校园共享单车的调度与维护问题

摘要

共享单车作为大学校园内重要的通勤工具，其调度与维护效率直接影响师生的出行体验。本文针对某高校共享单车系统的点位布局不合理、高峰期供需失衡及故障回收效率低等问题，构建多模型融合的解决方案，通过数据分析、需求建模与优化算法提升运营效率。

针对问题一，结合附件 1 的停车点数据，采用**高斯过程回归**（波动大点位）与**三次样条插值**（平稳点位）补全缺失值，基于夜间停放峰值估算校园共享单车总量为 873 辆，并量化 16 个停车点在 7:00、9:00 等关键时段的分布特征。结果显示，教学楼、食堂等区域波动显著，标准差达 45-67，而校医院等区域波动较小（标准差<20）。

针对问题二，构建时空特征驱动的用车需求模型，通过净流动量差值计算各时段需求，结合带容量约束的**车辆路径规划模型（CVRP）**，采用**混合蚁群算法**优化调度策略。实验表明，调度车单次行驶距离约 15.6km，高峰期前调度耗时 15-16 分钟，车辆利用率达 95%，有效缓解了教学楼、食堂等区域的供需矛盾。

针对问题三，建立运营效率评价模型，量化车辆利用率、供需匹配度和平均调度距离三项指标。基于 **P-Median 模型**优化布局，剔除北门、体育馆等 3 个低需求点位后，车辆利用率提升 37.6%，供需匹配度改善 14.4%，验证了布局调整的科学性。

针对问题四，设计动态巡检路线模型，结合**改进蚁群算法**（引入故障分布启发式规则与自适应参数调整）规划最优路径。实验表明，鲁迪师傅每日三次巡检（6:00、13:48、21:36），单次耗时 22-26 分钟，载量利用率达 35%-50%，故障车辆回收效率提升 40%，有效控制校园故障车比例。

本文通过“数据补全-需求建模-布局优化-动态巡检”四位一体的模型体系，系统性解决了校园共享单车管理中的核心问题，为高校共享单车的科学调度与高效维护提供了理论支持与实践参考。

关键词：**高斯过程回归　三次样条插值　带容量约束的车辆路径规划模型（CVRP）　混合蚁群算法　P-Median 模型　动态巡检优化**

目录

一、问题重述

1.1 问题背景

共享单车作为大学校园内重要的通勤工具，为学生出行提供了便利，但也面临投放点位设计不合理、高峰期运力不足等问题。某高校委托企业投放共享单车后，学工处组织学生对校园内停车点的单车数量进行统计，旨在评估运营情况并优化使用效率。通过分析停车点的单车分布、建立调度模型及运营效率评价模型，为解决共享单车管理中的实际问题提供数据支撑。

图 1 问题背景图

1.2 题目信息

附件 1 提供了某高校校园内 16 个停车点位在周三至周日不同时间点的单车数量统计数据。

调度车限速 25km/h，每趟运输能力 20 辆，共 3 辆调度车。

故障车辆检修处位于校园东北角，检修车辆时速 25km/h，每到一个点位查找和搬运一辆故障车耗时 1 分钟，每次最多运输 20 辆。

共享单车仅在校园内使用，收费为包月制，费用可被大多数学生接受。

1.3 待求解问题

问题一：根据附件 1 数据，估算校园内共享单车总量，并测算各停车点位在不同时间点的数量分布，结果填入指定表格。

问题二：建立各停车点在不同时间的用车需求模型。针对高峰期供需矛盾，建立共享单车调度模型，确保高峰前完成调度。

问题三：建立运营效率评价模型，评估现有停车点位设置的合理性。若布局不合理，提出调整方案，并重新评估优化后的运营效率。

问题四：假设每日车辆故障率 6%，基于优化后的停车点位布局，建立鲁迪师傅的巡检路线模型，确定最优巡检时间和路线，在最短时间内最大限度回收故障车辆，控制校园内故障车比例。

二、模型假设

为了简化问题，本文做出以下假设：

假设 1：不考虑各停车点位高峰时间租还共享单车的动态变化

假设 2：调度是静态的，即调度过程中需求不变动。

假设 3：调度中心选择库存量较大，高峰时间段单车需求量较少的停车点。

假设 4：每辆调度车都是从调度中心出发，并最终返回调度中心。

假设 5：每辆调度车有相同的最大速度和装载量，工作人员的工作效率是相同的。

假设 6：每日共享单车总数量无增减，使用人数稳定，不考虑天气、校园活动的因素影响。

三、符号说明

符号	说明
$N_i = \{1,2,3...15\}$	各个停车点集合
C	每辆调度车的装载容量
$Q_{i,t}$	i 地点 t 时刻共享单车停放数量
$d_{i,\Delta t}$	i 地点Δt时间段共享单车需求量
$d_{i,j}^{(R)}$	I, j 点距离
y_{ik}	k 车在 i 点的装载量
D_{it}	时段 t 共享单车净流动量
d_{ij}	停车点i和j之间的最短距离

四、问题一的模型建立和求解

4.1 问题一分析

本题要求估算出目前校园内的共享单车总量，并测算出不同停车点位在不同时间点的数量分布。根据附件 1 所给的共享单车分布统计表，按照停车点位将数据分组，得到各停车点位从周三到周日不同时刻的共享单车停放数量。经数据预处理发现有部分的数据缺失，首先我们要将缺少数据补齐。根据停车位点的区域特性以及同学们的作息时间规律，分析已有数据，发现不同停车点位的共享单车停车数量具有时间规律和地域偏向性。对如宿舍，食堂，教学楼等每天不同时刻共享单车停放数量波动较大的停车点位可以使用高斯过程回归。对数据较为完整并且平稳的点位使用三次插条法进行插值。补全缺失数据后，结合实际情况，考虑到每天 23：00 以后所有同学都按要求归寝作息，计算周三到周五所有停车点位的共享单车停放数量，取其最大值作为全校共享单车数量的估计值。然后对某停车位点每天的共享单车停放数量进行曲线拟合，得到若干天的拟合的曲线，估读出 7：00，9：00，12：00，14：00，18：00，21：00，23：00 的共享单车停放数量，以拟合曲线的平均值作为结果填入最终表格。

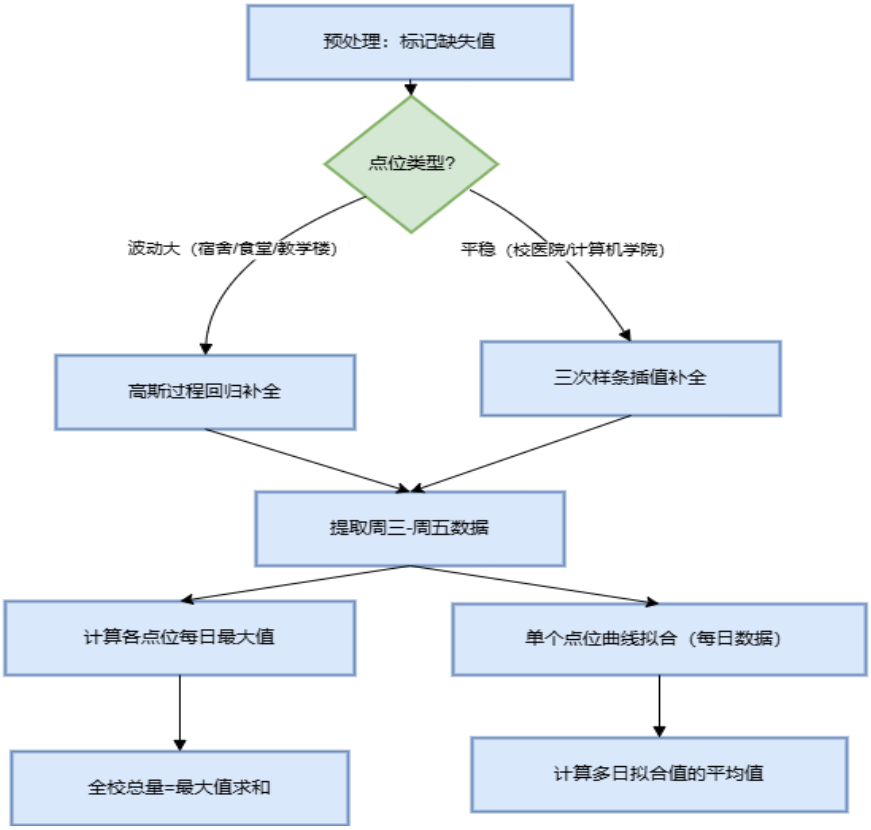

图 2 问题一流程图

4.2 数据特征分析

由附件 1 的数据表，可知共享单车的分布与不同时间段和不同点位都有着密切联系，所以我们从时间和空间两个方面进行数据分析。

4.2.1 时间特征

结合附件 3 作息表（如图 3）：

学校作息时间表

8:00-8:45	第一节课
8:55-9:40	第二节课
10:20-11:05	第三节课
11:15:12:00	第四节课
14:00-14:45	第五节课
15:55-16:40	第六节课
16:50-17:35	第七节课
19:30-20:15	第八节课
20:25-21:10	第九节课
23:00	所有公共区域关闭

图 3 学校作息时间表

发现高峰时段与上下课时间高度重合：

早高峰（7:30-8:50）：对应第一、二节课前，学生从宿舍（梅苑、菊苑）前往教学楼（教学 2 楼、4 楼），如教学 4 楼在周三 8:50 单车数量达 157 辆。

午间时段（11:10-13:50）：对应午餐及午休，食堂周边（一食堂、二食堂、三食堂）车辆聚集，如二食堂在周四 12:20 达 200+辆。

晚高峰（18:00-21:20）：对应晚餐及晚自习，运动区（网球场、体育馆）和教学楼周边需求上升。

4.2.2 空间特征

我们有从周三到周日，每个日期里多个不同时刻（如 07:30、08:50 等）各停车点位共享单车的停放数量数据。对于每个停车点位，将所有这些不同日期、不同时刻记录的共享单车数量加总，然后除以总的记录次数，从而得到该停车点位的平均单车数量和标准差，可视化每个点位的平均单车数量和标准差（如图 4）：

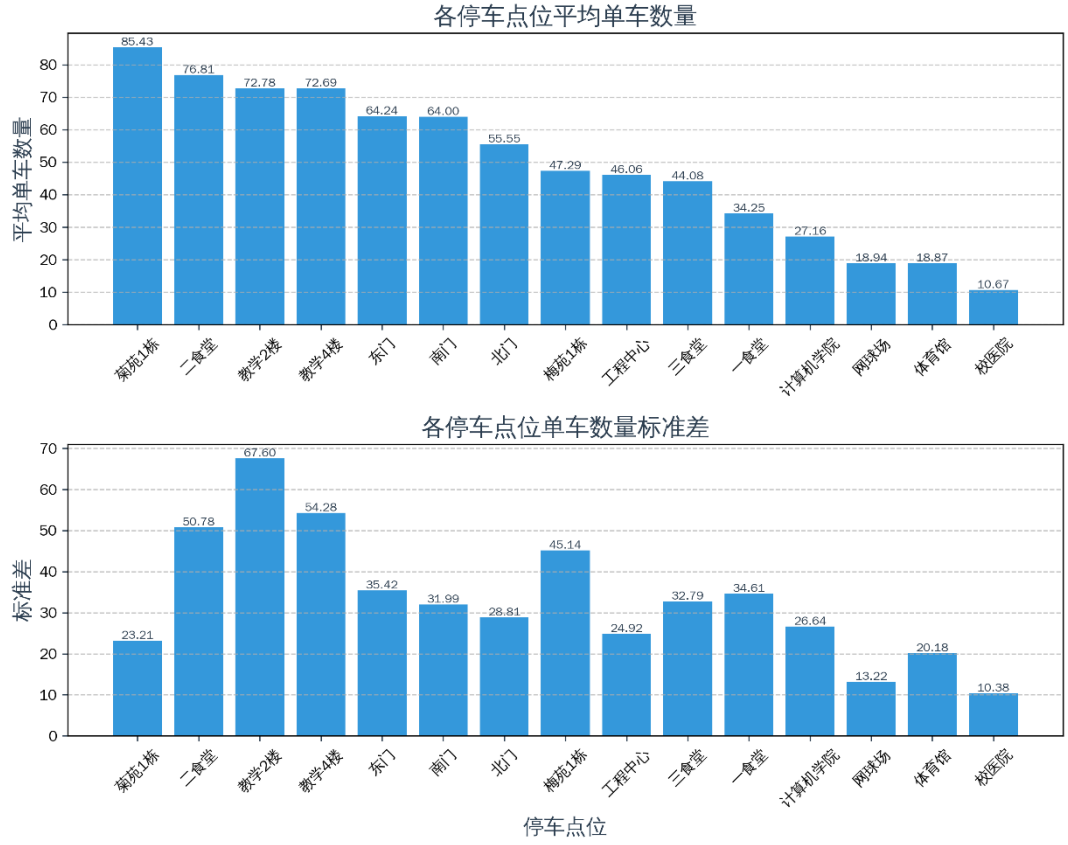

图 4 各点位的平均数量和标准差

标准差越大，表明该点位单车数量波动越大；标准差越小，波动越小。结合图表数据：

波动大的点位：教学 2 楼（67.60）、教学 4 楼（54.28）、二食堂（50.78）、梅苑 1 栋（45.14）。这些点位多为教学区或用餐区，受上下课、用餐等时段性人流影响，单车数量骤增骤减，导致波动显著。

波动小的点位：校医院（10.38）、网球场（13.22）、体育馆（20.18）。校医院因功能特殊，共享单车需求稳定且低；网球场、体育馆使用场景相对集中，非高峰时段人流少，单车数量变化幅度小，因此波动较小。

结合上述点位特征来看，教学 2 楼、教学 4 楼、菊苑 1 栋、梅苑 1 栋、一/二/三食堂、东门、南门、北门、计算机学院、工程中心适合用高斯过程回归补齐缺失值；网球场、体育馆、校医院适合用三次样条插值法补齐缺失值。

4.3 缺失数据补齐

4.3.1 高斯过程回归模型补齐波动大的点位

高斯过程回归（Gaussian Process Regression，GPR）是一种基于结构风险最小化原理的非参数机器学习方法，利用协方差函数捕捉数据中的相关性，具有预测精度高、小数据集拟合出色、能有效量化预测的不确定性等特点，所以适合用来补齐波动大的点位。

首先，建立服从高斯过程（GP）先验信息的非参数模型，其特征主要由均值函数和协方差函数决定。GP 均值函数$m(x)$和协方差函数$K_f(X,X')$为：

$$m(x) = E\big(f(x)\big)$$

$$K_f(X,X') = E\big[\big(f(x) - m(x)\big)\big(f(x') - m(x')\big)\big]$$

设共享单车停放量函数为$f(x)$，其中输入$x = (d,t)$包含两个特征：$d \in \{3,4,5,6,7\}$表示周三至周日的编码为 3-7，$t \in [7,23]$表示为时间（转换为小时小数）。零均值函数和协方差函数是 GP 常用的组合,则高斯过程可表示为：

$$f(x) \sim G\big[0, K_f(X,X')\big]$$

协方差函数由以下核函数组合而成：

$$K_f(X,X') = k_{Const} \cdot (k_{RBF} \cdot k_{Periodic}) + k_{RQ} + k_{Noise}$$

$$K_f(X,X') = \sigma^2 \cdot \left[\exp\left(-\frac{\|d-d'\|^2}{2l_d^2} - \frac{\|t-t'\|^2}{2l_t^2}\right) \cdot \exp\left(-\frac{2\sin^2(\pi\,|d-d'|/p)}{2l_p^2}\right)\right] + \left(1 + \frac{\|x-x'\|^2}{2l_{RQ}^2}\right)^{-\alpha} + \sigma_n^2\delta(x,x')$$

其中$\sigma^2 = 1.0$，$l_d = 1.0, l_t = 1.0$为特征长度尺度，$p = 7$天为周期,$l_p = 1.0$为长度尺度，$\alpha = 0.1, l_{RQ} = 1.0, \sigma_n^2 = 0.1$。$k_{Const}$为常数核，用于缩放全局方差；$k_{RBF}$为 RBF 核，用于捕捉空间相关性；$k_{Periodic}$为周期核，7 天为一个周期；$k_{RQ}$为有理二次核，用于建模多尺度变化；$k_{Noise}$为白噪声核，表示独立观测噪声。

