



中国研究生创新实践系列大赛
“华为杯”第二十一届中国研究生
数学建模竞赛

学 校 杭州电子科技大学

参赛队号 24103360004

1. 李俊辉

队员姓名

2. 陈威

3. 孔令聪

中国研究生创新实践系列大赛

“华为杯”第二十一届中国研究生

数学建模竞赛

题 目：高速公路应急车道紧急启用模型

摘要：

目前的高速公路中频繁发生道路拥堵现象，在一些道路合流处尤其严重。道路的拥堵会带来道路利用率低下、经济损失等负面影响。面对该拥堵问题，能否高效利用应急车道成为缓解道路拥堵的重要因素。传统的启用方式主要以经验为主，存在资源浪费、缓解效果不理想等弊端。本文通过分析交通流的主要参数，利用数据挖掘、数据分析等到规律，并构建合理的预测模型、评估模型、优化模型等，建模出一个高效和合理的应急车道启用模型。

数据预处理：通过 **yolov8** 和 **BoT-SORT** 多目标跟踪算法实现了对画面中出现所有车辆的 id 记录与轨迹跟踪，由于题目未给出道路上的距离标识，对道路上的参照物进行长度估计并假设后计算出本文所需信息，获得所需信息如速度、密度、车流量等后，对数据进行平滑处理，并使用箱线图清理异常数据。

针对问题一第一问：经过数据预处理，本文统计并绘制了四个观测点中车流量、车辆速度与车辆密度随时间的变化的散点图。进一步通过对这三个交通流参数的 **Spearman** 相关系数分析，发现在所有摄像头中，流量与密度的系数呈现正值，流量与车速的系数呈现负值，车速与密度的系数呈现负值，且 **p 值均小于 0.05**。推导得出以下结论：流量与密度之间存在显著的正相关性，车流量与车辆速度以及车辆速度与车辆密度之间表现出负相关性。这些发现为理解交通流特性及其相互关系提供了重要的理论依据。

针对问题一第二问：为了对第三点到第四点之间路段可能出现持续拥堵状态进行预警，本文构建了**交通拥堵实时预警模型**。基于视频数据求出拥堵系数，为了得到交通拥堵系数与道路是否拥堵的映射关系，需要求出拥堵阈值：利用**三维聚类算法 K-means** 在速度、密度、流量三维数据散点图中进行聚类，发现车速慢和密度大会形成较密集的簇，再统计这些聚类对应的拥堵系数发现交通拥堵系数分界点为 **8 左右**。为了验证聚类产生的结果准确性利用元胞自动机进行仿真，得到相同规律。基于以上数据构建交通拥堵实时预警模型：若 **108** 观测点出现交通拥堵系数大于阈值或即将大于阈值，且前两个观测点都处于交通拥堵系数大于阈值中，则认为第三点到第四点之间路段即将出现持续拥堵。

针对问题一第三问：在每一个站点选取第二个视频数据作为验证集，先以交通拥堵指数 TPI 阈值为 10 将三维散点图分类，再对比使用 **kmeans** 三维聚类算法的二分类结果，可以发现分类结果基本相似，两种类别的特征也符合交通拥堵与畅通的特征，这证明了阈值在视频数据中的有效性。使用交通拥堵实时预警模型对视频数据进行预测，可以发现从 **11:35 到 12:50 左右**，108 及前两个观测点交通拥堵系数长时间处于 8 以上，根据交通拥堵实时预警模型可以推断出从 **11:35 到 12:50** 这个时间段及以后 103 观测点也是拥堵的，而视频数据也能证实这一点。

针对问题二：构建应急车道启用模型为决策者提供临时启用应急车道决策，模型会统计四个观测点不同速度与密度情况下的拥堵概率，再结合交通拥堵实时预警模型得出**车速为 50 千米/时，或者车辆密度高于 200 辆/千米时，道路有高于 90% 的概率发生拥堵**。因此当**车速低于 50 千米/时或者车辆密度高于 200 辆/千米**需要决策者开启应急车道缓解拥堵，并且进一步考虑到道路上的复杂情况，当车速下降到**70 千米/时或者车辆密度高于 180 辆/千米**需要决策者先查看应急车道上的情况，再决策是否开启应急车道。

针对问题三：为了实时决策是否启用应急车道，本文采用大语言预测模型 **LSTPromt**，首先将拥堵指数与时间构成的时间序列数据变成文本数据，将要预测的时间跨度分为长时间和短时间预测，最后在模型使用之前，输入提示词“交通流”、“道路拥堵”，“道路突发情况”等，基于每个观测点当前数据对未来几分钟的数据进行预测，并将预测数据交予交通拥堵实时预警模型进行判断是否启用应急车道。当预测出现拥堵情况下，开启应急车道，并且对开启应急车道后的情况用元胞自动机进行了仿真，可以发现道路拥堵并开启应急车道后，车速提高为原来的 **3/2** 并且逐渐接自由流速度。

针对问题四：由于 103 观测点的应急车道通往服务区，从而导致该点的车流负荷能力较低，为了在控制成本的同时也能提升该路段应急车道临时启用决策的科学性，本文提出了**前后 1/4 道路预警法**。具体描述为，在道路的前 1/4 点位和最后 1/4 点位布置观测点 1、2。观测点 1 的作用是及时监控前 1/4 预警路段上的**交通拥堵指数 TPI**，当 TPI 大于 6 且趋势上升时，表明该路段将要发生拥堵，则提前开启应急车道，使这部分车尽早通过，从而分散车流，减少 103 观测点的负荷。观测点 2 的作用是及时监控后 1/4 监控路段上的车流状况，当发现拥堵从 103 观测点蔓延过来，这表明该路段上的车流对 103 观测点的负荷过大，这时需要提前关闭应急车道，从而减缓车流进入该路段的速度，减少 103 观测点的负荷。

关键词： yolov8 多目标跟踪算法 交通流模型 元胞自动机 保形评判 聚类分析

1 问题重述

1.1 问题背景

当前，我国高速公路仍存在着严重的交通堵塞问题，特别是在交叉路口。交通拥挤会造成道路利用率低，经济损失大等一系列问题。如何有效地使用紧急通道是解决交通拥挤的一个关键问题。传统的利用方法以经验为基础，存在着资源浪费和缓解效果不佳的缺点。在此基础上，采用数据挖掘、数据分析等方法，建立相应的预测模型、评价模型、优化模型等方法，建立快速、合理的应急车道设置模型，为决策者合理的决策依据，提高决策效率。

1.2 问题分析

为了缓解交通拥堵，让启动应急车道更加合理，本文依次提出如下问题并进行初步的分析：

问题一第一问：题目要求统计四个观测点的交通流参数随时间的变化规律，以便于建立交通流拥堵模型。交通流参数是评估交通状况和优化交通管理的重要指标，包括车流密度、车流量、车辆速度等。本问题旨在通过对四个观测点的交通流参数随时间的变化规律进行统计，识别潜在的趋势和模式，为交通管理和建模提供数据支持，所以需要从提供的视频数据中提取出车流密度、车流量、车辆速度随时间的变化情况。为了深入分析交通流参数之间是否存在关系，以及与拥堵情况的关系，需要进行这三个参数之间的相关性分析，提取出拥堵数据，为未来的交通建模提供数据支持。

问题一第二问：本文利用交通流在四个不同地方的三个主要交通流参数构建拥堵模型，并且给出预警的依据。先通过上述小问得到三个参数的相关性，计算拥堵指数，设定阈值作为评判的标准，在结合实际情况分析过后，给出发出预警的提示。

问题一第三问：题目要求验证模型的合理性，则需要仿照机器学习验证模型有效性的方法，将视频数据划分为训练集和测试集，在训练集上得到交通流拥堵模型，在测试集上验证模型的有效性。

问题二：题目要求构造应急车道启用模型，本文通过问题一分析得出的交通拥堵模型，给出拥堵的预警提示后告诉决策者应该在何时启用应急车道，但是这里需要考虑到一些特殊情况，比如应急车道有无被突发事故占有等情况。

问题三：在问题二的基础上，制定相应的规则运用到应急车道启用模型，并且对开放应急车道后，记录道路上三个交通流参数的可能变化趋势，并且进行仿真分析。

问题四：题目要求在控制成本的前提下，在三四点之间的路段布置观测点以提升应急车道临时启用决策的科学性。本文对现实路段情况进行分析，并制定合理的观测点布置与数量。

2 模型的假设

为了便于问题的解决，对问题中某些条件进行简化并作合理的假设。

1. 车辆运行过程的速度在短区间内保持匀速或者恒定；
2. 假设采集到的视频数据为真实且可以的数据，检测时不存在或者少存在缺检漏检情况；
3. 统一视频的车辆检测长度为 50 米；
4. 第二车道车辆速度为整体速度；

3 符号说明和名词解释

3.1 符号说明

表 3-1 符号说明

变量（符号）	说明	单位
Q	车流量	辆/小时
V	车速	千米/小时
K	车辆密度	辆/千米
TPI_{full}	判断拥堵的阈值	
$V_{pixel/z}$	速度	像素点/帧
$V_{pixel/s}$	速度	像素点/秒
Z	几帧代表一秒	秒/帧
V_{ave}	平滑化以后的速度	千米/时
n	窗口大小	
V_{0-1}	归一化以后速度	千米/时
V_{max}	时间段内最大速度	千米/时
$V_{simulate}$	模拟速度	千米/时
v_{tpi}	TPI 指数的变化率或斜率	

3.2 名词解释

表 3-2 名词解释

名词名称	含义
TPI	交通拥堵指数
交通流	道路上车辆和行人等交通参与者随时间和空间变化而移动形成的连续流动过程。
yolov8	目标检测算法
BoT-SORT	多目标跟踪算法
TimeDecomp	将时间序列数据划分为短期和长期
TimeBreath	每隔一定窗口大小对历史数据重新调整
保形预测	预测结果提供显式的非渐近保证
帧	本文中专门指元胞一步状态转移的时间单元
Spearman	非参数的统计度量，评估两个变量之间的相关性
数据长尾效应	极端值数据数量多于正常数据的数量
自由流	车辆以理想状态的速度运行

4 数据预处理

4.1 数据集描述

为正确建立高速公路应急车道启用模型，题目给出的数据集为东庐山服务区高速路段 4 个相邻监控点位的视频数据，具体的视频相关信息如表 4-1 所示：

表 4-1 视频数据详细信息

观测点编号	32.31.250.103	32.31.250.105	32.31.250.107	32.31.250.108
视频数量 (个)	2	3	2	2
视频长度 (分钟)	7: 04	7: 04	13: 47	13: 47
视频时间跨度 (小时)	1: 11: 00	1: 11: 00	2: 17: 00	2: 17: 00
视频帧数	25	25	33	33
帧数与真实时间比值	2.5	2.5	3.3	3.3

其中视频长度，视频时间跨度各有细小误差，可忽略不计。帧数与真实时间比值为计算得出，上表数据为四舍五入得到，经过验算不会对时间计算产生太大影响。

4.2 交通流参数描述

从该数据集中，本文获取与交通流有关的三个参数：车辆密度、流量、速度。

1. **流量：**某时刻单位时间内通过某点的车辆数；
2. **车流密度：**某时刻某处单位长度内的车辆数；
3. **车速：**某时刻通过某点的车流速度；

4.3 视频解析流程

本题使用基于 yolov8 的目标跟踪算法，通过该方法可以给视频中出现的每一辆车都编上编号，并记录该车的运动轨迹，简单展示如图 4-1。



图 4-1 视频解析展示图

4.3.1 yolov8 介绍

YOLOv8^[1] 是 Ultralytics 公司于 2023 年 1 月 10 日开源的重大更新版本，基于 YOLOv5 进行改进，支持图像分类、物体检测和实例分割等任务。其主要结构如图所图 4-2 所示。

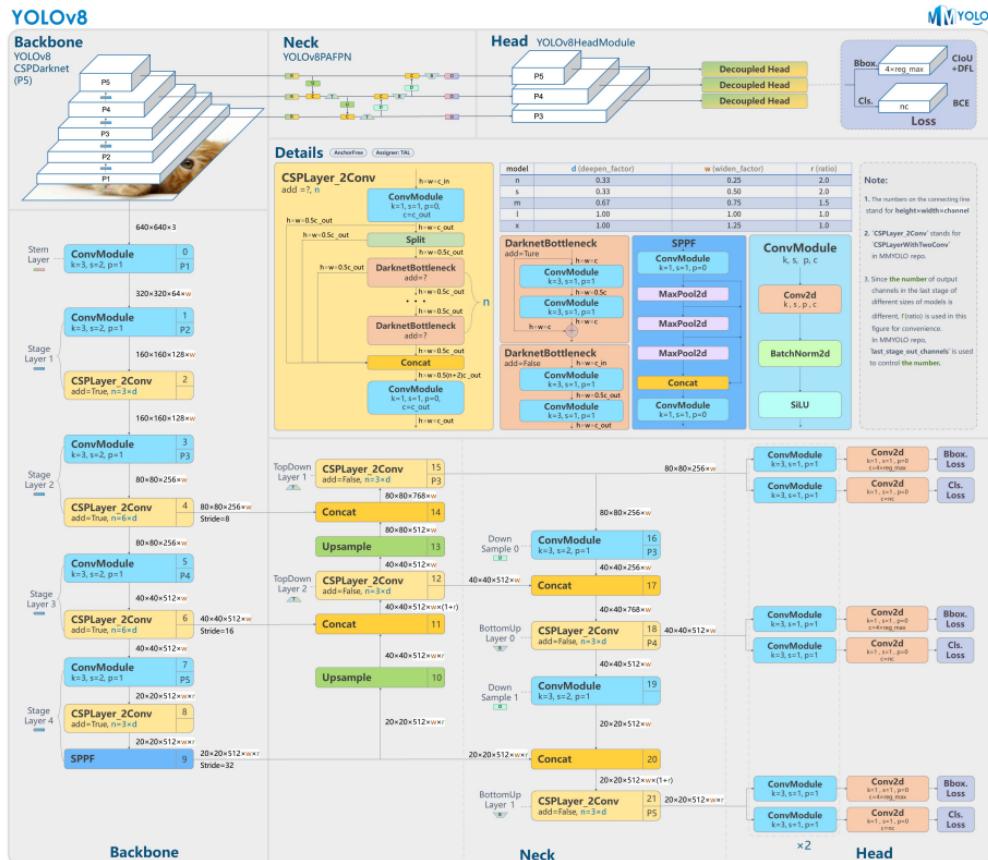


图 4-2 yolov8 模型架构图^[1]

分析该架构图后，可以找到 yolov8 模型的主要的三个部分为基于 CSP 的主干网络 (backbone)，特征增强网络 (neck) 和检测头 (head)。

Backbone 部分采用一系列卷积和反卷积层来提取特征，并利用残差连接和瓶颈结构以减小网络规模并提升性能。它使用 C2f 模块作为基本单元，相比 YOLOv5 的 C3 模块，C2f 模块在参数量上更少且特征提取能力更强。

Neck 部分通过多尺度特征融合技术，将来自 Backbone 不同阶段的特征图融合，增强特征表示能力，其中包括 SPPF 模块、PAA 模块和两个 PAN 模块。

Head 部分负责最终的目标检测和分类任务，包含一个检测头和一个分类头，检测头使用卷积和反卷积层生成检测结果，分类头则采用全局平均池化对特征图进行分类。经过一系列改进，YOLOv8 在保持了 YOLOv5 的优点同时，进行了更加精细的调整和优化。

4.3.2 多目标跟踪算法—BoT-SORT

基于多目标跟踪的 Tracking-by-detection 方法^[2], 通过将这些改进集成到著名的 Byte-Track 中, 算法设计者展示了两个新的先进跟踪器, BT-SORT 和 BoT-SORT-ReID。BoT-SORT-ReID 是 BoT-SORT 的扩展, 包含了 ReID 模块。算法流程如图 4-3 所示。

