所属类别	**2024 年“华数杯”全国大学生数学建模竞赛**	参赛编号
本科组		CM2401458

城市旅游吸引力评估与路线规划研究

摘要

本文针对外国友人至中国旅游路线的制定问题，通过附件数据和收集数据，针对各城市建立反映了其旅游吸引力的多指标评价模型，对模型筛选得出的高吸引力城市，建立旅行路线的多目标规划模型，分别尝试以游览城市数目、总费用为优化目标，通过分支定界法和遗传算法对模型进行求解，得到基于两种优化目标最佳的旅行路线方案。

针对问题一，首先对附件数据进行合并便于后续处理，筛选出对应城市与评分数据，引入偏度系数、峰度系数、变异系数对数据进行描述性统计，通过数据：**左偏且具有尖峰分布、不符合正态的分布特性**，采用中位数插补和 *IQR* 法对数据进行缺失值和异常值处理，接着对数据进行重复值剔除，通过 *python* 函数求解出**最高评分为 5 分，共有 2350 个景点得到最高分，获评最高评分景点最多的城市有玉溪(21)益阳(20)廊坊(18)大兴安岭(18)烟台(18)周口(16)邢台(16)自贡(16)淮安(14)保定(14)。**

针对问题二，首先收集了题目所述的城市规模、环境保护水平、人文底蕴、交通便利性、气候条件以及美食等多个因素的数据，接着对数据进行预处理并归类，建立了**旅游接待次数、平均接待游客量、单次最大接待量、旅游体验一致性、旅游景点的分布合理性、合理接待比例**共六个能有效反映城市旅游吸引力的评价指标，经**熵权法**确定权重分别为 0.11,0.12,0.17,0.23,0.16,0.19，再通过 ***TOPSIS* 法**，结合 6 个指标及其权重，建立多指标评价模型，对每个城市的旅游吸引力进行评价。基于模型的评价结果，本文筛选出了 50 个吸引力最高的城市，其中得分最高的三个城市分别为凉山,拉萨,成都,这三个城市的得分分别为 0.68,0.45 和 0.45。

针对问题三，首先从问题一处理后数据中筛选出问题二的 *TOP*50 城市数据并再次进行预处理，接着构建了以最大化访问的城市数量为目标函数，路径存在性变量、城市访问变量、每日出发城市变量共三个二元变量为决策变量，唯一访问约束、每日出发约束、路径连接约束、每日时间限制、往返广州约束、路径连续性约束为约束条件的**混合整数规划模型**，采用**分支定界法**得出了求解结果。

针对问题四，问题四在问题三的基础上增加了最小化费用的目标，采用**加权法**将多目标优化转变为单目标优化以便于求解，采用问题三定义的决策变量与约束条件，仍使用分支定界法来进行了结果求解。

针对问题五，首先从问题一处理后数据筛选出与山有关的景点的数据并进行预处理，接着采用 ***ε*-约束法**将多目标转化为单目标求解也即：**费用最小化作为硬约束，将访问尽可能多的山景作为优化目标**，结合问题四的优化模型，增加入境城市往返约束以及费用上限约束，通过**遗传算法**进行求解并对结果进行了合理性分析。

最后本文对所建立的模型进行了讨论和分析，综合评价了模型的优缺点。

关键词：多目标优化;混合整数规划;*TOPSIS*;路线规划;

一、 问题重述

1.1 问题背景

最近，“city 不 city”这一网络流行语在外国网红的推动下备受关注。随着我国过境免签政策的落实，越来越多外国游客来到中国，通过网络平台展示他们在华旅行的见闻，这不仅推动了中国旅游业的发展，更是在国际舞台上展现了一个真实而生动的中国，一举多得。 假设外国游客入境后能在中国境内逗留 144 小时，且能从任一城市附近的机场出境。由于每个城市景点较多，为了便于外国游客能够游览到更多的城市，现假定“每个城市只选择一个评分最高的景点游玩”，称之为“城市最佳景点游览原则”。现有一个包含中国（不含港澳台）352 个城市的旅游景点的数据集，每个城市的 csv 文件中有 100 个景点，每个景点的信息包含有景点名称、网址、地址景点介绍、开放时间、图片网址、景点评分、建议游玩时长、建议游玩季节、门票信息、小贴士等

1.2 问题要求

题目要求尝试建立数学模型讨论下列问题：

问题一: 352 个城市中所有 35200 个景点评分的最高分（Best Score，简称 BS 是多少，全国有多少个景点获评了这个最高评分（BS），获评了这个最高评分（BS 景点最多的城市有哪些，依据拥有最高评分（BS）景点数量的多少排序，列出前 10 个城市。

问题二: 假如外国游客遵循“城市最佳景点游览原则”，结合城市规模、环境环保、人文底蕴、交通便利，以及气候、美食等因素，请你对 352 个城市进行综合评价，选出“最令外国游客向往的 50 个城市”。

问题三: 现有一名外国游客从广州入境，他想在 144 小时以内游玩尽可能多的城市，同时要求综合游玩体验最好，请你规划他的游玩路线。需要结合游客的要求给出具体的游玩路线，包括总花费时间，门票和交通的总费用以及可以游玩 的景点数量。他的要求有：①遵循城市最佳景点游览原则；②城市之间的交通方式只选择高铁；③只在“最令外国游客向往的 50 个城市”中选择要游玩的城市。

问题四: 如果将问题 3 的游览目标改为：既要尽可能的游览更多的城市，又需要使门票和交通的总费用尽可能的少。请重新规划游玩路线，并给出门票和交通的总费用，总花费时间以及可以游玩的城市数量。

问题五: 现有一名外国游客只想游览中国的山景，他乘飞机入境中国的城市不限。请你为他选择入境的机场和城市，并个性化定制他的 144 小时旅游路线，既要尽可能的游览更多的山，又需要使门票和交通的总费用尽可能的少。需要结合游客的要求给出具体的游玩路线，包括总花费时间，门票和交通的总费用以及可以游玩的景点数量。他的要求有：①每个城市只游玩一座评分最高的山；②城市之间的交通方式只选择高铁；③旅游城市不局限于“最令外国游客向往的 50 个城市”，游览范围拓展到 352 个城市。

二、 问题分析

2.1 问题一分析

该问主要要求对原始数据的处理和整合，我们首先要进行数据预处理，将原始数据文件夹整合为一个数据集，然后提取出城市与评分的数据，通过 *python* 函数来对第一问进行求解。

2.2 问题二分析

第二问要求结合多种因素对 352 个城市进行综合评价并排名，所以我们首先要收集多个维度的数据，然后进行数据预处理，接着通过直接或间接计算的方式将原始数据降维为评价指标，通过基于熵权法的 *TOPSIS* 评价模型来对城市进行综合评分，最后进行排名。

2.3 问题三、四、五分析

问题三、四、五的路径规划，我们可以构建混合整数规划模型来进行求解：

针对第三问，我们首先要筛选出第二问中所评价的 *TOP*50 吸引力城市作为原始数据，接着我们需要收集城市的经纬度信息来计算距离以及交通成本，将最大化旅游城市数量作为目标函数，结合多种约束条件构建优化模型，采用较为精确的分支定界法来求解出最终路径。

针对第四问，我们需要在问题三的基础上增加总成本的考虑条件，单目标优化模型变为多目标优化模型，我们可以采用加权法来将多目标转化为单目标，再次采用分支定界法进行求解。

针对第五问，我们需要筛选出原始数据中山景点作为该问的输入，对于该问的多目标优化问题，我们使用 ε-约束法（*ε-constraint method*）来处理，同时由于本问放开了城市限制导致计算量增大，我们可以采用启发式算法来加快求解速度。

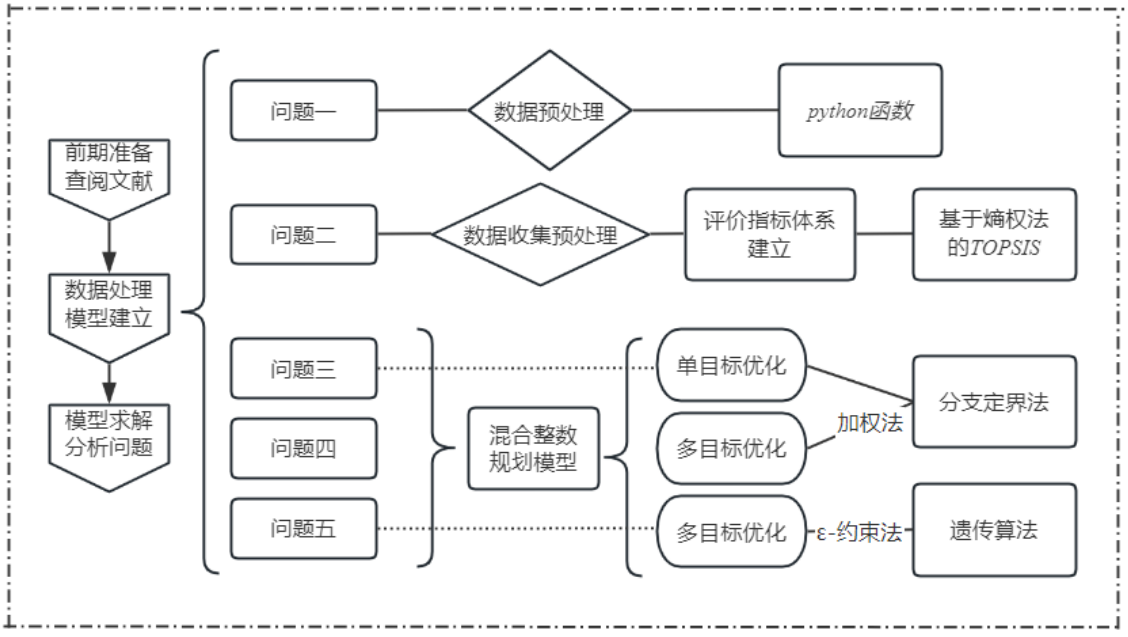


图 1 总体思路图

三、 模型假设与符号说明

1.假设可以通过 *haversine* 公式计算结果来代替两个城市间的交通距离。

2.数据完整性和准确性: 所有的城市和景点数据被认为是完整的且准确无误的，包括但不限于城市地理位置信息、景点的营业时间、评分以及门票价格等细节。

3.静态环境假设: 在整个旅行过程中，交通状况、景点的开放状态以及天气等外部条件被视为恒定不变，不考虑任何实时的变化因素。

符号	意义
S_k	偏度系数
K_u	峰度系数
CV	变异系数
$coord_i$	经纬度坐标
x_{Scale}	数据标准化
$average\_speed$	平均高速列车速度
$travel\_time_i$	旅行时间
$play\_time_i$	游玩时间
N	访问城市数量
C	总成本
α	权重系数
D	总旅行天数
S	入境城市
p_i	城市 i 中门票景点价格
C_{max}	费用上限约束

四、数据预处理

4.1 数据集整理

首先我们对原始的附件数据集进行了读取，并把所有景点的数据整合起来，以便后续处理，部分结果如下：

表 1 部分数据合并结果

名字	链接	地址	介绍	开放时间	图片链接	评分	建议游玩时间	建议季节	门票	小贴士	*Page*
石龙山	*shilongshan*	铁山乡境内	国家*AA*级旅游	8:00--17:00			3.7		15元		1
西大圈	*xidaquan*								现场公示		1
挠力河	*naolihe*								现场公示		1

