2.append(0)
    #print(len(C))
```

```
    #print(len(C2))
    #print(len(storedname))
    dftestnd = DataFrame({'name': storedname, 'obj1': storedobj1, 'obj2': storedobj2, 'obj1r
    #dftestnd.to_excel(filestoragend, sheet_name='sheet1', index=False)
    return dftestnd


def prepscatall(scat, resnd, coorydic, coorxdic):
    #prep scatterall
    countrx=[]
    country=[]

    coorxdic[scat]=[]
    coorydic[scat]=[]
    sort_resnd = sorted(resnd.items(), key=lambda x: x[1][0], reverse=True)
    for donnee in sort_resnd:
        coorxdic[scat].append(donnee[1][0])
        coorydic[scat].append(donnee[1][1])
        countrx.append(donnee[1][0])
        country.append(donnee[1][1])
    return sort_resnd, coorxdic, coorydic, countrx, country

def df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, countdf,
    #df count
    listdf= [dftest,dftestnd]
    namedfc= D+'_'+str(countries[D][0][0])
    dfc=DataFrame({namedfc:countries[D][0]})
    for cou in range(len(countries[D])-1):
        namedfc= D+'_'+str(countries[D][cou+1][0])
        dfc1=DataFrame({namedfc:countries[D][cou+1]})
        dfc=pd.concat([dfc,dfc1], ignore_index=True, axis=1)

    listdf.append(dfc)

    #df route prop
    dfr=DataFrame({'porportion':[D,'TSR', 'LSR', 'SC', 'TC', 'EUR', 'rail', 'road', 'maritir
    for rp in routeprop.keys():
        dfr2=DataFrame({rp:routeprop[rp]})
        dfr=pd.concat([dfr,dfr2], ignore_index=True, axis=1)
    listdf.append(dfr)

    #df route prop total
    dft=DataFrame({'porportion':[D,'TSR', 'LSR', 'SC', 'TC', 'EUR', 'rail', 'road', 'maritir
    for rt in routeproptot.keys():
        dft2=DataFrame({rt:routeproptot[rt]})
        dft=pd.concat([dft,dft2], ignore_index=True, axis=1)
    listdf.append(dft)
    #df route prop total
    dft=DataFrame({'porportion':[D,'TSR', 'LSR', 'SC', 'TC', 'EUR', 'rail', 'road', 'maritir
    for rt in routeproptot.keys():
        dft2=DataFrame({rt:routeproptot[rt]})
        dft=pd.concat([dft,dft2], ignore_index=True, axis=1)
    listdf.append(dft)


    writer = pd.ExcelWriter(xlspath, engine='xlsxwriter')

    for n, df in enumerate(listdf):
        df.to_excel(writer, sheet_name='sheet%s'%n, index=False)
    writer.save()
```

```
def scattercountries(scatcount,destinations, COUNTCOOX, COUNTCOOY,UTOPIA, NADIR):
    fig = plt.figure()
    ax = fig.add_subplot()
    colorco=['green', 'red', 'black', 'blue', 'magenta', 'orange', 'purple']
    for D,col in zip(destinations,colorco):
        plt.scatter(COUNTCOOX[D], COUNTCOOY[D], color=col, alpha=0.5, label=D)
        plt.scatter(UTOPIA[D][0], UTOPIA[D][1], color=col, alpha=0.5, marker='x')
        plt.scatter(NADIR[D][0], NADIR[D][1], color=col, alpha=0.5, marker='v')
        plt.plot(COUNTCOOX[D], COUNTCOOY[D], color=col,linestyle='solid')
    plt.title('Pareto Frontier: '+ scatcount)
    plt.legend()
    plt.xlabel('Total Cost (obj1) [USD]')
    plt.ylabel('Total Time (obj2) [h]')
    #plt.plot([0, 1], [1, 0],color='black',linestyle='dashed',linewidth=1)
    plt.show()
    fig.savefig(scatcount+'.png')

def proptotal(resnd, restest, resnwsa):
    count={}
    countnwsa={}
    for key in resnd.keys():
        #print(key)
        count[key]=1
        A=resnd[key][0]
        B=resnd[key][1]

        for kex in restest.keys():
            #print(kex)
            C=restest[kex][0]
            D=restest[kex][1]
            if kex != key:

                if abs(A-C)<=0.01:

                    if abs(B-D)<=0.01:
                        #print('ok')
                        count[key]=count[key]+1

