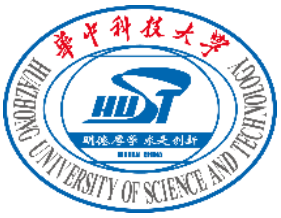


中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

学　校	武汉大学
参赛队号	22938880001
队员姓名	1.张华祖
	2.栗广岳
	3.李庞胤

中国研究生创新实践系列大赛

中国光谷·“华为杯”第十九届中国研究生数学建模竞赛

题 目 COVID-19 疫情期间生活物资的科学管理问题

摘 要：

新冠疫情在 2022 年伊始，频繁地大面积地出现爆发，秉承生命至上，人民至上的宗旨，使得政府不得不采取封闭式管理模式。在这一环境下，如何科学管理生活物资的发放（如蔬菜包）对疫情防控效率有着重要的意义，在全面提高疫情防控效率的同时，可以节约相关部门的人员投入与经费支出，同时也为今后的应急管理方案打下坚实的理论与实践基础。本文基于 SIR 传染病模型、最大覆盖模型、TOPSIS 评价模型等对蔬菜包发放的相关评价、调整以及选址问题做出了合理的设计，形成了生活物资智能化科学化管理系统。

针对问题一，需要判断疫情传播与蔬菜包发放的关系。本文首先构建了**SIR 传染病模型**来模拟未发放蔬菜包的条件下感染人数变化情况。对比真实的感染人数与模拟仿真得到的预测结果，从**定性**和**定量**两个角度判断相关性。从定性的角度，蔬菜包的加入使得疫情出现了明显拐点，因为认为两者存在相关关系；而从定量的角度，通过计算这两个变量的相关系数，判断蔬菜包发放与疫情有显著负相关关系，即蔬菜包的发放能够控制疫情，并进一步分析了原因。

针对问题二，对于投放点数量的评价与调整，利用**孤立森林算法**在各项指标中发现少数离群点，并阐述了不合理行政区设置的具体原因。随后，通过 **AdaBoost 分类器**，基于投放合理的行政区数据建立了回归模型，调整了不合理的投放数量；对于连续优化选址问题，通过栅格化的操作将该任务转化为离散化选址的问题，并针对栅格化后小区密度过大的现象，采用**栅格平滑**的方法模拟真实的小区分布情况。本文构建了**多目标优化的选址选取模型**，属于 NP 难问题。为了解决该问题，本文采用了一种**空间位置的编码方式**，使用**遗传算法**进行求解，计算得到最优的选址数量、规模及其潜在的备用场所位置。智能选址模型实现了选址数量尽可能小的同时，充分地供应了辖区内的所有人口，覆盖率达到了 100%。

针对问题三，需要寻找蔬菜包发放规律并评价和调整蔬菜包供应方案。对于发放规

律，结合问题一的结果与政府公告，本文将蔬菜包的发放划分为三个阶段：需求紧张阶段、供需平衡阶段和复工复市阶段。对于供应方案的评价和调整，我们综合考虑了蔬菜的运输损耗与过期变质，选取了量化生活物资发放的三大指标：人均蔬菜盈亏、疫情严重程度以及投放点工作强度。进一步，我们建立了 **TOPSIS 评价模型**，并利用**熵权法**的设定权重。对各行政单位的蔬菜包发放策略好坏进行综合评分，最终根据评价结果给出了具体的调整策略。

针对问题四，需要构建有序网络图，并考虑加入大卡车等货物载体约束后是否对预案产生影响。本文将有序网络图构建任务分为上-中游有序子网络和中-下游有序子网络两个子任务，结合附件三的路网拓扑关系和问题二的选址结果，基于 **TSP 模型**求解带权有向子图从而完成了有序网络图的构建。本文针对加入货物载体约束问题，提出了**两阶段贪婪优化算法**完成了有序网络图的重构。通过比较工作量指标，发现加入货物载体约束后，小卡车的工作量远远大于其他两种，会显著影响预案。

最后对模型的优缺点进行了分析，并提出了改进方法。

关键词：多目标优化；最大覆盖模型； 遗传算法；独立森林；TOPSIS 模型；旅行商问题；

目录

一、问题重述

1.1 问题背景

2019 年底，新冠病毒的爆发极大地威胁了人们的健康和经济活动，直至 2022 年仍未清零，在全国范围内了多次较大规模的爆发。我国在大规模疫情爆发期间采用封闭式管理方式从而实现疫情的快速清零，因此，如何在封闭管理时对各类人群进行科学的管理至关重要。

在疫情期间由于对确诊病例、无症状感染者及密切接触者采用集中治疗或隔离的手段，因此此类人群的科学管理较易实现。然而，对于大量、分散居家隔离人员实现统一化管理成为新的难点问题，主要有以下三个方面：生活物资的科学发放、特殊群体的管理以及为援助与医务人员提供高效服务。针对生活物资发放问题，蔬菜包的做法为疫情期间居民生活物资的有效发放起到了至关重要的作用，但其保质期较短，从而配送频率较高，容易造成人员间密切接触导致疫情传播。因此在不增加疫情传播风险的情况下，如何科学地供应食品，从而实现疫情期间生活物资的科学管理，对全面提高疫情防控效率的同时、节约相关部门的人员投入与经费支出有重要意义，并为今后的应急管理方案打下坚实的理论与实践基础。

1.2 问题提出

要求根据搜集到的长春市 9 个区的各项基础数据，不同区的隔离人口数量与生活物资投放点数量数据，9 个区的交通网络基础数据和主要小区的相关数据，建模分析以下问题：

问题 1：根据附件 1 中所提供的长春市 COVID-19 疫情期间病毒感染人数数据、其它附件数据以及搜集到的其他数据对长春市实行发放蔬菜包前后效果分析对长春市实行发放蔬菜包前后效果，并判断疫情的发展或被控制扑灭是否与生活物资发放方式有关；

问题 2：根据附件 3 中和附件 4 中有关数据，讨论投放点数量的合理性，并通过数学模型优化。此外，充分考虑未来疫情、自然灾害等特殊事件，对于政府储备物资和大规模物资分拣场所的位置与数量规模进行合理规划，并提出最优的选址数量、规模及其潜在的备用场所位置；

问题 3：根据附件 5 分析蔬菜包需求、发放规律，并根据附件 3 中的各小区位置与人口信息，评价并调整 4 月 10 日至 4 月 15 日蔬菜包供应方案；

问题 4：在第二、三问的基础上，结合附件 3，以节省人力（工作量按运输里程与小区居民人数乘积计算）、减少人员的直接、间接接触为目标，构建特殊时期保障居民生活物资供应的详细预案（有序网络图），其中，网络上游是各项物资来源（每个区选一个地点，参赛队可自行根据坐标选择），中游是各项物资的集散地(集散地数量自行选择，可以

先按附件 2 设置，再调整优化)，网络下游是长春市所有小区。在完成有序网络图后请进一步考虑用卡车运送物资，大卡车每辆可装 10 吨，小卡车每辆可装 4 吨，观察预案有无显著不同，并对照指标分析与评价给出的预案的优势。

二、问题分析

2.1 问题一分析

问题一要求我们结合附件 1 中所提供的长春市 COVID-19 疫情期间病毒感染人数数据及其它附件数据或能搜集到的数据对长春市实行发放蔬菜包前后效果进行判别与分析，即需要判断疫情传播与蔬菜包发放的相关性。因此，我们首先需要构建传染病模型来模拟未发放蔬菜包的条件下感染人数变化情况。将真实的感染人数与模拟仿真得到的预测结果进行对比，从定性和定量两个角度判断相关性。从定性的角度，结合真实的感染人数和预测结果判断拐点出现的日期，并和蔬菜包发放日期进行比较；而从定量的角度，通过对确诊人数和蔬菜包发放数量的这两个变量计算相关性，从而判断蔬菜包发放是否对疫情传播有影响，有何种影响，并分析原因。

2.2 问题二分析

问题二要求我们解决以下两个问题：一是讨论投放点数量的合理性，并通过数学模型进行优化；二是充分考虑未来疫情、自然灾害等特殊事件，对于政府储备物资和大规模物资分拣场所的位置与数量规模进行合理规划，并提出最优的选址数量、规模及其潜在的备用场所位置。

针对讨论投放点数量的合理性评价问题和优化问题，我们首先从实际出发，假设大部分行政区域都可以在蔬菜包发放过程中选取合适的投放点数量。因此已有数据中的投放数量普遍应该是合理，但仍有少部分值得进一步的优化。基于这种假设，合理性评价的问题即转换为异常探测问题。通过寻找数据中的异常，发现蔬菜包发放不合理的区域，并进行调整。首先，我们选择了大量评价投放点数量是否合理的指标，包括区域人口、小区数量、疫情数据、蔬菜包发放数量等等。然后，通过异常值挖掘方法，建立对投放点数量合理性进行评价的数学模型。由于大部分的投放数量合理，在筛选出异常值后对剩下的合理的投放数量与各影响因素构建定量的函数关系，可以得到投放数量的理论值，根据理论值对投放点数量进行调整。

针对优化选址问题，选址优化本质是一个连续选址的问题，但为了便于计算，可以通过栅格化的操作将该问题转化为离散化选址的问题。考虑未来疫情、自然灾害等特殊事件，因此首先需要提供足够的物资，并且应该具有较高的利用率，以防收到灾害等影响某些分拣所和储备物资的场所无法运转，因此该优化问题的目标应为：（1）设施覆盖范

围尽可能大；（2）在设施服务人口数一定的情况下，设施的利用率尽可能高。最后由于该模型属于 NP 难问题，因此可以选用启发式算法对模型进行求解。

2.3 问题三分析

问题三要求我们评价并调整蔬菜包供应方案，因此可以结合问题一的结果对疫情发展的阶段进行划分，结合成年人每天的蔬菜摄入，分阶段分析蔬菜包需求、发放规律。首先，需要选取一些量化生活物资发放的指标，然后设定这些指标权重并建立一个评价模型，对各行政单位的蔬菜包发放策略好坏进行综合评分，最后根据评价结果给出了具体的调整策略。

2.4 问题四分析

问题四要求我们构建一个有序网络图，并考虑加入小卡车和大卡车等货物载体约束是否对预案产生显著影响。有序网络图的构建可被分成上-中游有序子网络和中-下游有序子网络两个子任务，网络图的节点可根据附件三和问题二得到，有序子网络图的构建又可被拆解为每个子图的最优路径结果整合。因此，可以针对每个带权有向子图进行求解，从而得到有序网络图。加入货物载体约束后，可根据网络中游集散点的需求量进行模型重构建，并比较加入货物载体约束与不加货物载体约束的工作量大小，从而评价它是否对预案产生显著影响。

三、模型假设与约定

1、假设收集的数据真实可靠；

2、没有天生携带抗体的人；

3、总人数 N 不变且仅包含三种健康状态，健康人、病人和移出者的比例分别为 $S(t)$, $I(t)$, $R(t)$ ；

4、不考虑人口的自然出生，自然死亡以及其他类型的人口流动；

5、假设严格隔离的新型冠状病毒感染病人都不再传染他人；

6、假设大部分行政区内服务点数量设置合理。

四、符号说明及名词定义

表 1 符号说明及定义

符号	含义
t	第 t 天
$S(t)$	第 t 天的易感者人数

$I(t)$	第 t 天的感染者人数
β	易感者接触感染者后被感染的概率
γ	感染者恢复的概率
resolution	表示栅格化时二维规则格网单元的大小
pn	染色体个数
max_iter	最大迭代次数
pc	交叉概率
pm	变异概率

五、模型建立与求解

5.1 问题一的模型建立与求解

基于上述分析，根据所提供的长春市 COVID-19 疫情期间病毒感染人数数据以及额外搜集长春市总共的治愈人数、长春市疫情管理相关数据（包括疫情爆发时间、蔬菜包开始发放时间等），并在此基础上从**定性和定量**两个角度对疫情变化与生活物资发放的关系进行分析。

5.1.1 疫情时序数据预处理

1、确定政策变化阶段，查阅资料[1]，确定 3 月 11 日开始静态管理，3 月 26 日开始发放蔬菜包，4 月 15 日后疫情趋于平稳；

2、对阳性病例人数和无症状感染的人数进行合并，得到感染人数总和。

5.1.2 传染病模型建立

为了评价疫情的发展或被控制是否与生活物资发放有关，我们首先对自然状态下疫情的传播进行一个模拟，用来和投放蔬菜包后的情况对比。

SIR 模型是在 1927 年由两位传染病学家 A.G.McKendric 和 W.O.Kenmack 提出的[2]。SIR 模型是最经典的传染病模型之一，研究了某个封闭地区的疫情传播规律，因此本题用来模拟封控区域内无政府干预的情况下不同时刻的未感染人数、感染人数和康复人数。

SIR 模型将人群分为三类，分别是：易感者（Susceptible）、感染者（Infectious）和恢复者（Recovered）。在疫情发生时，感染者的移动与未感染者接触，会导致未感染者成为感染者，同时感染者也会不断康复，最终变为康复者。康复者有了抗体，在这一轮传播中不会再次被感染。SIR 模型就是基于对上述过程的一个数学描述。

设 β 为易感者接触感染者后被感染的概率，γ 为感染者恢复的概率，则得到如下微分方程组：

$$\begin{cases} \dfrac{dS}{dt} = -\alpha S(t)I(t) \\ \dfrac{dI}{dt} = \alpha S(t)I(t) - \beta I(t) \\ \dfrac{dR}{dt} = \beta I(t) \end{cases} \tag{1}$$

γ、β未知，但考虑在一个月内政策变化较小，因此可认为 β 和 γ 是定值，因此可以对 β 和 γ 进行拟合，以便后续对感染人数进行模拟。首先定义损失函数，在损失函数中，我们定义每日的感染者人数的预测值和真实值的均方误差和每日的治愈者人数的预测值和真实值之间的均方误差的和作为总的损失值。然后需要划分训练集和验证集，我们选择利用发放蔬菜包前的数据，即 3 月 11 日-3 月 25 日长春市总共的确诊人数（阳性+无症状），以及查阅资料得到的 3 月 11 日-3 月 25 日长春市总共的治愈人数，作为训练集对模型对 β 和 γ 进行训练，代入训练得到的 β 和 γ 即可得到一个感染人数仿真模拟模型，从而对 3 月 25 日后感染人数在自然条件下的情况进行预测。

5.1.3 相关性评价模型建立

选择 3 月 26 日-4 月 15 日长春市总共的确诊人数和 3 月 28 日-4 月 15 日长春市总共的蔬菜包发放数构建时间序列，为了分析两个序列趋势间的关系，对两个时间序列进行归一化，采用 *min-max* 标准化的方法，对原始数据进行线性变换，将结果映射到[0,1]之间，对序列 $x_1, x_2, \dots, x_n$的转换函数如下：

$$y_i = \frac{x_i - \min\limits_{1\le i\le n}\{x_j\}}{\max\limits_{1\le i\le n}\{x_j\} - \min\limits_{1\le i\le n}\{x_j\}} \tag{2}$$

其中$\max\limits_{1\le i\le n}\{x_j\}$为序列中数据的最大值，$\min\limits_{1\le i\le n}\{x_j\}$为序列中数据的最小值。

得到新序列 $y_1, y_2, \dots, y_n$ 无量纲。

对散点计算Person相关系数，用来定量的衡量发放蔬菜和感染情况的相关关系，计算公式如下：

$$\rho(X,Y) = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sqrt{\sum_{i=1}^{n}(X_i-\mu_X)^2}\sqrt{\sum_{i=1}^{n}(Y_i-\mu_Y)^2}} \tag{3}$$

5.1.4 问题一结果与分析

对于β 、γ 的拟合后得到 $\beta = 0.19$、$\gamma = 0.02$、$R_0 = 9.5$，与 omicron 真实相近[3]，表

明该仿真模型较为正确。基于 SIR 模型对自然状态下的感染人数、实际的感染人数进行可视化得到下图：

图 1 疫情感染人数与仿真感染人数变化图

从**定性**的角度，疫情感染人数的拐点出现在 4 月 2 号左右，接近于根据 SIR 预测得到的感染人数，此后感染人数呈现下降的趋势。4 月 2 号虽然在 3 月 26 日之后，但考虑蔬菜包的投放对疫情的影响存在一定滞后性，所以可以认为蔬菜包的投放对疫情的有控制作用。

