The computational complexity of network analysis can be a significant bottleneck. To improve efficiency, I have developed HAN, a novel Python heuristic algorithm. HAN achieves simplification by aggregating points with similar movement costs into larger square areas, reducing the network size while preserving essential information for accurate analysis.

```
import time
import pandas as pd
import math

tic = time.time()#examine time of processing

###open data from file, csv file must have columns with speed of moving around the area and X Y coordinates
dataa=pd.read_csv('punkty_duzasiec.csv', encoding='utf-8')
przerobione=pd.DataFrame()
iloscdoprzerobienia=len(dataa)
CenterOfSquares = pd.DataFrame(columns=['avg_x', 'avg_y', 'speed', 'cube'])#empty pd dataframe for storing results
print ('success of load data, number of points to processing: %s'%iloscdoprzerobienia)

#################processing point in loop, find squares of point######################
start_point_dim=5
while iloscdoprzerobienia > 0:
    a=0
    dlugosc=1
    firstpoint=dataa.iloc[0]
    XX=firstpoint['X']
    YY=firstpoint['Y']
    while dlugosc == 1:
        XXX=XX+a
        YYY=YY-a
```

```
        queryy="%s >= X and X >= %s and %s >= Y and Y >= %s" %(XXX,XX,YY,YYY)
        block=dataa.query(queryy)
        dlugosc=len(block.speed.unique())#to check whether the speed is the same in full selected square of point data
        iloscc=len(block)#to stop loop
        t=(a/start_point_dim)+1
        tt=a
        if iloscc != t**2:
            break
        if dlugosc == 1:
            blockwyn=block
        a=a+start_point_dim
    if iloscdoprzerobienia == 0:
        break
    listadousuniecia=blockwyn['fid'].values.tolist()
    dataa=dataa[dataa.fid.isin(listadousuniecia) == False]
    iloscdoprzerobienia=len(dataa)
    ###store results in CenterOfSquares pd dataframe
    avg_x=blockwyn.X.mean()
    avg_y=blockwyn.Y.mean()
    speed=blockwyn['speed'].iloc[0]
    cube=tt/2 #cube 1/2 face
    CenterOfSquares.loc[len(CenterOfSquares)]=[avg_x, avg_y, speed, cube]
    print(iloscdoprzerobienia)
toc = time.time()
timeproces=round((toc - tic),2)
print ('succesfull group points to squares, execution time: %s'%timeproces)
#CenterOfSquares.to_csv('CenterOfSquares.csv', index=False)#export data to file



####################find touches squares################################################
#generate lines (last point, first point)
```

```
CenterOfSquares = CenterOfSquares.round({'avg_x':3, 'avg_y':3})

reduce_CenterOfSquares = CenterOfSquares

CenterOfSquares_lines=pd.DataFrame()

for i in range(len(CenterOfSquares)): #len(CenterOfSquares)

    firstpoint=CenterOfSquares.iloc[i]

    Xa=firstpoint['avg_x']

    Ya=firstpoint['avg_y']

    da=firstpoint['cube']

    speed=firstpoint['speed']

    cube=firstpoint['cube']

    Xa__da=Xa+da

    Xa_da =Xa-da

    Ya__da=Ya+da

    Ya_da =Ya-da

    aaa='((%s)==(avg_x-cube) and (((%s)>=(avg_y-cube) and (%s)<=(avg_y+cube)) or ((%s)>=(avg_y-cube) and (%s)<=(avg_y+cube)))) or'%(Xa__da, Ya__da, Ya__da, Ya_da, Ya_da)

    bbb='((%s)==(avg_x+cube) and (((%s)>=(avg_y-cube) and (%s)<=(avg_y+cube)) or ((%s)>=(avg_y-cube) and (%s)<=(avg_y+cube)))) or'%(Xa_da, Ya__da, Ya__da, Ya_da, Ya_da)

    ccc='((%s)==(avg_y-cube) and (((%s)>=(avg_x-cube) and (%s)<=(avg_x+cube)) or ((%s)>=(avg_x-cube) and (%s)<=(avg_x+cube)))) or'%(Ya__da, Xa__da, Xa__da, Xa_da, Xa_da)

    ddd='((%s)==(avg_y+cube) and (((%s)>=(avg_x-cube) and (%s)<=(avg_x+cube)) or ((%s)>=(avg_x-cube) and (%s)<=(avg_x+cube)))) or'%(Ya_da, Xa__da, Xa__da, Xa_da, Xa_da)

    eee='((%s)==(avg_x-cube) and avg_y >= (%s) and avg_y <= (%s)) or '%(Xa__da, Ya_da, Ya__da)

    fff='((%s)==(avg_x+cube) and avg_y >= (%s) and avg_y <= (%s)) or '%(Xa_da, Ya_da, Ya__da)

    ggg='((%s)==(avg_y-cube) and avg_x >= (%s) and avg_x <= (%s)) or '%(Ya__da, Xa_da, Xa__da)