从部分结果我们可以看出，数据中存在一些缺失值需要我们后续进行处理。

五、 问题一模型的建立与求解

5.1 数据选取

我们选取出合并数据中的地名以及评分来进行问题一的求解：

表 2 问题一部分数据

名字	评分
万宝湖儿童公园 *Wanbaohu Children's Park*	4.6
桃山公园 *Taoshan Park*	4.6
红星森林公园 *Hongxing Forest Park*	4.5
石龙山森林公园 *Shilongshan National Forest Park*	4.4
西大圈森林公园 *Xidaquan National Forest Park*	4.7
石龙山	3.7

5.2 描述性统计与分析

接着我们对评分进行描述性统计，本文引入均值、最大值、最小值、中位数、标准差、偏度系数、峰度系数，变异系数来描述统计数据：

(1)偏度系数

偏度系数(*Skewness*)用于衡量数据分布的偏斜程度。$S_k = 0$表示数据近似对称分布，$S_k > 0$表示数据呈右偏分布，$S_k < 0$表示数据呈左偏分布。偏度系数的绝对值越大，数据分布的偏斜程度越明显。其计算公式如下：

$$S_k = E\left[\left(\frac{X-\mu}{\sigma}\right)^3\right] = \frac{\mu^3}{\sigma^3}$$

其中S_k为偏度系数，*E(X)*为均值，μ^3为三阶中心距

(2)峰度系数

峰度系数(*Kurtosis*)用于衡量数据分布的峰度程度。K_u=0 表示数据分布为正态分布，K_u< 0 表示数据分布的峰度较小，数据更分散，K_u> 0 表示数据分布的峰度较大，数据更集中。峰度系数的绝对值越大，数据分布的峰度程度越明显。其计算公式如下：

$$K_u = E\left[\left(\frac{X-\mu}{\sigma}\right)^4\right] = \frac{\mu^4}{\sigma^4}$$

其中K_u为峰度系数，*E(X)*为均值，μ^4为三阶中心距

(3)变异系数

变异系数是一个统计学中的度量指标，用来衡量一组数据的离散程度与平均值的比例关系。它通过将标准差与均值进行比较，提供了一个无单位的度量方式，因此特别适用于比较不同单位或不同尺度数据的离散程度。其计算公式如下：

$$CV = \frac{\sigma}{\mu} \times 100\%$$

其中σ是数据的标准差，μ是数据的均值。

统计结果如下：

表 3 问题一部分数据

变量名	最大值	最小值	平均值	标准差	中位数	方差	峰度	偏度	变异系数（*CV*）
评分	5	0	3.992	1.407	4.5	1.98	3.933	-2.262	0.352

结果分析：

• 评分整体上偏高，平均值接近于 4，并且中位数更高，说明评分主要集中在较高的区间内。

• 峰度和偏度的值表明评分分布呈现左偏尖峰分布，即大部分评分聚集在较高的一端，同时存在少量极低评分。

• 较小的变异系数表明评分的稳定性较好，大多数评分都集中在平均值附近。

• 标准差和方差虽然不是非常大，但结合平均值来看，说明评分仍然有一定的变化范围，不是高度一致的。

综上所述，这个结果表明用户对于所评价的对象普遍持正面态度，但同时也存在一

些显著的负面评价，这些负面评价可能是由一些特定因素引起的。

同时我们也可视化了评分的分布图像：

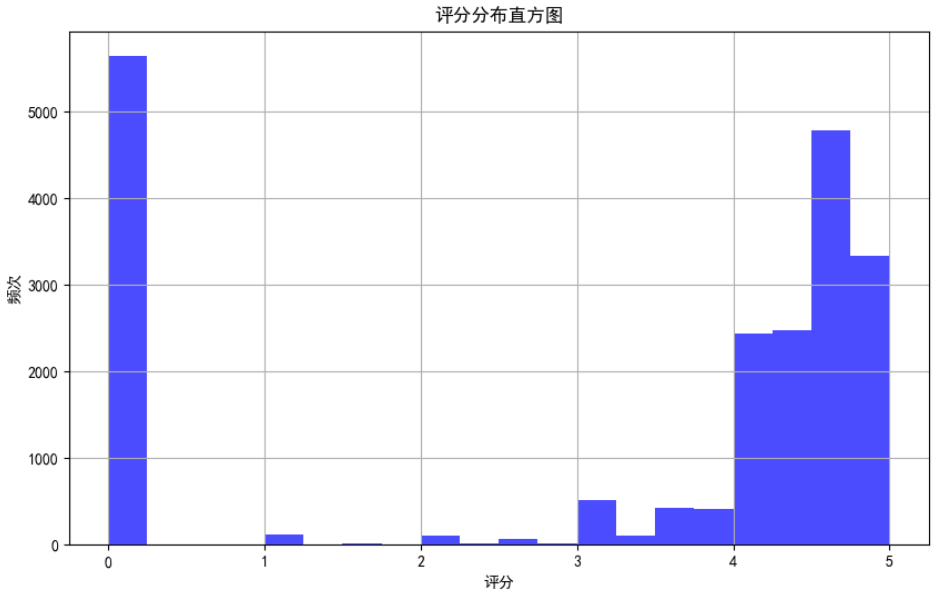


图 2 评分分布图

通过统计结果以及可视化图像，对于单变量的情况，我们使用中位数插补。因为数据左偏且具有尖峰分布，中位数插补能够更好地反映数据的中心趋势。

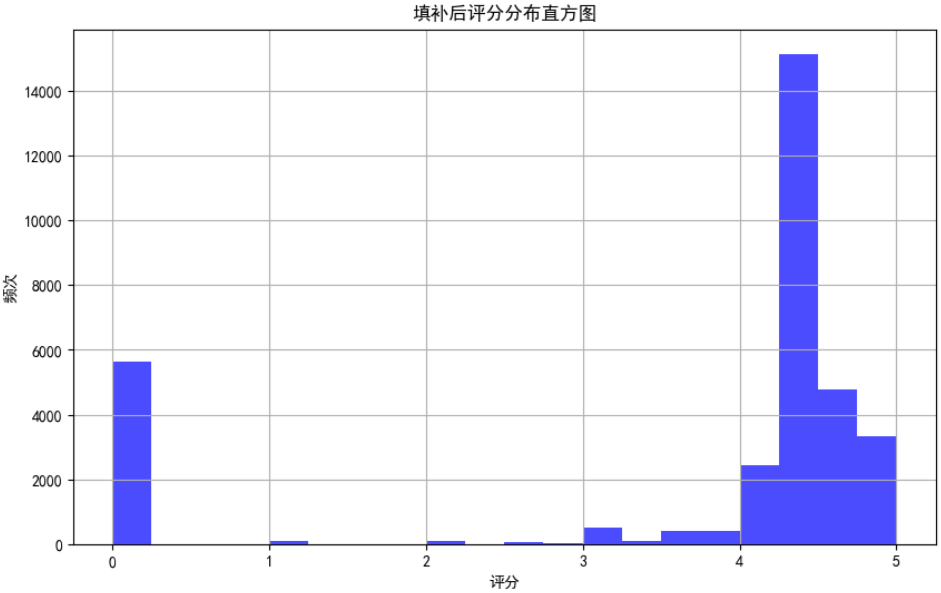


图 3 填补后评分分布图

填补后我们可以观察出，较多景点的评分都处在 4 分到 5 分之间，但还是有少数景点的评分很低，存在负面评价。

5.3 异常值剔除

通过填补后的评分分布，我们可以看出评分不符合正态分布，所以我们采用四分位数间距法来进行异常值剔除：

***IQR*(四分位数间距)法：**

IQR 法是基于数据的四分位数来识别异常值。这种方法首先计算第一四分位数(*Q1*)和第三四分位数(*Q3*),然后计算 *IQR*(四分位数间距),即

$$IQR = Q3 - Q1$$

然后通过使用 1.5 倍的 IQR 作为判断标准来识别异常值。

第一四分位数(*Q1*):数据集中位于 25%位置的值。

第三四分位数(*Q3*):数据集中位于 75%位置的值。

四分位数间距(*IQR*):

$$IQR = Q3 - Q1$$

异常值的判断标准：如果$x_i < Q1 - 1.5 \times IQR$或$x_i > Q3 + 1.5 \times IQR$,则$x_i$可能是异常值。

部分异常值结果如下：

表 4 异常值结果部分数据

名字	评分	建议游玩时间	门票	*outlier*
勃利密塞 *Bolimi Fortress*	0	1 小时 - 2 小时	现场公示	1
七台河市群众艺术馆	0		现场公示	1
东南亚风情村	3	2 小时 - 3 小时	免费	1
桥海牧场	0			1
蓝色海岸冲浪俱乐部	0		现场公示	1
鼎睿蔚蓝潜水俱乐部	0		现场公示	1

5.4 重复值剔除

我们使用 *df.drop_duplicates*(*subset*='名字', *inplace*=*True*)来剔除重复出现的景点，以保证结果的准确性。

5.5 问题一的解决

5.5.1 第一小问

基于上面处理后的数据，我们使用 *python* 的 *max* 函数来求解最大值结果如下：

最高评分（BS）是: 5.0

5.5.2 第二小问

我们通过将每个景点的分数与最高评分进行比对，得出共有 2350 个景点得到了最高评分。

5.5.3 第三小问

我们首先通过第二小问得出了 2350 个景点的信息：

表 5 部分景点信息

名字	评分	建议游玩时间	门票
勃利森林公园	5	12 小时 - 3 天	现场公示
亿达广场 *Yida Square*	5		现场公示
龙泉寺 *Longquan Temple*	5		现场公示
南山公园	5	2 小时 - 3 小时	现场公示
神州半岛灯塔	5	0.5 小时 - 1 小时	现场公示
青云塔	5		5 元