    for key in resnd.keys():
        #print(key)
        countnwsa[key]=1
        A=resnd[key][0]
        B=resnd[key][1]

        for kex in resnwsa.keys():
            #print(kex)
            C=resnwsa[kex][0]
            D=resnwsa[kex][1]
            if kex != key:

                if abs(A-C)<=0.01:

                    if abs(B-D)<=0.01:
                        print('ok')
                        countnwsa[key]=countnwsa[key]+1

    countdf = DataFrame({'name': list(count.keys()), 'count/33': list(count.values())})
    countnwsadf= DataFrame({'name': list(countnwsa.keys()), 'count/10nwsa': list(countnwsa.v
    return count, countnwsa, countdf, countnwsadf
```

```
def SA(nom, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, nadir, utopia):
        restest={}#all results objectives
        resnwsa={}
        #setup to later scatter
        coorxdic={}
        coorydic={}
        scat= D + nom
        #setup
        Tr=[]
        Tr= list(network.nodes()) #number transition nodes
        Tr.remove(O)
        Tr.remove(D)
        xlspath='SAoutput'+D+'_'+nom+'.xlsx'

        destination= []
        destination.append(D)
        origin= []
        origin.append(O)
        Ori_Dest=[D,O]
        #run general

        name='general'
        restest, countries, nadir, utopia=singleobj(name, O, D, countries, c, d, v, C, A, Mi
        #run MADM
        name='MADM'
        restest, countries, resnwsa=nwsa(resnwsa,name, O, D, countries, c, d, v, C, A, Mij,
        #run ECM
        number=10

        restest, countries=ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, res
        #create file with stored results
        dftest=storeresult(restest,utopia,nadir, D)
        resnd=noduplicates(restest,utopia,nadir)
        count, countnwsa, countdf, countnwsadf=proptotal(resnd, restest, resnwsa)
        dftestnd=dftestnduplicates(resnd, count, countnwsa)
        #scatter
        sort_resnd, coorxdic, coorydic, countrx, country= prepscatall(scat, resnd, coorydic,
        #df
        routeprop, routenet=getroute(countries[D], route, resnd)
        routeproptot, routenet2=getroute(countries[D], route, restest)

        df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, count
        draw(routenet, scat)

        linex= coorxdic[scat]
        liney= coorydic[scat]
        utx= utopia[D][0]
        uty=utopia[D][1]
        nax= nadir[D][0]
        nay=nadir[D][1]
        return linex, liney, utx, uty,nax, nay
#%%
"""======______main_____=============================================================
=====================================================================
======================================================================================
=====================================================================""" 

#once for normal
col1=[]
```

```
col1.append('destination')

destinations=['Athens', 'Sofia', 'Bucharest']

destinations=['Helsinki', 'Stockholm', 'Copenhagen', 'Warsaw', 'Riga', 'Tallinn', 'Vilnius']
destinations=['Berlin', 'Amsterdam', 'Paris', 'Brussels', 'London', 'Dublin']
#destinations=['Madrid', 'Lisbon', 'Roma', 'Zagreb', 'Athens', 'Ljubljana']
#destinations=['Bucharest', 'Sofia','Budapest','Prague','Vienna','Bratislava']
#destinations=['Athens', 'Sofia', 'Bucharest']
#destinations=['Brussels', 'Helsinki', 'Madrid', 'Budapest']
#destinations=['Brussels']
#destinations=['Brussels']
scatcount="Western EU capitals"

O='China'
filearc='test9.txt'
fileroute='test9b.txt'
filenode= 'test99.txt'
M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
C, T, Nt= createsetnode(filenode)
I, tranship, MI=createset(network, N, A, Mij)
nc, transit, nct= parameter(1000, 1, 2)

hubs={}
hubs2={}
hubD={}
hub=['Warsaw', 'Berlin', 'Brussels', 'Madrid', 'Lisbon', 'Zagreb', 'Sofia', 'Budapest', 'Buc
for des in hub:
    col1.append(des)
datafr=DataFrame({'hubs':col1})
for hb in hub:
    hubs[hb]=0
    hubs2[hb]=0
for D in destinations:
    hubD[D]={}
#for normal: define and empty
restest, nadir, utopia, countries, linenormalx,linenormaly, utopianorm, nadirnorma, COUNTCO(