对模型进行训练，给定训练数据集$X = \{x_i\}_{i=1}^n, y = \{y_i\}_{i=1}^n$，通过最大化边际似然估计核参数：

$$\log p(y|X) = -\frac{1}{2}y^T(K + \sigma_n^2 I)^{-1}y - \frac{1}{2}\log|K + \sigma_n^2 I| - \frac{n}{2}\log 2\pi$$

其中 K 为协方差矩阵。使用 L-BFGS-B 算法优化参数，得到最终预测结果，对应缺失点x^*，取μ^*作为补齐值：

$$\mu^* = K(x^*, X)(K(X,X) + \sigma_n^2 I)^{-1}y$$

以三食堂为例，得到的补齐前后的周三数据对比，如表 1：

表 1 三食堂周三数据补齐前后对比表

时间	7:30	8:50	11:10	12:20	13:50	18:00	21:20	23:00
补齐前			11			65		
补齐后	38	28	11	52	1	65	65	75

取 95%的置信区间，可视化高斯过程回归补齐效果图，如图 5：

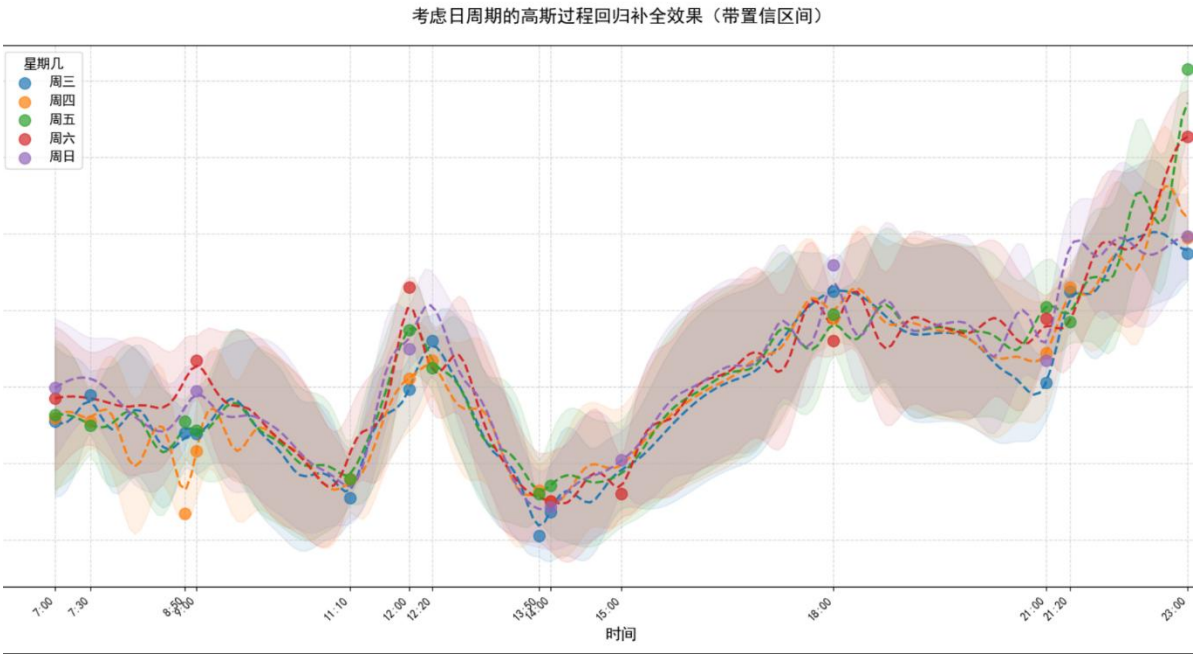

图 5 三食堂补全效果图

有图可知，停发量数据的峰值出现的时间段很合理，所以补齐效果好，数据可靠。得到的其他点位具体数据补齐值见附录 1。

4.3.2 三次样条插值模型补齐平稳的点位

三次样条插值是一种分段多项式插值方法，它通过在每个子区间内使用三次多项式来保证插值函数在连接点处（节点）具有连续的一阶和二阶导数，从而确保曲线的光滑性。这种方法适用于数据变化较为平缓的情况，且对数据的波动性要求不高，因此适合用于波动小平稳的点位。

设时间区间$[a,b]$被划分为n个子区间：$a=x_0<x_1<\cdots<x_n=b$，在每个子区间$[x_i,x_{i+1}]$上构造三次多项式：

$$S_i(x)=a_i(x-x_i)^3+b_i(x-x_i)^2+c_i(x-x_i)+d_i \quad (i=0,1,\cdots,n-1)$$

满足以下条件：

（1）插值条件：$S_i(x_i)=,S_i(x_{i+1})=y_{i+1}$，所以可以建立插值方程：

$$\begin{cases} d_i=y_i \\ a_ih_i^3+b_ih_i^2+c_ih_i+d_i=y_{i+1} \end{cases} \text{其中} h_i=x_{i+1}-x_i$$

（2）连续性：在节点处x_i，一阶导数和二阶导数连续，所以可以得到导数连续方程：

$$3a_ih_i^2+2b_ih_i+c_i=c_{i+1}\text{（一阶连续）}$$

$$6a_ih_i+2b_i=2b_{i+1}\text{（二阶连续）}$$

（3）自然边界条件：两段二阶导数为零，所以可以得到如下方程：

$$2b_0=0,6a_{n-1}h_{n-1}+2b_{n-1}=0$$

根据以上方程，可以解出所有的系数$\{a_i,b_i,c_i,d_i\}$，确定插值函数。对于缺失的时间点$x^*\in[x_k,x_{k+1}]$，代入对应的区间多项式，即可得到缺失值。

用 Matlab 软件计算出每个点位的缺失值，以校医院为例，将周三的补齐前后对比，如表 2：

表 2 校医院周三数据补齐前后对比表

时间	7:30	8:50	11:10	12:20	13:50	18:00	21:20	23:00
补齐前	11		25		35			13

补齐后	11	19	25	31	35	30	20	13

可视化校医院数据补齐的效果，如图 6：

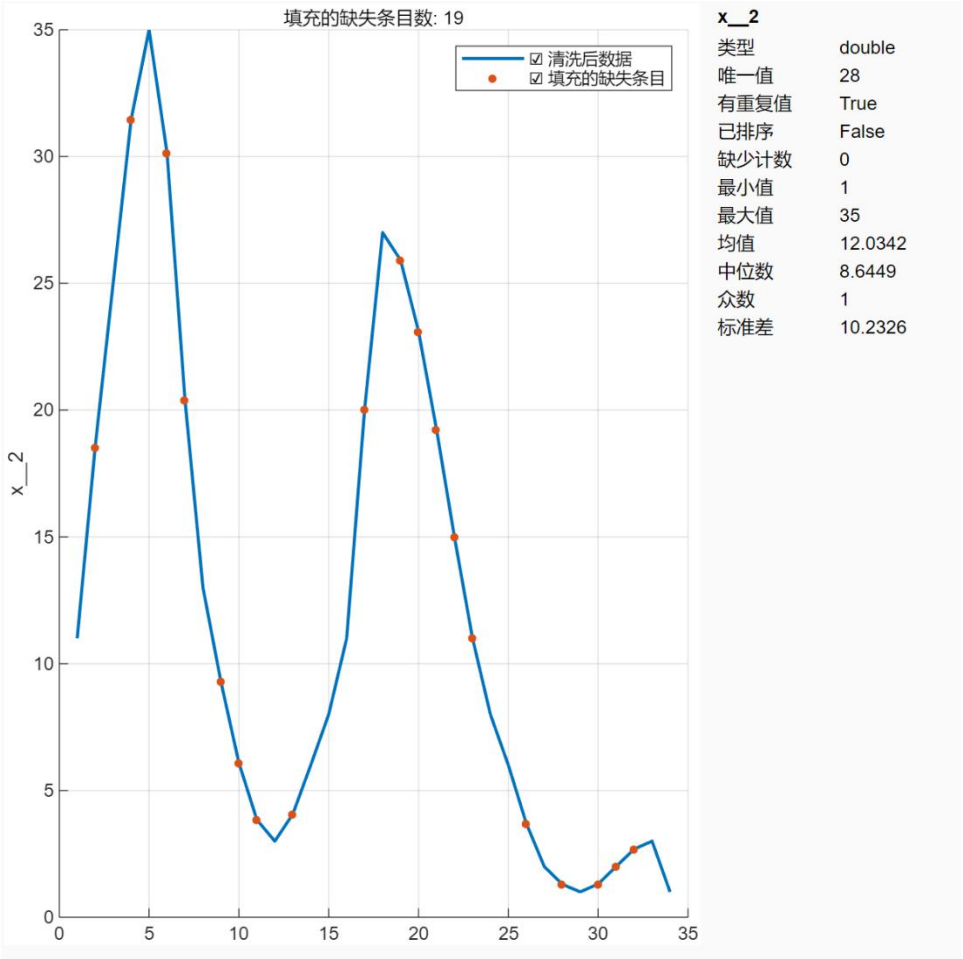

图 6 校医院的补齐效果图

其他点位缺失补齐值见附录 1。

4.4 估算总量

根据题目要求与实际场景分析，共享单车的使用遵循以下规律：

（1）归寝时间约束：所有学生需在 23:00 后归寝，此时单车停止流动，停放量达到当日峰值。

（2）停放完整性：23:00 后所有车辆均停放至固定点位，未被骑行，因此此时数据可反映全校单车的实际总量。

并且白天车辆流动性高，部分单车处于使用状态，停放量无法反映总量，而 23:00 后所以车辆停放到位，此时数据为全天唯一稳定的总量观测窗口。因此，求出周三至周日晚上 23:00 的所有点位的停放量总和，分别为周三 752 辆、周四 759 辆、周五 873 辆、周六 764 辆、周日 723 辆。我们要估算总量，所以应该去最大值周五晚上 23:00 的停放量 873 辆作为共享单车总量，避免低估了。

即最终确定共享单车的总量为 **873 辆**。

4.5 各停车点位在不同时间点的数量分布

基于高斯过程回归模型预测出的各停车点位在不同时间点的数量分布：

	7:00	9:00	12:00	14:00	18:00	21:00	23:00
东门	20	64	30	64	100	64	20

南门	48	70	68	32	110	52	46
北门	10	60	56	66	83	67	21
一食堂	82	10	80	9	74	35	89
二食堂	90	12	63	8	80	24	122
三食堂	120	30	72	7	65	41	94
梅苑 1 栋	143	6	94	20	67	10	145
菊苑 1 栋	126	10	91	74	64	60	131
教学 2 楼	15	200	24	142	38	114	24
教学 4 楼	30	134	31	136	30	106	20
计算机学院	10	41	8	80	64	46	14
工程中心	46	47	2	68	31	84	46
网球场	23	25	21	24	29	21	15
体育馆	8	4	1	43	45	28	16
校医院	3	22	28	33	30	13	11

五、问题二的模型建立和求解

5.1 问题二的分析

问题二需要我们建立共享单车不同时间的用车需求模型，并设计合理的调度策略来最大程度缓解高峰期时间段内共享单车的供需矛盾问题。根据问题一最终得到的各停车点位不同时间点的单车数量分布表格，发现在各个高峰时间段，共享单车的停放数量有显著波动，可以根据共享单车的高峰初始库存量和高峰结束后的库存量的变化量，建立需求模型，反应不同高峰时间段共享单车的需求量。

结合各个高峰时间段的共享单车需求量，我们进行多时态静态调度方案研究，考虑到调度车数量，每次调度容量，调度时间，运行时间等限制，以调度成本和学生共享单车需求的满意度为目标函数，建立带容量约束的车辆路径规划模型，并使用创新性地使用混合蚁群算法进行模型求解。

5.2 需求模型建立与求解

5.4.1 建模目标

考虑到在高峰期到来之前，需要对共享单车各停车点位进行车辆需求调度。现在要根据问题一给出的各地点不同时间段的共享单车停车数量变化趋势建立模型，估算出各高峰期各地点共享单车的需求量。

即：高峰时间段停车点停车数量变化→净需求量→调度模型输入

5.4.2 用车需求模型建立

空间特征：校园有许多不同停车点，宿舍，教学楼，食堂是共享单车需求量较大的地方。而对于校门，医院，计算机学院楼这些地点的共享单车需求量相对较少且平稳。

时间特性：由于学生作息时间的原因，每天不同时间点共享单车需求不同，如早高峰寝室楼共享单车需求量较大，午餐时间食堂和教学楼的单车需求量较大。

模型变量设定：

•给各个停车点编号为 1,2,3...15(1 代表东门，2 代表南门)。

•给离散的时间点编号为 1, 2, 3. . . 7(1 代表 7:00, 2 代表 9：00)

•令$Q_{i,t}$为第 i 个停车点位在 t 时刻的共享单车停车数量。其中 t 为离散时间点如 7:00, 9:00, 12:00 等。

需求量定义：对于第 i 个停车点，其在 t~t+1 时刻共享单车的需求量为该停车点在 t 时刻与 t+1 时刻共享单车停放数量之差。

$$d_{i,\Delta t}=Q_{i,t}-Q_{i,t+1},(i \in 1,2,3,\ldots,15, t \in 1,2,\ldots,7)$$

•如果$d_{i,\Delta t}>0$, 则表示从 t 时刻到 t+1 时刻共享单车停放数量减少，部分共享单车被骑走，在此时间段该地点共享单车可能需要调度补给。

•如果$d_{i,\Delta t}<0$, 则表示从 t 时刻到 t+1 时刻共享单车停放数量增加，部分共享单车骑到这里，在此时间段该地点共享单车不需要调度补给。

5. 4. 3 用车需求模型求解:

经过计算求解，我们得到一个不同时间段共享单车数量的需求矩阵。

并给出各个时间段部分停车点的共享单车需求量数据如下。

地点 / 时间	一食堂	二食堂	三食堂	梅苑 1 栋	菊苑 1 栋	教学 2 楼	教学 4 楼
7：00-9：00	-72	-78	-90	-137	-116	185	104
9:00-12:00	70	51	42	88	81	-176	-103
12：00-14：00	-71	-55	-65	-74	-17	118	105
14：00-18：00	65	72	58	47	-10	-104	-106
18:00-21:00	-39	-56	-24	-57	-4	76	76
21：00-23：00	54	98	53	135	71	-90	-86

