还在等外卖？python告诉你，为什么你的外卖这么慢

前言

> 某天中午，小编喜滋滋地点了一份牛肉饭外卖，然后翘首以盼等待配送小哥的到来。半个多小时过去了，软件上的地图显示小哥离我只有三百米的距离，牛肉饭已经近在咫尺。然而左等右等牛肉饭也没有到，再打开app一看，简直两眼发黑：小哥的距离竟然从三百米变成了一千米！

相信大家都曾遇到过这样的问题：外卖点的各种美食，或者跑腿购买的东西，还有淘宝的包裹，明明页面显示它们已经近在咫尺甚至只有几分钟的路程，结果配送小哥非要绕远去别的地方，在家翘首以盼包裹到来的你等到花儿都快谢了，让你无语凝咽：软件上的路线规划完全是人工智障！

然而，好奇心旺盛的小编陷入了沉思，为何这路线规划显得如此智障呢，莫非这里面隐藏着某些不为人知的秘密？这究竟是人性的缺失还是算法的沦丧？

刚好，天池最后一公里配送问题大赛提供了配送机制以及这个问题需要的数据，让我们来一探究竟。

配送机制

我们来看看淘宝的配送机制：

- 配送人员从网点将包裹配送到客户手上
- 每个配送员最多只能携带140个包裹
- 送完所有的包裹回到网点
- 配送点与网点的从属关系固定

数据介绍

老样子，使用pandas读取并观察数据

```
import pandas as pd
tp1=pd.read_csv(r"F:\data\tianchi\peisong\peisongshuju\1.csv",sep=",")
```

```
tp2=pd.read_csv(r"F:\data\tianchi\peisong\peisongshuju\2.csv",sep=",")
tp3=pd.read_csv(r"F:\data\tianchi\peisong\peisongshuju\4.csv",sep=",")
```

```
tp1.head()#网点ID以及对应的经度和纬度
```

	Site_id	Lng	Lat
0	A116	121.226536	31.013124
1	A051	121.746743	31.191404
2	A074	121.490155	31.250216
3	A001	121.486181	31.270203
4	A007	121.640596	31.245883

```
tp2.head()#配送点ID以及对应的经度和纬度
```

	Spot_id	Lng	Lat
0	B9189	121.520966	31.308001
1	B0010	121.451416	31.042305
2	B0021	121.706100	31.382221
3	B0052	121.599289	31.050518
4	B0058	121.885323	31.510674

```
tp3.head()#订单ID，配送点ID，网点ID以及网点需要送至配送点的电商包裹量
```

	Order_id	Spot_id	Site_id	Num
0	F0001	B0214	A001	21
1	F0002	B0288	A001	39
2	F0003	B0587	A001	55
3	F0004	B2027	A001	61
4	F0005	B2946	A001	61

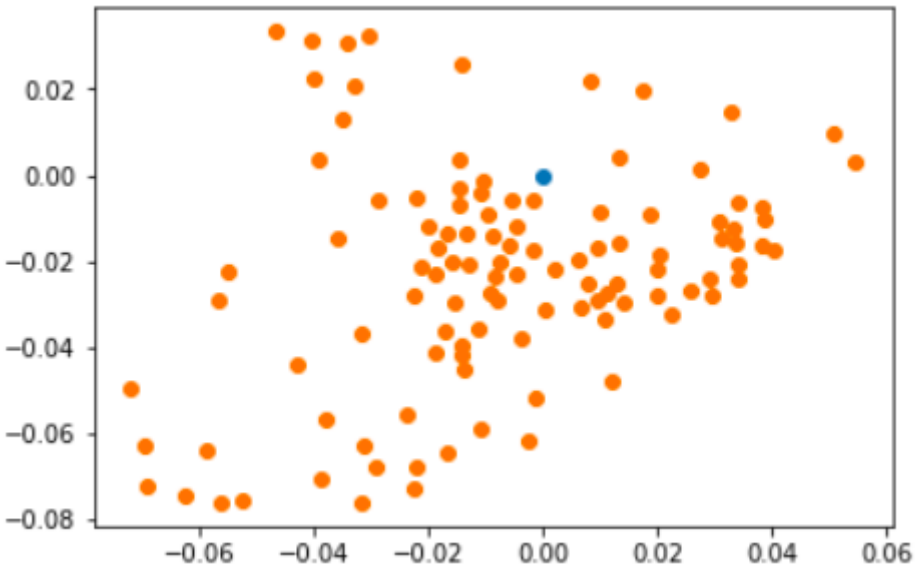
我们来看单个网点的路线规划，先将A117网点数据整合在一张表里

```
#整合A117网点数据
tp1=tp1.set_index('Site_id')
a="A117"
a1=tp3[tp3.Site_id=="A117"]
a1=pd.merge(a1,tp2) #获取对应配送点的坐标
a1.Lng=a1.Lng-tp1.loc["A117"]["Lng"]
a1.Lat=a1.Lat-tp1.loc["A117"]["Lat"]
#以网点为原点，只看配送点与网点之间的相对坐标
a1.head()
```