接着我们统计出了每个景点的所在城市，得出了结果：

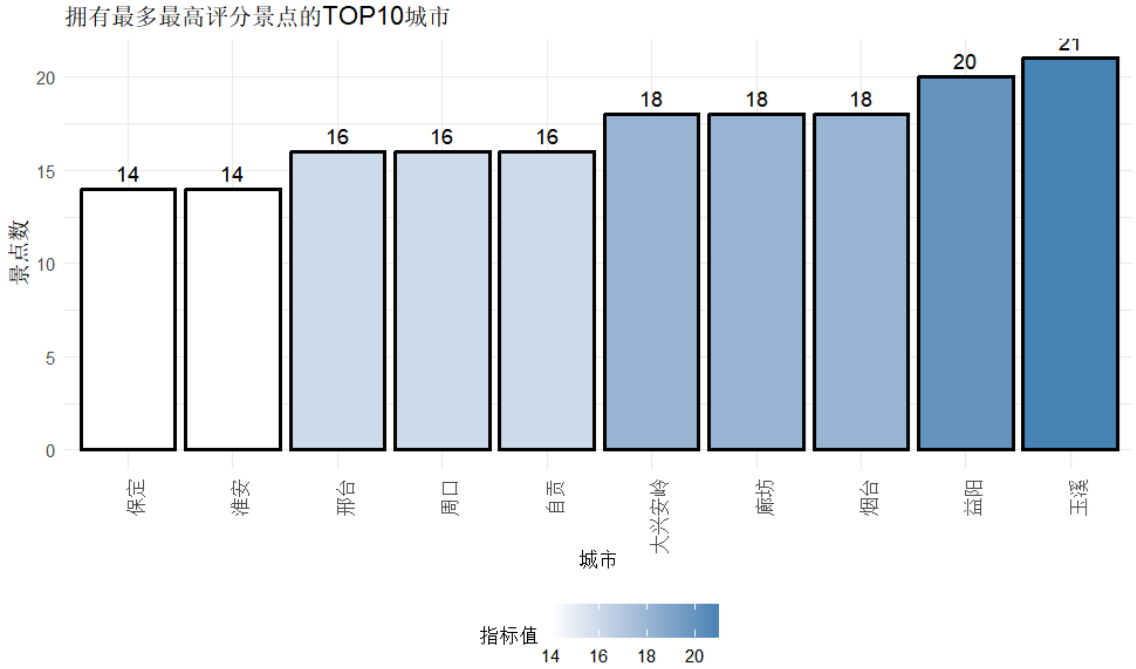


图 4 拥有最多最高评分景点的 TOP10 城市

六、 问题二模型的建立与求解

6.1 数据收集与预处理

6.1.1 数据收集

本问题要求基于 352 个城市的旅游景点数据集，对各个城市的旅游吸引力进行量化

分析，建立模型以反映城市吸引外国游客的能力，并在此基础上给出 50 个最吸引外国游客的城市。在选择这些城市时，需要综合考虑城市规模、环境保护水平、人文底蕴、交通便利性、气候条件以及美食等多个因素。所以我们首先收集了这些城市的相关数据，以建立一个全面严谨的评价体系，下面是部分指标的阐述解释：

- 空气质量指数(*AQI*)：衡量空气污染程度的指标，数值越高表示空气质量越差。
- 绿化率：城市中绿化覆盖的面积占总面积的百分比，反映城市的生态环境。
- 废水处理效率：城市废水经过处理的比例，反映城市污水处理能力。
- 废气处理效率：城市工业废气经过处理的比例，反映城市对大气污染的控制能力。
- 垃圾回收率：城市生活垃圾中被回收利用的比例，反映城市垃圾处理和资源回收的能力。
- 文化遗迹数量：城市中历史遗迹的数量，反映城市的历史文化深度。
- 博物馆数量：城市中博物馆的总数，反映城市文化设施的丰富程度。

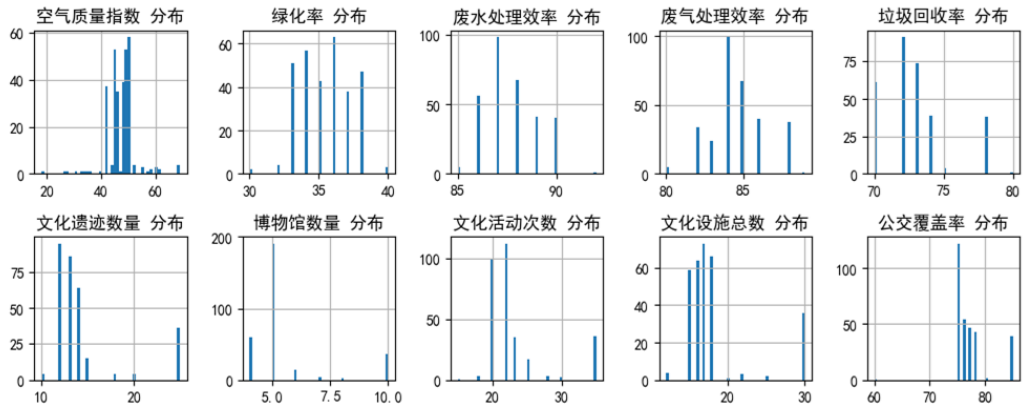
然后我们将所有指标进行归类：

表 6 指标归类汇总

类别	指标
城市规模	城市规模
环境保护水平	空气质量指数(*AQI*) 绿化率 废水处理效率 废气处理效率 垃圾回收率
人文底蕴	文化遗迹数量 博物馆数量 文化活动次数 文化设施总数
交通便利性	公交覆盖率 交通线路密度 高速里程 机场航班频次
气候条件	平均气温 年降水量 旅游适宜天数 空气湿度百分比
美食	餐饮设施数量 特色美食种类数 美食活动次数

6.1.2 数据处理

接下来我们可视化了原始数据分布，以及采用箱型图来剔除异常值：



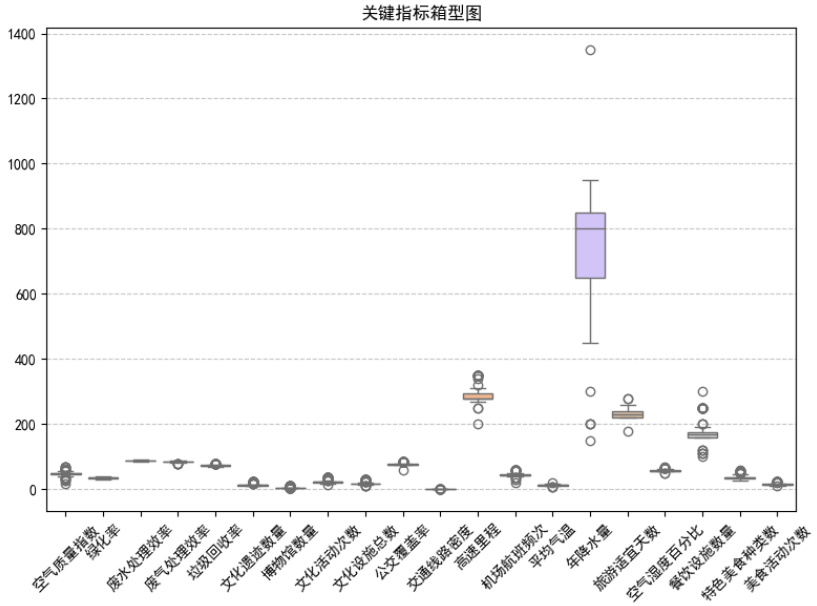


图 5 原始数据分布及箱型图

结合图表，我们对原始数据进行了缺失值和异常值的处理，便于后续评价模型的建立。

6.2 评价指标选取与计算

6.2.1 指标选取

根据相关研究，评价指标应当准确反映被评价对象的特点或属性。每一项所选取的评价指标均是从不同方面衡量评价对象所具备的特定特征的一个度量。当多个评价指标构成一个指标体系时，构建该指标体系的原则在于系统性、科学性、可比性、可测量性和独立性。依据上述原则，我们分析了每个城市的旅游景点数据，构建能够反映城市旅游吸引力的指标。本文从城市旅游服务能力与旅游体验稳定性两个角度出发，构建了以下几个指标：

旅游服务能力：

- 旅游接待次数（反映城市接待游客的能力）
- 平均接待游客量（反映城市平均每次接待游客的数量）

• 单次最大接待量（反映城市单次接待游客的最大容量）

旅游体验稳定性：

• 旅游体验一致性（反映城市旅游体验的稳定性）

• 旅游景点的分布合理性（反映城市景点分布是否均衡）

• 合理接待比例（反映城市接待游客量与城市规模的匹配程度）

以这六个指标为基础，我们建立了多指标评价模型，模型的输出结果作为城市旅游吸引力的重要程度。

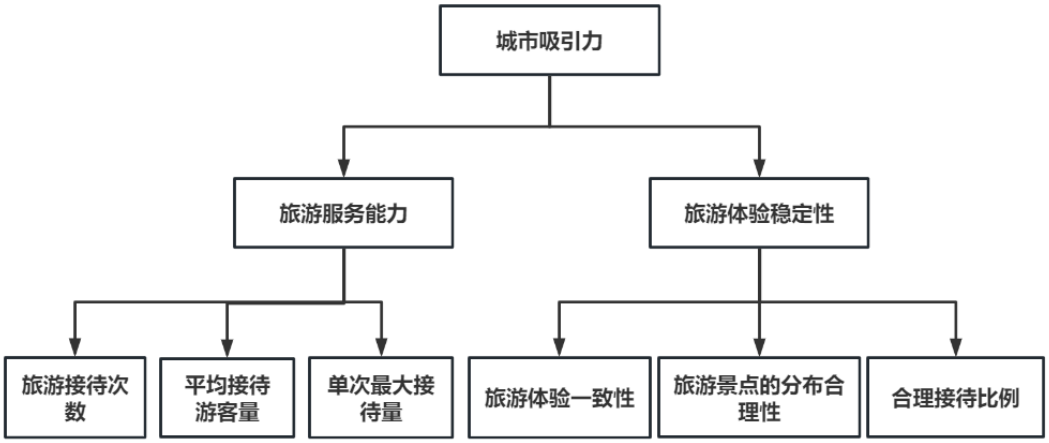


图 6 城市旅游吸引力评价模型

a. 旅游接待次数

该指标反映了一个城市在一定时间内接待游客的总次数。它是一个量化指标，可以显示城市旅游市场的活跃程度和旅游吸引力。

b. 平均接待游客量

该指标指的是在一定时间内，城市每次接待游客的平均数量。它可以帮助我们了解城市每次旅游活动的规模，以及城市在特定时间内能够接待的游客数量。

c. 单次最大接待量

该指标显示了城市在单次旅游活动中能够接待的最大游客数量。这通常与城市的旅游基础设施和资源承载能力有关。

d. 旅游体验一致性

该指标衡量的是游客在不同时间访问同一城市时所获得的旅游体验的一致性。高一致性意味着无论何时访问，游客都能获得相似的旅游体验，这对于建立游客的信任和满意度非常重要。

e. 旅游景点的分布合理性

该指标评估的是城市内旅游景点的地理分布是否均匀，是否方便游客访问。合理的分布可以避免游客过度集中于某些区域，从而提高游客的整体满意度和旅游体验。

f. 合理接待比例

该指标衡量的是城市接待游客的数量与城市规模（包括人口、面积、旅游基础设施等）之间的匹配程度。一个合理的比例意味着城市的旅游接待能力与其资源和规模相适应，既不会导致资源浪费，也不会因资源不足而影响游客体验。

6.2.2 指标计算

a. 旅游接待次数

旅游接待次数是指城市在一定时间内接待游客的总次数：

$$\text{旅游接待次数} = \alpha_1 \cdot \text{餐馆数量} + \alpha_2 \cdot \text{特色美食数量}$$

通过餐馆数量和特色美食数量来间接估计旅游接待次数，因为这两者通常与游客流量正相关。

b. 平均接待游客量

平均接待游客量是指城市在一定时间内每次接待游客的平均人数：

$$\text{平均接待游客量} = \beta_1 \cdot \frac{\text{适宜旅游天数}}{|\text{年平均气温 (℃)}|}$$

通过适宜旅游天数与年平均气温的关系来间接计算平均接待游客量，适宜的气候条件有利于吸引更多游客，从而提高平均接待游客量。

c. 单次最大接待量

单次最大接待量是指城市在一次接待活动中能够接待的最大游客数量：

$$\text{单次最大接待量} = \gamma_1 \cdot \text{公共交通覆盖率} \cdot \gamma_2 \cdot \text{线路密度}$$