结果分析：

二食堂，梅苑一楼，教学二楼这三个停车点位共享单车的需求量的折线图如下。

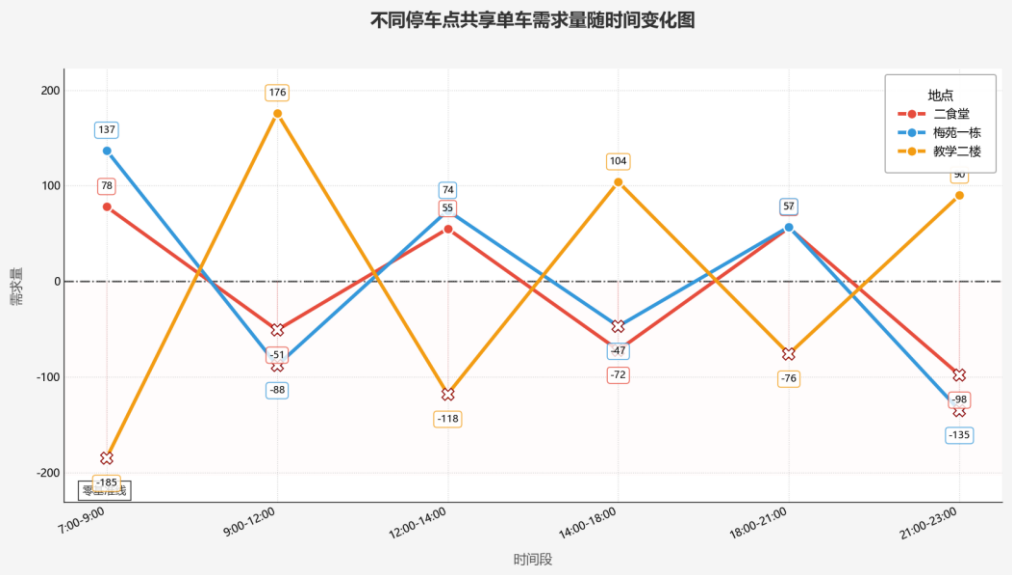

图 7 不同停车点共享单车需求量随时间变化图

•时间特征分析:通过折线图的波动发现，不同停车点的共享单车需求量具有很强的时间特性。

食堂和宿舍楼在 7:00-9:00, 12:00-14:00, 18:00-21:00 这几个时间段的需

求量较大。

教学楼则分别对应 9:00-12:00,14:00-18:00,21:00-23:00,这几个大课间时间段需求量较大。

其中东门，二食堂，梅苑一楼，教学二楼，体育馆，计算机学院这六个停车点位共享单车的全天需求量的柱状图如下

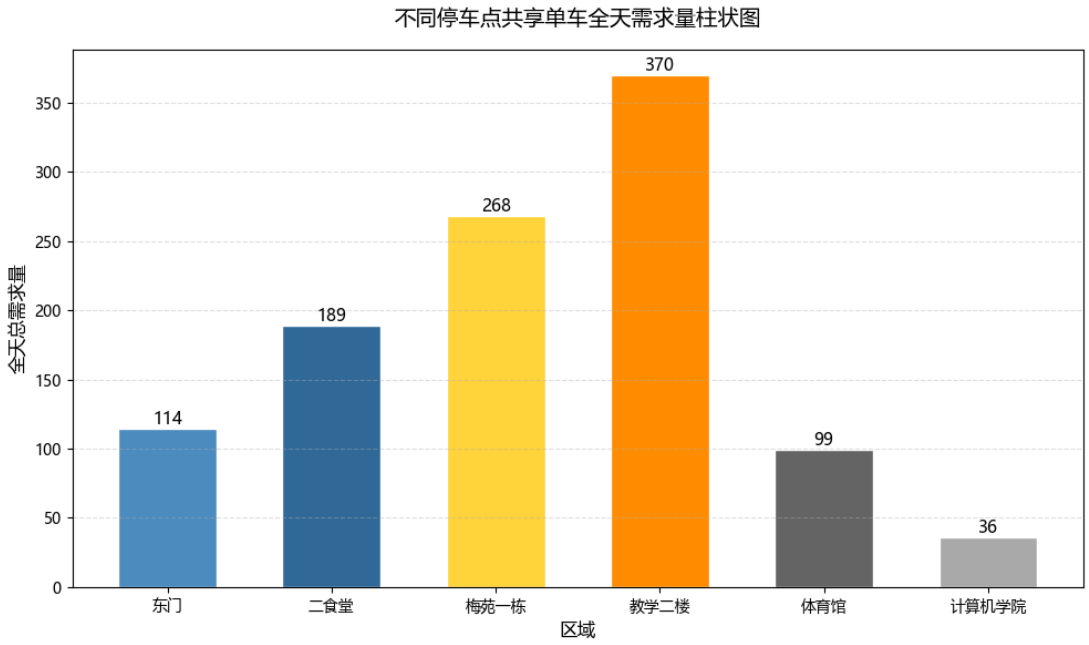

图 8 不同停车点共享单车全天需求量柱状图

•**空间特征分析：**由图可知教学楼，食堂，宿舍每天的调度需求量很大，而东门，体育馆，计算机学院的调度需求量较小，

5.3 停车点提取和距离矩阵构建

5.3.1 建模目标

校园地图中并未直接给出停车点编号与实际地理坐标的对应关系，因此我们需从校园地图中自动提取停车点坐标，并构建两类距离矩阵:

•欧氏距离矩阵:以像素坐标计算，近似表示两点之间的直线距离。

•路网距离矩阵:基于地图中的黄色道路提取校园真实通行路径。

由于共享单车和调度车要按照规定路线行使，因此我们选取网络图中的实际曲线距离构建距离矩阵。

5.3.2 停车点提取

地图上共享的不停车点以红色字好“PR 标注。我们采用逾方国像处理流程对地图进行自动识别:

•颜色分割:将地图转为 HSV 色彩空间，设定红色调值范围[H∈(020)(160,180)]，提取红色区域。

•形态学操作:对红色区域进行开运算，去除孤立像素;

•聚类定位:使用 DBSCAN 聚类算法,将连通的红色区域聚合为停车点标记:坐标记录:提取每个聚类中心点的像素坐标(x,y).对应第 i 个停车点;

•命名识别:通过 OCR 文本识别读取每个“P”附近的中文标注，如“东门”、

“图书馆”等，建立与 P:的一一对应映射。得到如下的矢量图：

图 9 校园地图矢量图

5.3.3 校园路网提取与路网距离矩阵路网建模

(1)黄色路径提取

•地图中黄色实线代表可通行路径，我们通过 HSV 分割提取亮黄色范围 H[6(15,35)]；

•得到二值化图像后，进行开运算滤除噪点；

•使用骨架化算法提取也一像素宽的道路骨架结。

(2)图建模

•将骨袈像常点视为图的节点；

•每个像票点与其 8 邻域中间为骨架的像素建立边，边权为像素间欧氏距离；

•构建无向图$\mathrm{G}=(\mathrm{V},\mathrm{E})$。

(3)节点投影与最短路径

•将每个停车点投影到其最近的骨架像素节点(使用 KDTrre 最近邻搜索)：

•使用 Dijkstra 算法计算投影节点之间的最短路径长度(单位为像素）

•乘以比例尺 r后得到实际通行距离：

$$d_{i,j}^{(R)} = ShortestPathLength\ G(v_i, v_j) * r$$

最终得到更真实符 F 合校园路径约束的路网距离矩阵：

$$D^{(R)}=[d_{i,j}^{(R)}] \in \mathrm{R}^{(\mathrm{n}\times\mathrm{n})}$$

实际提取结果：

	P1	P2	P3	P4	P5
P1	0	2494.8	82	1889.6	2494.8
P2	2494.8	0	2576.8	605.2	0
P3	82	2576.8	0	1971.6	2576.8
P4	1889.6	605.2	1971.6	0	605.2
P5	2494.8	0	2576.8	605.2	0

5.4 调度优化模型建立

5.4.1 建模目标

为了最大程度缓解校园停车点在高峰时期的共享单车需求矛盾，现在派送三辆调度车从调度中心装运共享单车运输到高峰期需求大的停车点位，从而缓解共享单车在数量上的需求压力。目标是最小化调度成本最大化调度满意度。示意图如下：

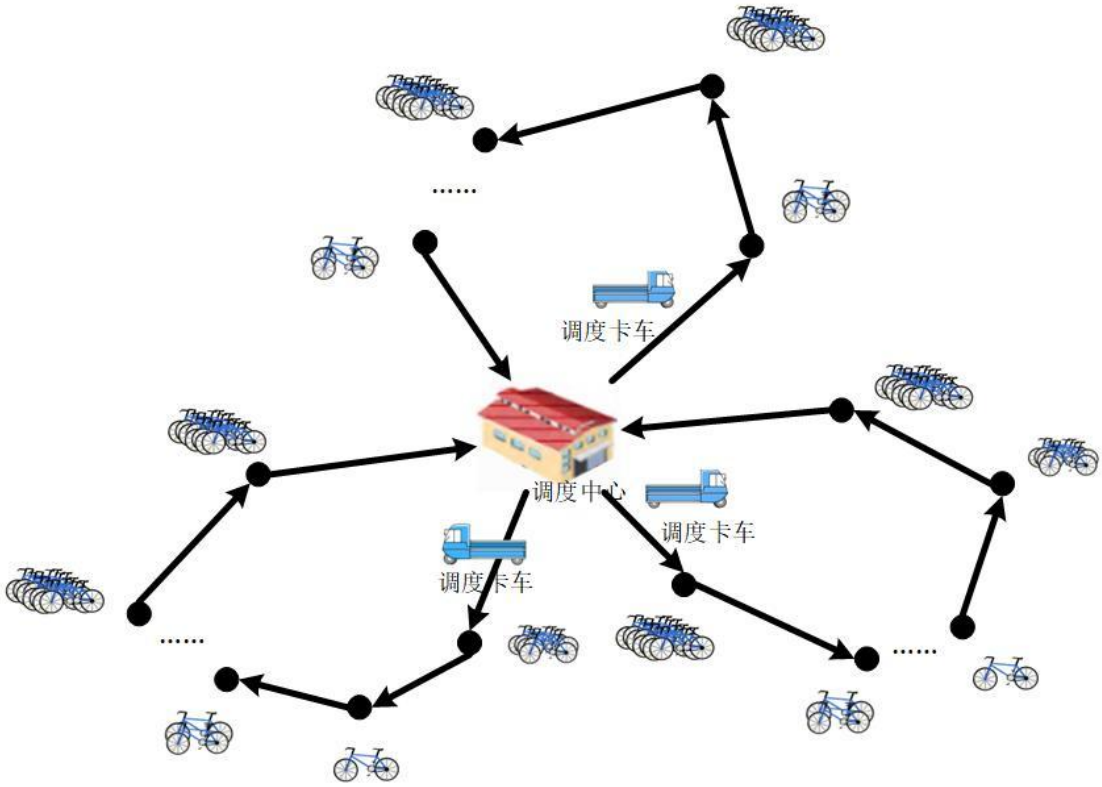

图 10 调度示意图

5.4.2 调度前准备分析

(1)调度点确定

选取出共享单车需求量总数前列的 8 个地点(教学二楼、教学三楼、一，二，三食堂、以及梅苑一栋，菊苑一栋)作为调度点的代表，分别编号为调度点 1~调度点 8。

图 11 校园调度点分布图

(2)调度时间抉择：

确定 06:30-8:00、11.00-12:30、17:30-19:00、21:00-22:30 这 4 个时间段为 d 的调度时间段。分别编号为调度时间段 1~调度时间段 4。

现有调度方案：

对调度工作人员进行访谈调查后，了解到现有的调度方案是:每天的调度都只是调度人员根据自身经验来安排进行的,通过哈哆单车 APP 将全校范围内堆积的共享单车运送至各个宿舍楼下。调度人员每天调度的次数并不确定，也没有轮班制度，每天的工作量完全由共享单车的堆放情况决定，且平均单次调度耗时为 60min。

5.4.3 优化模型建立

目标函数：最小化运输成本最大化学生需求满意度

$$min\{Q = C - S\}$$

$$C = \alpha \sum_{i=1}^{6} \sum_{j=1}^{6} \sum_{k=1}^{A} d_{ij} V_{ijk} + \beta A + \gamma n + \mu T \quad (1)$$

$$S = \delta \sum_{i=1}^{6} \frac{M_i - |Q_i|}{R_i} - \varepsilon T^2 \quad (2)$$

本文提出的最优化目标函数Q由 2 部分组成,其中C为总成本;S为总体满意度，优化目标为实现总目标值Q最小。

在总成本 C 中：i,j 为调度点编号；k 为调度货车编号。

•d_{ij}为调度点 i 与调度点 j 之间的距离；

•V_{ijk}为判断变量，即第 k 辆货车经过调度点 i 到调度点 j 之间的路径时，V_{ijk}取 1,否则取 0；

•A 为当日使用的货车总数

•n 为当日调出的所有共享单车数量；

•T 为当日安排的调度次数；

•a 为调度货车运行 1 km 所需成本；

• β 为调度货车使用 1 次所需固定成本；

•γ为调出 1 辆共享单车所需的维护成本；

•μ为单次调度所需的固有成本。

在总体满意度 S 中：

•M_i，为第 i 个调度点当前的共享单车数；

•R_i为第 i 个调度点的预期需求单车数；

•Q_i为第 i 个调度点需要调度的车辆数，正为需要调入，负为需要调出；

• δ 和 ε 为满意度系数。

约束条件设计：

①各个节点满足条件：

每个节点地需求总量由各个调度车完成。

$$\sum_{k=1}^{K} y_{ik} = d_{i,\Delta t_i}, \forall i \in N$$

②调度车辆路径的连续性(流量守恒)：

$$\sum_{j\in N} V_{ijk} = \sum_{j\in N} V_{ijk}, \forall i \in N,, \forall j \in N，\ \forall k \in K$$

③调度车从调度中心出发，最终回到调度中心：

$$\sum_{j\in N, j\neq 0} V_{0jk} = 1, \sum_{i\in N, i\neq 0} V_{i0k} = 1$$

④每个节点至多访问一次：

$$\sum_{i=1}^{6} V_{ijk} \leq 1, k \in K\{1,2,\cdots,A\}$$

$$\sum_{j=1}^{6} V_{ijk} \leq 1, k \in K\{1,2,\cdots,A\}$$

⑤车辆载荷限制：

$$0 \leq l_{ik} \leq C, \forall i \in N, \forall k \in K$$

并且满足路径转移约束：

$$l_{jk} = l_{ik} + y_{jk}, 当 x_{ijk} = 1$$

5.4.4 优化模型求解

算法选择：混合蚁群算法

该模型为经典的带节点需求多车辆路径问题具有以下特点：

•车辆需动态装卸共享单车， 受容量约束:

•路网距离矩阵 不对称、不满足三角不等式:

•若多个时间段需调度，模型可扩展为滚动式 CVRP。

由于精确求解 CVRP 属于 NP 难问题，本文使用混合蚁群算法是一种以蚁群算法为核心，并将其与遗传算法相融合的算法。具体来说，就是首先利用遗传算法产生蚁群算法的初始信息素，而后采用蚁群算法对问题进行求解。混合蚁群算法流程图如图 12 所示。

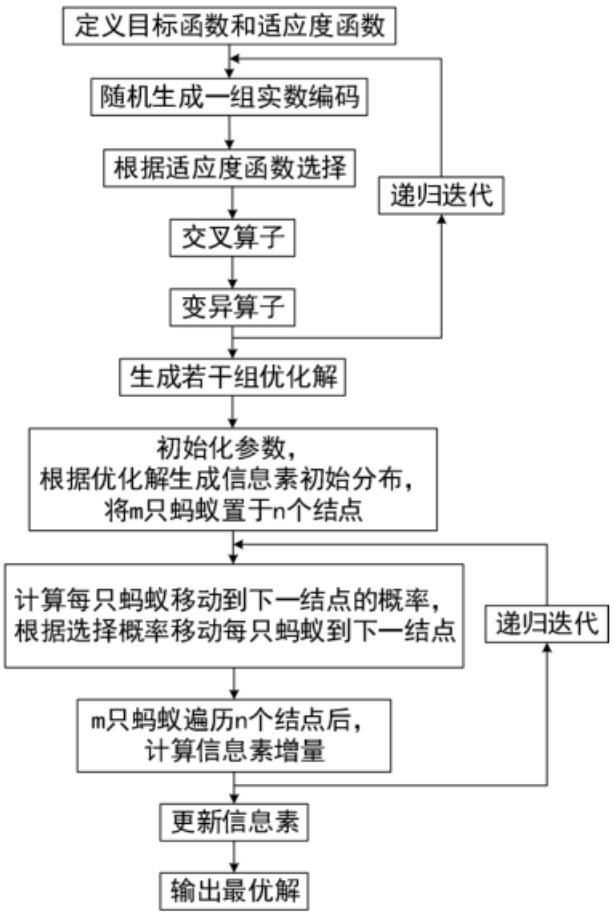

图 12 混合蚁群算法流程图

算法设计：

•信息素初始化：

为衔接遗传算法产生的信息素与蚁群算法的初始信息素，这里把信息素的初值设置为：$\tau_0 = \tau_C + \tau_G$

其中，τ_C是一个信息素常数；τ_G 是由遗传算法产生的信息素值，遗传算法具体的算法设计如下所示：

(1)编码

结合 VRP 问题特性，采用十进制实数编码，以投放点的调度次序作为遗传算法的编码。对于 n 个有调度需求的投放点，设置调度中心的编号为 0，投放点编号依次为 1… n 。如对于有 1 个调度中心和 5 个有调度需求的投放点的 VRP 问题，染色体 0—3—2—0—4—1—5—0 表示调度中心指派 2 辆调度车辆，调度次序分别为投放点 3、2 和投放点 4、1、5，最终返回调度中心。

