基于隐马尔可夫模型和维特比算法的地图匹配算法

语言

Python

备注

我可以提供数据，但我完全不会隐马尔可夫模型和维特比算法，解答不了技术问题

要求

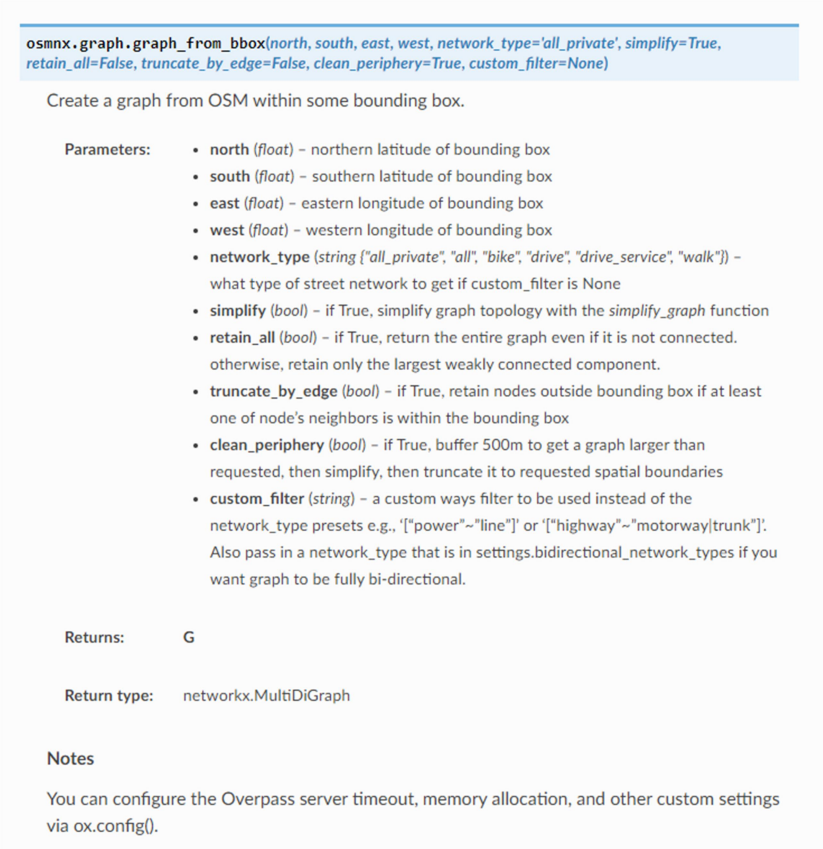
1、有程序的**详细详细再详细的**解释和说明，保证我能完全看懂整个程序
2、三个数据的程序要分开，不要写在一个程序里
3、有一定的泛用性，保证我换了数据也可以运行出正确结果
4、4 月 30 日（含）之前完成

主要算法

隐马尔可夫模型和维特比算法

1、获取地图数据

地图选取范围：比所给轨迹数据的最大经纬度大 0.001，比最小经纬度小 0.001 的矩形范围
最好使用下面这个函数来获取：



osmnx.graph.graph_from_bbox(*north, south, east, west, network_type='all_private', simplify=True, retain_all=False, truncate_by_edge=False, clean_periphery=True, custom_filter=None*)

Create a graph from OSM within some bounding box.

Parameters:

- **north** (*float*) – northern latitude of bounding box
- **south** (*float*) – southern latitude of bounding box
- **east** (*float*) – eastern longitude of bounding box
- **west** (*float*) – western longitude of bounding box
- **network_type** (*string {"all_private", "all", "bike", "drive", "drive_service", "walk"}*) – what type of street network to get if custom_filter is None
- **simplify** (*bool*) – if True, simplify graph topology with the *simplify_graph* function
- **retain_all** (*bool*) – if True, return the entire graph even if it is not connected. otherwise, retain only the largest weakly connected component.
- **truncate_by_edge** (*bool*) – if True, retain nodes outside bounding box if at least one of node's neighbors is within the bounding box
- **clean_periphery** (*bool*) – if True, buffer 500m to get a graph larger than requested, then simplify, then truncate it to requested spatial boundaries
- **custom_filter** (*string*) – a custom ways filter to be used instead of the network_type presets e.g., '["power"~"line"]' or '["highway"~"motorway|trunk"]'. Also pass in a network_type that is in settings.bidirectional_network_types if you want graph to be fully bi-directional.

