



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 国防科技大学，华东师范大学

参赛队号 22910020079

1. 冯云浩

队员姓名 2. 陈树华

3. 侯尹健

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 疫情下封闭式管理的保供物流网络优化

摘 要：

进入 2022 年以来，较大规模的疫情爆发事件仍在全国范围内陆续出现，统一采用的封闭式管理模式尽管能够较快实现疫情的快速清零，但仍面临着一些困难与典型问题。物资调度能力、物资储备能力、交通运输能力作为大规模突发传染疫情应急管理的重要影响因素，其指向的一项重要问题——生活物资的科学管理问题，更是关乎着隔离期居民的日常生活和疫情的防控效果。本文便聚焦于生活物资科学管理主题，研究生活物资投放点数量与选址、物资发放方案、生活物资供应详细预案等多个问题，总结为以下四个问题并提出解决方案。

问题一：生活物资流动方式对疫情影响的评估问题。对于问题一，本文建立了 **SEIR 传染病模型** 来拟合输入参数，由该模型计算长春市的每日暴露数，即每日平均感染能力，将发放蔬菜包前后的每日暴露数进行对比，以衡量疫情防控效果；同时为进一步确定与验证研究结论，本文上海市一时间段的疫情数据进行对比，结果认为蔬菜包发放模式对疫情封控效果并无明显影响，长春市日暴露人数的明显下降主要归因于“封城”的封闭式管理模式。

问题二：生活物资投放点数量与位置问题。对于问题二，子问题是度量投放点数量合理性，本文建立了**基于投放点数量合理性的评估模型（EQUI）**，考虑投放点数量、交通条件、人口条件、面积、连通占比等多类指标建立了投放点数量公平指数。首先，通过对 PR 值、节点所辖小区人口数量、靠近小区数等指标进行聚类，分行政区域计算路口节点在聚类结果中表现较好的占比，将占比与行政区平均度进行合并得出一个区域的交通便利程度指标。其次，综合以上考虑计算得出本文对于投放点数量合理性的公平性度量指标。在此基础上本文对各个区的投放点数量进行优化，优化后总计的投放点数量为 1604 个，相比优化前 1556 个增加 48 个投放点，差距较小。在原有投放点基础上，本文对二道区、朝阳区、汽开区、宽城区分别增加 381、164、14、87 个投放点，对经开区、南关区、净月区、绿园区分别缩减 1、74、163、360 个投放点，优化后的投放点数量在满足各区居民正常生活的情况下节省了人力资源，减少人员流动与接触。

子问题是大规模物流分拣中心的选址问题，本文基于配送费用最少构建了一个**多层级配送中心连续选址模型（MINC）**。先假定运费单价和周转上限等条件，再通过系统聚类划分出了分拣场所的数量上限，之后通过启发式算法与混合整数规划相结合的方式，以最小费用为目标函数，设定迭代条件，经过多轮迭代后算法快速收敛。每日保供费用由初始解的 70 多万下降到了 20 万，最后一共规划了 39 个物资分拣和储存场所。

问题三：生活物资的科学发放问题。对于问题三，本文依据所提供的蔬菜包数据，提取了各个行政区蔬菜包采购袋数、供给重量、发放袋数、发放重量的数据，分析蔬菜包需求与发放规律；同时，通过计算每日各行政区蔬菜包的供给需求差与发放需求差，依据此调整每日的蔬菜包供应方案并以表列示，得出结论认为整体发放过程呈现出供需失衡的状

态，表中需求差为正则表明供过于求，需减少对应数量的蔬菜包，需求差为负则表明供不应求，需增加对应数量的蔬菜包。

问题四：疫情封控下的生活物资发放预案。对于问题四，本文基于**混合整数线性规划（MILP）**的构建了多层次物流网络选址模型。首先将问题所涉及到的节点全部加入公路网中，通过假设和计算构建了一张公路网网络图并计算公路网中节点之间的最短距离。在公路网构建的基础上，进行 MILP 规划模型的构建，求出模型结果，计算得出了 71 个二级节点和 9 个一级节点。同时顺应题意要求，综合考量货车运输方式，重新计算出了二级节点和一级节点。在数量维持不变的前提下，二级节点只有很少一部分发生变化，一级节点大体不变，充分证明了本文所构建的模型的鲁棒性。同时，本方案基于成本最少进行优化，在同类方案里有着最低的成本。