	Order_id	Spot_id	Site_id	Num	Lng	Lat
0	F8713	B0052	A117	12	-0.017117	0.009326
1	F8714	B0152	A117	38	-0.029332	-0.015618
2	F8715	B0367	A117	15	-0.002888	-0.014543
3	F8716	B0491	A117	10	-0.064145	-0.058930
4	F8717	B0518	A117	36	-0.006451	0.034091

观察配送点与网点的位置关系

```
import pylab
pylab.plot([0],[0],"o")
pylab.plot(a1.Lat.values,a1.Lng.values,"o")
```



如图所示，蓝色的点是网点A117的位置，黄色的点是配送点的位置，配送小哥从蓝色点出发，把包裹送到黄色点，每次携带的包裹不大于140个。当携带的包裹配送完后，配送小哥需要再次返回到网点取包裹。

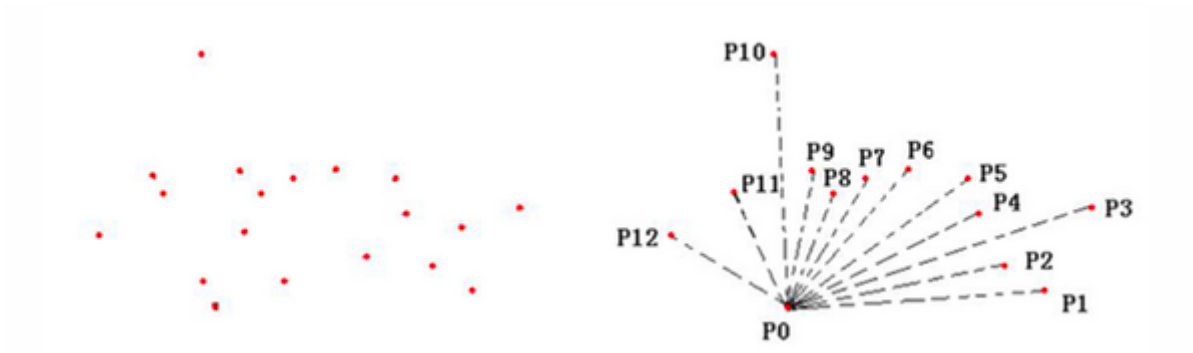
路线规划所考虑的问题是：怎么走才能使配送小哥走的路程最短呢？

为了简化起见，我们将经纬度下的曲线距离用直线距离来代替。

路线规划一

基于点的角度顺序配送

物流配送路径优化问题是一个很经典的问题，针对该问题有很多的解决方法。基于点的角度顺序配送是一个比较简单且运行良好的算法。。



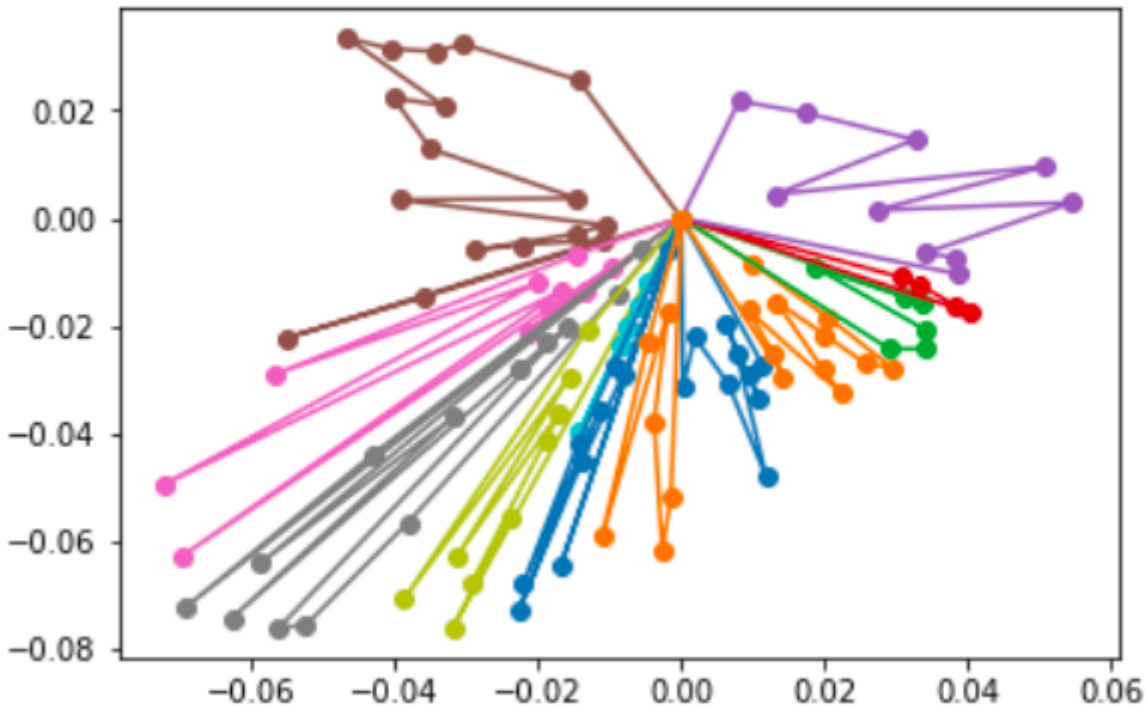
如上图所示，P0为起始点，其它点为配送需求点。

1. 采用极坐标来表示各点的相对位置，然后以P0点为坐标原点，以P1为起始点，定其角度为零度，以顺时钟或逆时钟方向开始扫描各个点，获得各点与原点连线P0Pn相对于P0P1的角度大小。
2. 根据角度大小确定其顺序，直至扫描完毕。
3. 扫描结束后获得的点的序列就是各点的配送顺序。