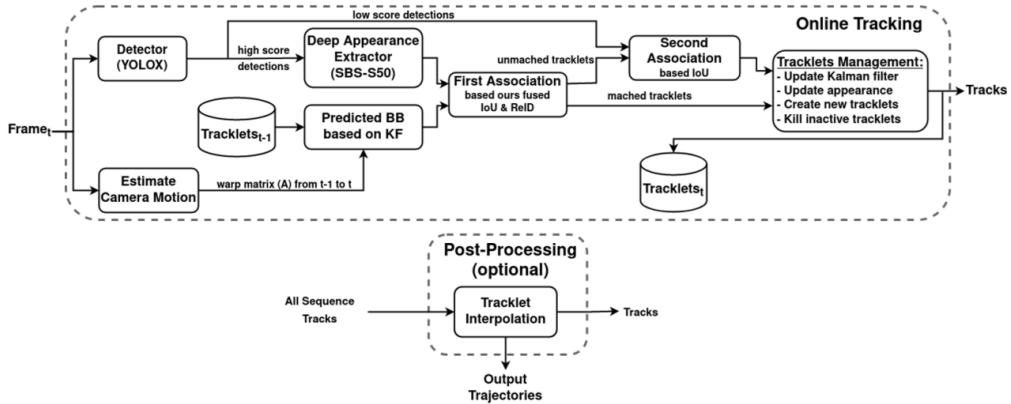


图 4-3 BoT-SORT 算法工作图^[2]

从上述流程图中可以看到 BoT-SORT 结合了运动和外观信息, 利用改进的卡尔曼滤波器、相机运动补偿 (CMC) 和 IoU-Re-ID 融合来提升跟踪性能。在 MOT 挑战中, BoT-SORT 在多个指标上表现优异, 特别是在 MOTA、IDF1 和 HOTA 方面。该算法通过补偿相机运动并融合运动与外观信息, 提高了跟踪的稳健性和准确性。

在本文中, 利用 BoT-SORT 多目标跟踪算法实现了对每一辆车的轨迹跟踪, 从而实现了速度计算。

数据统计方法 由上述基于 yolov8 的目标跟踪算法, 可以获得每一辆车的类型以及移动轨迹, 将类型分为大车与普通小车两种, 并且每个目标的点位之间的保存间隔为 1 帧。由此, 可以对视频数据进行以下信息统计。特别注意, 由于题目所给视频拍摄角度并不是正上方往下拍摄, 以及并没有给出参照物长度, 因此无法获得视频中道路的实际长度, 从而使密度和速度无法正常计算, 在下文中会给出的计算方法。

4.3.3 流量统计

车辆在图中第一次被检测到, 算法会给他分配一个 ID, 统计第一次出现的车辆数量即可实时获得车辆的流量。以下是对每个车道流量的细化统计。图 1 中分别展示了 4 个观测点位的道路截图, 前三个点位可以根据车辆第一次出现的坐标将车辆统计到它的车道中, 第四个点位实时检测车辆轨迹的当前坐标于上一次保存的坐标, 如果两个坐标之间的连线于图中 3 条线段中的某一条相交, 即可获得车辆所属车道, 如图 4-4 所示。



图 4-4 轨迹划分图

4.3.4 密度统计

由于无法得到真实的道路长度，通过观测，视频的车辆检测范围估计为 50 米。统计每一帧图像中存在的被检测出的车辆数量，该数量即可以认为是在这 50 米长的道路上存在的车辆数。经过 LWR 交通流偏微分方程计算可得在这一时刻的车辆密度。

1. 平滑化处理

由于各种因素的影响，包括车辆未完全进入画面就将其检测、由于外侧车辆的遮挡导致部分时刻车辆未被检测到等，会导致车辆速度波动极大。在此采用移动平均法对数据进行平滑化，窗口大小设置为 30（大约为 1 分钟）。这里以第一个节点的第一个视频为例：

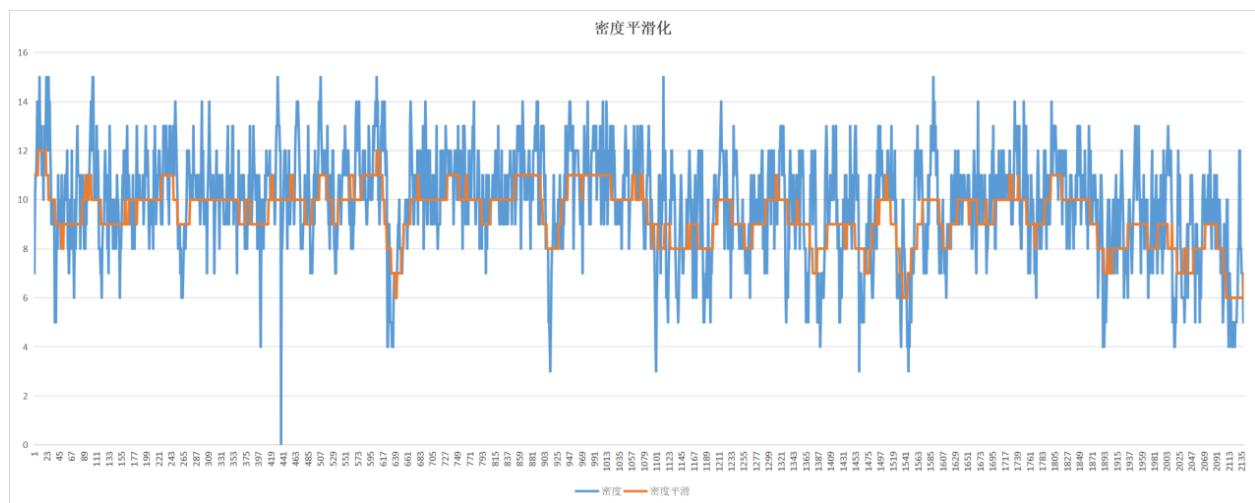


图 4-5 平滑处理结果图

从平滑化结果可以看出，密度平滑后波动减缓但趋势仍旧符合原数据，从而减少了数

据误差。

2. 单位换算

上文定义道路长度统一为 50 米，因此根据该数据将密度单位由原来的辆/秒转化成辆/小时。

4.3.5 速度统计

由于观测视频的角度为侧向拍摄，三个车道像素点代表的实际距离不同，因此统一取第二车道车辆速度为整体速度。速度统计分为以下几步：

1. 计算车辆走过像素点速度

计算给定区域内记录的车辆轨迹点之间的之间距离 (x_i, y_i) 于 (x_{i-1}, y_{i-1})

$$V_{pixel/z} = \sqrt{(x_i, y_i)^2 + (x_{i-1}, y_{i-1})^2} \quad (4.1)$$

$V_{pixel/z}$ 即为这一帧车辆走了多少像素点。再乘以表格中帧数与真实时间比值，得到每一秒车辆走了多少像素点。

$$V_{pixel/s} = V_{pixel/z} \times Z \quad (4.2)$$

2. 平滑处理以及归一化处理

由于各种因素的影响，包括车辆未完全进入画面就将其检测、由于外侧车辆的遮挡导致部分时刻车辆未被检测到等，会导致车辆速度波动极大。在此采用移动平均法对数据进行平滑化，窗口大小设置为 30（大约为 1 分钟）。

$$V_{ave} = \frac{\sum_{i=1}^n V_{pixel/s[i]}}{n} \quad (4.3)$$

这里以第一个节点的第一个视频为例：

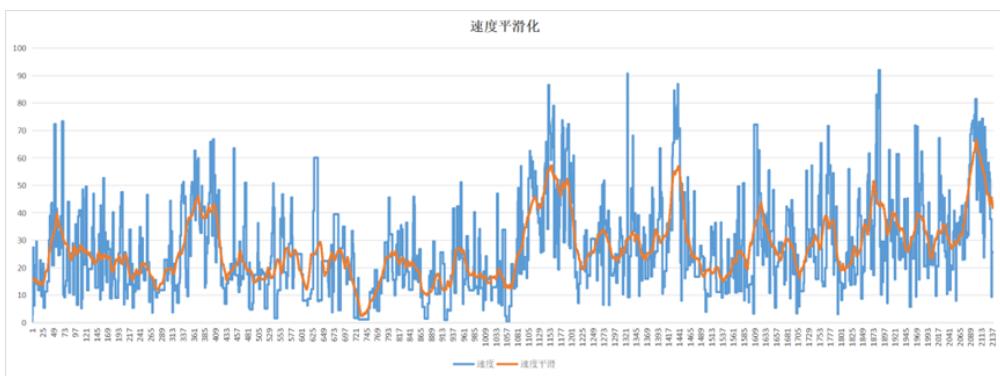


图 4-6 平滑处理图

从平滑化结果可以看出，速度平滑后波动减缓但趋势仍旧符合原数据，从而减少了数据误差。

在步骤一中得到每秒车辆的像素点速度并平滑处理后，对所有速度进行归一化处理

$$V_{0-1} = \frac{V_{ave}}{V_{max}} \quad (4.4)$$

$$V_{max} = \max(V_{ave}) \quad (4.5)$$

其中 V_{max} 为该视频范围内的最大速度 V_{0-1} 。

3. 异常值处理

下图为 9 个数据的速度箱线图，采用均值填充的方式对低于下边缘的数与高于上边缘的数进行处理，异常分析结果如图 4-7 所示。

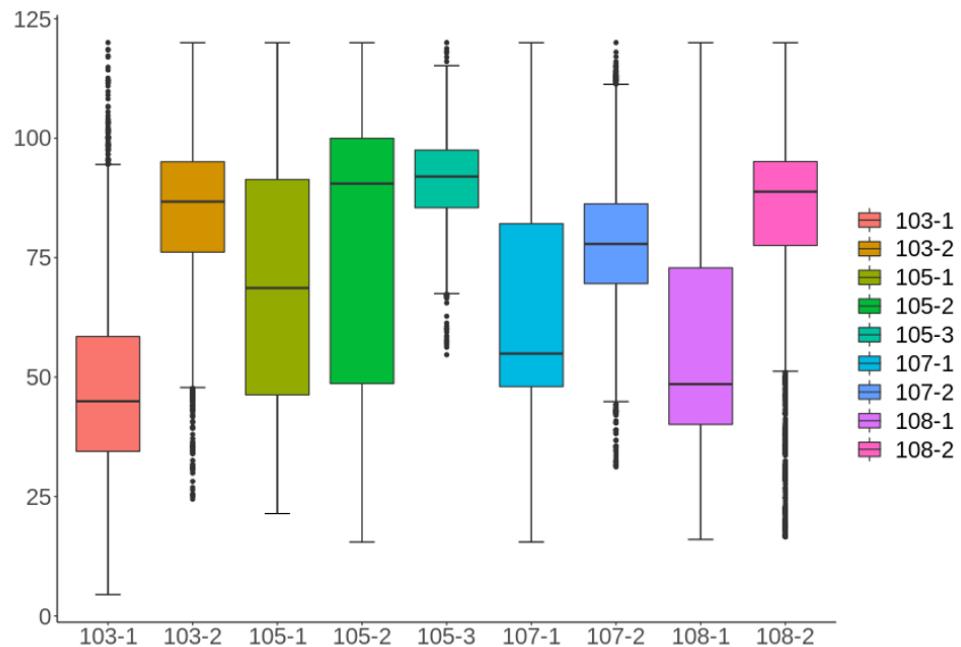


图 4-7 箱线图

4. 模拟真实速度

高速公路二车道道路的慢车道（本题的中间车道）限速为 100，但由生活经验可知往往这条道上的车辆瞬时速度会与快车道相似，因此将归一化以后的数据乘以 120 得到模拟真实速度 V 。公式转换如下：

$$V_{simulate} = V_{0-1} \times 120 \quad (4.6)$$

表 4-2 附件数据集指标情况

数据指标名称
时间 (秒)
第一车道出现的车总数量 (辆)
第一车道出现的大车数量 (辆)
第二车道出现的车总数量 (辆)
第二车道出现的大车数量 (辆)
第三车道出现的车总数量 (辆)
第三车道出现的大车数量 (辆)
车流量 (辆/时)
车辆密度 (辆/千米)
速度 (千米/时)

5 问题求解与模型建立

5.1 问题一第一问

5.1.1 问题分析

根据题意，需要统计出四个观测点的三个交通流参数与时间的关系，并且进一步分析三个参数之间的关系，可以更好的掌握交通流规律。

5.1.2 数据提取

通过题目提供的视频数据，识别出每 5 帧不同车道经过的车辆数、车辆移动的像素数、图上的车辆数分别计算出了 5 帧内的车流量、车辆速度与车辆密度（具体的处理过程在数据预处理部分已详细描述，在这里不再复述）。

5.1.3 数据预处理

为了更好的分析数据规律，需要对数据更进一步处理。根据视频的帧率和画面中的真实时间，计算得到了每一行数据的时间戳。然而，由于按帧识别的数据时间跨度过小，导致数据呈现出高度离散的特征，从而使得数据的规律性难以显现。为此，本文将数据按照 4 分钟作为最短时间尺度进行再处理：首先，累加每 4 分钟内每个车道通过的所有车辆数，

以此计算车流量，单位为辆/4分钟；其次，计算每4分钟内所有车辆的平均速度，单位为公里/小时；最后，计算4分钟内车辆的平均密度，单位为辆/公里。

5.1.4 数据处理

在预处理的数据基础上，通过某一帧的车辆数量计算得到了时刻t的道路占有率，随后，基于时刻t的道路时间占有率、车流量和车辆速度，进一步计算得出了交通拥堵指数TPI，以标记出道路拥堵时刻。

用得到的数据分别绘制车流量、车辆速度与车辆密度随时间变化的散点图，如图5-1所示。

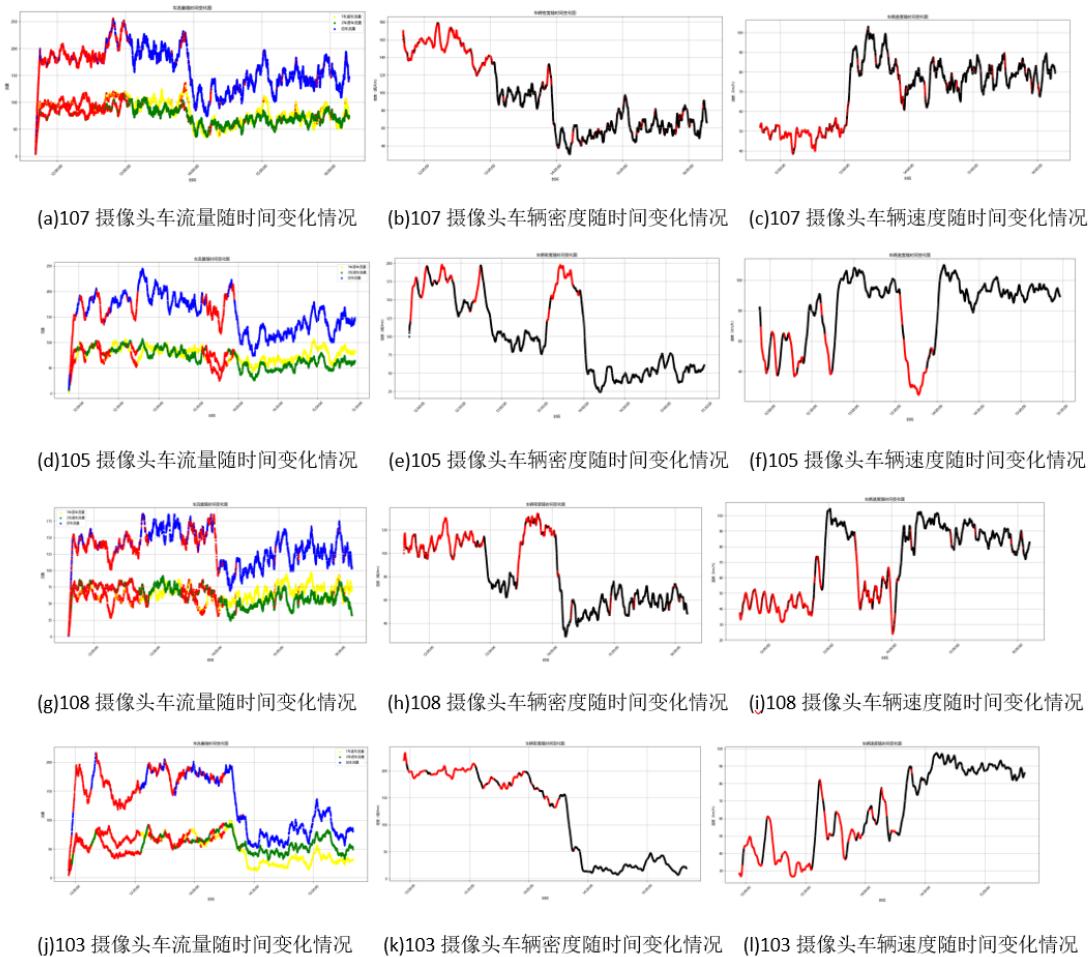


图5-1 车流量、车辆速度与车辆密度随时间变化散点图

5.1.5 数据分析

由图一可见，拥堵情况多出现于车流量较高、车辆密度变化的上升沿、车辆速度变化的下降沿节点。为了进一步探究车流量、车辆速度与车辆密度之间的关系，将每个摄像头的车流量、车辆速度与车辆密度表示在一张图上，如图5-2所示。