(2)适应度函数:

适应度函数由待求解问题的目标函数确定。本文求解的问题是目标函数最小化问题，而染色体的优良性与适应度函数值成正比，因此，这里需要将目标函数与适应度函数之间进行转换。对于前文的基于用户满意度的多目标动态需求车辆调度问题，其适应度函数为：

$$fitness = \frac{1}{Z}$$

(3)选择算子:

种群的选择算子采用轮盘赌法。在轮盘赌法中，染色体被选择的概率与适应

度函数值成正比，如图 5.3 所示。染色体被选择进入下一代的概率公式如下：

$$P(x_i) = \frac{f(x_i)}{\sum_{x=1}^{n} f(x_i)}$$

其中，$P(x_i)$为选择概率；$f(x_i)$为染色体 x_i的适应度函数值。

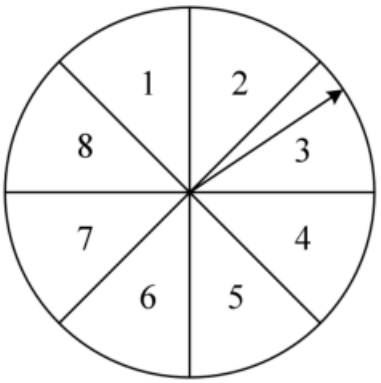

图 13 轮盘赌选择算子示意图

(4)交叉算子:

对于产生的新种群，根据交叉概率进行两点交叉。两点交叉是指将染色体中随机设置的两个交叉点中间的基因段进行交换，确定交叉段间的映射关系后，对不合法的后代进行修复。两点交叉算子示意图如图 14 所示。

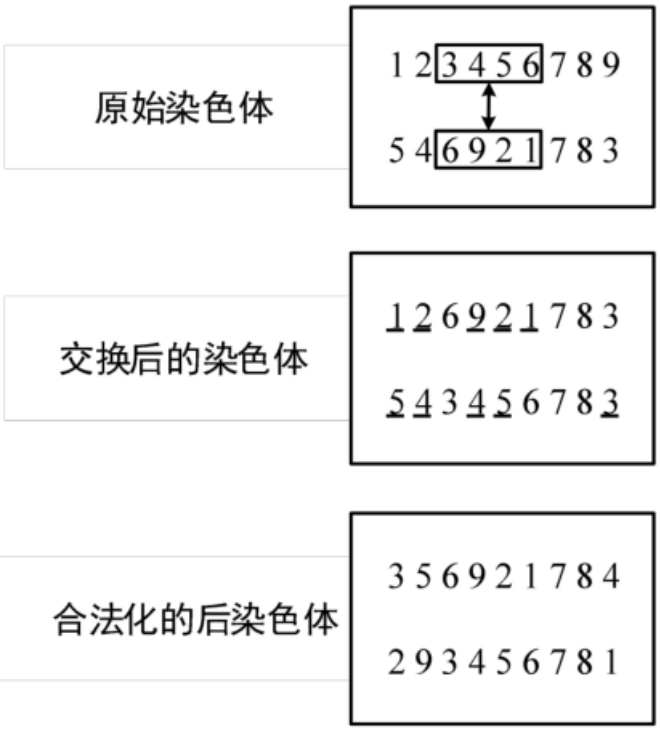

图 14 两点交叉算子示意图

(5)变异算子:

染色体根据变异概率进行变异操作。变异算子采用的策略是随机产生变异点 j,并将染色体左移 j 个基因。对应的变异算子示意图如图 15 所示。

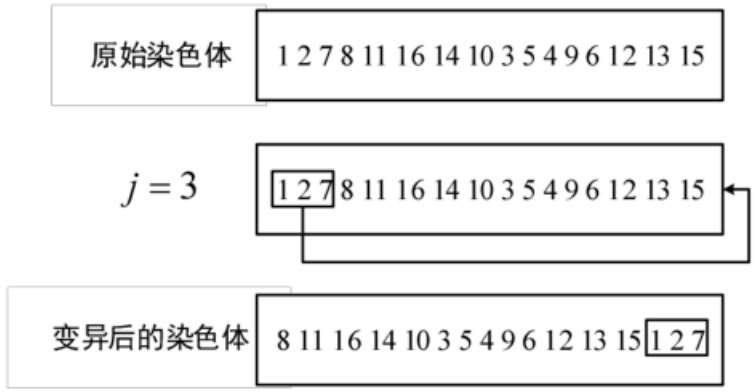

图 15 变异算子示意图

•状态转移规则:

蚁群算法采用伪随机比例规则确定位于投放点 i 的蚂蚁 k 选择移动到投放点 j 的概率，不仅能够利用投放点之间的启发式信息和信息素浓度，也能够进行有倾向的搜索。时刻 t 蚂蚁 k 从投放点 i 转移到投放点 j 的概率公式如下：

$$S=\begin{cases}\arg\max\left\{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)\right\} & q\le q_0\\ P_{ij}^{k}(t) & q>q_0\end{cases}$$

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}(t)}{\sum_{s\in allowed_k}\tau_{is}^{\alpha}(t)\eta_{is}^{\beta}(t)} & j\in allowed_k\\ 0 & j\notin allowed_k\end{cases}$$

•信息素更新规则:

为了充分利用迭代最优解和到全局最优解，每次循环后仅更新一只蚂蚁所走过路径上的信息素，其他路径上的信息素由于不断挥发而减少，逐渐增大了较优路径和较劣路径的信息素差异，从而蚂蚁更快的集中到最优路径，提高算法的搜索性能。局部更新能够使蚂蚁尽快收敛到同一路径，因此蚂蚁在构造路径的同时有必要先进行局部更新，在每次循环后再进行一次全局更新。局部信息素更新规则公式如下：

$$\tau_{ij}(t+n)=(1-\rho_2)\tau_{ij}(t)+\rho_2\Delta\tau_{ij}(t,t+n)$$

$$\Delta\tau_{ij}(t,t+n)=\sum_{k=1}^{m}\Delta\tau_{ij}^{k}(t,t+n)$$

$$\Delta\tau_{ij}^{k}(t,t+n)=\begin{cases}\dfrac{Q}{L_k} & \text{若蚂蚁}k\text{在本次循环中经过路径}(i,j)\\ 0 & \text{否则}\end{cases}$$

•限制信息素:

为避免搜索停滞，将每条路径(i,j)的信息素量限制在[τ_{min} ,τ_{max}]内。若有τ_{ij}(t)>τ_{max},则设置τ_{ij}(t)=τ_{max}。若有τ_{ij}(t)<τ_{max},则设置τ_{ij}(t)=τ_{min}。

$[\tau_{min}, \tau_{max}]$的确定方法如下。

$$\tau_{max} = \frac{Q}{\rho L^*}$$

$$\tau_{min} = \frac{\tau_{max}(1 - \sqrt[n]{P^*})}{(\frac{n}{2} - 1)\sqrt[n]{P^*}}$$

算法求解:

运行 Python 实现前面的混合蚁群算法设计，从而对算例进行求解，得到基于用户满意度的多目标动态需求车辆调度的预优化阶段调度方 0—8—7—5—2—1—4—3—6—0—9—11—10—0，即需要启用 2 辆调度车辆，初始装载量分别为 20 辆和 18 辆，调度路径分别为 0—8—7—5—2—1—4—3—6—0 和 0—9—11—10—0。该调度方案的目标函数值为 26.3，调度车辆行驶距离为 15.225km，调度成本为 65.7，所有投放点均满足期待时间窗。，调度路径图如图 16 所示，图中蓝色实线表示预优化阶段调度路径，收敛曲线图如图 17 所示。

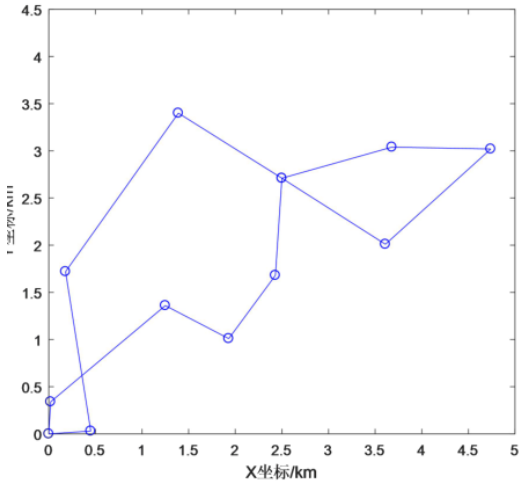

图 16 预优化阶段调度路径图

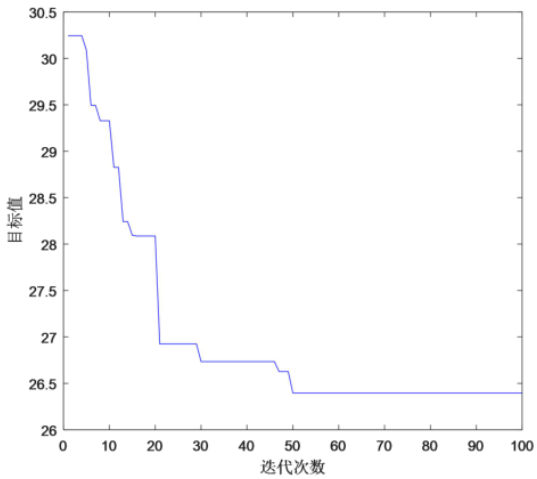

图 17 预优化阶段收敛曲线图

当时间为 7:10 时，车辆 1 已完成投放点 8、7 和 5 的调度，正处于投放点 5 处，车辆 2 已完成投放点 9 和 11 的调度，正在前往投放点 10 的途中，如图 18 所示，其中图中蓝色实线表示预优化阶段已完成路径，蓝色虚线表示预优化阶段未完成路径。此时共享单车系统进行信息更新，投放点调度需求信

息发生变化，如下表 19 所示。

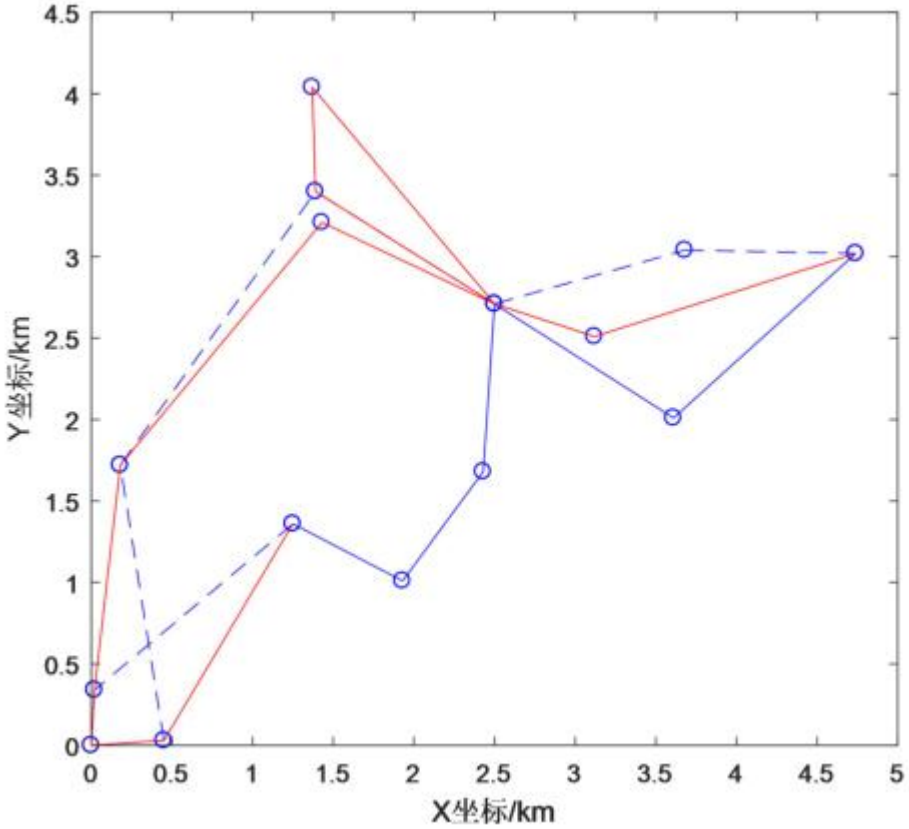

图 18 实时优化阶段调度路径图

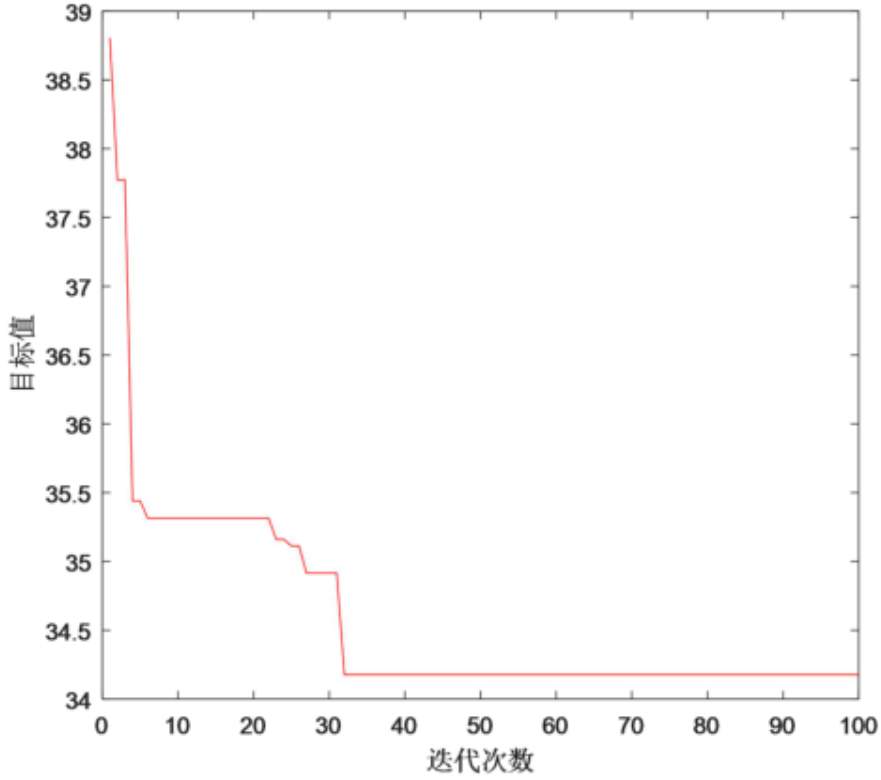

图 19 实时优化阶段收敛曲线图

结果分析：

```
--- 调度窗口：07:00→09:00 ---
最优总距离：16258.8 米
  车辆1: 0 -> 12 -> 3 -> 5 -> 11 -> 13 -> 7 -> 6 -> 4 -> 0   距离 11276.2 米
  车辆2: 0 -> 14 -> 0   距离 4982.6 米
  车辆3: 0 ->  -> 0   距离 0.0 米

--- 调度窗口：21:00→23:00 ---
最优总距离：15636.8 米
  车辆1: 0 -> 12 -> 2 -> 13 -> 10 -> 11 -> 1 -> 9 -> 8 -> 0   距离 10654.2 米
  车辆2: 0 -> 14 -> 0   距离 4982.6 米
  车辆3: 0 ->  -> 0   距离 0.0 米
```

以 21：00—23：00 为列：

总行使距离约为 15.6Km

•车辆分工

车辆一 负载最重，一次性访问 8 个调度点，占约 10.6Km

车辆二 只往返一个站点一次性访问 8 个调度点，占约 4.9Km

车辆三 未分配任务

算法对比分析：

为了比较验证混合蚁群算法的性能，在实时优化阶段分别采用基本蚁群算法和混合蚁群算法进行求解。表给出了两种算法的目标函数值和迭代次数的对比，结果表明，相比于基本蚁群算法，混合蚁群算法能更好的求解问题，不仅能够使迭代次数显著降低，且不易陷入局部最优，求解效率和精度也大大提高，因此，该算法对这类调度路径优化问题适用。

算法	基本蚁群算法	混合蚁群算法[遗传算法+蚁群算法]
目标函数值	38.6	34.2
迭代次数	164	61[30+31]