关键词：混合整数规划；启发式算法；SEIR 模型；聚类分析；多层级配送中心连续选址优化；网络选址优化；Gurobi 优化器

目录

1 引言	2
1.1 问题背景	2
1.2 问题重述	2
1.3 论文研究框架	2
2 模型假设	3
3 符号说明	4
4 问题一：生活物资流动方式对疫情影响的评估问题	5
4.1 问题背景	5
4.2 基于疫情影响的 SEIR 传染病模型（SEIR）	5
4.2.1 模型说明	5
4.2.2 模型假设	5
4.2.3 模型建立	6
5 问题二：生活物资投放点数量与位置问题	8
5.1 问题背景	8
5.2 基于投放点数量合理性的评估模型（EQUI）	8
5.3 基于配送费用最少的选址模型（MINC）	12
5.3.1 模型说明	12
5.3.2 模型建立	13
5.3.3 模型求解	15
6 问题三：生活物资的科学发放问题	17
6.1 蔬菜包需求与发放规律	17
6.2 4月10-15日蔬菜包供应方案评价与调整	19
7 问题四：疫情封控下的生活物资发放预案	22
7.1 问题背景	22
7.2 基于混合整数线性规划的多层次物流网络选址模型（MILP）	22
7.2.1 模型说明	22
7.2.2 模型假设与变量	23
7.2.3 模型建立与求解	23
7.2.4 模型改进（考虑货车运输）	25
8 模型总结与评价	27
8.1 模型建立优点	27
8.2 模型建立缺点	27
参考文献	27

1 引言

1.1 问题背景

进入 2022 年以来，较大规模的疫情爆发事件仍在全国范围内陆续出现，统一采用的封闭式管理模式尽管能够较快实现疫情的快速清零，但仍面临着一些困难与典型问题，也暴露出有关部门在应急管理决策过程中的不合理之处。金卫健等（2021）^[1]在其研究中提出 10 项大规模突发传染疫情应急管理的影响因素，其中包括防疫物资调度能力、防疫物资储备能力、应急决策指挥能力和交通运输能力等，这些影响因素也指向了疫情居家隔离期间需解决的一项重要问题——生活物资的科学管理问题，它关乎着隔离期居民的日常生活和疫情的防控效果。

相关数据分析表明，在疫情期间政府完全有能力调配充足的生活物资，但在现实情况中往往难以实现，由此，针对生活物资制定科学的管理方案是一个值得研究的问题。学术界现有关于疫情相关问题的应景性研究在逐渐涌现，有不少研究是关于应急管理主题，但聚焦于生活物资科学管理的研究却较少；针对具体的如物资分拣中心规划选址问题，更多的是从一般意义上的物流中心进行探讨，针对性不强，现实中存在的诸多问题也说明该问题尚未被研究透彻。因而，本文的研究内容可以对现有文献作一定的补充，并拓展应用场景。

作为一项复杂的系统工程，生活物资的科学管理涉及政府部门、街道社区、交通运输部门、仓储中心、分拣和配送中心等多个主体，需要解决生活物资采购、配送、发放等多个问题。本文便聚焦于生活物资科学管理主题，研究生活物资投放点数量与选址、物资发放方案、生活物资供应详细预案等多个问题，总结为以下四个问题。

1.2 问题重述

问题一：生活物资流动方式对疫情影响的评估问题。该问题聚焦于长春市实行蔬菜包发放模式前后防疫效果的评估，说明疫情期间采取的蔬菜包发放形式对疫情防控产是否产生积极作用。

问题二：生活物资投放点数量与位置问题。该问题可分为以下两个子问题，一是根据长春市不同区域投放点数量分布情况，对其数量的合理性进行评估，并通过数学建模进行适当优化；二是在应急管理主题下，考虑大规模物资分拣场所的数量、规模和位置，给出详实具体的选址信息。

问题三：生活物资的科学发放问题。依据长春市蔬菜包发放数据分析其需求与发放规律，并结合各小区位置与人口信息，对 4 月 10 日至 4 月 15 日的蔬菜包供应方案进行评价与优化。