通过公共交通覆盖率和线路密度来间接计算单次最大接待量，这两个指标反映了城市的交通便捷程度和服务能力。

d. 旅游体验一致性

旅游体验一致性是指游客在不同时间段内访问同一城市时，体验到的旅游服务质量的稳定性：

$$\text{旅游体验一致性} = \delta_1 \cdot \text{历史遗迹数量} + \delta_2 \cdot \text{博物馆数量} + \delta_3 \cdot \text{文化活动频次}$$

通 过历史遗迹数量、博物馆数量和文化活动频次来评估旅游体验一致性，这些因素共同构成了城市的旅游文化底蕴和活动丰富程度。

e. 旅游景点的分布合理性

旅游景点的分布合理性是指城市内旅游景点分布的均匀程度。

旅游景点的分布合理性：

$$\text{旅游景点的分布合理性} = \epsilon_1 \cdot \frac{\text{绿化覆盖率}}{\text{文化设施数量}}$$

通过绿化覆盖率与文化设施数量的比值来评估旅游景点分布的合理性，绿化覆盖率较高的地区通常环境更宜人，而丰富的文化设施意味着游客可以在城市的不同区域找到有趣的景点。

f. 合理接待比例

合理接待比例是指城市接待游客量与城市规模之间的匹配程度：

$$\text{合理接待比例} = \zeta_1 \cdot \frac{\text{公共交通覆盖率}}{\text{高速公路里程}}$$

通过公共交通覆盖率与高速公路里程的比值来评估合理接待比例，公共交通覆盖率反映了城市内部交通的便利性，而高速公路里程则反映了城市对外的可达性。

成本型指标是指那些数值越小越好的指标，而效应型指标则是数值越大越好的指标。在选定的六个指标中，没有明确的成本型指标。成本型指标通常涉及到一些负面因素，例如成本、污染程度等。

表 7 部分指标计算结果

城市来源	旅游接待次数	平均接待游客量	单次最大接待量	旅游体验一致性	旅游景点的分布合理性	合理接待比例
阿坝	80.64	5.076923077	48.75	13.1	2.25	0.267857143
阿克苏	79.4	5.5	45	12.6	2.266666667	0.277777778
阿拉尔	86.3	6.272727273	53.2	13.3	1.833333333	0.271428571
阿勒泰	80.64	5.076923077	48.75	13.1	2.25	0.267857143
阿里	86.3	4.5	54.6	12.9	2.176470588	0.268965517

效应型指标：

- 旅游接待次数：数值越大表示城市接待游客的能力越强。
- 平均接待游客量：数值越大表示城市平均每次接待游客的数量越多。
- 单次最大接待量：数值越大表示城市单次接待游客的最大容量越大。
- 旅游体验一致性：数值越大表示城市旅游体验的稳定性越好。
- 旅游景点的分布合理性：数值越大表示城市景点分布的均衡程度越高。
- 合理接待比例：数值越大表示城市接待游客量与城市规模的匹配程度越好。

效益型指标归一化：

$$x_{Scale} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

结合针对归一化处理后的指标，旅游城市吸引力评价模型建立如下：

$$\Delta_i = w_1 r_{i,1} + w_2 r_{i,2} + w_3 r_{i,3}$$

其中w_j即为第j个指标的权重，r_i, f即为城市i第j个指标的值。通过熵权法求解权重，代入上式，即可得城市i旅游吸引力Δ_i。

6.2.3 模型汇总

确定了能有效描述城市旅游吸引力的指标后，构建城市旅游吸引力评价模型，通过熵权法方法求解出各个指标的权重，建立基于 *TOPSIS* 的多指标评价模型，该得分即反映出旅游城市吸引力，值越大，则吸引力越高。

- 熵权法(*Entropy Weight Method,EWM*)的基本思路是根据指标变异性的大小 来确定客观权重。一般来说，若某个指标的信息熵越小，表明指标值得变异程度越大，提供的信息量越多，在综合评价中所能起到的作用也越大，其权重也就越大。相反，某个指标的信息熵越大，表明指标值得变异程度越小，提供的 信息量也越少，在综合评价中所起到的作用也越小，其权重也就越小

- *TOPSIS*(*Technique for Order Preference by Similarity to Ideal Solution*)法是一 种常用的组内综合评价方法，能充分利用原始数据的信息，其结果能精确地反映各评价方案之间的差距。基本过程为基于归一化后的原始数据矩阵，采用余弦法找出有限方案中的最优方案和最劣方案，然后分别计算各评价对象与最优方案和最劣方案间的距离，获得各评价对象与最优方案的相对接近程度，以此作为评价优劣的依据。

6.3 模型基于 *TOPSIS* 的多指标评价模型求解

6.3.1 算法求解

(1)熵权法求解流程如下：

熵权法求解流程

***Step* 1** 将数据导入 *Python*,对指标进行标准化处理
***Step* 2** 计算标准化后指标的熵值
***Step* 3** 计算各指标的差异系数
***Step* 4** 计算各指标的权重
***Step* 5** 分别用权重乘以归一化后的数据

(2)*TOPSIS* 法求解流程如下：

***TOPSIS* 算法求解流程**

***Step* 1** 将数据导入 *Python*,根据熵权法得到六个指标的权重
***Step* 2** 对六个指标数据进行指标属性同向化处理并构造归一化初始矩阵
***Step* 3** 确定最优方案和最劣方案
***Step* 4** 计算各评价对象与最优方案、最劣方案的接近程度
***Step* 5** 计算各评价对象与最优方案的贴近程度
***Step* 6** 根据贴近程度大小进行排序，给出评价结果
***Output*:**各城市 *TOPSIS* 评价结果

6.3.2 求解结果展示

表 8 指标权重结果

指标	权重
旅游接待次数	0.1142
平均接待游客量	0.1223
单次最大接待量	0.1741
旅游体验一致性	0.2316
旅游景点的分布合理性	0.1666
合理接待比例	0.1909

从这些权重可以看出，旅游体验的一致性和景区的合理规划对于城市旅游吸引力具有非常重要的影响。此外，景区的最大接待能力和合理的接待比例也是不可忽视的因素。

6.4 最令外国游客向往的五十个城市结果展示

表 9 评价结果

城市来源	综合得分	城市来源	综合得分	城市来源	综合得分
凉山	0.68	大理	0.45	贵阳	0.45
拉萨	0.68	琼海	0.45	通化	0.45
成都	0.45	嘉峪关	0.45	攀枝花	0.45
朔州	0.45	济源	0.45	白山	0.45
丽江	0.45	邵阳	0.45	福州	0.42
定西	0.45	儋州	0.45	包头	0.42
汕尾	0.45	恩施	0.45	阜新	0.38
雄安新区	0.45	惠州	0.45	博尔塔拉	0.38
常德	0.45	台州	0.45	巴彦淖尔	0.35
中山	0.45	临高	0.45	宝鸡	0.33
可克达拉	0.45	三亚	0.45	白银	0.32
呼和浩特	0.45	咸宁	0.45	甘孜	0.32
杭州	0.45	安庆	0.45	永州	0.31
北京	0.45	南充	0.45	漳州	0.31
潜江	0.45	泸州	0.45		
晋城	0.45	潍坊	0.45		
楚雄州	0.45	贺州	0.45		
扬州	0.45	五家渠	0.45		

七、 问题三模型的建立与求解

7.1 原始数据处理

7.1.1 数据筛选

首先我们根据问题的要求，从原始数据中筛选出了景点名称，评分，游玩时间，门票，所属城市这些数据以供后续优化模型的求解：

表 10 部分筛选结果

名字	开放时间	评分	建议游玩时间	门票	来源城市
加井岛	8:00-18:30	4.5	建议游览时间：3 小时 - 5 小时	{'石梅湾加井岛单浮潜体验+摩托艇出海体验大小同价\n\n': ['¥330 起'],等 '..	万宁
石梅湾	全天开放	4.6	建议游览时间：2 小时	{'石梅湾加井岛单浮潜体验+摩托艇出海体验大小同价\n\n': ['¥330 起']等	万宁

7.1.2 数据处理

接着我们对筛选结果做如下处理：

- 对于未说明的信息进行剔除
- 提取出正确的时间段信息
- 筛选出问题二中的五十大城市信息
- 游玩时间提取
- 遵循城市最佳景点游览原则；
- 城市之间的交通方式只选择高铁；
- 只在“最令外国游客向往的 50 个城市”中选择要游玩的城市；

我们筛选出了遵循城市最佳景点游览原则的数据并收集了各城市的经纬度位置：

表 11 问题三最终结果

名字	开放时间	评分	建议游玩时间	门票	来源城市	纬度	经度
五指山	7:30-16:00	5	2.5	0	三亚	18.2528	109.5119
孙中山故居纪念馆	09:00-17:30	4.6	2.5	0.5	中山	22.5176	113.3926
南明山景区	07:00-18:30	4.6	12	7	丽水	28.46763	119.922796
文峰寺	08:30-18:00	4.8	3.5	5	丽江	26.8575	100.2278
八达岭长城	07:30-16:00	4.7	3.5	35	北京	39.9042	116.4074

7.2 优化模型的建立

由于问题三的时间约束，所以我们首先要计算出景点间的旅行时间：
计算两个城市之间的直线距离：

$$distance_{ij} = haversine(coord_i, coord_j)$$

其中：$coord_i$和$coord_j$分别是城市i和j的经纬度坐标。

估计两城市间的旅行时间：

$$travel\_time_{ij} = \frac{distance_{ij}}{average\_speed}$$

其中$average_speed$是平均高速列车速度。

交通成本：

$$c_{ij} = distance_{ij} \cdot traffic\_cost\_per\_km$$

其中：$distance_{ij}$是城市i到城市j的距离；*traffic_cost_per_km* 是每公里的交通费用。

总成本计算：

$$C = \sum_{i \in cities} \sum_{j \in cities, i \neq j} r_{ij} \cdot x_{ij} + \sum_{i \in cities, i \neq 广州} p_i \cdot y_i$$

其中：c_{ij}表示城市i到城市j的交通成本; p_i表示城市i的门票价格。

接着我们建立了问题三的单目标优化模型：

目标函数：

最大化访问的城市数量(不包括广州):

$$max \sum_{i \in N \backslash \{广州\}} y_i$$

其中N是所有城市的集合。

决策变量：

•x_{ij}(路径存在性变量)是二元变量表示从城市i到城市j是否有直接路径。如果从城市i到城市j有直接路径则为 1，否则为 0。

• y_i(城市访问变量)是二元变量表示城市i是否被访问。如果城市i被访问则为 1，否则为 0。

• z_{id}(每日出发城市变量)是二元变量表示第d天是否从城市i开始。如果第d天从城市i开始则为 1，否则为 0

约束条件;