Returns: G

Return type: networkx.MultiDiGraph

Notes

You can configure the Overpass server timeout, memory allocation, and other custom settings via ox.config().

https://osmnx.readthedocs.io/en/latest/osmnx.html#module-osmnx.graph

2、建立路网有向图

路网数据一般为 shapefile 格式，通过读取 shp 文件和 dbf 文件，获取道路的节点 ID、坐标信息及属性信息，从而建立路网的有向图 $G(V, E)$，其中 V 的元素为道路端点，E 的元素为道路路段。常用的描述有向图的方法有邻接矩阵和邻接链表。考虑到交通路网相对稀疏，邻接链表占用内存空间较少，本文采用邻接链表描述有向图。

3、轨迹简化

$$PED_{p_j|(p_{Anchor},p_i)} = \frac{|(y_i-y_{Anchor})x_j-(x_i-x_{Anchor})y_j+x_iy_{Anchor}-x_{Anchor}y_i|}{\sqrt{(y_i-y_{Anchor})^2+\left((x_i-x_{Anchor})\right)^2}}$$

算法 2：TSSW

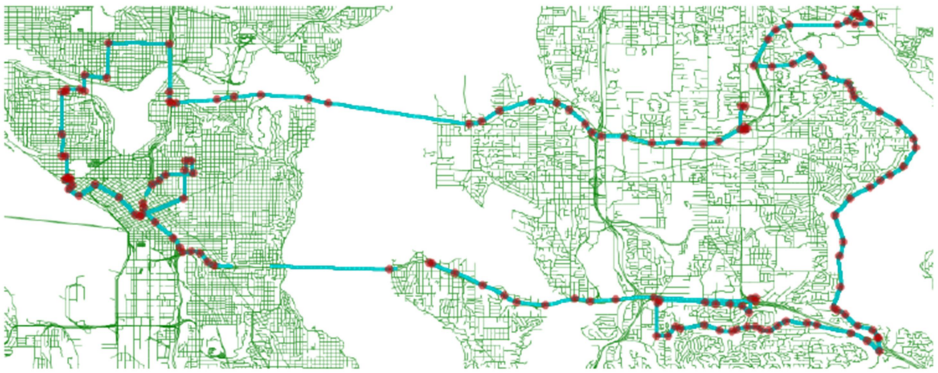
输入：轨迹T: $p_1 \to p_2 \to \cdots \to p_n$，距离阈值 *theta*

输出：简化轨迹T': $p_1 \to p_i \to \cdots \to p_n$

```
1. T' = [p_1]                                  # 初始化存储简化轨迹的列表
2. Anchor=p_1                                  # Anchor 表示关键点
3.for i in (2,n)                               #遍历原始轨迹
4.    SlidingWiondow=list(Anchor, p_{i-1})     #转化成滑动窗口
5.    for p_j in SlidingWiondow
6.        if PED_{p_j|(p_Anchor,p_i)} >theta   #如果滑动窗口内存在 PED 大于阈值
7.            Anchor=p_{i-1}                   #定义p_{i-1}为 Anchor,并更新窗口
8.            T'.append(Anchor)
9.            break
10. T'.append(p_n)
11. return T'
```

