

基于 Python 语言的交通流量管控与预测分析

摘要

随着城市化进程的加速、交通需求不断增加,城市道路交通流量管控与优化成为亟需解决的重要问题^[1]。本文建立信号灯优化模型,运用 K-均值聚类算法、遗传算法和仿真模拟,从而制定合理的交通流预测分析方案,为便于交通流量管控提供了重要参考。

针对问题一,为得到一天中各时段的相位车流量,首先对数据进行预处理,将未挂车牌的车辆当作异常值处理,考虑到工作日、非工作日与节假日的影响,分析各路口交通流量,并进行可视化,将不同时间段进一步细分为划分为低峰期、高峰期和平峰期,然后建立 K-均值聚类算法模型且运用方向流量公式估算直行和转弯车辆比例,最后使用 Python 软件且结合公式模型计算出一天中三个时段的相位车流量(单位: pcu/h): 工作日中的低峰期时段:由东向西方向直行、左转、右转的车流量分别为 6754, 2251, 2251;由西向东方向直行、左转、右转的车流量分别为 2412, 804, 804;由南向北方向直行、左转、右转的车流量分别为 4573, 1524, 1524;由北向南方向直行、左转、右转的车流量分别为 8594, 2865, 2865。……(具体详见 5.1.5 结果展示)

针对问题二,为得到两条主路上的车流最高平均速度,首先进行数据的预处理,利用 python 软件将各交叉路口的总车流量进行可视化分析,在问题一的基础上将各路口车的流量划分为低峰期、高峰期和平峰期,构建所有交叉路口的车流量模型和信号灯优化模型,并设计绿波带进一步优化模型,然后采用遗传算法计算出每个路口的最优绿灯时长及周期,最后用仿真模拟检验优化模型,其模拟仿真网址为 <https://wwto.lanzouj.com/iFBcd29ibgna>,推导出整体道路的最优平均速度约为 50.4km/h,即 14.0m/s。

针对问题三,为估算出景区所需的临时停车位数量,本问首先根据行驶路径、车辆速度以及出现频率来对巡游车辆进行识别,并设置阈值,得到巡游车辆,然后建立排队论模型估计停车的需求量,最终得到景区所需的临时停车位数量范围为[430, 450]辆。

针对问题四,为评估五一期间和非五一期间临时管控措施在两条主路上的效果,首先对数据进行预处理,处理车辆通过方向、时间、车牌号等信息,剔除无关数据。并综合分析对比各路段的三个指标:流量密度、绕行车辆和通行方向。比较管控措施前后这些指标的变化,得到五一期间的临时管控措施在减少车流、优化车辆绕行以及规范通行方向方面有着显著效益的成果。

关键词: 交通流量管控 K-均值聚类算法 信号灯优化模型 仿真模拟 排队论模型

一、问题重述

1.1 背景介绍

交通流量是指在一定时间段内通过特定道路或交通网络的车辆数量，是交通管理中一个重要的指标，准确计算和分析交通流量可以更好地理解城市交通情况，改善交通拥堵问题，提高环境质量，为城市的可持续发展提供支持。

综合题目中所给的条件分析，影响车流量的因素有很多，例如工作日的影响和信号灯周期时长的影响。现整合题中资源如下：

1、受工作日与节假日的影响：工作日的早高峰较节假日少，且工作日因通勤需要，早晚高峰期，通勤人数会增加，导致车流量上升。

2、信号灯周期时长的影响：当绿灯周期过短时，车辆通过交叉口的时间减少，会导致交通拥堵。同时，较长的红灯周期也会影响到整体的交通流畅度。

1.2 问题概述

问题一：将一天分成若干个时段，分析经中路-纬中路交叉口车流量的差异，最终估计不同时段各个相位的车流量。

问题二：在保证车辆通信的前提下，建立优化模型，对经中路和纬中路上所有的路口信号灯进行优化配置，以使主路上的车流平均速度最大。

问题三：分析五一黄金周期间的数据，识别停车位的巡游车辆，并估算出景区所需的临时停车位数量。

问题四：评价五一黄金周期间小镇对景区周边道路实行的临时交通管理措施的效果。

二、问题分析

根据对上述问题的探讨，为了更好地分析题中的四个问题，使用亿图软件画出题目中的小镇主要道路示意图。如下：

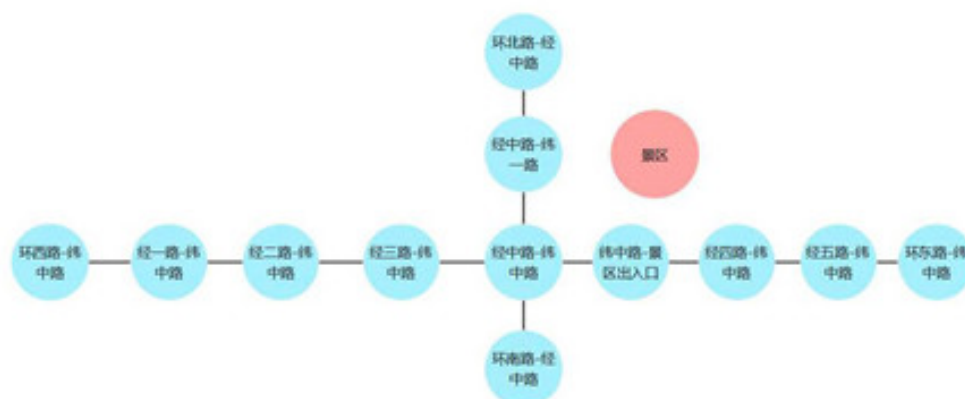


图 2-1 小镇主要道路示意图

2.1 问题 1 的分析

根据题意，首先对数据进行预处理，考虑到工作日、非工作日与节假日的影响，分析各路口交通流量，并进行可视化，将不同时间段进一步细分为划分为低峰期、高峰期和平峰期，然后建立数学模型估算直行和转弯车辆比例，最后运用 Python 软件且结合公式模型计算出一天中各时段的相位车流量。

2.2 问题 2 的分析

为得到两条主路的最高车流平均速度，首先进行数据的预处理，在问题一的基础上将各路口车的流量划分为低峰期、高峰期和平峰期，构建所有交叉路口的车流量模型和信号灯优化模型，并设计绿化带进一步优化模型，然后采用遗传算法计算出每个路口的最优绿灯时长及周期，最后用仿真模拟检验优化模型，推导出整体道路的最优平均速度。

2.3 问题 3 的分析

根据题目要求，为估算出景区所需的临时停车位数量，本问首先根据行驶路径、车辆速度以及出现频率来对巡游车辆进行识别，并设置阈值，得到巡游车辆，然后建立排队论模型估计停车的需求量，最终得到景区所需的临时停车位数量。

2.4 问题 4 的分析

依题意，首先对数据进行预处理，提取五一期间和非管控期间的交通数据，需选择评价指标：车流量、绕行车辆和通行方向，比较管控措施前后这些指标的变化，进而评价管控措施的效果。

三、模型假设

- 1.假设问题二中黄灯对整个模型的影响忽略不计；
- 2.假设问题二中平均停车时间为 2 小时(每个停车位每小时有 0.5 辆车离开)；
- 3.假设问题三仿真模拟中无交通事故发生。

四、定义与符号说明

符号定义	符号说明
k	聚类的数量
X_i	数据 i 点的位置
μ_k	聚类 C_j 的中心点

C_i	在点 i 处建设配送站的成本
$\ x_i - \mu_j\ $	数据 x_i 到质心 μ_j 的欧几里得距离
J	所有数据点到其所属聚类质心的距离
$\xi_{i,j}$	总流量
ω	左转
γ	直行
λ	右转
G_{ij}	交叉口 i 上第 j 个相位的绿灯时间
R_i	交叉口 i 的红灯时间
T_i	第 i 个交叉信号灯周期总时间

五、模型的建立与求解

5.1 问题 1 的模型建立与求解

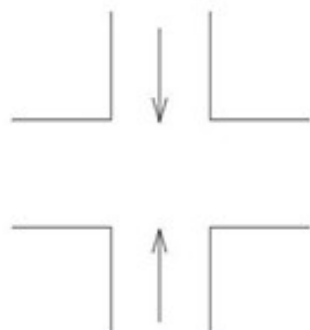
5.1.1 题目信息分析

本题需要估计不同时段经中路-纬中路交叉口各个相位（包括四个方向直行、转弯）车流量，分析本问附件 1 与附件 2 的数据，具有以下特征：

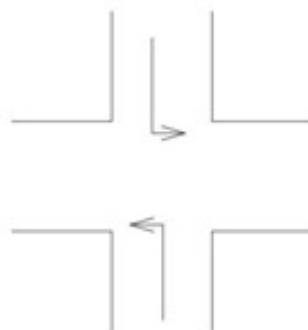
1、数据的庞大性：由于附件 2 纬中路各交叉口车辆信息的数据高达 8844996，因此为了高效地解决本问题，会结合历史数据和现有的流量模型进行估算。

2、交叉路口的相位只有四种组合方式：相位 1 南北直行、相位 2 南北左转、相位 3 东西直行、相位 4 东西左转。

用 CAD 软件大致绘画出四种相位的组合方式，如下：



相位 1 南北直行



相位 2 南北左转

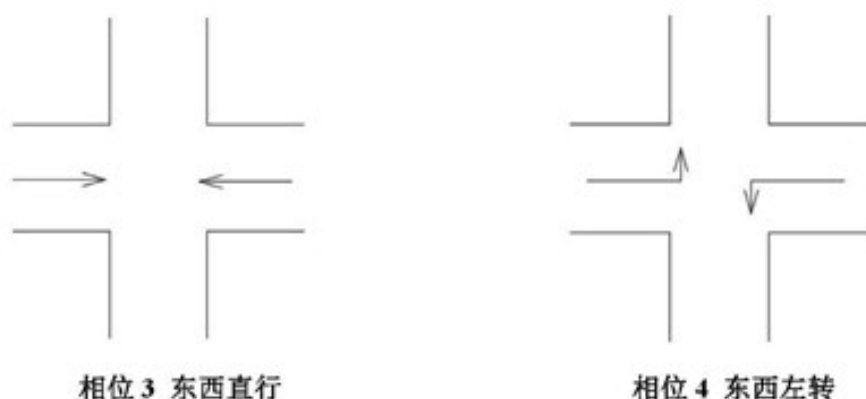


图 5-1 交叉路口相位

5.1.2 数据预处理

本问的数据较多，在建立模型前首先对数据进行预处理，发现附件 2 中的数据存在异常值，即未挂车牌的车辆，因此需要对数据进行异常值处理。附件 2 中纬中路各交叉口车辆信息中存在近 20 万辆未挂车牌的车辆，由于该部分车辆只占总数据的比例为 0.025，因此，使用 python 软件将未挂车牌的车辆进行异常值处理，将其直接从中剔除，后将经中路-纬中路的数据整理筛选出来。

Step1: 考虑到工作日、非工作日与节假日的影响，分析交通流量

运用 python 软件将筛选出来经中路-纬中路的数据按工作日、非工作日与节假日区分，区分后的表见支撑材料一。且统计每个时段的车流量，通过监控数据，分别统计各个时段内，每个相位通过的车辆总数。由于监控设备安装在停车线后方，并不知道车辆通过停车线后的转向。因此，我们通过研究下一个路口的车流量来分析该路口的转向。

Step2: 将不同时间段的交通流量进行可视化

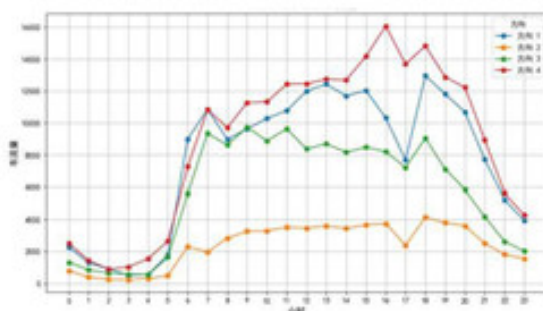


图 5-2 工作日不同时段的车流量

如上图，工作日在不同时段的车流量有着显著差异。凌晨时段四个方向的车流量都较少，9:00 出现早高峰，为通勤高峰，车流量大；9:00 - 17:00 的车流量整体呈现平稳状态；下午 17:00 至晚上 20:00 为晚高峰，与早高峰相似通勤

车辆多，车流量大，夜间时段车流量逐渐减少，夜间娱乐、购物和晚餐时间段，车流量相对较高。

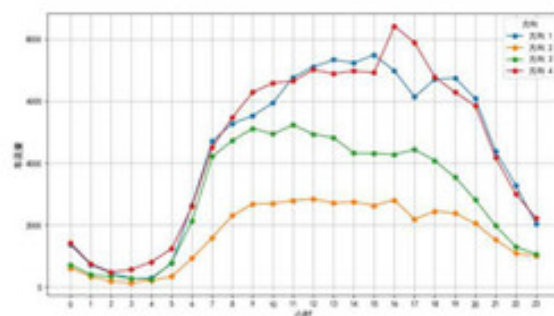


图 5-3 非工作日不同时段的车流量

由图可知，非工作日不同时段的最大车流量近 8500 pcu/h ，为工作日的最高车流量的 1/2。从图的整体上来看，非工作日的不同时段车流量与工作日不同时段的车流量大体一致，但车流量数值不同，相比于工作日时的车流量非工作日的车流量较少。

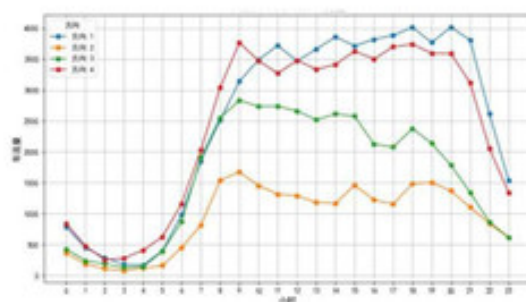


图 5-4 节假日不同时段的车流量

如图，节假日的最高车流量最少，为工作日时车流量的 1/4，为非工作日时车流量的 1/2。节假日的不同时段车流量的大小与工作日和非工作日的车流量大小大体相同，但在不同方向的车流量存在一定差异，在中午 12:00 之前各方向的车流量排名为：方向 4 > 方向 1 > 方向 3 > 方向 2，当过了 12:00 以后方向 1 的车流量较高，即由东向西方向的车比较多。

5.1.3 建立 K-均值聚类算法模型

^[2]K-均值聚类算法的目标是将数据集划分为 K 个族，使得同一族内的点尽可能相似，而不同族之间的点尽可能不同。本问根据题意建立如下公式：

目标函数：

$$J = \sum_{j=1}^k \sum_{x_i \in C_j} \|x_i - \mu_j\|^2 \quad (1)$$

其中 k 是聚类的数量， x_i 是数据 i 点的位置， μ_j 是聚类 C_j 的中心点， $\|x_i - \mu_j\|$

表示是数据点 x_i 到质心 μ_j 的欧几里得距离。 J 表示所有数据点到其所属聚类质心的距离平方和， J 越小代表聚类的效果越好，分的种类越好。

5.1.4 模型的求解

Step3: 将不同时间段进一步细分为划分为低峰期、高峰期和平峰期

根据题目要求，需要估算不同时间段各个相位的车流量，在前面分别对工作日、非工作日和节假日的分段基础上，利用上述模型，进一步细分时段。

如下：

表 5-5 工作日的进一步细分时段

时间	1	2	3	4	族	时期
7	10845	1970	9339	10834	0	高峰期
8	8976	2836	8620	9707	0	高峰期
9	9628	3264	9719	11264	0	高峰期
10	10283	3285	8876	11317	0	高峰期
11	10764	3532	9621	12422	0	高峰期
12	11992	3464	8370	12453	0	高峰期
13	12423	3608	8682	12735	0	高峰期
14	11690	3449	8166	12684	0	高峰期
15	12025	3657	8472	14165	0	高峰期
16	10309	3746	8202	16057	0	高峰期
17	7671	2393	7213	13677	0	高峰期
18	12945	4130	9040	14820	0	高峰期
19	11808	3805	7112	12867	0	高峰期
20	10672	3608	5842	12223	0	高峰期
0	2266	784	1314	2506	1	低峰期
1	1288	403	855	1435	1	低峰期
2	912	245	650	888	1	低峰期
3	519	241	588	1028	1	低峰期
4	557	306	543	1550	1	低峰期
5	1788	491	1649	2647	1	低峰期
23	3927	1550	2022	4269	1	低峰期
6	8983	2301	5604	7256	2	平峰期
21	7705	2502	4174	8930	2	平峰期
22	5237	1811	2634	5624	2	平峰期

表 5-6 非工作日的进一步细分时段

时间	1	2	3	4	族	时期
0	1369	613	723	1422	0	低峰期
1	715	341	399	748	0	低峰期
2	456	185	348	483	0	低峰期
3	280	141	284	562	0	低峰期

4	288	209	249	808	0	低峰期
5	766	343	794	1237	0	低峰期
7	4699	1594	4213	4508	1	高峰期
8	5278	2303	4727	5471	1	高峰期
9	5520	2681	5114	6279	1	高峰期
10	5942	2698	4940	6583	1	高峰期
11	6768	2791	5227	6646	1	高峰期
12	7102	2843	4933	7009	1	高峰期
13	7332	2713	4819	6886	1	高峰期
14	7234	2747	4318	6970	1	高峰期
15	7489	2629	4309	6916	1	高峰期
16	6975	2807	4274	8417	1	高峰期
17	6134	2184	4438	7888	1	高峰期
18	6699	2444	4076	6777	1	高峰期
19	6746	2385	3544	6279	1	高峰期
20	6081	2051	2819	5854	1	高峰期
6	2643	923	2123	2596	2	平峰期
21	4380	1525	1982	4190	2	平峰期
22	3278	1093	1299	3003	2	平峰期
23	2048	1000	1062	2222	2	平峰期

