基于轨迹分类的信号灯周期估计

摘要

本文主要研究了路口信号灯周期估计的问题，利用 DBSCAN 聚类算法、统计分析等方法，结合基于数学知识的机理分析，建立起路口信号灯周期的估计模型，能够较为准确的估计周期，这对于现实生活中城市交通的正常运作有着重要意义。

针对问题一，首先进行数据整理，包括轨迹整理与标准化。通过可视化轨迹信息，观察到路口形状不同，因此计算车辆瞬时速度与加速度作为新特征。其次通过设定速度和加速度阈值判断车辆的运动状态，并且提取关键的时间节点。接着将正则化处理后的时间数据输入 DBSCAN 模型，结合人工评估进行同周期车流聚类。最后对每波车流计算得到的红灯时长和信号灯周期总长使用统计分析的方法，确定最可能红灯时长和绿灯时长。

针对问题二，首先将定位误差、样本车辆比例、车流量因素分别加入模型中，定位误差以 x、y 坐标添加随机误差形式考虑，样本车辆比例以随机抽取车辆的形式考虑，车流量以速度、加速度、路口区域的阈值变化形式考虑，其次以平均误差为指标分析各因素对模型估计精度的影响，发现定位误差、样本车辆比例对模型估计精度影响较小，车流量影响相对较大。最后将三因素影响同时考虑，计算附件 2 中各路口的信号灯周期。

针对问题三，首先用 DBSCAN 算法对同周期车流聚类，用问题二中考虑影响因素后的模型，得到各周期信号灯总时长并按时间先后进行排序。其次考虑到相邻周期内可能无车通过，对过长的信号灯周期进行拆解，得到新的排序结果。最后通过局部加权多项式回归对排序结果进行处理，找出每个路口中信号灯变化的拐点，以拐点将总时间分段后找到每段的最大值作为该段红灯时长，并判断信号灯时长变化的可能原因为车流量影响。

针对问题四，首先求出该路口所有可能的行车方向，可视化后发现该路口为一个十字路口。其次尝试以所有车辆的 x、y 坐标为特征使用 DBSCAN 聚类，由于数据量过大与参数设置等问题，发现效果并不好，所以我们采用“起始点+起终点斜率”的判断方式进行统计分类，最终发现一共有 12 种行车方向，我们将这 12 类演变成为 12 种路口，采用前三问的模型进行求解。

关键字：DBSCAN 聚类算法；统计分析；数理分析；周期估计

目录

1. 问题重述

1.1 问题背景

电子地图与电子导航极大程度地便利了人们的生活，电子地图不仅提供了详细的地理信息，还能实时更新路况，帮助用户避开拥堵路段，选择最佳出行路线。一些电子地图服务商在更新道路信息的过程中面临着一些问题，其中包括获取城市路网中所有交通信号灯的红绿周期。由于许多路口的信号灯并未接入网络，因此无法从交通管理部门获取相关信息。为了解决这一问题，我们尝试通过建立数学模型，利用已有的行车轨迹数据来估计不同路口的交通信号灯周期。这种方法基于大量车辆的行驶数据，通过分析车辆在路口的行驶速度、停留时间等信息，来推断出信号灯的变化规律。通过这种方式，电子地图服务商能够更准确地掌握城市路网中的交通信号灯周期，进而为用户提供更精确的导航服务。这不仅有助于减少用户在路上的等待时间，提高出行效率，还能在一定程度上缓解城市交通拥堵问题。因此，建立数学模型估计交通信号灯周期是一项具有重要意义的工作。

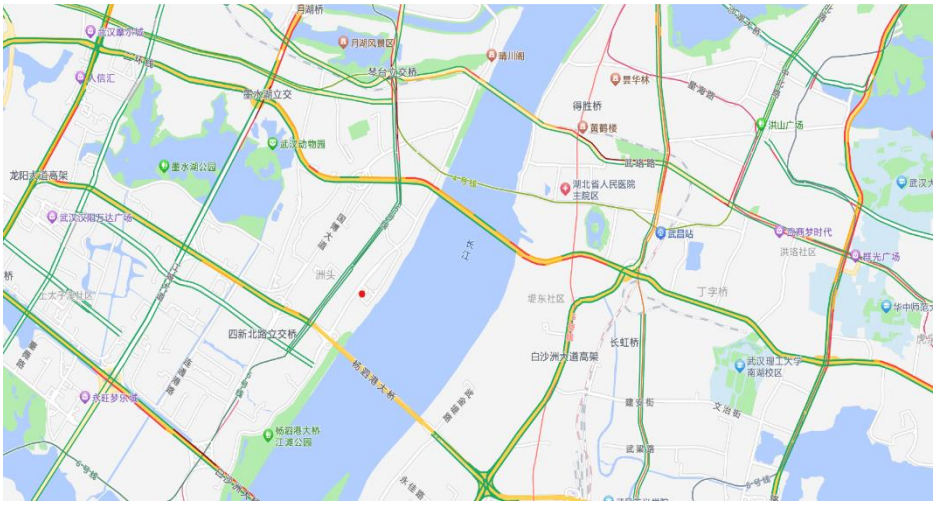

图 1　高德电子地图(开启显示路况)

1.2 问题回顾

本文具体要解决的问题如下：

问题一：根据附件 1，对数据进行分析和处理，假设信号灯周期固定不变，利用车辆行车轨迹数据，建立模型估计附件 1 中各路口信号灯的红绿周期。

问题二：讨论样本车辆比例、车流量、定位误差等因素对问题一所建立的模型估计精度的影响。将考虑影响因素后的模型应用于附件 2，求出其中各路口的信号灯周期。

问题三：建立模型，由附件 3 中的样本车辆轨迹数据，判断附件 3 中各路口信号灯周期在指定时间段内是否发生变化，如果发生变化，求出新旧周期参数，

并指明识别出周期变化所需的时间和条件。

问题四：建立模型，通过某路口连续 2 小时内所有方向样本车辆的轨迹数据，识别出该路口信号灯的周期。

2. 模型假设

为方便模型的建立与模型的可行性，我们这里首先对模型提出一些假设，使得模型更加完备，求出的结果更加合理。

（1）假设给出的数据均为真实数据，真实有效。

（2）假设对于一些较为异常的数据的出现具有一定的合理性。

（3）假设道路中没有交通事故，所有车辆均在正常运行。

（4）假设所有车辆车型相同，不考虑车长影响。

（5）假设车辆变道时不花费时间，即变道与速度变化在一秒中同时进行。

（6）假设定位产生的坐标误差大小是随机的。

（7）假设车辆停止期间不产生定位误差。

3. 符号说明

为了方便我们模型的建立与求解过程，我们这里对使用到的关键符号进行以下说明：

表 1　符号说明表

符号	符号说明
x_{it}	同一路口的第i辆车t时刻的x坐标
y_{it}	同一路口的第i辆车t时刻的y坐标
v_{it}	同一路口的第i辆车t时刻的速度
a_{it}	同一路口的第i辆车t时刻的加速度
v_0	判断状态节点的速度阈值
a_0	判断状态节点的加速度阈值
L_{it}	定义的距离变量，距信号灯的距离
k	判断是否进入路口区域的距离阈值
LR	每波车流遇到的红灯时长
ε_x	x坐标的坐标误差
ε_y	y坐标的坐标误差
x^*	修正后的x坐标
$\overline{\varepsilon_{rp}}$	平均相对误差

(注：这里只列出本文各部分通用符号，个别模型单独使用的符号在首次引用时会进行说明)

4. 问题一模型建立与求解

4.1 问题分析

针对附件 1 所给数据，首先将数据根据车辆 ID 进行有序整理，经数据预处理发现没有缺失值和明显异常值；根据车辆轨迹信息做散点图，观察到路口形状不同，车辆轨迹信息不统一，通过计算欧氏距离我们可以得到车辆瞬时速度，进而得到车辆瞬时加速度，作为新的轨迹特征写入附件一，得到路口形状标准化后的数据集。

针对问题一，本文首先通过设定的速度和加速度阈值判断车辆在某一时刻的运动状态，并且提取关键的时间节点；接着将正则化处理后的时间数据作为数据特征输入 DBSCAN 模型，结合人工评估进行同周期车流聚类；而后，对于每波车流计算得到的红灯时长和信号灯周期总长，使用统计分析的方法，确定最可能红灯时长和绿灯时长。

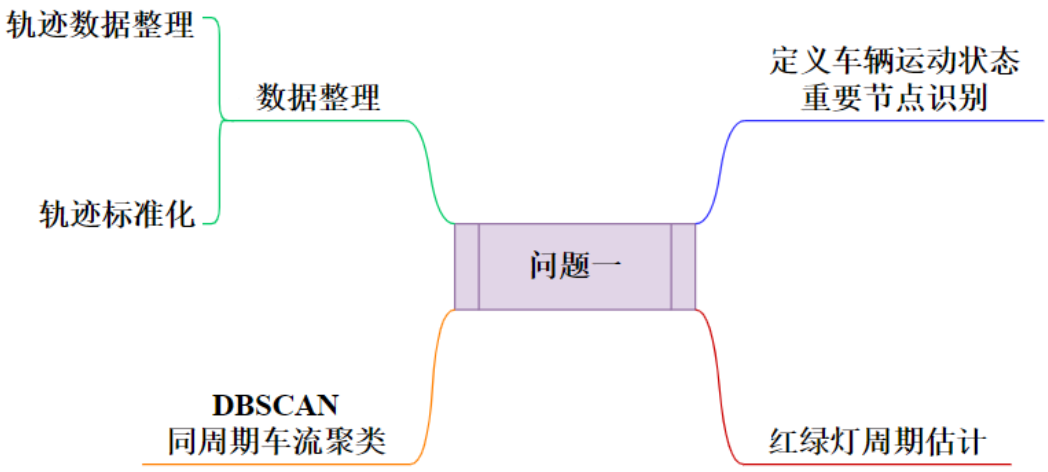

图 2　问题一思路图

4.2 数据整理

4.2.1 轨迹数据整理

首先，本文将车辆轨迹信息根据车辆 ID 进行分组，通过将数据可视化与检验判断出数据没有缺失值，进而通过绘制箱型图检验异常值，发现可能的异常值点后，结合实际情况进行判断，最终认为各个路口的所有车辆数据没有异常值。

具体做出以下处理：

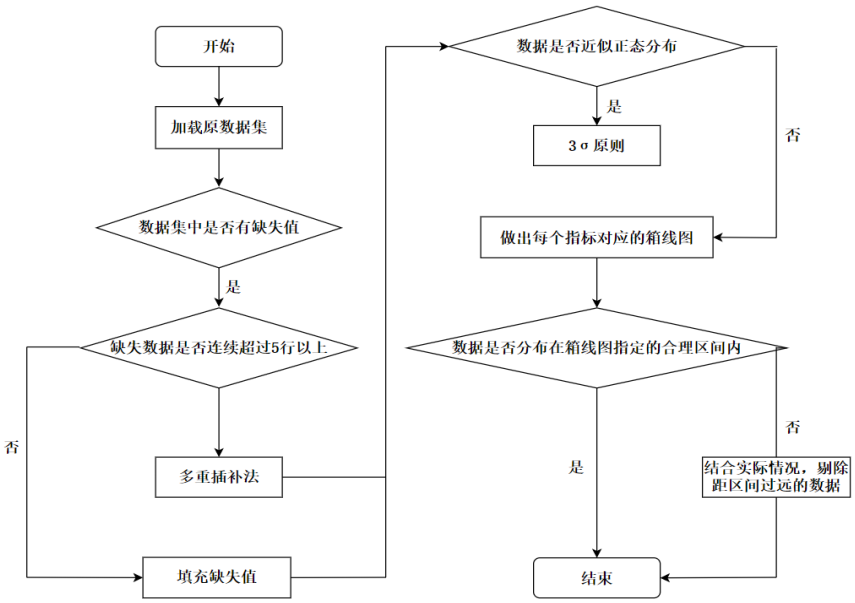

图 3　数据预处理流程图

将 1 小时内通过的所有车辆进行分组，对每一辆车分别进行缺失值和异常值检验。以路口 A3、ID 为 23 的车辆为例，绘制出其轨迹图如下图 3。

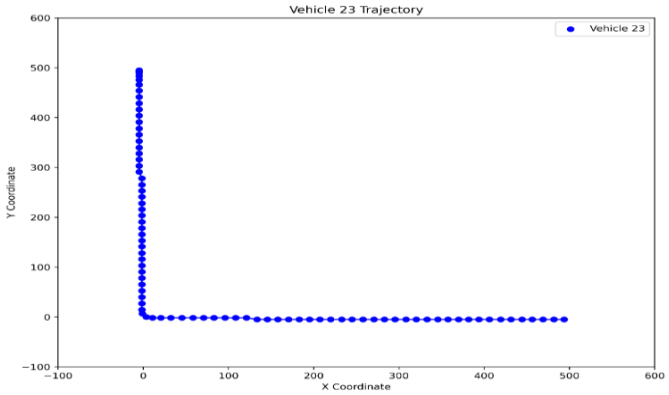

图 4　路口 A3 中 ID23 车辆的轨迹图

从散点中可清晰看出，数据中没有缺失值，再观测原数据发现确实没有缺失值。同理，运用上述方法检验认为五个路口所有车辆数据均不存在缺失值。

其次对数据进行异常值检验，首先通过 Shapiro-Wilk 检验数据的正态性，发现路口 A3、ID 为 23 的车辆的 X、Y 数据均未通过检验，无法使用3σ原则进行检验，所以考虑采用箱线图检验。