实际的感染人数和发放蔬菜包的数量变化构建时间序列，可视化后得到下图：

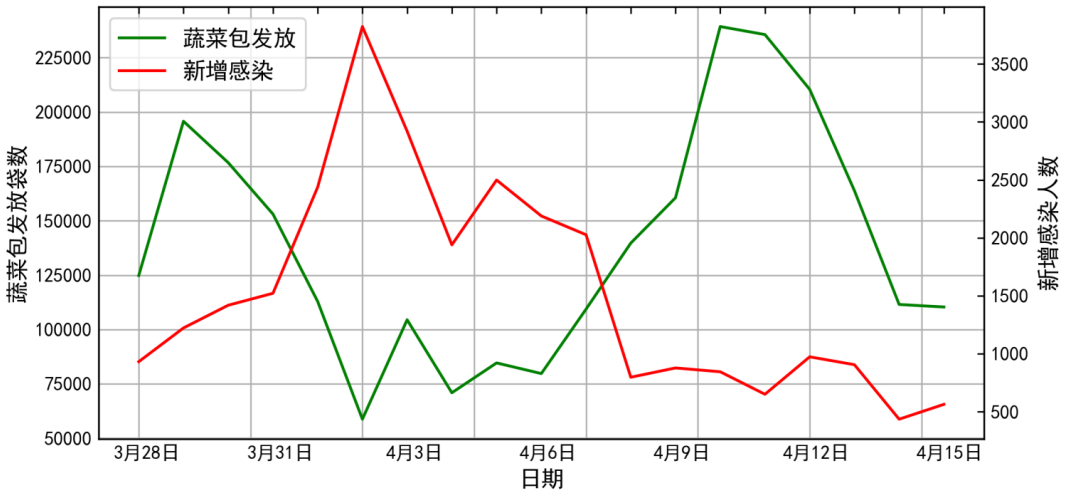


图 2 感染人数和发放蔬菜包时间序列图

从图中可以看出，蔬菜包的数量和新增感染人数的变化方向相反，并且峰值存在一定错位，从另一个角度证明了相关管理政策对疫情影响的滞后性。

接着对疫情的发展与生活物资发放方式的关系进行**定量**的评价。对时间序列的散点可视化，得到下图：

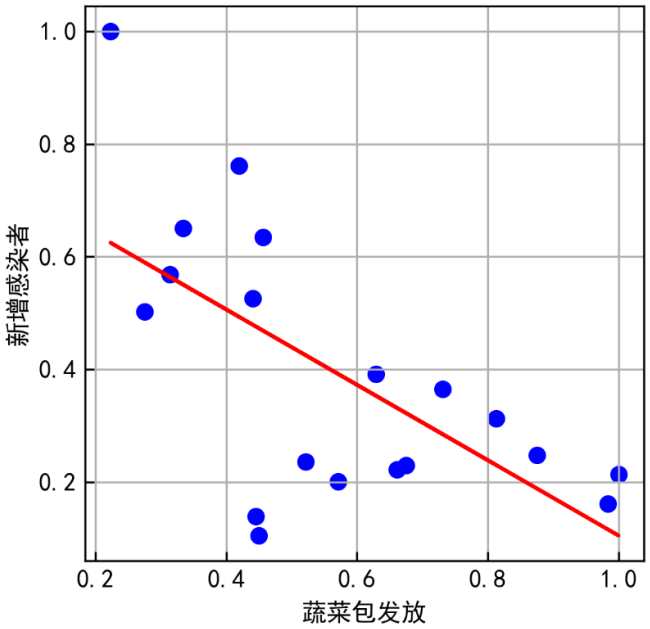


图 3 疫情的发展与生活物资发放散点图

对两个时间序列计算 Person 相关系数，得到相关系数$r = 0.64$，$p - value = 0.003$ <0.5，因此蔬菜包发放和疫情变化呈显著负相关，即蔬菜包的发放可以控制疫情。

对上述现象进行分析，蔬菜包的发放可以控制疫情的原因主要有：

（1）在蔬菜包出现之前，都去购买生活物资而导致大规模人员聚集，从而增加了疫情传播的风险。蔬菜包的出现满足了人们的基本生活需求，因此可以基本实现隔离，阻断疫情的传播；

（2）蔬菜包满足以新鲜蔬菜为主，与其他配送的食材一起，实现了荤素搭配，营养均衡，从而提高了人们的抵抗力，可以更有效地阻挡病毒感染。

5.2 问题二模型的建立与求解

5.2.1 基于孤立森林的投放点数量合理性评价模型

为了评价投放点数量的合理性，我们选取了小区楼栋数量、小区人口、蔬菜包发放数量、隔离人口以及新增病例作为评价指标。假设大部分行政区内服务点数量设置合理，那么评价投放点数量合理性的问题即可转化为对异常值的检测。

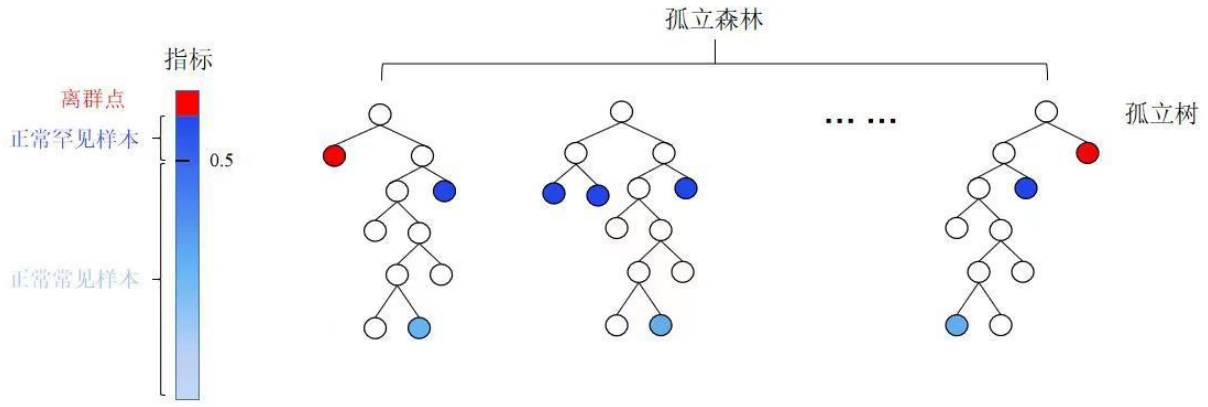


图 4 孤立森林示意图

孤立森林是一种无监督学习算法，属于组合决策树家族，能明确地筛选出异常值。孤立森林的条件有两个：一是异常数据占总样本量的比例很小；二是异常点的特征值与正常点的差异很大。本文使用孤立森林算法来拟合训练数据集中较为聚集的区域从而忽略异常值的影响，相较于传统算法，孤立森林面对高维数据具有很好的鲁棒性，分布稀疏位置的点可以通过较少次数的超平面划分被孤立出来，而分布密集位置的点需要更多次数的超平面划分才能被孤立。

孤立森林根据特征不断画出超平面，这个划分过程是随机的，需要用蒙特卡洛的方法获得一个收敛值，即反复从头划分，然后平均每一次的划分结果。算法大致分为两个阶段，第一阶段我们需要训练出 t 个孤立树，组成鼓励森林，然后将样本点带入每棵孤立树计算平均高度，然后再计算每个样本的异常值分数。第二阶段我们需要利用森林评估测试数据，对于每一个数据点 x_i，令其遍历每一棵孤立树，计算其在森林中的平均高度 $h(x_i)$，对所有点的平均高度做归一化处理，异常值分数的计算公式如下：

$$s(x,\psi) = 2^{-\frac{E(h(x))}{c(\Psi)}} \tag{4}$$

其中，$c(\Psi) = \begin{cases} 2H(\Psi-1) - 2(\Psi-1)/\Psi, \Psi > 2 \\ 1, \quad \Psi = 2 \\ 0, \quad otherwise \end{cases}$，$H(i)$ 是调和数，可以通过欧拉常数来估算，数值越小，表示数据越异常。

基于孤立森林方法能够剔除不合理的区域，则可以认为剩下的行政区投放数量合理，因此我们通过 AdaBoost 分类器回归的方式建立投放数量的预测模型。一般来说，定量输出称为回归，或者是连续变量预测，都称为回归任务。AdaBoost 是 Adaptive Boosting（自适应增强）的缩写。所谓自适应，是指被前一个基本分类器错误分类的样本权值会增大，而正确分类的样本权值会减小，这些权重会用来训练下一个基本分类器。同时将一个新

的弱分类器加入每一轮迭代，直到达到某个预定的足够小的错误率或预先设置的最大迭代次数后再确定最后的强分类器。

其算法步骤如下图所示：

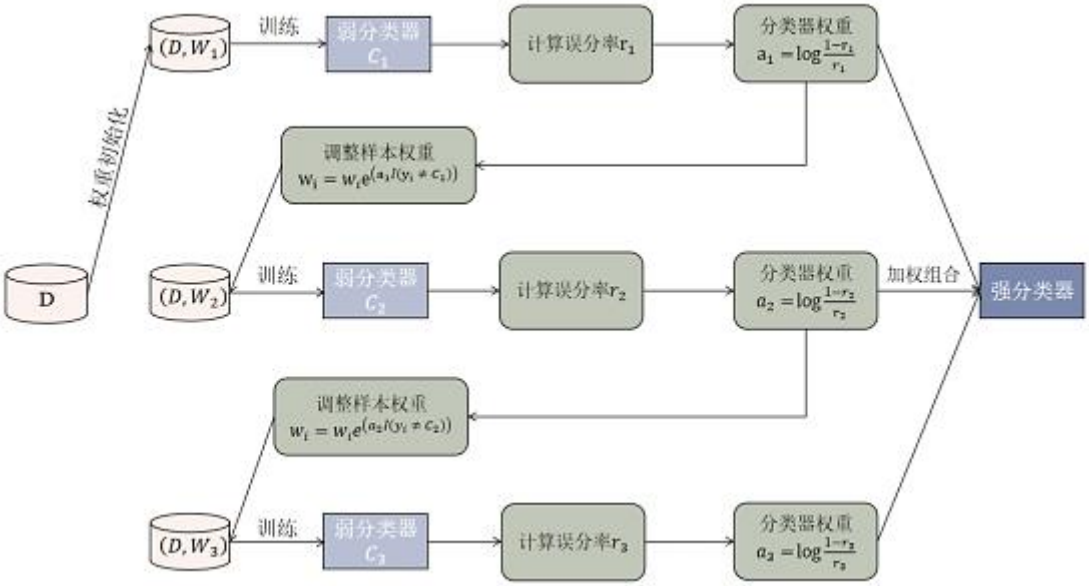


图 5 孤立森林算法步骤

步骤一：初始化数据权值分布D。如图所示，假设有N个训练样本数据，则每一个训练样本最开始时，都会被赋予相同的权值：

$$W_i = \frac{1}{N} \tag{5}$$

步骤二：训练弱分类器C_i。如果某个训练样本点能够被弱分类器C_i准确地分类，那么在构造下一个训练集过程中，该样本点对应的权值要减小；相反，如果某个训练样本点被错误地分类，那么它的权值将增大。更新过的权重样本将被用于训练下一个弱分类器，如此迭代往复。

步骤三：将每次训练得到的弱分类器组合成一个强分类器。各个弱分类器的训练过程结束后，增加分类误差率小的弱分类器的权重，使其在最终的分类函数中起着较大的决定作用。即，误差率低的弱分类器在最终强分类器中占的权重较大，否则较小。

5.2.2 基于最大覆盖模型（MCLP）的选址优化模型

本文从空间覆盖最大化角度，综合考虑多个影响因素进行投放点选址模型的构建。最大覆盖模型是在给定设施数目和最大设施覆盖半径的前提下，尽可能使最多的需求点被覆盖。

投放点选址在区域内有无限的潜在位置可供选择，因此，投放点选址的因素是一个连续选址问题[4]。为了简化计算，本文将研究区域栅格化处理以进行选址研究。首先，可

依据附件三的交通路口节点数据计算研究区域的边界框（Bounding Box），得到总体的坐标范围。接着，在 XOY 平面上划分格网，按照如下公式计算小区点栅格化后的坐标：

$$\begin{cases} i = \text{int}\,(y - y_{min})/\,resolution \\ j = \text{int}\,(x - x_{min})/\,resolution \end{cases} \tag{6}$$

式中，x_{min}, y_{min} 是交通路口节点数据的边界框在 x 轴和 y 轴上的最小值，*resolution* 表示栅格化后二维规则格网单元的大小，i, j 分别表示小区点栅格化后在 XOY 坐标系中的行号和列号，得到可视化结果如下图所示：

图 6 栅格化结果

由于把面要素小区抽象成一个需求量十分集中的点不够合理且难以进行有效分配，为了更好地拟合真实情况，我们对小区点栅格格网进行了平滑操作。该平滑操作是以小区点所在的格网为中心建立九宫格，九宫格内每个格网的人口数为原小区点人口数的平均，从而使得人口数的分布更为自然、合理。

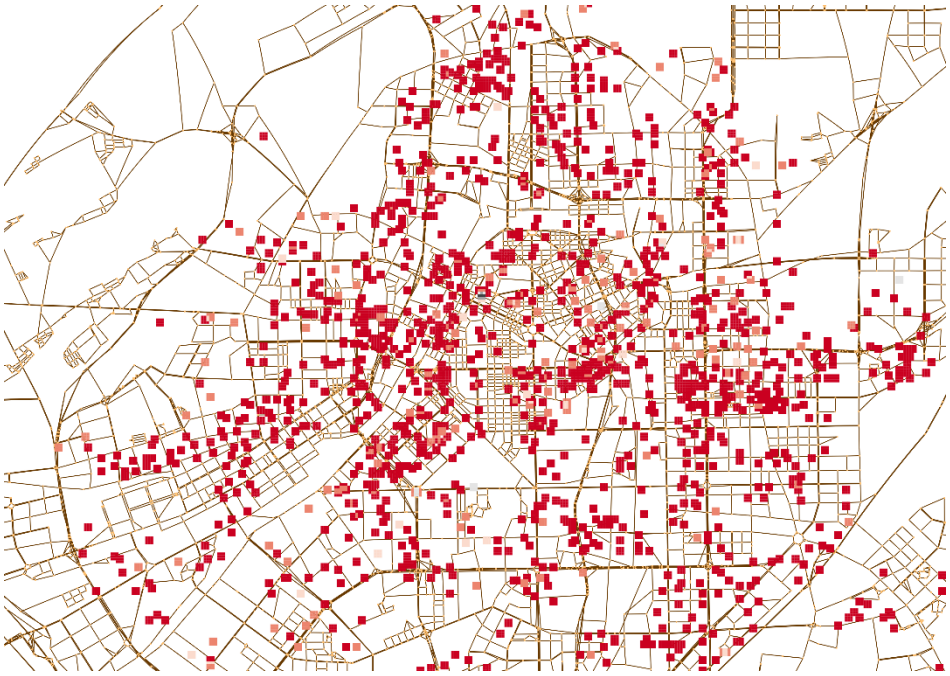


图 7 栅格平滑结果

本文通过栅格化研究区域的操作，将连续选址问题转化为离散型选址问题，使得问题可被描述为，在给定的 $m \times n$ 格网空间上选取 t 个格网作为分拣场所。该类问题通常牵扯到多个优化目标，即政府希望分拣场所建在人口分布密集且位于道路附近的位置点，从而可以服务更多的人口；居民则会考虑到出行的便利性和成本，希望分拣场所建立在自己附近区域；政府规划部门则更多地考虑到 k 个分拣场所的选址方案需要兼顾公平与效率。具体来说，分拣场所选址方案需满足：

1）设施覆盖范围尽可能大；

2）在设施服务人口数一定的情况下，设施的利用率尽可能高。

据此可建立本文的多目标优化模型如下：

$$f_1 = max \frac{\sum_{i=0}^{K} S(i)}{\sum_{i=0}^{M} \sum_{j=0}^{N} D(i,j)} \tag{7}$$

$$f_2 = max \frac{1}{K} \sum_{i=0}^{K} \frac{S(i)}{S} \tag{8}$$

综上，将多目标函数转化为单目标函数：

$$\max \alpha \left(\frac{\sum_{i=0}^{K} S(i)}{\sum_{i=0}^{M} \sum_{j=0}^{N} D(i,j)} \right) + \beta \left(\frac{1}{K} \sum_{i=0}^{K} \frac{S(i)}{S} \right) \tag{9}$$

式中，$D(i,j)$表示第 i 行第 j 列格网的需求量，也即人口数。$S(i)$表示第 i 个选址点已被分配的需求量，也即已服务人口数。S表示每个选址点所能分配的最大需求量，也即最大服务人口数。

5.2.3 基于遗传算法的选址优化模型求解

对于上述所构建的选址优化模型，本质上是一个典型的优化问题，属于 NP 难问题，即在多项式的时间内找不到确定的最优解。因此，针对这种问题目前采用最多的是启发式算法、遗传算法、蚁群算法等一系列启发式算法。为此，本文采用遗传算法进行选址模型的求解。

遗传算法(Genetic Algorithm，简称 GA)，是 1962 年美国的 J. Holland 教授受达尔文进化论的启发提出的一种启发式算法。该算法通过一种全局搜索寻优技术，它根据生物学中遗传与进化的原理，仿照基因、染色体等物质表达方式，从初始随机种群出发，通过模仿自然界基因复制、交叉和变异等操作，产生更适应环境的个体，种群不断得到迭代进化，并逐步逼近最优解。其主要特点是具有内在的隐并行性和更好的全局寻优能力。采用概率化的寻优方法，不需要确定的规则就能自动获取和指导优化的搜索空间，自适应地调整搜索方向。遗传算法采用二进制编码方式及基本的变异、交叉算子算法流程如下：