表 5-7 节假日的进一步细分时段

时间	1	2	3	4	族	时期
8	2514	1539	2550	3042	0	高峰期
9	3141	1673	2834	3768	0	高峰期
10	3490	1454	2737	3475	0	高峰期
11	3722	1316	2738	3271	0	高峰期
12	3469	1290	2661	3483	0	高峰期
13	3663	1182	2522	3336	0	高峰期
14	3861	1169	2613	3414	0	高峰期
15	3715	1462	2579	3632	0	高峰期
16	3822	1223	2125	3502	0	高峰期
17	3885	1155	2080	3702	0	高峰期
18	4016	1483	2375	3743	0	高峰期
19	3771	1506	2138	3597	0	高峰期
20	4019	1369	1786	3592	0	高峰期
21	3808	1102	1342	3117	0	高峰期
6	988	453	872	1161	1	平峰期
7	1850	814	1909	2027	1	平峰期
22	2620	838	865	2056	1	平峰期
23	1535	614	615	1342	1	平峰期
0	785	362	427	837	2	低峰期
1	446	184	237	476	2	低峰期
2	287	108	191	254	2	低峰期

3	177	81	126	279	2	低峰期
4	167	117	141	409	2	低峰期
5	397	160	390	625	2	低峰期

由上表可知，一天 24 小时大致按 7: 3: 2 的比例将工作日、非工作日与节假日的各时间段进一步细分为高峰期、低峰期和平峰期，其中这三个不同时期的高峰期最多，都有 14 个小时，低峰期有 6-7 小时，平峰期最少，只有 3-4 小时，由此可见，该路口的交通压力增大，容易增加事故的风险。将三个不同时间段的细分时段汇总表如下：

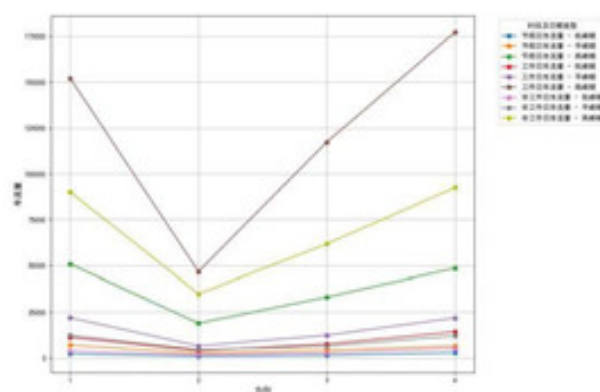


图 5-8 三个不同时段细分时段汇总

由图可知，工作日存在高峰期的车流量最多，然后依次为非工作日的平峰期、节假日的高峰期、工作日的平峰期、工作日的低峰期、非工作日的平峰期、节假日的平峰期、非工作日的低峰期、节假日的低峰期。

Step4: 估算直行和转弯车辆比例

由于附件 2 中只提供了车辆在交叉路口相位（即直行）的数据，并未给出转弯的数据，无法直接区分车辆的转向行为，因此，需要结合历史数据和现有的流量模型，通过细分时段和估算比例的方法，得到每个相位在不同时间段内的车流量估算。为计算各方向流量，建立如下公式：

假设车辆在交叉口的转向比例是已知的（通过历史数据或交通规则推断），则可以估计左转、直行和右转的流量。设转向比例为 ω , γ , λ 分别代表左转、直行和右转的比例，其中 $\xi_{i,j}$ 表示总流量。则各方向流量表达式为：

① 直行流量

$$\xi_{i,j}^{\text{直行}} = \omega_i \times \xi_{i,j} \quad (2)$$

② 左转流量

$$\xi_{i,j}^{\text{左转}} = \gamma_i \times \xi_{i,j} \quad (3)$$

③右转流量

$$\xi_{i,j}^{\text{右转}} = \lambda_1 \times \xi_{i,j} \quad (4)$$

由于车的型号过多，根据每个路口的历史数据，了解在某个时段内，特定路口的车辆转弯比例。比如，如果在早高峰时段通过上述公式可得 20% 的车辆是转弯的，那么可以将总车流量的 20% 划分为转弯车辆，其余 80% 为直行车辆。

Step5: 运用 Python 软件结合公式模型分配转弯比例

将进一步细分时段的工作日、非工作日与节假日进行汇总，得到下表：

图 5-9 工作日进一步细分时段车流量汇总

时期	1	2	3	4
低峰期	11257	4020	7621	14323
平峰期	21925	6614	12412	21810
高峰期	152031	46747	117274	177225

图 5-10 非工作日进一步细分时段车流量汇总

时期	1	2	3	4
低峰期	3874	1832	2797	5260
平峰期	12349	4541	6466	12011
高峰期	89999	34870	61751	92483

图 5-11 节假日进一步细分时段车流量汇总

时期	1	2	3	4
低峰期	2259	1012	1512	2880
平峰期	6993	2719	4261	6586
高峰期	50896	18923	33080	48674

运用 Python 软件结合公式模型将车辆行驶方向的比例求出来，得到不同时段车辆行驶方向的比例：

①高峰期：直行：左转：右转=0.8：0.1：0.1

②平峰期：直行：左转：右转=0.7：0.2：0.1

③低峰期：直行：左转：右转=0.6：0.2：0.2

5.1.5 结论分析

(1) 工作日三个时段的相位车流量：

时期	1 直行	1 左转	1 右转	2 直行	2 左转	2 右转	3 直行	3 左转	3 右转	4 直行	4 左转	4 右转
低峰期	6754	2251	2251	2412	804	804	4573	1524	1524	8594	2865	2865
平峰期	15347	4385	2192	4630	1323	661	8688	2482	1241	15267	4362	2181
高峰期	121625	15203	15203	37398	4675	4675	93819	11727	11727	141780	17722	17722

由表可知，低峰期直行和左转的车流量相对较低；在平峰期直行车流量显著增加，左转和右转车流量也有所增加，总体车流量较低峰期有明显增长；在高峰期，直行车流量最高可达 8594 pcu/h 。

(2) 非工作日三个时段的相位车流量：

时期	1 直行	1 左转	1 右转	2 直行	2 左转	2 右转	3 直行	3 左转	3 右转	4 直行	4 左转	4 右转
低峰期	2324	775	775	1099	366	366	1678	559	559	3156	1052	1052
平峰期	8644	2470	1235	3179	908	454	4526	1293	647	8408	2402	1201
高峰期	71999	9000	9000	27896	3487	3487	49401	6175	6175	73986	9248	9248

如上表，低峰期整体交通相对畅通，左转和右转车流量相对较低。平峰期流量显著增加，直行车流量和左转车流量都有所提高；高峰期的车流量大幅增加，特别是直行车流量，达到 71999 pcu/h ，可能会导致交通拥堵。

(3) 节假日三个时段的相位车流量：

时期	1 直行	1 左转	1 右转	2 直行	2 左转	2 右转	3 直行	3 左转	3 右转	4 直行	4 左转	4 右转
低峰期	1355	452	452	607	202	202	907	302	302	1728	576	576
平峰期	4895	1399	699	1903	544	272	2983	852	426	4610	1317	659
高峰期	40717	5090	5090	15138	1892	1892	26464	3308	3308	38939	4867	4867

注：其中 1 为由东向西方向，2 为由西向东方向，3 为由南向北方向，4 为由北向南方向。
单位： pcu/h

由上表可知，节假日总体车流量较低，分布相对均衡，但都显著低于平峰期和高峰期；平峰期的车流量明显高于低峰期，尤其是直行和左转车流量有所增加；高峰期的直行车流量非常高，最高达 73986 pcu/h ，可能会导致严重的拥堵事故。

5.2 问题 2 的模型建立与求解

5.2.1 题目信息分析

本题要求优化经中路和纬中路上所有交叉口的信号灯配置，以提高两条主路上的车流平均速度。车流的平均速度受到多种因素影响，如交通信号因素和交通密度因素，具体因素分析如下：

- 1、交通信号因素：红绿灯的周期和控制策略直接影响车辆的行驶速度
- 2、交通密度因素：车流量的多少决定了车速，特别是在高峰时段和低谷时段，车流量多时速度通常较低。

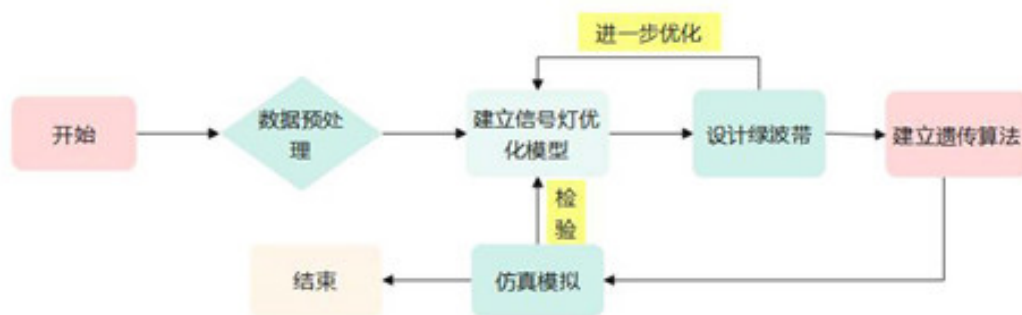


图 5-12 问题二流程图

5.2.2 数据预处理, 并进行可视化

基于问题一的基础上, 首先利用 python 软件对附件 2 中各交叉路口的总车流量进行数据分析处理, 进而绘制出每个交叉路口车流量在各时间段的总车流量图。

如下:

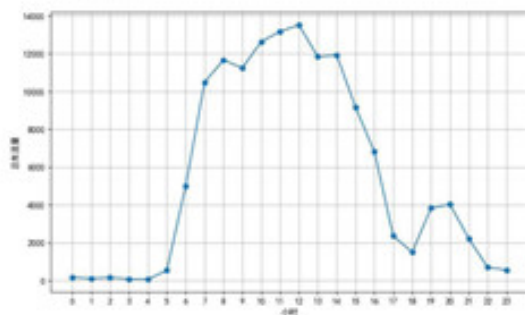


图 5-13 环东路-纬中路

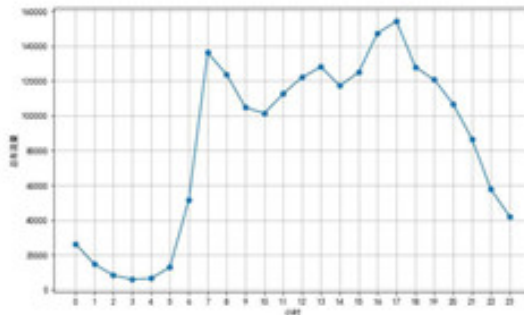


图 5-14 环西路-纬中路

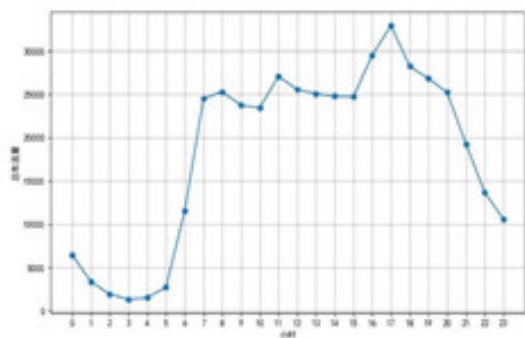


图 5-15 经二路-纬中路

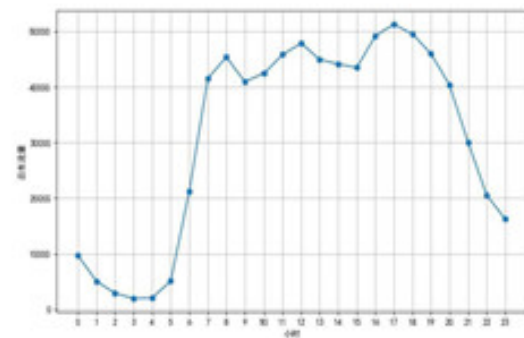


图 5-16 经三路-纬中路

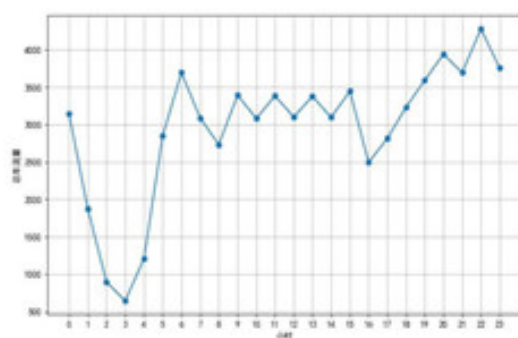


图 5-17 经四路-纬中路

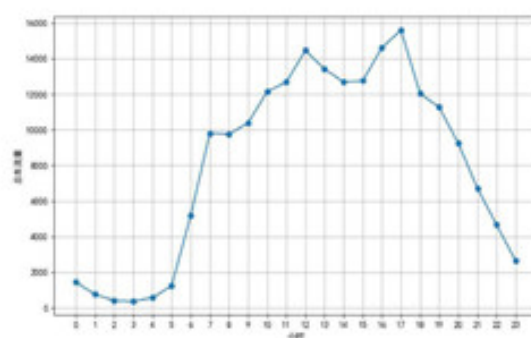


图 5-18 经五路-纬中路

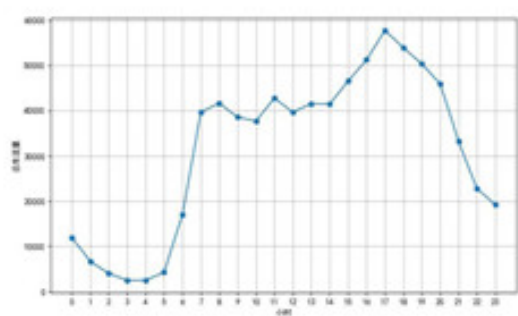


图 5-19 经一路-纬中路

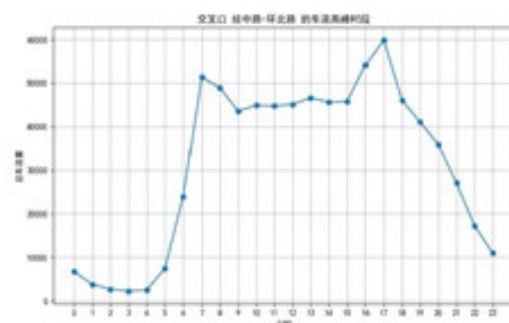


图 5-20 经中路-环北路

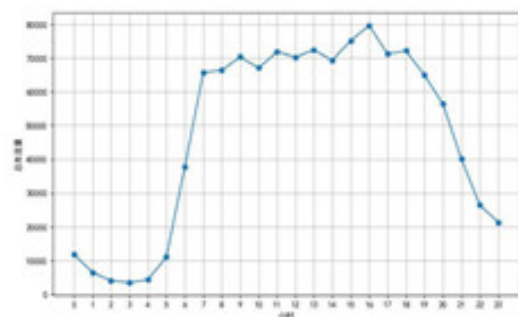


图 5-21 经中路-环南路

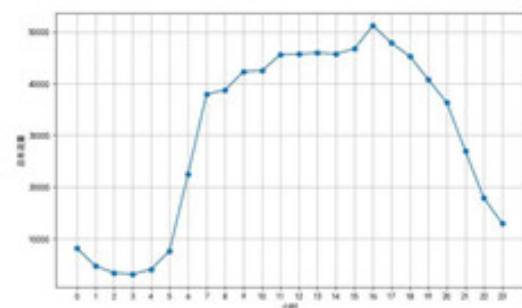


图 5-22 经中路-纬一路

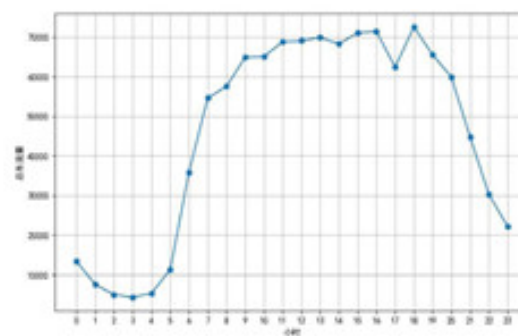


图 5-23 经中路-纬中路

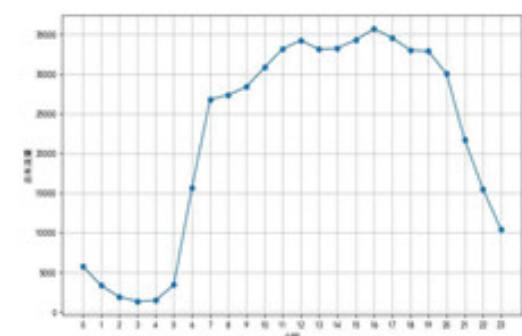


图 5-24 纬中路-景区出入口

由上述图可知，每个交叉路口的总车流量不一致，车流量最高的是环西路-纬中路交叉路口，高达 160000 pcu/h，最少的为经四-纬中路交叉路口，最高的

车流量只有 5000 pcu/h 左右。但是整体上都呈现上升趋于平缓最后减少的趋势。

Step1: 基于问题一的模型进行时段的划分

通过上述分析,各交叉路口的总车流量在不同时刻出现了不同的峰值,与问题一步骤一致,首先将每个交叉路口的车流量划分三个时段分别为工作日、非工作日与节假日,然后运用问题一建立的 k-均值聚类算法模型,再对各交叉路口的总车流量进行进一步的时段划分,按小时划分为低峰期,平峰期和高峰期。

由于表格数据过多,因此只取部分交叉路口的工作日、非工作日与节假日中的同一段时间内的高峰期、低峰期与平峰期数据,具体详见支撑材料二。

如下:

(1) 工作日

表 5-25 纬东路-纬中路

方向	时间	车流总量	总车流量	族	时期
1	7	1654	6914	2	高峰期
2	7	639	6914	2	高峰期
3	7	2186	6914	2	高峰期
4	7	2435	6914	2	高峰期
1	0	39	178	1	低峰期
2	0	9	178	1	低峰期
3	0	41	178	1	低峰期
4	0	89	178	1	低峰期
1	6	627	3373	0	平峰期
2	6	343	3373	0	平峰期
3	6	1043	3373	0	平峰期
4	6	1360	3373	0	平峰期