for D in destinations:
    for hb in hub:
        hubs[hb]=0
        hubs2[hb]=0
    restest={}#all results objectives
    resnwsa={}
    #setup to later scatter
    coorxdic={}
    coorydic={}
    scatall= D + ' '
    scat= D
    #setup
    xlspath='output'+D+'.xlsx'
    Tr=[]
    Tr= list(network.nodes()) #number transition nodes
    Tr.remove(O)
    Tr.remove(D)

    destination= []
    destination.append(D)
    origin= []
```

```
    origin.append(O)
    Ori_Dest=[D,O]
    #run general

    name='general'
    restest, countries, nadir, utopia=singleobj(name, O, D, countries, c, d, v, C, A, Mij, M
    #run MADM
    name='MADM'
    restest, countries, resnwsa=nwsa(resnwsa,name, O, D, countries, c, d, v, C, A, Mij, MI,
    #run ECM
    number=10

    restest, countries=ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest
    #create file with stored results
    dftest=storeresult(restest,utopia,nadir, D)
    resnd=noduplicates(restest,utopia,nadir)
    count, countnwsa, countdf, countnwsadf=proptotal(resnd, restest, resnwsa)
    dftestnd=dftestnduplicates(resnd, count, countnwsa)
    #scatter
    sort_resnd, coorxdic, coorydic, countrx, country= prepscatall(scat, resnd, coorydic, coo
    #df
    routeprop, routenet, hubs=getroute(countries[D], route, resnd, hubs, hub)
    routeproptot, routenet2, hubs2=getroute(countries[D], route, restest, hubs2, hub)
    hubD[D]=hubs
    df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, countdf,
    #draw(routenet, scatall)
    #scattercountries
    COUNTCOOX[D]=countrx
    COUNTCOOY[D]=country
    UTOPIA[D]=utopia[D]
    NADIR[D]=nadir[D]

     #keep the normal to scatter later in SA
    linenormalx[D]=coorxdic[scat]
    linenormaly[D]=coorydic[scat]
    utopianorm[D]=utopia[D]
    nadirnorma[D]=nadir[D]


    col2=[]
    col2.append(D)
    for des in hub:
        if des == D:
            col2.append(hubs[des])
        else:
            col2.append(hubs[des]/2)
    datafr2=DataFrame({D:col2})
    datafr=pd.concat([datafr,datafr2], ignore_index=True, axis=1)
datafr.to_excel("hubs.xlsx")
scattercountries(scatcount,destinations, COUNTCOOX, COUNTCOOY,UTOPIA, NADIR)

#%%SA Train speed variations
#for every run: define and empty
nomall='-Train speed variations along TSR'
labels= ['+50%', '0', '+20%'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']
```

```
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'TSR' in route[arck]:
            if keymode[2]==rail:
                v[keymode]=v[keymode]*1.2
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'TSR' in route[arck]:
            if keymode[2]==rail:
                v[keymode]=v[keymode]*1.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,

#%%SA Train speed variations
#for every run: define and empty
nomall='-Train speed variations'
labels= ['+50%', '0', '-50%'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']

#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==rail:
            v[keymode]=v[keymode]*0.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
```

```
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==rail:
            v[keymode]=v[keymode]*1.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,
#%%
#for every run: define and empty
nomall='-Ship speed variations'
labels= ['+50%', '0', '-50%'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']

#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==maritime:
            v[keymode]=v[keymode]*0.5
    transit = 1000*2
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==maritime:
            v[keymode]=v[keymode]*1.5
    transit = 1000*2
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,
#%%
#for every run: define and empty
#nomall='-Truck speed variations'
#labels= ['+50%', '0', '-50%'] #SA2, general, SA1
nomall='-Truck hours variations'
labels= ['20 hours', '15 hours', '10 hours'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']
```

```
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==road:
            v[keymode]=v[keymode]*10/15
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==road:
            v[keymode]=v[keymode]*20/15
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y
#%%
#for every run: define and empty
nomall='-Ship cost variations'
labels= ['+50%', '0', '-50%'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']

#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==maritime:
            c[keymode]=c[keymode]*0.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==maritime:
```

```
            c[keymode]=c[keymode]*1.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,
#%%
#for every run: define and empty
#nomall='-Train cost variations'
#labels= ['+50%', '0', '-50%'] #SA2, general, SA1
nomall='-Train subsidies variations'
labels= ['0', '0.8', '1.6'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']