问题四：疫情封控下的生活物资发放预案。在问题二、三的基础上，作出封控期间生活物资供应的有序网络图，分别考虑不加入货车和加入货车运送物资两种情况，并对比前后预案。

1.3 论文研究框架

为解决上述问题，本文将进行以下四部分的工作，具体如图 1 所示。

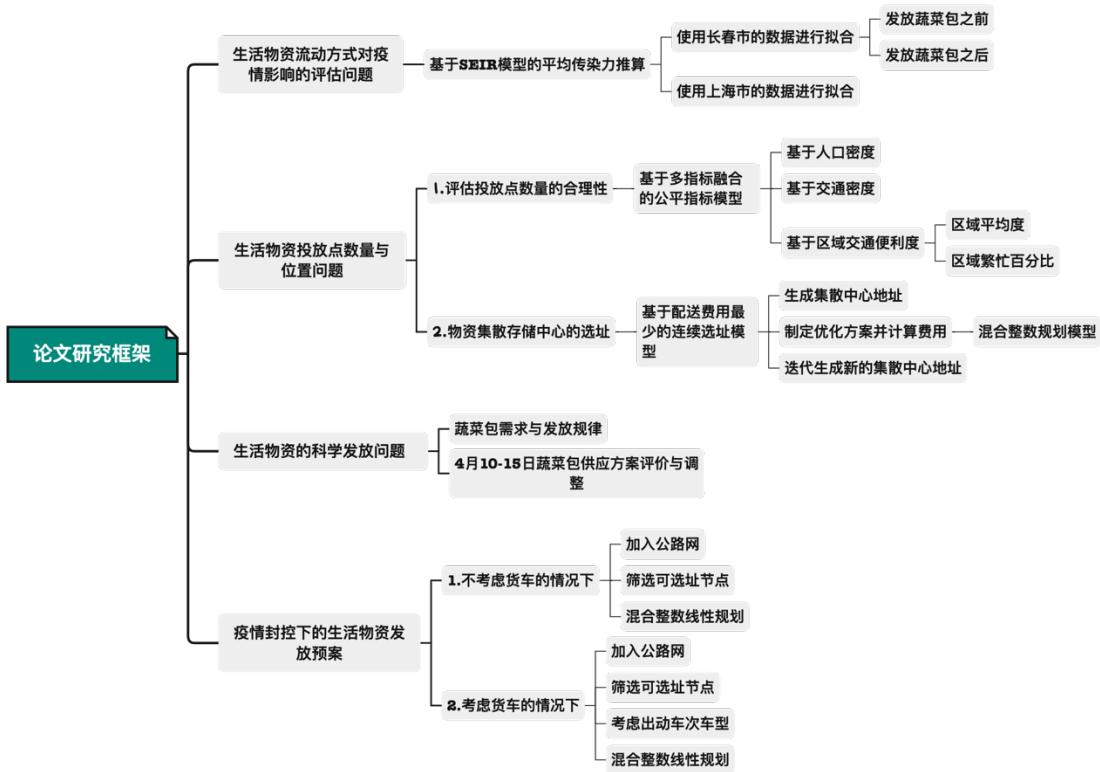


图 1 论文框架图

2 模型假设

根据问题解答和模型建立需要，本文整体作出如下重要假设，涉及的其他假设将在模型具体建立时提出。

- 不考虑人口的出生与死亡、迁入与迁出等更为细致的因素，因此总人数不发生变化；
- 最小时间单位为一天；
- 一个小区只匹配一个配送中心；
- 配送中心符合容量需求；
- 采用地理坐标对配送中心与小区的距离进行求解，并将其作为模型的配送中心与各小区距离数据；
- 集散中心的位置只能从路口节点进行选择；
- 一个集散中心可以服务多个小区；
- 所有的物资必须先从仓库到达集散中心，然后才能经集散中心到小区；
- 第四题集散中心最大数量为第二题中的结果，即 71 个；
- 每个集散中心单日所能中转的最大货运量与第二题相同，为 40000kg（40 吨）。