1. 每个城市（除了广州）只能被访问一次：

$$\sum_{j \in N \backslash \{i, 广州\}} x_{ij} = y_i, \ \forall i \in N \backslash \{广州\}$$

2.每日开始约束：确保每一天从一个城市开始。

$$\sum_{i \in N \backslash \{广州\}} z_{id} = 1, \quad \forall d \in D$$

其中D表示所有可能的天数。

3.连接z和x变量的约束：

$$\sum_{d\in D} z_{id} \geq x_{ij}, \quad \forall i,j \in N\backslash\{\text{广州}\}, i \neq j$$

4.每日旅行时间限制：每天的旅行时间和游玩时间总和不超过 12 小时。

$$\sum_{i\in C\backslash\{\text{广州}\}} play\_time_i z_{id} + \sum_{i,j\in C\backslash\{\text{广州}\}} travel\_time_{ij}\, x_{ij} \leq 12, \quad \forall d \in D$$

5.从广州出发并返回广州：

$$\sum_{j\in N\backslash\{\text{广州}\}} x_{\text{广州}j} = 1$$

$$\sum_{i\in N\backslash\{\text{广州}\}} x_{i\text{广州}} = 1$$

6.旅行路径连续性约束：

$$\sum_{j\in N\backslash\{i,\text{广州}\}} x_{ij} = \sum_{j\in N\backslash\{i,\text{广州}\}} x_{ji}, \ \forall i \in N\backslash\{\text{广州}\}$$

对于这样一个混合整数线性规划模型，我们采用基于分支定界法的 *CBC* 求解器来求解该模型：

分支定界法求解流程如下：

分支定界法求解流程

***Step* 1** 初始化：

- 定义原始问题为原问题P
- 解决松弛问题P^* (即去掉整数约束的线性规划问题)

***Step* 2** 计算松弛问题P^*的最优解z^*，如果z^*是一个整数解，则它是原问题P的最优解。 。否则，继续下一步。

***Step* 3** 选择一个非整数变量x_i并将其分割成两个子问题:

- 子问题$P_1: x_i \leq \lceil x_i^* \rceil$ • 子问题$P_2: x_i \geq \lceil x_i^* \rceil$

***Step* 4** 剪枝：如果子问题P_i的最优解z_i^*不小于z^{best},则剪枝： $z_i^* \geq z^{best} \Rightarrow$剪枝

***Step* 5** 更新最佳解：如果子问题P_i的最优解z_i^*是整数解且小于z^{best},则更新z^{best}:

***Step* 6** 当所有子问题都被解决且没有更优的解可以找到时，算法终止。

最终求解结果如下：

旅行城市:广州 → 中山 → 丽江 → 成都 → 汕尾 → 泸州 → 济南 → 潍坊 → 珠海 → 广州

游玩时间：140.93 小时

旅行成本：7081.3

旅行景点：孙中山故居纪念馆 → 文峰寺 → 西岭雪山 → 玄武山（元山寺） → 方山 → 灵岩寺 → 云门山风景区黄杨山

图 7 单目标优化旅行路径

八、问题四模型的建立与求解

8.1 问题预分析

通过分析问题要求：尽可能的游览更多的城市，又需要使门票和交通的总费用尽可能的少。我们分析出该题要求构建多目标优化模型，但是多目标优化模型难以求解，所以我们在本题中采用加权法来将多目标模型转化为单目标模型

优化模型的目标函数

最小化加权后的总成本减去访问城市数量的乘积，即：

$$min\alpha \cdot C - \beta \cdot N$$

其中：C是总成本(包括门票和交通费用)；N是访问的城市数量；α和β分别是成本和城市数量的权重系数。

目标函数可以表示为：

$$min\left(\alpha \cdot \sum_{i\in cities j\in cities, i\neq j} c_{ij} \cdot x_{ij} + \alpha \cdot \sum_{i\in cities, i\neq 广州} p_i \cdot y_i - \beta \cdot \sum_{i\in cities, i\neq 广州} y_i\right)$$

其中：α和β分别是成本和城市数量的权重系数。

8.2 优化模型概览

目标函数：

$$min\left(\alpha \cdot \sum_{i\in N j\in N, i\neq j} c_{ij} \cdot x_{ij} + \alpha \cdot \sum_{i\in N, i\neq 广州} p_i \cdot y_i - \beta \cdot \sum_{i\in N, i\neq 广州} y_i\right)$$

约束条件：

$$\begin{cases} \sum_{j\in N\backslash\{i,\text{广州}\}} x_{ij} = y_i, \forall i \in N\backslash\{\text{广州}\} \\ \sum_{i\in N\backslash\{\text{广州}\}} z_{id} = 1, \quad \forall d \in D \\ \sum_{d\in D} z_{id} \geq x_{ij}, \quad \forall i,j \in N\backslash\{\text{广州}\}, i \neq j \\ \sum_{j\in C\backslash\{\text{广州}\}} x_{\text{广州}j} = 1 \\ \sum_{i\in N\backslash\{\text{广州}\}} x_{i\text{广州}} = 1 \\ \sum_{j\in C\backslash\{i,\text{广州}\}} x_{ij} = \sum_{j\in N\backslash\{i,\text{广州}\}} x_{ij}, \forall i \in C\backslash\{\text{广州}\} \\ \sum_{i\in C\backslash\{\text{广州}\}} play\_time_i z_{id} + \sum_{i,j\in C\backslash\{\text{广州}\}} travel\_time_{ij}\, x_{ij} \leq 12, \quad \forall d \in D \end{cases}$$

8.3 优化模型求解

我们选择采用第三问的分支定界法来对第四问的优化模型进行求解，最终结果如下：

旅行城市:广州 →中山 → 汕尾 → 泸州 → 珠海 → 贵阳 → 广州

游玩时间: 128.93 小时

旅行成本：4310.5

旅行景点：孙中山故居纪念馆 →玄武山（元山寺） →方山 →黄杨山 →黔灵山公园

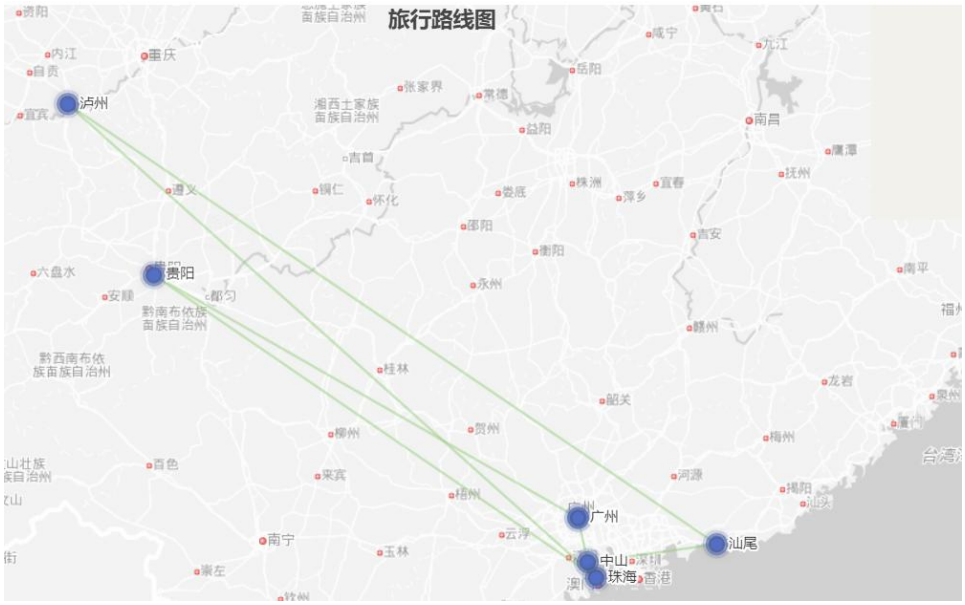


图 8 多目标优化旅行路径

从第四问的结果中我们可以看出，在多目标前提下，问题四游览的城市相较于问题三要更少，其次有些城市的选取明显考虑到了成本因素，例如选择了贵阳作为额外的一个目的地。

九、问题五模型的建立与求解

9.1 数据预处理

问题五放开了城市的限制，但是对山景增加了约束，所以我们首先从原始数据中筛选出了所有的山景，接着挑选出了每座城市山景评分最高的景点来为本问优化模型的求解提供基础，部分处理后数据如下：

表 12 问题五部分处理结果

名字	开放时间	评分	建议游玩时间	门票	来源城市
七仙岭温泉	08:00-22:30	5	6	30	保亭
七娘山	9:00-18:00	4.7	3	68	深圳
万绿谷	9:00-17:30	4.2	5	30	河源
三清山梯云岭	7:00-20:00	5	1.5	2	上饶

9.2 优化模型建立与求解

第五问要求在 144 小时内尽可能多地游览中国的山景，同时使门票和交通的总费用尽可能地少。我们选择使用 ε-约束法（*ε-constraint method*）来处理多目标优化问题。

ε-约束法：

ε-约束法是一种解决多目标优化问题的方法，它将其中一个目标作为硬约束，而将其他目标作为优化目标。在这种情况下，我们选择将**费用最小化作为硬约束，将访问尽可能多的山景作为优化目标**。

目标函数：

我们希望最大化访问的山景数量，同时将总费用控制在一个可接受的范围内。我们可以将费用设置为一个上限C_{max},并通过优化访问的山景数量N来实现这一目标。

$$maxN = \sum_{i \in cities} y_i$$

额外约束条件：

1.从入境城市出发并返回：

$$\sum_{j \in cities, j \neq S} x_{Sj} = 1$$

$$\sum_{j \in cities, j \neq S} x_{jS} = 1$$

2.总费用C必须小于或等于C_{max}:

$$C = \sum_{i \in cities} \sum_{j \in cities, i \neq j} c_{ij} \cdot x_{ij} + \sum_{i \in cities} p_i \cdot y_i \leq C_{max}$$

由于上述建立的混合整数二次规划模型涉及较多的决策变量和约束条件，使用常规算法计算最优解难度较大，因此使用遗传算法来解。遗传算法是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程的计算模型，是一种通过模拟自然进化过程探索最优解的方法。使用 *Python* 程序设计编写遗传算法对该规划进行求解，分别设置最大进化次数为 1500、突变概率为 0.5 以及交叉概率为 0.7