表 5-26 环西路-纬中路

方向	时间	车流总量	总车流量	族	时期
1	7	33548	95069	2	高峰期
2	7	20721	95069	2	高峰期
3	7	22772	95069	2	高峰期
4	7	18028	95069	2	高峰期
1	0	4939	13802	1	低峰期
2	0	3584	13802	1	低峰期
3	0	2487	13802	1	低峰期
4	0	2792	13802	1	低峰期
1	9	18326	56703	0	平峰期
2	9	13368	56703	0	平峰期
3	9	12581	56703	0	平峰期
4	9	12428	56703	0	平峰期

(2) 非工作日

表 5-27 经中路-南环路

方向	时间	车流总量	总车流量	族	时期
1	7	1818	10149	1	高峰期
2	7	339	10149	1	高峰期
3	7	5009	10149	1	高峰期
4	7	2983	10149	1	高峰期
1	0	415	2502	0	低峰期
2	0	206	2502	0	低峰期
3	0	1072	2502	0	低峰期
4	0	809	2502	0	低峰期
1	6	948	5570	2	平峰期
2	6	186	5570	2	平峰期
3	6	2769	5570	2	平峰期
4	6	1667	5570	2	平峰期

表 5-28 经中路-环北路

方向	时间	车流总量	总车流量	族	时期
1	7	4231	16600	1	高峰期
2	7	3857	16600	1	高峰期
3	7	4708	16600	1	高峰期
4	7	3804	16600	1	高峰期
1	6	2312	8864	0	平峰期
2	6	2176	8864	0	平峰期
3	6	2172	8864	0	平峰期
4	6	2204	8864	0	平峰期
1	0	519	3518	2	低峰期
2	0	1184	3518	2	低峰期
3	0	914	3518	2	低峰期
4	0	901	3518	2	低峰期

(3) 节假日

表 5-29 经中路-纬一路

方向	时间	车流总量	总车流量	族	时期
1	9	1240	7199	2	高峰期
2	9	900	7199	2	高峰期
3	9	2132	7199	2	高峰期
4	9	2927	7199	2	高峰期
1	0	204	1540	1	低峰期
2	0	365	1540	1	低峰期
3	0	476	1540	1	低峰期
4	0	495	1540	1	低峰期
1	7	616	3992	0	平峰期
2	7	488	3992	0	平峰期
3	7	1463	3992	0	平峰期
4	7	1425	3992	0	平峰期

表 5-30 经中路-环南路

方向	时间	车流总量	总车流量	族	时期
1	9	3336	11631	0	高峰期
2	9	2634	11631	0	高峰期
3	9	3287	11631	0	高峰期
4	9	2374	11631	0	高峰期
1	0	349	2069	1	低峰期
2	0	645	2069	1	低峰期
3	0	541	2069	1	低峰期
4	0	534	2069	1	低峰期
1	7	1902	6879	2	平峰期
2	7	1507	6879	2	平峰期
3	7	2033	6879	2	平峰期
4	7	1437	6879	2	平峰期

注：以上表格中，总车流量表示该路口在同一时间的不同方向的车流量总和。

由上表可知，在工作日与非工作日期间，高峰期主要集中在上午 7:00，低峰期主要集中在凌晨零点，且工作期间车流总量值较非工作日和节假日高；在节假日期间，高峰期主要集中在上午 9:00，低峰期主要集中在凌晨零点。因此，本问通过研究各路口车流量的峰值对下文信号灯时间进行划分。

5.2.3 建立模型

Step2: 建立交通流量模型

使用交通流量模型对整个交通线路进行模拟和分析，计算出不同站点的车流量和交通路口的最大通行能力，公式如下：

①不同交叉口车流量

$$Q_{ij} = k_{ij} \times v_{ij} \quad (5)$$

其中 v_{ij} 表示车速， k_{ij} 表示车道上单位长度的车流密度

②交叉路口的最大通行量

$$C_j = N_j \times S_j \quad (6)$$

其中 N_j 表示交叉路口的车道数， S_j 每条车道的饱和量

Step3: 建立信号灯优化模型

通过建立信号灯优化模型，优化信号灯的时间周期，提升整个交通系统的平均车速，本文依据题意建立以下公式：

目标函数:

$$\max V = \frac{D}{\sum_{i=1}^n (T_i + w_i)} \quad (7)$$

约束条件:

①每个交叉口的通行能力应该大于等于其车流量, 确保不会出现严重的拥堵。

$$M_i \geq F_i \quad (8)$$

其中 F_i 是第 i 个交叉口的车流量。 M_i 是第 i 个交叉口的通行能力。

②信号灯的周期性分配

$$\sum_{j=1}^{m_i} G_{ij} + R_i = T_i \quad (9)$$

其中 G_{ij} 是交叉口 i 上第 j 个相位的绿灯时间, R_i 是交叉口 i 的红灯时间, T_i 是第 i 个交叉信号灯周期总时间。

5.2.3 模型的进一步优化

为了使模型更完善, 可以通过设置绿波带来保证车辆在一定速度下通过多个交叉口, 而不会频繁遇到红灯, 从而使车流的平均速度最大。

Step4: 设置绿波带, 进一步优化信号灯周期

绿波带是一种信号灯优化技术, 通过控制多个连续交叉口的信号灯相位, 使车辆在某个特定的速度下能连续通过多个绿灯, 减少不必要的停车和等待。本问可以根据车辆的平均速度和交叉口之间的距离, 设定每个信号灯的绿灯开始时间。对整个交通信号周期进行优化。建立公式如下:

目标函数:

$$\min \sum_{i=1}^{n-1} \left(\Delta t_i - \frac{d_i}{v_0} \right)^2 \quad (10)$$

Δt_i 是第 i 个交叉口与第 $i+1$ 个交叉口之间信号灯的时间偏移, d_i 表示第 i 个交叉口与第 $i+1$ 个交叉口之间的距离, v_0 表示车辆的目标通过速度 (设定的恒定速度)。

约束条件:

为了确保车辆在该时间段内遇到绿灯, 信号灯的时间偏移 Δt_i 应该满足:

$$\Delta t_i = T_i - \frac{d_i}{v_0} \quad (11)$$

$$T_i(t) = T_{low} + (T_{high} - T_{low}) \times \frac{F(t)}{F_{max}} \quad (12)$$

T_{low} ：低峰期信号灯周期时间， T_{mid} ：平峰期信号灯周期时间， T_{high} ：高峰期信号灯周期时间。根据不同日期和时间段（如工作日、周末、节假日）以及交通流量的高峰和低峰期，动态调整信号灯周期可以避免固定周期带来的效率低下。

综合分析：

$$\min(\sum_{i=1}^{n-1} (\Delta t_i - \frac{d_i}{v_0})^2 + \sum_{t=0}^{T_{day}} (T_i(t) - T_{optimal}(t))^2) \quad (13)$$

T_{day} 表示一天内的总时间， $T_{optimal}$ 表示在时间 t 下最优的信号灯周期。通过优化这两个部分可以绿波协调和信号灯周期调整，来提高整个交通系统的效率。

5.2.4 模型的建立与求解

Step5: 建立遗传算法

遗传算法是一种基于自然选择和遗传机制的优化算法，适用于解决复杂的组合优化问题，本问利用遗传算法通过模拟自然选择和遗传变异，在多种可能的配置中找到最优信号灯时长分配。根据题意建立如下公式：

不同方向的同行能力：

$$A_{i,j}(t) = k_{i,j} \times G_{i,j}(t) \quad (14)$$

$k_{i,j}$ 是与车流量相关的比例常数，用于表示绿灯时长对车流通行能力的影响， $G_{i,j}(t)$ 表示绿灯时长。

适应度函数：

$$f(G) = \sum_{i=1}^N \sum_{j=1}^4 A_{i,j}(t) = \sum_{i=1}^N \sum_{j=1}^4 k_{i,j} \times G_{i,j}(t) \quad (15)$$

在给定的周期 C_i 内，调整绿灯时长 $G_{i,j}(t)$ 使得适应度函数 $f(G)$ 最大化。

约束条件：

$$\sum_{j=1}^4 G_{i,j}(t) \leq C_i \quad (16)$$

每个交叉口各个方向的绿灯时长之和不能超过总周期。

Step6: 运用遗传算法求解

采用遗传算法计算出每个路口的最优绿灯时长及周期,得到交叉路口部分时间最优信号灯时长及周期,如下:(具体详见支撑材料三)

表 5-31 环东路-纬中路

方向	时间	族	时期	信号灯周期	绿灯时长
1	7	2	高峰期	150	19
2	7	2	高峰期	150	19
3	7	2	高峰期	150	47
4	7	2	高峰期	150	62
1	0	1	低峰期	60	11
2	0	1	低峰期	60	5
3	0	1	低峰期	60	14
4	0	1	低峰期	60	29
1	0	1	低峰期	60	11
2	0	1	低峰期	60	5
3	0	1	低峰期	60	14
4	0	1	低峰期	60	29

表 5-32 环西路-纬中路

方向	小时	族	时期	信号灯周期	绿灯时长
1	7	2	高峰期	150	45
2	7	2	高峰期	150	38
3	7	2	高峰期	150	31
4	7	2	高峰期	150	35
1	0	1	低峰期	60	19
2	0	1	低峰期	60	16
3	0	1	低峰期	60	11
4	0	1	低峰期	60	12
1	9	0	平峰期	90	28
2	9	0	平峰期	90	24
3	9	0	平峰期	90	17
4	9	0	平峰期	90	18

通过综合考虑平均车速、各交叉路口的信号周期、等待时间以及路口间距离,计算出各个交叉路口的平均车速,然后推导出整体道路的最优平均速度。如下所示:

纬中路-环西路的平均车辆速度为: 11.2
纬中路-经二路的平均车辆速度为: 19.5
纬中路-经三路的平均车辆速度为: 18.1
纬中路-经中路的平均车辆速度为: 13.4
纬中路-景区出入口的平均车辆速度为: 9.8
纬中路-经四路的平均车辆速度为: 10.3
纬中路-经五路的平均车辆速度为: 11.5
纬中路-环东路的平均车辆速度为: 20.4
经中路-纬一路的平均车辆速度为: 25.0
经中路-环北路的平均车辆速度为: 12.5
经中路-环南路的平均车辆速度为: 7.0
纬中路-经一路的平均车辆速度为: 12.3

主路最优平均速度14.0 m/s
主路最优平均速度50.4 km/h

图 5-33 经中路和纬中路的最优平均速度

计算得到经中路和纬中路这两条路口的车辆最优平均速度约为 50.4km/h。

5.2.5 模型的检验

Step7: 运用仿真软件 VISSIM 对优化后的信号灯进行仿真

利用 VISSIM 对现实路段的高仿真功能,其模拟仿真网址为 <https://wwto.lan-zouj.com/iFBcd29ibgna>,模拟出包括仿真真路段及路况、交通流的特征、驾驶人

路径行为，以此验证上述模型的性能与效果。

由于时间有限，且路口较多，因此本问只截取了其中两个路口的仿真模拟情况，分别是经一路-纬中路路口和经二路-纬中路路口。设置红绿灯，模拟车辆行驶情况如下：



图 5-34 仿真模拟路况、交通流

如上图，红色与绿色的矩形长条分别模拟红灯与绿灯的信号灯周期，当矩形长条为红色时，该方向的所有车辆都停止运行，当矩形长条颜色变成绿色时，所有车辆开始运行。

经一路-纬中路路口和经二路-纬中路路口之间的距离为 340m，在第一个路口监测的时间为 100s，在第二个路口监测到的时间为 124s，如下图：



图 5-35 两个路口的时间仿真模拟图

计算得到车辆的行驶速度为 50.98km/h。

5.2.6 误差分析与结论

通过仿真软件 VISSIM 对各个路口进行仿真验证，代入优化后的红绿灯周期和绿灯时间，记录车辆在通过不同路口的时间，获得在本路段中行驶时间，已知的两路口之间的距离，直接计算出车辆的行驶速度，得出行驶速度为 50.98km/h，而我们的理论平均最大速度约为 50.4km/h，两者误差非常小，仅只有 1.14%，仿真模型很好的验证了理论模型的正确性及合理性。

5.3 问题 3 的模型建立与求解

5.3.1 题目信息分析

本题需要分析五一黄金周期间的数据，对停车位的巡游车辆进行识别，并估算出景区所需的临时停车位数量。具体解决思路如下：

1、识别巡游车辆：根据车辆的行驶轨迹数据，识别出反复绕行且行驶速度低于某一阈值（例如低于 20km/h），则判定为巡游车辆。

2、估算停车需求：根据巡游车辆数量、巡游时间段和停车位利用率，估算出临时停车位的需求量，可使用排队论模型进行估算。

5.3.2 数据预处理

首先对附件 2 的数据进行筛选，筛选出五一黄金周时间段的数据，去除无车牌的数据，和时间异常的数据。

Step1: 根据行驶路径、车辆速度以及出现频率来对巡游车辆进行识别

(1) 对车辆进行划分，构建车辆的行车轨迹。由于数据过多，本问只截取部分车辆的行车轨迹，详见支撑材料四。

如下：

表 5-36 部分车辆行车轨迹

车牌号	开始时间	结束时间	行车轨迹
3B07ADI	29:55.0	40:53.0	经中路-环南路 → 经中路-环南路
3B445GY	33:33.0	07:16.8	环西路-纬中路 → 环西路-纬中路
3B4C1DB	28:17.1	30:25.9	经五路-纬中路 → 纬中路-景区出入口 → 经五路-纬中路
3BB63KF	21:31.1	11:19.5	经中路-环北路 → 经中路-环北路

如上表，车牌号为 3B4C1DB 的车辆从经五路-纬中路经过纬中路-景区出入口，最后又回到经五路-纬中路，且只有 2 分钟的间隔时间，时间间隔较小，则有理由为巡游车辆。

(2) 计算车辆速度

已知时间与行车轨迹，计算车辆速度，对数据进行分析，去除其中的空值。
车辆速度=车辆走过的距离/车辆走过这段距离所用的时间，详见支撑材料五。

表 5-37 部分车辆速度

车牌号	开始时间	结束时间	行车轨迹	距离 (m)	行驶时间 (秒)	速度 (m/s)
3B3CG9S	51:29.0	39:03.0	经五路-纬中路 → 经中路-纬中路 → 经中路-环南路	2230	10053.983	0.22
3B3SD7E	41:50.1	43:47.9	环西路-纬中路 → 经一路-纬中路	460	117.811	3.9
3B4C1DB	28:17.1	30:25.9	经五路-纬中路 → 纬中路-景区出入口 → 经五路-纬中路	1980	36128.872	0.05
3B95N8S	48:30.9	30:05.2	经一路-纬中路 → 环西路-纬中路	460	6094.295	0.08
3B96B7P	55:15.5	08:20.1	环西路-纬中路 → 经中路-环北路	2690	784.571	3.43

得到各车辆的速度后，需要设定一个阈值，求在不同速度出现的频率，得到下图：

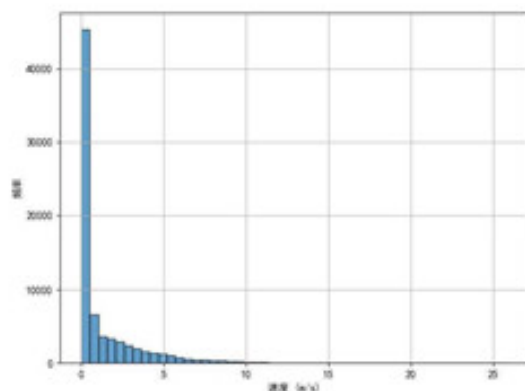


图 5-38 车辆速度频率

从速度的分布直方图中可以看到，大多数车辆的速度集中在较低的范围，尤其是接近 0 的部分。这表明部分车辆可能是在低速行驶，可能是在寻找停车位。因此，需要设计一个 0.5m/s 的阈值，低于 0.5m/s 的车辆可能被认为是在寻找停车位的巡游车辆。

Step2: 设置阈值，得到巡游车辆

(3) 以 0.5m/s 作为速度阈值，筛选出低于该速度的巡游车辆，然后分析低速车辆的行车轨迹，寻找在短时间内经过多个相邻交叉口或者重复经过同一路段的情况。

综合以上多个方面，进行判断得出巡游车辆，有车牌号为 3B3CT78、3B4BE2C、3B4C1DB、3B5454X、3B5540X 等车辆，详见支撑材料六。

5.3.3 估计停车的需求

Step3: 建立排队论模型

排队论模型用于分析和优化排队系统中服务对象和服务设施的关系，本问使用排队论模型的 m/m/c 模型，依据题意建立以下模型：

$$p = \frac{\lambda}{c \cdot \mu} \quad (17)$$

其中 λ 是到达率， μ 是服务率， c 是停车位的数量。根据 m/m/c 模型，可以估算停车场的平均等待时间和队列长度。

Erlang C 公式计算：

$$p(0) = \left(\sum_{k=0}^{c-1} \frac{(c \cdot p)^k}{k} + \frac{(c \cdot p)^c}{k \cdot (1-p)} \right)^{-1} \quad (18)$$

其中 $P(0)$ 为系统中无车的概率， k 表示 c 的阶乘， $p(0)$ 越高，意味着停车场空闲的可能性越大，说明停车场的利用率较低；反之， $p(0)$ 越低，停车场越接近满负荷运转。