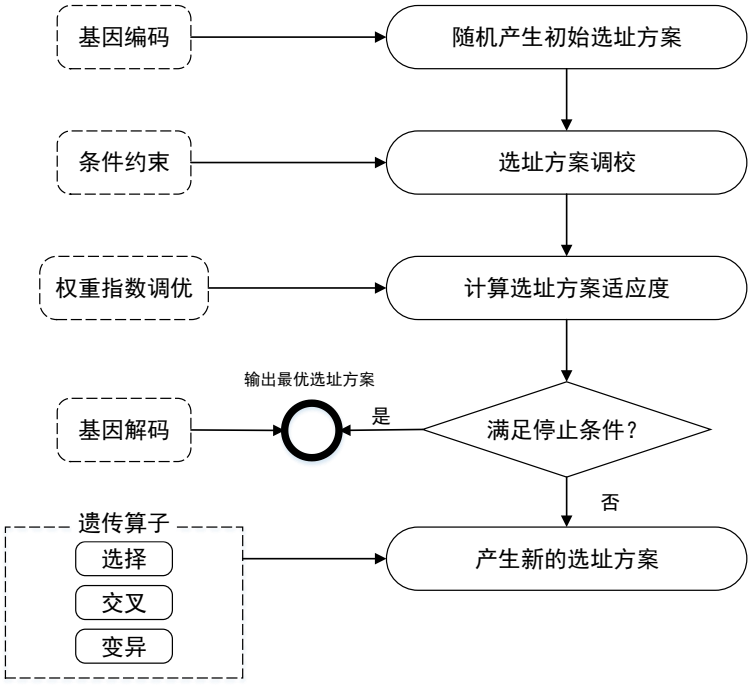


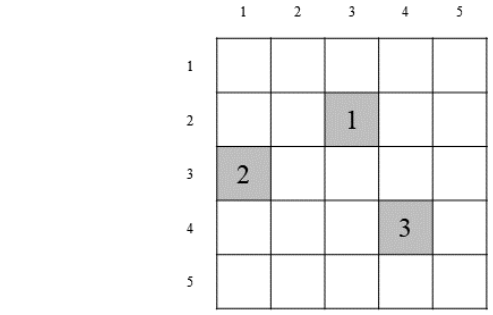
图 8 遗传算法流程图

（1）基因编码

对于传统的遗传算法多采用二进制编码方式，但不适用于大场景下的选址优化问题。因此对于待优化的选址优化问题，我们首先设计一种对投放点进行空间位置编码的编码方式。设将长春市栅格化后形成了 $m \times n$ 的格网，则对于某个栅格单元 G 在该格网上的空间位置可表示为 $G(i, j)$，其中 i 为该栅格单元的行号，j 为列号；$1 \leq i \leq m$，$1 \leq j \leq n$。栅格单元 G 表示一个基因。对于包含 n 位置点的选址问题，染色体 g 可以编码为：

$g(i_1,j_1;i_2,j_2;\cdots;i_n,j_n)$其中一个染色体包含 n 个基因单元。

以 5 个位置点选址问题为例，模型染色体遗传编码如下图所示：



(a)

位置编码	1	2	3	4	5	6
遗传编码	2	3	3	1	4	4
	1		2		3	

(b)

图 9 编码方式示意图：（a）格网空间；（b）染色体编码

（2）参数初始化，主要分为：

① 确定种群规模以及染色体长度，即完成选址任务的染色体个数 *pn* 和染色体长度对应的最多选址个数 *gn*；

②输入原始参数与给定参数，如求解该问题时的最大迭代次数 *max_iter*、交叉概率 *pc*、变异概率 *pm* 等

③ 随机产生一组由初始个体构成的种群组成初始种群，令第一代为 *iter*=1。

（3）适应度函数

对于每一个个体，需要计算其适应度函数；考虑到优化目标是使覆盖最大和利用率最高，因此定义适应度函数如下：

$$f=\alpha\left(\frac{\sum_{i=0}^{K}S(i)}{\sum_{i=0}^{M}\sum_{j=0}^{N}D(i,j)}\right)+\beta\left(\frac{1}{K}\sum_{i=0}^{K}\frac{S(i)}{S}\right) \tag{10}$$

式中，α 和 β 分别为两个目标函数的权重。

（4）选择算子

一般来说，模仿自然界物竞天择的条件下，适应度大的个体有更大的生存机会，而适应度较小的个体继续存在的机会较小，即适应度较高的基因保留的机会更大。因此定义保留前 *retainRate* * *pn* 的个体，对剩下的 (1-*retainRate*)* *pn* 的个体以一定的概率 P_{select} 保留。

（5）交叉算子

遗传算法需要通过交叉操作不断地产生新的个体，本文采用单点重组策略来实现交叉操作。首先，随机选取两个染色体作为父代染色体；然后，在染色体中选取一个随机位置，父代染色体交换该位置之后所有的编码信息，生成新的子代染色体。

将两个父本染色体之间的基因片段进行交换，如下图所示：

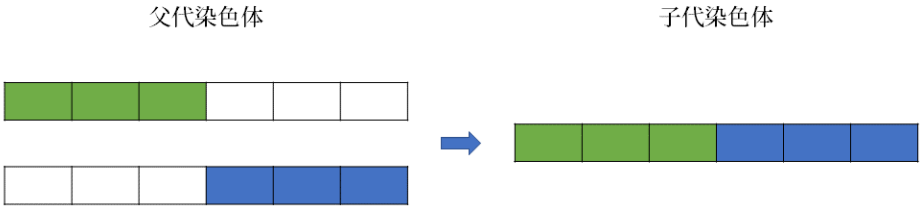


图 10 遗传算法染色体交叉示意图

（6）变异算子

利用遗传算法解决选址优化的规划问题时，保留一定的个体不参与变异操作有利于算法向着全局最优迭代，而不陷入局部最优。因此，本文定义种群中前 1/10 的个体不参与变异，对于剩下的个人以给定概率 P_{mutate} 决定是否进行变异操作。若进行变异操作，则随机选择[gn*0.025]个位置进行变异操作。

（7）终止条件

若满足任一终止条件则迭代停止，输出最优方案。终止条件为：达到最大迭代次数或覆盖率达到 100%且平均利用率达到 95%以上。

5.2.4 投放点数量合理性分析及调整

（1）投放点数量合理性分析

基于孤立森林的离散点检测的结果如表所示，可以看出**二道区**和**汽开区**的评分显著高于其他行政区域，因此被认为是不合理的。结合实际情况，我们可以发现，这两个区的总隔离人口分别为 43 和 22 万人，但其投放点数量则为 9 和 10 个。这意味着，在这两个区中，每个投放点管辖的平均人口在 2 万人以上。这极大地加剧了志愿者和疫情防控人员的工作量，同时也产生了更多的接触风险。因此我们认为二道区和汽开区的点数量设置是不合理的。

表 2 离群点得分

区域	朝阳区	二道区	经开区	净月区	宽城区	绿园区	南关区	汽开区	长春新区
是否为离群点	否	是	否	否	否	否	否	是	否
离群点得分	0.61	4.73	0.55	0.08	0.18	0.08	0.19	2.17	0.17

（2）投放点数量调整

为了回归的正确性，首先需要确定待回归参数之间的相关性。为此，我们计算了小

区楼栋数量、小区人口、蔬菜包发放数量、隔离人口、新增病例以及投放点数量之间的相关性，并制作了热力图如图所示：

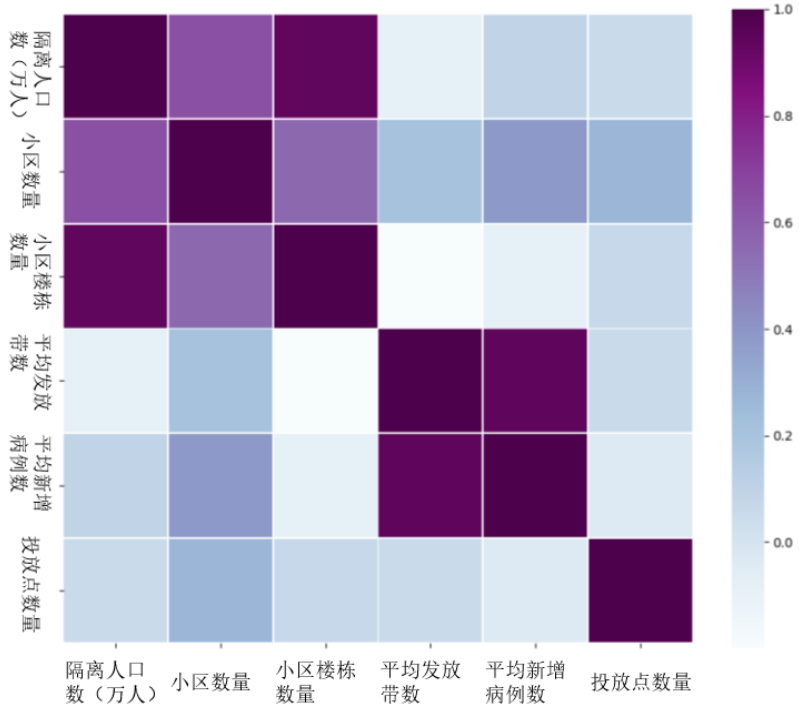


图 11 变量相关性热力图

从热力图中可以看出，隔离人口、小区数量、小区楼栋数量间存在显著相关性，平均发放带数与平均新增病例间存在显著相关性。因此在回归时，我们仅选择隔离人口和平均新增人数两个指标来确定合理的投放点数量。

我们从数据的角度出发，认为非孤立的数据中存在着合理的投放点设置规律。因此，非孤立的数据被认为是合理的，可以指导不合理行政区域设置投放点。处于这种思想，我们使用 AdaBoost 进行回归分析，其中决策树数量设置为 5，回归结果的 MAE 为 0.85。最终，分析得出二道区和汽开区合理设置的点数量分别应为 115 和 170。

5.2.5 选址优化模型结果

基于上述算法，通过 C++ 程序编写得到优化后的政府储备物资和大规模物资分拣场所，包括选址位置及所属区域、选址半径、管辖范围小区个数及管辖范围内人口数等。对选址结果进行可视化，如下图所示：

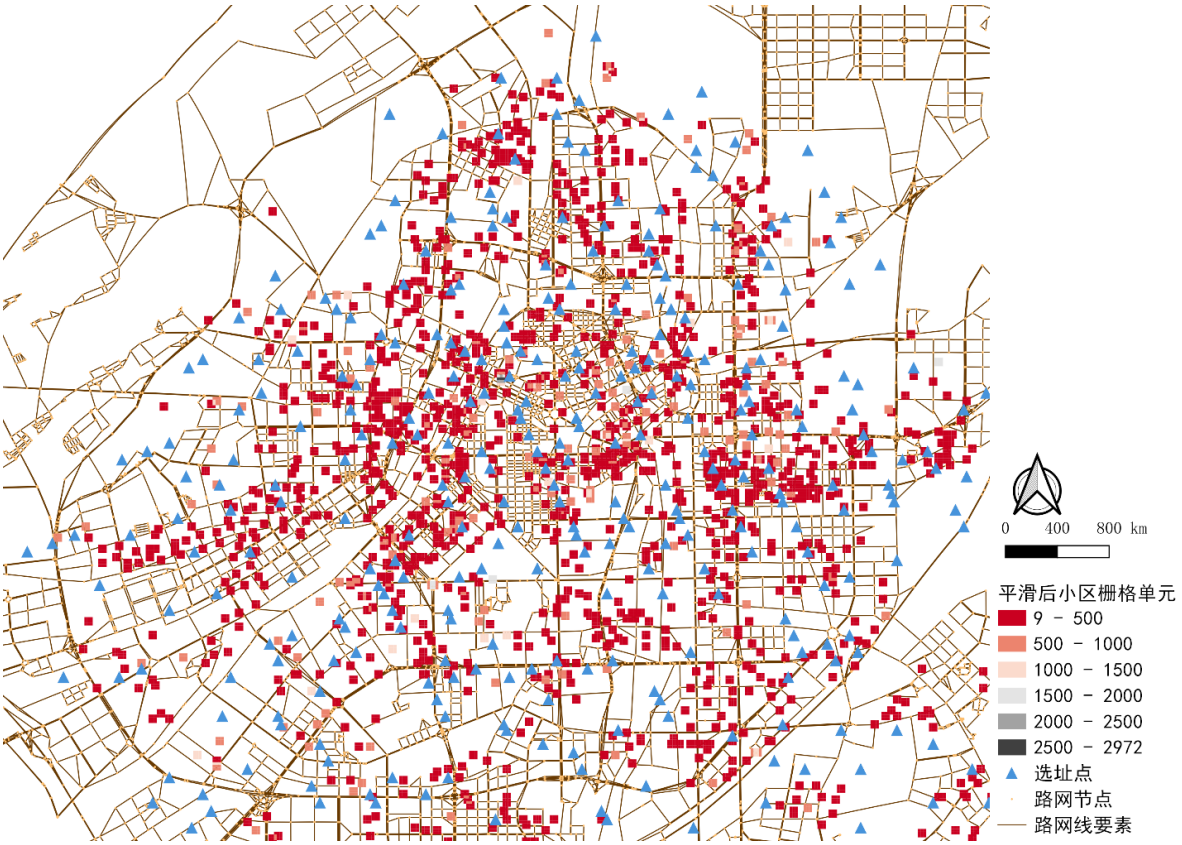


图 12 选址优化结果

从图中可以看出，场所的分布基本和人口密度的分布相符，且人口较少的区域也存在分拣场所，避免了较远距离的运输造成的人员流动。另外，与小区在地图上的分布相比，投放点的分布显然更加均衡。在郊区，具有比小区更大的出现概率。这种往郊区分布的特征，降低了人口密度聚集造成了传播风险，更有利于疫情防控的开展。

与各行政区原投放数量对比，我们给出的数量更加均衡。这在一定程度上缓解了疫情中资源分配不均衡，不充分的问题。并且，我们的选点建立在统筹规划各区的角度上，更有利于全局的调控，增大了各投放点的发放利用率，平均利用率达到 85.85%。

表 3 各行政区规划数量

所属区域	选址点个数
净月区	49
长春新区(高新)	45
南关区	48
汽开区	22
朝阳区	45
经开区	30
绿园区	44

二道区	37
宽城区	38

选址位置及所属区域、选址半径、管辖范围小区个数及管辖范围内人口数等部分结果如下表所示，完整结果见附录。

表 4 部分选址优化结果

选址点编号	横坐标	纵坐标	所属区域	选址半径	管辖小区个数	管辖人口数
5441	73.656	2.36243	净月区	4.93964	2	2311
6666	40.256	2.96243	长春新区(高新)	4.10366	2	9945
6806	68.256	2.96243	净月区	4.41814	1	891
6846	76.256	2.96243	净月区	4.32666	3	1838
7825	86.456	3.36243	净月区	4.68188	7	9999
11433	65.656	4.96243	净月区	4.93964	1	1182
11553	89.656	4.96243	净月区	4.04475	5	7452
11736	33.456	5.16243	长春新区(高新)	3.68782	3	9977
11756	37.456	5.16243	长春新区(高新)	4.93964	3	9486
11776	41.456	5.16243	长春新区(高新)	4.65188	4	9887
13117	31.256	5.76243	长春新区(高新)	4.40454	3	9961
13137	35.256	5.76243	长春新区(高新)	4.5607	3	9759
13337	75.256	5.76243	净月区	4.93964	1	580
15434	30.656	6.76243	长春新区(高新)	4.40454	3	9513
…	…	…	…	…	…	…
19182	37.856	8.36243	长春新区(高新)	3.40588	2	9682

结合可视化效果以及覆盖率等参数可以说明，在给定距离阈值内，该选址结果能够在分拣场所数量尽可能少的条件下满足最大覆盖，同时疫情时期的配送路经的规划节省时间，减少人员流动和人员接触，避免了疫情的扩散。

5.3 问题三模型的建立与求解

5.3.1 蔬菜包发放规律模型的建立与求解

蔬菜包发放规律反映了一段时间内城市人口对物资的需要程度，对未来更好的满足居民的生活水平，加快疫情的清零速度有着重要的作用。在本题中，为了找到蔬菜包的发放规律，探究居民每天的蔬菜所需是不可或缺的。通过查阅资料，成年人每天的蔬菜摄入至少应为 400g。结合第一问的结果，蔬菜包发放与疫情的走势有着紧密的关联，因