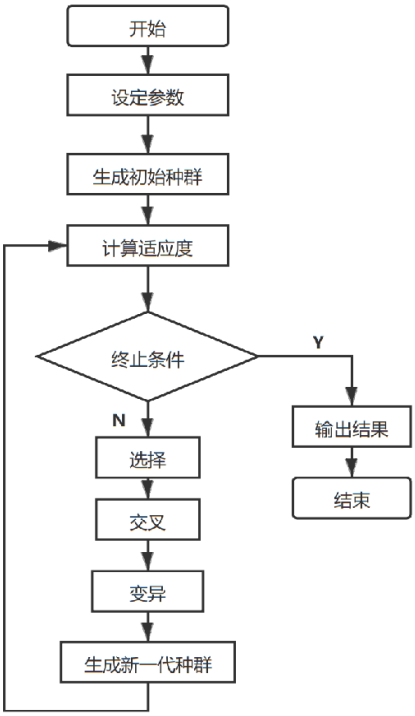


图 9 遗传算法流程图

最终结果如下：

旅行城市:北京 →石家庄→ 南京 →上饶 → 长沙 → 承德 → 北京

游玩时间: 131.12 小时

旅行成本：4361.5

旅行景点：银山塔林 → 五岳寨风景区 → 梅花山 → 鄣山村 → 汾山风景名胜区 → 双塔山

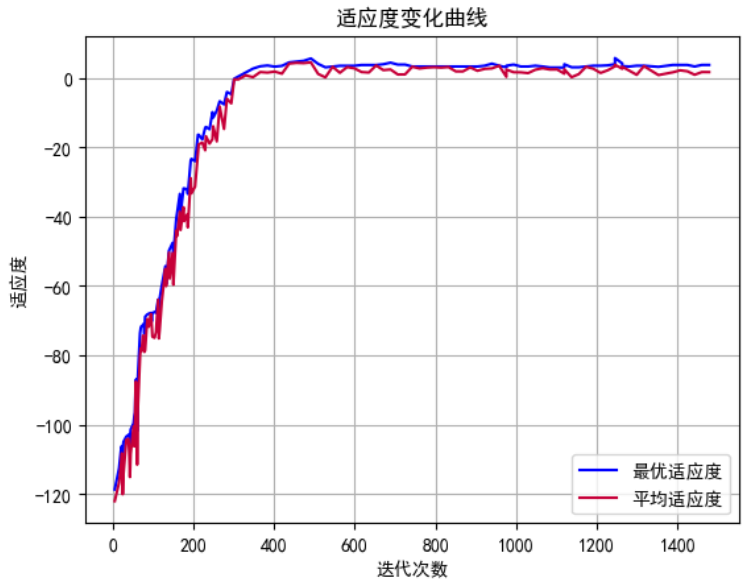


图 10 适应度变化图

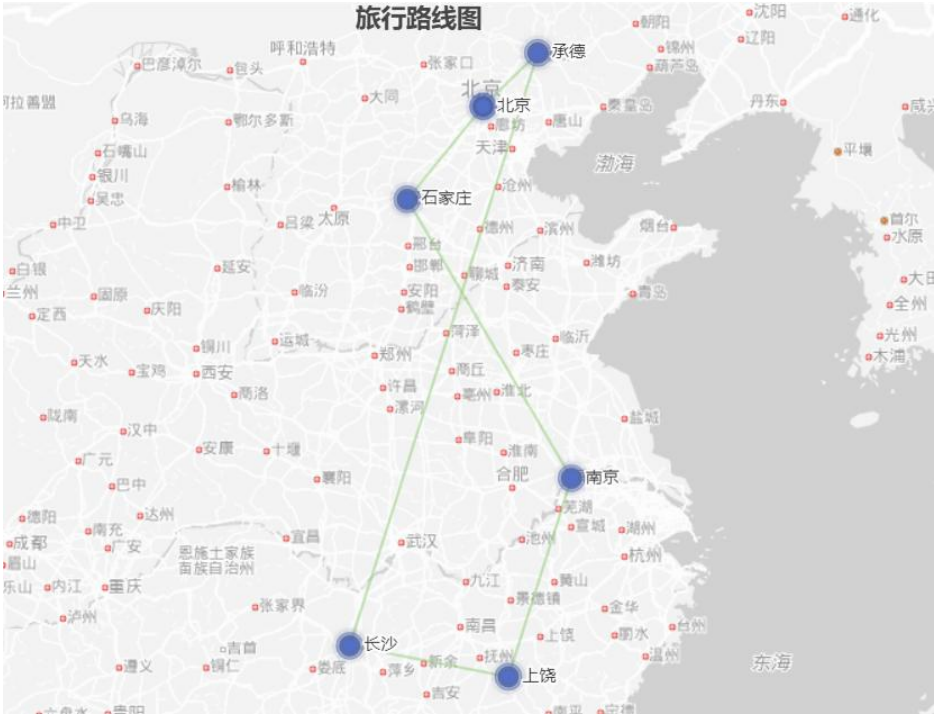


图 11 问题五旅游路径

旅游路径结合实际情况分析：

• 北京作为起点和终点，是合理的，因为北京是重要的交通枢纽，拥有多个国际机场，便于游客入境和离境。

• 石家庄、南京、上饶、长沙和承德都是中国内地的旅游城市，其中石家庄和南京是中国较大的城市，而上饶、长沙和承德则以其独特的自然风光而闻名。

• 这些城市之间的连接可以通过高铁实现，考虑到高铁在中国的发达程度，这种交通方式是可行的。

十、模型的评价与改进

模型优点：

(1)全面性：模型综合考虑了多个影响旅游吸引力的因素，包括城市规模、环境保护水平、人文底蕴、交通便利性、气候条件以及美食等，使得评估结果更贴近实际。

(2)灵活性：模型能够根据不同优化目标进行调整，既可以最大化访问的城市数量，也可以最小化总费用，适应不同游客的需求。。

(3)可靠性：使用了熵权法和 *TOPSIS* 法来确定指标权重，提高了评价结果的客观性和可靠性。

(4)扩展性：模型结构清晰，易于添加新的评价指标或调整现有指标权重，以适应不断变化的旅游市场。

模型缺点：

(1)数据依赖性：模型高度依赖于数据的完整性和准确性，如果数据质量不高，可能会影响模型的有效性。

(2)简化假设：有些假设可能过于简化，忽略了个人偏好和实际情况的差异。

(3)求解效率：对于大规模问题，分支定界法和遗传算法可能会遇到求解时间较长的问题。

改进方向：

(1)求解方法优化：探索更高效的求解算法，如启发式算法或近似算法，以减少求解时间。

(2)费用模型改进：采用更灵活的费用估计模型，考虑季节性变化、折扣优惠等因素。模型扩展：考虑加入更多的评价指标，如安全指数、文化活动丰富度等，进一步完善模型。

参考文献

[1]玄光南，程润伟. 遗传算法与工程优化[M]. 北京：清华大学出版社，2004

[2]方苏杰等. 基于旅行费用约束的景点及路径动态规划研究[J]. 计算机科学, 2019, (1): 12-23. DOI: 10.3969/j.issn.1000-386x.2018.12.061.

[3]王艳等. 旅游路线规划数学模型的建立与应用探讨[J]. 数学的实践与认识, 2016, 46(15): 125-133.

[4]张丽蓉，尤七七，罗灵茜，等. TOPSIS 法权重的选取：几种赋权方法的对比[J]. 统计学与应用，2023, 12(4): 901-909. DOI: 10.12677/sa.2023.124095.

[5]刘红雨，刘友存，孟丽红，等. 熵权法在水资源与水环境评价中的研究进展[J]. 冰川冻土，2022, 44(1): 299-306. DOI: 10.7522/j.issn.1000-0240.2021.0131.

[6]王欣，牟唯嫣，贾晓芳，等. 基于熵权法的 TOPSIS 综合评价法在城市公共交通中的应用[J]. 应用数学进展，2021, 10(12): 4253-4261. DOI: 10.12677/aam.2021.1012452.

[7]常亮, 孙文平, 张伟涛等. 旅游路线规划研究综述[J]. 智能系统学报, 2019, 14(1): 82-92.

[8]刘明明，崔春风，童小娇，等. 混合整数非线性规划的算法软件及最新进展. 中国科学：数学，2016, 46(1): 1-20. DOI: 10.1360/N012014-00278.

[9]Boukouvala F, Misener R, Floudas C A. Global optimization advances in mixed integer nonlinear programming, MINLP, and constraint derivative-free optimization, CDFO. European Journal of Operational Research, 2016, 252(3): 831-842. DOI: 10.1016/j.ejor.2015.12.01

附录

附录 1：支撑材料文件列表

附录 2：景点数据

附录 3：部分核心代码

附录 1：支撑材料文件列表

TOP50 景点数据
山类景点数据
问题一描述性统计与求解
熵权法与 TOPSIS
单目标优化与分支定界法
多目标优化与分支定界法
多目标优化与遗传算法

附录 2：景点数据

TOP50 景点数据

名字	开放时间	评分	建议游玩时间	门票	来源城市	纬度	经度
五指山 Wuzhishan	7:30-16:00	5	2.5	0	三亚	18.2528	109.5119
孙中山故居纪念馆 Sun Yat-Sen Memorial Hall	09:00-17:30	4.6	2.5	0.5	中山	22.5176	113.3926
南明山景区 Nanming Mountain	07:00-18:3	4.6	12	7	丽水	28.46763	119.922796

文峰寺 Wenfeng Temple	08:30-18:00	4.8	3.5	5	丽江	26.8575	100.2278
八达岭长城 Badaling Great Wall	07:30-16:00	4.7	3.5	35	北京	39.9042	116.4074
太蓬山 Tai Peng Mountain	08:00-18:00	4.7	1	100	南充	30.8373	106.1107
温岭方山景区 Wenling Fangshan Scenic Area	8:00-16:30	4.7	12	32	台州	28.6564	121.4208
黄河大峡谷?老牛湾旅游区 The Yellow River Grand Canyon Lao Niu Wan Tourist Area	8:00-19:00	4.4	1.5	35	呼和浩特	40.8426	111.7492
印象林花生态谷 Impression Linhua Ecological Valley	9:00-16:30	4.5	2.5	60	咸宁	29.8412	114.3224
嘉峪关文物景区	8:30-18:00	4.4	1.5	120	嘉峪关	39.7731	98.2891
鸡足山 Jizu Mountain	08:00-18:00	4.7	3.5	50	大理	25.6939	100.1617
天柱山 Tianzhu Mountain	7:00-17:30	4.7	108	130	安庆	30.5429	117.0638
贵清山旅游风景区 Guiqing Mountain Tourism Scenic Area	8:30-18:00	4.5	4	50	定西	35.5811	104.6258
关山草原 Guanshan Grasslands	08:00-18:00	4.4	4.5	55	宝鸡	34.3628	107.2377
夹山国家森林公园 Jiashan National Forest Park	8:00-17:30	4.3	4.5	55	常德	29.0316	111.6985
恩施地心谷景区 Enshi Dixin Valley Scenic Spot	08:00-	4.4	3.5	105	恩施	30.2722	109.4882

广东罗浮山风景名胜区 Luofu Mountain	15:00 07:00-18:00	4.4	108	54	惠州	23.1103	114.4158
西岭雪山 Xiling Snow Mountain	09:00-17:30	4.6	4	120	成都	30.5728	104.0668
捺山地质公园 Nashan Geological Park	8:00-17:00	4.4	2.5	66	扬州	32.3932	119.4132
药王山 Yaowang Mountain	7:00-19:00	4.6	1.5	38	拉萨	29.65	91.13
珏山 Jueshan Mountain	8:00-5:00	4.6	3.5	50	晋城	35.4907	112.8511
胡雪岩故居 Hu Xueyan's Former Residence	08:00-17:00	4.7	1.5	20	杭州	30.2741	120.1551
舜皇山国家森林公园 Shunhuangshan National Forest Park	8:00-18:00	4.3	3.5	35	永州	26.420394	111.612564
玄武山（元山寺）Xuanwu Mountain (Yuanshan Temple)	8:30-17:00	4.6	1.5	8	汕尾	22.7845	115.3753
方山 Fang Mountain	8:00-17:00	4.3	3.5	6	泸州	28.8717	105.4423
灵岩寺 Lingyan Temple	8:00-17:00	4.7	2.5	65.5	济南	36.67	117
王屋山-黛眉山地质博物馆	8:00-18:00	3	2.5	50	济源	35.0672	112.6025
云洞岩 Yundong Cave	08:00-18:00	4.6	2	6	漳州	24.5135	117.6473
云门山风景区 Yunmenshan Scenic Spot	8:00-17:30	4.6	3.5	1	潍坊	36.7069	119.1618