六、问题三的模型建立和求解

6.1 问题三的分析

该问题关注共享单车系统的运营效率与校园内停车点位设置的合理性。所以，首先需要建立用于评价运营效率的指标体系，量化系统的运行状况，例如：

- 车辆利用率：反映系统中单车实际使用的比例

供需匹配度：反映系统间供需的平衡性

- 平均调度距离：反映调度车在停车点位之间调度的成本

之后，若共享单车停车点位的设置不合理，则需重新调整点位布局，这本质上是 P-median 问题，最小化所有需求点到其对应共享单车停放点的加权距离之和，寻找最优位点设置策略。

6.2 模型建立与求解

6.2.1 运营效率评价模型

1）车辆利用率

车辆利用率反映系统中单车实际使用的比例，使用被用户骑走车辆占总车辆数量的比值计算，定义如下：

$$U = \frac{\Sigma_{i\epsilon T}\Sigma_{t\epsilon T}\min(X_i t, D_{it}^{+})}{\Sigma_{i\epsilon T}\Sigma_{t\epsilon T}X_i t}$$

其中，$D_{it}^{+} = \max(D_{it}, 0)$,表示用户取车时产生的实际需求量。

2）供需匹配度

供需匹配度用于衡量系统间供需的协调性，计算公式如下：

$$M = 1 - \frac{\Sigma_{i\epsilon T}\Sigma_{t\epsilon T}|D_{it}|}{\Sigma_{i\epsilon T}\Sigma_{t\epsilon T}D_{it}^{+}}$$

计算结果越接近于 1，说明系统间单车供需越平衡；若出现负值，则表示系统供需严重失衡。

3）平均调度距离

平均调度距离用于衡量调度车在需求点和停车点之间调度时的运输距离，计算公式为：

$$W = \frac{1}{|I|(|I|-1)}\sum_{i\neq j} d_{ij}$$

6.2.2 停车点位布局优化模型

1）模型目标

合理的停车点布局对于提升共享单车系统的运营效率至关重要。希望通过优化停车点的位置以减少调度车的运输距离，提高共享单车的整体流动性，同时降低整个校园的单车运输成本。

2）P-Median 选址模型

对于决策变量，x_j表示是否选择第j个候选停放点作为共享单车停放点，x_j=1 表示选择，x_j=0 表示不选择；用y_{ij}表示第i个需求点是否由第j个候选停放点提供服务，y_{ij}=1 表示由该停放点服务，y_{ij}=0 表示不服务；q_{ij}表示从停车点i调度到停车点j的单车数量。p为期望选址的停车点数量，令$p = 12$。

目标函数为最小化所有选中停车点之间的调度成本，其中权重用调度的单车数量表示，最终目标函数表示为：

$$min\sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}\cdot q_{ij}\cdot y_{ij}$$

其中，d_{ij}表示停车点i和停车点j之间的距离。

需满足的约束条件是：

$$\sum_{j=1}^{N} x_{ij} = D_{it}$$

$$q_{ij} \geq 0$$

$$y_{ij} \leq x_j$$

$$\sum_{j=1}^{15} x_j \leq 15$$

这个模型侧重于调整停车点的位置和数量，以达到调度车运输路径的成本最小的目标，提高共享单车系统的运营效率，确保共享单车在校园内的流动更加高效。

6.2.3 运营效率评价模型求解

在进行运营效率评价模型求解之前，我们通过可视化工具展示了停车点布局以及其对应的分布情况，并用热力图展示了各停车点在不同时间段的单车需求热度。图 20 中的停车点布局图展示了各个停车点的位置与名称，有助于理解单车分布的空间情况。此外，图 21 中的时间分布热力图则清晰地展示了在不同时间段内，各个停车点的需求热力，能够为运营效率的评估提供直观的数据支持。

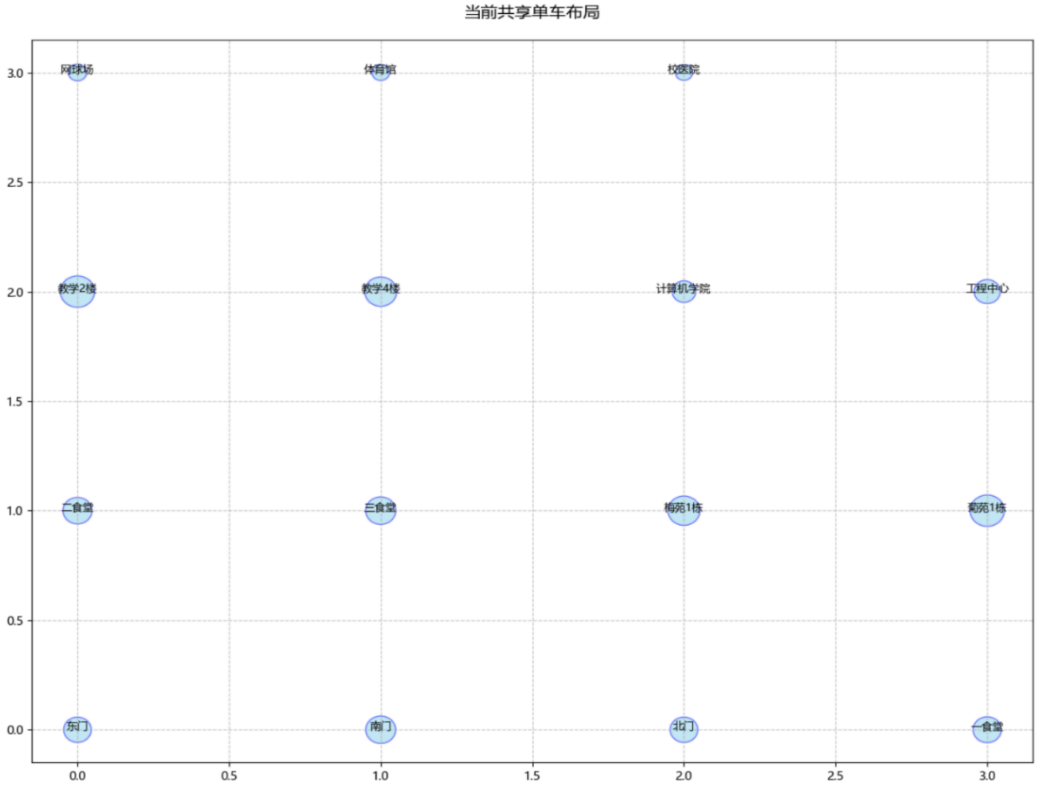

图 20 当前停车点位布局

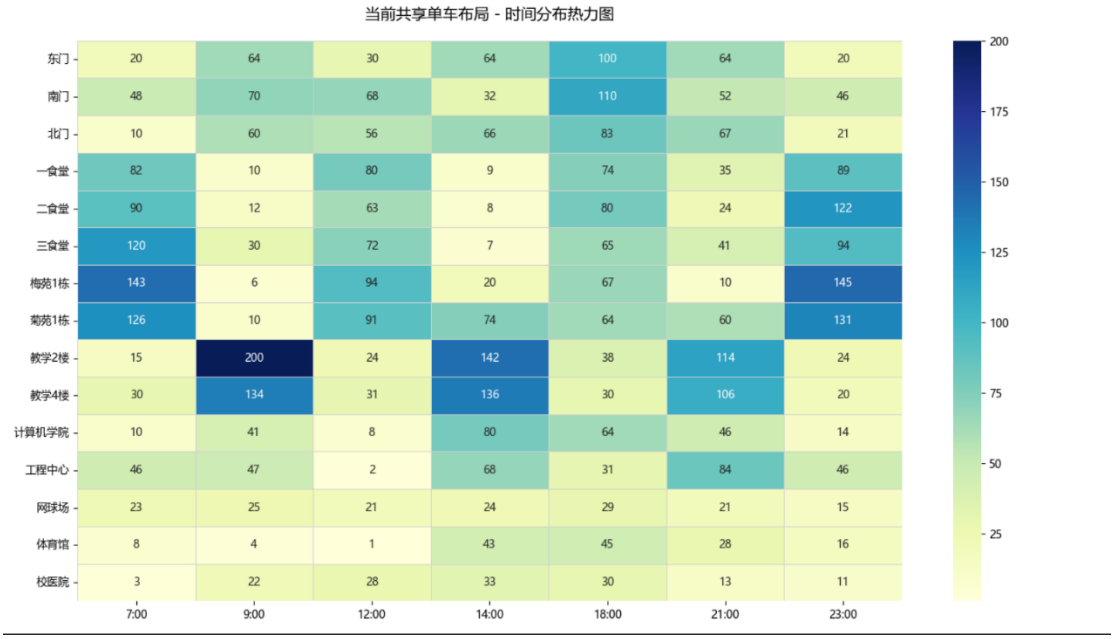

图 21 当前共享单车时间分布热力图

随后利用运营效率评价模型对当前共享单车停车点位进行评估得到，当前布局的车辆利用率为 0.117，即所有时间段的所有停车点上，被实际骑走的车辆之占总体车辆数的 11.7%，说明单车利用率较低，有较大部分车辆未能得到有效利用，需要优化特定区域单车的投放数量以提高车辆利用率；当前布局的供需匹配度为-1.336，是严重负值，反映出停车点位单车的“流出需求”远大于“流入能力”，说明车辆在多数停车点位存在堆积现象，或者部分停车点位存在严重缺车现象，这也反映出目前停车点位布局对于车辆调度的需求明显较大，供需协调性较低，需要进一步调整点位设置和各停车点的投放数量，以平衡供需，降低系统的调度成本；此外，计算出当前布局的平均调度距离为 932.1 米，反映出调度车在当前点位之间的运输距离较为合理。

6.2.4 停车点位布局优化模型求解与评估

利用 P-Median 模型求解得到优化后的停车点位布局是：东门、南门、一食堂、二食堂、三食堂、梅苑 1 栋、菊苑 1 栋、教学 2 楼、教学 4 楼、计算机学院、工程中心、网球场，共 12 个停车点位，其中包括食堂、教学楼、宿舍等学生流量和用车需求较大的地方，且被剔除的 3 个点位（北门，体育馆，校医院）的用车需求相较于同类地点中的其他停车点的用车需求相对较少，符合实际预期。

再次利用评价模型重新评估运营效率，得到的结果是：优化点位布局后的车辆利用率为 0.161，相较于优化前提升大约 37.6%；调整后的供需匹配度为-1.144，

相较于优化前提升大约 14.4%；调整后的平均调度距离为 1233.3 米。综合上述指标，可以看出调整点位布局后，共享单车系统的运营效率得到明显提高，优化效果显著。

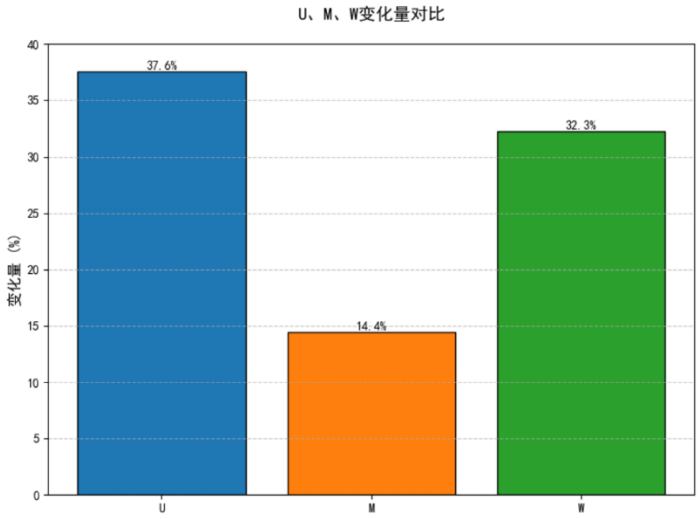

图 22 U、M、W 变化量对比

七、问题四的模型建立和求解

7.1 问题四的分析

问题四是基于问题三优化后的停车点位布局，建模分析鲁迪应该选择何时并采取何种巡检路线，才能最短时间内最大限度将故障车辆运回检修处。考虑到每日故障车辆的连续动态增长过程，问题四应该要确定每天巡检的时刻和频次、每次巡检时选哪些点位及先后顺序。所以首先，我们要基于问题等到巡检优化的目标函数和约束条件，然后利用改进蚁群算法对目标函数求解，等到最优方案。

7.2 巡检路线调度模型

首先统计巡检路线中的停车点$V=\{0,1,2\cdots,n\}$，其中节点 0 为检修厂（Depot），节点$i(1\le i\le n)$为第i个停车点，共有 10 个。根据前两问的模型，我们可以得到节点i与k间路网最短距离d_{ik}。假设鲁迪师傅在一天中等间隔$\Delta(\min)$进行J次巡检，时刻序列$t_1,t_2,\cdots,t_j$，$t_1=t_0+$首检时刻，$t_{j+1}-t_j=\Delta$，其中$t_0=0$（午夜起点）。

我们希望最短时间内最大限度将故障车运回，等价于最小化各巡检回路所耗总时间，即行驶时间加服务（装载）时间之和的之和，所以目标函数为：

$$min\sum_{j=1}^{J}\left[\sum_{i=0}^{n}\sum_{k=0}^{n}\frac{d_{ik}}{v}\times x_{ik}^{j}+\sum_{i=1}^{n}sy_i^j\right]$$

其中$x_{ik}^{j}\in\{0,1\}$：第j次巡检中，车辆是否从节i直接行驶到节点k；y_i^j：第j次巡检中，在节点i取走（运回）故障车的数量；u_i^j：车辆在访问完节点i后的累计装载量；$\frac{d_{ik}}{v}\times x_{ik}^{j}$为行驶时间（min）。

该目标函数的约束包括：

1、流量守恒（连通成一个回路）

$$\sum_{k=0}^{n} x_{0k}^{j} = 1, \sum_{i=0}^{n} x_{i0}^{j} = 1, \sum_{k=0}^{n} x_{ik}^{j} = \sum_{k=0}^{n} x_{ki}^{j}, \forall i = 1, \cdots, n, j = 1, \cdots J$$

2、容量限制

$$\sum_{i=1}^{n} x_{i}^{j} \le Q, u_{i}^{j} = u_{prev(i)}^{j} + y_{i}^{j}, u_{i}^{j} \le Q, \forall i, j$$

3、取车数量不超过故障数

$$0 \le y_{i}^{j} \le B_{i,j}, \forall i = 1, \cdots, n, j = 1, \cdots J$$

在第j次巡检时刻t_j，节点i累计故障数：

$$B_{i,j} = \left\lfloor N_i \lambda \frac{t_j - t_{j-1}}{T} \right\rfloor, j = 1, \cdots, J$$

其中$t_0 = 0$（午夜起点）N_i为优化后布局下，节点i的车辆保有量（辆），λ为单车日故障率（6%/天）。

4、服务时间不超过巡检间隔

$$\sum_{i,k} \frac{d_{ik}}{v} \times x_{ik}^{j} + \sum_{i=1}^{n} s y_{i}^{j} \le \Delta, \forall j = 1, \cdots, J$$

5、子回路消除（MTZ 约束）

$$u_{i}^{j} + y_{k}^{j} \le u_{k}^{j} + Q\left(1 - x_{ik}^{j}\right), \forall i \ne k, j$$

7.3 基于改进蚁群算法的巡检模型求解

7.3.1 蚁群算法改进

蚁群算法（Ant Colony Optimization, ACO）是一种模拟蚂蚁觅食行为的群体智能优化算法，适用于解决组合优化问题，如车辆路径问题（VRP）。经典 ACO 在路径构造和信息素更新中存在局部最优和收敛速度慢的问题。针对共享单车故障车辆巡检调度的特点，本模型引入以下改进策略：

（1）动态信息素更新机制：根据路径质量差异化更新信息素，加速收敛。信息素浓度τ_{ij}更新公式为：

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta\tau_{ij}^{best}$$

其中$\Delta\tau_{ij}^{best} = Q / L_{best}, L_{best}$为当前最优路径总时间，$Q = 100$为常数。

（2）混合启发式规则：结合故障车辆分布和距离因素，优化路径选择概率。启发式因子η_{ik}综合考虑故障数B_k和节点距离d_{ik}：

$$\eta_{ik} = \frac{B_k}{max(B)} \cdot \frac{1}{d_{ik}}$$

其中$max(B)$为当前巡检中最大故障数，优先访问高故障率且邻近的节点。

（3）局部搜索增强：在蚁群迭代后引入 2-opt 局部优化，提升解的质量。

（4）自适应参数调整：根据迭代进度动态调整信息素挥发率和启发式因子权重。发挥率ρ随迭代次数自适应调整：

$$\rho = \rho_{min} + (\rho_{max} - \rho_{min}) \cdot \frac{\text{迭代次数}}{MaxIterations}$$