此我们首先定型地探索蔬菜包发放量与疫情新增病例数的走势间的关系，使用问题一中的归一化方式绘制趋势图如下图所示：

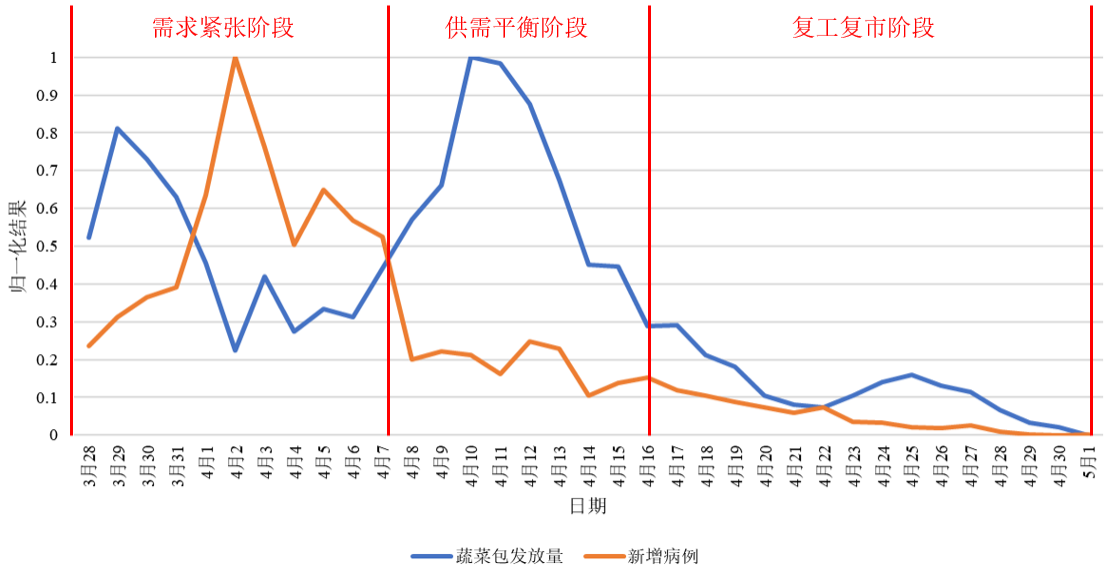


图 13 蔬菜包发放量与疫情新增病例数趋势图

从图中我们可以分析得出，蔬菜包的发放大致经历三个阶段：需求紧张阶段、供需平衡阶段和复工复市阶段：

（1）需求紧张阶段（3 月 28 日 – 4 月 6 日）

在这一阶段中，虽然长春市出台了蔬菜包政策，但是经验的缺失、人手的紧张、疫情的肆虐等因素导致蔬菜包的供给很难满足市民的需求。考虑到运输中蔬菜的损耗（百度一下引用），并且从到蔬菜保质期较短的角度出发，我们认为蔬菜包存在 20%左右的损耗。换言之，平均每日大约要发放给居民 500g 蔬菜才能保证饮食的均衡与健康。我们将每个行政区居民的蔬菜需求量与需求紧张阶段的蔬菜包发放量用柱状图的方式表现出来，如下图所示：

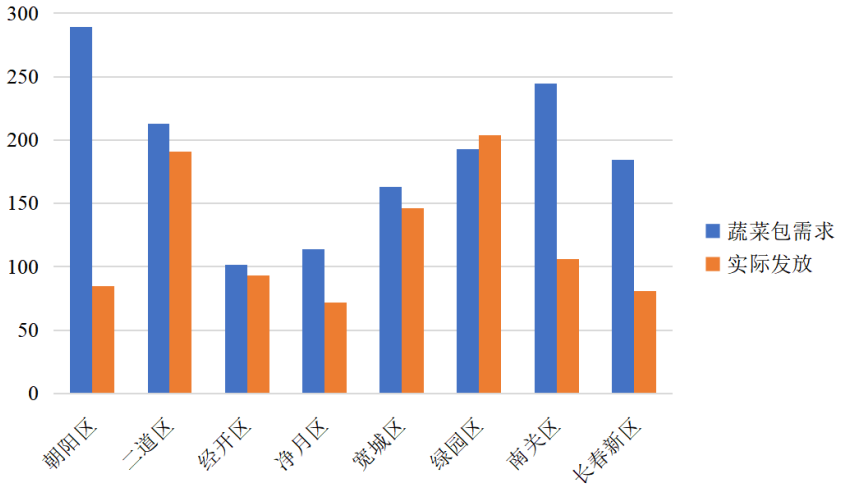


图 14 蔬菜包发放量

从中可以看出，蔬菜包的供应很难满足居民最基本的需求，如何找到合理的蔬菜包供应方案迫在眉睫。需求紧张的局面在 4 月 7 日发生了转变，随着发放策略的完善，供应链的补充，在 4 月 6 日蔬菜供应开始逐步上升并满足居民需求。

（2）供需平衡阶段（4 月 7 日 – 4 月 15 日）

虽然由于蔬菜包的发放，使得居民吃饭问题得到缓解，但是不平衡，不充分的情况仍然存在，如图展示了蔬菜包需求的满足情况，可以看出长春新区、汽开区、宽城区基本能满足居民的蔬菜包需求；绿园区、净月区有一定盈余，经开区盈余较多可能造成浪费；而南关区、二道区、朝阳区蔬菜包不能满足居民最低需求。因此，我们需要设计好的评价手段去评估各个区域的蔬菜包发放数量，找到更合理的分配方案。

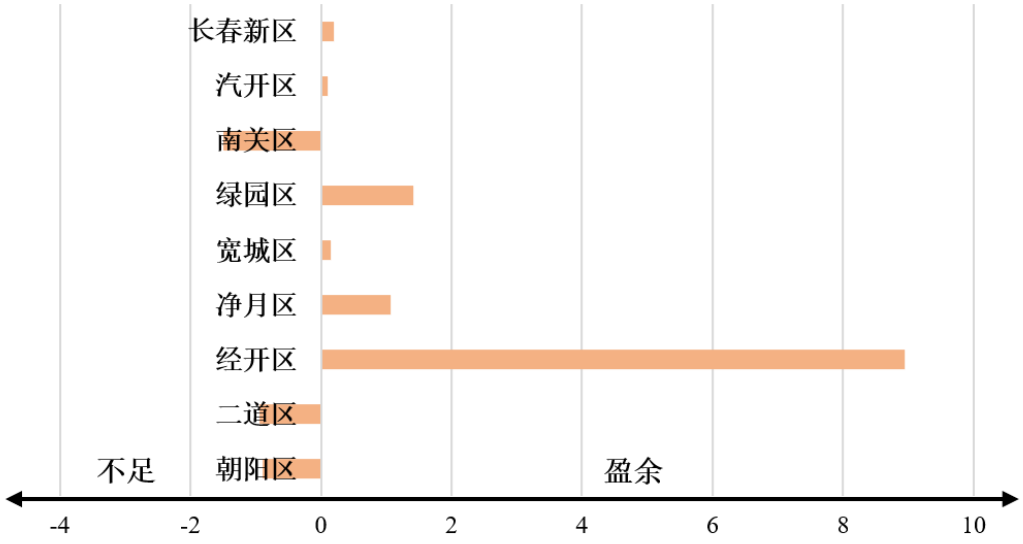


图 15 蔬菜包需求的满足情况

（3）复工复市阶段（4 月 16 日 - ）

根据《4.16 长春市工作日报》，自 4 月 16 日开始，长春市全面复工复市，意味着封控政策稍有放缓，疫情发展态势向好，总体可控。但有少量不确定因素可能会导致疫情二次爆发，因此，低风险区可进行少量人员流动，市场开放，政府不需要强制供给市民蔬菜包。市民可根据自己的需求，自行去商场购买蔬菜、肉类、奶制品等日常必需品，不再依赖蔬菜包的发放。

5.3.2 蔬菜包发放策略评价与调整模型的建立与求解

为了综合的评价蔬菜包发放策略，本文建立量化生活物资发放的三大指标：人数指标、疫情严重程度，工作程度。其中人数指标包括隔离人口数量，小区数量，小区楼栋数量，小区人口数量。人口指标反映了行政区域内人口密度的大小。考虑到在问题二中我们已经确定这三张具有线性相关性，所以这里只选择隔离人口总数来评价蔬菜包的需

求。疫情严重程度具体反应为新增病例数：新增病例越多，疫情就越严重。根据第一问的结果，就需要给这些地区分配更多的蔬菜包。接触频率与工作量反应为生活物资投放点数量。在供给一定的情况下，生活物资投放点数量越多少，每个投放点的工作压力越大，加重了本市志愿者的工作量也增大了人员对蔬菜的接触。

进一步，我们继续深化这三种指标：对于蔬菜包需求的满足情况，我们计算了人均蔬菜包实际获得量与蔬菜包需求量之间的差值，即人均蔬菜包盈亏，作为评价标准：

$$人均蔬菜包盈亏 = \frac{蔬菜包发放量}{区域总人数} - 5 \tag{11}$$

其中 5 是一万人需要的蔬菜重量，即 5t。人均蔬菜包盈亏应该在一个合理区间内，缺的太多会导致居民蔬菜摄入量过少影响健康，盈余太多又会导致一部分蔬菜吃不完或发不完造成浪费的同时加大了防疫人员的工作量。因此我们认为，人均蔬菜包盈亏保持在[-0.5, 1]的区间内最为合理。对于新增病例和投放点数量，都将其与蔬菜包实际获得量相除，去除总量不一致的影响。

然后，按照各项指标的数值特点将指标分为极大型数据，极小型数据和区间型数据，分类结果如下表所示：

表 5 指标分类结果

指标名称	分类
蔬菜包盈亏	区间型指标
新增病例数	极大型指标（效益性指标）
生活物资投放点数量	极小型指标（成本型指标）

5.3.2.1 基于 TOPSIS 的评价模型建立

为了评价蔬菜包发放策略，我们在设定的三种标准之上，建立了 TOPSIS 评价模型，具体流程如下图所示：

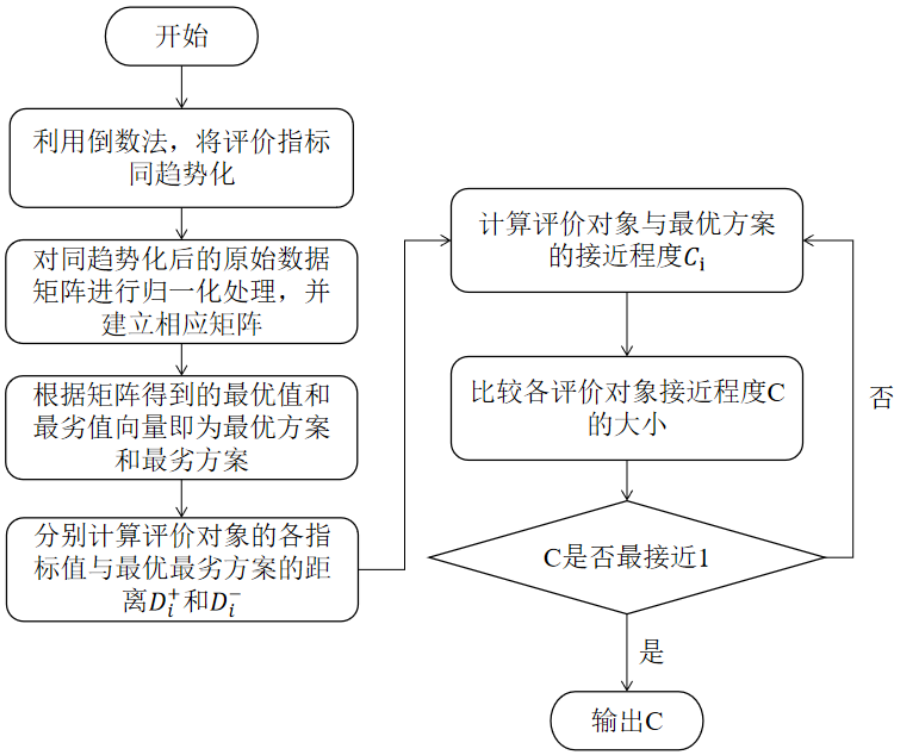


图 16 TOPSIS 算法流程图

步骤一：将各个指标同趋势化处理，处理方法包括：

(i) 利用$max - x$方式将极小型指标转化为极大型指标；

(ii)利用公式(11)，将区间型指标转化为极大型指标；

$$M = max\{a - \min\{x_i\}, \max\{x_i\} - b\}$$

$$\widetilde{x_i} = \begin{cases} 1 - \dfrac{a - x_i}{M}, & x_i < a \\ 1, & a \le x_i \le b \\ 1 - \dfrac{x_i - b}{M}, & x_i > b \end{cases} \tag{12}$$

步骤二：将同趋势化处理后的原始矩阵进行归一化处理，为了消除不同指标纲量的影响，利用公式(12)进行正向矩阵标准化。

$$z_{ij} = x_{ij} \Big/ \sqrt{\sum_{i=1}^{n} x_{ij}^2} \tag{13}$$

步骤三：使用熵权法计算各个指标所占权重，因为熵权法中信息熵的本质是对信息的期望值，如公式(13)所示。所以熵权法可以表示指标反映的信息量的多少，从而客观的给各个指标赋权重。

$$H(X) = \sum_{i=1}^{n} [p(x_i) I(x_i)] = -\sum_{i=1}^{n} [p(x_i)\, ln(p(x_i))] \tag{14}$$

在计算过程中，利用公式(14)计算每个评价指标的信息熵，再利用公式(15)计算信息效用值，最后利用公式(16)将信息效用值归一化，得到每一个指标的熵权。

$$e_j = -\frac{1}{ln\, n}\sum_{i=1}^{n} p_{ij}\, ln\, p_{ij}\ \ (j = 1,2,\dots,m) \tag{15}$$

$$d_j = 1 - e_j \tag{16}$$

$$W_j = {d_j}\Big/{\sum_{j=1}^{m} d_j}\ (j = 1,2,\dots,m) \tag{17}$$

步骤四：利用公式(17)(18)，求出最优矩阵Z^+和最劣矩阵Z^-。

$$Z^+ = (Z_1^+, Z_2^+, \dots, Z_m^+)$$

$$= (\max\{z_{11},\dots,z_{n1}\}, \max\{z_{12},\dots,z_{n2}\},\dots, \max\{z_{1m},\dots,z_{nm}\},) \tag{18}$$

$$Z^- = (Z_1^-, Z_2^-, \dots, Z_m^-)$$

$$= (\min\{z_{11},\dots,z_{n1}\}, \min\{z_{12},\dots,z_{n2}\},\dots, \min\{z_{1m},\dots,z_{nm}\},) \tag{19}$$

步骤五：利用公式(19)(20)计算评价方案的各个指标与最优值和最劣值的距离D^+,D^-。

$$D_i^+ = \sqrt{\sum_{j=1}^{m} \omega_j \left(Z_j^+ - z_{ij}\right)^2} \tag{20}$$

$$D_i^- = \sqrt{\sum_{j=1}^{m} \omega_j \left(Z_j^- - z_{ij}\right)^2} \tag{21}$$

步骤六：利用公式(21)计算评价方案与最优方案的接近程度。

$$C_i = \frac{D_i^-}{D_i^- + D_i^+} \tag{22}$$

5.3.2.2 基于 TOPSIS 模型的蔬菜包发放策略评估与调整

熵权法的权重设定结果如下表所示：

表 6 熵权法的权重设定

评价标准名称	综合得分指数
蔬菜包盈亏	0.6681898
新增病例数	0.22935203
生活物资投放点数量	0.10245817

由表可知，蔬菜包盈亏占权重较大，在 0.66 以上，生活物资投放点数量所占权重较小，约为 0.1。这与事实情况比较吻合，让居民人人有菜吃是我们最大的目标。进一步的，

我们借助 Python 进行 TOPSIS 求解，对各行政单位的蔬菜包发放策略好坏进行评价，分值越高，策略越好，如下表所示：

表 7 各行政区评价结果

行政区名称	综合得分指数	排序
长春新区	0.775788368	1
宽城区	0.683959103	2
汽开区	0.665470882	3
净月区	0.372133458	4
绿园区	0.330429897	5
南关区	0.297350741	6
朝阳区	0.153134382	7
经开区	0.138392391	8
二道区	0.097273476	9

为了更精细地分析影响各个区域评分的因素，并且调整发放政策，我们将 TOPSIS 方法内部不同区域在不同指标下的得分用雷达图进行了展示：

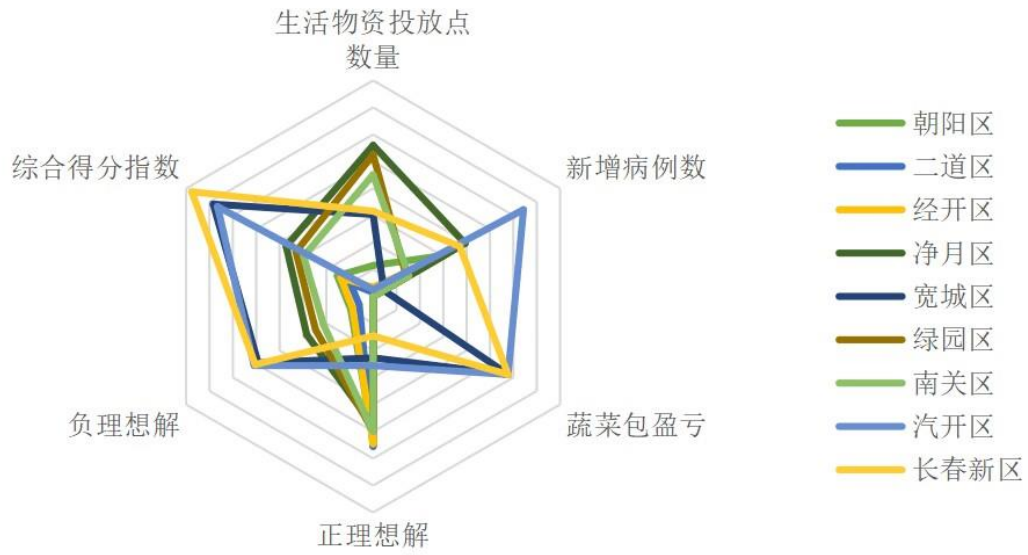


图 17 得分雷达图

通过观察可以从中发现，蔬菜包盈亏是评价中最大的影响因子，长春新区，宽城区，汽开区的评分较高均是因为在比较好的满足了居民日常需求的同时，降低了防疫人员的工作量与接触频率。因此，在今后的物资发放中，如何把握蔬菜包的分发量是重中之重。