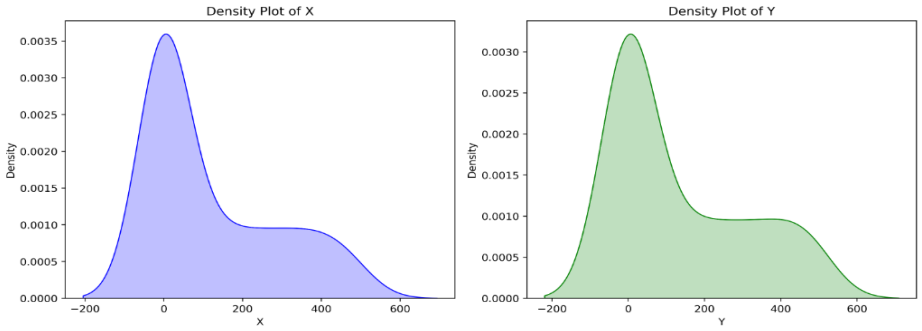

图 5　ID23 车辆的 X、Y 数据的 S-W 检验结果图

绘制得到该车辆 X、Y 的箱线图如下图 x，发现并没有异常点，采用上述相同方式判断检验认为五个路口的所有车辆数据均无异常值。

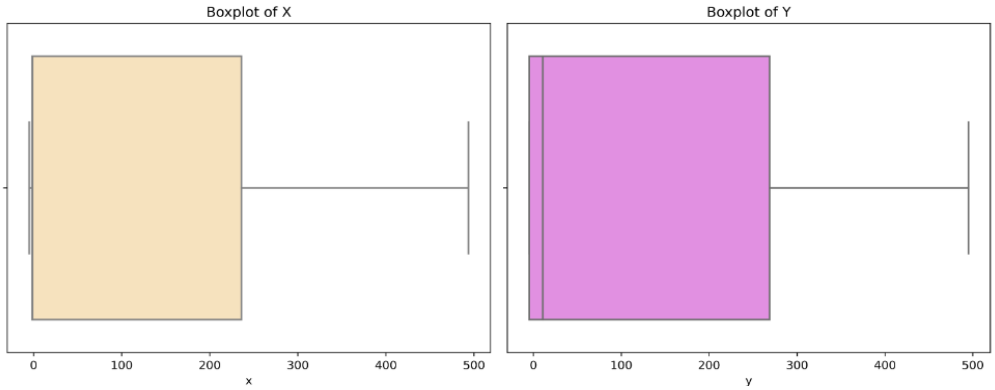

图 6　ID23 车辆的 X、Y 数据的箱线图

4.2.2 轨迹标准化

不同路口的形状不同，车辆在其中行驶的轨迹也就不同，因此需要将已有车辆轨迹信息转换为速度，便于统一度量。对于车辆i在某一时点t处的位置坐标(x_{it}, y_{it})和下一时点处的位置坐标(x_{it+1}, y_{it+1})，我们可以计算出其在t时的速度v_t为：

$$v_{it} = \frac{\sqrt{(x_{it+1} - x_{it})^2 + (y_{it+1} - y_{it})^2}}{(t+1) - t} \tag{1}$$

在计算得到时点处的速度后，我们可以相应地计算出$t+1$时的加速度a_{t+1}为:

$$a_{i(t+1)} = \frac{v_{i(t+1)} - v_{it}}{(t+1) - t} \tag{2}$$

4.3 信号灯的红绿周期估计

4.3.1 基于车辆运动状态的时间节点判断

（1）变量选择

结合数据整理后得到的车辆速度与加速度数据，我们可以将某车在路口的运动状态分为以下四类：正常行驶、开始减速、停止和加速。定义如下：

$$\begin{cases} \text{正常行驶：} v_{it} \geq v_0, |a_{it}| \leq |a_0| \\ \textbf{开始减速：} a_{it} \leq a_0 \\ \textbf{车辆停止：} v_{it} = 0, a_{it} \leq 0 \\ \textbf{车辆加速：} v_{it} > 0, a_{it} > 0 \end{cases}$$

其中，v_t代表t时刻该车的速度，a_t代表t时刻该车的加速度，a_0为设定的加速度阈值，v_0为设定的速度阈值。a_0与v_0作为可变参数，可结合不同路口的实际情况进行调整。

之后我们引入距离变量L_i，

$$L_{it} = \sqrt{(x_{it} - x_0)^2 + (y_{it} - y_0)^2} \tag{3}$$

其中，x_{it}、y_{it}与上文定义相同，(x_0, y_0)为信号灯的位置坐标，我们通过设距离阈值k，即当$L_{it} < k$时，可认为车辆进入了路口区域。

（2）节点判断

借助定义的运动状态与路口区域，我们可以判断出以下重要节点：

在$L_{it} < k$时，若a_{it}初次$\leq -a_0$，则将此时的t记为车辆减速时点$Time_point1$，可以认为此时信号灯由绿转红，驾驶员开始减速；

若v_{it}初次$= 0, a_{it} \leq 0$，则将此时的t记为车辆停止时点$Time_point2$，可以认为车辆减速到 0 开始等待；

若v_{it}、a_{it}最后一次$= 0$，则将此时的t记为$Time_point3$即起步时间，信号灯由红转绿。

因此可粗略认为一个信号灯的红灯时长为$Time_point3 - Time_point1$。

由于以上只是最理想状态下的简化模型，未考虑实际道路情况下路况、天气以及其他交通参与者的影响，因此我们需要根据已有数据进一步判断信号灯的红绿周期。

4.3.2 基于 DBSCAN 算法的同周期车流聚类

基于信号灯周期固定不变的前提，我们可以将相近时间段内路口处的车辆视为一波车流，将其作为研究对象以估计红绿灯周期。在一个红绿周期内，我们可以认为一波车流中的车的状态只有两种情况：分别是减速至停止再启动和不减速通过路口。通过分析这两类车辆出现的时点差值，我们可以大致推断信号灯红绿转换的规律。因此，使用聚类方法将一波车流中的车辆一起考虑是可行的。此外，在信号灯周期未知的情况下，1 小时内有多少波车流通过该路口也未知，也就相当于簇的数量未知，因此不能选择传统的聚类算法。

为移除时间跨度的影响，本文将上文中提到的$Time_point1$、$Time\_point2$、$Time_point3$进行正则化处理后转换为均值为 0，标准差为 1 的标准正态分布，采用 DBSCAN 算法进行聚类。

（1）算法介绍

DBSCAN（Density-Based Spatial Clustering of Applications with Noise）是一种基于密度的聚类算法，其主要思想是将密度高于某一阈值的区域视为簇，并能够在高密度区域中发现任意形状的聚类。

DBSCAN 算法通过两个核心参数来定义聚类的密度：邻域半径(ε)和最小样本数$MinPts$。邻域半径(ε)定义了点的邻域范围。如果一个点 A 在点 B 的ε邻域内，我们称点 A 是点 B 的邻居；最小样本数（$MinPts$）定义了一个点的ε邻域

内至少需要包含多少个点，当点数大于$MinPts$时该点才能被视为核心点。其工作流程如下图 6。

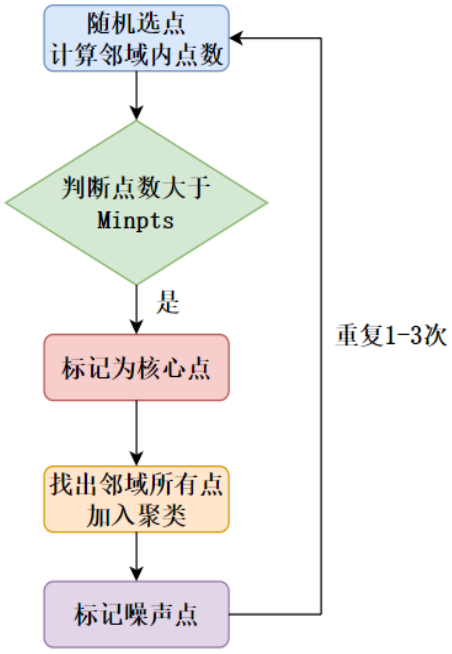

图 7　DBSCAN 工作流程

（2）聚类结果

本文对五个路口选择的 eps 、$MinPts$ 参数空间分别为(0.05,0.06,0.11,0.1,0.1)、(2,1,1,1,1)，通过使用随机搜索的方法在给定空间内搜索最优参数组合，以轮廓系数作为组合的优度评估指标。

轮廓系数是一种用于评估聚类质量的指标，其取值范围在$[-1,1]$之间，接近 1 表示样本点聚类得越好；接近-1 表示样本点更倾向于被分配到错误的簇中；接近 0 表示样本点位于两个簇的边界上。本文计算得到五个路口的轮廓系数值分别为 0.81、0.85、0.89、0.91、0.93，说明样本点聚类结果较好。

基于随机搜索得到的结果，使用散点图对聚类结果进行可视化处理，以及根据不同簇的特征进行人工评估。通过反复实验和验证，我们可以对聚类算法的参数进行不断调整，得到最优的同周期车流聚类结果。部分路口的聚类结果如下：

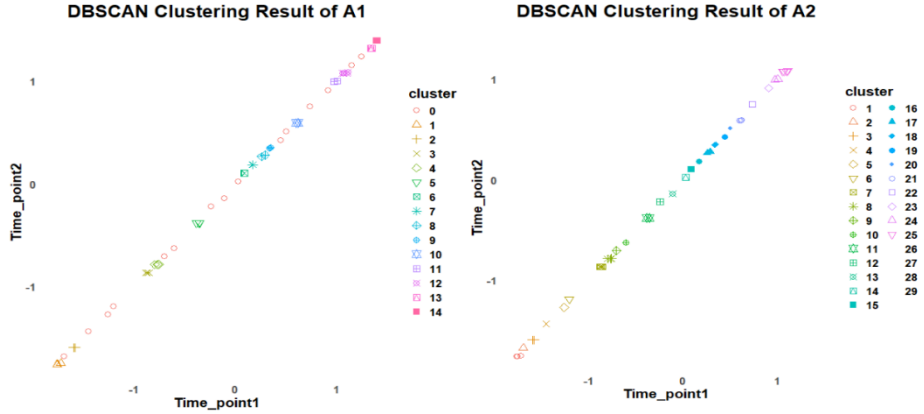

图 8　A1、A2 的聚类结果

4.3.3 模型建立

在完成车流的聚类后，对每一波车流，我们都可以得到其对应的红灯时长$Length\ of\ Red\ (LR)$；通过不同波次车流，可以得到两波车流之间的差值$Difference(D)$：

$$LR = Time\_point3_i - Time\_point1_i \quad (4)$$

$$D = Time\_point3_{i+1} - Time\_point3_i \quad (5)$$

其中i代表车流的波次，$i+1$即为下一波车流。

对每波车流得到的相关时长进行统计分析以了解数据的分布情况。我们选取路口 A1 的车流聚类结果来进行说明。首先对于红灯时长，每波车流的时长分布直方图如下图 8 所示。对于红灯时长的确认，应当使其尽可能大；但同时也应与其他较短的时长保持连续。对于下图中的最大值，与通过其他车流得到红灯时长有明显差异，且 75 至 100 之间有明显间断，这在实际中是不合理的。因此我们可以认为 A1 路口一个信号周期内红灯时长为 75 秒。

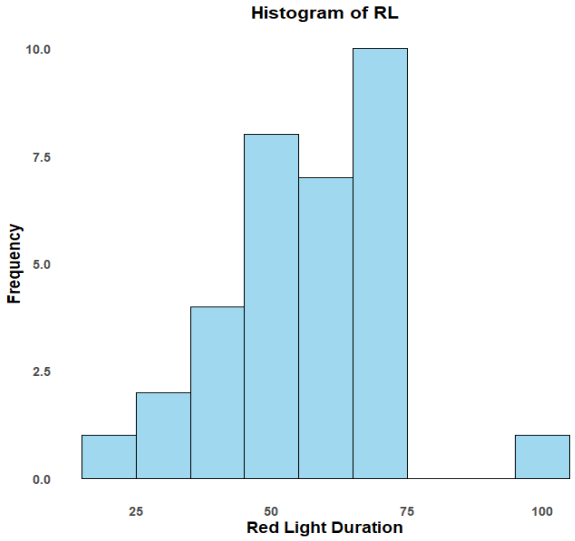

图 9　A1 红灯时长分布

通过观察相邻两波车流间的差值$Difference(D)$，我们发现$Difference(D)$的值为一个红绿灯的总时长或其倍数，所以我们选取每个路口所有波次的$Difference(D)$中的最小值，可以认为其就是一次红灯与绿灯的时长总和。之后，根据$D - LR$可以得到不同波次车流的绿灯时长。

使用同样的判定方法，我们可以分别根据附件中 A1-A5 五个路口的数据分别求出各自相应方向的信号灯周期，具体结果见下表 1：

表 2　路口 A1-A5 各自一个方向信号灯周期识别结果

路口	A1	A2	A3	A4	A5
红灯时长（秒）	75	64	82	72	64
绿灯时长（秒）	30	24	23	16	24

5. 问题二模型建立与求解

5.1 问题分析

对于问题二，题目指出实际中只能获取部分样本车辆的行车轨迹，同时会受各种因素的影响，需讨论定位误差、样本车辆比例、车流量等因素对模型估计精度的影响，并求出附件 2 中各路口的信号灯周期。本文首先将定位误差、样本车辆比例、车流量因素分别加入问题一模型中，定位误差以 x、y 坐标添加随机误差形式考虑，样本车辆比例以随机抽取车辆的形式考虑，车流量以速度、加速度、路口区域的阈值变化形式考虑，其次以平均误差为指标分析各因素对模型估计精度的影响，最后将三因素影响同时考虑，计算附件 2 中各路口的信号灯周期。