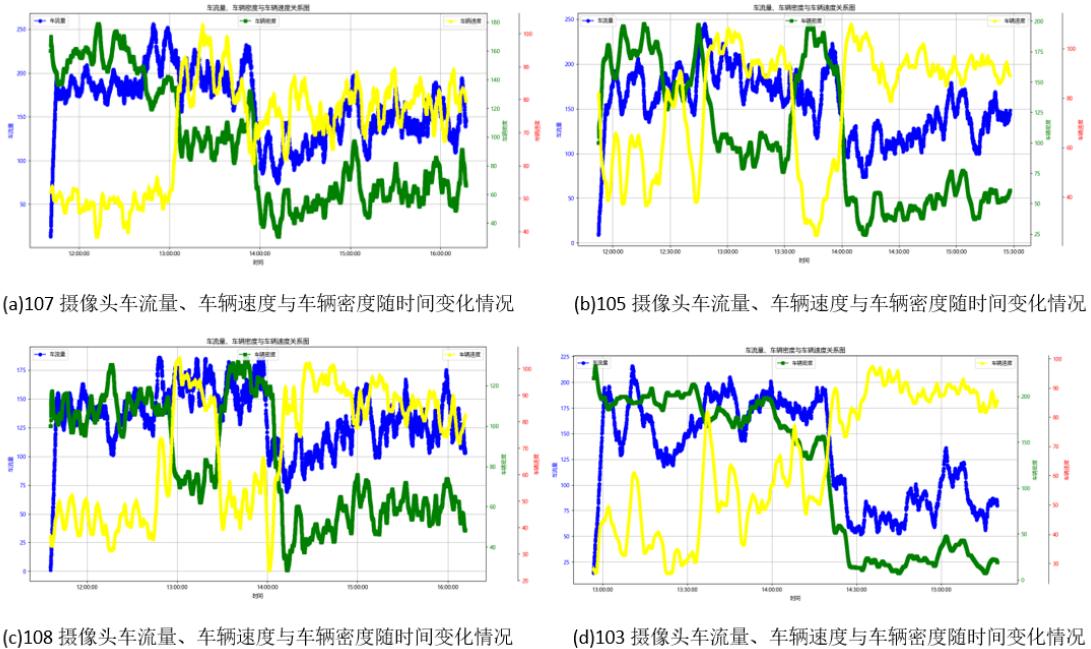


图 5-2 参数关系图

由于车流量、车辆速度与车辆密度三者的计量单位不同，且由于不同类型的车辆的表现差异、个别驾驶者的异常驾驶行为、道路条件变化等因素的影响，数据集中易出现异常值，所以使用 Spearman 相关系数来分析三者之间的相关性将更为合适，因为它能够捕捉到数值之间单调性上的相关性并对数据的分布更具鲁棒性。计算得到在 107 摄像头画面中车流量与车辆密度的 Spearman 相关系数为 0.80，车流量与车辆速度的 Spearman 相关系数为 -0.19，车辆速度与车辆密度的 Spearman 相关系数为 -0.52；在 105 摄像头画面中车流量与车辆密度的 Spearman 相关系数为 0.64，车流量与车辆速度的 Spearman 相关系数为 -0.18，车辆速度与车辆密度的 Spearman 相关系数为 -0.73；在 108 摄像头画面中车流量与车辆密度的 Spearman 相关系数为 0.55，车流量与车辆速度的 Spearman 相关系数为 -0.15，车辆速度与车辆密度的 Spearman 相关系数为 -0.80；在 103 摄像头中车流量与车辆密度的 Spearman 相关系数为 0.66，车流量与车辆速度的 Spearman 相关系数为 -0.54，车辆速度与车辆密度的 Spearman 相关系数为 -0.89，绘制热力图（图 5-3）。

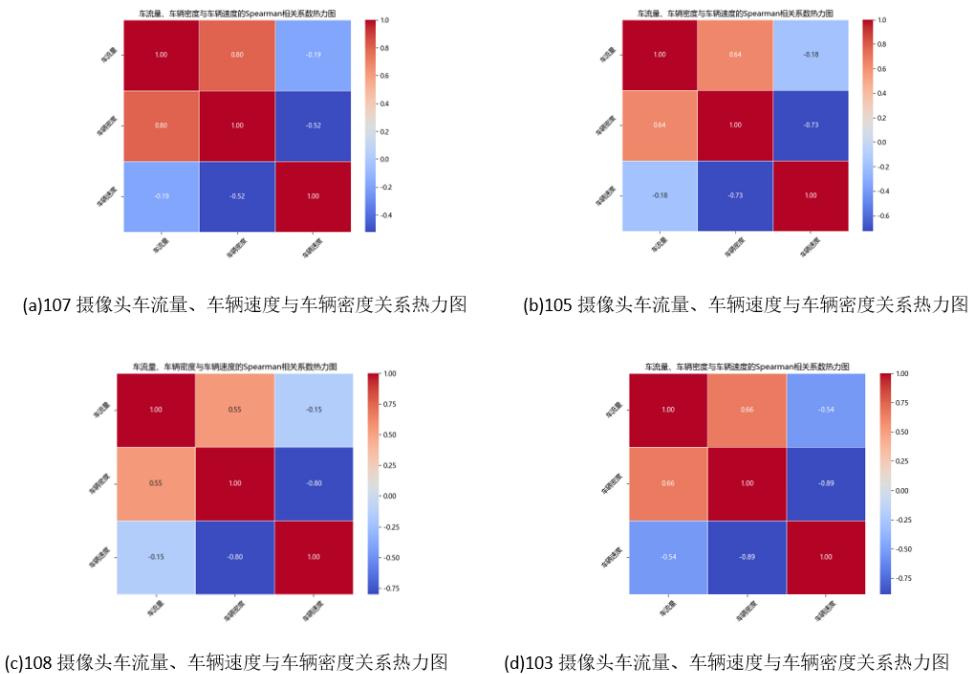


图 5-3 参数关系图

对四个摄像头中车流量、车辆速度与车辆密度之间的 Spearman 相关系数进行分析，可得出以下结论：

1. 车流量与车辆密度呈正相关性：在所有摄像头中，车流量与车辆密度的 Spearman 相关系数均为正值。这表明，随着车流量的增加，车辆密度也呈现出增大的趋势，反映了交通流量和拥堵之间的正相关关系。
2. 车流量与车辆速度呈负相关性：在所有摄像头中，车流量与车辆速度的 Spearman 相关系数均为负值。虽然相关性不强（107 和 105 的值接近于零），但在 103 摄像头中显示出较强的负相关性，这意味着当车流量增加时，车辆速度往往会降低，可能是由于交通拥堵的影响。
3. 车辆速度与车辆密度呈负相关性：在四个摄像头中，车辆速度与车辆密度的 Spearman 相关系数均为负值。随着车辆密度的增加，车辆速度显著降低，尤其在 108 和 103 摄像头中，负相关性较强，表明在高密度条件下，车辆的流动性受限。

5.2 问题一第二问

5.2.1 问题分析

题目需要利用交通流在四个不同地方的三个主要交通流参数构建拥堵模型，并且给出预警的依据。先通过上述小问得到三个参数的相关性，计算拥堵指数，设定阈值作为评判的标准，在结合实际情况分析过后，给出发出预警的提示。

5.2.2 道路状况分析

在对相关参数进行分析之前，本文需要对道路环境发生变化时车流量随之产生的变化有一个直观的了解。比如当道路停用应急车道或者发生交通事故时，本文使用元胞自动机对道路情况进行仿真获得车流随之产生的变化。

5.2.3 元胞自动机

简介

元胞自动机^[3]将时间、空间、状态都看成是离散的数据，利用空间相互作用和时间因果的关系来仿真局部联系和局部规则，具有强大的系统时空演化过程的能力。在交通流的模拟上也有很成熟的运用。

元胞的定义

基本单元为拥有不同状态的车辆，为了简化复杂的车流情况，以及突出三道路变为两道路车辆在不同密度下的拥堵程度，这里定义两种状态，分别为静止、运动。

元胞空间

设计元胞自动机的维度为 3×50 ，其中 3 表示车道数量，50 表示道路长度，在设计过程中为了模拟真实道路情况的变换，第三条道路在后半段被堵塞。

邻居和规则

以某个车辆为例，如果该车辆前面为空地标识，则保持当前速度前进，如果存在障碍物或者其他车辆，则寻找旁边车道进行切换位置，若上述都不满足，则保持静止。在实际操作过程中为了保证元胞之间不发生冲突，遍历顺序选择行驶最远依次到行驶最近。具体的状态转移流程如图 5-4：

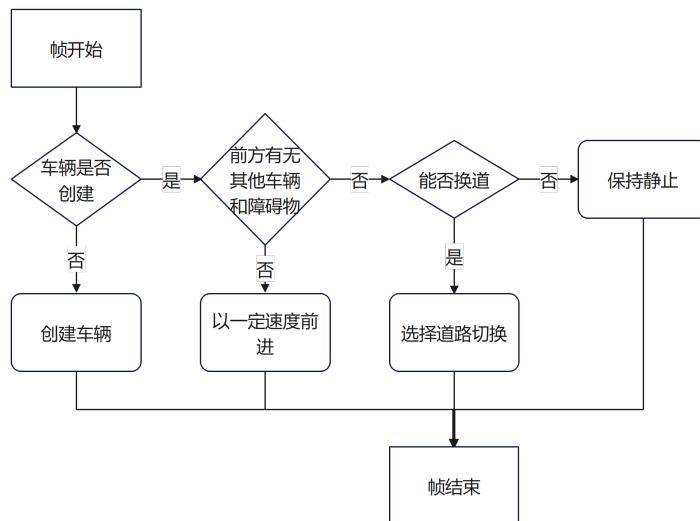


图 5-4 元胞状态转移流程图

边界条件

在设定道路的起始点创建该元胞定义存活状态，到运动到该道路尽头的时候定义死亡状态，后续不需要处理该元胞。

5.2.4 仿真设计

借助 matlab 编程语言，构建了简单的元胞自动机模拟交通流情况，具体展示如图（5-5，5-6）：

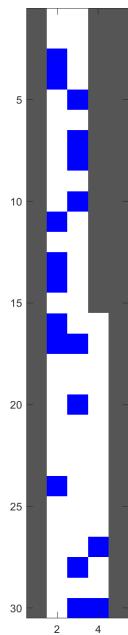


图 5-5 非高峰期车流情况

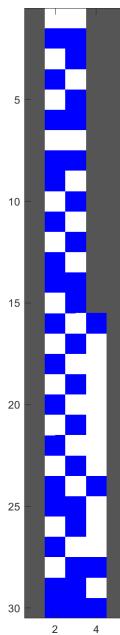


图 5-6 高高峰期车流情况

从以上两张图可以看出在非高峰期的时候，三车道变为两车道对道路拥堵几乎无影响，但是在高峰期的时候会马上发生拥堵。

为了让模拟更真实的路面情况，在仿真中加入了突发的交通事故，仿真结果如图（5-7，5-8）所示：

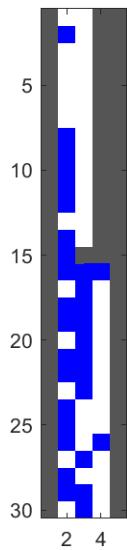


图 5-7 非高峰期车流情况 (有交通事故)

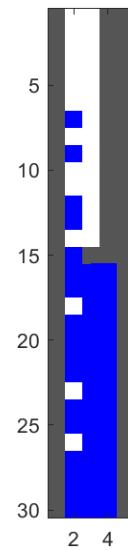


图 5-8 高峰期车流情况 (有交通事故)

为了让交通事故的负面影响效果最明显，在三车道变为二车道的关键处发生事故，具体在地图中加入障碍物标识，那么在以上两张图中可以看出不管是车流量少的时候还是车流量多的时候都会发生拥堵现象，并且在车流多的时候几乎完全拥堵。

5.2.5 仿真结果

通过仿真结果发现，在未发生交通事故的时候只有当车流量比较大的时候会发生拥堵，并且也不会完全拥堵，但是在实际道路情况错综复杂，为了缓解拥堵的现象，构建交通流拥堵模型和应急车道启用模型很有必要。

5.2.6 交通流拥堵模型

交通拥堵指数

交通拥堵指数^[4]是综合反映道路网畅通或拥堵的概念性指数值。单一的从车流量、车速上来评判道路是否拥堵存在很大的偏差，比如对于车流量来说在拥堵或不拥堵的时候根据观测点的不同可能会得到相同的数据。所以为了很好的反应道路上交通拥挤程度，引入了交通拥堵指数（TPI）这一指标。该指数对拥堵情况进行了量化处理，综合考虑了三个交通流参数。交通指数取值范围为 1~10，分为 5 个级别（严重拥堵，中度拥堵，轻度拥堵，基本畅通，畅通），具体细分如表 5-1：

表 5-1 交通拥堵指数分级表

拥堵指数	状态	行程时间	拥堵描述
1~2	畅通	T	基本无拥堵
3~4	基本畅通	1.2T~1.5T	少部分路段拥堵
5~6	轻度拥堵	1.5T~1.8T	部分路段拥堵
7~8	中度拥堵	1.8T~2.1T	大量路段拥堵
9~10	严重拥堵	>2.1T	大部分道路拥堵

从上述分级表中可以看出行程时间的延误多少是最直观反映拥堵指数大小。当行程时间越大，拥堵指数也越大。

构建拥堵评价体系

可以从三个参数分别来考虑，比如某点处车速的骤降，密度的骤增，同时是否到达设定的阈值来综合判断该点处是否会发生拥堵。实际该处的拥堵情况仍然需要交通拥堵指数来衡量，以下给出指数的计算公式：

$$\gamma^p(t) = \frac{occ^p(t)^2 \times 1000}{Q^p(t) \times v^p(t) + 1e^{-5}} \quad (5-1)$$

上述公式中 $\gamma^p(t)$ 为在 t 时刻地点 p 处的交通拥堵指数； $occ^p(t)$ 为在 t 时刻地点 p 处的道路占有率； $Q^p(t)$ 为在 t 时刻地点 p 处的车流量； $v^p(t)$ 为在 t 时刻地点 p 处的车速。

拥堵判定流程

设定阈值后就可以通过某时刻的拥堵指数来判断该时刻所在地的道路拥堵情况，但是由于某时刻的拥堵指数无法来完整准确的判断处围绕该时刻时段的道路情况，所以本文借助保形预测的思路，将单时刻点判断拥堵情况变为一种区间评判做法。先设定某时刻为 t_0 ，记录下该时刻前 L_1 时间窗口中的拥堵指数以及变化趋势，并且记录下该时刻后 L_2 时间窗口中的拥堵指数以及变化趋势。结合时间序列数据的特性，不应该过度依赖后时间窗口中的数据进行分析，所以让 L_1 远大于 L_2 ，最后得到完整的时间区间。在计算得到的交通拥堵指数以及区间拥堵指数数据后，通过评价规则，判断该点是否拥堵。

阈值的确定方法

1. 计算交通拥堵指数 TPI

使用评价体系中的计算公式结合三个交通流参数最终得到每个观测点在 t 时刻的交通拥堵指数 TPI。由于题目实际距离并未给出，因此计算出的交通拥堵指数值波动较大，所以在对指数进行标准化处理后在大致对比视频后可以明确确定交通拥堵指数大于 20 一定