阈值 *theta* 取 20m，如果编程用的是经纬度，具体值是多少您自己换算一下

此处需要输出一张轨迹简化的图，即分别在地图上标出未简化的轨迹点和简化后的轨迹点，类似下图



蓝色是原始轨迹，红色是简化后的轨迹

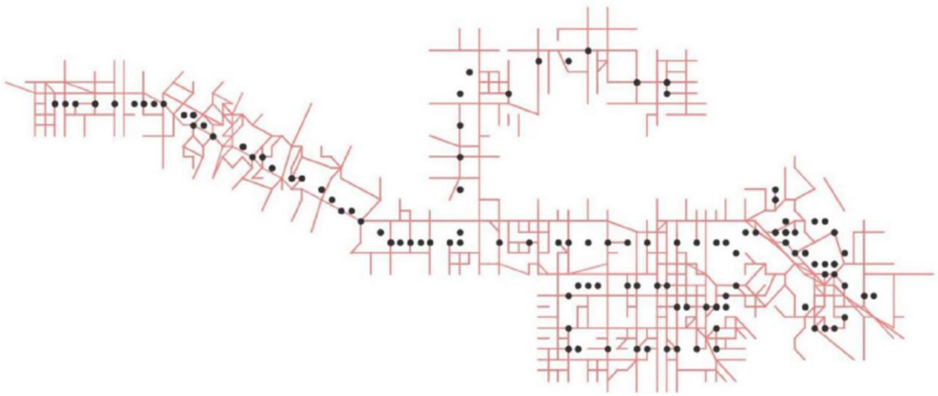
3、候选路段选取

以轨迹点为圆心，500 米为半径画圆，保留所有通过这个圆的路段作为候选路段。

这一步可以考虑计算轨迹点到路段的距离，小于 500 米的保留

每一个轨迹点 p 对应一个候选路段集 L

此处需要对结果进行可视化，类似下图



4、隐马尔可夫模型

发射概率

p_i 是轨迹点，l_i 是候选路段，k 是轨迹点 p_i 对应的候选路段集 L_i 中的候选路段数量

β 是我给的数据里的方向， θ 是地图数据里路段的方向

$$P(p_i|l_i^k)=P_d(p_i|l_i^k)P_\alpha(p_i|l_i^k)$$

$$P_d(p_i|l_i^k)=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\|p_i-c_i^k\|^2}{2\sigma^2}\right), \quad (1)$$

其中，σ 表示 GPS 的误差的标准差，一般取值为 20 m；$\|p_i-c_i^k\|$ 代表从轨迹点至候选路段的大圆距离。

$$P_\alpha(p_i|l_i^k)=(1+\cos\alpha_i^k)/2, \quad (2)$$

其中，α_i^k 为速度方向与匹配路段的夹角。以正北方向为基准，计算速度和匹配路段与正北方向的夹角分别为 β_i 和 θ_i^k，则 α_i^k 的计算公式为

$$\alpha_i^k=\begin{cases}\left|\beta_i-\theta_i^k\right|, & \left|\beta_i-\theta_i^k\right|<180^\circ,\\ 360^\circ-\left|\beta_i-\theta_i^k\right|, & \left|\beta_i-\theta_i^k\right|\geq 180^\circ。\end{cases} \quad (3)$$

传递概率

$$P(l_{t+1}^j|l_t^i)=\exp(-\mu\mathrm{Dis})$$

Dis 代表从候选路段间的实际路网距离，这个应该可以用 osmnx 的 shortest_path 来求， μ =5

5、维特比算法求解

我不会，基本逻辑和代码需要您自行解决，最终要输出一个路段序列 R

伪代码参考这个，但我看不懂

地图匹配使用的维特比算法主要包括两部分：最优路径终点的获取和回溯。算法的伪代码如下。

输入：轨迹数据 $P=(p_n \mid n=1,\ldots,N)$

输出：匹配的路段 $R=(r_n \mid n=1,\ldots,N)$

令 func[ ]为当前状态对应的联合概率，prev[ ]为当前状态的前一个状态

for$(i=1, i<N-1; i++)$

  获取 p_i 对应的候选路段集 L_i

  获取 p_{i+1} 对应的候选路段集 L_{i+1}

  for each l_1 in L_{i+1}

    计算 $P(p_{i+1}|l_1)$

    for each l_2 in L_i

      if i=1 //初始化

        计算 $P(p_i|l_2)$; func$[l_2]=P(p_i|l_2)$；

      计算传递概率 $P(l_1 \mid l_2)$

    func$[l_1]=P(p_{i+1}|l_1)\max_{l_2\in L_i}\{\text{func}[l_2]P(l_1 \mid l_2)\}$

    $k=\text{argmax}_{l_2\in L_i}\{\text{func}[l_2]P(l_1 \mid l_2)\}$

    prev$[l_1]=k$

  End $=\text{argmax}_{l\in L_n}\{\text{func}[l]\}$ //获取全局最优路径的终点

  R. append (End)

  for $(i=1, i<N-1; i++)$

    End=pre[End]

    R. append (End) //回溯过程

  R. reverse ( )

  return R

准确率

$$ac=\frac{\textit{正确匹配的路段数量}}{\textit{原始轨迹中的路段数量}}$$

其中，分子部分比较一下上面求出来的路段序列 R 和原始轨迹，把能对应起来的路段数量统计一下，就是分子

输出

1、匹配结果的可视化，即地图上要用不同颜色画出 GPS 轨迹点、原始轨迹和匹配结果

2、准确率