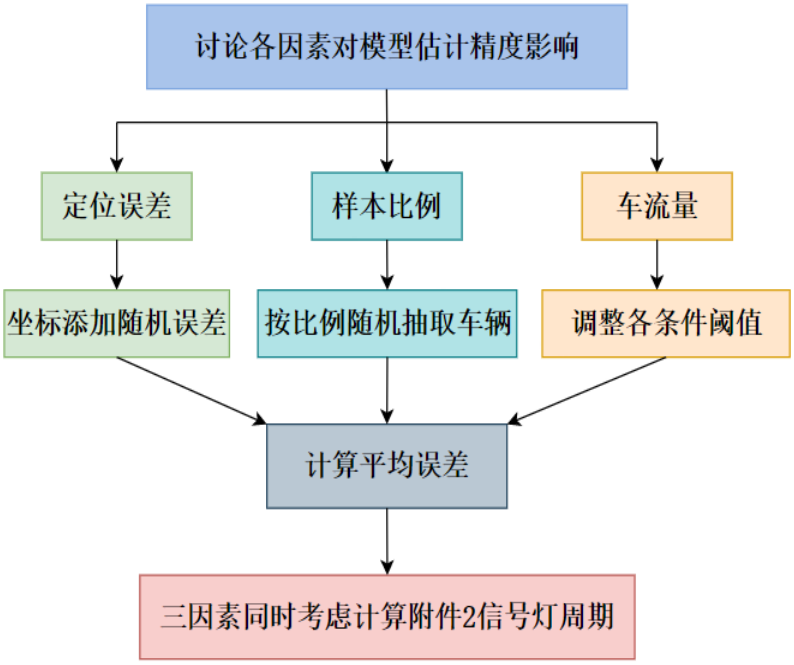

图 10　问题二思路图

5.2 讨论影响因素对模型估计精度的影响

5.2.1 考虑定位误差对模型精度的影响与修正模型

（1）考虑定位误差对模型精度的影响

考虑到定位误差会影响到速度、加速度的确定，进而影响对于每辆车所经历的红灯时常的识别，从而使估计的信号灯周期的精度降低。

（2）修正模型

为方便计算，我们提出以下假设：

假设一：假设该产品每次定位产生的误差大小是随机的，我们用相对误差ε_r这一指标来衡量该误差的大小，

$$\varepsilon_x = \frac{x^* - x}{x} \quad (6)$$

其中，x为定位位置，x^*为精确位置，此处以 x 方向上的位置点为例，y 方向的上位置点处理同理。

假设二：假设电子地图商的产品在投入市场前经过大量的检验测试，每次定位位置与精确位置的相对误差不超过 3%，即可认为$\varepsilon_x \sim N(0,0.03)$(y 方向同理)。

假设三：假设由于车辆停止时间较长，故认为车辆停止期间不产生定位误差。因此速度车辆为零的位置点处不需要进行定位误差的修正。

为具体确定坐标误差的情况，我们通过附件 1 中车辆轨迹判断出路口形状并不相同，五个路口的大致形状如下图 6。对于不同路口，我们依据其线路类型的特点采取不同的处理方式来修正其定位误差。

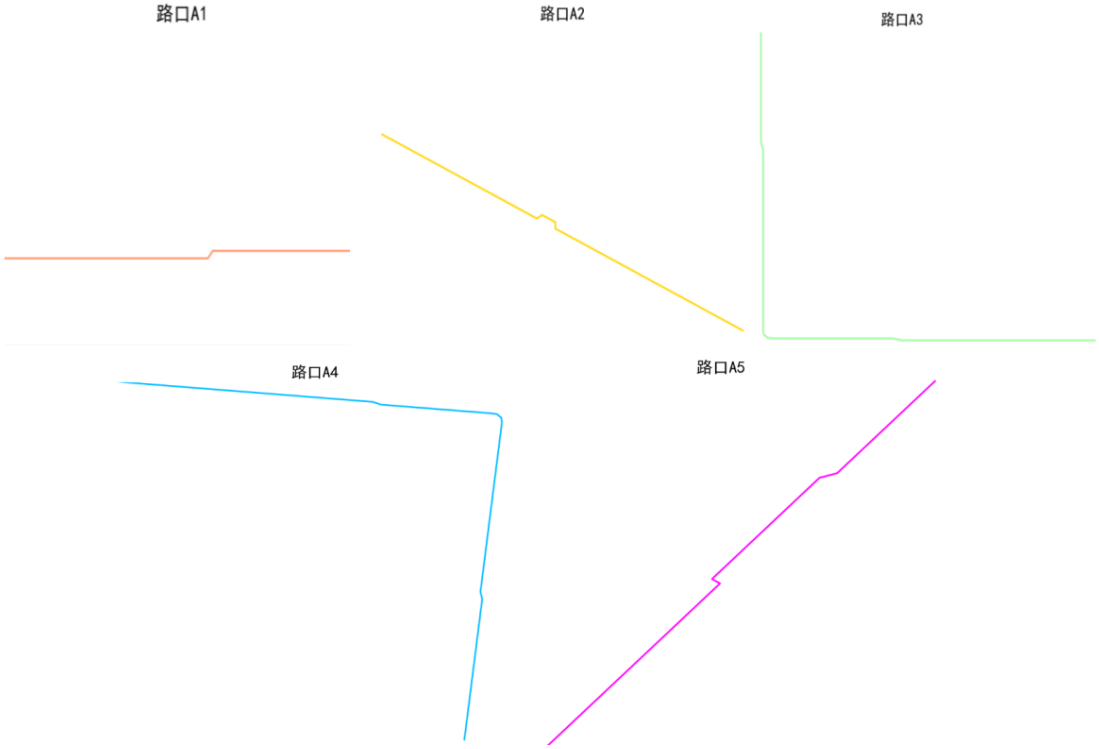

图 11　五个路口大致形状

对于 A1 路口，观察到其为 x 方向上的平行线，故只需对 x 轴上的位置点进行定位误差的修正；对于 A2、A4、A5 既不平行于 x 轴也不平行于 y 轴的路口，需要对每一位置点的 x 方向和 y 方向上进行修正；对于 A3 平行于坐标轴但需要拐弯的路口，拐弯前应对 y 方向上的位置进行修正，拐弯后应对 x 方向上的位置进行修正。

由此得到每个需要修正的位置点的相对误差，进而对位置进行修正：

$$x^* = x \times (1 + \varepsilon_x), y^* = y \times (1 + \varepsilon_y) \quad (7)$$

最终得到修正后的位置(x^*, y^*)。

（3）模型修正后的结果与检验

考虑定位误差后修正模型估计周期与原先的估计周期比较如下：

表 3　未考虑各因素前估计的各路口的信号灯周期

路口	A1	A2	A3	A4	A5
红灯时长（秒）	75	64	82	72	64
绿灯时长（秒）	30	24	23	16	24

表 4　修正定位误差后估计的各路口的信号灯周期

路口	A1	A2	A3	A4	A5
红灯时长（秒）	78	65	82	73	63
绿灯时长（秒）	27	23	23	11	25

下图 7 为考虑定位误差后修正模型估计周期与原先的估计周期的比较图。

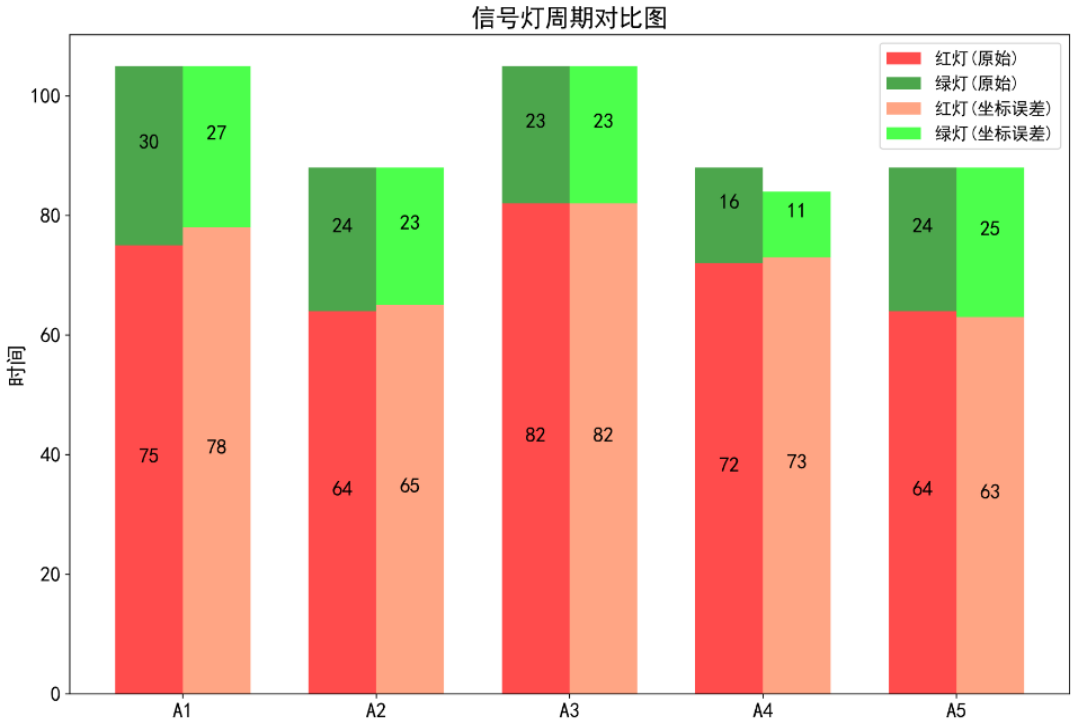

图 12　考虑坐标误差后的信号灯周期对比图

由此求出各路口由定位误差造成的红灯周期估计的相对误差：

$$\varepsilon_{ri} = \frac{T_i^* - T_i}{T_i} \tag{8}$$

其中，T_i^*为路口A_i的新的红灯周期，T_i为路口A_i的原先的红灯周期。

再将各路口的相对误差求平均，得到由定位误差造成的平均相对误差为 0.08514，说明坐标误差对模型估计精度的影响不大。

5.2.2 考虑样本比例对模型估计精度的影响与模型修正

（1）样本比例假设

假设一：假设该电子地图产品至少有一定的占有率，若占有率过低，即样本比例过低，则会导致模型估计信号灯周期准确度较低。因此认为样本比例不低于

30% 。

（2）样本比例考虑

设样本比例为p，n_i为A_i路口被统计的车辆数。对附件 1 中的各路口，通过简单随机抽样的方法抽取$p*n_i$辆车作为样本 ，若$p*n_i$不是整数，则将其先进行向下取整。

对各路口，仅依据抽取出的车辆的轨迹数据，采用问题一中的模型，估计出各路口新的红灯周期T_i^*。仍用公式 8 求出各路口在抽样比例p下造成的相对误差，再将各路口的相对误差求平均，得到样本比例为p时造成的平均相对误差：

$$\overline{\varepsilon_{rp}} = \sum_{i=1}^{5} \varepsilon_{rip} /5, \qquad (9)$$

将p依次取遍 30%至 90%，每次增加 1%。由此得到样本比例大小与红灯周期的平均相对误差绝对值的关系如下图 9 所示。

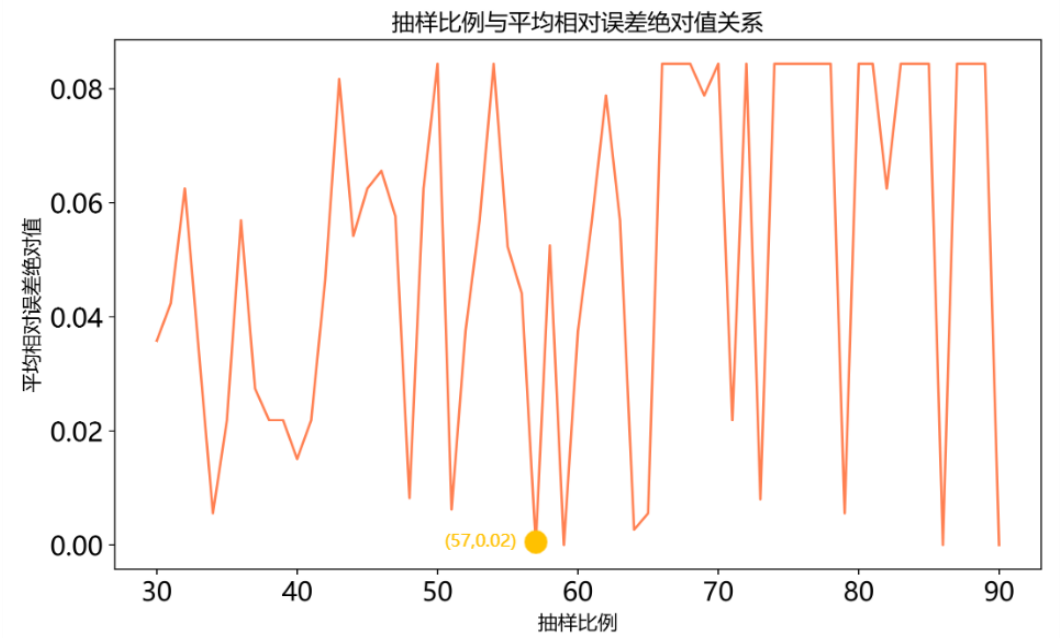

图 13　抽样比例与平均相对误差的绝对值的关系

由上图结果，当抽样比例为 57%时，红灯周期估计的平均相对误差的绝对值最小，约为 0.02，最高没有超过 0.08，说明样本车辆比例对模型估计精度影响相对较小。

5.2.3 考虑车流量对模型精度的影响与模型修正

车流量主要影响的是模型中关于阈值的确定。若车流量较小，则汽车在道路上整体行驶速度较快，因此识别其遇到红灯开始减速的阈值应设置为较大的数值；同理，若车流量较大，则汽车在道路上整体行驶速度较慢，因此识别其遇到红灯开始减速的阈值应设置为较小的数值。