#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==rail:
            if c[keymode]>=0.8*d[keymode]:
                c[keymode]=c[keymode]-0.8*d[keymode]
            else:
                c[keymode]=0
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==rail:
            c[keymode]=c[keymode]+0.8*d[keymode]
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,
#%%
#for every run: define and empty
nomall='-Truck cost variations'
labels= ['+50%', '0', '-50%'] #SA2, general, SA1
countries={}

#change within same SA
nom= [nomall + ' 1', nomall+'2']
```

```
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:

    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==road:
            c[keymode]=c[keymode]*0.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        if keymode[2]==road:
            c[keymode]=c[keymode]*1.5
    transit = 1000
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    scattersa(line1x, line1y, ut1x, ut1y, line2x, line2y, ut2x, ut2y,na1x, na1y, na2x, na2y,

#%%SC speed
#for every run: define and empty
nomall= ' Speed variations along SC'
countries={}
labels= ['0', '+50% speed SC', '+100% speed SC', '+200% speed SC', '+400% speed SC'] #SA2, g
#change within same SA
nom= ['+50% speed SC', '+100% speed SC', '+200% speed SC', '+400% speed SC']
summary={}
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:
    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
        arck=(keymode[0], keymode[1])
        if 'SC' in route[arck]:
            v[keymode]=v[keymode]*1.5
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
            arck=(keymode[0], keymode[1])
            if 'SC' in route[arck]:
```

```
                v[keymode]=v[keymode]*2
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, N

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
        arck=(keymode[0], keymode[1])
        if 'SC' in route[arck]:
            v[keymode]=v[keymode]*3
    countries[D]=[]
    nadir={}
    utopia={}
    line3x, line3y, ut3x, ut3y,na3x, na3y=SA(nom[2], O, D, countries, c, d, v, C, A, Mij, MI

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
        arck=(keymode[0], keymode[1])
        if 'SC' in route[arck]:
            v[keymode]=v[keymode]*5
    countries[D]=[]
    nadir={}
    utopia={}
    line4x, line4y, ut4x, ut4y,na4x, na4y=SA(nom[3], O, D, countries, c, d, v, C, A, Mij, MI

    scattersa2(line4x, line4y, ut4x, ut4y,na4x, na4y, line3x, line3y, ut3x, ut3y,na3x, na3y,
#%% speed SC graph countries
    #once for normal
destinations=['Athens', 'Sofia', 'Bucharest']
scatcount="South-Eastern EU capitals- +400% speed on SC"

O='China'
filearc='test9.txt'
fileroute='test9b.txt'
filenode= 'test99.txt'
M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
C, T, Nt= createsetnode(filenode)
I, tranship, MI=createset(network, N, A, Mij)
nc, transit, nct= parameter(1000, 1, 2)


#for normal: define and empty
restest, nadir, utopia, countries, linenormalx,linenormaly, utopianorm, nadirnorma, COUNTCO
for keymode in v.keys():
    arck=(keymode[0], keymode[1])
    if 'SC' in route[arck]:
        v[keymode]=v[keymode]*5

for D in destinations:
    restest={}#all results objectives
    resnwsa={}
    #setup to later scatter
    coorxdic={}
    coorydic={}
    scatall= D + ' '
    scat= D
    #setup
    xlspath='output'+D+'.xlsx'
    Tr=[]
```

```
    Tr= list(network.nodes()) #number transition nodes
    Tr.remove(O)
    Tr.remove(D)

    destination= []
    destination.append(D)
    origin= []
    origin.append(O)
    Ori_Dest=[D,O]
    #run general

    name='general'
    restest, countries, nadir, utopia=singleobj(name, O, D, countries, c, d, v, C, A, Mij, M
    #run MADM
    name='MADM'
    restest, countries, resnwsa=nwsa(resnwsa,name, O, D, countries, c, d, v, C, A, Mij, MI,
    #run ECM
    number=10

    restest, countries=ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest
    #create file with stored results
    dftest=storeresult(restest,utopia,nadir, D)
    resnd=noduplicates(restest,utopia,nadir)
    count, countnwsa, countdf, countnwsadf=proptotal(resnd, restest, resnwsa)
    dftestnd=dftestnduplicates(resnd, count, countnwsa)
    #scatter
    sort_resnd, coorxdic, coorydic, countrx, country= prepscatall(scat, resnd, coorydic, coo
    #df
    routeprop, routenet, hubs=getroute(countries[D], route, resnd, hubs, hub)
    routeproptot, routenet2, hubs=getroute(countries[D], route, restest, hubs, hub)