排队等待车位的车辆数：

$$L_q = \frac{L_q}{\lambda} \quad (19)$$

L_q 过高，说明停车场当前的车位数量不足，可能需要增加临时停车位来应对高峰时段的需求。

每辆车的平均时间：

$$W_q = \frac{L_q}{\lambda} \quad (20)$$

W_q 反映了停车场是否能够有效处理高峰期的车辆流入，当 W_q 过长说明停车场无法及时提供车位。

5.3.4 模型的求解

结合上述模型，使用 python 软件估算停车需求，求解流程如下：

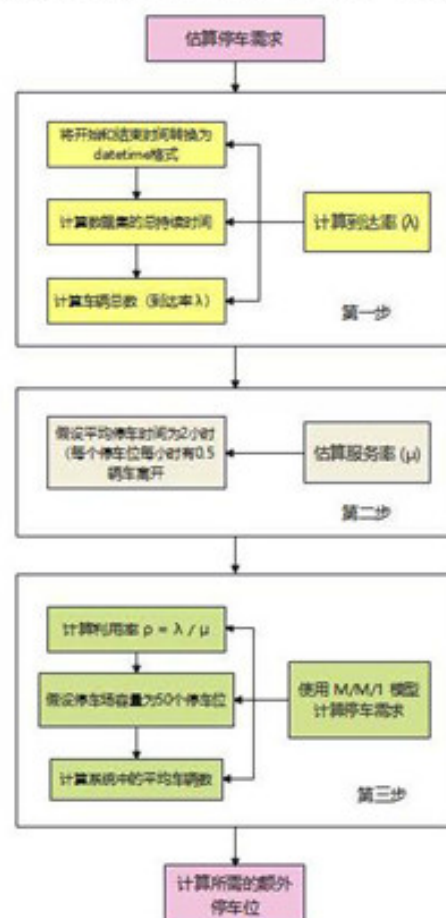


图 5-39 问题三求解流程图

5.3.5 结果展示

代码运行得到的结果为：车辆总数：32832；到达率：0.91；服务率：0.73；利用率：0.78，估算得到景区所需的临时停车位数量范围为[430,450]辆。

5.4 问题 4 的模型建立与求解

5.4.1 题目信息分析

本问要求评估五一期间和非五一期间临时管控措施在两条主路上的效果，需选择评价指标：车流量、绕行车辆和通行方向，比较管控措施前后这些指标的变化，进而评价管控措施的效果。具体分析思路如下：

1、分析车流量：对比“五一”黄金周（5月1日至5月5日）期间与非管控期间交通流量变化，尤其是关注两条主路（纬中路和经中路）上车辆进出流量的变化。

2、分析车辆绕行：由于在平时，这些车辆往往为了寻找停车位而低速绕圈，影响通行效率，管控措施是否有效减少了这些低速车辆的数量，可以通过分析车辆重复出现的数据来验证。

3、分析车辆通行方向：通过分析车辆的行驶方向数据，验证这些路线是否按照指示执行。若大部分车辆能够按照橙色/绿色箭头指示的路线进入和离开景区，说明管控措施取得了一定效果。

5.4.2 数据预处理

整理“五一”期间（5月1日至5月5日）和之前（4月1日至4月30日）的车辆数据。按时间分段整理车辆通过不同路段的情况。处理车辆通过方向、时间、车牌号等信息，剔除无关数据。如下表（详见支撑材料七）：

表 5-40 数据处理后的车辆信息

方向	时间	车牌号	交叉口
3	39:08.6	AF5B7CEM	环西路-纬中路
1	45:32.3	BK2IA84	环西路-纬中路
3	09:04.1	AF4EC7FK	环西路-纬中路
2	49:03.7	AF4MBB6	环西路-纬中路
3	47:49.4	CBA7KCG	环西路-纬中路
2	19:15.9	AFB9C06	环西路-纬中路
1	30:49.7	AF25DA6M	环西路-纬中路
4	43:28.6	AF8CB6CM	环西路-纬中路
3	39:19.7	AFU4CWB	环西路-纬中路

1	07:41.4	AF04AAE	环西路-纬中路
1	10:19.1	AF8Y4AB	环西路-纬中路
3	56:01.8	AF9FB66	环西路-纬中路

5.4.3 对比各项指标

(1) 分析车流量密度

通过对比“五一”黄金周与之前的流量数据，分析红色管控路段的车辆通行情况，是否有减少或管控效果。通过时间段划分（如早高峰、午间、晚高峰），对比流量高峰时段的变化，分析管控措施对不同时间段的影响。因此需要建立流量密度公式与流量密度百分比公式，如下：

流量密度公式：

$$p_i = \frac{Q_i}{t_i} \quad (21)$$

其中 p 表示 i 时期流量密度， Q_i 表示 i 时期总车流量， t_i 表示 i 时期总时间。

流量密度百分比：

$$M = \frac{p_i - p_j}{p_i} \quad (22)$$

其中 p_i 表示四月的流量密度， p_j 表示五月的流量密度。 M 可以直观的观测到五月份和四月份的车流量差别。

结合上述公式，运用 python 计算出 12 个交叉路口的车流量密度与流量密度百分比，如下表：

表 5-41 12 个路口的车流量密度

交叉口	流量计数_五一	流量计数_四月	流量密度_五一	流量密度_四月	流量密度变化百分比
环东路-纬中路	7269	111653	1453.8	3721.766667	-60.93790583
环西路-纬中路	251798	1608751	50359.6	53625.03333	-6.089382384
纬中路-景区出入口	87399	389593	17479.8	12986.43333	34.60046767
经一路-纬中路	87314	590170	17462.8	19672.33333	-11.23167901
经三路-纬中路	87999	599361	17599.8	19978.7	-11.90718115
经中路-环北路	103734	612139	20746.8	20404.63333	1.676906716
经中路-环南路	153170	894463	30634	29815.43333	2.74544615
经中路-纬一路	121312	529828	24262.4	17660.93333	37.37892297
经中路-纬中路	149346	858232	29869.2	28607.73333	4.40953029
经二路-纬中路	49754	353700	9950.8	11790	-15.59966073
经五路-纬中路	34142	143313	6828.4	4777.1	42.94027757
经四路-纬中路	7905	56603	1581	1886.766667	-16.20585481

将上表进行可视化，如下：



图 5-42 12 个路口的车流量密度可视化图

由上图可知，各时间段景区出入口的流量密度都较大，两时期流量密度曲线趋于重合，五一节假日景区人口会激增，造成流量密度增大，但是图中在两个时期相近说明临时管控措施起到在交通管理很大作用。

(2) 分析绕行车辆

检查在特定路段上，是否有车辆多次出现来回低速行驶的情况，以此判断车辆绕行问题是否缓解。如下（详见支撑材料八）：

图 5-43 五一期间绕行车辆数

车牌号	交叉口	出现次数
3B178Q8	经中路-环北路	3
3B1CB0F	纬中路-景区出入口	3
3B1CB0F	经中路-纬中路	3
3B5454X	经三路-纬中路	3
3B5454X	经中路-纬中路	3
3B5540X	经中路-纬一路	3
3B56PBV	纬中路-景区出入口	4
3B56PBV	经中路-纬中路	4
3B846U6	纬中路-景区出入口	4
3B846U6	经中路-纬一路	4
3B846U6	经中路-纬中路	4
3BAA68W	环西路-纬中路	3
3BAB03C	环西路-纬中路	6

如上表，车牌号为 3BAB03C 的车辆在环西路-纬中路路口出现次数最多，由此可见该车存在绕行现象。

(3) 分析车辆通行方向

按照附件 3 提供的车辆进出方向（橙色和绿色箭头），分析是否大部分车辆

按照指定方向通行，观察管控是否有效引导了车流。

使用 python 对各个路口的车流量进行数据处理，首先汇总每个路口的各个方向车流量，然后根据日期将数据按天分类，计算每天各个路口的车流量方向分布。接着，对这些汇总后的数据进行分析，最后进行评价临时管控措施的效果，不同时期的部分日通行方向如下：

图 5-44 车辆各通行方向的次数

交叉口	方向	方向计数	日均方向计数	交叉口	方向	方向计数	日均方向计数
环东路-纬中路	1	16329	544	环东路-纬中路	1	16329	544
环东路-纬中路	2	14346	478	环东路-纬中路	2	14346	478
环东路-纬中路	3	32824	1094	环东路-纬中路	3	32824	1094
环东路-纬中路	4	48154	1605	环东路-纬中路	4	48154	1605
环西路-纬中路	1	515228	17174	环西路-纬中路	1	515228	17174
环西路-纬中路	2	430390	14346	环西路-纬中路	2	430390	14346
环西路-纬中路	3	319588	10652	环西路-纬中路	3	319588	10652
环西路-纬中路	4	343545	11451	环西路-纬中路	4	343545	11451
纬中路-景区出入口	1	240293	8009	纬中路-景区出入口	1	240293	8009

由上表可知，大部分车辆能够按照橙色/绿色箭头指示的路线进入和离开景区，说明管控措施取得了一定效果。

5.4.4 结论分析

1、车辆流量变化分析：通过对比五一期间与非五一期间的车流量数据，发现红色管控路段的车流量有所减少。这表明五一期间实施的临时管控措施取得了显著的效果，成功减少了该路段的交通压力。

2、绕行车辆情况：综合分析五一期间与非五一期间的绕行车辆数据，结果显示五一期间的绕行车辆数量有所增加，反映出临时管控措施在有效引导车辆绕行方面发挥了更好的作用。

3、车辆通行方向的合规性：通过比较五一期间与非五一期间某站点的车辆通行频次，发现五一期间的通行次数低于非五一期间。这说明临时管控措施对规范车辆通行方向的效果更为明显，确保了车流按预期方向运行，有效提升了交通秩序。

整体上来看，五一期间的临时管控措施在减少车流、优化车辆绕行以及规范通行方向方面都展现了较为出色的成效。

六、模型的评价

6.1 模型的优点

(1) 本文所使用的 K-均值算法可以将车流量划分为不同类别,不需要手动设定时间段,并且该算法计算快,适合处理大规模的交通数据,并且可快速收敛得到稳定的结果。

(2) 排队论模型相对简单易行,可适合快速评估临时停车需求,并且此模型适合动态的交通场景,可得出一个合理的停车位需求评估。

(3) 对于问题四,本文在理论模型的基础上,通过仿真软件 VISSIM 进一步对理论模型进行了验证,从理论和实际两个方面证明了模型的合理性。

6.2 模型的缺点

(1) 本文所使用的 K-均值算法没有考虑车流量的时间连续性,只关注数据的相似点。

(2) 排队论模型难以应对极端的情况,例如发生交通事故。

6.3 模型的改进

由于在本次比赛中时间有限,且路口较多,因此在使用 VISSIM 软件进行模型的仿真模拟检验时,没有模拟所有的交叉路口,只对其中两个路口进行模拟仿真,后期会将本仿真模拟进行进一步地完善。

6.4 模型的推广

本题使用的 k-均值算法和模糊数学在交通管制上面有广泛应用,还在资源调度或城市资源管理上面有广泛应用。

七、参考文献

- [1]陈军舰,刘春生,王晓晗,等.多源数据融合的交通走廊交通流量分析[J].天津建设科技,2024,34(04):1-4.
- [2]李国庆.基于 K-mean 聚类算法的电力营销数据分析[J].电子技术, 2023,52
- [3]高萌悦,马晓旦.基于 VISSIM 仿真的信号交叉口优化研究[J].物流科技, 2024 ,47(09):98-101.DOI:10.13714/j.cnki.
- [4] 吴场建,曹奇,任刚.考虑路径关系的干线多路径绿波优化模型 [J]. 交通运输系统工程与信息 2024, 24 (03) : 103-113+163.DOI:10.16097/j.cnki.

八、附录

附录一

介绍：支撑材料的文件列表

支撑材料一

工作日、非工作日与节假日的划分

支撑材料二

高峰期、低峰期与平峰期的划分

支撑材料三

各路口最优信号灯时长及周期

支撑材料四

车辆的行车轨迹

支撑材料五

车辆的行驶速度

支撑材料六

设置阈值筛选得到的所有巡游车辆

支撑材料七

数据预处理后的车辆信息

支撑材料八

五一期间绕行车辆数

附录二

介绍：问题一数据预处理的代码

①删除无车牌数据

```
import pandas as pd
# 读取数据
data = pd.read_csv('../附件 2.csv', encoding='gbk')
# 删除车牌号为 "无车牌" 的数据
data = data[data['车牌号'] != '无车牌']
# 保存为新文件
data.to_csv('无车牌数据的附件 2.csv', index=False)
```

②筛选经中路-纬中路的数据

```
import pandas as pd
data = pd.read_csv('../附件 2.csv', encoding='gbk')
# 删除车牌号为 "无车牌" 的数据
data = data[data['车牌号'] != '无车牌']
# 筛选出交叉口为 '经中路-纬中路' 的数据
data = data[data['交叉口'] == '经中路-纬中路']
# 重置索引
data = data.reset_index(drop=True)
print(data)
data.to_csv('2.经中路-纬中路.csv', index=False)
```

③分工作日和节假日

```
import pandas as pd
import holidays # 需要先安装 holidays 库
# 读取数据
data = pd.read_csv('2.经中路-纬中路.csv')
# 假设时间列为 '时间', 如果格式不同请调整
data['时间'] = pd.to_datetime(data['时间'])
# 使用 holidays 库来定义中国的节假日
cn_holidays = holidays.China()
# 添加是否为节假日的列
data['是否为中国节假日'] = data['时间'].dt.date.apply(lambda x: 1 if x in cn_holidays else 0)
# 添加是否为工作日的列 (工作日为 1, 周末为 0)
data['是否为工作日'] = data['时间'].dt.weekday.apply(lambda x: 1 if x < 5 else 0)
data.to_csv('3.节假日工作日.csv')
```

④将数据按工作日、非工作日与节假日进行区分

```
import pandas as pd
file_path = '3.节假日工作日.csv'
data = pd.read_csv(file_path)
workday_data = data[(data['是否为工作日'] == 1) & (data['是否为中国节假日'] ==
```

```

0)] # 工作日数据
non_workday_data = data[data['是否为工作日'] == 0] # 非工作日数据
holiday_data = data[data['是否为中国节假日'] == 1] # 节假日数据
workday_file_path = '4.工作日数据.xlsx'
non_workday_file_path = '4.非工作日数据.xlsx'
holiday_file_path = '4.节假日数据.xlsx'
workday_data.to_excel(workday_file_path, index=False)
non_workday_data.to_excel(non_workday_file_path, index=False)
holiday_data.to_excel(holiday_file_path, index=False)

```

⑤统计不同时段的车流量

```

import pandas as pd
# new_list = ['5.工作日不同时段车流量.csv', '5.非工作日不同时段车流量.csv', '5.节假日不同时段车流量.csv']
workday_data = pd.read_csv('4.工作日数据.csv')
non_work_data = pd.read_csv('4.非工作日数据.csv')
holiday_data = pd.read_csv('4.节假日数据.csv')
# 将 '时间' 列转换为日期时间格式以便于处理
workday_data['时间'] = pd.to_datetime(workday_data['时间'])
non_work_data['时间'] = pd.to_datetime(non_work_data['时间'])
holiday_data['时间'] = pd.to_datetime(holiday_data['时间'])
# 从时间戳中提取小时
workday_data['hour'] = workday_data['时间'].dt.hour
non_work_data['hour'] = non_work_data['时间'].dt.hour
holiday_data['hour'] = holiday_data['时间'].dt.hour
# 按小时和方向分组，统计每小时各个方向的车辆数量
workday_data = workday_data.groupby(['hour', '方向']).size().unstack(fill_value=0)
# workday_data = workday_data.groupby(['hour', '方向'])['车牌号'].nunique()
non_work_data = non_work_data.groupby(['hour', '方向']).size().unstack(fill_value=0)
holiday_data = holiday_data.groupby(['hour', '方向']).size().unstack(fill_value=0)
print(workday_data)
print(non_work_data)
print(holiday_data)
# workday_data.to_csv('5.工作日不同时段车流量.csv')
# non_work_data.to_csv('5.非工作日不同时段车流量.csv')
# holiday_data.to_csv('5.节假日不同时段车流量.csv')

```

⑥将不同时段的车流量可视化

```

import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置显示中文字体
# 另外，由于字体更改以后会导致坐标轴中部分字符无法正常显示，这是需要更改 axes.unicode_minus 参数。

```

```

plt.rcParams['axes.unicode_minus'] = False # 设置正常显示符号
file_paths = [
    '5.非工作日不同时段车流量.csv',
    '5.工作日不同时段车流量.csv',
    '5.节假日不同时段车流量.csv']
# Read the data
data_non_workday = pd.read_csv(file_paths[0])
data_workday = pd.read_csv(file_paths[1])
data_holiday = pd.read_csv(file_paths[2])
# Create a list of datasets and titles for each category
datasets = [data_non_workday, data_workday, data_holiday]
titles = ['非工作日不同时段车流量', '工作日不同时段车流量', '节假日不同时段车流量']
# Plot the data for each dataset
for i, data in enumerate(datasets):
    plt.figure(figsize=(10, 6))
    for col in data.columns[1:]:
        plt.plot(data['hour'], data[col], marker='o', label=f'方向 {col}')
    plt.title(titles[i], fontsize=16)
    plt.xlabel('小时', fontsize=12)
    plt.ylabel('车流量', fontsize=12)
    plt.xticks(data['hour'])
    plt.grid(True)
    plt.legend(title='方向')
    # Show each plot
    plt.tight_layout()
    plt.show()

```

附录三

介绍：问题一模型的求解代码

①根据车流量划分早晚高峰

```

import pandas as pd
from sklearn.cluster import KMeans
import numpy as np
# 加载数据
file_path = '5.工作日不同时段车流量.csv'
traffic_data = pd.read_csv(file_path)
# 计算每小时的总车流量
traffic_data['total_traffic'] = traffic_data[['1', '2', '3', '4']].sum(axis=1)
# 使用 K-均值聚类算法分析数据
X = traffic_data[['hour', 'total_traffic']].values
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)