（1）车流辆大小的界定与分类

通过使用问题一中的模型，结合统计分析，我们确认了每一次完整红绿灯周

期下每波车流的车辆数均在 0-6 辆之间，由于数量太小，我们在尝试多种聚类方法后发现效果并不好，所以最终决定将车流量进行排序后，按照每波车辆数小于 3 作为车流量低组，每波车辆数为 3 或 4 记为车流量中组，每波车辆数为 5 或 6 记为车流量高组。以路口 B1 为例，我们将所有车辆分为三组如下表 5。

表 5　路口 B1 的车流量分组

车流量组	车辆 ID
低车流量组	168、194、259、268、344、380、666、668、823、913、938、1094、1102、1178、1199、1340、1425、1434
中车流量组	30、44、56、106、107、142、314、325、329、480、485、496、521、545、561、581、583、586、594、620、625、628、644、708、715、743、745、987、985、1002、1114、1117、1123、1457、1459、1465
高车流量组	402、405、414、420、424、425、436、440、441、456、748、755、757、765、766

（2）阈值的调整

针对不同的车流量组，我们认为不同车流量时车辆的行车速度与遇到红灯时减速的加速度不同，将各车流量组在第一问中的状态划分与节点识别时的加速度、速度、路口区域的阈值进行调整，以下为各组调整结果。

表 6　阈值调整表

车流量组	速度阈值$v_0(m/s)$	减速时加速度阈值$a_0(m/s^2)$	路口区域阈值$L(m)$
问题一模型初始设置	10	-2	100
低车流量组	15	-3	50
中车流量组	10	-3.5	100
高车流量组	5	-4	150

（3）模型修正后的结果与检验

采用问题一中模型计算得到考虑车流量后各个路口的信号灯周期如下表 7，

表 7　考虑车流量因素后估计的各路口的信号灯周期

路口	A1	A2	A3	A4	A5
红灯时长（秒）	74	57	82	73	64
绿灯时长（秒）	31	31	23	11	24

绘制和原始数据的对比柱形图如下图 14。

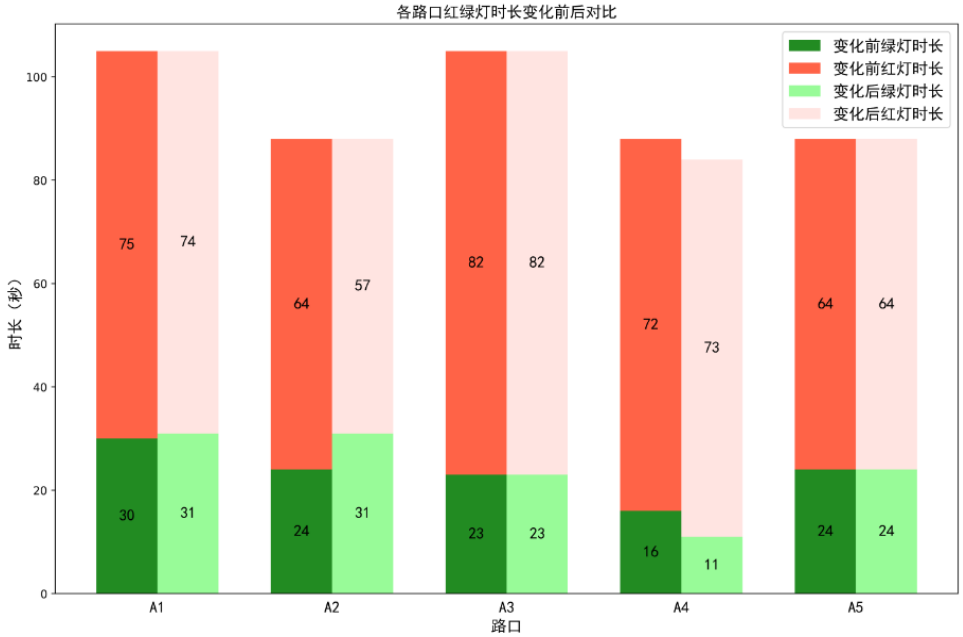

图 14　考虑车流量后的信号灯周期对比图

仍用前文使用的相对误差计算公式 8，得到各个路口的相对误差，最后计算平均误差为 0.135697，说明考虑车流量时对模型估计精度影响相对较大。

综上所述，定位误差与样本车辆比例对于问题一中模型估计精度的影响相对较小，车流量对于模型估计精度的影响较大。下面将同时考虑三个因素，求出附件 2 中各路口的信号灯周期。

5.3 估计信号灯周期

对附件 2 中的数据，首先绘制出各路口的线路特点，参照上文得到修正定位误差的类似做法后，对每一位置点的坐标进行修正。

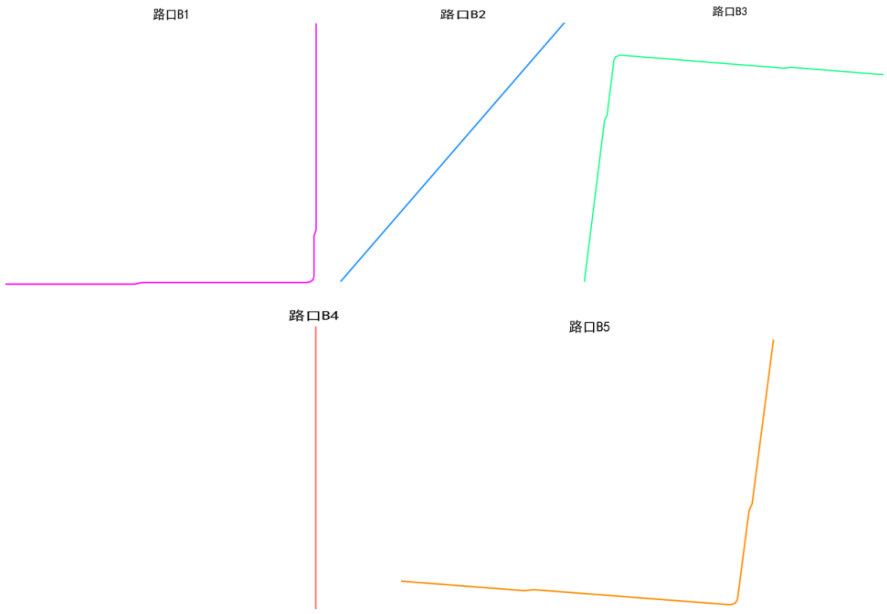

图 15　附件 2 各路口形状

接着由修正好的坐标位置，计算每辆车每时间点上的速度，然后用考虑了定位误差和车流量因素的修正后的模型，估计出附件二中每个路口的红灯周期T_i。

我们由附件 1 中的数据得到了样本比例为p时造成的平均相对误差$\overline{\varepsilon_{rp}}$。因此，若附件2的样本比例为$p$，对估计出的附件2中每个路口的红灯周期$T_i$进行修正，得到最终的红灯周期$T_i^*$：

$$T_i^* = T_i \times (1 + \overline{\varepsilon_{rp}}) \quad (10)$$

假设附件 2 样本占总体的比重，恰好等于平均相对误差绝对值最小时的样本比例，即使用考虑样本比例时求得的最小平均相对误差 0.02 代入由此估计出附件 2 中各路口的红灯周期。

表 7　估计出的附件 2 中各路口的信号灯周期

路口	B1	B2	B3	B4	B5
红灯时长（秒）	75	64	73	80	97
绿灯时长（秒）	30	24	15	25	19

6. 问题三模型建立与求解

6.1 问题分析

问题三要求我们考虑如果信号灯周期有可能发生变化，能否尽快检测出这种变化，以及求出变化后的新周期。本文首先用 DBSCAN 算法的对同一信号灯周期通过路口的车流聚类，用问题二中考虑影响因素后的模型，得到每一周期信号灯总时长并按发生的时间先后进行排序。其次考虑到相邻周期内可能无车通过，对过长的信号灯时长进行拆解，得到新的排序结果。最后通过局部加权多项式回归对排序结果进行处理，找出每个路口中信号灯变化的拐点，以拐点将总时间分段后找到每段的最大值作为该段红灯时长，并判断信号灯时长变化的可能原因。

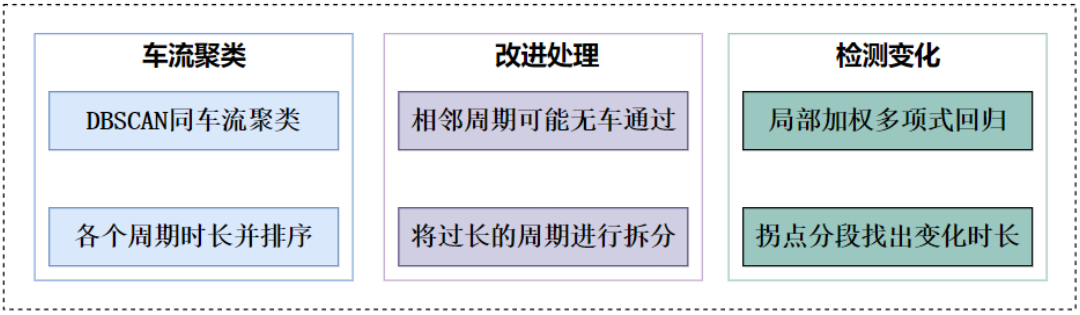

图 16　问题三思路图

6.2 模型建立与求解

（1）模型建立

我们在问题二中，考虑了样本车辆比例、车流量、定位误差等因素，得到了新的估计信号灯周期模型。我们按照问题一中的处理方式，对附件 3 中各路口，先用 DBSCAN 算法的对同一信号灯周期通过路口的车流聚类进行聚类，然后用改进的新模型，得到每一周期内红灯和绿灯总时长的估计并按发生的时间先后进行排序，画出时间序列的散点图后将各点连线，得到如下图所示结果：（下面以路口 C1 为例，给出具体的处理方法，其他路口的处理方法同理。）

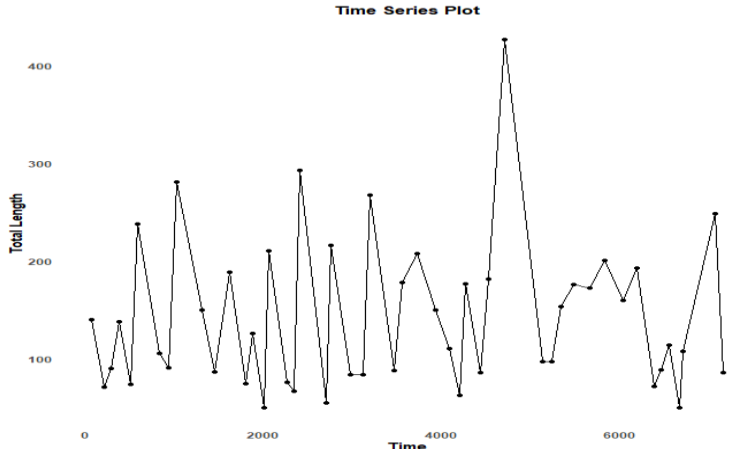

图 17　C1 路口处的交通信号灯每周期的红绿灯总时长估计

观察到不同周期内，红灯和绿灯总时长的估计相差较大，且数值上呈现较明显的 2 倍或 4 倍的倍数关系。我们认为，由于车流量较稀疏，连续相邻的几个信号灯周期内均无车辆通过路口，从而导致估计出的某一周期内的红灯和绿灯总时长较长。因此图像上折线的密度，能够反应对应车流量的大小。

对于上图 11 所示路口，其估计出的信号灯周期的时长数值应在 100 左右浮动，对此我们做出相应的改进处理：若估计出的信号灯周期总时长的数值超过 200 但未超过 400，则将相应的数值除以 2；若估计出的信号灯周期总时长的数值超过 400，则将相应的数值除以 4。调整后得到下图结果：

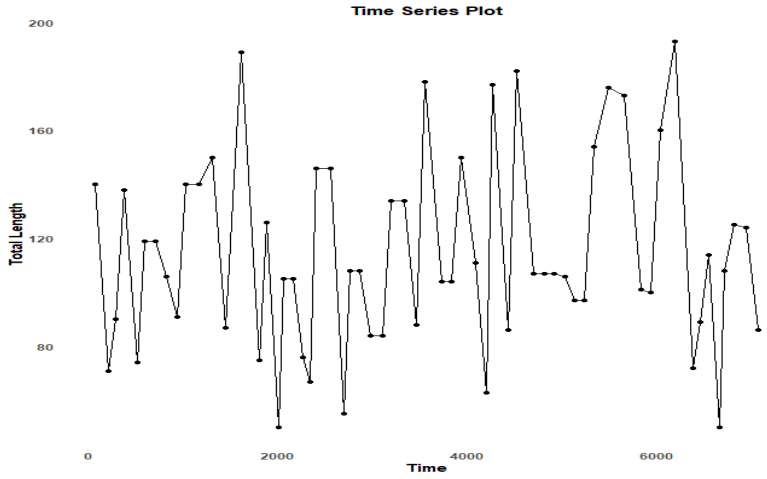

图 18　调整后的 C1 路口处交通信号灯每周期的红绿灯总时长估计

由上图结果，不能直接观测直接得到信号灯周期的变化，故对上图进行平滑化处理的尝试。经过大量尝试，最终找到局部加权多项式回归平滑法的平滑效果最好。

局部加权多项式回归平滑法的核心思想是在每个数据点附近拟合一个局部多项式模型，然后使用这些局部模型的加权平均来获得平滑曲线。在每个局部区域，距离某个数据点越近的数据点将获得更大的权重，而距离较远的数据点将获得较小的权重，其具体算法流程如下：