黄杨山 Huangyang Mountain	8:30-17:30	4.6	3.5	1	珠海	22.27	113.5767
贡嘎山 Mount Gongga	8:00-18:00	4.7	72	0	甘孜	30.04952	101.96231
青云山 Qingyun Mountain	8:00-16:30	4.5	72	32	福州	26.0745	119.2965
黔灵山公园 Qianling Mountain Park	07:00-18:00	4.6	3.5	9	贵阳	26.6477	106.6302
姑婆山 Gupo Mountain	08:00-17:00	4.6	12	60	贺州	24.4035	111.566

部分山类景点数据

名字	开放时间	评分	建议游玩时间	门票	来源城市
巴山云顶旅游度假区	08:00-16:00	4	0.35	180	达州
千年瑶寨 Millennium Yaozhai	09:00-17:30	4.7	0.4	100	清远
华山 Mount Hua	08:00-16:00	4.7	0.4	160	渭南
天柱山 Tianzhu Mountain	7:00-17:30	4.7	0.4	130	安庆
太行大峡谷景区 Taihang Grand Canyon Scenic Area	8:00-17:30	4.7	0.5	95	安阳
梅里雪山国家公园 Meili Snow Mountain	6:00-16:30	4.7	0.75	8	迪庆
丹霞山 Mount Danxia	8:30-17:20	4.6	0.75	50	韶关
老君山景区 Luoyang Laojun Mountain	8:00-18:00	4.6	0.75	96	洛阳
尖峰岭国家森林公园 Jianfeng (Sharp Peak) Ridge National Forest Park	7:00-0:00	4.4	0.75	5	乐东
广东罗浮山风景名胜区 Luofu Mountain	07:00-18:00	4.4	0.75	54	惠州

大崳山岛 Dayushan Island	8:00-17:00	4.3	1	70	宁德
湖北三角山旅游度假区 Hubei Mount Sanjiao Tourism and Resort Area	8:00-17:00	4.3	1	48	黄冈
大容山森林公园 Darong Mountain Forest Park	8:00-17:00	4.2	1	58	玉林
僖山景区	08:00-18:00	5	1	30	商丘
马鞍山马钢盆山 Ma'anshan Magangpen Mountain	8:00-17:00	5	1	5	马鞍山
少室山景区 The Shaoshi Mountain Scenic Area	8:00-17:00	4.8	1	10 0	郑州
灵空山 Lingkong Mountain	6:00-18:00	4.7	1	8	长治
西双版纳猴山 Sipsongpanna Monkey Mountain	08:00-17:00	4.7	1	20	西双版纳
贡嘎山 Mount Gongga	8:00-18:00	4.7	1	0	甘孜
长白山南坡景区 Changbai Mountain South Slope Sceneic Area	09:30-11:30	4.7	1	10 5	白山

附录 3：部分核心代码

问题一描述性统计与求解核心代码

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
import warnings
warnings.filterwarnings('ignore')
matplotlib.rcParams['font.family'] = 'sans-serif'
matplotlib.rcParams['font.sans-serif'] = ['SimHei']  # 使用思源黑体
matplotlib.rcParams['axes.unicode_minus'] = False

df = pd.read_excel("C:\\Users\\30766\\Desktop\\df.xlsx")
df1 = pd.read_excel("C:\\Users\\30766\\Desktop\\relevant_data.xlsx")
median_values = df['评分'].median()
df1['评分'] = pd.to_numeric(df1['评分'], errors='coerce')
df1.drop_duplicates(subset='名字', inplace=True)
# 找出最高评分
best_score = df1['评分'].max()
```

```
print(f"最高评分（BS）是: {best_score}")

# 计算第一四分位数和第三四分位数
Q1 = df1['评分'].quantile(0.25)
Q3 = df1['评分'].quantile(0.75)

# 计算四分位数间距
IQR = Q3 - Q1
# 确定异常值的界限
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
# 剔除异常值
df_cleaned = df1[(df1['评分'] >= lower_bound) & (df1['评分'] <= upper_bound)]

# 2. 计算获评最高评分的景点数量
count_best_score = df2[df2['评分'] == best_score].shape[0]
```

熵权法与 TOPSIS 核心代码

```
data= pd.read_excel("C:\\Users\\30766\\Desktop\\问题二.xlsx")

data.info()
data_numeric = data.drop(['城市来源'], axis=1)

from sklearn.preprocessing import MinMaxScaler
# 数据标准化
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data_numeric)
scaled_df = pd.DataFrame(scaled_data, columns=data_numeric.columns,
index=data.index)

# 构建新指标
def calculate_new_indicators(data):
# 旅游服务能力
α1, α2 = 0.38, 0.62
β1, β2 = 0.5, 0.5  # 同上
γ1, γ2 = 0.5, 0.5  # 同上
δ1, δ2, δ3 = 0.4, 0.3, 0.3  # 同上
ϵ1, ζ1 = 0.5, 0.5  # 同上

# 旅游接待次数
data['旅游接待次数'] = data['餐馆数量'] * α1 + data['特色美食数量'] * α2
# 平均接待游客量
```

```
data['平均接待游客量'] = (data['适宜旅游天数'] / np.abs(data['年平均气温 (°C)'])) * β1
# 单次最大接待量
data['单次最大接待量'] = data['公共交通覆盖率'] * γ1 * data['线路密度'] * γ2
# 旅游体验一致性
data['旅游体验一致性'] = data['历史遗迹数量'] * δ1 + data['博物馆数量'] * δ2 + data['文化活动频次'] * δ3
# 旅游景点的分布合理性
data['旅游景点的分布合理性'] = data['绿化覆盖率'] / data['文化设施数量'] * ϵ1
# 合理接待比例
data['合理接待比例'] = data['公共交通覆盖率'] / data['高速公路里程'] * ζ1

return data

# 构建新指标
new_indicators_df = calculate_new_indicators(scaled_df)

# 熵权法计算指标权重
def calculate_entropy_weights(data):
# 计算比例矩阵
p = data / data.sum()
# 计算信息熵
e = -np.where(p > 0, p * np.log2(p), 0)
entropy = e.sum(axis=0) / np.log2(len(data))
# 计算差异系数
g = 1 - entropy
# 计算熵权
weights = g / g.sum()
return weights

# 计算熵权
entropy_weights = calculate_entropy_weights(new_indicators_df)

# 输出权重
print("Entropy Weights:")
for i, col in enumerate(new_indicators_df.columns):
print(f"{col}: {entropy_weights[i]}")

# TOPSIS 法计算综合得分
def topsis(data, weights):
# 正负理想解
positive_ideal_solution = data.max()
negative_ideal_solution = data.min()

# 计算距离
```

```
d_positive = np.sqrt(((data - positive_ideal_solution)**2).sum(axis=1))
d_negative = np.sqrt(((data - negative_ideal_solution)**2).sum(axis=1))

# 接近度系数
closeness_coefficients = d_negative / (d_positive + d_negative)

# 加权接近度系数
weighted_closeness = (closeness_coefficients * weights).sum(axis=1)

return weighted_closeness

# 计算综合得分
weighted_closeness = topsis(scaled_df, entropy_weights)

# 添加综合得分到原始 DataFrame
data['综合得分'] = weighted_closeness

# 排序并选择前 50 个城市
top_cities = data.sort_values(by='综合得分', ascending=False).head(50)[['来源城市', '综合得分']]

# 输出结果
print(top_cities)
```

单目标优化与分支定界法核心代码

```
# 加载景点数据
best_spots_data = pd.read_excel("C:\\Users\\30766\\Desktop\\best_spots_data.xlsx")

guangzhou_data = {
'名称': 'Guangzhou',
'建议游玩时间': 0,  # 广州不计入游玩时间
'门票': 0,  # 广州不计入门票费用
'来源城市': '广州',
'纬度': 23.129110,
'经度': 113.264385
}

# 将广州数据追加到 DataFrame
best_spots_data = pd.concat([best_spots_data, pd.DataFrame(guangzhou_data, index=[4])])

# 设置索引为'来源城市'以方便访问
best_spots_data.set_index('来源城市', inplace=True)

# 从 DataFrame 中提取必要的数据
```

```
play_times = best_spots_data['建议游玩时间']
ticket_prices = best_spots_data['门票']

# 获取所有城市的列表
cities = best_spots_data.index.tolist()

# 高铁平均速度
average_speed = 300  # 千米/小时

# 计算两个城市之间的直线距离
def calculate_distance(city1, city2):
coord1 = (best_spots_data.loc[city1]['纬度'], best_spots_data.loc[city1]['经度'])
coord2 = (best_spots_data.loc[city2]['纬度'], best_spots_data.loc[city2]['经度'])
return haversine(coord1, coord2, unit=Unit.KILOMETERS)

# 估算两城市间的旅行时间
def estimate_travel_time(city1, city2):
distance = calculate_distance(city1, city2)
travel_time = distance / average_speed
return travel_time

# 构建旅行时间矩阵
travel_times = pd.DataFrame(index=cities, columns=cities)
for city1 in cities:
for city2 in cities:
travel_times.loc[city1, city2] = estimate_travel_time(city1, city2)

prob = LpProblem("TravelRoute", LpMaximize)

# 定义决策变量
x = LpVariable.dicts("Route", ((i, j) for i in cities for j in cities if i != j), 0, 1,
cat='Binary')  # 路径变量
y = LpVariable.dicts("Visited", cities, 0, 1, cat='Binary')  # 是否访问过某城市变量
z = LpVariable.dicts("DayStart", ((i, d) for i in cities for d in range(12)), 0, 1,
cat='Binary')  # 某天开始于某城市的变量

# 总成本计算（包括门票和旅行时间成本）
total_cost = lpSum(ticket_prices[city] * y[city] for city in cities if city != '广州') + \
lpSum(travel_times.loc[i, j] * 0.5 * x[i, j] for i in cities for j in cities if i != j)  # 假设每小
时固定的旅行成本
# 目标函数：最大化访问的城市数量
prob += lpSum(y[city] for city in cities if city != '广州'), "MaxCities"

# 约束条件
```

```
# 每个城市（除广州外）只能访问一次
for i in cities:
if i != '广州':
prob += lpSum(x[i, j] for j in cities if i != j) == y[i], f"VisitConstraint_{i}"