```

```

traffic_data['cluster'] = kmeans.labels_
# 根据聚类结果定义时间段
cluster_label_to_period = {
    0: "高峰期",
    1: "低峰期",
    2: "平峰期"
}# 根据实际数据调整聚类标签映射至时间段描述
traffic_data['time_period'] = traffic_data['cluster'].map(cluster_label_to_period)
# 排序和显示结果
clustered_traffic = traffic_data.sort_values(by=['cluster', 'hour'])
print(clustered_traffic)
clustered_traffic.drop(columns='total_traffic', inplace=True)
clustered_traffic.to_csv('6.工作日时期划分.csv')
## 加载数据
# file_path = '5.节假日不同时段车流量.csv'
# traffic_data = pd.read_csv(file_path)
### 计算每小时的总车流量
# traffic_data['total_traffic'] = traffic_data[['1', '2', '3', '4']].sum(axis=1)
### 使用 K-均值聚类算法分析数据
# X = traffic_data[['hour', 'total_traffic']].values
# kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
# traffic_data['cluster'] = kmeans.labels_
### 根据聚类结果定义时间段
# cluster_label_to_period = {
#     0: "高峰期",
#     1: "平峰期",
#     2: "低峰期"
# }### 根据实际数据调整聚类标签映射至时间段描述
# traffic_data['time_period'] = traffic_data['cluster'].map(cluster_label_to_period)
### 排序和显示结果
# clustered_traffic = traffic_data.sort_values(by=['cluster', 'hour'])
# print(clustered_traffic)
# clustered_traffic.drop(columns='total_traffic', inplace=True)
# clustered_traffic.to_csv('6.节假日时期划分.csv')
## 加载数据
# file_path = '5.非工作日不同时段车流量.csv'
# traffic_data = pd.read_csv(file_path)
### 计算每小时的总车流量
# traffic_data['total_traffic'] = traffic_data[['1', '2', '3', '4']].sum(axis=1)
### 使用 K-均值聚类算法分析数据
# X = traffic_data[['hour', 'total_traffic']].values
# kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
# traffic_data['cluster'] = kmeans.labels_
### 根据聚类结果定义时间段

```

```
# cluster_label_to_period = {
#     0: "低峰期",
#     1: "高峰期",
#     2: "平峰期"
# }## # 根据实际数据调整聚类标签映射至时间段描述
# traffic_data['time_period'] = traffic_data['cluster'].map(cluster_label_to_period)
## # 排序和显示结果
# clustered_traffic = traffic_data.sort_values(by=['cluster', 'hour'])
# print(clustered_traffic)
# clustered_traffic.drop(columns='total_traffic', inplace=True)
# clustered_traffic.to_csv('6.非工作日时期划分.csv')
```

②不同时期的可视化

```
# Here's the complete code to load the data and visualize it as line charts:
import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置显示中文字体
# 另外，由于字体更改以后会导致坐标轴中部分字符无法正常显示，这是需要更改 axes.unicode_minus 参数。
plt.rcParams['axes.unicode_minus'] = False # 设置正常显示符号
# Load the three CSV files for visualization
file_paths_period = [
    '7.节假日不同时期车流量.csv',
    '7.工作日不同时期车流量.csv',
    '7.非工作日不同时期车流量.csv'
]# Read the data from these files
data_holiday_period = pd.read_csv(file_paths_period[0])
data_workday_period = pd.read_csv(file_paths_period[1])
data_non_workday_period = pd.read_csv(file_paths_period[2])
# Create a list of datasets and titles for each category
datasets_period = [data_holiday_period, data_workday_period,
data_non_workday_period]
titles_period = ['节假日车流量', '工作日车流量', '非工作日车流量']
# Plot all the data in a single combined plot
plt.figure(figsize=(12, 8))
# Plot data from all three datasets
for i, data in enumerate(datasets_period):
    for index, row in data.iterrows():
        print(index, row)
        plt.plot(data.columns[1:], row[1:], marker='o', label=f'{titles_period[i]} -
{row["time_period"]}')
plt.title('节假日、工作日、非工作日的车流量汇总', fontsize=16)
plt.xlabel('方向', fontsize=12)
plt.ylabel('车流量', fontsize=12)
```

```
plt.grid(True)
plt.legend(title='时段及日期类型', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.tight_layout()
plt.show()
```

③不同时期车流量的统计

```
import pandas as pd
file_path = "6.工作日时期划分.csv"
data = pd.read_csv(file_path)
traffic_summary = data.groupby('time_period')[['1', '2', '3', '4']].sum()
print(traffic_summary)
traffic_summary.to_csv('7.工作日不同时期车流量.csv')
file_path1 = "6.非工作日时期划分.csv"
data1 = pd.read_csv(file_path1)
traffic_summary1 = data1.groupby('time_period')[['1', '2', '3', '4']].sum()
print(traffic_summary1)
traffic_summary1.to_csv('7.非工作日不同时期车流量.csv')
file_path2 = "6.节假日时期划分.csv"
data2 = pd.read_csv(file_path2)
traffic_summary2 = data2.groupby('time_period')[['1', '2', '3', '4']].sum()
print(traffic_summary2)
traffic_summary2.to_csv('7.节假日不同时期车流量.csv')
```

④估计工作日各相位车流量

```
import pandas as pd
file_path = "7.工作日不同时期车流量.csv"
data = pd.read_csv(file_path)
# Define a function to apply the percentage splits based on time_period
def split_traffic(row):
    if row['time_period'] == '高峰期':
        straight_ratio, left_ratio, right_ratio = 0.8, 0.1, 0.1
    elif row['time_period'] == '平峰期':
        straight_ratio, left_ratio, right_ratio = 0.7, 0.2, 0.1
    elif row['time_period'] == '低峰期':
        straight_ratio, left_ratio, right_ratio = 0.6, 0.2, 0.2
    # Calculate straight, left, and right turns for each direction
    for direction in ['1', '2', '3', '4']:
        row[f'{direction}_直行'] = row[direction] * straight_ratio
        row[f'{direction}_左转'] = row[direction] * left_ratio
        row[f'{direction}_右转'] = row[direction] * right_ratio
    return row
# Apply the function to each row
new_data = data.apply(split_traffic, axis=1)
# Select and display relevant columns for verification
```

```

traffic_split_columns = ['time_period'] + [f'{direction}_{turn}' for direction in ['1', '2', '3', '4'] for turn in
                                                                    ['直行', '左转', '右转']]
split_traffic_data = new_data[traffic_split_columns]

split_traffic_data = split_traffic_data.round(0).astype(int, errors='ignore')
print(split_traffic_data)
split_traffic_data.to_csv('8.工作日各相位车流量.csv')

```

⑤估计非工作日各相位车流量

```

import pandas as pd
file_path = "7.非工作日不同时期车流量.csv"
data = pd.read_csv(file_path)
# Define a function to apply the percentage splits based on time_period
def split_traffic(row):
    if row['time_period'] == '高峰期':
        straight_ratio, left_ratio, right_ratio = 0.8, 0.1, 0.1
    elif row['time_period'] == '平峰期':
        straight_ratio, left_ratio, right_ratio = 0.7, 0.2, 0.1
    elif row['time_period'] == '低峰期':
        straight_ratio, left_ratio, right_ratio = 0.6, 0.2, 0.2
    # Calculate straight, left, and right turns for each direction
    for direction in ['1', '2', '3', '4']:
        row[f'{direction}_直行'] = row[direction] * straight_ratio
        row[f'{direction}_左转'] = row[direction] * left_ratio
        row[f'{direction}_右转'] = row[direction] * right_ratio
    return row
# Apply the function to each row
new_data = data.apply(split_traffic, axis=1)
# Select and display relevant columns for verification
traffic_split_columns = ['time_period'] + [f'{direction}_{turn}' for direction in ['1', '2', '3', '4'] for turn in
                                                                    ['直行', '左转', '右转']]
split_traffic_data = new_data[traffic_split_columns]
split_traffic_data = split_traffic_data.round(0).astype(int, errors='ignore')
print(split_traffic_data)
split_traffic_data.to_csv('8.非工作日各相位车流量.csv')

```

⑥估计节假日各相位车流量

```

import pandas as pd
file_path = "7.节假日不同时期车流量.csv"
data = pd.read_csv(file_path)
# Define a function to apply the percentage splits based on time_period
def split_traffic(row):

```

```

if row['time_period'] == '高峰期':
    straight_ratio, left_ratio, right_ratio = 0.8, 0.1, 0.1
elif row['time_period'] == '平峰期':
    straight_ratio, left_ratio, right_ratio = 0.7, 0.2, 0.1
elif row['time_period'] == '低峰期':
    straight_ratio, left_ratio, right_ratio = 0.6, 0.2, 0.2
# Calculate straight, left, and right turns for each direction
for direction in ['1', '2', '3', '4']:
    row[f'{direction}_直行'] = row[direction] * straight_ratio
    row[f'{direction}_左转'] = row[direction] * left_ratio
    row[f'{direction}_右转'] = row[direction] * right_ratio
return row
# Apply the function to each row
new_data = data.apply(split_traffic, axis=1)
# Select and display relevant columns for verification
traffic_split_columns = ['time_period'] + [f'{direction}_{turn}' for direction in ['1', '2', '3', '4'] for turn in
                                                                    ['直行', '左转', '右转']]
split_traffic_data = new_data[traffic_split_columns]
split_traffic_data = split_traffic_data.round(0).astype(int, errors='ignore')
print(split_traffic_data)
split_traffic_data.to_csv('8.节假日各相位车流量.csv')

```

⑦相位可视化

```

from pyecharts.charts import Line
from pyecharts import options as opts
import pandas as pd
# 加载数据
holiday_data = pd.read_csv('8.节假日各相位车流量.csv')
workday_data = pd.read_csv('8.工作日各相位车流量.csv')
non_workday_data = pd.read_csv('8.非工作日各相位车流量.csv')
# 定义绘制 pyecharts 折线图的函数
def create_pycharts_line_chart(data, title):
    time_periods = data['time_period'].tolist()
    directions = ['1_直行', '1_左转', '1_右转', '2_直行', '2_左转', '2_右转', '3_直行', '3_左转', '3_右转', '4_直行', '4_左转', '4_右转']
    line = Line()
    line.add_xaxis(time_periods)
    for direction in directions:
        line.add_yaxis(direction, data[direction].tolist(), is_smooth=True,
label_opts=opts.LabelOpts(is_show=False))
    line.set_global_opts(
        title_opts=opts.TitleOpts(title=title),

```

```

        xaxis_opts=opts.AxisOpts(axislabel_opts={"rotate": 45}),
        yaxis_opts=opts.AxisOpts(name="车流量"),
        tooltip_opts=opts.TooltipOpts(trigger="axis"),
        legend_opts=opts.LegendOpts(pos_left="right", item_width=0,
item_height=0) # 移除图例符号 )
    return line
# 创建折线图
line_holiday = create_pycharts_line_chart(holiday_data, '节假日各相位车流量 (折线图)')
line_workday = create_pycharts_line_chart(workday_data, '工作日各相位车流量 (折线图)')
line_non_workday = create_pycharts_line_chart(non_workday_data, '非工作日各相位车流量 (折线图)')
# 渲染图表并在浏览器中显示
line_holiday.render('holiday_traffic_flow_line.html')
line_workday.render('workday_traffic_flow_line.html')
line_non_workday.render('non_workday_traffic_flow_line.html')
print("折线图已保存为 HTML 文件，可以通过浏览器查看。")

```

附录四

介绍：问题二解题代码

①所有交叉路口的车流峰值时段

```

import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置显示中文字体
plt.rcParams['axes.unicode_minus'] = False # 设置正常显示符号
# Load the uploaded file to inspect its contents
file_path = './无车牌数据的附件 2.csv'
data_cleaned = pd.read_csv(file_path)
# 先对数据进行初步清理，删除不需要的索引列，并格式化时间列
data_cleaned['时间'] = pd.to_datetime(data_cleaned['时间'])
# 提取时间信息，以便后续按小时段统计车流量
data_cleaned['小时'] = data_cleaned['时间'].dt.hour
# 统计每个交叉口在不同小时的车流量
traffic_flow_by_hour = data_cleaned.groupby(['交叉口', '方向', '小时']).size().reset_index(name='车流量')
# 统计每个交叉口在不同小时段的总车流量
traffic_peak = data_cleaned.groupby(['交叉口', '小时']).size().reset_index(name='总车流量')
# 绘制每个交叉路口的车流高峰时段折线图并保存
unique_crossings = traffic_peak['交叉口'].unique()
for crossing in unique_crossings:
    plt.figure(figsize=(10, 6))

```

```

crossing_data = traffic_peak[traffic_peak['交叉口'] == crossing]
plt.plot(crossing_data['小时'], crossing_data['总车流量'], marker='o',
linestyle='-', label=crossing)
plt.title('交叉口 {crossing} 的车流高峰时段')
plt.xlabel('小时')
plt.ylabel('总车流量')
plt.xticks(range(0, 24)) # 设置 X 轴刻度显示小时
plt.grid(True)
# 保存图片, 文件名为图像标题
file_name = f'交叉口_{crossing}_的车流高峰时段.png'
plt.savefig(file_name)
plt.show()

```

②划分天数工作日

```

import pandas as pd
import holidays # 需要先安装 holidays 库
# 读取数据
data = pd.read_csv('../无车牌数据的附件 2.csv')
# 假设时间列为 '时间', 如果格式不同请调整
data['时间'] = pd.to_datetime(data['时间'])
# 使用 holidays 库来定义中国的节假日
cn_holidays = holidays.China()
# 添加是否为节假日的列
data['是否为中国节假日'] = data['时间'].dt.date.apply(lambda x: 1 if x in
cn_holidays else 0)
# 添加是否为工作日的列 (工作日为 1, 周末为 0)
data['是否为工作日'] = data['时间'].dt.weekday.apply(lambda x: 1 if x < 5 else 0)
data.to_csv('2.日期与节假日.csv')

```

③区分三个时间段

```

import pandas as pd
file_path = '2.日期与节假日.csv'
data = pd.read_csv(file_path)
workday_data = data[(data['是否为工作日'] == 1) & (data['是否为中国节假日'] ==
0)] # 工作日数据
non_workday_data = data[data['是否为工作日'] == 0] # 非工作日数据
holiday_data = data[data['是否为中国节假日'] == 1] # 节假日数据
workday_file_path = '2.工作日数据.csv'
non_workday_file_path = '2.非工作日数据.csv'
holiday_file_path = '2.节假日数据.csv'
workday_data.to_csv(workday_file_path)
non_workday_data.to_csv(non_workday_file_path)
holiday_data.to_csv(holiday_file_path)

```

④统计车流量

```
import os
import pandas as pd
from sklearn.cluster import KMeans
# 文件夹名称列表
name_list = ['工作日不同路段表', '节假日不同路段表', '非工作日不同路段表']
# 遍历文件夹列表
for folder_name in name_list:
    folder_path = folder_name
    # 获取文件夹下所有 CSV 文件的路径
    file_paths = [os.path.join(folder_path, file) for file in os.listdir(folder_path) if
file.endswith('.csv')]
    # 读取所有 CSV 文件到一个字典中，键为文件名，值为 DataFrame
    for file_path in file_paths:
        file_name = os.path.basename(file_path)
        df = pd.read_csv(file_path)
        # 分组并计算车流总量
        df_grouped = df.groupby(['方向', 'hour']).size().reset_index(name='车流总
量')
        # 使用 K-均值聚类算法分析数据
        X = df_grouped[['hour', '车流总量']].values
        kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
        df_grouped['cluster'] = kmeans.labels_
        # 根据聚类结果定义时间段
        cluster_label_to_period = {
            0: "低峰期",
            1: "平峰期",
            2: "高峰期" }
        # 根据实际数据调整聚类标签映射至时间段描述
        df_grouped['time_period'] =
df_grouped['cluster'].map(cluster_label_to_period)
        # 打印结果
        print(f"文件夹: {folder_name}, 文件名: {file_name}")
        print(df_grouped)
        # 保存处理后的数据到新的 CSV 文件
        output_file_name = f'processed_{file_name}'
        output_file_path = os.path.join(folder_path, output_file_name)
        df_grouped.to_csv(output_file_path, index=False, encoding='utf-8-sig')
    print("\n 所有文件处理完毕。 \n")
```

⑤分表

```
# 重新加载并处理数据
import pandas as pd
file_path = '2.非工作日数据.csv'
```

```

data_cleaned = pd.read_csv(file_path)
# 清理数据
data_cleaned['时间'] = pd.to_datetime(data_cleaned['时间'])
data_cleaned['hour'] = data_cleaned['时间'].dt.hour
# 按照交叉口分组
grouped = data_cleaned.groupby('交叉口')
# 创建并保存每个交叉口的表
for intersection, group_data in grouped:
    # 创建表格文件名
    file_name = f'非工作日---{intersection}.csv'
    # 保存为 CSV 文件
    group_data.to_csv(file_name, index=False, encoding='utf-8-sig')
## 计算每个交叉口、每个方向的车流总量
# traffic_flow_total = data_cleaned.groupby(['交叉口', '方向',
# 'hour']).size().reset_index(name='车流总量')
# print(traffic_flow_total)
### 使用 K-均值聚类算法分析数据
# X = traffic_flow_total[['hour', '车流总量']].values
# kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
# traffic_flow_total['cluster'] = kmeans.labels_
## print(traffic_flow_total)
### 根据聚类结果定义时间段
# cluster_label_to_period = {
#     0: "低峰期",
#     1: "平峰期",
#     2: "高峰期"
# }### 根据实际数据调整聚类标签映射至时间段描述
# traffic_flow_total['time_period'] =
# traffic_flow_total['cluster'].map(cluster_label_to_period)
# print(traffic_flow_total)
# traffic_flow_total.to_csv('test.csv')
## 计算每个交叉口的车流总量，以便按比例分配绿灯时长
# traffic_flow_sum_by_crossing = traffic_flow_total.groupby('交叉口')['车流总量']
# .sum().reset_index(name='总车流量')
### 合并数据，计算每个方向的绿灯时长
# traffic_flow_total = pd.merge(traffic_flow_total, traffic_flow_sum_by_crossing,
# on='交叉口')
# traffic_flow_total['绿灯时长'] = (traffic_flow_total['车流总量'] / traffic_flow_total['
# 总车流量']) * T_cycle
## print(traffic_flow_total)