拥堵，因此先将大于 20 的 TPI 更新为 20，并且将值域空间从 0-20 映射到 0-10。

2. 分析交通拥堵指数 TPI

从公式可以看出，最终影响交通拥堵指数的一共有三个因素，分别为车流量 Q、车速 V 与车辆密度 K。确定是否拥堵阈值可以从这三个因素入手：拥堵时，有两个变量的趋势是可以明确的，分别是车速降低和车辆密度增加，而车流量较难确定，因为无论是拥堵或者畅通时，车流量都有可能变得很大或很小。现在对他们之间的关系进行分析：

将车速、车辆密度、车流量分别绘制成三维图像的 x,y,z 轴坐标，再以时间为单位将每个点填入，这里选取 103 站点第二个视频数据作为演示。

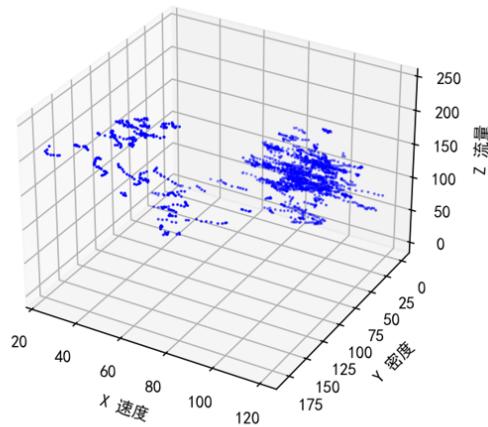


图 5-9 三维聚类结果

3. 基于 k-means 的三维聚类算法计算阈值

K-means^[5] 算法是一种在迭代式优化基础上进行聚类分析的一种方法。它将数据集合中的数据点分成 k 个类，使每一个数据点都归属于离它最近的类心，而类的中心是所有数据点的均值。因为该算法使用简单，并且具有分析两个具有相似特性的对象，因此被广泛应用

在本题上的应用

在每一个站点选取第一个视频数据进行聚类，得到如下结果：

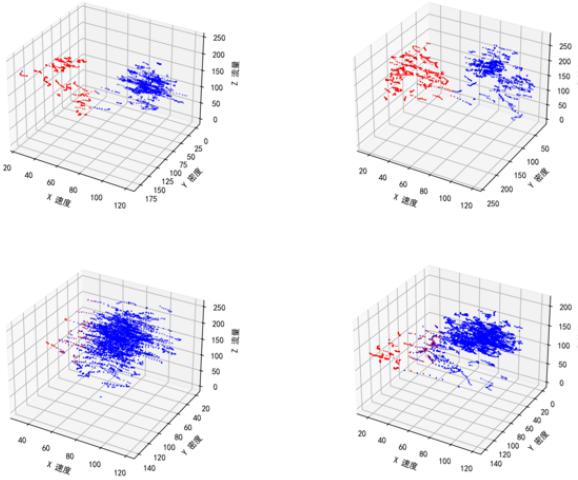


图 5-10 四个站点三个交通流参数的聚类结果

以上四张图分别是对 103-2, 105-2, 107-2 和 108-2 的视频数据进行聚类的结果，对聚类结果进行分析可以发现红色时间点非常符合拥堵的特征，即速度低、密度大，因此可以认为这部分即为拥堵部分，而蓝色部分特征为速度快，密度大多很小，符合道路畅通的特征。在对比这两类点的交通拥堵指数 TPI 后得到阈值为 8。

在实际的高速道路上存在复杂、动态的变换情况，所以为了体现流的特性并且提高评判和判定拥堵现象的准确性，在原有的评判基础上，设立了新的评判规则。

基于区间规则的拥堵判定模型

考虑实际的视频获取和处理存在不可避免的误差，所以在展示区间评定模型的时候采用理论的拥堵评价体系（1~10 指数体系）来表述模型。即设定阈值为 10，当拥堵指数超过 10 后，发生严重拥堵现象。为了规范评判体系用符号 TPI_{full} 来表示阈值

拥堵评判规则

为了方便叙述以及清晰表述，设 p 为观测地点， t 为需要判定的时刻点， tp 为过去的时刻点， tn 为未来的时刻点。其中 tp 点为 t 点向左滑动 L_1 长度窗口所得到， tn 点为 t 点向右滑动 L_2 长度窗口所得到，同时保证三个时刻点的数据是在同一个地点观测得到。 TPI_{tp} 为在 tp 时刻的拥堵指数， TPI_t 为在 t 时刻的拥堵指数， TPI_{tn} 为在 tn 时刻的拥堵指数，具体的评判规则如下：

1. 当 TPI_t 大于 TPI_{full} ，则判定为该时刻点在 p 处发生严重拥堵。
2. 当 TPI_t 小于 TPI_{full} ， TPI_{tp} 大于 TPI_{full} ，从时刻 tp 到时刻 t 总体呈现下降趋势，但是从时刻 t 到时刻 tn 总体呈现上述趋势，并满足 TPI_{tn} 超过 TPI_{full} 或者满足以下不等式也可以认为是严重拥堵。

$$(TPI_{tn} - TPI_t) / TPI_{tn} > (TPI_{tp} - TPI_t) / TPI_{tp} \quad (5-2)$$

3. $TPI_{tp}, TPI_t, TPI_{tn}$ 都小于 TPI_{full} , 但是整个区间满足以下不等式也认为是严重拥堵。其中 TPI_i 表示为后一个时刻的拥堵指数, TPI_{i-1} 表示为前一个时刻的拥堵指数。

$$\begin{cases} TPI_i - TPI_{i-1} > 0 \quad (i \in (tp, tn]) \\ TPI_{tp} + TPI_{tn} > 1.5TPI_{full} \end{cases} \quad (5-3)$$

进一步通过拥堵指数随时间变换趋势图来展示评判规则。

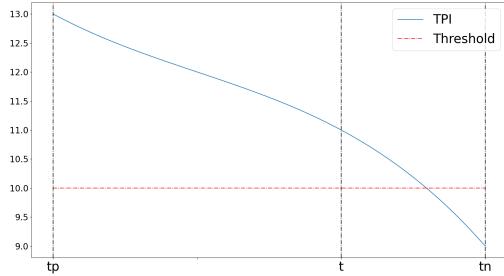


图 5-11 拥堵评判模式一

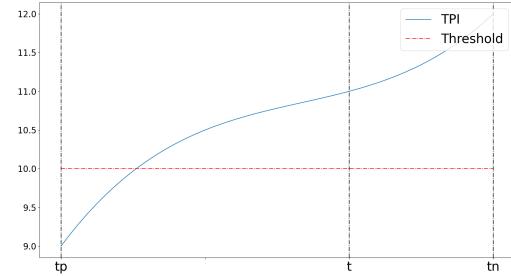


图 5-12 拥堵评判模式二

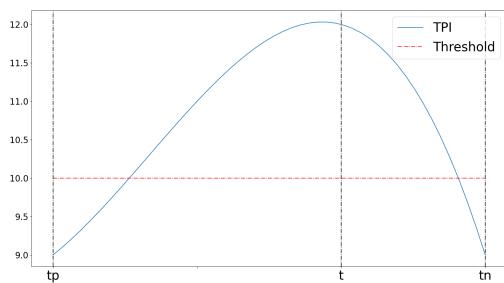


图 5-13 拥堵评判模式三

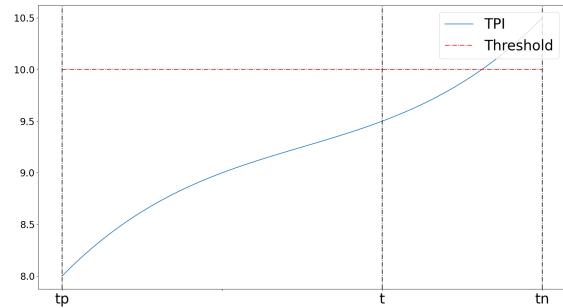


图 5-14 拥堵评判模式四

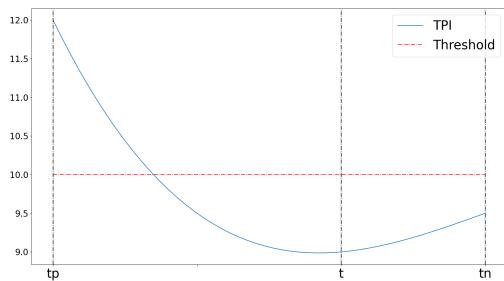


图 5-15 拥堵评判模式五

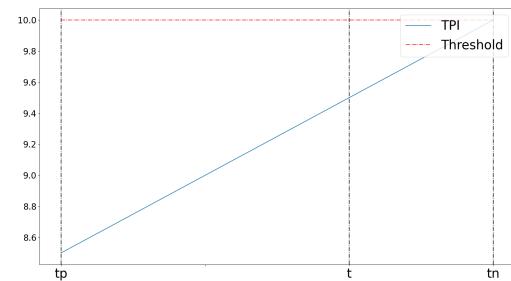


图 5-16 拥堵评判模式六

三个规则由上述六个模式所展示, 可以看到, 只要在时刻 t 处的拥堵指数超过阈值就一定评判为拥堵, 其他如若符合规则也满足条件, 也可以评判为拥堵。

综合理论的评判方法和区间评判方法可以得到交通流拥堵模型具体的工作流程图 5-17:

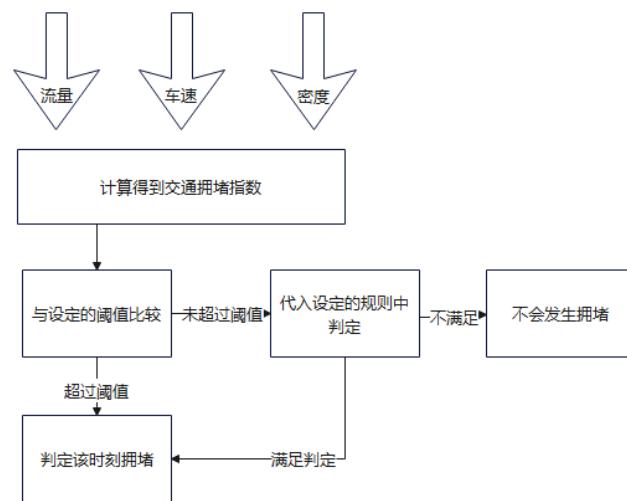


图 5-17 拥堵判定流程图

在流程图中本文仍然沿用阈值判定是否拥堵的方法，并且设立了新的规则来重新评估不满足达到阈值，却仍然很大可能发生拥堵的情况。

若 108 观测点出现交通拥堵系数大于阈值或即将大于阈值，且前两个观测点都处于交通拥堵系数大于阈值中，则认为第三点到第四点之间路段即将出现持续拥堵，具体如图所示。

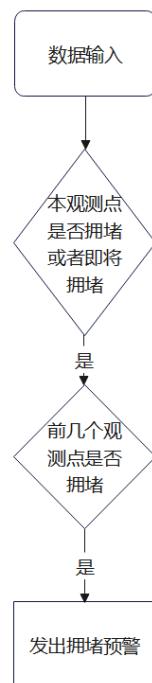


图 5-18 流程图

拥堵预警

通过区间评判的方法不仅可以用来估计某时刻在某点处有无发生拥堵，并且也可以用来预测该点处未来某时间段内拥堵指数的变化，并且所要预测的未来窗口大小与过去窗口大小有一定的关系，沿用上述评判拥堵的模式五和模式六，为了清晰表述，再次展示两模式中拥堵指数随时间变化的情况，具体如图（5-19, 5-20）所示：

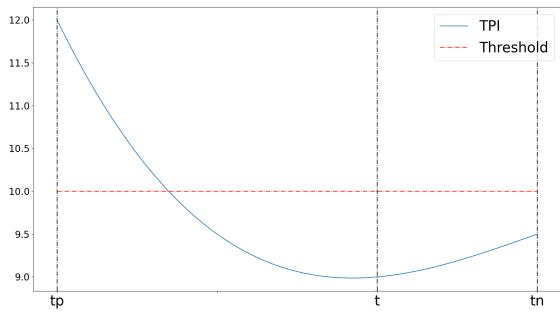


图 5-19 拥堵评判模式五

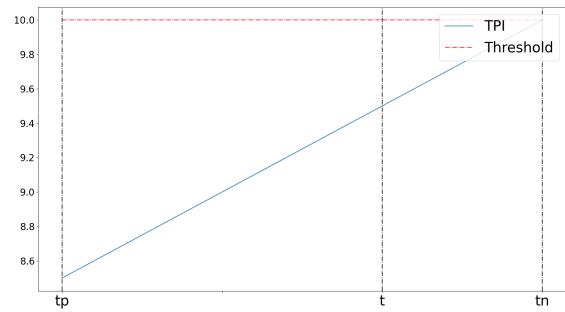


图 5-20 拥堵评判模式六

上图中时刻点 t 表示需要预警的时刻点， tn 表示发生拥堵的时刻点，在评判规则中认为时刻点 t 已经发生拥堵，但是在此无法评判为拥堵，具体原因归结为时间窗口大小的选择。在评判拥堵的时候，无论是过去窗口的大小还是未来窗口的大小都选的很小，几乎是在时刻点 t 的领域附近波动。在预警的时候需要较大的时间跨度，主要是发出预警后需要有足够的措施来应对，比如当发出预警的时候需要开放应急车道，那么就需要道路管理人员检查应急车道是否畅通，是否有专用车当前需要占有该车道等情况的处理。

量化具体预警时刻点

本文考虑交通拥堵指数的变化率来判断时刻点 t 在未来一段时间内是否会发生拥堵，具体操作流程为如下步骤：

1. 计算处在时刻点 t 处具体的拥堵指数记为 TPI_t ，选取长度为 L 的过去时间窗口（该时间窗口的大小依赖与所要预警的时间跨度），并且记下过去时刻点为 tp ，取时刻 t 和时刻 tp 的中间时刻记为 t_{mid} ，求出在时刻点 tp 到时刻点 t_{mid} 的 TPI 指数的平均变化率为 v_{tp} ，计算出时刻点 t_{mid} 到时刻点 t 处 TPI 指数的变平均化率为 v_{mid} 。

2. 记时刻点 tp 处的拥堵指数为 TPI_{tp} ，时刻点 t 处的拥堵指数为 TPI_t ，所要预警的时刻点 tn 处的拥堵指数为 TPI_{tn} 。得到初步的预警公式：

$$TPI_{tn} = 2TPI_t - TPI_{tp} \quad (5-4)$$

上述公式简单粗略的计算处未来时刻点的 TPI 指数，但是并未考虑到在过去时刻窗口中 TPI 变化所带来的影响，所以加上变化率后的公式为：

$$TPI_{tn} = TPI_t + (\alpha v_{tp}^- + \beta v_{mid}^-) * L \quad (\alpha + \beta = 1) \quad (5-5)$$

上述公式中结合了前半段时间窗口的平均变化率和后半段时间窗口的平均变化率，并且在实际运用过程中保证 $\beta > \alpha$ 。以上时刻点的分布以及窗口的选取如图 5-21 所示：

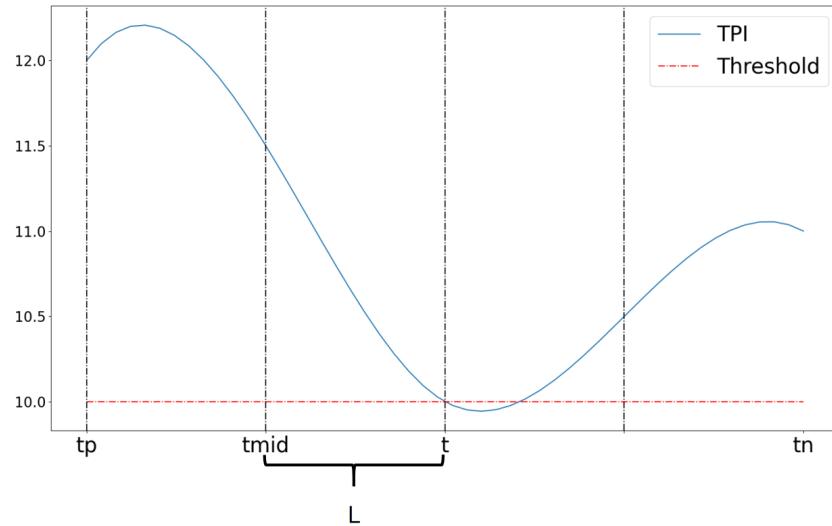


图 5-21 时间窗口预警图

在上述图为了我选取时间窗口的长度为 L ，若是要预警 10 分钟后的情况，可以选取长度为 10 分钟，或者较大于 10 分钟长度的窗口即可。

简单提高预测准确性的方法

在实际运用过程中为了节省成本，本文可以结合上述的预警公式，同时根据时间窗口 L 内行驶到应急车道上的数量与总车量的比例关系，用定量的计算公式给出预估时刻点的拥堵指数，并且再结合定性的方式（可以通过观测时间窗口内车辆行驶到应急车道上的频率）来给出最终的结果。