为了调整 4 月 10 日到 4 月 15 日的蔬菜包发放量，我们将每天各个行政区的蔬菜包盈亏量可视化处理，如图：

时间	长春新区	净月区	绿园区	朝阳区	经开区	宽城区	南关区	汽开区	二道区
4月7	-1.81	-0.17	-0.36	-2.28	0.25	0.07	-0.02	nan	-0.48
4月8	0.50	0.52	-2.08	-1.05	-0.45	4.51	-0.33	nan	-1.51
4月9	-0.08	-0.13	-1.97	0.36	14.92	0.72	0.15	nan	0.19
4月10	0.83	0.78	2.04	2.45	9.15	7.30	1.00	nan	2.23
4月11	1.71	-0.76	6.32	2.23	26.82	-0.25	0.30	nan	2.17
4月12	4.39	4.09	11.15	-0.14	6.04	-0.43	-1.31	nan	-0.12
4月13	1.35	4.70	3.48	-0.74	4.30	-3.17	-3.44	5.09	-1.72
4月14	0.84	3.18	-1.61	-1.95	11.00	-2.37	-2.97	-1.27	-2.35
4月15	-1.64	1.40	-0.38	-2.57	14.75	-1.04	-2.36	-2.18	-2.67

图 18 蔬菜包盈亏量可视化

图中，绿色代表该区在该时段的需求不足，红色代表该时段的需求过高。通过对图的观察我们进一步调整蔬菜包的发放策略：

1、整个供需平衡阶段，经开区的蔬菜包发放过多。图中可以看到，经开区蔬菜包的发放远大于当地的需求。这不仅造成了资源的过剩导致的浪费，也消耗了有限的防疫资源。因此对于经开区的蔬菜应当统筹管理，多调配给有需要的区，达到供需平衡。

2、4 月 12 日到 4 月 13 日，宽城区、南关区的发放过少。宽城区、南关区在 4 月 13 日都呈现了极大的需求缺口（约 3 吨/万人）。因此，必须在其他行政区的盈余中抽取一部分以弥补亏空。经开区、静月区的蔬菜发放仍有一定盈余，应予以调配。

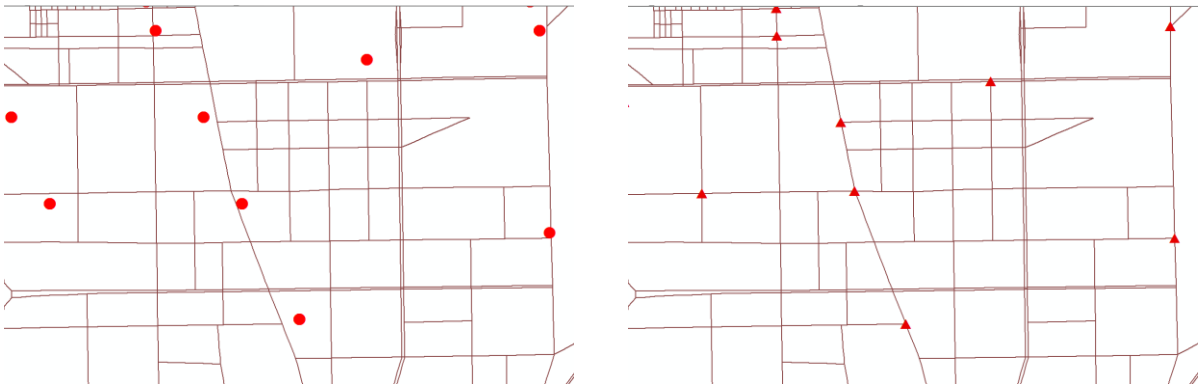
3、4 月 14 日到 4 月 15 日，复工复市的影响已经显现。在此阶段，各个区的蔬菜包的需求已经下降。在 4 月 15 日一天，77.8%的行政区域都没有收到足够的蔬菜。我们认为，在这之后，随着自由市场的恢复，居民可以自行购买生活物资，整体的蔬菜供应量应当减少。

5.4 问题四模型的建立与求解

5.4.1 数据预处理

由于小区和集散地不在路网上的，所以我们需要将这些点匹配到路网上，以便于我们指定特殊时期保障居民生活物资供应的详细预案。对此，我们采用了地图匹配方法将小区和集散地匹配到路网上。考虑小区一般离路网距离较小，因此采用最近邻方法对这些点进行地图匹配，即对每个小区和集散地遍历所有交通路线结点，找到最近邻的交通

结点。



(a) (b)

图 19 地图匹配效果图：（a）匹配前；（b）匹配后

进行地图匹配后我们就找到了距离每个小区最近的交通结点，据此可以计算出两点之间的路网距离。

5.4.2 基于 TSP 模型的有序网络模型建立

网络上游是各项物资来源的位置，具有统筹规划的重要价值。我们选择以每个区的重心构成网络上游。网络中游是各项物资的集散地，对于协调管控小区至关重要。本文采用问题二求解得到的合理选址地作为集散地，即网络中游。网络下游则由长春市的各个小区构成。分析可知，构建有序网络可分为两个子任务，即构建上-中游有序子网络和构建中-下游有序子网络，从而给出整体的有序网络图预案。

针对上-中游有序子网络的构建，通过分析可知，在不考虑货物载体和蔬菜吨数限制的情况下，可将其归结为一个目标优化问题。建立目标优化模型如下：

$$f = min\sum_{i=0}^{n}\sum_{j\neq i,j=0}^{n} w_{ij}x_{ij}$$

$$s.t.\begin{cases} 0 \le x_{ij} \le 1 \\ \sum_{i=0,i\neq j}^{n} x_{ij} = 1 \ \ (i = 0,1,\dots,N) \\ \sum_{j=0,j\neq i}^{n} x_{ij} = 1 \ \ (j = 0,1,\dots,N) \end{cases} \tag{23}$$

其中，

$$x_{ij} = \begin{cases} 1, & \text{存在元素} i,j \text{之间的路径} \\ 0, & \text{不存在元素} i,j \text{之间的路径} \end{cases}$$

式中，w_{ij}是从 i 到 j 的权值或是代价，通常是距离，在本文中则是指运输里程与小区

居民人数乘积得到的指标。上述整数规划模型也可当作旅行商问题(Travelling salesman problem, TSP)的形式化表达，因此，本文采用 TSP 模型求解上述目标优化模型，从而进行上-中游有序子网络的构建，即通过 TSP 算法得出一条最短路径，从而构建出高效、简洁的上-中游有序子网络。

针对中-下游有序子网络的构建，通过分析可知，集散点与小区点之间的空间关系同样可利用 TSP 模型进行建模，从而得到每个集散点与小区点的局部连接关系，继而得到一个整体的中-下游有序子网络。

5.4.3 基于 TSP 模型的有序网络模型求解

在应用 TSP 模型进行有序网络构建之前，首先需要进行带权有向图的生成。对于上-中游有序子网络，每个区的网络上游点与其所辖的网络中游集散点构成一个子图；对于中-下游有序子网络，每个网络中游集散点与其所辖的小区点构成一个子图。由此，有序网络图的构建可被拆解成在一个个子图上使用TSP模型求解得到的最优路径结果。因此，针对一个子图进行求解即可完成有序网络图的构建。

带权有向图的组成元素是节点与带有权值的边。在本文中，权值的计算是运输里程与小区居民人数乘积，其中运输里程指的是路网距离。因此，本文首先对网络上游点、网络中游集散点和网络下游小区点进行了地图匹配，从而找到其对应的最近交通路网节点，再由附件三给出的交通路口节点数据以及交通路口节点拓扑关系和距离构建路网拓扑关系，继而可以计算有序网络图中各点之间的路网距离。生成带权有向图后，即可在该子图上使用 TSP 模型求解最优路径，综合所有子图的 TSP 模型求解结果，即可得到整体的有序网络图。

本文构建的部分有序网络图如下表所示，完整的表格见附录。

表 8 上-中游有序子网络

网络上游点	有序路径	工作量
宽城区	136660->135283->131103->125527->123199->128766->136640->138027->140810->146387->150557->150537->139384->153766->159838->166832->159858->156154->153826->149190->147804->153866->147844->145060->138087->139464->133907->131123->128806->126953->125547->117207->121387->121367->121347->114393->115760	504,698,200
汽开区	80868->76707->59981->61341->59941->59921->80788->82189->75246->71095->73896->80848->83609->86410->87821->92013->89231->86470->96214->100396->99015->100416	375,587,200
...	...	...
经开区	88041->92213->100576->97795->111730->107580->	374,482,624.5

104788->93653->97815->96414->92233->90842->89
451->79708->75506->81068->85240->81048->85220
->88001->78247->74076->69924->65763->67163->6
4382->85300->88081

表 9 中-下游有序子网络

网络中游点	有序路径	工作量
7825	1211->1169->1240->1206->1267->1246	2,603,362.18
15674	1195->1191->1262->1216->1252->1212	1,565,504.48
60181	1199->1156->1235->1176->1163->1133->1239	6,719,955.28
42034	1233->1220->1242->1150->1165->1159->770->857.0	4,736,951.28
...	...	...
62982	933->937->986->1002->938->925->963	4,504,261.49
62902	838->873->864->805->768->851->823->837->738	6,826,955.08
71095	1387->1355->1372->1391->1363->1405->1308	7,148,114.77
32250	901->737->772->745->888->797->788.0	4,159,559.59

在不考虑货物载体和蔬菜吨数限制的情况下，该有序网络图的指标为

表 10 不考虑货物载体和蔬菜吨数限制的工作量

网络名称	工作量
上-中游有序子网络	4,787,079,093.89666
中-下游有序子网络	2,847,569,733.33668
有序网络图	7,634,648,827.23334

若考虑大卡车和小卡车对有序网络图的影响，需分别对其进行建模。由问题二可知，每个集散点的最大服务人数P为 10,000 人，依据附件四可知，每人每日蔬菜消耗量W为 400 克，则集散点的最大服务人数可换算为最大蔬菜总量为$T = P * W = 400 * 10000 = 4{,}000{,}000$ 克 $= 4$ 吨。由于 4 吨均小于小卡车和大卡车的承载能力，因此，小卡车与大卡车对中-下游有序子网络的构建没有影响，仅对上-中游有序子网络的构建有影响。因此，仅讨论小卡车和大卡车对上-中游有序子网络的影响。

第一，讨论小卡车对上-中游有序子网络的影响。小卡车的加入带来了货物载体的约束，此时可将网络中游集散点抽象为小区点，则每个网络中游集散点的需求量均为 4 吨。由于小卡车的承载能力也是 4 吨，因此目标优化模型失效，此时不能再用 TSP 模型进行上-中游有序子网络图的构建。由此，此时最优上-中游有序子网络图的构建方法是在每个子图中，网络上游点派遣与网络中游离散点相同数量的小卡车进行资源运输。

第二，讨论大卡车对上-中游有序子网络的影响。同第一点，将网络中游集散点抽象

为需求量为4吨的小区，但此时大卡车的承载能力是10吨，足以进行小区间的配送任务。因此，本文针对该情况，提出一种两阶段贪婪寻优算法来进行上-中游有序子网络的构建。两阶段贪婪寻优算法的思路为，网络上游点首先派遣大卡车去距离其最近的网络中游集散点，然后该网络中游集散点再派遣大卡车去距离其最近的其他中游集散点。由此，可完成上-中游有序子网络图的构建。

由下表可知，加入小卡车和大卡车相当于加入了一种货物载体约束，会比不考虑卡车的时候工作量大。其中考虑小卡车的工作量是不考虑卡车的 3 倍左右，考虑大卡车是不考虑卡车的 2 倍左右，而考虑小卡车是考虑大卡车的 1.5 倍左右。因此，若需要加入货物载体约束，则最好考虑使用大卡车进行资源运输。

表 11 考虑交通方式的工作量

交通方式	上-中游有序子网络的工作量
不考虑卡车	4,787,079,093.89666
考虑小卡车	13,307,233,892.157204
考虑大卡车	9,048,760,126.036358

六、模型评价与改进

6.1 模型优点

1、本文基于传染病模型的仿真结果对疫情传播与蔬菜包发放的关系进行了分析；

2、本文针对栅格化后小区密度过大，提出了一种栅格平滑的方式，使得计算结果更加全面准确、贴近实际情况；

3、本文提出了一种空间位置的编码方式，使得选址优化模型能够通过遗传算法对模型进行求解，且收敛速度较高;

4、本文从研究模型的内在机理出发到模型建立、求解、优化、验证、拓展，整体框架较为完整，验证了模型较好的正确性，并给出了生活物资方案的合理方案。

6.2 模型缺点

1、构建 SIR 模型对感染人数进行评估时只考虑了自然条件的影响，没有考虑隔离政策等的影响；

2、由于 NP 难问题遍历复杂度较高，而本文所采用的基于遗传算法求解策略只能寻找到该情况下的局部最优解，而不是全局最优。

参考文献

[1] 长春市卫生健康委员会[EB/OL]. [2022/10/10]. http://wjw.changchun.gov.cn/.

[2] Kermack W O, McKendrick A G. Contributions to the mathematical theory of epidemics--I. 1927[J]. Bull Math Biol, 1991,53(1-2):33-55.

[3] Zhang X, Zhang W, Chen S. Shanghai's life-saving efforts against the current omicron wave of the COVID-19 pandemic[J]. Lancet, 2022,399(10340):2011-2012.

[4] 吴钦钦王珂樊文有彭玉玲. 基于连续空间需求的公共图书馆最大覆盖选址方法——以武汉市主城区为例[J]. 地理与地理信息科学, 2020.

附录

一、问题一代码

```
from scipy.integrate import odeint
def loss(parameters,infectious, recovered, y0):
    # 确定训练模型的天数
    size = len(infectious)
    # 设置时间跨度
    t = np.linspace(1,size,size)
    beta, gamma = parameters
    # 计算预测值
    solution = odeint(SIR, y0, t, args=(beta, gamma))
    # 计算每日的感染者人数的预测值和真实值的均方误差
    l1 = np.mean((solution[:,1] - infectious)**2)
    # 计算每日的治愈者人数的预测值和真实值之间的均方误差
    l2 = np.mean((solution[:,2] - recovered)**2)
    # 返回 SIR 模型的损失值
    return l1+l2
#   我们定义函数的名称为 SIR
def SIR(y,t,beta,gamma):
    S,I,R = y
    dSdt = -S*(I/(S+I+R))*beta
    dIdt = beta*S*I/(S+I+R)-gamma*I
    dRdt = gamma*I
    return [dSdt,dIdt,dRdt]

# 设置人群总人数为 N
N = 3019825
# 设置初始时的感染人数 I0 为 239
I0 = 239
# 设置初始时的恢复人数 R0 为 31
R0 = 31
# 所以，初始易感者人群人数 = 总人数 - 初始感染人数 - 初始治愈人数
S0 = N - I0 - R0
# 设置初始值
y0 = [S0, I0, R0]
import pandas as pd
import numpy as np
data = pd.read_csv('data/i&r.csv',encoding = "gbk")

# 确定训练集每天的感染者人数
infectious_train = data['全市感染者'].to_numpy()
```

```
infectious_valid = infectious_train[15:27]
infectious_train = infectious_train[0:15]
print(infectious_train)
# =SIR 模型的恢复者
recovered_train = data['治愈'].values
recovered_train = recovered_train[0:15]
print(recovered_train)
# 确定训练集每天的易感者人数
susceptible_train = N - recovered_train - infectious_train
print(susceptible_train)

# 模型初始值
I0 = 160
R0 = 0
S0 = N - I0 - R0
y0 = [S0,I0,R0]

# 导入 minimize 函数
from scipy.optimize import minimize

# 训练模型
optimal = minimize(loss,[0.0001,0.0001],
                   args=(infectious_train,recovered_train,y0),
                   method='L-BFGS-B',
                   bounds=[(0.00000001, 1), (0.00000001, 1)])

beta,gamma = optimal.x
# 输出 beta、gamma 值
print([beta,gamma])

# 确定初值
I0_valid = 1128

R0_valid = 403
S0_valid = N - I0_valid- R0_valid
y0_valid = [S0_valid, I0_valid, R0_valid]

import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif']=['SimHei']
plt.rcParams['axes.unicode_minus']=False

#%matplotlib inline
# 确定观察的时间周期
T = 12
```

```
# 设置估计疫情的时间跨度为 T 天
t = np.linspace(1,T,T)
# 估计三种人数的数量
solution = odeint(SIR, y0_valid, t, args = (beta, gamma))
# 绘图
fig, ax = plt.subplots(facecolor='w', dpi=100)
# 绘制估计的 I 曲线与真实的 I 曲线
ax.plot(t, infectious_valid, 'r', alpha=0.8, lw=2, label='实际感染人数')
ax.plot(t, solution[:,1], 'b', alpha=0.8, lw=2, label='预测感染人数')
# 绘制估计的 R 曲线与真实的 R 曲线
#ax.plot(t, recovered_valid, 'g-.', alpha=0.5, lw=2, label='recovered_valid')
ax.plot(t, solution[:,2], 'g', alpha=0.8, lw=2, label='预测治愈人数')
# 设置横纵坐标轴
labels = ["", "3 月 27 日","", "3 月 29 日", "", "3 月 31 日", "", "4 月 2 日","", "4 月
4 日", "", "4 月 6 日"]
plt.xticks(t, labels)

ax.set_xlabel('日期')
ax.set_ylabel('感染人数')
# 添加图例
ax.legend()
ax.grid()
plt.savefig("result/sir 拟合图.png", dpi=300)
#plt.box(False)
   plt.show()
```