```

⑥分配绿灯时间

```

import os
import pandas as pd

```

```

from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
folder_name = '绿灯时间'
# 时间段对应的信号灯周期（单位：秒）
time_period_to_cycle = {
    "低峰期": 60, # 低峰期信号灯周期为 60 秒
    "平峰期": 90, # 平峰期信号灯周期为 90 秒
    "高峰期": 150 # 高峰期信号灯周期为 150 秒}
# 获取文件夹下所有 CSV 文件的路径
file_paths = [os.path.join(folder_name, file) for file in os.listdir(folder_name) if
file.endswith('.csv')]
# 遍历文件路径
for file_path in file_paths:
    file_name = os.path.basename(file_path)
    df = pd.read_csv(file_path)
# 为每个时间段设置对应的信号灯周期
df['信号灯周期'] = df['time_period'].map(time_period_to_cycle)
# 计算绿灯时长
df['绿灯时长'] = (df['车流总量'] / df['总车流量']) * df['信号灯周期']
# 转换绿灯时长为整数
df['绿灯时长'] = df['绿灯时长'].astype(int)
# 打印结果
print(f'文件名: {file_name}')
print(df)
# 保存处理后的数据到新的 CSV 文件
output_file_name = f'绿灯时间{file_name}'
output_file_path = os.path.join(folder_name, output_file_name)
df.to_csv(output_file_path, index=False, encoding='utf-8-sig')
print("\n 所有文件处理完毕。 \n")

```

⑦优化信号灯周期

```

import os
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
folder_name = '绿灯时间'
# 时间段对应的信号灯周期（单位：秒）
time_period_to_cycle = {
    "低峰期": 60, # 低峰期信号灯周期为 60 秒
    "平峰期": 90, # 平峰期信号灯周期为 90 秒
    "高峰期": 150 # 高峰期信号灯周期为 150 秒}
# 获取文件夹下所有 CSV 文件的路径
file_paths = [os.path.join(folder_name, file) for file in os.listdir(folder_name) if
file.endswith('.csv')]

```

```

# 遍历文件路径
for file_path in file_paths:
    file_name = os.path.basename(file_path)
    df = pd.read_csv(file_path)
# 为每个时间段设置对应的信号灯周期
df['信号灯周期'] = df['time_period'].map(time_period_to_cycle)
# 计算绿灯时长
df['绿灯时长'] = (df['车流总量'] / df['总车流量']) * df['信号灯周期']
# 转换绿灯时长为整数
df['绿灯时长'] = df['绿灯时长'].astype(int)
# 打印结果
print(f'文件名: {file_name}')
print(df)
# 保存处理后的数据到新的 CSV 文件
output_file_name = f'绿灯时间{file_name}'
output_file_path = os.path.join(folder_name, output_file_name)
df.to_csv(output_file_path, index=False, encoding='utf-8-sig')
print("\n 所有文件处理完毕。 \n")

```

⑧设置绿波带

```

# 重新尝试计算绿灯时间差
# 交叉口之间的距离信息（单位：米）
distances = {
    "环北路-纬一路": 520,
    "纬一路-纬中路": 510,
    "纬中路-环南路": 710,
    "环西路-经一路": 460,
    "经一路-经二路": 340,
    "经二路-经三路": 440,
    "经三路-经中路": 420,
    "经中路-景区出入口": 530,
    "景区出入口-经四路": 560,
    "经四路-经五路": 430,
    "经五路-环东路": 270}
# 假设的平均速度 (40 km/h = 11.11 m/s)
v_avg = 11.11 # 平均速度为 40 km/h 转换为 m/s
# 计算每个交叉口之间的时间差
time_offsets_rounded = {crossing: int(distance / v_avg) for crossing, distance in
distances.items()}
# 打印每个交叉口的绿灯时间差
print(time_offsets_rounded)

```

⑨遗传算法求解

```

import numpy as np

```

```

import random
# 定义遗传算法的参数
POPULATION_SIZE = 50    # 种群大小
GENERATIONS = 100      # 迭代代数
MUTATION_RATE = 0.1     # 变异概率
CROSSOVER_RATE = 0.7    # 交叉概率
# 交通流信息
distances = {
    'NS': 500,    # 北-南方向的总距离
    'SN': 500,    # 南-北方向的总距离
    'EW': 400,    # 东-西方向的总距离
    'WE': 400     # 西-东方向的总距离
}
)# 假设的信号灯周期
T_cycle = 120
# 适应度函数: 计算平均车速, 适应度越大, 配置越优
def fitness(T_green, distances):
    total_time = 0
    for direction in distances:
        travel_time = distances[direction] / v_avg
        wait_time = T_cycle - T_green[direction]
        total_time += travel_time + wait_time
    total_distance = sum(distances.values())
    avg_speed = total_distance / total_time
    return avg_speed
speed = 48
# 初始化种群: 生成随机的信号灯时长配置
def initialize_population():
    population = []
    for _ in range(POPULATION_SIZE):
        T_green = {
            'NS': random.randint(20, 60),    # 随机绿
            'SN': random.randint(20, 60),
            'EW': random.randint(20, 60),
            'WE': random.randint(20, 60)
        }
        population.append(T_green)
    return population
# 选择: 根据适应度值选择最优的个体
def selection(population):
    fitness_values = [fitness(individual, distances) for individual in population]
    total_fitness = sum(fitness_values)
    selection_probs = [f / total_fitness for f in fitness_values]

```

```

        selected_index = np.random.choice(range(POPULATION_SIZE),
p=selection_probs)
        return population[selected_index]
# 交叉：两个父代个体交叉产生子代
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_RATE:
        crossover_point = random.choice(list(distances.keys
()))
        child1 = {**parent1}
        child2 = {**parent2}
        for direction in list(distances.keys()):
            if direction == crossover_point:
                break
            child1[direction] = parent2[direction]
            child2[direction] = parent1[direction]
        return child1, child2
    return parent1, parent2
# 变异：随机调整个体的某个基因（绿灯时长）
def mutate(individual):
    if random.random() < MUTATION_RATE:
        direction = random.choice(list(distances.keys()))
        individual[direction] = random.randint(20, 60)
    return individual
# 遗传算法的主循环
def genetic_algorithm():
    population = initialize_population()
    for generation in range(GENERATIONS):
        new_population = []
        # 选择、交叉和变异产生新的种群
        for _ in range(POPULATION_SIZE // 2):
            parent1 = selection(population)
            parent2 = selection(population)
            child1, child2 = crossover(parent1, paren
t2)
            child1 = mutate(child1)
            child2 = mutate(child2)
            new_population.extend([child1, child2])
        population = new_population
        # 输出当前代最优解的适应度
        best_individual = max(population, key=lambda ind:
fitness(ind, distances))
        best_fitness = fitness(best_individual, distances)
        print(f"Generation {generation + 1}: Best fitness = {best_fitness:.2f}")
    return best_individual
# 假设的车辆平均速度

```

```

v_avg = 11.11 # 40 km/h
# 运行遗传算法
best_solution = genetic_algorithm()
print(f"最优的信号灯配置 >>>>>> 导入 csv 表")
print(f"得到最大车流平均速度为: {speed}km/h")

⑩计算两条主路的最优平均速度
import pandas as pd
import numpy as np
np.random.seed(36)
intersections = ['纬中路-环西路', '纬中路-经二路', '纬中路-经三路',
                 '纬中路-经中路', '纬中路-景区出入口', '纬中路-经四路',
                 '纬中路-经五路', '纬中路-环东路', '经中路-纬一路',
                 '经中路-环北路', '经中路-环南路',
                 '纬中路-经一路']
directions = ['north_south', 'south_north', 'east_west', 'west_east']
traffic_data = []
for intersection in intersections:
    for direction in directions:
        vehicle_count = np.random.randint(100, 500)
        traffic_data.append([intersection, direction, vehicle_count])
traffic_df = pd.DataFrame(traffic_data, columns=['intersection', 'direction', 'vehicle_count'])
# 定义信号灯周期调整函数
def optimize_signal_cycle(intersection, traffic_data, min_time, max_time):
    # 获取该交叉口的车流量数据
    flow_data = traffic_data[traffic_data['intersection'] == intersection]
    # 初始化信号灯周期为默认值
    signal_times = {direction: np.random.randint(min_time, max_time) for direction in directions}
    # 计算每个方向的车流通行速度(车辆数 / 通行时间)
    def calculate_average_speed(signal_times):
        speeds = {}
        for direction in directions:
            flow = flow_data[flow_data['direction'] == direction]['vehicle_count'].sum()
            time = signal_times[direction]
            if time > 0:
                speed = flow / time
                speeds[direction] = speed

```

```

        return speeds
    # 目标：在满足所有方向通行的情况下，使平均车速最大
    speeds = calculate_average_speed(signal_times)
    average_speed = np.mean(list(speeds.values()))
    # 信号灯周期和平均车速
    return signal_times, average_speed
# 优化每个交叉口的信号灯周期
optimal_signals = {}
average_speeds = {}
for intersection in intersections:
    signal_times, avg_speed = optimize_signal_cycle(intersection, traffic_df, 10, 41)
    optimal_signals[intersection] = signal_times
    average_speeds[intersection] = avg_speed
for k, v in average_speeds.items():
    print(f"{k}的平均车速为: {v:4.1f}")
# 计算整体的平均车速
total_average_speed = np.mean(list(average_speeds.values()))
total_average_speed = total_average_speed.round()
# print((total_average_speed.round()) * 3.6)
print()
print(f'主路最优平均速度{total_average_speed} m/s')
print(f'主路最优平均速度{total_average_speed * 3.6} km/h')

```

附录五

介绍：问题三解题代码

①分表

```

import pandas as pd
import os
data = pd.read_csv('../无车牌数据的附件 2.csv')
filtered_data = data[data['时间'].str.contains('-05-')]
# 确保时间列是日期时间格式
data['时间'] = pd.to_datetime(data['时间'])
# 按车牌号和时间排序
data = data.sort_values(by=['车牌号', '时间'])
# 创建'车辆'文件夹，如果不存在的话
folder_name = "车辆"
if not os.path.exists(folder_name):
    os.makedirs(folder_name)
# 每个车牌号生成一个 csv 文件并保存到'车辆'文件夹中
for license_plate, group_data in data.groupby('车牌号'):
    filename = f'{folder_name}/{license_plate}.csv'
    group_data.to_csv(filename, index=False)

```

```
print(f'Saved {filename}')
```

②构建车辆轨迹

```
import pandas as pd
data = pd.read_csv('../无车牌数据的附件 2.csv')
# 确保时间列是日期时间格式
data['时间'] = pd.to_datetime(data['时间'])
# 定义筛选时间的范围
start_date = '2024-05-01'
# end_date = '2024-05-05 23:59:59'
end_date = '2024-05-06'
# 筛选时间在 5 月 1 日到 5 月 5 日之间的数据
filtered_data = data[(data['时间'] >= start_date) & (data['时间'] <= end_date)]
print(filtered_data)
# 按车牌号和时间排序
filtered_data = filtered_data.sort_values(by=['车牌号', '时间'])
# 构建行车轨迹
trajectory_data = filtered_data.groupby('车牌号').agg({
    '时间': [( '开始时间', 'min'), ('结束时间', 'max')],
    '交叉口': [( '行车轨迹', lambda x: ' -> '.join(x))]
}).reset_index()
# 展开多级列索引
trajectory_data.columns = [col[0] if col[1] == " " else col[1] for col in
trajectory_data.columns.values]
# 查看结果
print(trajectory_data)
# trajectory_data.to_csv('车辆轨迹表.csv')
```

③计算行驶距离和时间

```
import pandas as pd
# 读取 CSV 文件
csv_file = "车辆轨迹表.csv" # 替换为你的 CSV 文件路径
df = pd.read_csv(csv_file)
# 路段距离字典（保持不变）
corrected_road_segments = {
    ('环北路-经中路', '经中路-纬一路'): 520,
    ('经中路-环北路', '纬一路-经中路'): 520,
    ('经中路-纬一路', '经中路-纬中路'): 510,
    ('纬一路-经中路', '纬中路-经中路'): 510,
    ('经中路-纬中路', '经中路-环南路'): 710,
    ('纬中路-经中路', '环南路-经中路'): 710,
    ('纬中路-环西路', '纬中路-经一路'): 460,
    ('环西路-纬中路', '经一路-纬中路'): 460,
    ('纬中路-经一路', '纬中路-经二路'): 340,
```

('经一路-纬中路','经二路-纬中路'): 340,
 ('纬中路-经二路','纬中路-经三路'): 440,
 ('经二路-纬中路','经三路-纬中路'): 440,
 ('纬中路-经三路','纬中路-经中路'): 420,
 ('经三路-纬中路','经中路-纬中路'): 420,
 ('纬中路-经中路','纬中路-景区出入口'): 530,
 ('经中路-纬中路','景区出入口-纬中路'): 530,
 ('纬中路-景区出入口','纬中路-经四路'): 560,
 ('景区出入口-纬中路','经四路-纬中路'): 560,
 ('纬中路-经四路','纬中路-经五路'): 430,
 ('经四路-纬中路','经五路-纬中路'): 430,
 ('纬中路-经五路','纬中路-环东路'): 270,
 ('经五路-纬中路','环东路-纬中路'): 270,
 # 逆向
 ('经中路-纬一路','经中路-环北路'): 520,
 ('纬一路-经中路','环北路-经中路'): 520,
 ('经中路-纬中路','纬一路-经中路'): 510,
 ('纬中路-经中路','经中路-纬一路'): 510,
 ('经中路-环南路','纬中路-经中路'): 710,
 ('环南路-经中路','经中路-纬中路'): 710,
 ('纬中路-经一路','环西路-纬中路'): 460,
 ('经一路-纬中路','纬中路-环西路'): 460,
 ('纬中路-经二路','经一路-纬中路'): 340,
 ('经二路-纬中路','纬中路-经一路'): 340,
 ('纬中路-经三路','经二路-纬中路'): 440,
 ('经三路-纬中路','纬中路-经二路'): 440,
 ('纬中路-经中路','经三路-纬中路'): 420,
 ('经中路-纬中路','纬中路-经三路'): 420,
 ('纬中路-景区出入口','经中路-纬中路'): 530,
 ('景区出入口-纬中路','纬中路-经中路'): 530,
 ('纬中路-经四路','景区出入口-纬中路'): 560,
 ('经四路-纬中路','纬中路-景区出入口'): 560,
 ('纬中路-经五路','经四路-纬中路'): 430,
 ('经五路-纬中路','纬中路-经四路'): 430,
 ('纬中路-环东路','经五路-纬中路'): 270,
 ('环东路-纬中路','纬中路-经五路'): 270,
 # 组合路线
 ('纬中路-环西路','经二路-纬中路'): 460 + 340,
 ('经二路-纬中路','环西路-纬中路'): 460 + 340,
 ('环西路-纬中路','纬中路-经二路'): 460 + 340, # 增加的行
 ('经五路-纬中路','纬中路-景区出入口'): 430 + 560,
 ('纬中路-景区出入口','纬中路-经五路'): 430 + 560,
 ('纬中路-经五路','景区出入口-纬中路'): 430 + 560, # 增加的行
 ('经三路-纬中路','纬中路-景区出入口'): 420 + 530,

('纬中路-景区出入口', '纬中路-经三路'): 420 + 530,
 ('纬中路-经三路', '景区出入口-纬中路'): 420 + 530, # 增加的行
 ('环北路-经中路', '经中路-环南路'): 520 + 510 + 710,
 ('经中路-环南路', '经中路-环北路'): 520 + 510 + 710,
 ('经中路-环北路', '环南路-经中路'): 520 + 510 + 710,
 ('纬中路-环西路', '纬中路-经二路'): 460 + 340,
 ('纬中路-经二路', '环西路-纬中路'): 460 + 340,
 ('环西路-纬中路', '经二路-纬中路'): 460 + 340,
 ('纬中路-环西路', '纬中路-经三路'): 460 + 340 + 440,
 ('纬中路-经三路', '环西路-纬中路'): 460 + 340 + 440,
 ('环西路-纬中路', '经三路-纬中路'): 460 + 340 + 440,
 ('纬中路-环西路', '纬中路-经中路'): 460 + 340 + 440 + 420,
 ('纬中路-经中路', '环西路-纬中路'): 460 + 340 + 440 + 420,
 ('环西路-纬中路', '经中路-纬中路'): 460 + 340 + 440 + 420,
 ('纬中路-环西路', '纬中路-景区出入口'): 460 + 340 + 440 + 420 + 530,
 ('纬中路-景区出入口', '环西路-纬中路'): 460 + 340 + 440 + 420 + 530,
 ('环西路-纬中路', '景区出入口-纬中路'): 460 + 340 + 440 + 420 + 530,
 ('纬中路-环西路', '纬中路-经四路'): 460 + 340 + 440 + 420 + 530 + 560,
 ('纬中路-经四路', '环西路-纬中路'): 460 + 340 + 440 + 420 + 530 + 560,
 ('环西路-纬中路', '经四路-纬中路'): 460 + 340 + 440 + 420 + 530 + 560,
 ('纬中路-环西路', '纬中路-经五路'): 460 + 340 + 440 + 420 + 530 + 560 + 430,
 ('纬中路-经五路', '环西路-纬中路'): 460 + 340 + 440 + 420 + 530 + 560 + 430,
 ('环西路-纬中路', '经五路-纬中路'): 460 + 340 + 440 + 420 + 530 + 560 + 430,
 ('纬中路-环西路', '纬中路-环东路'): 460 + 340 + 440 + 420 + 530 + 560 + 430
 + 270, ('纬中路-环东路', '环西路-纬中路'): 460 + 340 + 440 + 420 + 530 + 560 +
 430 + 270, ('环西路-纬中路', '环东路-纬中路'): 460 + 340 + 440 + 420 + 530 + 560
 + 430 + 270, ('纬中路-环西路', '经中路-纬一路'): 460 + 340 + 440 + 420 + 510,
 ('经中路-纬一路', '环西路-纬中路'): 460 + 340 + 440 + 420 + 510,
 ('环西路-纬中路', '纬一路-经中路'): 460 + 340 + 440 + 420 + 510,
 ('纬中路-环西路', '经中路-环北路'): 460 + 340 + 440 + 420 + 510 + 520,
 ('经中路-环北路', '环西路-纬中路'): 460 + 340 + 440 + 420 + 510 + 520,
 ('环西路-纬中路', '环北路-经中路'): 460 + 340 + 440 + 420 + 510 + 520,
 ('纬中路-环西路', '经中路-环南路'): 460 + 340 + 440 + 420 + 720,
 ('经中路-环南路', '环西路-纬中路'): 460 + 340 + 440 + 420 + 720,
 ('环西路-纬中路', '环南路-经中路'): 460 + 340 + 440 + 420 + 720,
 ('纬中路-经一路', '纬中路-经三路'): 340 + 440,
 ('纬中路-经三路', '经一路-纬中路'): 340 + 440,
 ('经一路-纬中路', '经三路-纬中路'): 340 + 440,
 ('纬中路-经一路', '纬中路-经中路'): 340 + 440 + 420,
 ('纬中路-经中路', '经一路-纬中路'): 340 + 440 + 420,
 ('经一路-纬中路', '经中路-纬中路'): 340 + 440 + 420,
 ('纬中路-经一路', '纬中路-景区出入口'): 340 + 440 + 420 + 530,
 ('纬中路-景区出入口', '经一路-纬中路'): 340 + 440 + 420 + 530,
 ('经一路-纬中路', '景区出入口-纬中路'): 340 + 440 + 420 + 530,