# 每天旅行时间限制（例如，每天 10 小时）
daily_travel_time_limit = 10  # 小时

# 约束确保每天从一个城市开始
for d in range(12):
prob += lpSum(z[i, d] for i in cities if i != '广州') == 1, f"StartOfDay_{d}"

# 约束连接 z 和 x 变量
for i in cities:
if i != '广州':
for j in cities:
if i != j:
prob += lpSum(z[i, d] for d in range(12)) >= x[i, j], f"LinkZ_X_{i}_{j}"

# 约束确保每天的旅行时间和游玩时间不超过每天的限制
for d in range(12):
prob += lpSum((play_times[city] * z[city, d] for city in cities if city != '广州')) + \
lpSum((travel_times.loc[i, j] * x[i, j] for i in cities for j in cities if i != j and i != '广州')) <=
daily_travel_time_limit, f"DailyTimeLimit_{d}"

# 确保从广州出发并返回广州
prob += lpSum(x['广州', j] for j in cities if j != '广州') == 1, "StartFromGuangzhou"
prob += lpSum(x[i, '广州'] for i in cities if i != '广州') == 1, "ReturnToGuangzhou"

# 确保旅行路径的连续性
for i in cities:
if i != '广州':
prob += lpSum(x[i, j] for j in cities if j != i and j != '广州') == lpSum(x[j, i] for j in cities if
j != i and j != '广州'), f"ContinuityConstraint_{i}"

# 解决模型
status = prob.solve()

# 输出结果
if LpStatus[prob.status] == 'Optimal':
print("找到了最优解！")
print("访问的城市数量:", value(lpSum(y[city] for city in cities if city != '广州')))
visited_cities = [city for city in cities if y[city].varValue == 1 and city != '广州']
print("访问的城市:", visited_cities)
```

多目标优化与分支定界法部分代码

```
# 读取数据
best_spots_data = pd.read_excel("C:\\Users\\30766\\Desktop\\best_spots_data.xlsx")

guangzhou_data = {
'名称': 'Guangzhou',
'建议游玩时间': 0,  # 广州不计算游玩时间
'门票': 0,  # 广州不计算门票价格
'来源城市': '广州',
'纬度': 23.129110,
'经度': 113.264385
}

# 将广州的数据追加到 DataFrame 中
best_spots_data = pd.concat([best_spots_data, pd.DataFrame(guangzhou_data, index=[4])])

# 设置索引为'来源城市'以便于访问
best_spots_data.set_index('来源城市', inplace=True)

# 从 DataFrame 中提取必要的数据
play_times = best_spots_data['建议游玩时间']
ticket_prices = best_spots_data['门票']

# 城市集合
cities = best_spots_data.index.tolist()

# 高速列车平均速度
average_speed = 300  # 千米/小时

# 计算两城市之间的直线距离
def calculate_distance(city1, city2):
coord1 = (best_spots_data.loc[city1]['纬度'], best_spots_data.loc[city1]['经度'])
```

```
coord2 = (best_spots_data.loc[city2]['纬度'], best_spots_data.loc[city2]['经度'])
return haversine(coord1, coord2, unit=Unit.KILOMETERS)

# 估计两城市之间的旅行时间
def estimate_travel_time(city1, city2):
distance = calculate_distance(city1, city2)
travel_time = distance / average_speed
return travel_time

# 构建旅行时间矩阵
travel_times = pd.DataFrame(index=cities, columns=cities)
for city1 in cities:
for city2 in cities:
travel_times.loc[city1, city2] = estimate_travel_time(city1, city2)

# 创建一个 LpProblem 实例
prob = LpProblem("TravelRoute", LpMinimize)

# 定义决策变量
x = LpVariable.dicts("Route", ((i, j) for i in cities for j in cities if i != j), 0, 1, cat='Binary')
y = LpVariable.dicts("Visited", cities, 0, 1, cat='Binary')
z = LpVariable.dicts("DayStart", ((i, d) for i in cities for d in range(12)), 0, 1, cat='Binary')

# 每公里的交通费用
traffic_cost_per_km = 1

# 计算两城市之间的交通费用
def calculate_traffic_cost(city1, city2):
distance = calculate_distance(city1, city2)
return distance * traffic_cost_per_km

# 将交通费用加入总成本
total_traffic_cost = lpSum(calculate_traffic_cost(i, j) * x[i, j] for i in cities for j in cities if
i != j)

# 更新目标函数
total_cost = lpSum(ticket_prices[city] * y[city] for city in cities if city != '广州') + \
total_traffic_cost  # 交通费用
num_cities = lpSum(y[city] for city in cities if city != '广州')

alpha = 1  # 成本权重系数
beta = 1    # 城市数量权重系数
prob += alpha * total_cost - beta * num_cities, "WeightedObjective"
```

```
# 约束条件
# 每个城市（除了广州）只能访问一次
for i in cities:
if i != '广州':
prob += lpSum(x[i, j] for j in cities if i != j) == y[i], f"VisitConstraint_{i}"

# 每天旅行时间限制
daily_travel_time_limit = 12  # 小时

# 确保每一天从一个城市开始
for d in range(12):
prob += lpSum(z[i, d] for i in cities if i != '广州') == 1, f"StartOfDay_{d}"

# 约束链接 z 和 x 变量
for i in cities:
if i != '广州':
for j in cities:
if i != j:
prob += lpSum(z[i, d] for d in range(12)) >= x[i, j], f"LinkZ_X_{i}_{j}"

# 确保每天的旅行时间和游玩时间不超过每日限制
for d in range(12):
prob += lpSum((play_times[city] * z[city, d] for city in cities if city != '广州')) + \
lpSum((travel_times.loc[i, j] * x[i, j] for i in cities for j in cities if i != j and i != '广州')) <=
daily_travel_time_limit, f"DailyTimeLimit_{d}"

# 确保从广州出发并返回广州
prob += lpSum(x['广州', j] for j in cities if j != '广州') == 1, "StartFromGuangzhou"
prob += lpSum(x[i, '广州'] for i in cities if i != '广州') == 1, "ReturnToGuangzhou"

# 确保旅行路径的连续性
for i in cities:
if i != '广州':
prob += lpSum(x[i, j] for j in cities if j != i and j != '广州') == lpSum(x[j, i] for j in cities if
j != i and j != '广州'), f"ContinuityConstraint_{i}"

# 解决问题
status = prob.solve()

# 输出结果
if LpStatus[prob.status] == 'Optimal':
print("找到最优解！ ")
print("访问的城市数量:", value(num_cities))
visited_cities = [city for city in cities if y[city].varValue == 1 and city != '广州']
```

```
print("访问的城市:", visited_cities)

# 计算总花费时间
total_play_time = lpSum(play_times[city] * y[city] for city in cities)
total_travel_time = lpSum(travel_times.loc[i, j] * x[i, j] for i in cities for j in cities if i != j)
total_spent_time = value(total_play_time + total_travel_time)
print("总花费时间:", total_spent_time, "小时")

# 计算总成本
total_cost_value = value(total_cost)
print("总成本:", total_cost_value)
else:
print("未找到最优解。")
```

多目标优化与遗传算法部分代码

```
# 高铁平均速度
average_speed = 300  # 千米/小时

# 计算两个城市之间的直线距离
def calculate_distance(city1, city2):
coord1 = (best_spots_data.loc[city1]['纬度'], best_spots_data.loc[city1]['经度'])
coord2 = (best_spots_data.loc[city2]['纬度'], best_spots_data.loc[city2]['经度'])
return haversine(coord1, coord2, unit=Unit.KILOMETERS)

# 估算两城市间的旅行时间
def estimate_travel_time(city1, city2):
distance = calculate_distance(city1, city2)
travel_time = distance / average_speed
return travel_time

# 构建旅行时间矩阵
travel_times = pd.DataFrame(index=cities, columns=cities)
for city1 in cities:
for city2 in cities:
travel_times.loc[city1, city2] = estimate_travel_time(city1, city2)

# 遗传算法参数
POPULATION_SIZE = 50  # 种群大小
P_CROSSOVER = 0.1  # 交叉概率
P_MUTATION = 0.5  # 变异概率
MAX_GENERATIONS = 500  # 最大代数
```

```
HALL_OF_FAME_SIZE = 1  # 名人堂大小

# 定义问题类型
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

# 创建个体
def create_individual():
return random.sample(cities, len(cities))

# 自定义部分匹配交叉算子
def cxPartialyMatchedStr(ind1, ind2):
size = min(len(ind1), len(ind2))
p1, p2 = {}, {}  # 使用字典来跟踪位置

# 初始化每个城市在个体中的位置
for i in range(size):
p1[ind1[i]] = i
p2[ind2[i]] = i

# 选择交叉点
cxpoint1 = random.randint(1, size)
cxpoint2 = random.randint(1, size - 1)
if cxpoint2 >= cxpoint1:
cxpoint2 += 1
else:  # 交换两个交叉点
cxpoint1, cxpoint2 = cxpoint2, cxpoint1

# 在交叉点之间应用交叉
for i in range(cxpoint1, cxpoint2):
# 记录要交换的值
temp1 = ind1[i]
temp2 = ind2[i]
# 交换匹配的值
ind1[i], ind1[p1[temp2]] = temp2, temp1
ind2[i], ind2[p2[temp1]] = temp1, temp2
# 更新位置信息
p1[temp1], p1[temp2] = p1[temp2], p1[temp1]
p2[temp1], p2[temp2] = p2[temp2], p2[temp1]

return ind1, ind2

# 适应度评估函数
def evaluate(individual):
```

```
# 检查个体中的所有城市是否都在旅行时间矩阵中
if not set(individual).issubset(set(travel_times.index)):
raise ValueError("一些城市不在旅行时间矩阵中。")

# 计算访问的景点数量
num_visited_mountains = sum(1 for i in individual if i != 'Guangzhou')

# 计算总成本（旅行时间 + 游玩时间）
total_cost = 0
for i in range(len(individual) - 1):
current_city = individual[i]
next_city = individual[i + 1]
total_cost += travel_times.loc[current_city, next_city]  # 两城市间旅行时间
if current_city != 'Guangzhou':
total_cost += play_times[current_city]  # 当前城市的游玩时间

# 应用 ε-约束：如果总成本超过阈值，则减少适应度
if total_cost > 10:  # 示例阈值
num_visited_mountains -= 1  # 作为惩罚减少访问的景点数量

return num_visited_mountains,

# 创建工具箱
toolbox = base.Toolbox()
toolbox.register("create_individual", create_individual)
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.create_individual)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", evaluate)
toolbox.register("mate", cxPartialyMatchedStr)  # 自定义交叉算子
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

# 初始化种群
population = toolbox.population(n=POPULATION_SIZE)

# 运行遗传算法
hof = tools.HallOfFame(HALL_OF_FAME_SIZE)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", np.mean)
stats.register("std", np.std)
stats.register("min", np.min)
stats.register("max", np.max)

population, logbook = algorithms.eaSimple(population, toolbox, cxpb=P_CROSSOVER,
```

```
mutpb=P_MUTATION,
ngen=MAX_GENERATIONS, stats=stats, halloffame=hof, verbose=True)

# 输出最佳解
best_individual = hof.items[0]
print("最佳个体:", best_individual)
print("适应度:", best_individual.fitness.values)
print("访问的城市:", best_individual)
print("总成本:", evaluate(best_individual)[0])
```