    df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, countdf,
    draw(routenet, scatall)
    #scattercountries
    COUNTCOOX[D]=countrx
    COUNTCOOY[D]=country
    UTOPIA[D]=utopia[D]
    NADIR[D]=nadir[D]



scattercountries(scatcount,destinations, COUNTCOOX, COUNTCOOY,UTOPIA, NADIR)
#%%TC speed
#for every run: define and empty
nomall= ' Speed variations along TC'
countries={}
labels= ['0', '+50% speed TC', '+100% speed TC', '+200% speed TC', '+400% speed TC'] #SA2, g
#change within same SA
nom= ['+50% speed TC', '+100% speed TC', '+200% speed TC', '+400% speed TC']
summary={}
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:
    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
```

```
        arck=(keymode[0], keymode[1])
        if 'TC' in route[arck]:
            v[keymode]=v[keymode]*1.5
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
            arck=(keymode[0], keymode[1])
            if 'TC' in route[arck]:
                v[keymode]=v[keymode]*2
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
        arck=(keymode[0], keymode[1])
        if 'TC' in route[arck]:
            v[keymode]=v[keymode]*3
    countries[D]=[]
    nadir={}
    utopia={}
    line3x, line3y, ut3x, ut3y,na3x, na3y=SA(nom[2], O, D, countries, c, d, v, C, A, Mij, MI

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
        arck=(keymode[0], keymode[1])
        if 'TC' in route[arck]:
            v[keymode]=v[keymode]*5
    countries[D]=[]
    nadir={}
    utopia={}
    line4x, line4y, ut4x, ut4y,na4x, na4y=SA(nom[3], O, D, countries, c, d, v, C, A, Mij, MI

    scattersa2(line4x, line4y, ut4x, ut4y,na4x, na4y, line3x, line3y, ut3x, ut3y,na3x, na3y,
#%% speed TC graph countries
    #once for normal
destinations=['Athens', 'Sofia', 'Bucharest']
scatcount="South-Eastern EU capitals- +400% speed on TC"

O='China'
filearc='test9.txt'
fileroute='test9b.txt'
filenode= 'test99.txt'
M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
C, T, Nt= createsetnode(filenode)
I, tranship, MI=createset(network, N, A, Mij)
nc, transit, nct= parameter(1000, 1, 2)


#for normal: define and empty
restest, nadir, utopia, countries, linenormalx,linenormaly, utopianorm, nadirnorma, COUNTCO
for keymode in v.keys():
    arck=(keymode[0], keymode[1])
    if 'TC' in route[arck]:
        v[keymode]=v[keymode]*5
```

```
for D in destinations:
    restest={}#all results objectives
    resnwsa={}
    #setup to later scatter
    coorxdic={}
    coorydic={}
    scatall= D + ' '
    scat= D
    #setup
    xlspath='output'+D+'.xlsx'
    Tr=[]
    Tr= list(network.nodes()) #number transition nodes
    Tr.remove(O)
    Tr.remove(D)

    destination= []
    destination.append(D)
    origin= []
    origin.append(O)
    Ori_Dest=[D,O]
    #run general

    name='general'
    restest, countries, nadir, utopia=singleobj(name, O, D, countries, c, d, v, C, A, Mij, M
    #run MADM
    name='MADM'
    restest, countries, resnwsa=nwsa(resnwsa,name, O, D, countries, c, d, v, C, A, Mij, MI,
    #run ECM
    number=10

    restest, countries=ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest
    #create file with stored results
    dftest=storeresult(restest,utopia,nadir, D)
    resnd=noduplicates(restest,utopia,nadir)
    count, countnwsa, countdf, countnwsadf=proptotal(resnd, restest, resnwsa)
    dftestnd=dftestnduplicates(resnd, count, countnwsa)
    #scatter
    sort_resnd, coorxdic, coorydic, countrx, country= prepscatall(scat, resnd, coorydic, coo
    #df
    routeprop, routenet=getroute(countries[D], route, resnd)
    routeproptot, routenet2=getroute(countries[D], route, restest)

    df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, countdf,
    #draw(routenet, scatall)
    #scattercountries
    COUNTCOOX[D]=countrx
    COUNTCOOY[D]=country
    UTOPIA[D]=utopia[D]
    NADIR[D]=nadir[D]



scattercountries(scatcount,destinations, COUNTCOOX, COUNTCOOY,UTOPIA, NADIR)
#%%SC cost
#for every run: define and empty
nomall= ' Cost variations along SC'
countries={}
labels= ['0', '-50% cost SC', '-70% cost SC', '-90% cost SC'] #SA2, general, SA1
#change within same SA
```