5.2.7 预警结果

使用交通拥堵实时预警模型对视频数据进行预测，可以发现从 11:35 到 12:50 左右 108 及前两个观测点交通拥堵系数长时间处于 8 以上，根据交通拥堵实时预警模型可以推断出从 11:35 到 12:50 这个时间段及以后 103 观测点也是拥堵的，而视频数据也能证实这一点。预警结果的准确性证明了模型的有效性。

5.3 问题一第三问

阈值验证

现对上文计算得到的拥堵阈值为 8 进行验证：在对剩下的视频数据画三维图时，以交通拥堵指数 $TPI=8$ 为分界点将点分为红蓝两色。再使用 k-means 三维聚类算法将点分类，可以看出以 $TPI=8$ 为分界点画出的三维图与使用 k-means 三维聚类算法分类的三维图基本重合，这也验证了阈值的正确性。

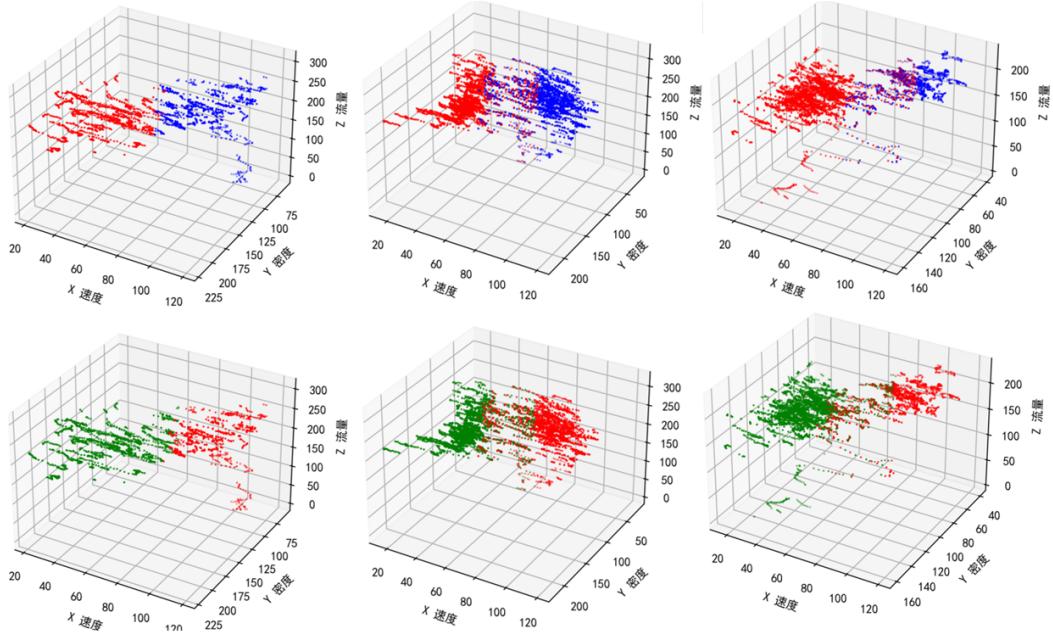


图 5-22 阈值分类与 Kmeans 聚类对比图

通过在一个观测点得到的阈值，在其他观测点用同样的阈值进行划分，发现符合实际的情况，并且通过观测视频数据发现，确实在对应的时间段发生一定程度上的拥堵。

5.4 问题二的分析与求解

5.4.1 问题分析

为了构建合理启用应急车道模型，需要借助问题一中的交通流拥堵模型来完成，实际上交通流拥堵模型中用了顶点判定和区间判定的方法，并且将区间判定的方法拓展到了预警未来可能发生拥堵的时刻，所以本题就根据问题一构建的拥堵模型中的评判规则来为决策者提供是否开启应急车道的理论依据。

5.4.2 高速公路应急车道模型

为了让决策者更加直观的对路面情况进行决策，本文中让判定规则中的决策因素从拥堵系数变为了车速。这样就可以从车速的变化幅度来判定是否需要开启应急车道。

拥堵概率估计

本文通过分析每个时间段的车速与密度和交通状况来估计可能发生拥堵的概率，并且以此来确定下一个时段会不会发生拥堵的概率，借助以往的拥堵概率评估模型^[1] 分析如下：

$$F_c(q) = P(c \leq q) = 1 - \prod_{i:q_i \leq q} \frac{k_i - d_i}{k_i}, i \in B \quad (5-6)$$

上述公式中 q 表示为一小时内单个车道的车辆通过的数量， c 表示是道路的容量， P 表示为容量小于观测到流量的概率， q_i 是在间隔为 i 处观察到的车流量，表示为速度下降

之前的交通流量; k_i 指 $q \geq q_i$ 的时间间隔长度; d_i 是指交通流量达到 q_i 时的时间间隔数, B 是拥堵间隔 $\{B_1, B_2, B_3, \dots\}$ 的集合。

从该公式中可以发现利用该公式来评估拥堵概率需要较多的参数, 并且该模型对数据的灵敏度及其高, 需要对数据的采集和统计做到非常小的误差, 才能使用该公式, 所以为了解模型的灵敏度, 在本题对拥堵概率的评估采用公式:

$$P(v = v_0) = \frac{cnt_{v_0}}{cnt_{v_0}} \quad (5-7)$$

上述公式中 $P(v = v_0)$ 表示为车速在 v_0 出可能发生的概率, cnt_{v_0} 表示车速为 v_0 发生拥堵的车的数量, cnt_{v_0} 为车速为 v_0 总的数量。(在统计的时候为了减少数据量少可能带来的数据长尾效应统计四个观测点的所有数据来减少这种效应) 拥堵概率与车速的关系结果如图 (5-23) 所示:

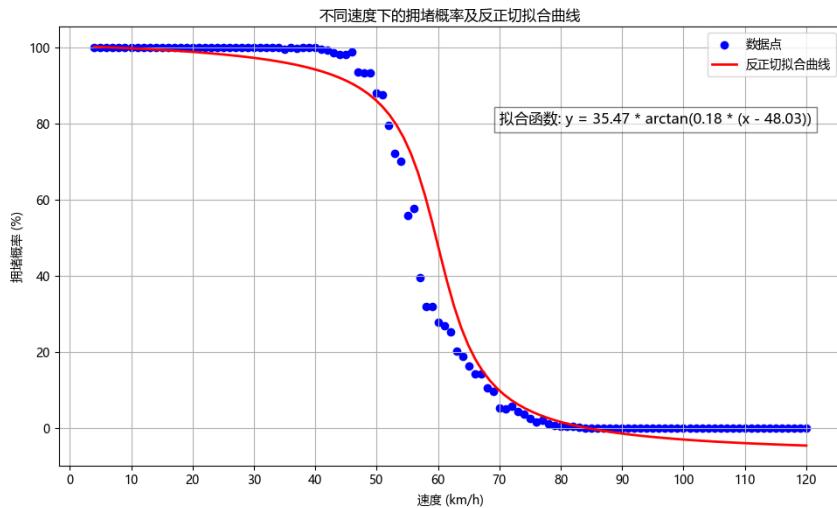


图 5-23 速度与拥堵概率的关系

上述中描述了速度与拥堵发生概率的关系, 可以看出符合实际情况, 当速度较大时接近自由流速度时, 几乎不发生拥堵, 当速度变小时发生拥堵的概率变大。选取速度为 60 千米/时可以用来对应预警的拥堵指数。

图 (5-24) 表示了密度与拥堵概率的关系

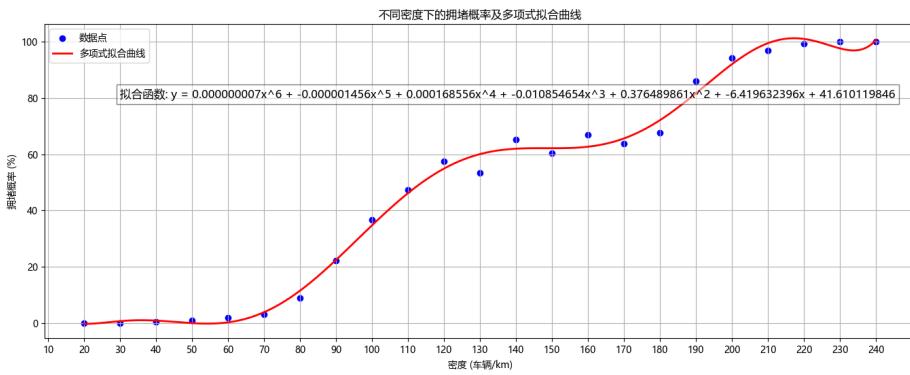


图 5-24 密度与拥堵概率的关系

从上述图中可以发现，当密度较大时，发生拥堵的概率大，当密度较小时，发生拥堵的概率小。

5.4.3 问题总结

本文通过分析车速与拥堵发生概率的关系后得到当大部分车辆的速度低于 60 千米/时的时候可以告诉决策者可以启用应急车道了，但是考虑在实际过程中道路上会有一些紧急情况的发生，所以需要当速度低于 65 千米/时的时候决策者需要先得到道路上应急车道的占有情况再当车速下降到 60 千米/时做出决策。

5.5 问题三的分析与求解

5.5.1 问题分析

为了构建合理的规则来决策是否启用应急名单，在利用拥堵评定规则的时候，需要对未来数据进行预测，考虑到数据处理的误差，道路情况的复杂，所以采用大预言预测模型-LSTPrompt 模型^[7] 来进行序列的预测，将预测出现拥堵情况时，开启应急道路，利用元胞自动机仿真，得到直观的路面变化情况。

5.5.2 LSTPrompt 模型

方法原理

LSTPrompt 的核心思想是将时间序列预测任务分解为短期和长期预测子任务，并为每个任务定制提示（prompt）。通过这种方式，LLMs 可以在没有额外训练数据的情况下，直接对时间序列进行预测。

5.5.3 构建预测模型

LSTPrompt 构建预测模型的步骤如下：

转换数据格式

将时间序列数据转换为适合 LLMs 处理的格式，如文本形式。在本题中需要把拥堵指数的时间序列数据转换为文本数据，以下以表格形式展示随机 3 个时间点数据，以及转换后的文本数据。

表 5-2 时间序列数据转换为文本数据过程

时间点	拥堵指标
13: 15	20
13: 30	30
14: 00	30
文本数据	
” 20, 30, 30 “	

分解长短期任务

将时间序列预测任务分解为短期和长期预测子任务。短期预测关注于近期的数据变化，而长期预测则关注于整体趋势。在本题中要在 10 分钟之前报警，那么我就需要预测至少 10 分钟之后的交通情况，以分钟为单位的预测窗口为短期预测，因为短期预测强调趋势变化和动态模式，而长期预测强调统计特性和周期性模式，模型采用 TimeDeComp 将划分短期和长期。该次模型预测主要使用短期模式来完成预测工作。

提示词定制

为每个子任务定制特定的提示，以引导 LLMs 进行预测。这些提示可能包含对时间序列的描述、历史数据的概述以及预测要求等。

为了让大语言模型对世界序列预测更加准确，需要通过一些提示词进行引导，在本题中我输入的提示词有高峰期、车流量和车速等信息。在提示词之前可以输入一些领域知识让模型进行自学习，可以小幅度提高预测精度。

模型的工作流程展示如图（5-25）：

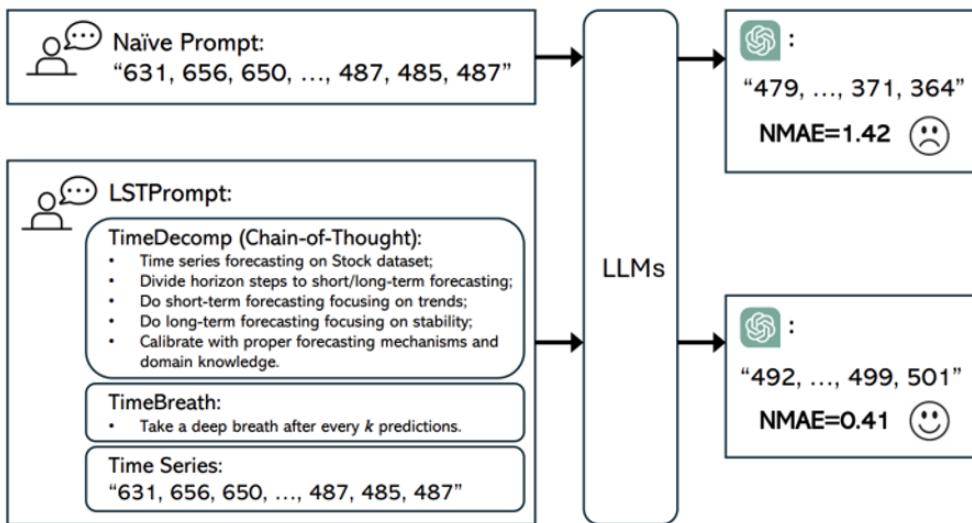


图 5-25 LSTPromt 预测流程^[7]

上述流程图中可以看到通过 TimeBreak 方式将未来需要预测的数据定为 k 步骤，在本题为了方便检验模型本文选择后 10 分钟的预测结果。

每个站点根据现有数据，对未来的数据进行预测，下图是根据部分数据进行预测的结果图。

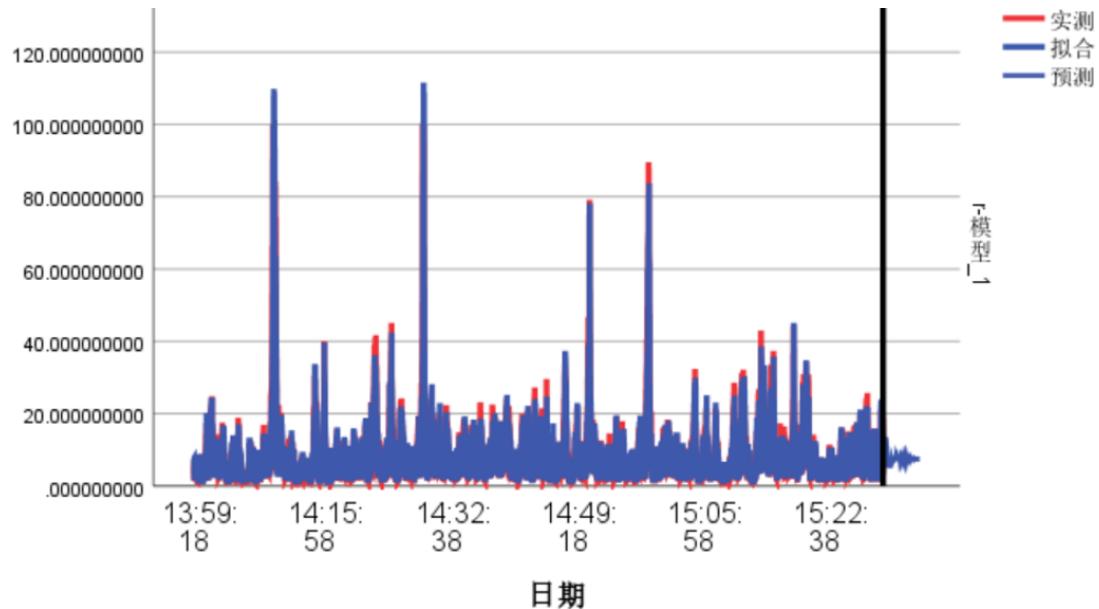


图 5-26 LSTPromt 预测流程

上述图中的纵坐标表示为拥堵指数，为了更加贴切真实数据，所以没有对拥堵指数进行映射。

此时该站点就可以根据预测出来的数据实时进行决定是否启用应急车道，需要注意的是，由于使用的数据具有时间跨度短，规律性低的因素，预测未来数据不能太长，否则会变得不准确。