    hhh='((%s)==(avg_y+cube) and avg_x >= (%s) and avg_x <= (%s)) '%(Ya_da, Xa_da, Xa__da)

    query=aaa+bbb+ccc+ddd+eee+fff+ggg+hhh #collection of conditions stating that the designated squares are touched

    block=reduce_CenterOfSquares.query(query)
```

```
    block=block.assign(Xa=Xa)
    block=block.assign(Ya=Ya)
    block=block.assign(speed_a=speed)
    block=block.assign(cube_a=cube)
    reduce_CenterOfSquares =
reduce_CenterOfSquares.drop(reduce_CenterOfSquares.iloc[0].name)
    CenterOfSquares_lines=pd.concat([CenterOfSquares_lines, block])
toc = time.time()
timeproces=round((toc - tic),2)
print ('succesfull generate lines between squares, execution time: %s'%timeproces)
#CenterOfSquares_lines.to_csv('wyn_lines.csv', index=False)#export data to file



####################cutting lines to squares at moving
speed##########################
lines_azimuth=pd.DataFrame()
for i in range(len(CenterOfSquares_lines)):
    line=CenterOfSquares_lines.iloc[i]
    X1=line['Xa']
    Y1=line['Ya']
    d1=line['cube_a']
    speed1=line['speed_a']
    X2=line['avg_x']
    Y2=line['avg_y']
    d2=line['cube']
    speed2=line['speed']
    dy=X2-X1#reversed in relation to geodesy
    dx=Y2-Y1#reversed in relation to geodesy
    if dx==0 and dy>0:
        azymut=90
    elif dx==0 and dy<0:
        azymut=270
    elif dx>0 and dy==0:
```

```
        azymut = 0
    elif dx<0 and dy==0:
        azymut = 180
    else:
        czwartak=math.degrees(math.atan(dy/dx))
        if dx>0 and dy>0:
            azymut = czwartak
        elif dx<0 and dy>0:
            azymut = czwartak + 180
        elif dx<0 and dy<0:
            azymut = 180 + czwartak
        else:
            azymut = czwartak + 360
    azzi=round(azymut,1)#for the definition of special conditions
    #lines at angles 45,135,225 and 315 should be cutting because they are diagonals of
squares and have a connection with other lines
    if (azzi == 45.0 or azzi == 135.0 or azzi == 225.0 or  azzi == 315.0):
        odleglosc=math.sqrt(dy**2+dx**2)
        D=d1*odleglosc/(d1+d2)
        X3=X1+D*math.sin(math.radians(azymut))#odwrotnienizwgeodezji
        Y3=Y1+D*math.cos(math.radians(azymut))#odwrotnienizwgeodezji
        line1='LINESTRING('+str(X1)+' '+str(Y1)+','+str(X3)+' '+str(Y3)+')'
        line2='LINESTRING('+str(X3)+' '+str(Y3)+','+str(X2)+' '+str(Y2)+')'
        line1compl=pd.DataFrame({'geometryWKT': [line1], 'speed':[speed1]})
        line2comp2=pd.DataFrame({'geometryWKT': [line2], 'speed':[speed2]})
        lines_azimuth=pd.concat([lines_azimuth, line1compl])
        lines_azimuth=pd.concat([lines_azimuth, line2comp2])
    elif speed1==speed2:#if moving speed is the same lines are not cutting
        line1='LINESTRING('+str(X1)+' '+str(Y1)+','+str(X2)+' '+str(Y2)+')'
        line1compl=pd.DataFrame({'geometryWKT': [line1], 'speed':[speed1]})
        lines_azimuth=pd.concat([lines_azimuth, line1compl])
    else:
        odleglosc=math.sqrt(dy**2+dx**2)
```

```
        D=d1*odleglosc/(d1+d2)
        X3=X1+D*math.sin(math.radians(azymut))#odwrotnienizwgeodezji
        Y3=Y1+D*math.cos(math.radians(azymut))#odwrotnienizwgeodezji
        line1='LINESTRING('+str(X1)+' '+str(Y1)+','+str(X3)+' '+str(Y3)+')'
        line2='LINESTRING('+str(X3)+' '+str(Y3)+','+str(X2)+' '+str(Y2)+')'
        line1compl=pd.DataFrame({'geometryWKT': [line1], 'speed':[speed1]})
        line2comp2=pd.DataFrame({'geometryWKT': [line2], 'speed':[speed2]})
        lines_azimuth=pd.concat([lines_azimuth, line1compl])
        lines_azimuth=pd.concat([lines_azimuth, line2comp2])

toc = time.time()
timeproces=round((toc - tic),2)
print ('successful line cutting, execution time: %s'%timeproces)
lines_azimuth.to_csv('linieWKT.csv', index=False)#export lines to file CSV
print ('data was saved in a CSV file')
```