```
nom= ['-50% cost SC', '-70% cost SC', '-90% cost SC']
summary={}
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, tranship, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:
    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in v.keys():
        arck=(keymode[0], keymode[1])
        if 'SC' in route[arck]:
            c[keymode]=c[keymode]*0.5
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
            arck=(keymode[0], keymode[1])
            if 'SC' in route[arck]:
                c[keymode]=c[keymode]*0.3
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'SC' in route[arck]:
            c[keymode]=c[keymode]*0.1
    countries[D]=[]
    nadir={}
    utopia={}
    line3x, line3y, ut3x, ut3y,na3x, na3y=SA(nom[2], O, D, countries, c, d, v, C, A, Mij, MI

    """M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'SC' in route[arck]:
            c[keymode]=c[keymode]*0.1
    countries[D]=[]
    nadir={}
    utopia={}
    line4x, line4y, ut4x, ut4y,na4x, na4y=SA(nom[3], O, D, countries, c, d, v, C, A, Mij, MI

    #scattersa2(line4x, line4y, ut4x, ut4y,na4x, na4y, line3x, line3y, ut3x, ut3y,na3x, na3y
#%% speed SC graph countries
    #once for normal
destinations=['Athens', 'Sofia', 'Bucharest']
scatcount="South-Eastern EU capitals- -90% cost on SC"

O='China'
filearc='test9.txt'
fileroute='test9b.txt'
filenode= 'test99.txt'
```

```
M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
C, T, Nt= createsetnode(filenode)
I, tranship, MI=createset(network, N, A, Mij)
nc, transit, nct= parameter(1000, 1, 2)


#for normal: define and empty
restest, nadir, utopia, countries, linenormalx,linenormaly, utopianorm, nadirnorma, COUNTCO
for keymode in c.keys():
    arck=(keymode[0], keymode[1])
    if 'SC' in route[arck]:
        c[keymode]=c[keymode]*0.1

for D in destinations:
    restest={}#all results objectives
    resnwsa={}
    #setup to later scatter
    coorxdic={}
    coorydic={}
    scatall= D + ' '
    scat= D
    #setup
    xlspath='output'+D+'.xlsx'
    Tr=[]
    Tr= list(network.nodes()) #number transition nodes
    Tr.remove(O)
    Tr.remove(D)

    destination= []
    destination.append(D)
    origin= []
    origin.append(O)
    Ori_Dest=[D,O]
    #run general

    name='general'
    restest, countries, nadir, utopia=singleobj(name, O, D, countries, c, d, v, C, A, Mij, M
    #run MADM
    name='MADM'
    restest, countries, resnwsa=nwsa(resnwsa,name, O, D, countries, c, d, v, C, A, Mij, MI,
    #run ECM
    number=10

    restest, countries=ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest
    #create file with stored results
    dftest=storeresult(restest,utopia,nadir, D)
    resnd=noduplicates(restest,utopia,nadir)
    count, countnwsa, countdf, countnwsadf=proptotal(resnd, restest, resnwsa)
    dftestnd=dftestnduplicates(resnd, count, countnwsa)
    #scatter
    sort_resnd, coorxdic, coorydic, countrx, country= prepscatall(scat, resnd, coorydic, coo
    #df
    routeprop, routenet=getroute(countries[D], route, resnd)
    routeproptot, routenet2=getroute(countries[D], route, restest)

    df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, countdf,
    draw(routenet, scatall)
    #scattercountries
    COUNTCOOX[D]=countrx
    COUNTCOOY[D]=country
```

```
    UTOPIA[D]=utopia[D]
    NADIR[D]=nadir[D]



scattercountries(scatcount,destinations, COUNTCOOX, COUNTCOOY,UTOPIA, NADIR)
#%%TC cost
#for every run: define and empty
nomall= ' Cost variations along TC'
countries={}
labels= ['0', '-50% cost TC', '-70% cost TC', '-90% cost TC'] #SA2, general, SA1
#change within same SA
nom= ['-50% cost TC', '-70% cost TC', '-90% cost TC']
summary={}
#filearc=
#filenode=
#M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
#C, T, Nt= createsetnode(filenode)
#I, transhipp, MI=createset(network, N, A, Mij)
#nc, transit, nct= parameter(1000, 1, 2)
for D in destinations:
    scatall= D+ nomall
    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'TC' in route[arck]:
            c[keymode]=c[keymode]*0.5
    countries[D]=[]
    nadir={}
    utopia={}
    line1x, line1y, ut1x, ut1y,na1x, na1y=SA(nom[0], O, D, countries, c, d, v, C, A, Mij, MI