应急车道作用量化

当开启应急车道时，道路由原来的两车道变成三车道，车辆密度变为原来的 $2/3$ ，从而导致道路负荷降低，车辆速度快等，最终减少了车辆的拥堵程度。最后，为了直观看到应急车道启用后的效果，也进行了仿真，结果展示如图：

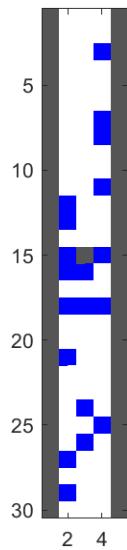


图 5-27 非高峰期车流情况（开启应急车道）

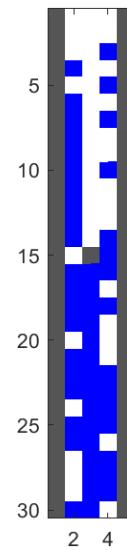


图 5-28 高高峰期车流情况（开启应急车道）

从以上仿真结果可以发现，当开启应急车道后，无论是高峰期还是非高峰期，对道路拥堵现象都会有所缓解。

5.6 问题四的分析与求解

讨论如何布置视频监控点之前，本文先对第三个点到第四个点之间路段进行分析。下图是两个观测点之间的道路模拟图，根据视频信息可以获得两个基本事实：当道路拥堵时，部分司机会自发开上应急车道。由于 108 观测点前方就是服务区，并且大部分司机并不想去服务区，因此当道路拥堵时，103 观测点三条车道都有车开，而 103 观测点则只有两条车道有车。

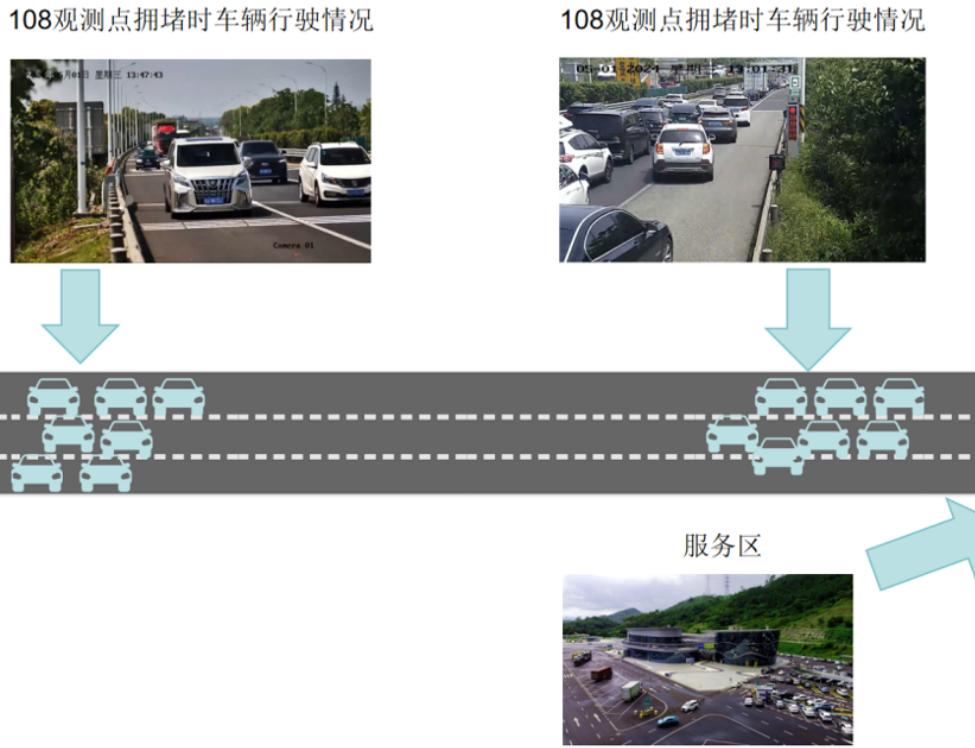


图 5-29 道路实际情况展示图

从以上两点不难发现，该段到路是一段“宽进严出”道路，具体表现为：面对同样足以导致拥堵的车流量，车辆在通过 108 观测点的时间一定比通过 103 观测点时间要短。这说明 103 观测点的车辆负荷能力远小于它前面的道路。

为了提升第三个观测点到第四个观测点之间路段应急车道临时启用决策的科学性，同时控制成本，本文提出以下观测点布置方案：在道路的前四分之一路段和最后四分之一路段布置观测点。

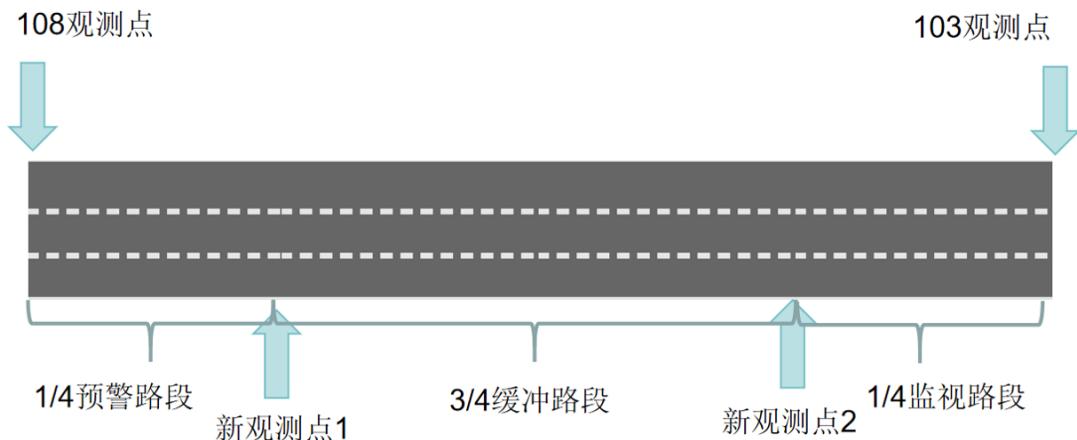


图 5-30 道路简单建模展示图

上文提到，103 观测点的车辆负荷能力低于它前面的道路，因此需要对该路段的车流更加“敏感”。新观测点 1 的布置是为了能更早地得到进入这段道路的车流状况并提早做出判断，当预警路段没有发生拥堵但是交通拥堵指数 TPI 大于 15 且趋势上升时，这表明道路上即将接近拥堵，新观测点就要做出预警，开启应急车道让这部分车加快通过，起到了分散车流的作用，减少了这部分车流到达 103 观测点时的负荷。对于新观测点 2，它的作用是为了观测监视路段的车流状况，当 103 观测点负荷过大时，拥堵状况的发生一定是从 103 观测点蔓延向新观测点 2 的，因此新观测点 2 的布置靠近 103 观测点的目的是为了尽早发现 103 的负荷程度，若拥堵从 103 观测点蔓延到新观测点 2，这表明 103 观测点的车流负荷过大，因此需要提前关闭应急车道使得减缓车流进入路段的速度。

6 模型评价

对本文启用应急车道模型进行灵敏度分析，在其他参数不变的情况下，对速度以及密度进行改变可以发现其对结果的影响较大，而其余参数变化对结果的影响不大，这表明本文的模型较为稳定。

参考文献

- [1] Jocher, G., Chaurasia, A., Qiu, J. (2023). Ultralytics YOLO (Version 8.0.0) [Computer software]. <https://github.com/ultralytics/ultralytics>
- [2] AHARON N, ORFAIG R, BOBROVSKY B Z. BoT-SORT: Robust Associations Multi-Pedestrian Tracking[J]. 2022.
- [3] 黄晨, 屠媛, 王予, 等. 融合多源信息的元胞自动机交通流模型 [J]. 江苏大学学报(自然科学版), 2023, 44(06):680-686.
- [4] 杨阳, 刘强, 石英杰. 高速公路饱和路段动态应急车道开放决策模型研究 [J]. 公路工程, 2022, 47(3) : 172 – 176.
- [5] 陈昊烜, 周侗, 王辰怡, 等. 面向热区发现的空间聚类算法适用性研究 [J]. 物流工程与管理, 2021, 43(10):34-36+30.
- [6] Xu, C., Xie, Y. (2022). Sequential Predictive Conformal Inference for Time Series. International Conference on Machine Learning.
- [7] LSTPrompt: Large Language Models as Zero-Shot Time Series Forecasters by Long-Short-Term Prompting. In Findings of the Association for Computational Linguistics ACL 2024, pages 7832–7840, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

附录 A MATLAB 源程序

A.1 元胞自动机仿真

Cellular.m

```
clc;
close all;
%=====GUI部分=====
plotbutton=uicontrol('style','pushbutton',...
    'string','Run', ...
    'fontsize',12, ...
    'position',[100,400,50,20], ...
    'callback','run=1;');
erasebutton=uicontrol('style','pushbutton',...
    'string','Stop', ...
    'fontsize',12, ...
    'position',[200,400,50,20], ...
    'callback','freeze=1;');
quitbutton=uicontrol('style','pushbutton',...
    'string','Quit', ...
    'fontsize',12, ...
    'position',[300,400,50,20], ...
    'callback','stop=1;close;');
number = uicontrol('style','text', ...
    'string','1', ...
    'fontsize',12, ...
    'position',[20,400,50,20]);
%=====初始化车道量=====
B=3;
roadlength=30;
merge =15;
barrier = 2;
scale = 0;
switch_ratio = 0.8;
```

```

road=zeros(roadlength,B+2);
v=zeros(roadlength,B+2);

%road(merge+1:roadlength,2) = -1;
% 设定障碍物（仿真交通事故）
road(merge + 1:merge + 1,3) = -1;
road(1:roadlength,[1,B+2]) = -1;

h = imagesc(road);
set(h, 'erasemode', 'none')
axis equal
axis tight
vmax = zeros(50,6);
probC=1;
probV=[0.5,0.5];
probSlow=-1;
VTypes=[1,1];
[road,v,vmax]=new_cars(road,v,probC,probV,VTypes,vmax,scale);
ROAD=rot90(road,2);
h=show_road(ROAD,h,0.1);
%=====进行迭代=====
global all_num;
all_num = 0;
all_note = zeros(11,1);
run = 1;
stop = 0;
freeze = 0;
probC = 0.2;
for i = 1:5
    while(stop<1000)
        if (run==1)
            [v,LUP,LDOWN]=road_count(road,v);
            [road,v,vmax]=switch_lane(road,v,vmax,LUP,LDOWN,barrier,
            switch_ratio);
            [road,v,vmax]=move_forward(road,v,vmax,probSlow,merge);
            [road,v,vmax]=new_cars(road,v,probC,probV,VTypes,vmax,scale);
            ROAD=rot90(road,2);
            h=show_road(ROAD,h,0.1);
        end
        run = run + 1;
        stop = stop + 1;
    end
end

```

```

    end
    if (freeze==1)
        run = 0;
        freeze = 0;
    end
    drawnow
    stop = stop+1;
end
stop = 0;
road=zeros(roadlength,B+2);
probc = probc + 0.1;
v=zeros(roadlength,B+2);
% scale = scale + 0.1;
road(merge:roadlength,2) = -1;
road(1:roadlength,[1,B+2])=-1;
all_note(i,1) = all_num;
all_num = 0;
end

```

moveForward.m

```

function [road,v,vmax]=moveForward(road,v,vmax,probslow,merge)
global all_num;
[L,W]=size(road);%车道大小, 包括边界
for lanes=2:4;
    temp=find(road(:,lanes)>=1);
    nn=length(temp);
    for k=1:nn
        i=temp(nn - k + 1);
        if(lanes == 2)
            merge_new = L - 2
            front = find(road(i+1:merge_new,lanes)> 0, 1) + i;
            if (i == 8)
                display(front + i)
            end
            % 调试
            if(i+v(i,lanes)>=merge_new ||
               (~isempty(front)&(i+v(i,lanes)>=front)))
                if(i+v(i,lanes)>=merge_new)

```

```

if(merge_new-i>1)
    %如果离边界大于1格
    v(i+1,lanes) = 1;
    road(i+1,lanes) = road(i,lanes);
    road(i,lanes) = 0;
    v(i,lanes) = 0;
    vmax(i+1,lanes) = vmax(i,lanes);
    vmax(i,lanes) = 0;
elseif(merge_new==1)
    %如果离边界等于1格
    v(i+1,lanes) = 0;
    road(i+1,lanes) = road(i,lanes);
    road(i,lanes) = 0;
    v(i,lanes) = 0;
    vmax(i+v(front,lanes),lanes) = vmax(i,lanes);
    vmax(i,lanes) = 0;
elseif(merge_new==i)
    v(i,lanes) = 0;
end
elseif(v(front,lanes) ~= 0 & road(front, lanes) ~= 1)
    %小于安全距离进行减速, 减速为前车速度
    %如果前面有车, 且速度不为0
    num = min(i+v(front,lanes),max(front-1,1));
    road(num,lanes) = road(i,lanes);
    road(i,lanes) = 0;
    v(i,lanes) = 0;
    v(num,1) = v(front,lanes);
    vmax(num,lanes) = vmax(i,lanes);
    vmax(i,lanes) = 0;
else
    %如果前面的车速度为0, 间隔大于1, 则速度变为1, 否则速度变为0
    if(front - i > 1)
        v(i+1,lanes) = 1;
        road(i+1,lanes) = road(i,lanes);
        road(i,lanes) = 0;
        v(i,lanes) = 0;
        vmax(i+1,lanes) = vmax(i,lanes);
        vmax(i,lanes) = 0;
    elseif(front-i==1)

```

```

    v(i,lanes) = 0;

    end
end
elseif(road(i,lanes)==1&&rand<probslow)
a_v = v(i,lanes);
num = min(i+v(front,lanes),max(front-1,1));
v(i+a_v,lanes) = max(v(i,lanes)-1,1); %随机慢化
v(i,lanes) = 0;
road(i+a_v,lanes) = road(i,lanes);
road(i,lanes) = 0;
vmax(i+a_v,lanes) = vmax(i,lanes);
vmax(i,lanes) = 0;
elseif(isempty(front) | front-i>= v(i,lanes)+1)
v(i+1,lanes) = 1; %加速
v(i,lanes) = 0;
road(i+1,lanes) = 1;
road(i,lanes) = 0;
vmax(i+1,lanes) = 1;
vmax(i,lanes) = 0;

elseif(front-i <=v(i,lanes))

a_v = v(i,lanes);
v(i+a_v,lanes) = 1 %减速
v(i,lanes) = 0;
road(i+a_v,lanes) = road(i,lanes);
road(i,lanes) = 0;
vmax(i+a_v,lanes) = vmax(i,lanes);
vmax(i,lanes) = 0;
else

v(i+1,lanes) = 1; %加速
v(i,lanes) = 0;
road(i+1,lanes) = 1;
road(i,lanes) = 0;
vmax(i+1,lanes) = 1;
vmax(i,lanes) = 0;
end

```

```

elseif(lanes==3)
    front = find(road(i+1:end,lanes)>0,1)+i
    if(i+v(i,lanes)>= merge)
        if(merge-i>1)
            %如果离边界大于1格
            v(i+1,lanes) = 1;
            road(i+1,lanes) = road(i,lanes);
            road(i,lanes) = 0;
            v(i,lanes) = 0;
            vmax(i+1,lanes) = vmax(i,lanes);
            vmax(i,lanes) = 0;
        elseif(merge-i==1)
            %如果离边界等于1格
            v(i+1,lanes) = 0;
            road(i+1,lanes) = road(i,lanes);
            road(i,lanes) = 0;
            v(i,lanes) = 0;
            vmax(i+v(front,lanes),lanes) = vmax(i,lanes);
            vmax(i,lanes) = 0;
        elseif(merge==i)
            v(i,lanes) = 0;
    end
else
    front = find(road(i+1:end,lanes)>0,1)+i;
    a = find(road(:,lanes)~= 0);
    c = road(a,lanes);
    if(isempty(front)&i+v(i,lanes)>=L)
        v(i,lanes) = 0;
        vmax(i,lanes) = 0;
        road(i,lanes) = 0;
        all_num = all_num + 1;
    elseif(front-i==1)
        v(i,lanes) = min(v(front,lanes),vmax(i,lanes));
    elseif((isempty(front) | front-1>=v(i,lanes)+i) &
        road(i,lanes)==1 & rand<probslow)
        a_v = v(i,lanes);
        if(road(i+a_v,lanes)==0)
            v(i+a_v,lanes) = max(v(i,lanes)-1,1); %随机慢化
    end
end