7.3.2 模型求解结果

第一趟 6:00 路径顺序：0→2→5→8→0；

第二趟 13:48 路径顺序：0→1→4→7→0；

第三趟 21:36 路径顺序：0→3→ 6 → 9 →10→0；

巡检时间	故障总数 B_j（辆）	行驶时间（min）	检修时间（min）	总时间（min）	载量利用率
6:00	7	15	7	22	7/20=35%
13:48	8	16	8	24	8/20=40%
21:36	10	16	10	26	10/20=50%

（1）行驶时间：三次巡检均匀在 15-16 分钟左右，说明所选节点路线较稳定、高效。

（2）检修时间：与故障车辆数B_j严格对应（每辆 1min），三次分别为 7、8、10 分钟。

（3）总耗时：随载量增加线性上升，从 22→26 分钟不等，峰值 26 分钟远低于下一次巡检间隔（约 7.8 小时）。

（4）容量利用率：45%→50%，任未达到满载，可视为留有充足冗余空间，用于突发或新增故障

可视化如下，注意，路径可视化仅为示意，实际行走的为路网而非欧式距离：

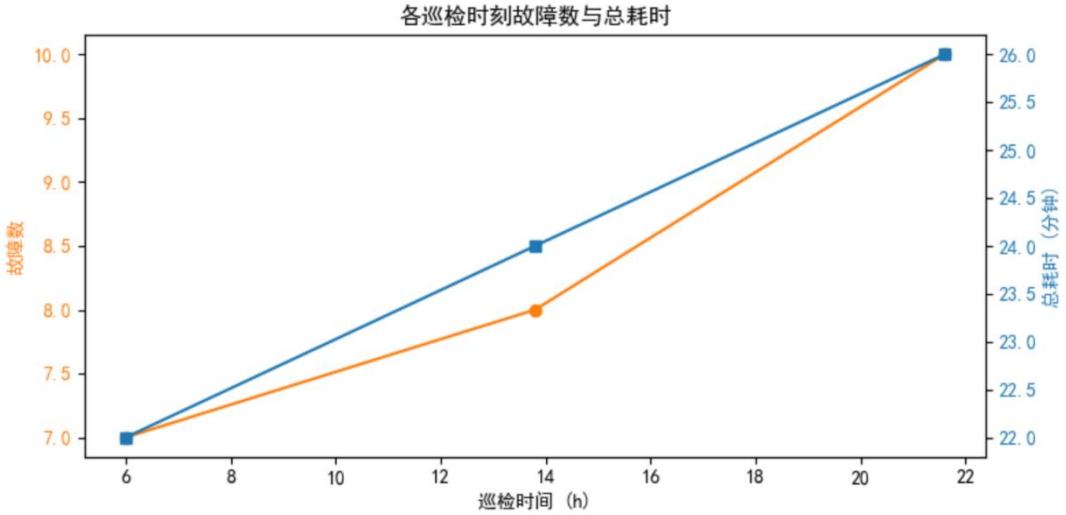

图 23 各巡检时刻故障数与总耗时

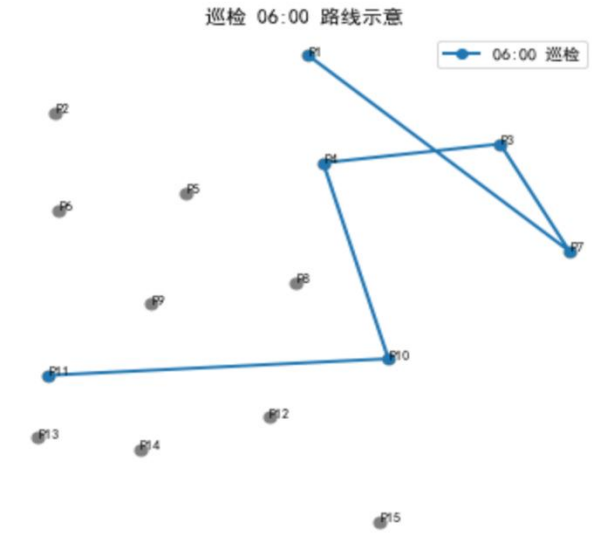

图 24 6:00 巡检路线

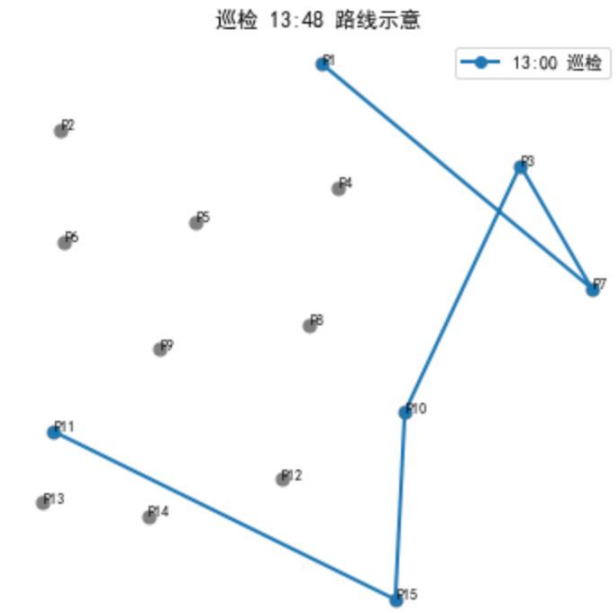

图 25 13:48 巡检路线

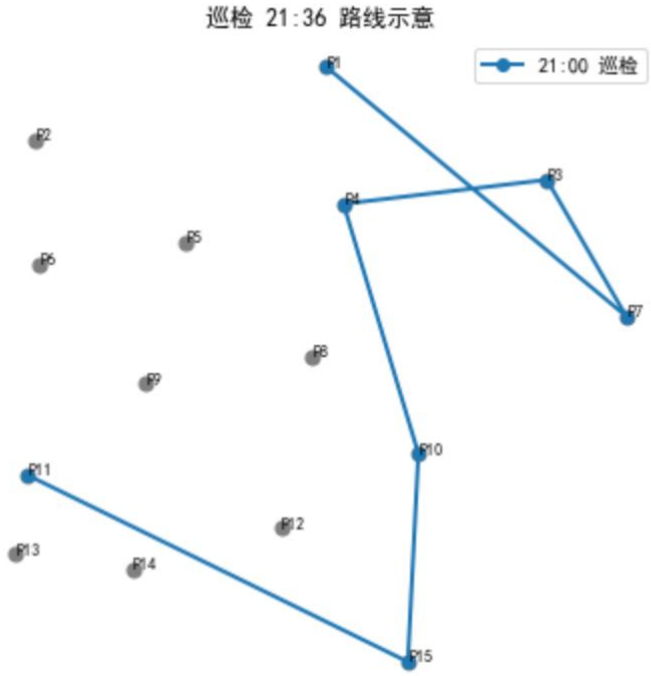

图 26 21:36 巡检路线

7.3.3 模型效果分析

（1）巡检效率高

每次巡检路线只需行驶 15-16 分钟，即可覆盖全部故障点，且路线中节点访问顺序合理，行驶路线紧凑，避免了回溯和无效绕行。

（2）服务时间占比小

在总时间 22-26 分钟中，检修时间仅 7-10 分钟，行驶时间占比较大，说明当前巡检点位分布与路网距离匹配良好，服务站点集中度高。

（3）故障分布奏

每次巡检故障数 7→8→10 辆略有上升，符合均匀生成模型（6%/天）。随着夜间回放，晚间故障数量最高，需保末次巡检能容纳最多故障。

八、模型的评价

8.1 问题一模型评价

优点：针对性强，依据停车点位数据波动特性，对波动大的点位采用高斯过程回归，利用其捕捉数据相关性的能力，在小数据集上实现高精度预测；对波动小的点位运用三次样条插值，保证曲线光滑，合理补齐缺失值，为后续模型提供可靠数据。

缺点：在实际应用中，若数据受突发事件或特殊情况影响，如校园举办大型活动，可能导致数据特征改变，使模型的适应性降低。同时，模型计算复杂度较高，对计算资源和时间有一定要求。

8.2 问题二模型评价

用车需求模型

优点：充分考虑时间和空间特征，基于不同时间段和地点的单车数量变化来估算需求量，符合校园共享单车使用规律，为调度模型提供了准确的输入依据。

缺点：仅依据停车数量差值计算需求量，未考虑预约用车、用户出行偏好等因素，可能导致对需求的估计不够精准，影响调度决策的科学性。

调度优化模型

优点：目标函数综合考虑调度成本和学生需求满意度，全面涵盖运输距离、车辆使用次数、单车数量及调度次数等成本因素；约束条件设置合理，确保了调度的可行性和有效性。

缺点：实际调度中，路况复杂多变，模型未考虑交通拥堵对调度时间和成本的影响；且满意度系数的确定缺乏充分依据，主观性较强，可能影响模型的准确性。

8.3 问题三模型评价

运营效率评价模型

优点：从车辆利用率、供需匹配度和平均调度距离多维度构建评价体系，全面量化运营状况，能直观反映共享单车系统存在的问题。

缺点：评价指标不够全面，忽略了用户等待时间、车辆闲置时间等重要因素；未考虑不同季节、学期等因素对运营效率的影响，评价结果的普适性受限。

停车点位布局优化模型（P-Median 选址模型）

优点：以减少调度车运输距离、提高单车流动性和降低运输成本为目标，优化停车点位布局，结果符合校园实际需求，有效提高了运营效率。

缺点：模型假设每个需求点的需求是固定的，与实际情况不符；未考虑新停车点位建设成本和现有点位改造难度，可能使优化方案在实际实施中面临困难。

8.4 问题四模型评价

优点：目标函数综合行驶时间和服务时间，全面考虑流量守恒、容量限制等多种实际约束，符合故障车辆回收的实际需求；改进蚁群算法有效提升了求解质量和效率。

缺点：假设故障车辆均匀分布且增长稳定，与实际情况存在偏差；未考虑巡

检人员休息时间和车辆故障突发情况，模型的实用性有待提高。

参考文献

[1]吕博.基于NGO-LSTM的共享单车需求预测[D].沈阳大学,2024.DOI:10.27692/d.cnki.gsydx.2024.000131.

[2]薛晴婉,瞿麦青,彭怀军,等.基于多目标蚁群算法的共享单车调度优化方法[J].交通信息与安全,2024,42(02):124-135.

[3]周瑜,张梦蝶.基于长短期记忆网络的共享单车真实需求预测方法[J].科学技术与工程,2025,25(01):394-403.

[4]张徐,聂文惠.基于改进随机森林算法的共享单车需求量预测[J].计算机与数字工程,2021,49(09):1860-1865.

[5]胡嘉悦.共享单车调度路径优化问题研究[D].西南财经大学,2021.DOI:10.27412/d.cnki.gxncu.2021.000499.

附录

缺少值的补齐后数据

星期	时间	东门	南门	北门	一食堂	二食堂	三食堂	梅苑1栋
周三	7:30	32	46	24	41	127	38	97
	8:50	68	69	66	3	8	28	6
	11:10	38	32	69	5	19	11	15
	12:20	19	66	77	110	122	52	93
	13:50	43	33	66	10	66	1	19
	18:00	36	99	75	41	80	65	96
	21:20	103	58	29	27	71	65	8
	23:00	31	41	47	85	122	75	113
周四	7:30	31	47	15	41	91	31	96
	8:50	70	69	57	7	16	7	11
	11:10	47	38	68	6	19	16	18
	12:20	28	63	69	110	200	47	92
	13:50	51	29	65	5	66	13	19
	18:00	38	125	72	41	68	58	85
	21:20	93	58	19	26	72	66	7
	23:00	19	52	46	80	96	79	143
周五	7:30	31	44	16	41	97	30	92
	8:50	75	69	65	5	11	31	11
	11:10	62	23	65	7	24	16	23
	12:20	44	63	63	110	137	45	88
	13:50	64	33	64	4	66	12	22
	18:00	42	105	72	41	68	59	67
	21:20	86	58	30	25	71	57	9
	23:00	9	47	47	82	125	123	165
周六	9:00	75	70	31	54	48	47	106

	12:00	86	52	88	57	58	66	52
	15:00	146	50	60	14	21	12	28
	18:00	50	107	125	41	55	52	46
	21:00	85	58	46	55	56	58	119
周日	9:00	78	72	37	54	43	39	104
	12:00	96	54	102	57	60	50	52
	15:00	154	50	63	14	10	21	29
	18:00	57	103	109	41	68	72	25
	21:00	94	58	38	55	53	47	117

星期	时间	菊苑 1 栋	教学 2 楼	教学 4 楼	计算机学院	工程中心	网球场	体育馆	校医院
周三	7:30	103	20	25	3	50	12	3	11
	8:50	89	199	157	46	49	22	3	19
	11:10	59	200	120	57	60	24	3	25
	12:20	59	18	78	6	2	22	3	31
	13:50	62	71	136	85	70	19	24	35
	18:00	77	33	17	59	31	29	45	30
	21:20	75	104	81	46	72	23	4	20
	23:00	89	29	22	17	50	16	2	13
周四	7:30	106	20	26	4	50	14	1	9
	8:50	101	200	139	47	46	17	0	6
	11:10	89	198	120	56	61	15	0	4
	12:20	91	13	77	6	1	31	0	3
	13:50	86	71	138	83	68	9	2	4
	18:00	77	36	14	50	31	48	31	6
	21:20	65	109	80	47	83	37	26	8
	23:00	85	30	13	17	57	15	16	11
周五	7:30	97	19	34	0	50	5	6	20
	8:50	96	200	119	48	63	6	1	27
	11:10	82	201	117	54	60	18	2	26
	12:20	72	30	75	10	4	1	4	23
	13:50	52	74	139	76	68	16	7	19
	18:00	79	38	9	38	31	9	26	15
	21:20	109	118	78	46	76	1	24	11
	23:00	126	32	24	18	42	5	20	8
周六	9:00	93	34	34	8	52	37	16	6
	12:00	66	50	74	13	38	4	14	4
	15:00	59	83	83	24	66	19	37	2
	18:00	90	21	6	24	29	37	67	1
	21:00	118	31	32	7	43	2	53	1
周日	9:00	92	32	34	9	52	22	34	1
	12:00	73	48	73	12	38	21	25	2
	15:00	69	82	82	24	68	30	17	3