3 符号说明

表 1 列示了论文中使用的主要符号, 由于使用符号众多, 其余使用的符号、参数等将在使用时注明。

表 1 符号说明

符号	单位	描述
$s(t)$	-	t 时刻易感者人群占总人口的比例
$e(t)$	-	t 时刻暴露者人群占总人口的比例
$i(t)$	-	t 时刻患者人群占总人口的比例
$r(t)$	-	t 时刻治愈康复者人群占总人口的比例
γ	人	日暴露数, 即每个患者每天有效接触的易感者的平均人数
θ	-	日发病率, 即每天发病成为患者的暴露者占暴露者总人数的比率
μ	-	日治愈隔离率, 即每天被治愈的患者人数占病人总数的比率
σ	人	传染期接触数, 即每个患者在传染期内有效接触的易感者人数
<i>Equity</i>	-	投放点数量公平指数
$Number_{drop}$	个	当前行政区投放点数量
<i>Population</i>	人	当前行政区的隔离人口数
$Distance_{road}$	km	当前行政区道路总长度
<i>Area</i>	km^2	当前行政区面积
$Number_{road}$	-	平均度, 即当前行政区平均每个路口连接的公路数
<i>PR</i>	-	繁忙百分比, 即该节点的 PageRank 值
<i>C</i>	元	配送中心向各小区配送生活物资的运输费用总额
(X_i, Y_i)	-	配送中心的坐标
d_{ij}	km	两配送中心坐标之间的距离
(x_i, y_i)	-	小区的坐标
p_j	吨	小区生活物资每日需求量
D_j	km	允许配送的最大距离
q	元	运输单价, 为 0.2 元/kg/km
b_{ij}	元	配送中心 i 到各小区 j 的运输费用
w_{ij}	吨	配送中心 i 到各小区 j 的运输量
v_{ij}	吨	每个分拣中心每天能够分拣的物资量
K	-	备选的二级节点位置, 某一个备选点用 k 来表示
L	-	级节点集合, 某一个三级节点用 1 表示
Q_{kl}	-	从二级节点 k 送到三级节点 1 的供货量
S_{kl}	-	从二级节点 k 到小区 1 的距离
R_k	-	是否要选择 k 作为集散中心, 0 表示不选, 1 表示选
D_l	-	小区 1 的单日需求
B_k	-	二级节点 k 每天用来输送物资的大车出动车次
L_k	-	二级节点 k 每天用来输送物资的小车出动车次

4 问题一：生活物资流动方式对疫情影响的评估问题

4.1 问题背景

COVID-19 大规模疫情爆发期间，封闭式管理的快速清零模式在全国范围内推行，形成科学的统一管理模式十分必要。其中，部分研究人员认为疫情期间采取的蔬菜包发放形式对生活物资的科学管理产生了一定的积极作用，问题一便聚焦于长春市实行蔬菜包发放模式前后防疫效果的评估。

为准确评估长春市实行蔬菜包发放前后的防疫效果，本文根据长春市政府提供的 3 月 5 日至 5 月 23 日的病毒感染人数数据（包括确诊人数和无症状感染人数）建立了 SEIR 传染病模型，通过查阅权威文献确定平均潜伏期、平均暴露数、平均治愈率等研究数据，进而固定系数 θ 和参数 μ ，最终向前求解的参数 γ 的值用以衡量疫情防控效果。此外，为进一步确定与验证研究结论，本文与国内一无蔬菜包发放的清零案例上海市 3 月 28 日至 4 月 30 日的数据进行对比，以提高研究结论的可信度。

4.2 基于疫情影响的 SEIR 传染病模型（SEIR）

4.2.1 模型说明

传染病模型是处理传染病问题的基本数学模型，用以研究传染病的传播速度、空间范围、传播途径、动力学机理等问题，以指导对传染病的有效地预防和控制。按照传染病类型进行分类，常见的传染病模型可以分为 SI 模型、SIR 模型、SIRS 模型、SEIR 模型等；而按照传播机理进行分类，可以分为基于常微分方程、偏微分方程、网络动力学等的传染病模型，本文将契合问题的特殊场景选择合适的传染病模型进行求解。

针对此关于具有潜伏期的新冠疫情传播的问题，传统的 SIR 模型显然已不再适用，在此基础上，我们还需考虑一类需要一段潜伏期才能判断是否感染的人群，即暴露者^[2]。因而，对于存在易感者、暴露者、感染者和康复者四类人群且有潜伏期的疾病，SEIR 模型更为适用，问题一的解决就依赖于 SEIR 模型的应用。