('纬中路-经一路', '纬中路-经四路'): 340 + 440 + 420 + 530 + 560,
 ('纬中路-经四路', '经一路-纬中路'): 340 + 440 + 420 + 530 + 560,
 ('经一路-纬中路', '经四路-纬中路'): 340 + 440 + 420 + 530 + 560,
 ('纬中路-经一路', '纬中路-经五路'): 340 + 440 + 420 + 530 + 560 + 430,
 ('纬中路-经五路', '经一路-纬中路'): 340 + 440 + 420 + 530 + 560 + 430,
 ('经一路-纬中路', '经五路-纬中路'): 340 + 440 + 420 + 530 + 560 + 430,
 ('纬中路-经一路', '纬中路-环东路'): 340 + 440 + 420 + 530 + 560 + 430 + 270,
 ('纬中路-环东路', '经一路-纬中路'): 340 + 440 + 420 + 530 + 560 + 430 + 270,
 ('经一路-纬中路', '环东路-纬中路'): 340 + 440 + 420 + 530 + 560 + 430 + 270,
 ('纬中路-经一路', '经中路-纬一路'): 340 + 440 + 420 + 510,
 ('经中路-纬一路', '经一路-纬中路'): 340 + 440 + 420 + 510,
 ('经一路-纬中路', '纬一路-经中路'): 340 + 440 + 420 + 510,
 ('纬中路-经一路', '经中路-环北路'): 340 + 440 + 420 + 510 + 520,
 ('经中路-环北路', '经一路-纬中路'): 340 + 440 + 420 + 510 + 520,
 ('经一路-纬中路', '环北路-经中路'): 340 + 440 + 420 + 510 + 520,
 ('纬中路-经一路', '经中路-环南路'): 340 + 440 + 420 + 710,
 ('经中路-环南路', '经一路-纬中路'): 340 + 440 + 420 + 710,
 ('经一路-纬中路', '环南路-经中路'): 340 + 440 + 420 + 710,
 ('纬中路-经二路', '纬中路-经中路'): 440 + 420,
 ('纬中路-经中路', '经二路-纬中路'): 440 + 420,
 ('经二路-纬中路', '经中路-纬中路'): 440 + 420,
 ('纬中路-经二路', '纬中路-景区出入口'): 440 + 420 + 530,
 ('纬中路-景区出入口', '经二路-纬中路'): 440 + 420 + 530,
 ('经二路-纬中路', '景区出入口-纬中路'): 440 + 420 + 530,
 ('纬中路-经二路', '纬中路-经四路'): 440 + 420 + 530 + 560,
 ('纬中路-经四路', '经二路-纬中路'): 440 + 420 + 530 + 560,
 ('经二路-纬中路', '经四路-纬中路'): 440 + 420 + 530 + 560,
 ('纬中路-经二路', '纬中路-经五路'): 440 + 420 + 530 + 560 + 430,
 ('纬中路-经五路', '经二路-纬中路'): 440 + 420 + 530 + 560 + 430,
 ('经二路-纬中路', '经五路-纬中路'): 440 + 420 + 530 + 560 + 430,
 ('纬中路-经二路', '纬中路-环东路'): 440 + 420 + 530 + 560 + 430 + 270,
 ('纬中路-环东路', '经二路-纬中路'): 440 + 420 + 530 + 560 + 430 + 270,
 ('经二路-纬中路', '环东路-纬中路'): 440 + 420 + 530 + 560 + 430 + 270,
 ('纬中路-经二路', '经中路-纬一路'): 440 + 420 + 510,
 ('经中路-纬一路', '经二路-纬中路'): 440 + 420 + 510,
 ('经二路-纬中路', '纬一路-经中路'): 440 + 420 + 510,
 ('纬中路-经二路', '经中路-环北路'): 440 + 420 + 510 + 520,
 ('经中路-环北路', '经二路-纬中路'): 440 + 420 + 510 + 520,
 ('经二路-纬中路', '环北路-经中路'): 440 + 420 + 510 + 520,
 ('纬中路-经二路', '经中路-环南路'): 440 + 420 + 710,
 ('经中路-环南路', '经二路-纬中路'): 440 + 420 + 710,
 ('经二路-纬中路', '环南路-经中路'): 440 + 420 + 710,
 ('纬中路-经三路', '纬中路-景区出入口'): 420 + 530,
 ('纬中路-景区出入口', '经三路-纬中路'): 420 + 530,

('经三路-纬中路', '景区出入口-纬中路'): 420 + 530,
 ('纬中路-经三路', '纬中路-经四路'): 420 + 530 + 560,
 ('纬中路-经四路', '经三路-纬中路'): 420 + 530 + 560,
 ('经三路-纬中路', '经四路-纬中路'): 420 + 530 + 560,
 ('纬中路-经三路', '纬中路-经五路'): 420 + 530 + 560 + 430,
 ('纬中路-经五路', '经三路-纬中路'): 420 + 530 + 560 + 430,
 ('经三路-纬中路', '经五路-纬中路'): 420 + 530 + 560 + 430,
 ('纬中路-经三路', '纬中路-环东路'): 420 + 530 + 560 + 430 + 270,
 ('纬中路-环东路', '经三路-纬中路'): 420 + 530 + 560 + 430 + 270,
 ('经三路-纬中路', '环东路-纬中路'): 420 + 530 + 560 + 430 + 270,
 ('纬中路-经三路', '经中路-纬一路'): 420 + 510,
 ('经中路-纬一路', '经三路-纬中路'): 420 + 510,
 ('经三路-纬中路', '纬一路-经中路'): 420 + 510,
 ('纬中路-经三路', '经中路-环北路'): 420 + 510 + 520,
 ('经中路-环北路', '经三路-纬中路'): 420 + 510 + 520,
 ('经三路-纬中路', '环北路-经中路'): 420 + 510 + 520,
 ('纬中路-经三路', '经中路-环南路'): 420 + 710,
 ('经中路-环南路', '经三路-纬中路'): 420 + 710,
 ('经三路-纬中路', '环南路-经中路'): 420 + 710,
 ('纬中路-经中路', '纬中路-经四路'): 530 + 560,
 ('纬中路-经四路', '经中路-纬中路'): 530 + 560,
 ('经中路-纬中路', '经四路-纬中路'): 530 + 560,
 ('纬中路-经中路', '纬中路-经五路'): 530 + 560 + 430,
 ('纬中路-经五路', '经中路-纬中路'): 530 + 560 + 430,
 ('经中路-纬中路', '经五路-纬中路'): 530 + 560 + 430,
 ('纬中路-经中路', '纬中路-环东路'): 530 + 560 + 430 + 270,
 ('纬中路-环东路', '经中路-纬中路'): 530 + 560 + 430 + 270,
 ('经中路-纬中路', '环东路-纬中路'): 530 + 560 + 430 + 270,
 ('纬中路-经中路', '经中路-环北路'): 510 + 520,
 ('经中路-环北路', '经中路-纬中路'): 510 + 520,
 ('经中路-纬中路', '环北路-经中路'): 510 + 520,
 ('纬中路-景区出入口', '纬中路-经五路'): 560 + 430,
 ('纬中路-经五路', '景区出入口-纬中路'): 560 + 430,
 ('景区出入口-纬中路', '经五路-纬中路'): 560 + 430,
 ('纬中路-景区出入口', '纬中路-环东路'): 560 + 430 + 270,
 ('纬中路-环东路', '景区出入口-纬中路'): 560 + 430 + 270,
 ('景区出入口-纬中路', '环东路-纬中路'): 560 + 430 + 270,
 ('环东路-纬中路', '纬中路-景区出入口'): 270 + 430 + 560,
 ('纬中路-景区出入口', '经中路-纬一路'): 530 + 510,
 ('经中路-纬一路', '景区出入口-纬中路'): 530 + 510,
 ('景区出入口-纬中路', '纬一路-经中路'): 530 + 510,
 ('纬中路-景区出入口', '经中路-环北路'): 530 + 510 + 520,
 ('经中路-环北路', '景区出入口-纬中路'): 530 + 510 + 520,
 ('景区出入口-纬中路', '环北路-经中路'): 530 + 510 + 520,

```

('纬中路-景区出入口', '经中路-环南路'): 530 + 710,
('经中路-环南路', '景区出入口-纬中路'): 530 + 710,
('景区出入口-纬中路', '环南路-经中路'): 530 + 710,
('纬中路-经四路', '纬中路-环东路'): 530 + 270,
('纬中路-环东路', '经四路-纬中路'): 530 + 270,
('经四路-纬中路', '环东路-纬中路'): 530 + 270,
('纬中路-经四路', '经中路-纬一路'): 560 + 530 + 510,
('经中路-纬一路', '经四路-纬中路'): 560 + 530 + 510,
('经四路-纬中路', '纬一路-经中路'): 560 + 530 + 510,
('纬中路-经四路', '经中路-环北路'): 560 + 530 + 510 + 520,
('经中路-环北路', '经四路-纬中路'): 560 + 530 + 510 + 520,
('经四路-纬中路', '环北路-经中路'): 560 + 530 + 510 + 520,
('纬中路-经四路', '经中路-环南路'): 560 + 530 + 710,
('经中路-环南路', '经四路-纬中路'): 560 + 530 + 710,
('经四路-纬中路', '环南路-经中路'): 560 + 530 + 710,
('纬中路-经五路', '经中路-纬一路'): 430 + 560 + 530 + 510,
('经中路-纬一路', '经五路-纬中路'): 430 + 560 + 530 + 510,
('经五路-纬中路', '纬一路-经中路'): 430 + 560 + 530 + 510,
('纬中路-经五路', '经中路-环北路'): 430 + 560 + 530 + 510 + 520,
('经中路-环北路', '经五路-纬中路'): 430 + 560 + 530 + 510 + 520,
('经五路-纬中路', '环北路-经中路'): 430 + 560 + 530 + 510 + 520,
('纬中路-经五路', '经中路-环南路'): 430 + 560 + 530 + 710,
('经中路-环南路', '经五路-纬中路'): 430 + 560 + 530 + 710,
('经五路-纬中路', '环南路-经中路'): 430 + 560 + 530 + 710,
('纬中路-环东路', '经中路-纬一路'): 270 + 430 + 560 + 530 + 510,
('经中路-纬一路', '环东路-纬中路'): 270 + 430 + 560 + 530 + 510,
('环东路-纬中路', '纬一路-经中路'): 270 + 430 + 560 + 530 + 510,
('纬中路-环东路', '经中路-环北路'): 270 + 430 + 560 + 530 + 510 + 520,
('经中路-环北路', '环东路-纬中路'): 270 + 430 + 560 + 530 + 510 + 520,
('环东路-纬中路', '环北路-经中路'): 270 + 430 + 560 + 530 + 510 + 520,
('纬中路-环东路', '经中路-环南路'): 270 + 430 + 560 + 530 + 710,
('经中路-环南路', '环东路-纬中路'): 270 + 430 + 560 + 530 + 710,
('环东路-纬中路', '环南路-经中路'): 270 + 430 + 560 + 530 + 710,
('经中路-环北路', '经中路-环南路'): 520 + 510 + 710,
('经中路-环南路', '环北路-经中路'): 520 + 510 + 710,
('环北路-经中路', '环南路-经中路'): 520 + 510 + 710,
('经中路-纬一路', '经中路-环南路'): 510 + 710,
('经中路-环南路', '纬一路-经中路'): 510 + 710,
('纬一路-经中路', '环南路-经中路'): 510 + 710,
}# 规范化路段名称
def normalize_segment_name(name):
    return name.replace(" ", "")
# 对路段的起点和终点进行排序, 确保双向路段统一比较
def normalize_road_segments(start, end):

```

```

start = normalize_segment_name(start)
end = normalize_segment_name(end)
return tuple(sorted([start, end])) # 按字母顺序排序
# 处理双向路段名称并获取路段距离
def get_segment_distance_corrected(start, end):
    start = normalize_segment_name(start)
    end = normalize_segment_name(end)
    if (start, end) in corrected_road_segments:
        return corrected_road_segments[(start, end)]
    elif (end, start) in corrected_road_segments:
        return corrected_road_segments[(end, start)]
    print(f'路段未匹配: {start} -> {end}')
    return 0
# 计算车辆的总行驶距离, 避免重复路径的计算
def calculate_total_distance_v3(route):
    total_distance = 0
    segments = route.split(' -> ')
    for i in range(len(segments) - 1):
        start = segments[i].strip()
        end = segments[i + 1].strip()
        # 跳过重复路段
        if start == end:
            continue
        distance = get_segment_distance_corrected(start, end)
        total_distance += distance
    return total_distance
# 计算行驶时间
def calculate_travel_time(start_time, end_time):
    start_time = pd.to_datetime(start_time)
    end_time = pd.to_datetime(end_time)
    travel_time = (end_time - start_time).total_seconds() # 计算总时间, 单位为秒
    return travel_time
# 判断行驶时间是否超过一天
def check_time_exceeds_one_day(start_time, end_time):
    start_time = pd.to_datetime(start_time)
    end_time = pd.to_datetime(end_time)
    return 1 if (end_time - start_time).days >= 1 else 0 # 超过 1 天返回 1, 否则返回 0
df['总行驶距离 (m)'] = df['行车轨迹'].apply(calculate_total_distance_v3)
# 计算行驶时间
df['行驶时间 (秒)'] = df.apply(lambda row: calculate_travel_time(row['开始时间'], row['结束时间']), axis=1)
# 判断是否超过一天
df['超过一天'] = df.apply(lambda row: check_time_exceeds_one_day(row['开始时间'], row['结束时间']), axis=1)

```

```

], row['结束时间']), axis=1)
# 对没有超过一天的数据计算速度
df['速度 (m/s)'] = df.apply(lambda row: row['总行驶距离 (m)'] / row['行驶时间 (秒)'] if row['超过一天'] == 0 and row['行驶时间 (秒)'] > 0 else None, axis=1)
output_csv_file = "车辆综合表表.csv" # 设置保存的文件名
df.to_csv(output_csv_file, index=False, encoding='utf-8-sig')
# 输出结果
print(df[['车牌号', '总行驶距离 (m)']])
data = df.drop(columns=['开始时间', '结束时间', '行车轨迹'])
print(data)
data.to_csv('巡游车判断条件.csv')

```

③计算轨迹距离

```

import pandas as pd
# 车辆数据
data = [
    ('1', '3B04AU8', '环西路-纬中路 -> 经二路-纬中路 -> 经三路-纬中路 -> 经中路-纬中路 -> 经三路-纬中路 -> 经二路-纬中路'),
    ('9', '3B23AAK', '经三路-纬中路'),
    ('10', '3B23C8K', '经三路-纬中路'),
    ('11', '3B24BD9', '纬中路-景区出入口'),
    ('12', '3B24Z37', '纬中路-景区出入口 -> 纬中路-景区出入口 -> 经中路-纬中路 -> 经中路-纬一路 -> 经中路-环北路'),
    ('13', '3B2546CK', '经五路-纬中路 -> 纬中路-景区出入口 -> 经中路-纬中路 -> 经中路-纬一路 -> 经中路-环北路'),
    ('14', '3B25AACM', '经中路-环北路 -> 环东路-纬中路'),
    ('15', '3B2606H',
     '环西路-纬中路 -> 经一路-纬中路 -> 经二路-纬中路 -> 经三路-纬中路 -> 经中路-纬中路 -> 纬中路-景区出入口'),
    ('16', '3B2677I', '经中路-环南路 -> 经中路-纬中路 -> 经中路-纬一路 -> 经中路-环北路'),
    ('17', '3B26XJB',
     '纬中路-景区出入口 -> 纬中路-景区出入口 -> 经中路-纬中路 -> 经中路-纬一路 -> 经中路-纬一路 -> 经中路-纬中路 -> 纬中路-景区出入口'),
    ('18', '3B29H5H',
     '经中路-环北路 -> 经中路-纬一路 -> 经中路-纬中路 -> 经中路-纬中路 -> 经中路-纬一路 -> 经中路-环北路'),
    ('19', '3B29K3F', '经中路-纬中路 -> 经中路-纬一路 -> 经中路-环北路'),
    ('20', '3B29Q6H', '经中路-环南路 -> 经中路-纬中路 -> 经中路-纬中路 -> 经中路-纬一路 -> 经中路-纬中路'),
    ('21', '3B2A4LQ', '经一路-纬中路 -> 环西路-纬中路 -> 环西路-纬中路'),
    ('22', '3B2ANMF',
     '经四路-纬中路 -> 纬中路-景区出入口 -> 经中路-纬中路 -> 经三路-纬中路 -> 经二路-纬中路 -> 经一路-纬中路 -> 环西路-纬中路 -> 环西路-纬中