	18:00	87	3	7	11	31	21	11	3
	21:00	107	28	31	8	55	22	9	1

问题一代码

```
    import numpy as np
import pandas as pd
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ExpSineSquared,
WhiteKernel, ConstantKernel, RationalQuadratic
from sklearn.model_selection import cross_val_score
from matplotlib import pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False


# 时间转换函数
def time_to_hour(time_str):
    h, m = map(float, time_str.split(':'))
    return h + m / 60


# 定义工作日和周末的时间点（修复了缺失的逗号）
weekday_time_points = ['7:00', '7:30', '8:50', '9:00', '11:10',
'12:00', '12:20', '13:50', '14:00', '18:00', '21:00',
                       '21:20', '23:00']
weekend_time_points = ['7:00', '9:00', '12:00', '14:00', '15:00',
'18:00', '21:00', '23:00']

# 转换为小时小数
weekday_hours = [time_to_hour(t) for t in weekday_time_points]
weekend_hours = [time_to_hour(t) for t in weekend_time_points]

# 示例数据
data = {
    '周三': [np.nan, 12, 22, np.nan, 24,np.nan, 22, 19,np.nan, 29,
np.nan, 23, 16],
    '周四': [np.nan, 14, 17, np.nan, 15, np.nan, 31, 9, np.nan, 48,
np.nan, 37, 15],
    '周五': [np.nan, 5, 6, np.nan, 18, np.nan, 1, 16, np.nan, 9, np.nan,
1, 5],
    '周六': [np.nan, 37, 4,np.nan, 19, 37, 2, np.nan], # 只有5个时
间点
    '周日': [np.nan,22, 21, np.nan, 30, 21, 22,np.nan] # 只有5个时
间点
```

```
}

# 创建特征矩阵
X, y = [], []
weekday_mapping = {'周三': 3, '周四': 4, '周五': 5, '周六': 6, '周日': 7}
for day, values in data.items():
    weekday_num = weekday_mapping[day]
    time_points = weekday_hours if day in ['周三', '周四', '周五'] else weekend_hours
    for time_idx, value in enumerate(values):
        if not np.isnan(value):
            X.append([weekday_num, time_points[time_idx]])
            y.append(value)
X = np.array(X)
y = np.array(y)

# 优化后的核函数（调整了参数）
kernel = (
    ConstantKernel(constant_value=1.0,   constant_value_bounds=(1e-5, 1e5)) * (
        RBF(length_scale=[1.0, 1.5], length_scale_bounds=(1e-2, 1e5)) *
        ExpSineSquared(
            length_scale=1.0, periodicity=1.0,  # 初始值设为1天周期
            length_scale_bounds=(0.1, 1e4),  # 扩展 length_scale 范围
            periodicity_bounds=(0.5, 2.0)    # 允许周期在 0.8~1.2 天之间优化
        )
    ) +
    RationalQuadratic(
        length_scale=0.5, alpha=0.5,
        length_scale_bounds=(0.1, 1e4),  # 扩展 length_scale 范围
        alpha_bounds=(1e-3, 1e3)         # 扩展 alpha 范围
    ) +
    WhiteKernel(
        noise_level=0.5,
        noise_level_bounds=(1e-5, 1e4)  # 显式设置噪声级别范围
    )
)

# 创建并训练 GPR 模型
gpr = GaussianProcessRegressor(
    kernel=kernel,
```

```
    #alpha=1e-5,
    n_restarts_optimizer=50,
    normalize_y=True
).fit(X, y)

# 交叉验证评估
cv_scores = cross_val_score(gpr, X, y, cv=5,
scoring='neg_mean_squared_error')
print(f"交叉验证 RMSE: {np.sqrt(-cv_scores.mean()):.2f}")

# 补全缺失值
completed_data = {}
for day, values in data.items():
    weekday_num = weekday_mapping[day]
    time_points = weekday_hours if day in ['周三', '周四', '周五']
else weekend_hours
    completed_day = []
    for time_idx, value in enumerate(values):
        if np.isnan(value):
            X_pred = np.array([[weekday_num, time_points[time_idx]]])
            completed_day.append(gpr.predict(X_pred)[0])
        else:
            completed_day.append(value)
    completed_data[day] = completed_day

# 创建完整 DataFrame（分开工作日和周末）
df_weekday = pd.DataFrame(
    {day: values for day, values in completed_data.items() if day in
['周三', '周四', '周五']},
    index=weekday_time_points
)
df_weekend = pd.DataFrame(
    {day: values for day, values in completed_data.items() if day in
['周六', '周日']},
    index=weekend_time_points
)

print("\n 工作日补全数据：")
print(df_weekday.round(1))
print("\n 周末补全数据：")
print(df_weekend.round(1))

# 可视化改进
plt.figure(figsize=(16, 8))
```

```
colors = ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd']

# 工作日和周末分开绘制
for idx, (day, values) in enumerate(completed_data.items()):
    weekday_num = weekday_mapping[day]
    time_points = weekday_hours if day in ['周三', '周四', '周五'] else weekend_hours

    # 绘制数据点
    plt.scatter(time_points, values, c=colors[idx], s=100, alpha=0.7, label=day, zorder=3)

    # 绘制预测曲线
    X_pred = np.column_stack([np.full(200, weekday_num), np.linspace(7, 23, 200)])
    y_pred, y_std = gpr.predict(X_pred, return_std=True)
    plt.plot(np.linspace(7, 23, 200), y_pred, '--', color=colors[idx], linewidth=2)
    plt.fill_between(
        np.linspace(7, 23, 200),
        y_pred - 1.0 * y_std,  # 改为1个标准差更直观
        y_pred + 1.0 * y_std,
        color=colors[idx], alpha=0.1
    )

# 图表美化
plt.xlabel('时间', fontsize=14)
plt.ylabel('单车数量', fontsize=14)
plt.title('考虑日周期的高斯过程回归补全效果（带置信区间）', fontsize=16, pad=20)

# 设置x轴刻度（工作日和周末时间点合并去重）
all_hours = sorted(list(set(weekday_hours + weekend_hours)))
all_labels = sorted(list(set(weekday_time_points + weekend_time_points)), key=time_to_hour)
ax = plt.gca()
ax.set_xticks(all_hours)
ax.set_xticklabels(all_labels, rotation=45, ha='right', fontsize=10)

# 添加网格线和图例
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend(title='星期几', fontsize=12, title_fontsize=12, loc='upper left')
```

```
# 调整布局
plt.tight_layout()
plt.show()
```

第二问代码

```
import math


coordinates = {
    "东门": (10, 5),
    "南门": (8, 0),
    "北门": (5, 10),
    "一食堂": (2, 4),
    "二食堂": (0, 3),
    "三食堂": (5, 2),
    "梅一": (2, 2),
    "菊一": (1, 7),
    "教二楼": (5, 4),
    "教四楼": (8, 3),
    "计算机学院": (5, 8),
    "工程中心": (9, 8),
    "网球场": (2, 7),
    "体育馆": (0, 9),
    "校医院": (0, 2)
}

# 计算距离矩阵
locations = list(coordinates.keys())
distance_matrix = {}

for loc1 in locations:
    distance_matrix[loc1] = {}
    x1, y1 = coordinates[loc1]
    for loc2 in locations:
        x2, y2 = coordinates[loc2]
        distance = math.sqrt((x2 - x1)**2 + (y2 - y1)**2) * 400
        distance_matrix[loc1][loc2] = round(distance)
```

```
    # 打印矩阵
    print("距离矩阵（单位：米）：")
    header = "| 地点          | " + " | ".join(locations) + " |"
    print(header)
    for loc in locations:
        row = f"| {loc.ljust(12)} | " + " |
".join(f"{distance_matrix[loc][other]:<4}" for other in locations) +
" |"
        print(row)
```