了解了算法原理，我们来试验一下，A117这个网点的配送顺序是如何的。

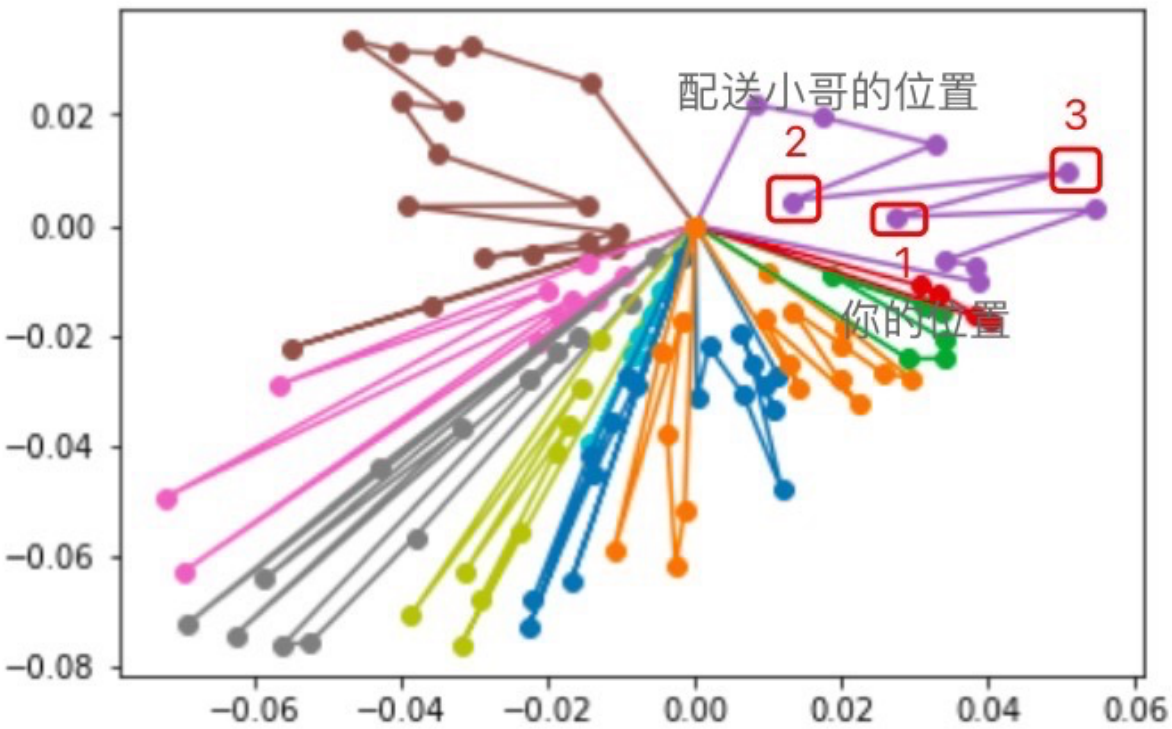
下面，就开始路线规划啦

```
a1=pd.concat([a1,a1.Lng/a1.Lat],axis=1) #计算各个配送点的正切值
a1.rename(columns={0:"zhengqie"},inplace=True)

a1=pd.concat([a1,(a1.Lng**2+a1.Lat**2)**0.5],axis=1)#计算各个配送点与中心点的
距离
a1.rename(columns={0:"r"},inplace=True)

lu1=a1[a1.Lat>0].sort_values(["zhengqie"])
lu2=a1[a1.Lat<0].sort_values(["zhengqie"])
lu=pd.concat([lu1,lu2])

zz=0
x=[0]
y=[0]
ju=0
lat=0
lng=0
for i in lu.iterrows():
    zz=zz+i[1]["Num"]
    if zz>140:
        x.append(0)
        y.append(0)
        pylab.plot(x,y,"-o")
        ju=ju+(lat**2+lng**2)**0.5 #加入回程距离
        lat=0
        lng=0
        x=[0]
        y=[0]
        zz=i[1]["Num"]
    ju=ju+((i[1]["Lat"]-lat)**2+(i[1]["Lng"]-lng)**2)**0.5#加入了路径长度的计
算
    lat=i[1]["Lat"]
    lng=i[1]["Lng"]
    x.append(lat)
    y.append(lng)

ju=ju+((0-lat)**2+(0-lng)**2)**0.5
x.append(0)
y.append(0)
print(ju)
pylab.plot(x,y,"-o")
pylab.show()
```



通过这份路线规划图，就不难明白配送小哥为何会绕远了。

以顺时针方向进行配送
假设你在图中1的位置，
配送小哥正在2的位置进行配送，
按照实际距离来讲，快递小哥离你最近，下一个应该首先为你进行配送，但是根据角度大小来规划路线，快递小哥却去了更远的3这个位置！



我饿着肚子痴痴地等待着我的牛肉饭，明明我是最近的，却让我白白地多等了半小时，就因为我家地理位置的正切值比别人家的大？这路线规划一点儿也不合理！别急，我们再来看看另一种路线规划方法。