二、问题二代码及选址结果

```
8
9   void GA::GAInit(int chromosomeLen, int populationNum, int initVariation, int nIterCount) {
10      this->chromosomeLen = chromosomeLen;
11      this->populationNum = populationNum;
12      this->initVariationNum = initVariation;
13      this->nIterCount = nIterCount;
14      this->selectInterval = int(this->candidatePnts.size() / this->chromosomeLen);
15
16      if (this->chromosomeLen != NULL && this->populationNum != NULL && this->initVariationNum != NULL) {
17          generatePopulation();
18      }
19  }
20
21  void GA::generatePopulation() {
22      // 创建一个初始种群，即解空间
23      for (int i = 0; i < this->populationNum; i++) {
24          // 先创建一个全置零的染色体
25          vector<unsigned short> curChromosome;
26          for (int m = 0; m < this->chromosomeLen; m++) {
27              curChromosome.push_back(0);
28          }
29
30          int curVariationNum = 0;//当前已变异个数
31          // 随机在initVariationNum个位置上变异成1
32          while (curVariationNum < this->initVariationNum) {
33              int curVariationPos = rand() % this->chromosomeLen;//当前变异的号码
34              curChromosome[curVariationPos] = 1;
35              curVariationNum++;
36          }
37          this->population.push_back(curChromosome);//将生成的染色体放入种群中
38          vector<unsigned short>().swap(curChromosome);
39      }
40  }
41
```

```
41
42  void GA::run() {
43      for (int i = 0; i < this->nIterCount; i++) {
44          cout << endl << "第 " << i << " 次迭代:" << endl;
45          evolve();
46      }
47  }
48
49  void GA::evolve() {
50      fitness(); //计算个体适应度
51      selection(); //选择
52      crossover(); //交叉
53      mutation(); //变异
54  }
55
56
57  void GA::fitness_thread(int o) {
58      int numCount = float(this->populationNum) / 10; // 和线程数对应
59
60      for (int num = o * numCount; num < (o + 1) * numCount; num++) {
61          float sumDistri = 0; // 总覆盖量和总需求量之比
62          float averDistr = 0; // 每个候选点位置的平均分配比
63
64          vector<unsigned short> curChromosome = this->population[num];
65          vector<vector<float>> candidates;//染色体对应位置的医院
66          vector<bool> isAssignReqPnts; //让每个需求点只能被分配一次
67
68          // 对染色体反解码，拿到当前选择的候选点集
69          for (int i = 0; i < curChromosome.size(); i++) {
70              if (curChromosome[i] == 1) {
71                  candidates.push_back(candidatePnts[i * selectInterval]);
72              }
73          }
74
```

选址点编号	横坐标	纵坐标	所属区域	选址半径	管辖范围小区个数	管辖范围内人口数
5441	73.656	2.362433333	净月区	4.93964	2	2311
6666	40.256	2.962433333	长春新区(高新)	4.10366	2	9945
6806	68.256	2.962433333	净月区	4.41814	1	891

6846	76.256	2.962433333	净月区	4.32666	3	1838
7825	86.456	3.362433333	净月区	4.68188	7	9999
11433	65.656	4.962433333	净月区	4.93964	1	1182
11553	89.656	4.962433333	净月区	4.04475	5	7452
11736	33.456	5.162433333	长春新区(高新)	3.68782	3	9977
11756	37.456	5.162433333	长春新区(高新)	4.93964	3	9486
11776	41.456	5.162433333	长春新区(高新)	4.65188	4	9887
13117	31.256	5.762433333	长春新区(高新)	4.40454	3	9961
13137	35.256	5.762433333	长春新区(高新)	4.5607	3	9759
13337	75.256	5.762433333	净月区	4.93964	1	580
15434	30.656	6.762433333	长春新区(高新)	4.40454	3	9513
15674	78.656	6.762433333	净月区	4.20476	7	4874
15714	86.656	6.762433333	净月区	3.67151	3	2052
16875	40.456	7.362433333	长春新区(高新)	4.91935	3	9199
17035	72.456	7.362433333	净月区	4.40454	2	1980
18929	80.056	8.162433333	净月区	4.9679	3	1780
18949	84.056	8.162433333	净月区	4.90306	5	1589
19182	37.856	8.362433333	长春新区(高新)	3.40588	2	9682
23110	81.056	9.962433333	净月区	4.9679	6	3746
25168	28.656	10.96243333	长春新区(高新)	4.36807	3	9786
25248	44.656	10.96243333	长春新区(高新)	4.94773	2	9962
25368	68.656	10.96243333	净月区	2.20907	3	9440
25448	84.656	10.96243333	净月区	4.49444	5	1000
27042	32.256	11.76243333	长春新区(高新)	2.4	3	9900
27122	48.256	11.76243333	南关区	4.91935	7	9975
27202	64.256	11.76243333	净月区	3.64966	2	6289
27262	76.256	11.76243333	净月区	4.93964	2	3772
28462	37.856	12.36243333	长春新区(高新)	4.20476	4	9918
28482	41.856	12.36243333	长春新区(高新)	4.93964	5	9919
29379	35.656	12.76243333	长春新区(高新)	4.28019	4	9985
29559	71.656	12.76243333	净月区	4.44072	5	9898
31807	57.256	13.76243333	南关区	4.93964	9	9912
31887	73.256	13.76243333	净月区	4.49444	8	9986
31927	81.256	13.76243333	净月区	4.91935	7	5138
31967	89.256	13.76243333	净月区	4.9679	4	1362
32250	53.056	13.96243333	南关区	4.93964	11	9958
32430	89.056	13.96243333	净月区	4.95177	3	333
33831	90.856	14.56243333	净月区	4.93964	3	1698
34911	28.456	15.16243333	长春新区(高新)	4.87032	5	8981
35091	64.456	15.16243333	净月区	4.72017	4	9990
35151	76.456	15.16243333	净月区	4.66905	2	3527
36412	50.256	15.76243333	南关区	4.44072	8	9993
36452	58.256	15.76243333	南关区	4.81664	4	9965
37702	29.856	16.36243333	长春新区(高新)	4.24264	5	9647
37842	57.856	16.36243333	南关区	4.93964	3	3235
39073	25.656	16.96243333	长春新区(高新)	4.81664	1	9450
39133	37.656	16.96243333	长春新区(高新)	4.36807	6	9898
39153	41.656	16.96243333	长春新区(高新)	4.30813	10	9990
39293	69.656	16.96243333	净月区	4.93964	9	9999

39353	81.656	16.96243333	净月区	4.44072	4	4666
39373	85.656	16.96243333	净月区	4.87032	5	4947
40483	29.256	17.56243333	长春新区(高新)	4.23792	3	8293
40543	41.256	17.56243333	长春新区(高新)	4.94773	9	9864
40563	45.256	17.56243333	南关区	3.45254	6	9997
41914	37.056	18.16243333	长春新区(高新)	4.88262	4	9930
42034	61.056	18.16243333	南关区	4.9679	13	9966
42174	89.056	18.16243333	净月区	4.91935	4	1660
43255	26.856	18.76243333	长春新区(高新)	4.86621	3	4059
43335	42.856	18.76243333	南关区	4.9679	8	9870
43355	46.856	18.76243333	南关区	4.75395	9	9944
44675	32.456	19.36243333	长春新区(高新)	4.77074	2	7839
44795	56.456	19.36243333	南关区	4.9679	7	5558
44875	72.456	19.36243333	净月区	4.93964	6	9904
44895	76.456	19.36243333	净月区	4.91935	4	6402
44935	84.456	19.36243333	净月区	4.9679	4	2792
46096	38.256	19.96243333	长春新区(高新)	4.81664	4	9928
46256	70.256	19.96243333	净月区	4.9679	10	9911
47576	55.856	20.56243333	南关区	4.93964	6	4574
47736	87.856	20.56243333	净月区	4.30813	1	234
48877	37.656	21.16243333	长春新区(高新)	4.88262	4	9867
48957	53.656	21.16243333	南关区	4.94773	12	9598
49097	81.656	21.16243333	净月区	4.40454	6	5619
50417	67.256	21.76243333	净月区	4.66476	16	9982
51264	51.056	22.16243333	南关区	4.88262	5	510
51304	59.056	22.16243333	南关区	4.95177	2	9024
51384	75.056	22.16243333	净月区	4.94773	3	3138
51404	79.056	22.16243333	净月区	4.93964	1	416
52111	34.856	22.56243333	长春新区(高新)	3.53836	2	9918
52191	50.856	22.56243333	南关区	4.44072	9	9992
52291	70.856	22.56243333	净月区	4.9679	9	9968
52371	86.856	22.56243333	净月区	4.94773	4	3408
53482	30.656	23.16243333	长春新区(高新)	4.93964	5	8641
53502	34.656	23.16243333	长春新区(高新)	4.61736	5	9684
53562	46.656	23.16243333	长春新区(高新)	4.219	7	9978
53642	62.656	23.16243333	净月区	4.9679	8	9967
56343	46.056	24.36243333	长春新区(高新)	4.83322	10	9977
56383	54.056	24.36243333	南关区	3.44093	6	9987
56423	62.056	24.36243333	净月区	4.93964	6	9828
56543	86.056	24.36243333	净月区	4.91935	2	1138
57693	37.656	24.96243333	长春新区(高新)	4.88262	7	9876
57813	61.656	24.96243333	净月区	4.95177	7	8928
59124	45.456	25.56243333	长春新区(高新)	3.82099	5	9995
59921	19.256	25.96243333	汽开区	4.68188	6	9981
59941	23.256	25.96243333	汽开区	4.83735	6	9945
59981	31.256	25.96243333	汽开区	4.74131	5	9997
60061	47.256	25.96243333	长春新区(高新)	4.9679	6	9968
60081	51.256	25.96243333	朝阳区	4.36807	7	9927
60181	71.256	25.96243333	净月区	4.72017	14	9988

61341	24.856	26.56243333	汽开区	4.68188	4	7245
61461	48.856	26.56243333	长春新区(高新)	3.68782	3	9957
61481	52.856	26.56243333	朝阳区	4.91935	8	9966
61501	56.856	26.56243333	南关区	4.95177	8	9990
62782	34.656	27.16243333	长春新区(高新)	4.87032	4	9982
62802	38.656	27.16243333	长春新区(高新)	4.93964	6	9993
62842	46.656	27.16243333	长春新区(高新)	4.68615	7	9886
62882	54.656	27.16243333	南关区	4.84149	8	9982
62902	58.656	27.16243333	南关区	4.93964	11	9986
62982	74.656	27.16243333	净月区	4.60435	10	9917
64202	40.256	27.76243333	长春新区(高新)	4.93964	5	9878
64322	64.256	27.76243333	南关区	4.80416	7	9989
64362	72.256	27.76243333	净月区	4.9679	7	9985
64382	76.256	27.76243333	经开区	4.90306	4	7497
64462	92.256	27.76243333	净月区	4.23792	1	1088
65563	34.056	28.36243333	长春新区(高新)	4.84149	8	9945
65643	50.056	28.36243333	朝阳区	3.40588	4	9954
65763	74.056	28.36243333	经开区	4.93964	5	7508
66963	35.656	28.96243333	长春新区(高新)	4.9679	5	6672
67143	71.656	28.96243333	净月区	4.91935	7	6974
67163	75.656	28.96243333	经开区	4.94773	5	4161
68374	39.456	29.56243333	长春新区(高新)	4.83735	6	9992
68394	43.456	29.56243333	长春新区(高新)	4.93964	10	9949
69764	39.056	30.16243333	长春新区(高新)	4.49444	10	9990
69804	47.056	30.16243333	朝阳区	3	8	9999
69924	71.056	30.16243333	经开区	4.94773	7	6787
71095	26.856	30.76243333	汽开区	4.87032	10	9959
71175	42.856	30.76243333	朝阳区	4.56946	4	9943
71195	46.856	30.76243333	朝阳区	4.56508	6	9951
73896	30.256	31.96243333	汽开区	4.52548	10	9997
73956	42.256	31.96243333	朝阳区	3.54401	12	9984
74036	58.256	31.96243333	南关区	4.94773	10	9985
74056	62.256	31.96243333	南关区	4.70744	8	9994
74076	66.256	31.96243333	经开区	4.93964	9	9988
75246	21.856	32.56243333	汽开区	4.83735	7	9988
75346	41.856	32.56243333	朝阳区	4.12311	10	9975
75506	73.856	32.56243333	经开区	4.9679	11	6176
76707	35.656	33.16243333	汽开区	4.9679	13	9974
76747	43.656	33.16243333	朝阳区	4.80416	10	9988
76767	47.656	33.16243333	朝阳区	4.30813	7	9993
76907	75.656	33.16243333	经开区	4.94773	2	630
78127	41.256	33.76243333	朝阳区	4.90306	14	9983
78167	49.256	33.76243333	朝阳区	3.94462	8	9966
78187	53.256	33.76243333	朝阳区	4.91935	9	9992
78227	61.256	33.76243333	南关区	3.96989	7	9932
78247	65.256	33.76243333	经开区	4.93964	9	9992
79528	43.056	34.36243333	朝阳区	4.61736	12	9996
79708	79.056	34.36243333	经开区	4.77074	1	726
80788	16.656	34.96243333	汽开区	4.75395	3	9362

80848	28.656	34.96243333	汽开区	4.40454	10	9999
80868	32.656	34.96243333	汽开区	4.90306	7	9994
80988	56.656	34.96243333	南关区	3.68782	9	9985
81048	68.656	34.96243333	经开区	4.66905	16	9994
81068	72.656	34.96243333	经开区	4.93964	13	9973
82189	18.456	35.56243333	汽开区	4.80416	3	5212
82349	50.456	35.56243333	朝阳区	4.90306	8	9981
82389	58.456	35.56243333	南关区	4.80833	7	9992
82409	62.456	35.56243333	南关区	4.9679	7	6512
83589	20.056	36.16243333	绿园区	3.49285	1	378
83609	24.056	36.16243333	汽开区	4.91935	3	5265
83709	44.056	36.16243333	朝阳区	4.219	15	9985
83769	56.056	36.16243333	朝阳区	4.38634	6	9989
83809	64.056	36.16243333	南关区	4.93964	15	9958
85120	47.856	36.76243333	朝阳区	3.2249	8	9993
85140	51.856	36.76243333	朝阳区	4.86621	7	9984
85160	55.856	36.76243333	朝阳区	4.75395	8	9995
85180	59.856	36.76243333	南关区	4.80416	6	9971
85220	67.856	36.76243333	经开区	4.93964	13	9999
85240	71.856	36.76243333	经开区	4.52548	14	9999
85300	83.856	36.76243333	经开区	4.94773	5	6298
86410	27.456	37.36243333	汽开区	4.86621	8	9888
86470	39.456	37.36243333	汽开区	4.80416	8	9998
86570	59.456	37.36243333	南关区	4.91935	10	9986
86590	63.456	37.36243333	南关区	4.9679	14	9915
87821	31.256	37.96243333	汽开区	4.94773	15	9966
87981	63.256	37.96243333	南关区	4.9679	19	9962
88001	67.256	37.96243333	经开区	4.93964	10	9997
88041	75.256	37.96243333	经开区	4.88262	11	9985
88081	83.256	37.96243333	经开区	4.9679	10	9933
89231	34.856	38.56243333	汽开区	4.5607	13	9955
89291	46.856	38.56243333	朝阳区	3.98497	7	9993
89311	50.856	38.56243333	朝阳区	4.81664	9	9992
89331	54.856	38.56243333	朝阳区	4.61736	9	9984
89351	58.856	38.56243333	朝阳区	4.94773	9	9970
89411	70.856	38.56243333	二道区	4.44072	11	9996
89451	78.856	38.56243333	经开区	4.9679	12	8410
90802	70.656	39.16243333	二道区	4.68615	12	9974
90842	78.656	39.16243333	经开区	4.9679	7	3571
91993	30.456	39.76243333	绿园区	4.66476	6	9828
92013	34.456	39.76243333	汽开区	4.93964	12	9935
92073	46.456	39.76243333	朝阳区	4.87032	7	9990
92113	54.456	39.76243333	朝阳区	4.63897	7	9982
92153	62.456	39.76243333	南关区	4.94773	12	9959
92173	66.456	39.76243333	二道区	4.83735	12	9931
92213	74.456	39.76243333	经开区	4.44072	7	9973
92233	78.456	39.76243333	经开区	4.9679	5	3406
93373	28.056	40.36243333	绿园区	2.97321	1	3860
93473	48.056	40.36243333	朝阳区	3.60555	10	9987