1）使用权值函数$W(x)$ ：

$$W(x)=\begin{cases}1, & |x|<1\\ 0, & |x|\geq 1\end{cases}$$

2）将 x 轴方向上的数据映射到区间[− 1 , 1]上对应的坐标；

3）带入函数$W(x)$ ，计算出每个点对应的权重 w_i ；

4）使用加权回归得出模型：$\hat{Y}=X(X^TwX)^{-1}X^TwY$ ；

局部加权多项式回归平滑法的的最终效果如下：

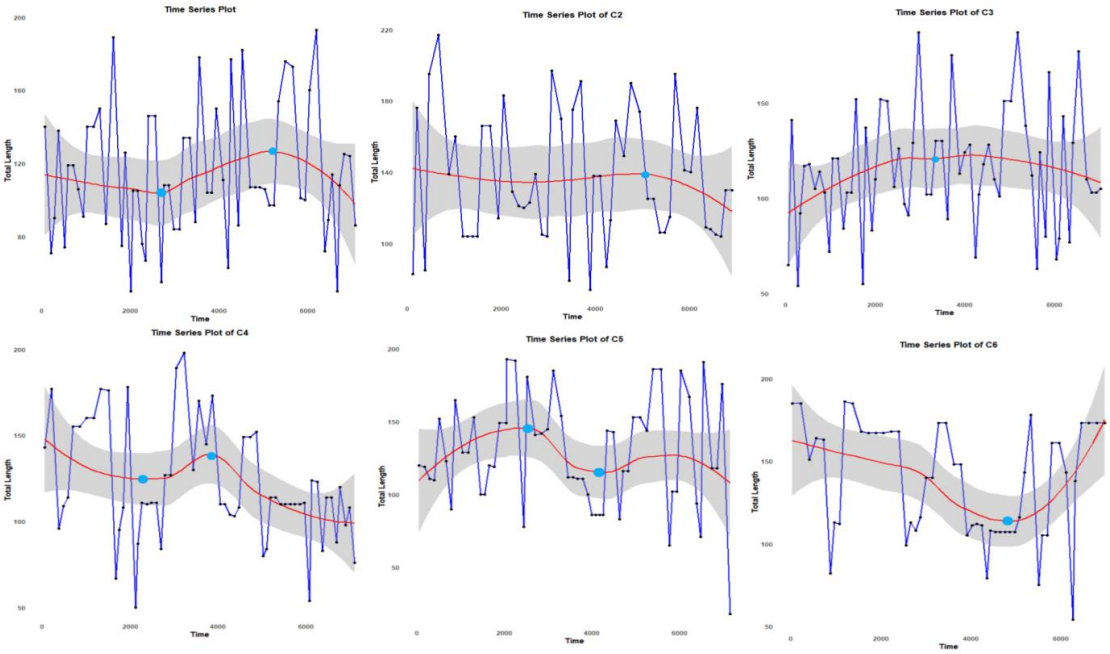

图 19　平滑后的效果图

（2）结果分析

从上图 13 中，注意到 C1、C4、C5 平滑后的曲线上，有两个拐点，每个拐点前后的每周期红绿灯总时长变化趋势不同，且两个拐点分成的三部分点的密度(也即车流量不同)，因此认为每个拐点对应一个交通信号灯周期改变的时刻。

对于 C2、C3、C6，注意到平滑后的曲线上，有一个较明显的拐点，拐点前后估计出的红绿灯总时长的趋势发生了较明显的变化，且拐点对应时刻之后的车流量相较于该时刻之前的车流量较为不同。因此认为有一个时刻交通信号灯的周期发生了改变。

拐点将统计的总时间划分为了不同的时间段，我们认为每个时间段内的交通信号灯是一致的、保持不变的。按照问题一中的处理方法得到每个时间段信号灯周期的具体红绿参数。

表 9　估计的附件 3 中各路口的交通信号灯的周期

路口	C1	C2	C3	C4	C5	C6
周期 1 红灯时长（秒）	57	66	80	69	46	72
周期 1 绿灯时长（秒）	31	22	25	36	42	33
周期切换时刻（第_秒）	2757	5116	4024	3915	2438	4136
周期 2 红灯时长（秒）	53	64	73	78	59	78
周期 2 绿灯时长（秒）	35	24	32	27	29	27
周期切换时刻（第_秒）	5134			3871	4184	
周期 3 红灯时长（秒）	48			64	50	
周期 3 绿灯时长（秒）	40			41	38	

7. 问题四模型建立与求解

7.1 问题分析

问题四要求我们通过某路口连续 2 小时内所有方向样本车辆的轨迹数据，识别出该路口信号灯的周期。首先，要求出该路口的所有车辆方向，经过初步绘图发现该路口为一个十字路口。接着尝试以所有车辆的 x、y 坐标为特征使用 DBSCAN 聚类，由于数据量过大与参数设置等问题，发现效果并不好，所以我们采用“起始点+起终点斜率”的判断方式进行统计分类，最终发现一共有 12 种行驶方向，我们将这 12 种行驶方向演变成为类似问题三的 12 种路口采用前三问的模型进行求解。

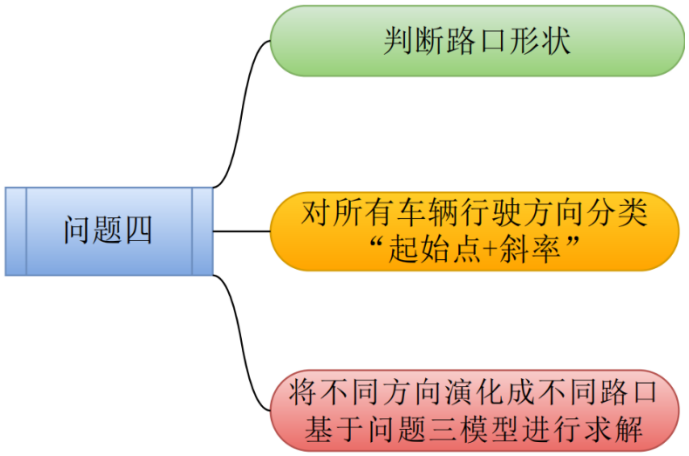

图 20　问题四流程图

7.2 车辆轨迹分类

首先，题目中需要我们找出所有可能的行车方向，我们对所有车辆的行车轨迹进行绘制，发现路口为十字路口，以每个方向三种可能通行方向(直行、左转、右转)考虑，最多会有 12 种通行可能。

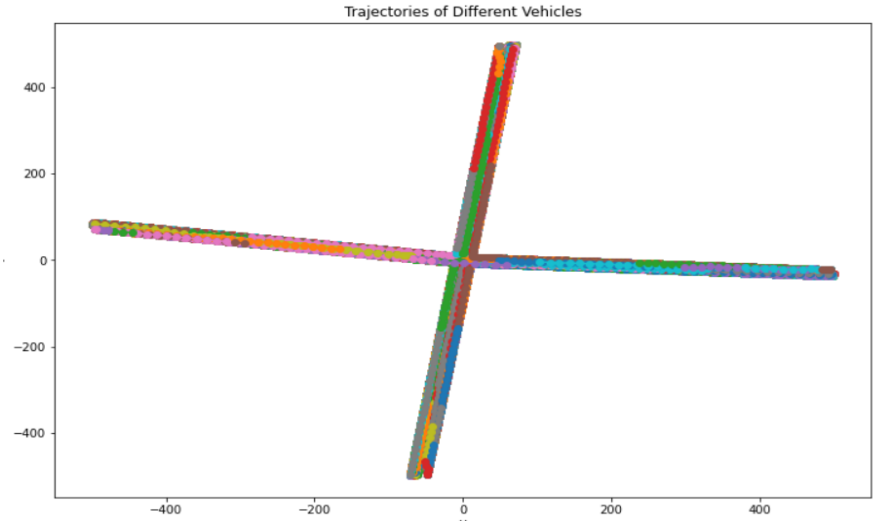

图 21　所有车辆轨迹图

但由于未知具体类数，无法采用传统的聚类方法，所以我们仍然尝试使用 DBSCAN 算法，以车辆的 x、y 坐标为特征进行聚类，但由于参数设置与数据量过大等问题，DBSCAN 算法并未成功实现聚类。接着我们考虑采用“起始点+起终点斜率”的方式进行分类，我们将上图四个方向分别简记为 A、B、C、D，由于从 A 到 C 与从 C 到 A、从 B 到 D、从 D 到 B 的斜率相同(其他同理)，所以需要加入起始点坐标进行区分，由此得到十二种分类方式如下表 9。

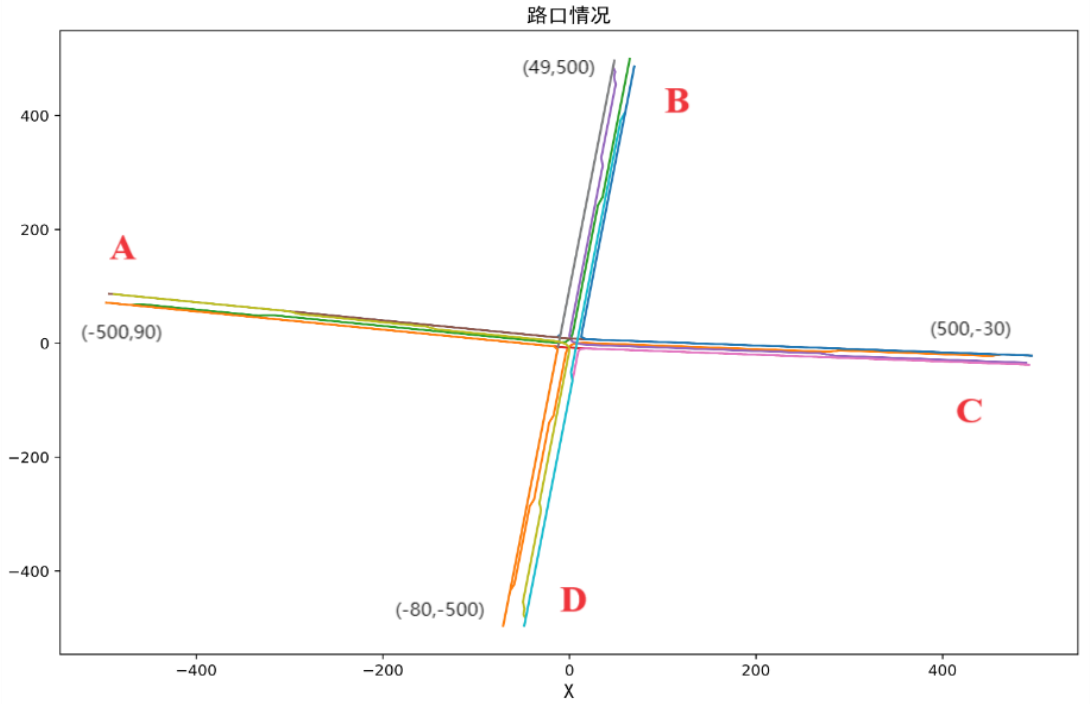

图 22　路口方向图

表 10　分类与条件表

类别	条件
第一类（从 A 到 C 直行）	起点坐标 x<-490 且-0.3<斜率<0
第二类（从 A 到 B 左转）	起点坐标 x<-490 且 0.7<斜率<1
第三类（从 A 到 D 右转）	起点坐标 x<-490 且斜率<-1
第四类（从 B 到 D 直行）	起点坐标 48<x<100 且斜率>1
第五类（从 B 到 C 左转）	起点坐标 48<x<100 且斜率<-1
第六类（从 B 到 A 右转）	起点坐标 48<x<100 且 0.7<斜率<1
第七类（从 C 到 A 直行）	起点坐标 x>490 且-0.3<斜率<0
第八类（从 C 到 D 左转）	起点坐标 x>490 且 0.7<斜率<1
第九类（从 C 到 B 右转）	起点坐标 x>490 且斜率<-1
第十类（从 D 到 B 直行）	起点坐标-100<x<-40 且斜率>1
第十一类（从 D 到 A 左转）	起点坐标-100<x<-40 且斜率<-1
第十二类（从 D 到 C 右转）	起点坐标-100<x<-40 且 0.7<斜率<1