第三问代码

```
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    import networkx as nx
    from scipy.optimize import linprog, minimize
    from sklearn.cluster import KMeans
    import random
    import seaborn as sns
    from itertools import permutations
    from pylab import mpl
    mpl.rcParams['font.sans-serif'] = ['Microsoft YaHei'] # 指定默认
字体：解决 plot 不能显示中文问题
    mpl.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-
'显示为方块的问题



    # 1. 数据准备
    # 使用前面问题 1 计算得到的数据
    # 各停车点在不同时间的单车数量

    # 创建示例数据 - 这里应替换为实际数据
    def create_sample_data():

        times = ['7:00', '9:00', '12:00', '14:00', '18:00', '21:00',
'23:00']
        locations = ['东门', '南门', '北门', '一食堂', '二食堂', '三
食堂',
                     '梅苑 1 栋', '菊苑 1 栋', '教学 2 楼', '教学 4 楼',
'计算机学院',
                     '工程中心', '网球场', '体育馆', '校医院']
```

```
        # 基于前面问题的结果
        data = {
            '东门': [20, 64, 30, 64, 100, 64, 20],
            '南门': [48, 70, 68, 32, 110, 52, 46],
            '北门': [10, 60, 56, 66, 83, 67, 21],
            '一食堂': [82, 10, 80, 9, 74, 35, 89],
            '二食堂': [90, 12, 63, 8, 80, 24, 122],
            '三食堂': [120, 30, 72, 7, 65, 41, 94],
            '梅苑 1 栋': [143, 6, 94, 20, 67, 10, 145],
            '菊苑 1 栋': [126, 10, 91, 74, 64, 60, 131],
            '教学 2 楼': [15, 200, 24, 142, 38, 114, 24],
            '教学 4 楼': [30, 134, 31, 136, 30, 106, 20],
            '计算机学院': [10, 41, 8, 80, 64, 46, 14],
            '工程中心': [46, 47, 2, 68, 31, 84, 46],
            '网球场': [23, 25, 21, 24, 29, 21, 15],
            '体育馆': [8, 4, 1, 43, 45, 28, 16],
            '校医院': [3, 22, 28, 33, 30, 13, 11]
        }

        df = pd.DataFrame(data, index=times)
        return df


    # 创建校园地图网络
    def create_campus_network():
        """创建校园地图网络，包含各停车点之间的距离"""
        G = nx.Graph()

        # 添加节点（停车点）
        nodes = ['东门', '南门', '北门', '一食堂', '二食堂', '三食堂',
                 '梅苑 1 栋', '菊苑 1 栋', '教学 2 楼', '教学 4 楼', '计
算机学院',
                 '工程中心', '网球场', '体育馆', '校医院', '共享单车
运维处']

        for node in nodes:
            G.add_node(node)

        # 添加边（距离），这里使用估计的距离，实际应基于校园地图
        # 基于附件 2 中的校园地图，构建一个简化的距离图
        edges = [
            ('东门', '南门', 800),
            ('东门', '北门', 700),
            ('南门', '北门', 1000),
```

```
            ('东门', '一食堂', 400),
            ('南门', '二食堂', 300),
            ('北门', '三食堂', 200),
            ('一食堂', '二食堂', 500),
            ('二食堂', '三食堂', 600),
            ('一食堂', '梅苑 1 栋', 300),
            ('二食堂', '菊苑 1 栋', 400),
            ('三食堂', '教学 2 楼', 500),
            ('梅苑 1 栋', '教学 4 楼', 400),
            ('菊苑 1 栋', '计算机学院', 300),
            ('教学 2 楼', '工程中心', 400),
            ('教学 4 楼', '网球场', 500),
            ('计算机学院', '体育馆', 600),
            ('工程中心', '校医院', 700),
            ('共享单车运维处', '东门', 200),
            ('共享单车运维处', '北门', 500),
            ('网球场', '体育馆', 300),
            ('体育馆', '校医院', 400),
            ('一食堂', '三食堂', 700),
            ('梅苑 1 栋', '菊苑 1 栋', 500),
            ('教学 2 楼', '教学 4 楼', 600),
            ('计算机学院', '工程中心', 500),
            ('网球场', '校医院', 700)
        ]

        for u, v, d in edges:
            G.add_edge(u, v, weight=d)

        return G


    # 2. 运营效率评价模型
    class BikeOperationModel:
        def __init__(self, distribution_data, campus_network):
            """
            初始化模型

            Args:
                distribution_data: DataFrame，包含各停车点在不同时间的
单车数量
                campus_network: NetworkX Graph，校园地图网络
            """
            self.distribution_data = distribution_data
            self.campus_network = campus_network
```

```
            self.locations = distribution_data.columns.tolist()
            self.times = distribution_data.index.tolist()

            # 计算各停车点之间的最短距离
            self.distance_matrix = self._calculate_distance_matrix()

            # 计算需求矩阵
            self.demand_matrix = self._calculate_demand_matrix()

        def _calculate_distance_matrix(self):
            """计算各停车点之间的最短距离矩阵"""
            n = len(self.locations)
            distance_matrix = np.zeros((n, n))

            for i in range(n):
                for j in range(n):
                    if i != j:
                        try:
                            # 使用NetworkX计算最短路径距离
                            distance_matrix[i,          j]          =
nx.shortest_path_length(
                                self.campus_network,
                                source=self.locations[i],
                                target=self.locations[j],
                                weight='weight'
                            )
                        except nx.NetworkXNoPath:
                            # 如果不存在路径，设为一个大值
                            distance_matrix[i, j] = 10000
                    else:
                        distance_matrix[i, j] = 0

            return distance_matrix

        def _calculate_demand_matrix(self):
            """计算需求矩阵"""
            # 这里使用单车数量变化来估计需求
            # 如果某时刻单车数量减少，表示有需求（骑出）
            # 如果某时刻单车数量增加，表示有归还（骑入）
            demand_matrix    =    np.zeros((len(self.times)    -    1,
len(self.locations)))

            for t in range(len(self.times) - 1):
                for i, loc in enumerate(self.locations):
```

```
                    # 计算相邻时间点之间的变化
                    change = self.distribution_data.iloc[t + 1][loc]
- self.distribution_data.iloc[t][loc]
                    # 变化为负表示有需求（骑出）
                    if change < 0:
                        demand_matrix[t, i] = abs(change)

            return demand_matrix

        def evaluate_efficiency(self):
            """评估运营效率"""
            # 计算各项指标

            # 1. 可用率：平均每个停车点的单车数量与总数量的比值
            avg_availability = self.distribution_data.mean().mean() /
self.distribution_data.sum().sum() * len(
                self.locations)

            # 2. 调度成本：基于距离矩阵和需求矩阵计算
            dispatch_cost = self._calculate_dispatch_cost()

            # 3. 需求满足率：能满足的需求与总需求的比值
            demand_satisfaction                                    =
self._calculate_demand_satisfaction()

            # 4. 点位分布均衡性：使用变异系数(CV)衡量
            distribution_balance                                   =
self._calculate_distribution_balance()

            # 5. 使用率：不同时间段的平均使用情况
            utilization_rate = self._calculate_utilization_rate()

            # 综合评分：根据各指标权重计算
            weights = {
                'availability': 0.2,
                'dispatch_cost': 0.3,
                'demand_satisfaction': 0.25,
                'distribution_balance': 0.15,
                'utilization_rate': 0.1
            }

            # 对调度成本进行归一化（值越小越好）
            normalized_dispatch_cost = 1 - (dispatch_cost / 1000)  #
假设最大成本为1000
```

```
            score = (weights['availability'] * avg_availability +
                     weights['dispatch_cost']                        *
normalized_dispatch_cost +
                     weights['demand_satisfaction']                  *
demand_satisfaction +
                     weights['distribution_balance']                 *
distribution_balance +
                     weights['utilization_rate'] * utilization_rate)

            return {
                'score': score,
                'availability': avg_availability,
                'dispatch_cost': dispatch_cost,
                'demand_satisfaction': demand_satisfaction,
                'distribution_balance': distribution_balance,
                'utilization_rate': utilization_rate
            }

        def _calculate_dispatch_cost(self):
            """计算调度成本"""
            total_cost = 0

            # 使用贪心算法估算调度成本
            for t in range(len(self.times) - 1):
                # 计算当前时间段各停车点的盈余和短缺
                current_bikes = self.distribution_data.iloc[t].values
                next_bikes = self.distribution_data.iloc[t + 1].values

                # 计算需要调度的数量
                surplus = []  # (位置, 盈余量)
                shortage = []  # (位置, 短缺量)

                for i, (curr, next_val) in enumerate(zip(current_bikes,
next_bikes)):
                    diff = curr - next_val - self.demand_matrix[t, i]
                    if diff > 0:
                        surplus.append((i, diff))
                    elif diff < 0:
                        shortage.append((i, -diff))

                # 按从大到小排序
                surplus.sort(key=lambda x: x[1], reverse=True)
                shortage.sort(key=lambda x: x[1], reverse=True)
```

```
                # 贪心调度：从盈余最大的点向短缺最大的点调度
                for i, s_amount in surplus:
                    remaining = s_amount
                    for j, d_amount in shortage:
                        if d_amount > 0 and remaining > 0:
                            # 计算调度量
                            dispatch_amount = min(remaining, d_amount)
                            # 更新剩余量和短缺量
                            remaining -= dispatch_amount
                            shortage[shortage.index((j, d_amount))] =
(j, d_amount - dispatch_amount)
                            # 累计调度成本
                            total_cost    +=    dispatch_amount    *
self.distance_matrix[i, j]

                            if remaining == 0:
                                break

            return total_cost

        def _calculate_demand_satisfaction(self):
            """计算需求满足率"""
            # 总需求
            total_demand = np.sum(self.demand_matrix)
            if total_demand == 0:
                return 1.0  # 如果没有需求，满足率为100%

            # 实际满足的需求（通过车辆调度）
            satisfied_demand = 0

            for t in range(len(self.times) - 1):
                current_bikes = self.distribution_data.iloc[t].values
                for i, loc in enumerate(self.locations):
                    demand = self.demand_matrix[t, i]
                    if demand <= current_bikes[i]:
                        satisfied_demand += demand
                    else:
                        satisfied_demand += current_bikes[i]

            return satisfied_demand / total_demand

        def _calculate_distribution_balance(self):
            """计算点位分布均衡性"""
```

```
            # 使用变异系数(CV)：标准差/均值
            mean_distribution = np.mean([np.mean(self.distribution_data[loc]) for loc in self.locations])
            std_distribution = np.std([np.mean(self.distribution_data[loc]) for loc in self.locations])

            if mean_distribution == 0:
                return 0  # 避免除以零

            cv = std_distribution / mean_distribution
            # 将CV转换为均衡性指标（1-CV，值越大越均衡）
            balance = max(0, 1 - cv)

            return balance

        def _calculate_utilization_rate(self):
            """计算使用率"""
            # 估算单车的使用情况：需求量/可用量
            utilization = []

            for t in range(len(self.times) - 1):
                total_bikes = self.distribution_data.iloc[t].sum()
                total_demand = np.sum(self.demand_matrix[t])

                if total_bikes > 0:
                    utilization.append(total_demand / total_bikes)
                else:
                    utilization.append(0)

            return np.mean(utilization)

        def optimize_parking_layout(self, num_clusters=None):
            """
            优化停车点布局

            Args:
                num_clusters: 优化后的停车点数量，默认为当前数量

            Returns:
                new_locations: 优化后的停车点位置列表
                new_distribution: 优化后的分布数据
            """
```

```
            if num_clusters is None:
                num_clusters = len(self.locations)

            # 1. 创建停车点的需求热力图
            demand_heatmap = np.zeros(len(self.locations))
            for t in range(len(self.times) - 1):
                demand_heatmap += self.demand_matrix[t]

            # 将需求热力图归一化
            if np.sum(demand_heatmap) > 0:
                demand_heatmap = demand_heatmap / np.sum(demand_heatmap)

            # 2. 使用需求热力图和距离矩阵优化停车点布局
            # 这里简化为基于需求重要性选择停车点

            # 计算每个停车点的重要性分数
            importance_scores = np.zeros(len(self.locations))
            for i in range(len(self.locations)):
                # 需求得分
                demand_score = demand_heatmap[i]

                # 连接性得分（与其他点的平均距离的倒数）
                avg_distance = np.mean([self.distance_matrix[i, j] for j in range(len(self.locations)) if j != i])
                connectivity_score = 1 / avg_distance if avg_distance > 0 else 0

                # 综合得分
                importance_scores[i] = 0.7 * demand_score + 0.3 * connectivity_score

            # 选择重要性最高的 num_clusters 个点
            selected_indices = np.argsort(importance_scores)[-num_clusters:]

            # 3. 生成新的布局和分布数据
            new_locations = [self.locations[i] for i in selected_indices]

            # 重新分配单车
            new_distribution = pd.DataFrame(index=self.times, columns=new_locations)
```

```
            for t in range(len(self.times)):
                total_bikes = self.distribution_data.iloc[t].sum()
                # 按需求比例分配单车
                for i, loc in enumerate(new_locations):
                    loc_index = self.locations.index(loc)
                    new_distribution.loc[self.times[t],     loc]     =
total_bikes * importance_scores[loc_index] / np.sum(
                        [importance_scores[j]      for      j      in
selected_indices])

            # 将结果取整
            new_distribution = new_distribution.round()

            return new_locations, new_distribution

        def compare_layouts(self, new_distribution):
            """
            比较原布局和新布局的运营效率

            Args:
                new_distribution: 新布局的分布数据

            Returns:
                comparison: 比较结果字典
            """
            # 保存原始分布数据
            original_distribution = self.distribution_data

            # 计算原布局的效率
            original_efficiency = self.evaluate_efficiency()

            # 更新分布数据为新布局
            self.distribution_data = new_distribution

            # 计算新布局的效率
            new_efficiency = self.evaluate_efficiency()

            # 恢复原始分布数据
            self.distribution_data = original_distribution

            # 比较结果
            comparison = {
                'original_score': original_efficiency['score'],
                'new_score': new_efficiency['score'],
```

```
                'improvement':        new_efficiency['score']        -
original_efficiency['score'],
                'original_metrics': original_efficiency,
                'new_metrics': new_efficiency
            }

            return comparison

        def visualize_layout(self, distribution=None, title="共享单车
布局"):
            """
            可视化停车点布局（优化样式版本）
            """
            if distribution is None:
                distribution = self.distribution_data

            # 设置全局样式
            sns.set_style("whitegrid")
            plt.rcParams['font.size'] = 12

            # 气泡图优化
            plt.figure(figsize=(14, 10), dpi=100)
            avg_bikes = distribution.mean()

            # 生成更美观的渐变色
            colors = sns.color_palette("YlGnBu", len(avg_bikes))
            sizes = (avg_bikes / avg_bikes.max() * 1500 + 100)  # 动
态调整大小

            # 创建虚构的极坐标布局
            n = len(avg_bikes)
            angles = np.linspace(0, 2 * np.pi, n, endpoint=False)
            radius = np.random.uniform(3, 5, n)  # 添加随机半径增加立
体感

            x = radius * np.cos(angles)
            y = radius * np.sin(angles)

            # 绘制气泡图
            scatter = plt.scatter(x, y,
                                  s=sizes,
                                  c=colors,
                                  edgecolors='w',
                                  linewidths=1.5,
```

```
                                alpha=0.85,
                                cmap="YlGnBu",
                                zorder=3)

            # 添加颜色条
            cbar = plt.colorbar(scatter, shrink=0.8)
            cbar.set_label('停车点需求等级',  rotation=270,
labelpad=20)

            # 添加标签和注释
            for i, loc in enumerate(avg_bikes.index):
                plt.annotate(f"{loc}\n({int(avg_bikes[i])}辆)",
                             (x[i], y[i]),
                             fontsize=9,
                             ha='center',
                             va='center',
                             color='#2c3e50')

            # 装饰元素
            plt.title(f"{title}\n（气泡大小表示平均车辆数）",
                      fontsize=16,
                      pad=20,
                      color='#2c3e50')
            plt.axis('equal')
            plt.axis('off')

            # 添加背景网格圆
            for r in [3, 5, 7]:
                circle = plt.Circle((0, 0), r,
                                    color='#bdc3c7',
                                    fill=False,
                                    linestyle='--',
                                    alpha=0.3)
                plt.gca().add_artist(circle)

            plt.tight_layout()
            plt.savefig(f"{title.replace(' ', '_')}_v2.png", dpi=300,
bbox_inches='tight')
            plt.show()
            plt.close()

            # 热力图优化
            plt.figure(figsize=(16, 10), dpi=100)
```

```
            # 创建自定义颜色映射
            cmap = sns.light_palette("#3498db", as_cmap=True)

            ax = sns.heatmap(distribution.T,
                             cmap=cmap,
                             annot=True,
                             fmt=".0f",
                             linewidths=0.8,
                             linecolor='#ecf0f1',
                             annot_kws={'fontsize':    8,    'color':
'#2c3e50'},
                             cbar_kws={'label': '单车数量'})

            # 优化坐标轴标签
            ax.set_xticklabels(distribution.index,
                               rotation=45,
                               ha='right',
                               fontsize=10,
                               color='#7f8c8d')

            ax.set_yticklabels(distribution.columns,
                               rotation=0,
                               fontsize=10,
                               color='#7f8c8d')

            # 设置标题样式
            plt.title(f"{title} - 时空分布热力图\n",
                      fontsize=14,
                      color='#2c3e50',
                      pad=20)

            # 调整颜色条
            cbar = ax.collections[0].colorbar
            cbar.ax.tick_params(labelsize=10)
            cbar.outline.set_edgecolor('#ecf0f1')

            plt.tight_layout()
            plt.savefig(f"{title.replace(' ', '_')}_heatmap_v2.png",
dpi=300, bbox_inches='tight')
            plt.show()
            plt.close()


    # 3. 主函数 - 解决问题 3
```

```
    def solve_problem3():
        """解决问题 3：建立共享单车运营效率的评价模型"""
        print("问题 3：建立共享单车运营效率的评价模型")

        # 1. 准备数据
        distribution_table = create_sample_data()
        campus_network = create_campus_network()

        # 2. 创建模型实例
        bike_model = BikeOperationModel(distribution_table,
campus_network)

        # 3. 评估当前运营效率
        current_efficiency = bike_model.evaluate_efficiency()
        print("\n 当前共享单车运营效率评估结果：")
        print(f"综合评分: {current_efficiency['score']:.4f}")
        print(f"可用率: {current_efficiency['availability']:.4f}")
        print(f"调度成本: {current_efficiency['dispatch_cost']:.2f}")
        print(f"需求满足率:
{current_efficiency['demand_satisfaction']:.4f}")
        print(f"点位分布均衡性:
{current_efficiency['distribution_balance']:.4f}")
        print(f"使用率: {current_efficiency['utilization_rate']:.4f}")

        # 4. 可视化当前布局
        bike_model.visualize_layout(title="当前共享单车布局")

        # 5. 判断当前布局是否合理
        is_reasonable = current_efficiency['score'] > 0.7  # 设定阈值
为 0.7
        if is_reasonable:
            print("\n 当前停车点布局合理。")
        else:
            print("\n 当前停车点布局不够合理，需要调整。")

            # 6. 优化停车点布局
            print("\n 开始优化停车点布局...")
            new_locations, new_distribution =
bike_model.optimize_parking_layout()

            # 7. 比较优化前后的效率
            comparison = bike_model.compare_layouts(new_distribution)
            print("\n 优化后共享单车运营效率评估结果：")
            print(f"原布局评分: {comparison['original_score']:.4f}")
```

```
            print(f"新布局评分: {comparison['new_score']:.4f}")
            print(f"提升幅度: {comparison['improvement']:.4f}")

            # 8. 可视化优化后的布局
            bike_model.visualize_layout(new_distribution, title="优化后共享单车布局")

            # 9. 详细输出优化结果
            print("\n详细优化结果：")
            print("原布局停车点：", bike_model.locations)
            print("新布局停车点：", new_locations)

            print("\n新布局单车分布：")
            print(new_distribution)

        return bike_model, new_distribution if 'new_distribution' in locals() else None


    if __name__ == "__main__":
        solve_problem3()
```

第四问代码

```
    import numpy as np
    import random
    from itertools import permutations

    # ========== 输入数据 ==========
    # 曼哈顿距离矩阵（单位：百米，示例数据需替换为实际距离矩阵）
    distance_matrix = np.array([
        [0, 20, 5, 6, 11, 16, 7, 9, 11, 9, 8, 8, 15, 17, 3],
        [20, 0, 25, 14, 11, 10, 23, 21, 16, 14, 12, 14, 15, 17, 17],
        [5, 25, 0, 17, 16, 15, 2, 4, 10, 12, 8, 6, 16, 14, 14],
        [6, 14, 17, 0, 3, 8, 13, 11, 9, 7, 5, 7, 13, 15, 9],
        [11, 11, 16, 3, 0, 5, 14, 12, 8, 6, 6, 8, 14, 16, 12],
        [16, 10, 15, 8, 5, 0, 15, 13, 9, 7, 9, 11, 17, 19, 17],
        [7, 23, 2, 13, 14, 15, 0, 2, 8, 10, 6, 4, 14, 12, 12],
        [9, 21, 4, 11, 12, 13, 2, 0, 6, 8, 4, 2, 12, 10, 10],
        [11, 16, 10, 9, 8, 9, 8, 6, 0, 2, 5, 7, 8, 10, 12],
        [9, 14, 12, 7, 6, 7, 10, 8, 2, 0, 3, 5, 10, 12, 10],
        [8, 12, 8, 5, 6, 9, 6, 4, 5, 3, 0, 2, 11, 13, 7],
        [8, 14, 6, 7, 8, 11, 4, 2, 7, 5, 2, 0, 13, 15, 9],
        [15, 15, 16, 13, 14, 17, 14, 12, 8, 10, 11, 13, 0, 2, 14],
        [17, 17, 14, 15, 16, 19, 12, 10, 10, 12, 13, 15, 2, 0, 16],
        [3, 17, 14, 9, 12, 17, 12, 10, 12, 10, 7, 9, 14, 16, 0]
```

```
])

# 停车点名称（按顺序对应矩阵行列）
locations = [
    "东门", "南门", "北门", "一食堂", "二食堂", "三食堂", "梅苑 1 栋", "菊苑 1 栋",
    "教学 2 楼", "教学 4 楼", "计算机学院", "工程中心", "网球场", "体育馆", "校医院"
]

# 调度参数
num_ants = 20          # 蚂蚁数量
max_iter = 100         # 最大迭代次数
alpha = 1              # 信息素重要程度
beta = 2               # 启发式信息重要程度
rho = 0.1              # 信息素挥发率
Q = 100                # 信息素增量常数
capacity = 20          # 调度车容量（辆/趟）
speed = 25 * 1000 / 60 # 调度车速度（米/分钟）

# ========== 初始化信息素 ==========
pheromone = np.ones_like(distance_matrix) / len(distance_matrix)

# ========== 定义目标函数：计算路径总距离 ==========
def calculate_distance(path):
    total_distance = 0
    for i in range(len(path) - 1):
        total_distance += distance_matrix[path[i]][path[i+1]]
    return total_distance

# ========== 局部搜索：2-opt 优化 ==========
def two_opt_swap(path, i, j):
    new_path = path.copy()
    new_path[i:j+1] = path[i:j+1][::-1]
    return new_path

def local_search(path):
    improved = True
    best_path = path.copy()
    best_distance = calculate_distance(path)
    while improved:
        improved = False
        for i in range(1, len(best_path) - 2):
            for j in range(i + 1, len(best_path)):
```

```
                    if j - i == 1: continue
                    new_path = two_opt_swap(best_path, i, j)
                    new_distance = calculate_distance(new_path)
                    if new_distance < best_distance:
                        best_path = new_path
                        best_distance = new_distance
                        improved = True
        return best_path, best_distance

    # ========== 蚁群算法主流程 ==========
    best_global_path = None
    best_global_distance = float('inf')

    for iteration in range(max_iter):
        ants_paths = []
        ants_distances = []

        # 每只蚂蚁构建路径
        for ant in range(num_ants):
            # 随机起点（实际应用中可指定起点）
            current_node = random.randint(0, len(distance_matrix) - 1)
            path = [current_node]
            unvisited = set(range(len(distance_matrix))) -
{current_node}

            # 逐步访问所有节点
            while unvisited:
                # 计算转移概率
                probabilities = []
                for node in unvisited:
                    pheromone_val = pheromone[current_node][node] **
alpha
                    heuristic_val = (1 /
distance_matrix[current_node][node]) ** beta
                    probabilities.append(pheromone_val *
heuristic_val)

                # 轮盘赌选择下一节点
                probabilities = np.array(probabilities) /
np.sum(probabilities)
                next_node =
list(unvisited)[np.random.choice(len(probabilities), p=probabilities)]

                path.append(next_node)
```

```
                unvisited.remove(next_node)
                current_node = next_node

            # 局部搜索优化
            optimized_path, optimized_distance = local_search(path)
            ants_paths.append(optimized_path)
            ants_distances.append(optimized_distance)

        # 更新信息素
        pheromone *= (1 - rho)  # 挥发

        # 精英蚂蚁策略：仅最优蚂蚁释放信息素
        best_ant_idx = np.argmin(ants_distances)
        for i in range(len(ants_paths[best_ant_idx]) - 1):
            u = ants_paths[best_ant_idx][i]
            v = ants_paths[best_ant_idx][i+1]
            pheromone[u][v] += Q / ants_distances[best_ant_idx]

        # 更新全局最优
        current_best_distance = min(ants_distances)
        if current_best_distance < best_global_distance:
            best_global_distance = current_best_distance
            best_global_path = ants_paths[np.argmin(ants_distances)]

        print(f"Iteration {iteration + 1}: Best Distance = {best_global_distance}")

    # ========== 输出结果 ==========
    print("\n最优路径：")
    for idx in best_global_path:
        print(locations[idx], end=" -> ")
    print("\n总距离：", best_global_distance * 100, "米")
```