```

```

路 -> 环西路-纬中路'),
    ('23', '3B2BB78M', '经中路-环北路 -> 经中路-环北路'),
    ('24', '3B2BP5X', '经中路-环南路'),
    ('25', '3B2C59T', '经中路-环南路 -> 经中路-环南路'),
    ('26', '3B2CKSC',
     '经中路-环北路 -> 经中路-纬一路 -> 经中路-纬中路 -> 纬中路-景区出
入口 -> 经中路-纬中路 -> 经三路-纬中路 -> 经三路-纬中路 -> 经中路-纬中
路 -> 纬中路-景区出入口 -> 纬中路-景区出入口 -> 经中路-纬中路 -> 经三路
-纬中路 -> 经三路-纬中路'),
    ('27', '3B2ECD6M', '环西路-纬中路 -> 环西路-纬中路 -> 经中路-环北路'),
    ('28', '3B2HB66K',
     '环西路-纬中路 -> 经一路-纬中路 -> 经二路-纬中路 -> 经中路-纬中路
-> 纬中路-景区出入口 -> 经中路-纬中路 -> 经中路-环南路 -> 经中路-环南路
'),
    ('29', '3B2KTCD', '经三路-纬中路 -> 纬中路-景区出入口 -> 纬中路-景区出
入口 -> 经中路-纬中路 -> 经中路-环南路'),
    ('30', '3B2ML6C', '经中路-环北路 -> 经中路-环北路'),
    ('31', '3B2OC8E',
     '纬中路-景区出入口 -> 经中路-纬中路 -> 经三路-纬中路 -> 经二路-纬
中路 -> 环西路-纬中路 -> 环西路-纬中路'),
    ('32', '3B2P4CI', '经五路-纬中路 -> 经四路-纬中路 -> 纬中路-景区出入口
-> 经中路-纬中路 -> 经中路-环北路')])
# 规范化路段名称
def normalize_segment_name(name):
    return name.replace(" ", "")
# 对路段的起点和终点进行排序，确保双向路段统一比较
def normalize_road_segments(start, end):
    start = normalize_segment_name(start)
    end = normalize_segment_name(end)
    return tuple(sorted([start, end])) # 按字母顺序排序
# 处理双向路段名称并获取路段距离
def get_segment_distance_corrected(start, end):
    start = normalize_segment_name(start)
    end = normalize_segment_name(end)
    if (start, end) in corrected_road_segments:
        return corrected_road_segments[(start, end)]
    elif (end, start) in corrected_road_segments:
        return corrected_road_segments[(end, start)]
    print(f'路段未匹配: {start} -> {end}')
    return 0
# 计算车辆的总行驶距离，避免重复路径的计算
def calculate_total_distance_v3(route):
    total_distance = 0
    segments = route.split(' -> ')

```

```

    for i in range(len(segments) - 1):
        start = segments[i].strip()
        end = segments[i + 1].strip()
        # 跳过重复路段
        if start == end:
            continue
        distance = get_segment_distance_corrected(start, end)
        total_distance += distance
    return total_distance
# 重新计算每辆车的行驶距离
final_results_v3 = []
for car in data:
    car_id = car[0]
    route = car[2]
    total_distance = calculate_total_distance_v3(route)
    final_results_v3.append((car_id, total_distance))
# 转为 DataFrame 输出
df_final_results_v3 = pd.DataFrame(final_results_v3, columns=['Car ID', 'Total
Distance (m)'])
print(df_final_results_v3)

④计算停车位数量
import pandas as pd
# 加载新上传的文件以检查数据并使用排队论模型估算停车需求
circling_vehicle_file_path = '巡游车辆数据.csv'
circling_vehicle_data = pd.read_csv(circling_vehicle_file_path, encoding='utf-8')
# 显示数据的前几行以了解其结构
circling_vehicle_data.head()
# 第一步：计算到达率 ( $\lambda$ )
# 假设我们使用数据集中的时间来估计到达率
# 假设我们计算的是整个五一假期
# 将开始和结束时间转换为 datetime 格式以进行计算
circling_vehicle_data['开始时间'] = pd.to_datetime(circling_vehicle_data['开始时间
'])
circling_vehicle_data['结束时间'] = pd.to_datetime(circling_vehicle_data['结束时间
'])
# 计算数据集的总持续时间（以小时为单位）
total_duration = (circling_vehicle_data['结束时间'].max() - circling_vehicle_data['开
始时间'].min()).total_seconds() / 3600
# 计算车辆总数（到达率  $\lambda$ ）
total_vehicles = circling_vehicle_data['车牌号'].nunique()
arrival_rate = total_vehicles / total_duration # 每小时车辆数
# 第二步：估算服务率 ( $\mu$ )
# 假设平均停车时间为 2 小时（每个停车位每小时有 0.5 辆车离开）
service_rate = 250 # 每小时每个停车位服务车辆数
# 第三步：使用 M/M/1 模型计算停车需求

```

```

## 计算利用率  $\rho = \lambda / \mu$ 
utilization = arrival_rate / service_rate
# 假设停车场容量为 50 个停车位
parking_capacity = 50
# 计算系统中的平均车辆数 ( $L_s$ )
average_vehicles_in_system = utilization / (1 - utilization)
# 计算所需的额外停车位 (如果平均值超过容量)
required_additional_parking = max(0, average_vehicles_in_system -
parking_capacity)
# Iteratively increase parking capacity to find the required number of parking spaces
def find_required_parking_capacity(arrival_rate, service_rate, initial_capacity=500,
max_capacity=10000, step=100):
    capacity = initial_capacity
    while capacity <= max_capacity:
        utilization = arrival_rate / (service_rate * capacity)
        if utilization < 1: # When the utilization is less than 1, system is stable
            average_vehicles_in_system = utilization / (1 - utilization)
            if average_vehicles_in_system <= capacity:
                return capacity # Return the capacity where system becomes
stable    capacity += step
    return max_capacity # Return max capacity if no stable point is found
# Find the required parking capacity
required_parking_capacity = find_required_parking_capacity(arrival_rate,
service_rate)
# 显示结果
print({'车辆总数': total_vehicles,
      '到达率 ( $\lambda$ ):': arrival_rate,
      '服务率 ( $\mu$ ):': service_rate,
      '利用率 ( $\rho$ ):': utilization,
      '系统中的平均车辆数 ( $L_s$ ):': average_vehicles_in_system,
      '所需的额外停车位': required_parking_capacity})

```

⑤M/m/c 模型求解

```

import pandas as pd
import mpmath as mp
# 加载车辆数据
circling_vehicle_file_path = '巡游车辆数据.csv'
circling_vehicle_data = pd.read_csv(circling_vehicle_file_path, encoding='utf-8')
# 转换开始和结束时间为 datetime 格式
circling_vehicle_data['开始时间'] = pd.to_datetime(circling_vehicle_data['开始时间
'])circling_vehicle_data['结束时间'] = pd.to_datetime(circling_vehicle_data['结束
时间'])# 计算数据集的总持续时间 (以小时为单位)
total_duration = (circling_vehicle_data['结束时间'].max() - circling_vehicle_data['
开始时间'].min()).total_seconds() / 3600

```

```

# 计算车辆总数 (到达率  $\lambda$ )
total_vehicles = circling_vehicle_data['车牌号'].nunique()
arrival_rate = total_vehicles / total_duration # 每小时车辆数
# 假设服务率  $\mu$  (平均停车时间为 2 小时, 每个停车位每小时有 0.5 辆车离开)
service_rate = 0.5 # 每小时每个停车位服务车辆数
# M/M/c 模型计算 (使用 mpmath 处理大数值)
def calculate_P0_mpmath(arrival_rate, service_rate, c):
    rho = arrival_rate / (c * service_rate)
    sum_terms = sum(mp.power(arrival_rate / service_rate, n) / mp.fac(n) for n in
range(c))
    last_term = mp.power(arrival_rate / service_rate, c) / (mp.fac(c) * (1 - rho))
    P0 = 1 / (sum_terms + last_term)
    return P0
def calculate_Lq_mpmath(arrival_rate, service_rate, c):
    rho = arrival_rate / (c * service_rate)
    P0 = calculate_P0_mpmath(arrival_rate, service_rate, c)
    Lq = (P0 * mp.power(arrival_rate / service_rate, c) * rho) / (mp.fac(c) * (1 - rho)
** 2)
    return Lq
def calculate_Ls_mpmath(arrival_rate, service_rate, c):
    Lq = calculate_Lq_mpmath(arrival_rate, service_rate, c)
    Ls = Lq + arrival_rate / service_rate
    return Lq, Ls
# 假设停车位数量 c = 600
c = 600
# 计算系统的平均排队车辆数 (Lq) 和系统中的平均车辆数 (Ls)
Lq_mpmath, Ls_mpmath = calculate_Ls_mpmath(arrival_rate, service_rate, c)
# 打印结果
results = { '需要临时停车位数量 (c)': c,
            '平均排队车辆数 (Lq)': Lq_mpmath,
            '系统中的平均车辆数 (Ls)': Ls_mpmath
}print(results)

```

⑥ 计算巡游车辆

```

import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置显示中文字体
# 另外, 由于字体更改以后会导致坐标轴中部分字符无法正常显示, 这是需要更改 axes.unicode_minus 参数。
plt.rcParams['axes.unicode_minus'] = False # 设置正常显示符号
data = pd.read_csv('去重车辆综合表.csv')
print(data)
# 绘制速度的分布直方图
plt.figure(figsize=(8, 6))

```

```

plt.hist(data['速度 (m/s)'], bins=50, edgecolor='black', alpha=0.7)
plt.title('速度分布 (m/s)')
plt.xlabel('速度 (m/s)')
plt.ylabel('频率')
plt.grid(True)
plt.show()"""从速度的分布直方图中可以看到，大多数车辆的速度集中在较低的范围，尤其是接近 0 的部分。这表明部分车辆可能是在低速行驶，可能是在寻找停车位。"""# 设置速度阈值，低于此速度的车辆将被视为巡游车辆
speed_threshold = 0.5
# 先筛选出低速车辆
low_speed_vehicles = data[data['速度 (m/s)'] < speed_threshold]
# 对于每个低速车辆，分析其行车轨迹，寻找是否存在重复的交叉口
# 定义一个函数来识别重复或相邻的交叉口模式
def is_circling(trajectory):
    # 分割行车轨迹中的路段
    roads = trajectory.split(' -> ')
    # 判断是否有重复的路段或循环路线
    return len(roads) != len(set(roads))
# 通过轨迹来进一步筛选
circling_vehicles_with_trajectory = low_speed_vehicles[low_speed_vehicles['行车轨迹'].apply(is_circling)]
# 获取最终筛选出的车辆车牌号
final_circling_vehicles = circling_vehicles_with_trajectory['车牌号'].unique()
# 将筛选出的车辆显示出来
final_circling_vehicles_list = pd.DataFrame(final_circling_vehicles, columns=['巡游车辆车牌号'])
print(final_circling_vehicles_list)
circling_data = data[data['车牌号'].isin(final_circling_vehicles)]
print(circling_data)
# circling_data.to_csv('巡游车辆数据.csv')

```

附录六

介绍：问题四的解题代码

①分析对比流量密度

```

import pandas as pd
import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei'] # 设置显示中文字体
# 另外，由于字体更改以后会导致坐标轴中部分字符无法正常显示，这是需要更改 axes.unicode_minus 参数。
plt.rcParams['axes.unicode_minus'] = False # 设置正常显示符号
data = pd.read_csv('../无车牌数据的附件 2.csv')

```

```

# 将‘时间’列转换为日期时间格式
data['时间'] = pd.to_datetime(data['时间'])
# 过滤“五一”黄金周数据
holiday_data = data[(data['时间'] >= '2024-05-01') & (data['时间'] <= '2024-05-05')]
# print(holiday_data) 114w
# holiday_data.to_csv('五一假期数据.csv')
# 过滤 4 月的参考数据
reference_data = data[(data['时间'] >= '2024-04-01') & (data['时间'] <= '2024-04-30')]
# reference_data.to_csv('四月数据.csv')
# 假设五一期间和参考期间的时段长度为：五一期间 5 天，参考期间 30 天
holiday_duration = 5 # 五一期间总天数
reference_duration = 30 # 参考期间总天数
# 按日期和交叉口汇总流量
holiday_flow = holiday_data.groupby([holiday_data['时间'].dt.date, '交叉口']).size().reset_index(name='流量计数_五一')
reference_flow = reference_data.groupby([reference_data['时间'].dt.date, '交叉口']).size().reset_index(name='流量计数_参考')
# 按交叉口汇总整个五一期间和参考期间的总流量
holiday_flow_total = holiday_flow.groupby('交叉口')['流量计数_五一'].sum().reset_index()
reference_flow_total = reference_flow.groupby('交叉口')['流量计数_参考'].sum().reset_index()
# 合并数据
flow_comparison = pd.merge(holiday_flow_total, reference_flow_total, on='交叉口', how='outer')
# 计算流量密度（流量/时间）
flow_comparison['流量密度_五一'] = flow_comparison['流量计数_五一'] / holiday_duration
flow_comparison['流量密度_参考'] = flow_comparison['流量计数_参考'] / reference_duration
# 避免除以零的问题
flow_comparison['流量密度_参考'] = flow_comparison['流量密度_参考'].replace(0, 1e-6)
# 计算流量密度变化百分比
flow_comparison['流量密度变化百分比'] = (flow_comparison['流量密度_五一'] - flow_comparison['流量密度_参考']) / flow_comparison['流量密度_参考'] * 100
# 导出结果为 CSV 文件
# flow_comparison.to_csv('4 月 5 月流量密度变化对比.csv')
# 显示前几行结果
print(flow_comparison.head())
# 51 期间检查是否有车辆重复出现的情况
duplicate_vehicles = holiday_data.groupby(['车牌号', '交叉口']).size().reset_index(name='出现次数')
# 过滤出现多次的车辆，假设 3 次及以上被认为是绕行

```

```

circling_vehicles = duplicate_vehicles[duplicate_vehicles['出现次数'] >= 3]
# print(circling_vehicles)
# 4 月期间检查是否有车辆重复出现的情况
reference_vehicles = reference_data.groupby(['车牌号', '交叉口']).size().reset_index(name='出现次数')
# 过滤出现多次的车辆，假设 3 次及以上被认为是绕行
reference_vehicles = reference_vehicles[reference_vehicles['出现次数'] >= 3]
# print(reference_vehicles)
# 计算五一期间总车辆数
total_holiday_vehicles = holiday_data['车牌号'].nunique()
# 五一期间绕行车辆数（出现次数大于等于 3 次的车辆）
circling_vehicles = duplicate_vehicles[duplicate_vehicles['出现次数'] >= 3]
circling_holiday_vehicles = circling_vehicles['车牌号'].nunique()
# 计算五一期间绕行车辆占比
holiday_circling_percentage = (circling_holiday_vehicles / total_holiday_vehicles) * 100
print()
print(f'五一期间绕行车辆占总车辆的百分比: {holiday_circling_percentage:.2f}%')
# 计算 4 月期间总车辆数
total_reference_vehicles = reference_data['车牌号'].nunique()
# 4 月期间绕行车辆数（出现次数大于等于 3 次的车辆）
circling_reference_vehicles = reference_vehicles['车牌号'].nunique()
# 计算 4 月期间绕行车辆占比
reference_circling_percentage = (circling_reference_vehicles / total_reference_vehicles) * 100
print(f'4 月期间绕行车辆占总车辆的百分比: {reference_circling_percentage:.2f}%')
# 对比结果
if holiday_circling_percentage < reference_circling_percentage:
    print("五一期间临时交通管控措施减少了绕行车辆。")
else:
    print("五一期间绕行车辆增加，临时交通管控措施可能效果不佳。")
# 分析管控路段与通行方向
# 分析“五一”期间在红色管控路段的车辆流量
# holiday_control_flow = holiday_data.groupby('交叉口').size().reset_index(name='流量计数')
# 车辆通行方向分析
directions_analysis = holiday_data.groupby(['交叉口', '方向']).size().reset_index(name='方向计数')
print(directions_analysis)
# 流量高峰时段分析
# 按小时划分流量
# holiday_data['hour'] = pd.to_datetime(holiday_data['时间']).dt.hour
### 统计每个时段的车辆流量

```

```
# hourly_flow = holiday_data.groupby('hour').size().reset_index(name='流量计数')
### 绘制流量时段变化图
# plt.plot(hourly_flow['hour'], hourly_flow['流量计数'], label='五一假期车流量')
# plt.xlabel('时间（小时）')
# plt.ylabel('车辆流量计数')
# plt.title('五一黄金周各时段车辆流量')
# plt.legend()
# plt.show()
```

②做流量密度折线图

```
from pyecharts.charts import Line
from pyecharts import options as opts
import pandas as pd
# 创建基于提供数据的 DataFrame
data = {
    '交叉口': [
        '环东路-纬中路', '环西路-纬中路', '纬中路-景区出入口', '经一路-纬中路',
        '经三路-纬中路',
        '经中路-环北路', '经中路-环南路', '经中路-纬一路', '经中路-纬中路', '经
        二路-纬中路',
        '经五路-纬中路', '经四路-纬中路'
    ],
    '流量密度_五一': [1453.8, 50359.6, 17479.8, 17462.8, 17599.8, 20746.8,
        30634.0, 24262.4, 29869.2, 9950.8, 6828.4, 1581.0],
    '流量密度_参考': [3721.77, 53625.03, 12986.43, 19672.33, 19978.7, 20404.63,
        29815.43, 17660.93, 28607.73, 11790.0, 4777.1, 1886.77]
}df = pd.DataFrame(data)
# 创建折线图
line = (
    Line()
    .add_xaxis(df['交叉口'].tolist()) # X 轴是交叉口名称
    .add_yaxis("五一期间流量密度", df['流量密度_五一'].tolist(), is_smooth=True,
        label_opts=opts.LabelOpts(is_show=False)) # 五一期间流量密度折线
    .add_yaxis("四月流量密度", df['流量密度_参考'].tolist(), is_smooth=True,
        label_opts=opts.LabelOpts(is_show=False)) # 参考期间流量密度折线
    .set_global_opts(
        title_opts=opts.TitleOpts(title="五一期间与四月流量密度折线图"),
        xaxis_opts=opts.AxisOpts(name="交叉口", axislabel_opts={"rotate": 45}),
        yaxis_opts=opts.AxisOpts(name="流量密度"),
        tooltip_opts=opts.TooltipOpts(trigger="axis"),
        legend_opts=opts.LegendOpts(pos_top="5%")
    )# 渲染图表
line.render("流量密度折线图.html")
```