对于 SEIR 传染病模型的符号定义可解释为如下：

- **易感者 (Susceptible-S)**: 指缺乏免疫能力健康人，与感染者接触后容易受到感染；
- **暴露者 (Exposed-E)**: 指接触过感染者但暂时没有传染性的人；
- **患病者 (Infectious-I)**: 指有传染性的病人，可以传播给 S，最后将其变为 E 或 I；
- **治愈康复者 (Recovered-R)**: 指病愈后具有免疫力的人，本文假定患者被治愈后不会重新被感染。

4.2.2 模型假设

- 易感者 (S) 与患病者 (I) 接触后变成暴露者 (E)，暴露者 (E) 经过平均潜伏期 (7.8 天)^[3-6] 可以成为患病者 (I)，患病者 (I) 可以被治愈成为康复者 (R)，康复者 (R) 有一定几率再次成为易感者 (S)，但也有机会不再易感；
- t 时刻各类人群占总人数的比例分别记为 s_t 、 e_t 、 i_t 、 r_t ，起始条件为 s_0

个易感者, e_0 个暴露者, i_0 个感染者, r_0 个康复者;

- 日暴露数记为 γ , 即每个患病者每天有效接触的易感者的平均人数;
- 日发病率记为 θ , 即每天发病成为患病者的暴露者占暴露者总人数的比率;
- 日治愈隔离率记为 μ , 即每天被治愈的患病者人数占病人总数的比率;
- 传染期接触数记为 σ , $\sigma = \lambda/\mu$, 即每个患病者在整个传染期 $\frac{1}{\mu}$ 天内有效接触的易感者人数;
- 不考虑隔离者解除隔离及治愈康复者治愈出院重新成为易感者的情况;
- 不考虑人口的出生与死亡、迁入与迁出等更为细致的因素, 因此总人数不发生变化, 四类人群数量总和等于总人口;
- 最小时间单位为一天。

4.2.3 模型建立

STEP 1 确定微分方程

由上述模型假设, 可得到微分方程如下:

$$\begin{cases} \frac{de_t}{dt} = \lambda s_t i_t - \theta * e_t \\ \frac{ds_t}{dt} = -\lambda s_t i_t \\ \frac{di_t}{dt} = \theta e_t - \mu i_t \\ \frac{dr_t}{dt} = \mu i_t \\ s_t + e_t + i_t + r_t = 1 \end{cases} \quad (1)$$

STEP 2 求解系数 γ

通过查阅官方资料与期刊文献, 得到系数 θ 和参数 μ 的平均值并固定, 已知起始条件为 s_0 个易感者, e_0 个暴露者, i_0 个感染者, r_0 个康复者, 并知 s_t 每日的数据, 由此向前求解, 倒推得到长春市的每日暴露数 γ 。在本问题中, 需要评估蔬菜包发放前后对疫情的影响, 而反映这个影响力即防疫效果的关键指标即为每日暴露数 γ , 本轮疫情从 3 月 4 日开始, 蔬菜包从 3 月 26 日开始发放, 如若 γ 的值在 3 月 26 日前后有较为明显的下降, 从数据的角度看则认为蔬菜包发放模式对疫情防控有一定的积极影响。本文得到的疫情期间 γ 值如图 2 所示。

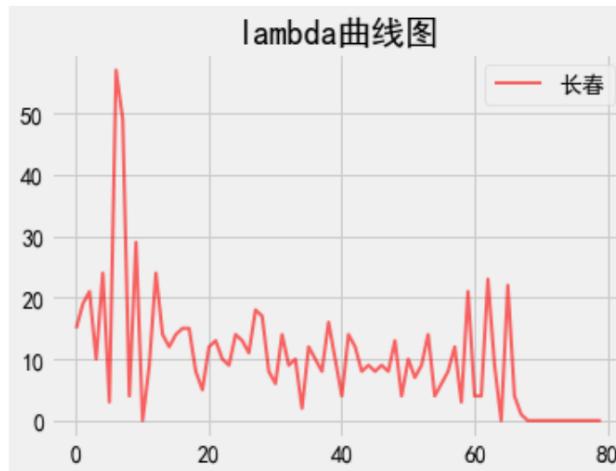


图 2 长春市 lambda 曲线图

STEP 3 上海市对比案例

为进一步确定与验证研究结论，本文选取疫情期间未实施蔬菜包发放模式的上海市作为对比对象，分析其3月28日至4月30日疫情的防控效果，求得这个时段上海市的每日暴露数 γ ，两者对比结果如图4所示。