```

```

v(i,lanes) = 0;
road(i+a_v,lanes) = road(i,lanes);
road(i,lanes) = 0;
vmax(i+a_v,lanes) = vmax(i,lanes);
vmax(i,lanes) = 0;
elseif(road(i+a_v,lanes)~=0)
    v(front-1,lanes) = 1;           %随机慢化
    v(i,lanes) = 0;
    road(front-1,lanes) = road(i,lanes);
    road(i,lanes) = 0;
    vmax(front,lanes) = vmax(i,lanes);
    vmax(i,lanes) = 0;
end
elseif(isempty(front) | front-1>=v(i,lanes)+i&
    front > i+1)      %大于才能进行加速
    a_v = v(i,lanes);
    if(road(i+a_v,lanes)==0)
        v(i+a_v,lanes) =
            min(v(i,lanes)+1,vmax(i,lanes)); %加速
        v(i,lanes) = 0;
        road(i+a_v,lanes) = road(i,lanes);
        road(i,lanes) = 0;
        vmax(i+a_v,lanes) = vmax(i,lanes);
        vmax(i,lanes) = 0;
    elseif(a_v~=0)
        v(front-1,lanes) =
            min(v(i,lanes)+1,vmax(i,lanes)); %加速
        v(i,lanes) = 0;
        road(front,lanes) = road(i,lanes);
        road(i,lanes) = 0;
        vmax(front,lanes) = vmax(i,lanes);
        vmax(i,lanes) = 0;
    else
        v(i,lanes) = 1;
    end
elseif(~isempty(front) & i+v(i,lanes)> front &
    front > i+1)
    num = min(i+v(front,lanes),max(front-1,1));
    if(num~=i)

```

```

        road(num,lanes) = road(i,lanes);
        road(i,lanes) = 0;
        v(num,1) = v(i,lanes);
        v(i,lanes) = 0;
        vmax(num,lanes) = vmax(i,lanes);
        vmax(i,lanes) = 0;
    end
end
b = find(road(:,lanes)~=0);
end

elseif(lanes==4)
    front = find(road(i+1:end,lanes)>0,1)+i;
    a = find(road(:,lanes)~= 0);
    c = road(a,lanes);
    if(isempty(front)&i+v(i,lanes)>=L)
        v(i,lanes) = 0;
        vmax(i,lanes) = 0;
        road(i,lanes) = 0;
        all_num = all_num + 1;
    elseif(front-i==1)
        v(i,lanes) = min(v(front,lanes),vmax(i,lanes));
    elseif((isempty(front) | front-1>=v(i,lanes)+i) &
        road(i,lanes)==1 & rand<probslow)
        a_v = v(i,lanes);
        if(road(i+a_v,lanes)==0)
            v(i+a_v,lanes) = max(v(i,lanes)-1,1); %随机慢化
            v(i,lanes) = 0;
            road(i+a_v,lanes) = road(i,lanes);
            road(i,lanes) = 0;
            vmax(i+a_v,lanes) = vmax(i,lanes);
            vmax(i,lanes) = 0;
        elseif(road(i+a_v,lanes)~=0)
            v(front-1,lanes) = 1; %随机慢化
            v(i,lanes) = 0;
            road(front-1,lanes) = road(i,lanes);
            road(i,lanes) = 0;
            vmax(front,lanes) = vmax(i,lanes);
            vmax(i,lanes) = 0;
        end
    end
end

```

```

    end

elseif isempty(front) | front-1>=v(i,lanes)+i& front >
i+1)      %大于才能进行加速
a_v = v(i,lanes);
if (road(i+a_v,lanes)==0)
v(i+a_v,lanes) =
min(v(i,lanes)+1,vmax(i,lanes)); %加速
v(i,lanes) = 0;
road(i+a_v,lanes) = road(i,lanes);
road(i,lanes) = 0;
vmax(i+a_v,lanes) = vmax(i,lanes);
vmax(i,lanes) = 0;

elseif (a_v~=0)
v(front-1,lanes) =
min(v(i,lanes)+1,vmax(i,lanes)); %加速
v(i,lanes) = 0;
road(front,lanes) = road(i,lanes);
road(i,lanes) = 0;
vmax(front,lanes) = vmax(i,lanes);
vmax(i,lanes) = 0;

else
v(i,lanes) = 1;
end

elseif (~isempty(front) & i+v(i,lanes)> front & front >
i+1)
num = min(i+v(front,lanes),max(front-1,1));
if (num~=i)
road(num,lanes) = road(i,lanes);
road(i,lanes) = 0;
v(num,1) = v(i,lanes);
v(i,lanes) = 0;
vmax(num,lanes) = vmax(i,lanes);
vmax(i,lanes) = 0;
end
end

b = find(road(:,lanes)~=0);
end
end
end

```

```
end
```

newCars.m

```
function [road,v,vmax] =
    newCars(road,v,probC,probV,VTYPES,vmax,scale)
%UNTITLED2 此处显示有关此函数的摘要
% 此处显示详细说明
[~,W]=size(road);
for lanes=2:4
    if(rand<=probC&&road(1,lanes)==0)%在该位置随机产生一个车子
        tmp=rand;
        if(rand<scale)
            road(1,lanes) = 1;
        else
            road(1,lanes) = 1;
        end
        for k=1:length(probV)%随机生成一个车子应该有的最大车速
            if(tmp<=probV(k))
                vmax(1,lanes)=1;%判断属于哪个挡的车速并赋值
                v(1,lanes)=1;%对当前位置随机赋予一个初速度
                break;
            end
        end
    end
end
end
```

roadCount.m

```
function [v,LUP,LDOWN]=roadCount(road,v)
[L,W]=size(road);%车道大小，包括边界

%step2:计算每辆车与前面一辆车的距离gap

%step3:计算每车左车道的前后车的距离是否在要求范围内
LUP=zeros(L,W);
LDOWN=zeros(L,W);
for lanes=2:3;
```

```

temp=find(road(:,lanes)>=1);
nn=length(temp);
for k=1:nn;
    i=temp(k);
    a = find(road(1:i,lanes+1)>0, 1, 'last');
    if isempty(a)
        LDOWN(i,lanes) = 1;
    elseif(a<i)
        if(v(a,lanes+1)+a <= i-1)      %后面两个是否为空
            LDOWN(i,lanes) = 1;
        elseif(road(a,lanes+1)==2)
            LDOWN(i,lanes) = 1;
        end
    end
    if(k==nn)                      %最后一辆车
        if(sum(road(i:i+v(i,lanes)+1,lanes+1))==0)
            LUP(i,lanes)=1;
        end
        continue;
    end
    a = find(road(i:i+v(i,lanes)+1,lanes+1)>0, 1)+i-1;
    if isempty(a)
        LUP(i,lanes) = 1;
    elseif(a>i)
        if(v(i,lanes)+i <= a-1)      %后面两个是否为空
            LUP(i,lanes) = 1;
        end
    end
    end
end
end

```

showRoad.m

```

function h = showRoad(road,h,n)
%UNTITLED3 此处显示有关此函数的摘要
% 此处显示详细说明
[L,W]=size(road);
temp=road;
temp(temp==1)=0;%create the palza without any cars

```

```

road_draw=road;

ROAD(:,:,1)=road_draw;
ROAD(:,:,2)=road_draw;
ROAD(:,:,3)=temp;
ROAD=1-ROAD;
ROAD(ROAD>1)=ROAD(ROAD>1)/6;

if ishandle(h)
set(h, 'CData', ROAD);
pause(n);
else
colorbar;
set(h, 'CData', ROAD);
plot([(0:W)', (0:W)']+0.5, [0,L]+0.5, 'k');
plot([0,W]+0.5, [(0:L)', (0:L)']+0.5, 'k');
axis image
set(h, 'xtick', [], 'ytick', []);
pause(n);
end
end

```

switchLane.m

```

function [road,v,vmax]=switchLane(road,v,vmax,LUP,LDOWN,barrier,
switch_ratio)
[L,W]=size(road);%车道大小，包括边界
changeL=zeros(L,W);%可向左变道
%测量是否可以向左变道
for lanes = 2:3
temp=find(road(:,lanes)>=1);
nn=length(temp);
for k=1:nn
i=temp(k);
if(i > barrier && LUP(i,lanes)==1 && LDOWN(i,lanes) == 1
&& road(i, lanes) ~= 1 && road(i, lanes) ~= -1 )
changeL(i,lanes)=1;
end

```

```

    end
end

%向左换道
for lanes=2:3
    temp=find(changeL(:,lanes)==1);
    nn=length(temp);
    for k=nn:-1:1
        i=temp(k);
        if(road(i,lanes)==2)
            if(sum(road(i:i+v(i,lanes),lanes+1))==0)
                road(i,lanes+1)=2;
                v(i,lanes+1)=max(v(i,lanes)-1,1);
                vmax(i,lanes+1)=vmax(i,lanes);
                road(i,lanes)=0;
                v(i,lanes)=0;
                vmax(i,lanes)=0;
            end
        elseif(road(i,lanes)==1&&rand<switch_ratio)
            if(sum(road(i:i+v(i,lanes),lanes+1))==0)
                road(i,lanes+1)=1;
                v(i,lanes+1)=max(v(i,lanes)-1,1);
                vmax(i,lanes+1)=vmax(i,lanes);
                road(i,lanes)=0;
                v(i,lanes)=0;
                vmax(i,lanes)=0;
            end
        end
    end
end
end
end

```

附录 B Python 源程序

B.1 视频解析

all.py

```
import pandas as pd
# 读取表格
path = "the path of your data"
all_df = pd.read_csv(path + 'all.csv', encoding='ISO-8859-1')
output1_df = pd.read_csv(path + '103output1.csv',
    encoding='ISO-8859-1')
output2_df = pd.read_csv(path + '103output2.csv',
    encoding='ISO-8859-1')
#output3_df = pd.read_csv(path + '105output3.csv',
#    encoding='ISO-8859-1')
# 创建新列
all_df['103时间'] = None
all_df['103速度'] = None
all_df['103密度'] = None
all_df['103拥堵指数'] = None

# 将output1_df的数据复制到all_df中
for idx, row in output1_df.iterrows():
    # 使用位置索引
    matching_rows = all_df[all_df.iloc[:, 0] == row.iloc[10]]
    if not matching_rows.empty:
        all_df.loc[matching_rows.index, '103时间'] = row.iloc[10]
        all_df.loc[matching_rows.index, '103速度'] = row.iloc[12]
        all_df.loc[matching_rows.index, '103密度'] = row.iloc[14]
        all_df.loc[matching_rows.index, '103拥堵指数'] =
            row.iloc[16]
# 将output2_df的数据复制到all_df中
for idx, row in output2_df.iterrows():
    # 使用位置索引
    matching_rows = all_df[all_df.iloc[:, 0] == row.iloc[10]]
    if not matching_rows.empty:
        all_df.loc[matching_rows.index, '103时间'] = row.iloc[10]
        all_df.loc[matching_rows.index, '103速度'] = row.iloc[12]
        all_df.loc[matching_rows.index, '103密度'] = row.iloc[14]
        all_df.loc[matching_rows.index, '103拥堵指数'] =
```

```

    row.iloc[16]

# # 将output2_df的数据复制到all_df中
# for idx, row in output3_df.iterrows():
#     # 使用位置索引
#     matching_rows = all_df[all_df.iloc[:, 0] == row.iloc[10]]
#     if not matching_rows.empty:
#         all_df.loc[matching_rows.index, '107时间'] = row.iloc[10]
#         all_df.loc[matching_rows.index, '107速度'] = row.iloc[12]
#         all_df.loc[matching_rows.index, '107密度'] = row.iloc[14]
#         all_df.loc[matching_rows.index, '107拥堵指数'] =
#             row.iloc[16]

# 保存修改后的数据
all_df.to_csv('103.csv', index=False)
print("数据处理完成, 已保存到 updated_all.csv")

```

all 替换 100.py

```

import pandas as pd

# 读取数据
path = "the path of your data"
all_df = pd.read_csv(path + 'all.csv', encoding='ISO-8859-1')
# 指定要处理的列
columns_to_check = [3, 6, 9, 12] # 0-based index
# 替换值
for column_index in columns_to_check:
    all_df.iloc[:, column_index] = all_df.iloc[:, column_index].apply(
        lambda x: 20 if x > 15 else
            15 if 10 < x <= 15 else
            10 if 5 < x <= 10 else
            5 if 0 <= x <= 5 else
            x
    )
# 保存修改后的数据
all_df.to_csv('updated_all.csv', index=False)
print("数值已根据条件替换, 结果保存到 updated_all.csv")

```

car.py

```

import cv2
from ultralytics import YOLO
from collections import defaultdict

model = YOLO('weights/yolov8s.pt')

cap = cv2.VideoCapture(path = "the path of your data")
fps = cap.get(cv2.CAP_PROP_FPS)
size = (int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)), int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)))
fNUMS = cap.get(cv2.CAP_PROP_FRAME_COUNT)

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
videoWriter = cv2.VideoWriter("path = "the path of your data",
                             fourcc, fps, size)

def box_label(image, box, label='', color=(128, 128, 128),
              txt_color=(255, 255, 255)):
    #得到目标矩形框的左上角和右下角坐标
    p1, p2 = (int(box[0]), int(box[1])), (int(box[2]),
                                             int(box[3]))
    #绘制矩形框
    cv2.rectangle(image, p1, p2, color, thickness=1,
                  lineType=cv2.LINE_AA)
    if label:
        #得到要书写的文本的宽和长, 用于给文本绘制背景色
        w, h = cv2.getTextSize(label, 0, fontScale=2 / 3,
                               thickness=1)[0]
        #确保显示的文本不会超出图片范围
        outside = p1[1] - h >= 3
        p2 = p1[0] + w, p1[1] - h - 3 if outside else p1[1] + h +
            3
        cv2.rectangle(image, p1, p2, color, -1, cv2.LINE_AA)
        #填充颜色
        #书写文本
        cv2.putText(image,
                    label, (p1[0], p1[1] - 2 if outside else p1[1] + h
                           + 2),
                    0,

```

```

        2 / 3,
        txt_color,
        thickness=1,
        lineType=cv2.LINE_AA)
track_history = defaultdict(lambda: [])
# 车道车辆的计数变量
car1 = 0 #1车道经过车辆数量
car2 = 0 #2车道经过车辆数量
car3 = 0 #3车道经过车辆数量
vehicle_in = 0
vehicle_out = 0

# 视频帧循环
while cap.isOpened():
    success, frame = cap.read()
    if success:
        results = model.track(frame, conf=0.3, persist=True)
        #track_ids = results[0].boxes.id.int().cpu().tolist()
        if results and results[0].boxes is not None and
            results[0].boxes.id is not None:
            track_ids = results[0].boxes.id.int().cpu().tolist()
        else:
            track_ids = [] # 或者其他适合的默认值
        tt = 0
        for track_id, box in zip(track_ids,
                                 results[0].boxes.data):
            if box[-1] == 2: # 目标为小汽车
                box_label(frame, box, '#' + str(track_id) + ' car',
                           (167, 146, 11))
                x1, y1, x2, y2 = box[:4]
                x = (x1 + x2) / 2
                y = (y1 + y2) / 2

                track = track_history[track_id]
                track.append((float(x), float(y))) #
                    追加当前目标ID的坐标

            if len(track) > 1:
                _, h = track[-2] # 提取前一帧的目标纵坐标