93573	68.056	40.36243333	二道区	4.29418	6	9997
93653	84.056	40.36243333	经开区	4.24264	7	9459
94874	49.856	40.96243333	朝阳区	4.60435	13	9986
94894	53.856	40.96243333	朝阳区	4.01995	15	9999
94954	65.856	40.96243333	二道区	4.94773	6	9955
94974	69.856	40.96243333	二道区	4.68188	8	9997
95034	81.856	40.96243333	经开区	4.9679	1	540
96134	23.456	41.56243333	绿园区	4.9679	1	2835
96174	31.456	41.56243333	绿园区	4.95177	6	9755
96214	39.456	41.56243333	汽开区	4.44072	16	9997
96294	55.456	41.56243333	朝阳区	3	12	9992
96354	67.456	41.56243333	二道区	4.80833	4	9966
96414	79.456	41.56243333	经开区	4.93964	6	9861
97535	25.256	42.16243333	绿园区	3.67696	1	810
97655	49.256	42.16243333	朝阳区	4.87032	14	9997
97675	53.256	42.16243333	朝阳区	4.61736	11	9990
97735	65.256	42.16243333	二道区	4.83735	7	9994
97795	77.256	42.16243333	经开区	4.93964	4	7394
97815	81.256	42.16243333	经开区	4.84149	3	6564
98935	26.856	42.76243333	绿园区	4.81664	2	6808
99015	42.856	42.76243333	汽开区	4.90306	17	9997
99035	46.856	42.76243333	朝阳区	4.83735	12	9995
99075	54.856	42.76243333	朝阳区	4.86621	9	9971
99095	58.856	42.76243333	南关区	3.60555	15	9937
100396	40.656	43.36243333	汽开区	4.81664	14	9990
100416	44.656	43.36243333	汽开区	4.20476	9	9995
100456	52.656	43.36243333	朝阳区	4.83735	9	9971
100496	60.656	43.36243333	南关区	3.00666	8	9925
100536	68.656	43.36243333	二道区	3.6	5	9993
100576	76.656	43.36243333	经开区	4.94773	5	7481
101766	36.256	43.96243333	绿园区	4.94773	8	9977
101866	56.256	43.96243333	南关区	4.86621	8	9952
101946	72.256	43.96243333	二道区	4.41814	6	9968
103197	44.056	44.56243333	绿园区	4.40454	18	9999
103237	52.056	44.56243333	朝阳区	4.80416	8	9986
103257	56.056	44.56243333	朝阳区	4.90306	8	9902
103297	64.056	44.56243333	二道区	4.86621	8	9977
104528	31.856	45.16243333	绿园区	4.91935	5	9334
104568	39.856	45.16243333	绿园区	4.41814	16	9973
104628	51.856	45.16243333	朝阳区	4.83735	9	9975
104708	67.856	45.16243333	二道区	3.45254	10	9996
104728	71.856	45.16243333	二道区	4.81664	10	9994
104748	75.856	45.16243333	二道区	4.9679	6	6120
104788	83.856	45.16243333	经开区	4.44072	3	8962
106003	48.456	45.76243333	绿园区	4.60435	13	9984
106023	52.456	45.76243333	朝阳区	4.61736	7	9991
106063	60.456	45.76243333	南关区	4.90306	8	9966
106103	68.456	45.76243333	二道区	2.47386	12	9994
107320	33.456	46.36243333	绿园区	4.94773	11	9931

107380	45.456	46.36243333	绿园区	4.44072	12	9992
107400	49.456	46.36243333	朝阳区	4.60435	9	9985
107440	57.456	46.36243333	南关区	4.04969	7	9997
107460	61.456	46.36243333	南关区	3.92938	6	9977
107580	85.456	46.36243333	经开区	4.32666	1	7740
108746	40.256	46.96243333	绿园区	3.29848	14	9980
108866	64.256	46.96243333	南关区	4.17612	11	9991
108886	68.256	46.96243333	二道区	4.81664	11	9971
108926	76.256	46.96243333	二道区	4.95177	3	2776
110133	39.256	47.56243333	绿园区	4.36807	10	9979
110173	47.256	47.56243333	绿园区	4.17612	12	9988
110193	51.256	47.56243333	朝阳区	4.12311	3	9997
110213	55.256	47.56243333	朝阳区	4.36807	8	9989
110233	59.256	47.56243333	南关区	3.98497	7	9995
110253	63.256	47.56243333	南关区	3.92938	5	9977
110313	75.256	47.56243333	二道区	4.94773	2	9786
111470	28.256	48.16243333	绿园区	3.93954	1	1278
111610	56.256	48.16243333	朝阳区	4.04969	6	9996
111630	60.256	48.16243333	南关区	3.98497	4	9987
111670	68.256	48.16243333	二道区	4.94773	10	9963
111730	80.256	48.16243333	经开区	2.63059	1	2776
112867	29.256	48.76243333	绿园区	4.95177	3	9934
112887	33.256	48.76243333	绿园区	4.83735	8	9930
112927	41.256	48.76243333	绿园区	4.17612	8	9999
112967	49.256	48.76243333	绿园区	4.56946	11	9998
112987	53.256	48.76243333	朝阳区	4.93964	8	9989
113027	61.256	48.76243333	南关区	4.94773	8	9985
113047	65.256	48.76243333	二道区	4.94773	12	9993
113067	69.256	48.76243333	二道区	4.87032	7	9964
114293	36.056	49.36243333	绿园区	3.80526	5	9935
114353	48.056	49.36243333	绿园区	3.77359	13	9988
114373	52.056	49.36243333	朝阳区	4.81664	12	9998
114393	56.056	49.36243333	宽城区	4.94773	9	9988
114413	60.056	49.36243333	南关区	3.98497	9	9973
114433	64.056	49.36243333	南关区	4.47214	9	9997
115720	43.056	49.96243333	绿园区	4.87032	6	9998
115740	47.056	49.96243333	绿园区	4.94773	14	9997
115760	51.056	49.96243333	宽城区	3.79473	10	9994
117107	42.056	50.56243333	绿园区	3.42345	10	9992
117127	46.056	50.56243333	绿园区	3.98497	8	9996
117207	62.056	50.56243333	宽城区	4.49444	8	9987
118543	50.856	51.16243333	绿园区	3.92938	13	9993
118643	70.856	51.16243333	二道区	3.49285	5	9937
119840	31.856	51.76243333	绿园区	4.94773	8	9957
119880	39.856	51.76243333	绿园区	4.83735	12	9955
120000	63.856	51.76243333	南关区	4.75395	7	9993
121247	34.856	52.36243333	绿园区	4.86621	7	9878
121267	38.856	52.36243333	绿园区	4.90306	8	9942
121327	50.856	52.36243333	绿园区	4.86621	13	9938

121347	54.856	52.36243333	宽城区	4.72017	8	9971
121367	58.856	52.36243333	宽城区	4.94773	10	9980
121387	62.856	52.36243333	宽城区	4.94773	10	9988
123119	38.056	53.16243333	绿园区	4.86621	7	8155
123199	54.056	53.16243333	宽城区	4.83735	9	9980
123239	62.056	53.16243333	南关区	3	5	9973
124556	47.056	53.76243333	绿园区	3.94462	12	9986
124676	71.056	53.76243333	二道区	4.49444	7	9976
125467	43.656	54.16243333	绿园区	3.64966	11	9966
125487	47.656	54.16243333	绿园区	4.20476	8	9974
125527	55.656	54.16243333	宽城区	4.70744	9	9997
125547	59.656	54.16243333	宽城区	4.93964	7	9994
125587	67.656	54.16243333	二道区	3.82099	3	9995
125627	75.656	54.16243333	二道区	3.96989	5	9945
126813	34.456	54.76243333	绿园区	4.81664	1	1584
126953	62.456	54.76243333	宽城区	4.83735	7	9977
128766	53.856	55.56243333	宽城区	4.83735	10	9997
128806	61.856	55.56243333	宽城区	4.68188	9	9980
128846	69.856	55.56243333	二道区	3.80526	7	9948
128886	77.856	55.56243333	二道区	4.93964	3	3944
131043	45.256	56.56243333	绿园区	3.2249	10	9967
131103	57.256	56.56243333	宽城区	4.90306	13	9994
131123	61.256	56.56243333	宽城区	4.93964	9	9985
131163	69.256	56.56243333	二道区	4.11825	9	9983
132580	74.256	57.16243333	二道区	4.80416	2	9790
133807	41.256	57.76243333	绿园区	4.91935	7	9848
133907	61.256	57.76243333	宽城区	4.94773	11	9948
135203	42.056	58.36243333	绿园区	4.94773	10	9928
135243	50.056	58.36243333	绿园区	3.67151	5	9957
135283	58.056	58.36243333	宽城区	4.61736	12	9995
135323	66.056	58.36243333	二道区	3.32866	6	9899
135343	70.056	58.36243333	二道区	3.16228	9	9988
136620	47.056	58.96243333	绿园区	4.94773	9	9934
136640	51.056	58.96243333	宽城区	4.93964	12	9994
136660	55.056	58.96243333	宽城区	4.66476	11	9996
136740	71.056	58.96243333	二道区	3.79473	8	9985
136760	75.056	58.96243333	二道区	4.80416	1	1719
138027	50.056	59.56243333	宽城区	4.42719	14	9996
138087	62.056	59.56243333	宽城区	4.95177	8	9956
139384	43.056	60.16243333	宽城区	3.82099	3	2445
139464	59.056	60.16243333	宽城区	4.93964	8	9908
140810	49.856	60.76243333	宽城区	4.93964	15	9971
140930	73.856	60.76243333	二道区	4.81664	3	9719
143674	65.856	61.96243333	二道区	3.68782	6	9770
145060	64.656	62.56243333	宽城区	4.94773	7	9997
146387	51.656	63.16243333	宽城区	4.17612	14	9980
147804	56.656	63.76243333	宽城区	4.94773	9	9894
147844	64.656	63.76243333	宽城区	4.47214	3	9711
147884	72.656	63.76243333	二道区	4.61736	2	9236

149190	55.456	64.36243333	宽城区	4.86621	12	9954
149250	67.456	64.36243333	二道区	3.62215	3	6879
150537	46.456	64.96243333	宽城区	4.01995	7	9957
150557	50.456	64.96243333	宽城区	4.93964	9	9933
153766	42.656	66.36243333	宽城区	4.93964	1	318
153826	54.656	66.36243333	宽城区	4.81664	9	9982
153866	62.656	66.36243333	宽城区	4.88262	4	9987
156154	56.256	67.36243333	宽城区	3.53836	3	9408
157590	65.056	67.96243333	宽城区	4.87032	1	3560
159838	50.656	68.96243333	宽城区	4.38634	6	9616
159858	54.656	68.96243333	宽城区	4.20476	2	9760
166832	57.456	71.96243333	宽城区	3.62215	2	5250

三、问题三代码

```
def dataDirection_1(datas, offset=0):
    def normalization(data):
        return 1 / (data + offset)

    return list(map(normalization, datas))


def dataDirection_2(datas, x_min, x_max):
    def normalization(data):
        if data <= x_min or data >= x_max:
            return 0
        elif data > x_min and data < (x_min + x_max) / 2:
            return 2 * (data - x_min) / (x_max - x_min)
        elif data < x_max and data >= (x_min + x_max) / 2:
            return 2 * (x_max - data) / (x_max - x_min)

    return list(map(normalization, datas))


def dataDirection_3(datas, x_min, x_max, x_minimum, x_maximum):
    def normalization(data):
        if data >= x_min and data <= x_max:
            return 1
        elif data <= x_minimum or data >= x_maximum:
            return 0
        elif data > x_max and data < x_maximum:
            return 1 - (data - x_max) / (x_maximum - x_max)
        elif data < x_min and data > x_minimum:
            return 1 - (x_min - data) / (x_min - x_minimum)
```

```
    return list(map(normalization, datas))

import pandas as pd
import numpy as np

def entropyWeight(data):
    data = np.array(data)
    # 归一化
    P = data / data.sum(axis=0)

    # 计算熵值
    E = np.nansum(-P * np.log(P) / np.log(len(data)), axis=0)

    # 计算权系数
    return (1 - E) / (1 - E).sum()

def topsis(data, weight=None):
    # 归一化
    data = data / np.sqrt((data ** 2).sum())

    # 最优最劣方案
    Z = pd.DataFrame([data.min(), data.max()], index=['负理想解', '正理想解'])

    # 距离
    weight = entropyWeight(data) if weight is None else np.array(weight)
    Result = data.copy()
    Result['正理想解'] = np.sqrt(((data - Z.loc['正理想解']) ** 2 * weight).sum(axis=1))
    Result['负理想解'] = np.sqrt(((data - Z.loc['负理想解']) ** 2 * weight).sum(axis=1))

    # 综合得分指数
    Result['综合得分指数'] = Result['负理想解'] / (Result['负理想解'] + Result['正理想解'])
    Result['排序'] = Result.rank(ascending=False)['综合得分指数']

    return Result, Z, weight

data = pd.read_csv('data/topsis.csv')
# data = pd.DataFrame(
#     {'人均专著': [0.1, 0.2, 0.4, 0.9, 1.2], '生师比': [5, 6, 7, 10, 2], '科研经费': [5000,
6000, 7000, 10000, 400],
#      '逾期毕业率': [4.7, 5.6, 6.7, 2.3, 1.8]}, index=['院校' + i for i in list('ABCDE')])
# print(data)
# exit()
```

```
data['吨/万人_盈余'] = dataDirection_3(data['吨/万人_盈余'], 0, 0.5, 0, 1)   # 师生比数据为区间型指标
data['吨/点'] = 1 / data['吨/点']

out,Z, weight = topsis(data)    # 设置权系数
#out.to_csv('result/topsis.csv')
    print(weight)
```