将共计 3298 辆车按照条件分到不同类别中，下图为部分初始分类图，其中某车属于某类记为 1，不属于记为 0，经检验每类中均有车辆且每辆车只被分到了一类中。

车辆ID	第一类:A到C	第二类:A到B	第三类:A到D	第四类:B到D	第五类:B到C	第六类:B到A	第七类:C到A	第八类:C到D	第九类:C到B	第十类:D到B	第十一类:D到A	第十二类:D到C
vehicle_0	0	0	0	0	0	1	0	0	0	0	0	0
vehicle_2	0	0	0	0	0	0	0	1	0	0	0	0
vehicle_4	0	1	0	0	0	0	0	0	0	0	0	0
vehicle_6	0	0	0	0	0	1	0	0	0	0	0	0
vehicle_8	1	0	0	0	0	0	0	0	0	0	0	0
vehicle_9	1	0	0	0	0	0	0	0	0	0	0	0
vehicle_13	0	0	0	0	1	0	0	0	0	0	0	0
vehicle_16	0	1	0	0	0	0	0	0	0	0	0	0
vehicle_17	1	0	0	0	0	0	0	0	0	0	0	0
vehicle_18	0	0	0	0	0	0	1	0	0	0	0	0
vehicle_19	0	1	0	0	0	0	0	0	0	0	0	0
vehicle_20	0	0	0	0	1	0	0	0	0	0	0	0
vehicle_21	0	0	0	0	0	1	0	0	0	0	0	0
vehicle_22	0	0	0	0	0	0	0	0	0	0	0	1
vehicle_25	0	0	0	0	0	0	1	0	0	0	0	0
vehicle_27	0	0	0	1	0	0	0	0	0	0	0	0
vehicle_29	0	0	0	1	0	0	0	0	0	0	0	0
vehicle_32	0	0	0	0	1	0	0	0	0	0	0	0
vehicle_33	0	0	0	0	0	0	0	1	0	0	0	0
vehicle_34	0	0	0	0	0	0	0	0	0	0	1	0
vehicle_35	0	1	0	0	0	0	0	0	0	0	0	0
vehicle_38	0	0	0	0	0	0	0	0	0	1	0	0
vehicle_41	0	0	0	0	0	0	0	0	1	0	0	0
vehicle_42	0	0	0	1	0	0	0	0	0	0	0	0
vehicle_44	0	1	0	0	0	0	0	0	0	0	0	0
vehicle_51	0	0	0	0	0	0	0	1	0	0	0	0

图 23　部分车辆初始分类图

之后，我们将车辆所属类别具体展现出来，得到最终分类结果如下图 x，数字即为车辆 ID，完整分类结果见支撑材料。

第一类A到C	第二类A到B	第三类A到D	第四类B到D	第五类B到C	第六类B到A	第七类C到A	第八类C到D	第九类C到B	第十类D到B	第十一类D到A	第十二类D到C
8	4	116	27	13	0	18	2	41	38	34	22
9	16	121	29	20	6	25	33	90	93	52	73
17	19	140	42	32	21	65	51	91	101	79	117
55	35	143	54	63	53	66	62	99	115	85	124
77	44	184	102	82	56	88	69	142	125	157	129
95	59	279	119	84	72	134	71	169	132	198	139
111	98	281	133	96	75	155	151	173	147	207	145
120	137	290	138	153	87	159	218	188	149	211	156
126	152	301	165	158	162	197	236	190	176	238	191
141	167	352	166	189	163	201	371	192	241	262	247
194	170	385	202	215	174	221	373	232	275	271	256
196	203	399	212	229	187	230	386	273	283	312	284
208	205	402	267	233	209	250	391	335	294	317	296
231	242	406	319	282	210	266	423	350	308	324	299
239	243	413	323	310	216	274	433	358	329	334	307
254	251	436	341	336	220	325	473	404	348	342	318
291	252	484	351	366	261	332	480	443	374	380	320
328	253	534	357	392	263	345	499	452	377	412	344
353	287	536	389	397	288	383	516	457	407	419	354
365	311	541	411	401	292	463	521	458	420	441	387
378	321	559	424	426	313	511	531	474	427	450	418
394	434	575	428	449	315	535	545	488	467	454	444
403	469	587	439	453	326	547	567	501	472	455	456
442	496	597	451	471	327	555	574	529	485	487	461
445	518	638	460	493	370	561	589	537	500	506	483
459	556	642	509	504	414	569	600	553	512	543	528

图 24　部分车辆最终分类结果

接着，我们按照问题三的思想，将这十二种行车方式演化为十二个不同的路口按照前文模型进行求解，探索其周期及变化情况，为方便说明结果，我们将这十二类分别简记为 D1-D12，得到最终结果如下表。

表 11　问题四结果表

方向	D1	D2	D3	D4	D5	D6
周期一红灯时长	104	114	97	92	119	89
周期一绿灯时长	38	28	45	50	23	53
周期切换时刻	3008	2487	3262	2340	3908	2685
周期二红灯时长	96	110	88	97	114	96
周期二绿灯时长	46	32	54	45	28	46
周期切换时刻		4269		4319		4168

周期三红灯时长		108		91		90
周期三绿灯时长		34		51		52
方向	D7	D8	D9	D10	D11	D12
周期一红灯时长	103	115	99	93	113	92
周期一绿灯时长	39	27	43	49	29	50
周期切换时刻	2670	4997	4181	3791	2775	1827
周期二红灯时长	100	110	92	101	119	85
周期二绿灯时长	42	32	50	41	23	57
周期切换时刻	5172			5522		3795
周期三红灯时长	98			95		92
周期三绿灯时长	44			47		50

8. 模型的评价、改进与推广

8.1 模型的优点与创新

对于问题一，我们创新性地使用了 DBSCAN 算法，对属于同一红绿灯周期通过路口的车流进行聚类。相比于 K-Means 等传统聚类算法，DBSCAN 不需要提前指定簇的数量，并且能够有效处理噪声点，从而得到最优的分类方案 。

对于问题二，我们充分考虑了样本车辆比例、车流量、定位误差等因素对问题一中模型估计精度的影响，并对问题一中的模型进行了修正。在讨论定位误差对模型估计精度的影响时，我们考虑到了误差产生的随机性，从而对各位置点的坐标进行了更加合理的误差修正。

对于问题三，我们经过大量尝试，最终找到平滑效果最好的方法——局部加权多项式回归平滑法。我们借助平滑后的曲线，对于交通信号灯周期的变化能够有更直观、更准确的判断。

对于问题四，我们采用“起始点+斜率”的分类方法，成功精确地判断出每一辆车所属地行驶路线类别，分类结果比聚类算法的结果更为精确，实施更方便。

8.2 模型的缺点

对于问题一，尽管我们有结合物理常识、实际生活经验，并且有参考大量文献，来对速度阈值进行设定的，但是由于缺乏更多关于路况的具体信息，因此我们不能保证阈值设置的完全合理性。此外，DBSCAN 算法本身具有一定的缺陷，如果参数设置不当，可能导致聚类结果不佳。

对于问题二，我们不能保证修正定位误差后的位置与精确位置完全重合，即使是修正后的位置依然与真实的精确位置存在误差。此外，我们对于阈值的调整存在一定的主观性，我们不能保证阈值的调整能够充分反应道路的实际情况。

对于问题三，尽管我们求解平滑曲线前的所有过程，有充分的、符合基本数学逻辑的机理分析做支撑，而我们仅依据平滑曲线变化趋势的图像，来直观地判

断各路口交通信号灯的时间周期是否发生变化，以及确定变化的时刻点。这种做法在逻辑的完整性与严谨性上有所欠缺。

对于问题四，“起始点+斜率”这一分类方法在检测点固定时使用效果较好，但当换至其他路口时，需要重新计算分类条件，斜率与初始点坐标判断都将改变，在可推广性上有所欠缺。

8.3 模型的改进与推广

我们的模型能够在有限数据信息的情况下，对交通信号灯周期及其红绿灯时长，做出较未准确的估计，并且能够识别出交通信号灯周期的变化。我们的模型不仅适用于单行线道路，还能推广到路况复杂的交叉路口。由此，我们的模型能够获取到，部分未接入网络的信号灯的详细数据，从而提高了获取信号灯周期数据的便利性。我们可以通过改进 DBSCAN 算法得到更好的聚类效果；或者尝试实地调研具体路况，依据调研得到的真实数据，改进设定与调整阈值的方法。模型由此能够更准确地估计结果，从而能更好地应用于各种现实状况。

9. 参考文献

[1]任柏寒,丁雪梅.基于流量大数据的交叉口信号控制时段划分方法研究[J].海峡科技与产业,2021,34(08):51-53.

[2]李恒,张氢,秦仙蓉,等.基于短时傅里叶变换和卷积神经网络的轴承故障诊断方法[J].振动与冲击,2018,37(19):124-131.DOI:10.13465/j.cnki.jvs.2018.19.020.

[3]雷代文.智能交通信号灯配时策略研究[D].重庆大学,2018.

[4]闫瑞雪.信号灯最佳周期的确定方法及其仿真研究[D].东北林业大学,2010.

[5]唐铁桥,黄海军,徐刚,等.考虑信号灯影响的交通流模型与数值模拟[J].物理学报,2008,(01):56-60.

[6]赵凤展,杨仁刚.基于短时傅里叶变换的电压暂降扰动检测[J].中国电机工程学报,2007,(10):28-34+109.

[7]陈传明,张铃,赵姝,等.智能交通信号灯配时及优化设计[J].微机发展,2005,(03):4-6+75.

[8]程军圣.基于 Hilbert-Huang 变换的旋转机械故障诊断方法研究[D].湖南大学,2005.

[9]荣秋生,颜君彪,郭国强.基于 DBSCAN 聚类算法的研究与实现[J].计算机应用,2004,(04):45-46+61.

[10]周水庚,周傲英,曹晶.基于数据分区的 DBSCAN 算法[J].计算机研究与发展,2000,(10):1153-1159.

10. 附件

10.1 支撑材料

以下是本文模型搭建涉及的全部代码及数据支撑材料列表：

表 12　支撑材料列表

文件名	作用描述
数据预处理代表--A3、ID23.xlsx	路口 A3、ID23 车辆数据预处理结果
vehicle_Ai.xlsx(i=1,…,5)	问题一中第 i 个路口各车辆的速度与加速度
Aitime.xlsx(i=1,…,5)	问题一中第 i 个路口各车辆减速、开始加速的时间节点
Ai 车流量分类.xlsx（i=1,…,5）	问题二中讨论路口 A1-A5 的车流量影响
vehicle_Ai_11.xlsx（i=1,…,5）	问题二中 A1-A5 考虑坐标变换后第 i 个路口的各车辆速度与加速度
time_points_1.xlsx(i=1,…,5）	问题二中 A1-A5 考虑坐标误差后第 i 个路口各车辆减速、开始加速的时间节点
Bi 车流量分类.xlsx(i=1,…,5)	问题二中附件 2 B1-B5 车流量的分类结果
vehicle_Bi_updated.xlsx(i=1,…,5)	问题二中 B1-B5 考虑坐标误差后的新坐标
vehicle_Bi_11.xlsx(i=1,…,5)	问题二中 B1-B5 考虑坐标误差后各车辆速度与加速度
Bitime.xlsx(i=1,…,5)	问题二中 B1-B5 考虑坐标误差后第 i 个路口各车辆减速、开始加速的时间节点
样本比例估计.xlsx	问题二中考虑样本比例（从 30 到 90）各路口红灯结果
Ci.xlsx(i=1,…,6)	问题三中 C1-C6 考虑坐标误差后的新坐标
vehicle_Ci.xlsx(i=1,…,6)	问题三中 C1-C6 考虑坐标误差后各车辆速度与加速度
time_pointsCi.xlsx(i=1,…,6)	问题三中 C1-C6 考虑坐标误差后第 i 个路口各车辆减速、开始加速的时间节点
red_light_timesCi.xlsx(i=1,…,6)	问题三中 C1-C6 考虑坐标误差后第 i 个路口各车辆遇到的红灯时间
问题四各车辆的起终点与斜率.xlsx	问题四分类时用到的起终点坐标与斜率
第四问分类结果.xlsx	问题四所有车辆按照行驶方向的分类结果
vehicle_Di.xlsx(i=1,…,12)	问题四十二类车辆的速度与加速度
D1time.xlsx(i=1,…,12)	问题四十二类车辆减速、开始加速的时间节点
red_light_timesDi(i=1,..,12)	问题四十二类车辆各车遇到的红灯时间
第一问.py	第一问求解全部相关代码
##问题二 第一问 考虑坐标误差后.py	第二问全部相关代码
#问题三 求红灯时间.py	第三问全部相关代码
#第四问.py	第四问全部相关代码
DBSCAN.R	DBSCAN 聚类相关代码