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
            arck=(keymode[0], keymode[1])
            if 'TC' in route[arck]:
                c[keymode]=c[keymode]*0.3
    countries[D]=[]
    nadir={}
    utopia={}
    line2x, line2y, ut2x, ut2y, na2x, na2y=SA(nom[1], O, D, countries, c, d, v, C, A, Mij, M

    M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'TC' in route[arck]:
            c[keymode]=c[keymode]*0.1
    countries[D]=[]
    nadir={}
    utopia={}
    line3x, line3y, ut3x, ut3y,na3x, na3y=SA(nom[2], O, D, countries, c, d, v, C, A, Mij, MI

    """M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
    for keymode in c.keys():
        arck=(keymode[0], keymode[1])
        if 'TC' in route[arck]:
            c[keymode]=c[keymode]*0.1
    countries[D]=[]
    nadir={}
    utopia={}
```

```
    line4x, line4y, ut4x, ut4y,na4x, na4y=SA(nom[3], O, D, countries, c, d, v, C, A, Mij, MI

    scattersa2(line4x, line4y, ut4x, ut4y,na4x, na4y, line3x, line3y, ut3x, ut3y,na3x, na3y,
#%% speed TC graph countries
    #once for normal
destinations=['Athens', 'Sofia', 'Bucharest']
scatcount="South-Eastern EU capitals- -90% cost on TC"

O='China'
filearc='test9.txt'
fileroute='test9b.txt'
filenode= 'test99.txt'
M, route,network,A, N, Mij, c, d, v=createnet(filearc, fileroute)
C, T, Nt= createsetnode(filenode)
I, transhipment, MI=createset(network, N, A, Mij)
nc, transit, nct= parameter(1000, 1, 2)


#for normal: define and empty
restest, nadir, utopia, countries, linenormalx,linenormaly, utopianorm, nadirnorma, COUNTCO
for keymode in c.keys():
    arck=(keymode[0], keymode[1])
    if 'TC' in route[arck]:
        c[keymode]=c[keymode]*0.1

for D in destinations:
    restest={}#all results objectives
    resnwsa={}
    #setup to later scatter
    coorxdic={}
    coorydic={}
    scatall= D + ' '
    scat= D
    #setup
    xlspath='output'+D+'.xlsx'
    Tr=[]
    Tr= list(network.nodes()) #number transition nodes
    Tr.remove(O)
    Tr.remove(D)

    destination= []
    destination.append(D)
    origin= []
    origin.append(O)
    Ori_Dest=[D,O]
    #run general

    name='general'
    restest, countries, nadir, utopia=singleobj(name, O, D, countries, c, d, v, C, A, Mij, M
    #run MADM
    name='MADM'
    restest, countries, resnwsa=nwsa(resnwsa,name, O, D, countries, c, d, v, C, A, Mij, MI,
    #run ECM
    number=10

    restest, countries=ecm(number, O, D, countries, c, d, v, C, A, Mij, MI, I, T, N, restest
    #create file with stored results
    dftest=storeresult(restest,utopia,nadir, D)
    resnd=noduplicates(restest,utopia,nadir)
    count, countnwsa, countdf, countnwsadf=proptotal(resnd, restest, resnwsa)
```

```
    dftestnd=dftestnduplicates(resnd, count, countnwsa)
    #scatter
    sort_resnd, coorxdic, coorydic, countrx, country= prepscatall(scat, resnd, coorydic, coo
    #df
    routeprop, routenet=getroute(countries[D], route, resnd)
    routeproptot, routenet2=getroute(countries[D], route, restest)

    df(xlspath, dftest, dftestnd, D, countries, routeprop, routenet, routeproptot, countdf,
    #draw(routenet, scatall)
    #scattercountries
    COUNTCOOX[D]=countrx
    COUNTCOOY[D]=country
    UTOPIA[D]=utopia[D]
    NADIR[D]=nadir[D]



scattercountries(scatcount,destinations, COUNTCOOX, COUNTCOOY,UTOPIA, NADIR)
```