四、问题四代码及预测结果

```
import numpy as np
import pandas as pd
import networkx as nx
from math import sqrt
import csv


def print_path(path, i, j, res_path):
    if j != 0:
        res_path.append(i)
        next_city = path[i][j]
        print_path(path, next_city, (j - (1 << (next_city - 1))), res_path)
    else:
        res_path.append(i)


def solve_TSP(cost, centre_path_dict, t_ids):  # cost 为权重矩阵
    node_num = len(cost)
    cnt = 2 ** (node_num - 1)
    dp = [[float('inf') for col in range(cnt)] for row in range(node_num + 1)]
    path = [[0 for col in range(cnt)] for row in range(node_num + 1)]  # 记录路径
    for i in range(1, node_num):
        dp[i][0] = cost[i][0]

    for i in range(1, cnt - 1):  # 列标(集合)+
        for j in range(1, node_num):  # 行标
           if i & (1 << (j - 1)) == 0:  # j 不在集合中, (1 << (j-1)) <==> 2^(j-1)
                next_city = j
                for k in range(1, node_num):
                    if (1 << (k - 1)) & i:  # k 在集合中
                      tmp = dp[j][i]
                        dp[j][i] = min(dp[j][i], cost[j][k] + dp[k][i - (1 << (k - 1))])
                        if dp[j][i] != tmp:
                            next_city = k
```

```
                path[j][i] = next_city

    next_city = 0
    for k in range(1, node_num):
        if (1 << (k - 1)) & (cnt - 1):  # k 在集合中
           tmp = dp[0][cnt - 1]
            dp[0][cnt - 1] = min(dp[0][cnt - 1], cost[0][k] + dp[k][cnt - 1 - (1 << (k - 1))])
            if tmp != dp[0][cnt - 1]:
                next_city = k
    path[0][(1 << (node_num - 1)) - 1] = next_city

    res_path_idx = []
    print_path(path, 0, (1 << (node_num - 1)) - 1, res_path_idx)
    print("最短路径: ", dp[0][cnt - 1])

    t_ids = np.array(t_ids)
    res_path = t_ids[res_path_idx]
    centre_path_dict[res_path[0]] = np.insert(res_path, 0, dp[0][cnt - 1])

    return dp[0][cnt - 1]


def main():
    iter_pnts_path = r'C:\Users\dell\Desktop\Problem_F\交通路网拓扑关系.csv'
    # 用拓扑文件来创建带权图，用来计算两点之间的路网距离
    G = nx.read_weighted_edgelist(iter_pnts_path, delimiter=',')  # edges with weighted

    centre_cand_pairs_path = r'C:\Users\dell\Desktop\Problem_F\Problem_4\centre_cand_1.csv'
    centre_cand_pairs = pd.read_csv(centre_cand_pairs_path).values

    # 分别拿到选址点信息(对应的路网节点, 本身坐标), 上游中心点信息, 以及各上游中心点所覆盖的选址点 id
    # 此时 centre_cand_dict 中每一项都是一个子任务, 一个子图
    centre_info, cand_info, centre_cand_dict = {}, {}, {}
    for centre_cand_pair in centre_cand_pairs:
        cand_id, cand_x, cand_y, cand_vertex = centre_cand_pair[0], centre_cand_pair[1],
centre_cand_pair[2], \
                                               centre_cand_pair[-1]
        cand_popu = 10000  # 抽象成服务一万人
       if cand_id not in cand_info:
            cand_info[cand_id] = [cand_x, cand_y, cand_vertex + 1, cand_popu]  # +1 是因为此处路
网节点与真实路网节点编号差 1

        centre_id, centre_x, centre_y, centre_vertex = centre_cand_pair[7],
centre_cand_pair[8], centre_cand_pair[9], \
```

```
                                                    centre_cand_pair[10]
        if centre_id not in centre_info:
            centre_info[centre_id] = [centre_x, centre_y, centre_vertex + 1]

        if centre_id not in centre_cand_dict:
            centre_cand_dict[centre_id] = []
        centre_cand_dict[centre_id].append(cand_id)

    centre_path_dict = {}
    weighted_sum_index = 0
    # 对每个子图进行权重矩阵建立，然后进行 TSP 求解
    for centre_id, cand_ids in centre_cand_dict.items():
        t_ids = [centre_id]  # 为了与 node 保持一致，方便获取到其他属性信息，如人口信息
        nodes = [centre_info[centre_id][-1]]  # 让中心点是第一个节点
        for cand_id in cand_ids:
            nodes.append(cand_info[cand_id][-2])
            t_ids.append(cand_id)

        node_num = len(cand_ids) + 1
        cost = np.zeros((node_num, node_num))  # 初始化当前子图的权重矩阵
        for i in range(node_num):
            for j in range(1, node_num):  # 即其他选址点到中心点的距离也给无效掉，这里用 0 还是-1 或者是无穷有待考量
                if i == j:  # 此时是自己跟自己，即无效，这里用 0 还是-1 或者是无穷有待考量
                    continue
                min_dist = nx.dijkstra_path_length(G, source=str(nodes[i]),
target=str(nodes[j]))
                if min_dist == 0:
                    if i != 0:  # 即此时求距离的两个节点中不含中心点
                        f_x, f_y, s_x, s_y = cand_info[t_ids[i]][0], cand_info[t_ids[i]][1],
cand_info[t_ids[j]][0], \
                                             cand_info[t_ids[j]][1]
                    elif i == 0:  # 若是求中心点到其他选址点的距离
                        f_x, f_y = centre_info[t_ids[i]][0], centre_info[t_ids[i]][1]
                        s_x, s_y = cand_info[t_ids[j]][0], cand_info[t_ids[j]][1]
                    min_dist = sqrt((f_x - s_x) ** 2 + (f_y - s_y) ** 2)
                popu = cand_info[t_ids[j]][-1]
                cost[i, j] = min_dist * popu  # 工作量 = 运输里程 * 小区居民人数
        weighted_sum_index += solve_TSP(cost, centre_path_dict, t_ids)

    with open('centre_path_pair.csv', 'w', encoding='utf8', newline='') as f:
        writer = csv.writer(f)
        for _, item in centre_path_dict.items():
            writer.writerow(item)
```

```
    print("总指标为: ", weighted_sum_index)


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

预测结果：

工作量	有序路径
22269395.36	11736->1116->1056
2603362.182	7825->1211->1211->1169->1240->1206->1267->1246
4770743.24	5441->1226->1234
1508801.54	6846->1177->1255
2223086.28	11553->1171->1225->1171->1225->1143
16426249.73	6666->1065
1565504.458	15674->1195->1191->1262->1216->1252->1212
7025928.675	11756->1096->1042->1067
1429976.08	6806->1250
3543223.7	13117->1050->1054
9940371.28	13137->1119
509312.86	15714->1160->1167
11510416.9	15434->1052->1069
17606068.48	16875->1079
3030365.52	11776->1108
14887736.92	19182->1039
3814496.825	25368->1241->1254
3560372.47	13337->1237
11949375.35	25248->1041
1817117.64	17035->1218->1182
10331001.1	27202->1155
3308030.55	25168->1045
1135406.14	27042->1111->1061
2163460.14	18929->1245->1265->1188
11816219.2	28462->1040->1083
1234250.85	18949->1179->1268
7789736.5	27122->822->907->809->821->775->754
3311055.364	23110->1174->1145->1236->1187->1208->1229
4034488.91	31807->755->735->757->723
445142.6058	25448->1256->1227->1231->1205
8008420.086	29379->1099->1060->1071
6699595.095	28482->1091->733->752->744
4159559.59	32250->901->737->772->745->888->797->788
4146032.47	27262->1257->1183
8677029.67	34911->1105->1112->1059
5224143.62	39133->1095->1084->1072->1087->1101
6898461.457	36452->874->919->725->722

396442.41	31967->1249->1261->1222
16295362.75	37702->1080->1092
9447902.75	36412->747->724
5290485.667	29559->1189->1194->1149->1131
4942845.232	43355->862->847->886->769->848->834->761->883
5037117.69	40563->792->730->767->729
3402619.84	31887->1157->1204->1154
808564.7389	31927->1259->1202->1139->1266->1266
2828274.53	39153->863->779->868
2279328.414	35151->1223->1125
7813885.82	39293->1135->1198->1168->1130->1147
26173750.17	35091->1127->1124->1209
1402262.308	39353->1213->1210->1184->1152
8536132.52	40543->1055->1081->850->858
6742977.81	39373->1219->1173->1238->1123
1194025.14	32430->1162
928710.8314	33831->1247->1230->1192
17300032.52	43335->1063->1094->1078->918->910
4736951.284	42034->1233->1220->1242->1150->1165->1159->770->857
661407.24	42174->1263->1224
23684226.72	39073->1046
9036289.38	44875->1232->1232->1186->1186
8192142.772	46256->1200->1181->1161->1148->1128->1128->1140->1197->1136
8260849.77	41914->1102->1044->1093
4207387.6	40483->1053
2490961.8	44795->801->832->827
7006185.64	50417->1201->1185->1228->1251->1126->1144->1129
5561619.96	47576->882->923->856->753->739
14420392.68	44675->1051
3016328.6	44935->1138->1180
7810230.88	46096->1120->1122
7979275.075	44895->1253->1244->1260->1264
7896662.06	43255->1285->1288->1318
1453303.488	49097->1193->1172->1178->1258->1243
7652612.32	48877->1113->1075
6834866.08	52291->1221->1146->1207->1137->1196->1141->1142
789806.2	47736->1215
23810577.03	52111->1043
11913826.28	52191->1118->498
9073071.3	59921->1304->1283->1275->1298
5566876.1	59941->1293->1290->1278->1277->1348
3796539.36	53642->1190->1158->1134->1153->1170
3048986.16	52371->1164->1217->1175
2067445.88	48957->825->787->741->800->532->532->800
10971265.47	53482->1062->1329->1388->1296
3424902.406	57693->1109->1066->1117->1089

6719955.281	60181->1199->1156->1235->1176->1163->1133->1239
6024390.03	51384->1248->1132
3785428.01	53502->1085->1070->1047->1073
6825514.04	56343->1106->1103->1082->1100->1076->1115->1086
9588620.865	53562->1048->1098->1074->1104
2495757.11	56383->808->782->764->762->894
16750118.7	51304->732
830307.1	51404->1214
22677071.58	59981->1064->1280->1270
4598487.455	60081->826->390->376->469
4504261.487	62982->933->937->986->1002->938->925->963
4139400	56543->1151
10719774.27	64362->1203->942->998->935->945
9668036.3	61341->1302->1274
16438645.91	59124->393->1121->1038
5441092.803	57813->912->778->810->913
3982660.363	56423->742->776->876->785->727->726
11183841.63	64322->929->972->996->969
16382674.24	62782->1049->1269
15544030.03	60061->433->1097->341
2015133.653	61501->866->811->852->750->731
14117956.38	61461->422
2871946.02	61481->386->392
5809363.56	64462->1166
4974638.881	62802->1114->1110->1058->1077->1090
8831115.22	62842->1088->1107->527
17627039.17	65643->490->456->348
6826955.077	62902->838->873->864->805->768->851->823->837->738
16559201.7	65563->1317->1299->1297->353
4440494.289	64202->478->370->381
4210502.51	74056->804->871->1020->957->994->982->977
4894050.18	62882->426->428->356->895->859
7374127.31	64382->1000->930
7701404.53	74036->415->796->457->534->435->526
7761772.193	68374->460->384->1057->509->1068
11487949.5	67143->932->980
5123847.401	69924->1005->968->940->1035->999->1015
8009561.42	69804->377->483
7148114.774	71095->1387->1355->1372->1391->1363->1405->1308
6791492.42	68394->411->358->410
1906007.07	66963->1394->1370->1314
4176572.17	69764->452->416->374->346->344
9598528.59	71175->355->355->465
3755369.86	71195->453
1818773.383	65763->989->931->959->949
2807633.274	67163->1032->1003->974

5019106.81	73956->383->481->473->462
12309818.57	76767->420->342->480
3214168.13	74076->961->958->943->927->946
6129233.58	73896->1347->1401->1364->1375->1379->1313->1331
10820481.9	78187->371->839->786->467->398->375
4385213.83	75346->454->343->494->380->354->396
7335174.067	75246->1380->1295->1333->1291->1289->1353
5740237.26	76747->397->345->340->444->401->496
1785102.337	78227->904->880->814->911
2255311.147	75506->979->1006->995->1030->987->1018->988->1026
3157456.38	78127->529->338->364
4921705.67	80848->1327->1359->1345->1337
3000352.04	78167->506->367->484->514
7401048.7	82189->575->1336
3659005.5	76707->1399->1382->1392->1408->1357->1361->1378->1376
16887958.32	78247->849->763->1014->952->997->1036
6695512.95	79528->528->402->414->423->347->368->449
8942411.64	80788->595->1301
4004189.04	82349->515->537->351->351->470
7426378	83609->1343->1407->570
3736302.41	80868->1325->1309->1335->1346
3025802.134	83709->406->1340->466->450->475
4666241.484	80988->793->424->524->361->379->382->436->522->369
4174860.902	81048->1009->1010->1017->1008->944->971->1027->955->329->1007
5006056.855	85120->417->387->510->429->474->349
6402673.88	82389->870->914->824
5596456.95	86410->1398->695->1342->1356->1402
16672883.26	86470->378->1294->1384->1279->1279->1281->1368
9757878.16	85140->360->339
6842123.87	89291->508->439->408->505->451
3370692.376	87821->1403->1374->1396->1367->1358->1324->1328->1371->1316->1315
3439321.23	96214->1365->1395->1377->1369->1389->1326->1400->1311->1341->1320->1383
2694666.77	89311->438->531->489->359
3662020.375	81068->320->306->308->290->288->328->277->303->1029->298->1019
2620843.8	79708->1031
5974968.58	92073->352->1322->1349
10431477.16	83769->421->412->440
1650872.647	82409->878->831->830->846
3573097.74	85220->324->327->287->282->187->208->264
3606410.271	89231->1334->1344->1362->1332->1366->1303
3049342.28	93473->492->362->350->1393->407->1373
4542906.4	89331->504->517->507->447->400
4187277.767	83809->252->271->284->267->236->178->322->253->272->184
900002.1612	85160->535->409
8504610.008	85180->771->431
1937810.682	85240->302->289->319->1024->984

5625189.15	86570->855->899->518->372
1963766.885	92113->485->413->519
6122481.92	88001->202->300->220->331->191->296
4505142.96	88041->301->291->964->950->249->234
5429814.32	92013->556->548->1406->1352->1292->1319->1404->1323
8262247.72	94874->513->499->501->501->491
5749661.12	89351->437
6647295.98	87981->890->780->840->891->232->283->197->202
6812369.18	94894->520->503->394->905->905->872->902->903
3255299.111	92153->795->813->909->881->488->807->756->820
5645995.737	86590->845->818->297->255->259->313->314->254->263->836
5794557.569	85300->975->954->948->976->1016
2763054.26	91993->691
7384096.62	89411->199->212->193
1876506.68	97655->1386->1282->1360->1351->533->639->442->497->463->500
7369394.81	90802->279->201->965->325->198->216
3637448.523	89451->1034->991->1004->992->1023->967->1021
2693538.024	96294->479->472->516->477->898
6823525.14	88081->978->939->924->934->951
3891610.522	99015->1354->1300->1330->1321->1310->1271->1272->1409->521->476->493
4187126.86	97675->404->365->536->511->893
6028828.273	92173->226->225->897->260->250->269->210
7008197.84	99035->1307->512->458->459
6469134.36	92213->256->990->189->172
3741172.537	99075->403->908->789->867->885->783
5746168.3	97735->802->749->773
4797054.08	93373->546
989304.68	99095->920->892
37550762.03	94954->173->192->865->854
3649648.06	94974->315
5824318.49	92233->1028->985->928
5706492.84	93653->966->981->973->993->962->1013
1205741.856	100496->760->906->486->896
2689823.58	90842->1022->983->1012->1033->956
4959466.59	96354->183
6676646.02	100456->419->525->604->502->495->835
5781937	101866->896->816->799
19598973.07	100536->182->174
3697426.285	100396->625->1273->1339->1287->1312->1350->1338->1276->1390->573
3079510.07	100416->1306->552->596->468->432->1381
6623574.85	101946->170->175
11005398.94	96174->554->660->718
11848624.22	103297->169->889->317
7629008.44	97795->926->218
2207830.459	103257->917->900->921->759
3500495.19	103197->684->580->629->1385->1284->1397->1286

5717825.85	103237->461->441->579->395
4396119.92	106063->869->746->877
6609682.2	104708->181->266->180
3051006.811	104568->571->655->650->585->597->558->697->576->638->1305
2244184.13	106003->645->674->690->549->609->670
6181697.748	104628->425->705->445->557->590
4302782.78	104728->176->217->205->179->171
2521430.366	100576->275->333->334->196
3787449.66	101766->551->681
8647438.6	107440->781->751
9633150.88	107460->875->743->860->748->812
3510965.483	96414->970->960->941->947
1339668.683	106103->228->337->261->203->188->224
11807359.2	96134->654
6408353.61	106023->464->607->363
4251933.8	104748->286->305->330
14830363.8	98935->630
9044001.574	104528->547->538->539->584->584
3636629.829	108746->541->592->641->651->653->700->628->703
3359290.168	107380->659->613->582->540
4555748.61	108886->207->312->214->190
7343617.13	97815->936->1011
5110320.22	107320->621->569->610->562->564->643
6293532.9	107400->391->605->662->699->543
4393394.723	108866->246->295->285->828->806->819
11191131.96	110193->482
3675164.77	110213->887->861
6245481.773	110133->701->568->636->704->566->599->611
3224663.562	110173->677->591->707->434->530->388
5696527.77	110233->879
22973727.2	110253->736->758->323
1921590.11	111670->307->204->194
6145047.4	111610->430->385
11177124.98	111630->843->798
2748296.01	112967->647->373->471
11985143.15	112987->389->790
6193258.152	112927->669->578->612
6156704.87	113027->740->853->774->523->443
4260074.25	113047->258->213->262->240->215
5661539.28	104788->953->1025
3755686.35	114353->622->574->448->357
4563069.61	114373->686->698->446->427
2818704.27	115740->581->706->405->487
9147585.39	114393->366->784->455
8584102.58	114413->922->734->884->829->78
4320730.883	115720->719->644->561->680->603->633

7346968	114293->687->720->542
3907844.67	114433->817->765->815
4070111.29	108926->318->233
15368103.94	117207->841->794
2030706.604	113067->274->242
10355665.18	110313->177->200
28562211.3	107580->1001
4789122.41	115760->399->418->634->606
4044338.63	117107->586->635->565->631->663
2659614.92	112887->615->545->563->555->658
3376140.14	121347->113->36->147->844->915->48->117
7611283.92	120000->161->60->31->138->195
5384277.22	117127->642->616->594->708->577->671
3120426.39	118543->710->593->121->154->418->42->59
10497026.95	121367->842->148->155->130
5517178.219	124556->712->664->550->696->600->619->721->685->626->688
1536727.425	121387->791->137->766->916
9226975.21	119840->588->661
3852368.7	121327->714->135->123->84->118->646->668
11738375.7	118643->186
3379721.1	111730->1037
14794379.03	112867->665->657->553
4391667.14	124676->332->332->304->229
5496964.068	119880->693->623->694->618->715->709->559
21017220.4	123239->158->46->728
843578.25	111470->683
14412661.71	121247->717->587
16740790.99	125587->211->211->206
5726319.59	123199->131->128->95->114->108->144
5579792.819	125547->146->833->26->777->803
10458348.84	128846->270->237->309->244
5778818.93	121267->620->673->572->614->682->632
5519012.83	125627->292->321->247->310
8621331.62	125527->76->3->124->49->68
3571719.818	125467->544->601->675->692->624
2072703.561	131163->294->273->326->336->238
2617913.36	123119->627
1950406.485	125487->617->583->711->637
4514758.66	131043->679->672->702->678->649->648->689
5879388.19	126953->72->223
2327357.005	128766->102->39->166->676->602
2408208.844	128806->94->163->86->69->107->62->133->82->105
5197428.444	131103->91->25->140->61->132->29->65
5082690.21	135323->299->281->231->276
18088854.16	133807->667->589->608
16488998.36	135243->560->656

14769507.66	132580->209->221
9259692.94	135203->55->22->145->713->713->666
10053807.7	131123->97->20->156->116->139
3504515.41	135343->239->235->251
10317109.34	128886->245
10224436.06	136620->104->640->110->598->567
6444795.48	136640->652->79->12->96->92->92
2537990.404	136740->222->265->293->257->185
5686501.845	133907->134->74->101->127->103->115->7->17
4676350.934	135283->66->120->40->143->58->106->45->88
12257489.17	140930->316->268->311
4429331.85	126813->716
5014008.214	138087->168->51->21->33->1
5386078.76	143674->241->335->227->230
16233031.18	136660->93->160->32->109->149->164->2
1716364.369	139384->81->87->159
15166552.31	139464->30->80->5->15
25739234.32	145060->219->243->41
33456281.3	147884->278->248
5530046.755	138027->151->141->38->119->27
5192280.404	146387->98->112->122->56->63->129->37->85
3704855.641	140810->162->90->100->47->125->71->111->167->142
6592624.411	147804->23->24->28->10->152->150
17328653.48	147844->75->280
4378969.307	150537->6->67->77->43
10832877.11	149190->34->153->126->57->73->70->44
3949228	149250->50
13392382.22	153866->35->16
8430865.408	153826->9->19->13->89->4
2430940.291	150557->52->64->136->53
2849291.1	153766->165
12204344.92	159838->54->8->157
26315968	159858->18
8591971.705	156154->11->11->99
18951.66	166832->83->14