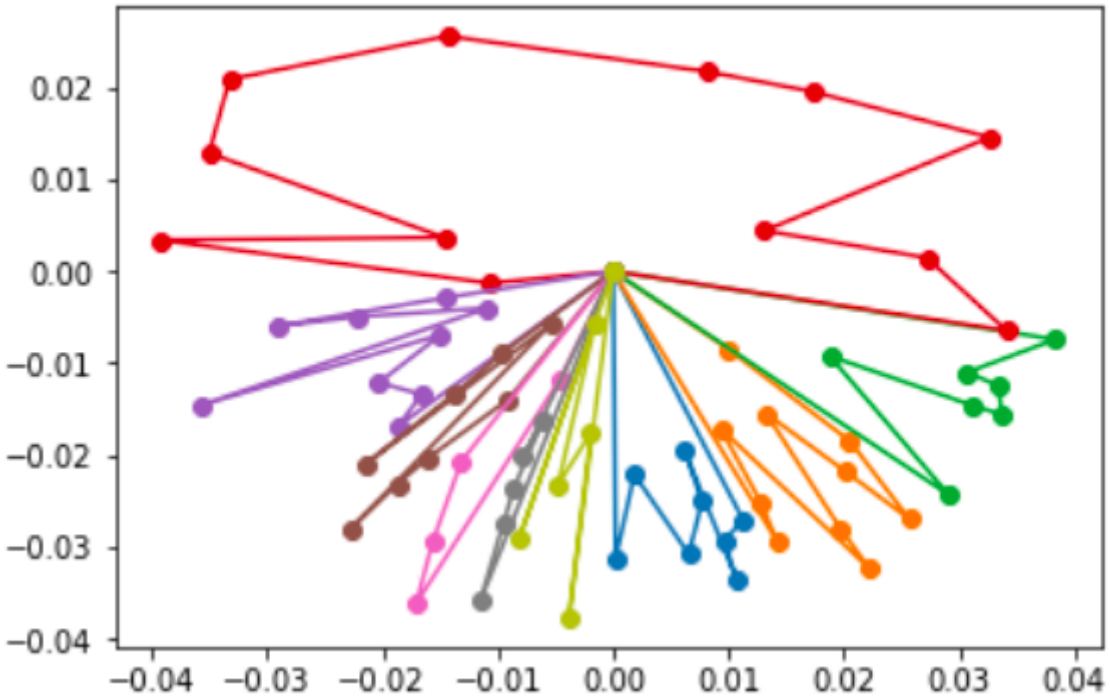
路线规划二

环形扫描法

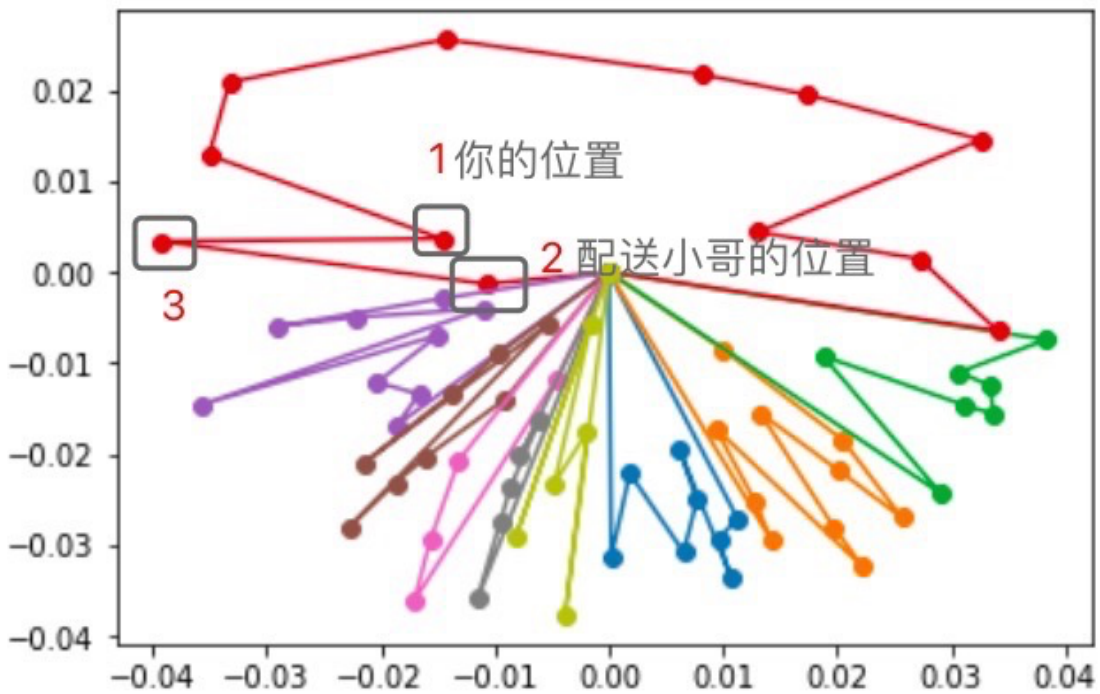
由于仅仅按照角度顺序分配，导致径向距离来回的浪费。因此我们把点分为不同径向长度的环，环内按照角度排序，减少径向行走。

```
def c1(a):#网点与配送点的数据准备
    a1=tp3[tp3.Site_id==a]
    a1=pd.merge(a1,tp2)
    a1.Lng=a1.Lng-tp1.loc[a]["Lng"]
    a1.Lat=a1.Lat-tp1.loc[a]["Lat"]
    a1=pd.concat([a1,a1.Lng/a1.Lat],axis=1)
    a1.rename(columns={0:"zhengqie"},inplace=True)
    a1=pd.concat([a1,(a1.Lng**2+a1.Lat**2)**0.5],axis=1)
    a1.rename(columns={0:"r"},inplace=True)
    return a1

def c2(a1):#角度排序
    lu1=a1[a1.Lat>0].sort_values(["zhengqie"])
    lu2=a1[a1.Lat<0].sort_values(["zhengqie"])
    return pd.concat([lu1,lu2])

def c3(lu):#按照最大140的负荷布置路径
    zz=0
    x=[0]
    y=[0]
    ju=0
    lat=0
    lng=0
    for i in lu.iterrows():
        zz=zz+i[1]["Num"]
        if zz>140:
            x.append(0)
            y.append(0)
            pylab.plot(x,y,"-o")
            ju=ju+(lat**2+lng**2)**0.5
            lat=0
            lng=0
            x=[0]
            y=[0]
            zz=i[1]["Num"]
        ju=ju+((i[1]["Lat"]-lat)**2+(i[1]["Lng"]-lng)**2)**0.5
        lat=i[1]["Lat"]
        lng=i[1]["Lng"]
        x.append(lat)
        y.append(lng)

    ju=ju+((0-lat)**2+(0-lng)**2)**0.5
    x.append(0)
    y.append(0)
```

```
    pylab.plot(x,y,"-o")
    pylab.show()
    return ju
p1=c1(a) #数据处理
r0=p1.r.mean()#配送点与网点的平均距离
h0=p1[p1.r<r0]
p2=c2(h0)
p3=c3(p2)
print(p3)
```



使用环形扫描法比基本的扫描法效果好了很多。配送小哥少跑了很多冤枉路，但是可以看到，它仍然无法完全解决你的困扰！



若为顺时针扫描，假设你在位置1，配送小哥正在位置2，虽然你距离他很近，但仍然，他需要先到达更远的位置3，再到你的位置！

事实上，通过其他算法也可以得到或近似得到配送小哥的最短路径规划方案。但不论如何，配送小哥距离你的位置最近就一定会为你最先配送吗？答案是否定的！毕竟考虑到配送小哥工作量如此之大，在家嗷嗷待哺的我们就稍微耐心一点吧！