```

```

        if h < size[1] - tt and y >= size[1] - tt:
            vehicle_out += 1 # out计数加1

        if h > size[1] - tt and y <= size[1] - tt:
            vehicle_in += 1 # in计数加1

    elif box[-1] == 5: # 目标为巴士
        box_label(frame, box, '#' + str(track_id) + ' bus',
                  (67, 161, 255))

        x1, y1, x2, y2 = box[:4]
        x = (x1 + x2) / 2
        y = (y1 + y2) / 2
        track = track_history[track_id]
        track.append((float(x), float(y))) # x, y center
        point

        if len(track) > 1:
            _, h = track[-2]
            if h < size[1] - tt and y >= size[1] - tt:
                vehicle_out += 1
            if h > size[1] - tt and y <= size[1] - tt:
                vehicle_in += 1

    elif box[-1] == 7: # 目标为卡车
        box_label(frame, box, '#' + str(track_id) + ' truck',
                  (19, 222, 24))

        x1, y1, x2, y2 = box[:4]
        x = (x1 + x2) / 2
        y = (y1 + y2) / 2
        track = track_history[track_id]
        track.append((float(x), float(y))) # x, y center
        point

        if len(track) > 1:
            _, h = track[-2]
            if h < size[1] - tt and y >= size[1] - tt:
                vehicle_out += 1
            if h > size[1] - tt and y <= size[1] - tt:

```

```

        vehicle_in += 1

    elif box[-1] == 3: # 目标为摩托车
        box_label(frame, box, '#' + str(track_id) + ' '
                  'motor', (186, 55, 2))

        x1, y1, x2, y2 = box[:4]
        x = (x1 + x2) / 2
        y = (y1 + y2) / 2
        track = track_history[track_id]
        track.append((float(x), float(y))) # x, y center
        point

    if len(track) > 1:
        _, h = track[-2]
        if h < size[1] - tt and y >= size[1] - tt:
            vehicle_out += 1
        if h > size[1] - tt and y <= size[1] - tt:
            vehicle_in += 1
            # 绘制基准线
        cv2.line(frame, (30, size[1] - tt), (size[0] - 30,
                                              size[1] - tt), color=(25, 33, 189), thickness=2,
                 lineType=4)

    # 实时显示进、出车辆的数量
    cv2.putText(frame, 'in: ' + str(vehicle_in), (595,
                                                   size[1] - 410),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
    cv2.putText(frame, 'out: ' + str(vehicle_out), (573,
                                                   size[1] - 370),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    cv2.putText(frame, "https://blog.csdn.net/zhaocj", (25,
                                                       50),
                cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    cv2.imshow("YOLOv8 Tracking", frame) # 显示标记好的当前帧图像
    videoWriter.write(frame) # 写入保存

    if cv2.waitKey(1) & 0xFF == ord("q"): # 'q'按下时，终止运行
        break

```

```
        else: # 视频播放结束时退出循环
            break

# 释放视频捕捉对象，并关闭显示窗口
cap.release()
videoWriter.release()
cv2.destroyAllWindows()
```

dealVideo.py

```
import cv2
import pandas as pd
path1 = "the path of your data"
path2 = "the path of your data"

# 读取CSV文件
df = pd.read_csv(path1 + '103output2.csv',
                  encoding='ISO-8859-1')

# 读取视频
cap = cv2.VideoCapture(path2 + '2.MP4')

# 获取视频的基本信息
fps = cap.get(cv2.CAP_PROP_FPS)
frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# 创建一个 VideoWriter 对象来保存新视频
fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter('output_video2.mp4', fourcc, fps, (width,
                                                       height))

# 存储上一次打印的文本
last_text = ''

# 逐帧处理
for i in range(frame_count):
    ret, frame = cap.read()
```

```

if not ret:
    break

frame_num = i + 1
if frame_num in df.iloc[:, 0].values: # 第一列数据为帧数，从1开始
    row_index = df.index[df.iloc[:, 0] == frame_num].tolist()
    if row_index: # 确保找到了对应的行
        last_text = str(df.iloc[row_index[0], 16]) # 更新文本

# 在帧上绘制文本
cv2.putText(frame, f'r--> : {last_text}', (595, height -
180), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 4)

# 写入新视频
out.write(frame)

# 释放资源
cap.release()
out.release()
cv2.destroyAllWindows()

```

giveTop.py

```

import pandas as pd

# 读取CSV文件
path = "your data path"
df = pd.read_csv(path + '108output2.csv', encoding='ISO-8859-1')

# 获取第17列数据
column_17 = df.iloc[:, 16] # 注意：索引从0开始，所以第17列的索引是16

# 创建新列表，应用条件
new_data = [min(value, 100) for value in column_17]

# 将新列添加到原数据中
df['r'] = new_data

# 保存为新文件
df.to_csv('108output2.csv', index=False)

```

getFirstZhen.py

```
import cv2

# 读取视频文件
video_path = 'your data path'
cap = cv2.VideoCapture(video_path)

# 检查视频是否打开
if cap.isOpened():
    frame_number = 450 # 选择要读取的帧数（从0开始计数）
    cap.set(cv2.CAP_PROP_POS_FRAMES, frame_number) # 设置当前帧位置

    ret, frame = cap.read() # 读取指定帧
    if ret:
        cv2.imwrite(f'frame_{frame_number}.jpg', frame) # 保存指定帧
    else:
        print("无法读取指定帧")
else:
    print("无法打开视频文件")

cap.release() # 释放视频对象
```

getPointPosition.py

```
import cv2

# 存储点击的坐标
coordinates = []

# 鼠标点击回调函数
def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        coordinates.append((x, y)) # 保存坐标
        # 在图像上绘制圆点
        cv2.circle(img, (x, y), 5, (0, 255, 0), -1) # 绿色圆点
        # 在右侧绘制坐标文本
        cv2.putText(img, f"({x}, {y})", (x + 10, y - 10),
```

```

        cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2) # 红色文本
cv2.imshow('Image', img) # 显示更新后的图像
print(f"Clicked coordinates: ({x}, {y})")

# 读取图像
img_path = 'E:\\projects\\yolov8\\ultralytics\\car\\1.jpg'
img = cv2.imread(img_path)

# 创建窗口并设置鼠标回调
cv2.imshow('Image', img)
cv2.setMouseCallback('Image', click_event)

# 等待按键以退出
while True:
    key = cv2.waitKey(1) # 每毫秒检查一次按键
    if key == ord('s'): # 按 's' 保存图像
        cv2.imwrite('updated_image.jpg', img) # 保存图像
        print("Image saved as 'updated_image.jpg'")
    elif key == 27: # 按 'Esc' 键退出
        break

cv2.destroyAllWindows()

```

calculateAcc.py

```

import pandas as pd

# 读取CSV文件
path = "your data path

df = pd.read_csv(path, encoding='ISO-8859-1')

# 转换相关列为数值类型
df.iloc[:, 13] = pd.to_numeric(df.iloc[:, 13], errors='coerce')
df.iloc[:, 12] = pd.to_numeric(df.iloc[:, 12], errors='coerce')
# 转换时间列为秒
df.iloc[:, 10] = pd.to_timedelta(df.iloc[:, 10]).dt.total_seconds()
# 计算新列'occ'

```

```
df['occ'] = ((df.iloc[:, 13] / df.iloc[:, 12]) * 180) /  
    (df.iloc[:, 10] - df.iloc[:, 10].shift(1))  
  
# 保存结果  
df.to_csv('108output2.csv', index=False)
```

calculateAcc.py

```
import pandas as pd  
  
# 读取CSV文件  
path = "your data path  
  
df = pd.read_csv(path, encoding='ISO-8859-1')  
  
# 转换相关列为数值类型  
df.iloc[:, 13] = pd.to_numeric(df.iloc[:, 13], errors='coerce')  
df.iloc[:, 12] = pd.to_numeric(df.iloc[:, 12], errors='coerce')  
# 转换时间列为秒  
df.iloc[:, 10] = pd.to_timedelta(df.iloc[:,  
    10]).dt.total_seconds()  
# 计算新列'occ'  
  
df['occ'] = ((df.iloc[:, 13] / df.iloc[:, 12]) * 180) /  
    (df.iloc[:, 10] - df.iloc[:, 10].shift(1))  
  
# 保存结果  
df.to_csv('108output2.csv', index=False)
```

draw1.py

```
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from matplotlib.dates import DateFormatter  
from scipy.stats import pearsonr  
from scipy.stats import spearmanr  
import seaborn as sns  
  
plt.rcParams['font.family'] = 'Microsoft YaHei'
```

```

plt.rcParams['axes.unicode_minus'] = False

# 加载CSV文件
data1 = pd.read_excel('数据/103output1.xlsx', header=0)
data2 = pd.read_excel('数据/103output2.xlsx', header=0)

data1 = data1[['时间', '1车道经过车数', '2车道经过车数',
               '经过车总数(流量)', '密度(辆/km)', '模拟速度(km/h)',
               'occ', 'R', 'r']]
data2 = data2[['时间', '1车道经过车数', '2车道经过车数',
               '经过车总数(流量)', '密度(辆/km)', '模拟速度(km/h)',
               'occ', 'R', 'r']]
data2 = data2.drop(index=0)

data1['时间'] = pd.to_datetime(data1['时间'], format='%H:%M:%S',
                               errors='coerce')
data2['时间'] = pd.to_datetime(data2['时间'], format='%H:%M:%S',
                               errors='coerce')

# 校准合并
data2['时间'] = data2['时间'] - pd.Timedelta(seconds=4)
data = pd.concat([data1, data2], ignore_index=True)

data['时间p'] = data['时间'].dt.hour * 3600 +
               data['时间'].dt.minute * 60 + data['时间'].dt.second

data.set_index('时间', inplace=True)
data['1车道4分钟经过车数'] =
    data['1车道经过车数'].rolling('4min').sum().shift(-3)
data['2车道4分钟经过车数'] =
    data['2车道经过车数'].rolling('4min').sum().shift(-3)
data['4分钟经过车总数'] =
    data['经过车总数(流量)'].rolling('4min').sum().shift(-3)
data['4分钟内车辆密度均值'] =
    data['密度(辆/km)'].rolling('4min').mean()
data['4分钟内车辆平均速度'] =
    data['模拟速度(km/h)'].rolling('4min').mean()
data.reset_index(inplace=True)

```

```

data = data.dropna(subset=['1车道4分钟经过车数',
                           '2车道4分钟经过车数', '4分钟经过车总数',
                           '4分钟内车辆密度均值', '4分钟内车辆平均速度'])

# cov1 = data['4分钟经过车总数'].cov(data['4分钟内车辆密度均值'])
# cov2 = data['4分钟经过车总数'].cov(data['4分钟内车辆平均速度'])
# cov3 = data['4分钟内车辆密度均值'].cov(data['4分钟内车辆平均速度'])
# print("车流量与车辆密度的协方差: ", cov1)
# print("车流量与车辆速度的协方差: ", cov2)
# print("车辆速度与车辆密度的协方差: ", cov3)
#
# cor1, p1 = pearsonr(data['4分钟经过车总数'],
#                       data['4分钟内车辆密度均值'])
# cor2, p2 = pearsonr(data['4分钟经过车总数'],
#                       data['4分钟内车辆平均速度'])
# cor3, p3 = pearsonr(data['4分钟内车辆密度均值'],
#                       data['4分钟内车辆平均速度'])
# print("车流量与车辆密度的Pearson相关系数:", cor1)
# print("车流量与车辆速度的Pearson相关系数:", cor2)
# print("车辆速度与车辆密度的Pearson相关系数:", cor3)

corr1, p_1 = spearmanr(data['4分钟经过车总数'],
                        data['4分钟内车辆密度均值'])
corr2, p_2 = spearmanr(data['4分钟经过车总数'],
                        data['4分钟内车辆平均速度'])
corr3, p_3 = spearmanr(data['4分钟内车辆密度均值'],
                        data['4分钟内车辆平均速度'])
print("车流量与车辆密度的Spearman相关系数:", corr1)
print("车流量与车辆速度的Spearman相关系数:", corr2)
print("车辆速度与车辆密度的Spearman相关系数:", corr3)

pea_matrix = data[['4分钟经过车总数', '4分钟内车辆密度均值',
                   '4分钟内车辆平均速度']].corr(method='spearman')
plt.figure(figsize=(8, 6))
heatmap = sns.heatmap(pea_matrix, annot=True, cmap='coolwarm',
                      linewidths=0.5, fmt='.2f')
heatmap.set_xticklabels(['车流量', '车辆密度', '车辆速度'],
                       rotation=45)
heatmap.set_yticklabels(['车流量', '车辆密度', '车辆速度'],

```

```

    rotation=45)
plt.title('车流量、车辆密度与车辆速度的Spearman相关系数热力图')
plt.show()

plt.figure(figsize=(14, 7))
color_1 = np.where(data['r'] > 20, 'red', 'yellow') # 1车道颜色
color_2 = np.where(data['r'] > 20, 'red', 'green') # 2车道颜色
color_t = np.where(data['r'] > 20, 'red', 'blue') # 总流量颜色
plt.scatter(data['时间'], data['1车道4分钟经过车数'],
            color=color_1, s=30, label='1车道车流量', edgecolor='none')
plt.scatter(data['时间'], data['2车道4分钟经过车数'],
            color=color_2, s=30, label='2车道车流量', edgecolor='none')
plt.scatter(data['时间'], data['4分钟经过车总数'], color=color_t,
            s=30, label='总车流量', edgecolor='none')
plt.title('车流量随时间变化图')
plt.gca().xaxis.set_major_formatter(DateFormatter('%H:%M:%S'))
plt.xlabel('时间')
plt.ylabel('流量')
plt.legend()
plt.grid()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(14, 7))
color = np.where(data['r'] > 20, 'red', 'black')
plt.scatter(data['时间'], data['4分钟内车辆密度均值'], color=color,
            s=30, edgecolor='none')
plt.title('车辆密度随时间变化图')
plt.gca().xaxis.set_major_formatter(DateFormatter('%H:%M:%S'))
plt.xlabel('时间')
plt.ylabel('密度(辆/km)')
plt.grid()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

plt.figure(figsize=(14, 7))
color = np.where(data['r'] > 20, 'red', 'black')

```

```

plt.scatter(data['时间'], data['4分钟内车辆平均速度'], color=color,
            s=30, edgecolor='none')
plt.title('车辆速度随时间变化图')
plt.gca().xaxis.set_major_formatter(DateFormatter('%H:%M:%S'))
plt.xlabel('时间')
plt.ylabel('速度(km/h)')
plt.grid()
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

fig, ax1 = plt.subplots(figsize=(14, 7))
ax1.plot(data['时间'], data['4分钟经过车总数'], color='blue',
          label='车流量', marker='o')
ax1.set_xlabel('时间')
ax1.set_ylabel('车流量', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')

ax2 = ax1.twinx()
ax2.plot(data['时间'], data['4分钟内车辆密度均值'], color='green',
          label='车辆密度', marker='s')
ax2.set_ylabel('车辆密度', color='green')
ax2.tick_params(axis='y', labelcolor='green')

ax3 = ax1.twinx()
ax3.spines['right'].set_position(('outward', 60))
ax3.plot(data['时间'], data['4分钟内车辆平均速度'], color='yellow',
          label='车辆速度', marker='^')
ax3.set_ylabel('车辆速度', color='red')
ax3.tick_params(axis='y', labelcolor='red')

plt.title('车流量、车辆密度与车辆速度关系图')
ax1.xaxis.set_major_formatter(DateFormatter('%H:%M:%S'))

# 添加图例
ax1.legend(loc='upper left')
ax2.legend(loc='upper center')
ax3.legend(loc='upper right')

```

```
# 显示网格  
ax1.grid()  
  
# 调整布局  
plt.tight_layout()  
plt.show()
```

calculateAcc.py

```
import pandas as pd  
  
# 读取CSV文件  
path = "your data path  
  
df = pd.read_csv(path, encoding='ISO-8859-1')  
  
# 转换相关列为数值类型  
df.iloc[:, 13] = pd.to_numeric(df.iloc[:, 13], errors='coerce')  
df.iloc[:, 12] = pd.to_numeric(df.iloc[:, 12], errors='coerce')  
# 转换时间为秒  
df.iloc[:, 10] = pd.to_timedelta(df.iloc[:,  
    10]).dt.total_seconds()  
# 计算新列'occ'  
  
df['occ'] = ((df.iloc[:, 13] / df.iloc[:, 12]) * 180) /  
    (df.iloc[:, 10] - df.iloc[:, 10].shift(1))  
  
# 保存结果  
df.to_csv('108output2.csv', index=False)
```

LSTPrompt

```
# 特别说明引用他人论文中的代码来实现。地址为  
https://github.com/AdityaLab/lstprompt
```