10.2 相关代码

C: > Users > 86178 > Desktop > 第一问.py > ...

```python
import pandas as pd
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/A5.xlsx')
data = data.sort_values(by=['vehicle_id', 'time'])
# 定义计算欧式距离的函数
def compute_distance(df):
    df['distance'] = np.sqrt(df['x'].diff()**2 + df['y'].diff()**2)
    df['distance'].fillna(0, inplace=True)  # 填充第一个NaN值
    return df
# 定义计算速度的函数
def compute_speed(df):
    df['speed'] = df['distance'] / df['time_diff']
    return df
# 定义计算加速度的函数
def compute_acceleration(df):
    df['acceleration'] = df['speed'].diff()
    df['acceleration'].fillna(0, inplace=True)  # 填充第一个NaN值
    return df
# 分组计算欧式距离
data = data.groupby('vehicle_id').apply(compute_distance)
# 计算时间差
data['time_diff'] = data.groupby('vehicle_id')['time'].diff()
# 分组计算速度
data = data.groupby('vehicle_id').apply(compute_speed)
# 分组计算加速度
data = data.groupby('vehicle_id').apply(compute_acceleration)
# 使用均值填充缺失值
data['speed'] = data.groupby('vehicle_id')['speed'].transform(lambda x: x.fillna(x.mean()))
```

```python
# 分组计算速度
data = data.groupby('vehicle_id').apply(compute_speed)
# 分组计算加速度
data = data.groupby('vehicle_id').apply(compute_acceleration)
# 使用均值填充缺失值
data['speed'] = data.groupby('vehicle_id')['speed'].transform(lambda x: x.fillna(x.mean()))
# 选择需要的列并保存到Excel文件
result = data[['time', 'vehicle_id', 'x', 'y', 'speed']].copy()
result.to_excel('C:/Users/86178/Desktop/vehicle_A5.xlsx', index=False)

print("速度已成功计算并保存到 vehicle.xlsx")
Run Cell | Run Below | Debug Cell
#%%    红灯时间(时间节点相减)

import pandas as pd
import numpy as np

# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/vehicle_A1.xlsx', sheet_name=None)

time_points = {'Vehicle_ID': [], 'Time_point1': [], 'Time_point2': []}
red_light_times = {}  # 存储每辆车的红灯时间

for vehicle_id, group in data.items():
    # 计算与信号灯的距离
    group['distance_to_signal'] = np.sqrt((group['x'] - 0)**2 + (group['y'] - 0)**2)
    # 筛选距离信号灯小于等于150的数据
    group = group[group['distance_to_signal'] <= 150].copy()
    if group.empty:
```

```
        red_light_times[vehicle_id] = 0
        continue
    # 提取加速度开始达到减速状态要求的时间点
    deceleration_start = group.loc[(group['acceleration'] < -2) & (group['speed'] > 0), 'time'].min()
    # 提取停车状态的最后一个时间点
    stop_time = group.loc[group['speed'] == 0, 'time'].max()
    time_points['Vehicle_ID'].append(vehicle_id)
    time_points['Time_point1'].append(deceleration_start if not pd.isna(deceleration_start) else np.nan)
    time_points['Time_point2'].append(stop_time if not pd.isna(stop_time) else np.nan)

    # 计算红灯时间
    if pd.isna(deceleration_start) or pd.isna(stop_time):
        red_light_times[vehicle_id] = 0
    else:
        red_light_times[vehicle_id] = stop_time - deceleration_start + 1
# 创建DataFrame
df_time_points = pd.DataFrame(time_points)
# 保存到Excel文件
df_time_points.to_excel('C:/Users/86178/Desktop/time_points_1.xlsx', index=False)
print("Time points saved to 'time_points_1.xlsx'")
# 输出每辆车红灯时间
for vehicle_id, time in red_light_times.items():
    print(f"Vehicle {vehicle_id}: Estimated red light time = {time} time points")

# 计算并输出每个车辆遇到红灯的均值
mean_red_light_time = np.mean(list(red_light_times.values()))
```

```
# 创建DataFrame
df_time_points = pd.DataFrame(time_points)
# 保存到Excel文件
df_time_points.to_excel('C:/Users/86178/Desktop/time_points_1.xlsx', index=False)
print("Time points saved to 'time_points_1.xlsx'")
# 输出每辆车红灯时间
for vehicle_id, time in red_light_times.items():
    print(f"Vehicle {vehicle_id}: Estimated red light time = {time} time points")

# 计算并输出每个车辆遇到红灯的均值
mean_red_light_time = np.mean(list(red_light_times.values()))
max_red_light_time=max(list(red_light_times.values()))
print(f"Mean red light time across all vehicles: {mean_red_light_time} time points")
print(f"Max red light time across all vehicles: {max_red_light_time} time points")

# 保存红灯时间到Excel文件
df_red_light_times = pd.DataFrame(list(red_light_times.items()), columns=['Vehicle_ID', 'Red_Light_Time'])
df_red_light_times.to_excel('C:/Users/86178/Desktop/red_light_times.xlsx', index=False)
print("Red light times saved to 'red_light_times.xlsx'")
```

C: > Users > 86178 > Desktop > ##问题二 第一问 考虑坐标误差后.py > ...

```
##问题二    第一问   考虑坐标误差后
Run Cell | Run Below | Debug Cell
#%%先在原数据中加入速度
import pandas as pd
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/B4.xlsx')
# 按vehicle_id和time排序
data = data.sort_values(by=['vehicle_id', 'time'])
# 定义计算欧式距离的函数
def compute_distance(df):
    df['distance'] = np.sqrt(df['x'].diff()**2 + df['y'].diff()**2)
    df['distance'].fillna(0, inplace=True)  # 填充第一个NaN值
    return df
# 定义计算速度的函数
def compute_speed(df):
    df['speed'] = df['distance'] / df['time_diff']
    return df
# 定义计算加速度的函数
def compute_acceleration(df):
    df['acceleration'] = df['speed'].diff()
    df['acceleration'].fillna(0, inplace=True)  # 填充第一个NaN值
    return df
# 分组计算欧式距离
data = data.groupby('vehicle_id').apply(compute_distance)
# 计算时间差
data['time_diff'] = data.groupby('vehicle_id')['time'].diff()
# 分组计算速度
data = data.groupby('vehicle_id').apply(compute_speed)
```

```python
# 分组计算加速度
data = data.groupby('vehicle_id').apply(compute_acceleration)
# 使用均值填充缺失值
data['speed'] = data.groupby('vehicle_id')['speed'].transform(lambda x: x.fillna(x.mean()))
# 选择需要的列并保存到Excel文件
result = data[['time', 'vehicle_id', 'x', 'y', 'speed']].copy()
result.to_excel('C:/Users/86178/Desktop/vehicle_B4.xlsx', index=False)
print("速度已成功计算并保存到 vehicle.xlsx")

Run Cell | Run Above | Debug Cell
#%%  加入随机误差
import pandas as pd
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/vehicle_B4.xlsx')
# 随机生成坐标误差
def generate_error():
    return np.random.normal(0, 0.03)
# 对speed不为0的行进行坐标随机误差
non_zero_speed = data[data['speed'] != 0]
non_zero_speed['error'] = non_zero_speed.apply(lambda x: generate_error(), axis=1)
# 更新x和y值
#non_zero_speed['x'] *= (1 + non_zero_speed['error'])
non_zero_speed['y'] *= (1 + non_zero_speed['error'])
# 更新原始数据
data.loc[non_zero_speed.index, ['x', 'y']] = non_zero_speed[['x', 'y']]
# 保存更新后的数据
data.to_excel('C:/Users/86178/Desktop/vehicle_B4_updated.xlsx', index=False)
print("数据已成功更新并保存到 vehicle_B4_updated.xlsx")

#%%求新坐标速度
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/vehicle_B1_updated.xlsx')
# 定义计算速度的函数
def compute_speed(df):
    df['speed'] = np.sqrt(df['x'].diff()**2 + df['y'].diff()**2)
    return df
# 定义计算加速度的函数
def compute_acceleration(df):
    df['acceleration'] = df['speed'].diff()
    df['acceleration'].fillna(0, inplace=True)  # 填充第一个NaN值
    return df
# 初始化Excel writer
with pd.ExcelWriter('C:/Users/86178/Desktop/vehicle_B1_11.xlsx') as writer:
    for vehicle_id, group in data.groupby('vehicle_id'):
        # 计算速度
        group = compute_speed(group)
        # 计算加速度
        group = compute_acceleration(group)
        # 使用均值填充缺失值
        group['speed'].fillna(group['speed'].mean(), inplace=True)
        # 将速度、加速度、x 和 y 数据保存到Excel的不同sheet中
        group[['time', 'x', 'y','speed', 'acceleration']].to_excel(writer, sheet_name=f'vehicle_{vehicle_id}', index=
Run Cell | Run Above | Debug Cell
#%%  红灯时间(时间节点相减)
import pandas as pd
import numpy as np
```

```python
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/vehicle_B1_11.xlsx', sheet_name=None)
time_points = {'Vehicle_ID': [], 'Time_point1': [], 'Time_point2': []}
red_light_times = {}  # 存储每辆车的红灯时间
for vehicle_id, group in data.items():
    # 计算与信号灯的距离
    group['distance_to_signal'] = np.sqrt((group['x'] - 0)**2 + (group['y'] - 0)**2)
    # 筛选距离信号灯小于等于150的数据
    group = group[group['distance_to_signal'] <= 50].copy()
    if group.empty:
        red_light_times[vehicle_id] = 0
        continue
    # 提取加速度开始达到减速状态要求的时间点
    deceleration_start = group.loc[(group['acceleration'] < -4) & (group['speed'] > 0), 'time'].min()
    # 提取停车状态的最后一个时间点
    stop_time = group.loc[group['speed'] == 0, 'time'].max()
    time_points['Vehicle_ID'].append(vehicle_id)
    time_points['Time_point1'].append(deceleration_start if not pd.isna(deceleration_start) else np.nan)
    time_points['Time_point2'].append(stop_time if not pd.isna(stop_time) else np.nan)
    # 计算红灯时间
    if pd.isna(deceleration_start) or pd.isna(stop_time):
        red_light_times[vehicle_id] = 0
    else:
       red_light_times[vehicle_id] = stop_time - deceleration_start + 1
# 创建DataFrame
df_time_points = pd.DataFrame(time_points)
# 保存到Excel文件
df_time_points.to_excel('C:/Users/86178/Desktop/time_pointsB1.xlsx', index=False)
print("Time points saved to 'time_points.xlsx'")
# 输出每辆车的红灯时间
for vehicle_id, time in red_light_times.items():
    print(f"Vehicle {vehicle_id}: Estimated red light time = {time} time points")
# 计算并输出每个车辆遇到红灯的均值
mean_red_light_time = np.mean(list(red_light_times.values()))
max_red_light_time=max(list(red_light_times.values()))
print(f"Mean red light time across all vehicles: {mean_red_light_time} time points")
print(f"Max red light time across all vehicles: {max_red_light_time} time points")
```

Run Cell | Run Below | Debug Cell

```python
#%% 样本比例估计
import pandas as pd
import numpy as np
import random
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/支撑材料/第一问各车辆速度与加速度/vehicle_A1.xlsx', sheet_name=None)
# 抽样比例范围
sample_rates = range(30, 91)
max_red_light_times = {'抽样比例': [], '红灯时间': []}
for p in sample_rates:
    # 随机选择车辆数据
    sample_vehicle_ids = random.sample(list(data.keys()), int(len(data.keys()) * p / 100))
    red_light_periods = []
    for vehicle_id in sample_vehicle_ids:
        group = data[vehicle_id].copy()
        # 计算与信号灯的距离
        group['distance_to_signal'] = np.sqrt((group['x'] - 0)**2 + (group['y'] - 0)**2)
        # 筛选距离信号灯小于等于150的数据
        group = group[group['distance_to_signal'] <= 150].copy()
        if group.empty:
            red_light_periods.append(0)
            continue
        # 提取加速度开始达到减速状态要求的时间点
        deceleration_start = group.loc[(group['acceleration'] < -2) & (group['speed'] > 0), 'time'].min()
        # 提取停车状态的最后一个时间点
        stop_time = group.loc[group['speed'] == 0, 'time'].max()
        # 计算红灯时间
        if pd.isna(deceleration_start) or pd.isna(stop_time):
            red_light_period = 0
```

```
            red_light_period = 0
        else:
            red_light_period = stop_time - deceleration_start + 1

        red_light_periods.append(red_light_period)
    # 记录每次抽样中的最大红灯时间
    max_red_light_time = max(red_light_periods)
    max_red_light_times['抽样比例'].append(p)
    max_red_light_times['红灯时间'].append(max_red_light_time)
# 创建DataFrame
df_max_red_light_times = pd.DataFrame(max_red_light_times)
# 保存到Excel文件
df_max_red_light_times.to_excel('C:/Users/86178/Desktop/max_1.xlsx', index=False)
print("Max red light times saved to 'max_red_light_times.xlsx'")

Run Cell | Run Above | Debug Cell
#%% 画散点图
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
# 原始总体数据
overall_red_light_times = [75, 64, 82, 72, 64]
df = pd.read_excel('C:/Users/86178/Desktop/样本比例估计.xlsx')
# 计算相对误差
for column in df.columns[1:]:
    df[f'相对误差_{column}'] = df[column].apply(lambda x: (x - overall_red_light_times[df.columns.get_loc(column)-1]) / overall_red_light_times[df.columns.get_loc
    (column)-1])
# 计算每行的平均相对误差
df['平均相对误差'] = df[df.columns[6:]].mean(axis=1)
# 计算平均相对误差的绝对值

    df[f'相对误差_{column}'] = df[column].apply(lambda x: (x - overall_red_light_times[df.columns.get_loc(column)-1]) / ov
    (column)-1])
# 计算每行的平均相对误差
df['平均相对误差'] = df[df.columns[6:]].mean(axis=1)
# 计算平均相对误差的绝对值
df['平均相对误差绝对值'] = df['平均相对误差'].abs()
# 绘制抽样比例与平均相对误差的连线图
plt.figure(figsize=(10, 6), dpi=500)
plt.plot(df['抽样比例'], df['平均相对误差绝对值'],  linestyle='-',c='coral')
plt.xlabel('抽样比例', fontsize=12)  # 修改字体大小
plt.ylabel('平均相对误差绝对值', fontsize=12)  # 修改字体大小
plt.title('抽样比例与平均相对误差绝对值关系', fontsize=14)  # 修改字体大小
plt.grid(False)
plt.show()
# 找到最低相对误差对应的抽样比例和最低相对误差值
min_index = df['平均相对误差绝对值'].idxmin()
min_sample_ratio = df.loc[min_index, '抽样比例']
min_relative_error = df.loc[min_index, '平均相对误差绝对值']
print(f"最低相对误差对应的抽样比例为：{min_sample_ratio}")
print(f"最低相对误差为：{min_relative_error}")
```

C: > Users > 86178 > Desktop > #问题三 求红灯时间.py > ...

```
#问题三    求红灯时间
Run Cell | Run Below | Debug Cell
#%%求新坐标速度
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/第四问分类好/D12.xlsx')
# 定义计算速度的函数
def compute_speed(df):
    df['speed'] = np.sqrt(df['x'].diff()**2 + df['y'].diff()**2)
    return df
# 定义计算加速度的函数
def compute_acceleration(df):
    df['acceleration'] = df['speed'].diff()
    df['acceleration'].fillna(0, inplace=True)  # 填充第一个NaN值
    return df
# 初始化Excel writer
with pd.ExcelWriter('C:/Users/86178/Desktop/第四问速度/vehicle_D12.xlsx') as writer:
    for vehicle_id, group in data.groupby('vehicle_id'):
        # 计算速度
        group = compute_speed(group)
        # 计算加速度
        group = compute_acceleration(group)
        # 使用均值填充缺失值
        group['speed'].fillna(group['speed'].mean(), inplace=True)
        # 将速度、加速度、x 和 y 数据保存到Excel的不同sheet中
        group[['time', 'x', 'y','speed', 'acceleration']].to_excel(writer, sheet_name=f'vehicle_{vehicle_id}', index=False)
Run Cell | Run Above | Debug Cell
#%%  红灯时间(时间节点相减)
import pandas as pd
```

```
Run Cell | Run Above | Debug Cell
#%%  红灯时间(时间节点相减)
import pandas as pd
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/第四问速度/vehicle_D12.xlsx', sheet_name=None)
red_light_times = {}  # 存储每辆车的红灯时间
for vehicle_id, group in data.items():
    # 计算与信号灯的距离
    group['distance_to_signal'] = np.sqrt((group['x'] - 0)**2 + (group['y'] - 0)**2)
    # 筛选距离信号灯小于等于150的数据
    group = group[group['distance_to_signal'] <= 50].copy()
    if group.empty:
        red_light_times[vehicle_id] = 0
        continue
    # 提取加速度开始达到减速状态要求的时间点
    deceleration_start = group.loc[(group['acceleration'] < -4) & (group['speed'] > 0), 'time'].min()
    # 提取停车状态的最后一个时间点
    stop_time = group.loc[group['speed'] == 0, 'time'].max()
    # 计算红灯时间
    if pd.isna(deceleration_start) or pd.isna(stop_time):
        red_light_times[vehicle_id] = 0
    else:
        red_light_times[vehicle_id] = stop_time - deceleration_start + 1
# 创建DataFrame
df_red_light_times = pd.DataFrame(list(red_light_times.items()), columns=['Vehicle_ID', 'Red_Light_Time'])
# 保存到Excel文件
df_red_light_times.to_excel('C:/Users/86178/Desktop/第四问各口每辆车红灯时间/red_light_timesD12.xlsx', index=False)
print("Red light times saved to 'red_light_times.xlsx'")


# 保存到Excel文件
df_red_light_times.to_excel('C:/Users/86178/Desktop/第四问各口每辆车红灯时间/red_light_timesD12.xlsx', index=False)
print("Red light times saved to 'red_light_times.xlsx'")
Run Cell | Run Above | Debug Cell
#%%提取三个时间点
import pandas as pd
import numpy as np
# 读取数据
data = pd.read_excel('C:/Users/86178/Desktop/第四问速度/vehicle_D12.xlsx', sheet_name=None)
time_points = {'Vehicle_ID': [], 'Time_point1': [], 'Time_point2': [], 'Time_point3': []}  # 添加Time_point3
# 初始化字典来存储每辆车的红灯时间
red_light_times = {}
for vehicle_id, group in data.items():
    # 计算与信号灯的距离
    group['distance_to_signal'] = np.sqrt((group['x'] - 0)**2 + (group['y'] - 0)**2)
    # 筛选距离信号灯小于等于150的数据
    group = group[group['distance_to_signal'] <= 150].copy()
    if group.empty:
        red_light_times[vehicle_id] = 0
        continue
    # 提取加速度开始达到减速状态要求的时间点
    deceleration_start = group.loc[(group['acceleration'] < -2) & (group['speed'] > 0), 'time'].min()
    # 提取速度第一次变为0的时间点
    first_zero_speed_time = group.loc[group['speed'] == 0, 'time'].min()
    # 提取停车状态的最后一个时间点
    stop_time = group.loc[group['speed'] == 0, 'time'].max()
    time_points['Vehicle_ID'].append(vehicle_id)
    time_points['Time_point1'].append(deceleration_start if not pd.isna(deceleration_start) else np.nan)
    time_points['Time_point2'].append(stop_time if not pd.isna(stop_time) else np.nan)

    # 提取加速度开始达到减速状态要求的时间点
    deceleration_start = group.loc[(group['acceleration'] < -2) & (group['speed'] > 0), 'time'].min()
    # 提取速度第一次变为0的时间点
    first_zero_speed_time = group.loc[group['speed'] == 0, 'time'].min()
    # 提取停车状态的最后一个时间点
    stop_time = group.loc[group['speed'] == 0, 'time'].max()
    time_points['Vehicle_ID'].append(vehicle_id)
    time_points['Time_point1'].append(deceleration_start if not pd.isna(deceleration_start) else np.nan)
    time_points['Time_point2'].append(stop_time if not pd.isna(stop_time) else np.nan)
    time_points['Time_point3'].append(first_zero_speed_time if not pd.isna(first_zero_speed_time) else np.nan)  # 添加Time_point
# 创建DataFrame
df_time_points = pd.DataFrame(time_points)
# 保存到Excel文件
df_time_points.to_excel('C:/Users/86178/Desktop/附件4时间点/D12time.xlsx', index=False)
print("Time points saved to 'time_points.xlsx'")
```

```
Run Cell | Run Above | Debug Cell
#%%  分类
import pandas as pd
# 读取数据
file_path = 'C:/Users/86178/Desktop/vehicles_start_end_slope.xlsx'
df = pd.read_excel(file_path)
# 去除坐标中的括号和逗号，并将其转换为浮点数
df['起点坐标_x'] = df['起点坐标'].apply(lambda x: float(x.split(',')[0][1:]))
df['斜率'] = df['斜率'].astype(float)
# 定义分类条件
categories = {
    '第一类:A到C': (df['起点坐标_x'] < -490) & (df['斜率'] > -0.3) & (df['斜率'] < 0),
    '第二类:A到B': (df['起点坐标_x'] < -490) & (df['斜率'] > 0.7) & (df['斜率'] < 1),
    '第三类:A到D': (df['起点坐标_x'] < -490) & (df['斜率'] < -1),
    '第四类:B到D': (df['起点坐标_x'] > 48) & (df['起点坐标_x'] <100) & (df['斜率'] > 1),
    '第五类:B到C': (df['起点坐标_x'] > 48) & (df['起点坐标_x'] <100) & (df['斜率'] < -1),
    '第六类:B到A': (df['起点坐标_x'] > 48) & (df['起点坐标_x'] < 100) & (df['斜率'] > 0.7) & (df['斜率'] < 1),
    '第七类:C到A': (df['起点坐标_x'] > 490) & (df['斜率'] > -0.3) & (df['斜率'] < 0),
    '第八类:C到D': (df['起点坐标_x'] > 490) & (df['斜率'] > 0.7) & (df['斜率'] < 1),
    '第九类:C到B': (df['起点坐标_x'] > 490) & (df['斜率'] < -1),
    '第十类:D到B': (df['起点坐标_x'] < -40) & (df['起点坐标_x'] > -100) & (df['斜率'] > 1),
    '第十一类:D到A': (df['起点坐标_x'] < -40) & (df['起点坐标_x'] > -100) & (df['斜率'] < -1),
    '第十二类:D到C': (df['起点坐标_x'] < -40) & (df['起点坐标_x'] > -100) & (df['斜率'] > 0.7) & (df['斜率'] < 1)
}

# 分类并保存结果到新的Excel文件
result_df = pd.DataFrame(index=df['车辆ID'])
for category, condition in categories.items():
    vehicles = df[condition]['车辆ID']
---
# 分类并保存结果到新的Excel文件
result_df = pd.DataFrame(index=df['车辆ID'])
for category, condition in categories.items():
    vehicles = df[condition]['车辆ID']
    result_df[category] = result_df.index.isin(vehicles).astype(int)
# 检查是否有车被分进多类
duplicated_vehicles = result_df.sum(axis=1) > 1
duplicated_vehicles_list = result_df[duplicated_vehicles].index.tolist()
if duplicated_vehicles_list:
    print(f"以下车辆被分到了多个类别: {', '.join(duplicated_vehicles_list)}")
else:
    print("所有车辆都被正确分类，没有车辆被分到多个类别。")
# 检查是否每一行只出现一次
sum_of_categories = result_df.sum(axis=1)
if (sum_of_categories == 1).all():
    print("每一行只出现一次，分类完全正确。")
else:
    print("分类存在问题，有车辆未被正确分类。")
# 保存到Excel文件
output_file_path = 'C:/Users/86178/Desktop/第四问分类好/vehicles_classification.xlsx'
result_df.to_excel(output_file_path, engine='openpyxl')
Run Cell | Run Above | Debug Cell
#%%放入分类数据
import pandas as pd
# 读取分类结果
file_path = 'C:/Users/86178/Desktop/第四问分类结果.xlsx'
classification_df = pd.read_excel(file_path)
#%%放入分类数据
import pandas as pd
# 读取分类结果
file_path = 'C:/Users/86178/Desktop/第四问分类结果.xlsx'
classification_df = pd.read_excel(file_path)
# 读取原始数据
data_file_path = 'C:/Users/86178/Desktop/D.xlsx'
data_df = pd.read_excel(data_file_path)
# 将data_df['vehicle_id']转换为字符串类型
data_df['vehicle_id'] = data_df['vehicle_id'].astype(str)
# 获取分类类别列表
categories = classification_df.columns.tolist()
# 针对每一列，筛选数据并保存到对应的Excel文件中
for category in categories:
    # 获取该类别下的车辆ID
    vehicle_ids = classification_df[category].dropna().astype(int).astype(str).tolist()
    # 根据车辆ID筛选原始数据
    filtered_data = data_df[data_df['vehicle_id'].isin(vehicle_ids)]
    # 保存筛选后的数据到Excel文件
    output_file_path = f'C:/Users/86178/Desktop/第四问分类好/{category}.xlsx'
    filtered_data.to_excel(output_file_path, index=False)
    print(f"保存了{category}类的数据到 {output_file_path}")
```

```r
install.packages("dbscan")
library(readxl)
library(fpc)
library(dbscan)
library(ggplot2)
library(cluster)

data = read_xlsx("C:/Users/Administrator/Desktop/各路口各车辆时间点/A1time.xlsx")
# 提取开始减速时点和停止时长作为特征
data = na.omit(data)
data$Time_point1 = scale(data$Time_point1)
data$Time_point2 = scale(data$Time_point2)
data$Time_point3 = scale(data$Time_point3)
features = data[,c("Time_point1","Time_point2","Time_point3")]

num_random_search = 100  # 随机搜索的次数
eps_values <- seq(0.01, 0.2, length.out = num_random_search)  # 确保eps覆盖合适的范围
minPts_values <- seq(1, 10, length.out = num_random_search)  # 确保minPts覆盖合适的范围
# 初始化最优参数和最优轮廓系数
best_params <- list(eps = NULL, minPts = NULL)
best_silhouette <- -Inf

for (i in 1:num_random_search) {
  eps <- eps_values[i]
  minPts <- minPts_values[i]

  # 使用当前参数进行 DBSCAN 聚类
  dbscan_result <- dbscan(features, eps = eps, MinPts = minPts)
  cluster_labels <- dbscan_result$cluster

  if (length(unique(cluster_labels)) > 1) {
    silhouette_result <- silhouette(cluster_labels, dist(features))
    silhouette_avg = mean(silhouette_result[,"sil_width"])
  } else {
    silhouette <- Inf
  }
  # 如果当前轮廓系数优于当前最佳轮廓系数，则更新最佳参数
  if (silhouette_avg > best_silhouette) {
    best_params$eps <- eps
    best_params$minPts <- minPts
    best_silhouette <- mean(silhouette)
  }
}
print(best_params)
print(best_silhouette)

# 输出聚类结果
print(cluster_labels)
silhouette_result = silhouette(cluster_labels,dist(features))
silhouette_avg = mean(silhouette_result[,"sil_width"])

# 绘制散点图，显示不同参数组合下的聚类结果
windows()
plot_df <- data.frame(eps = rep(eps_values, each = length(minPts_values)),
                      minPts = rep(minPts_values, length(eps_values)),
                      cluster_labels = unlist(cluster_results))
ggplot(plot_df, aes(x = eps, y = minPts, color = factor(cluster_labels))) +
  geom_point() +
  scale_color_manual(values = c("grey", rainbow(max(plot_df$cluster_labels)))) +
  labs(title = "DBSCAN Clustering with Different Parameters",
       x = "eps", y = "minPts")

windows()
cluster_df  =data.frame(features,cluster  =as.factor(cluster_labels))
ggplot(cluster_df, aes(x = Time_point1, y = Time_point2, color = cluster)) +
  geom_point() +
  labs(title = "DBSCAN Clustering Result of A1",
       x = "Time Point 1",
       y = "Time Point 2",
       color = "Cluster") +
  theme_minimal()+ theme(panel.grid = element_blank(),  # 去掉网格线
                         text = element_text(size = 12, face = "bold"),
                         plot.title = element_text(hjust = 0.5))  # 设置标题居中)  # 调整文字大小和粗细

```