STEP 4 研究结论

首先，根据长春市日感染数据，本文绘制了感染者每日增长人数曲线图，如图3所示，可以看到长春市每日感染人数在3月5日至4月2日期间呈波动上升的状态，4月2日至5月11日呈波动下降状态，波动幅度较大，且在4月26日前后无明显变化。

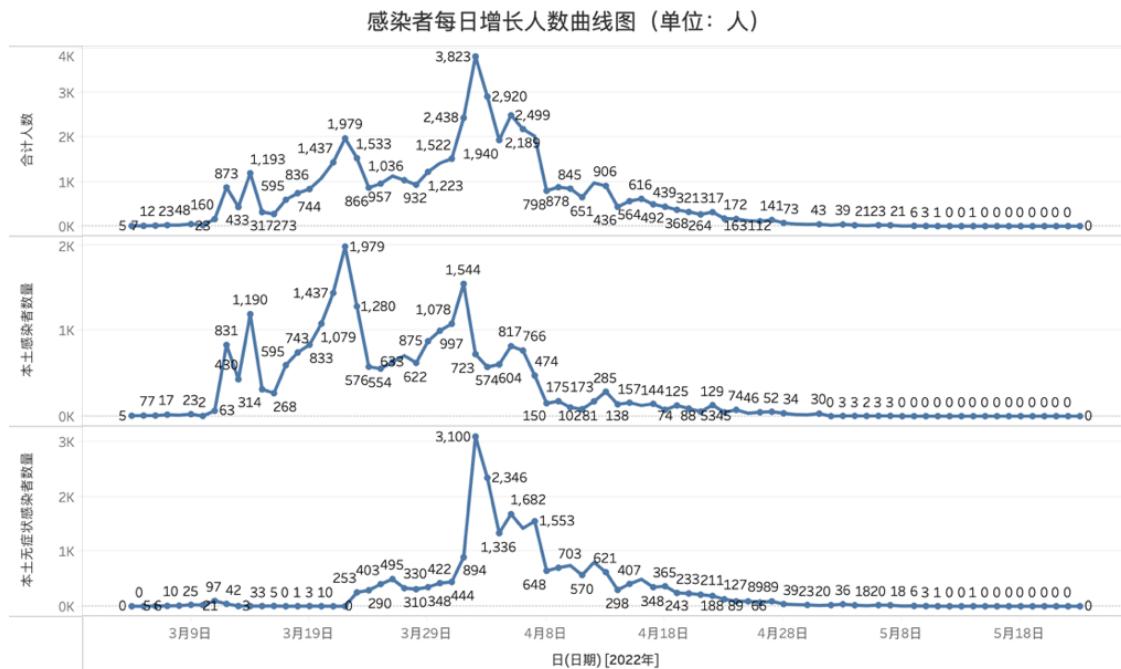


图3 长春市感染者每日增长人数曲线图（单位：人）

其次，根据模型解得的长春市与上海市日暴露人数绘制对比折线图，本文发现长春市日暴露人数在全时段呈大幅波动的状态，疫情开始阶段尤甚，且在4月26日即蔬菜包开始发放日前后日暴露人数无明显变化，并波动上升。在与上海市对比中我们发现，两市暴露人数发展趋势无明显差异，且上海市 γ 值整体偏小，联系到上半年上海疫情的现实情况，并不能说明蔬菜包发放的积极效果。

由此，从数据角度本文认为蔬菜包发放模式对疫情封控效果并无明显影响，从实际角度解释，长春市日暴露人数的明显下降主要归因于“封城”的封闭式管理模式，家家户户宅家已限制了人员流动，减少了社交距离，这对疫情的控制有直接作用，而生活物资的流动属于封控后的后勤保障活动，满足居民的日常生活并提高满意度和幸福感^[7]。诚然，生活物资发放模式在一定程度上会造成疫情二次传播的可能性，蔬菜包的集中式发放可以减少人员接触，但本文认为这不是主要原因，无法对疫情防控产生行之有效的效果。

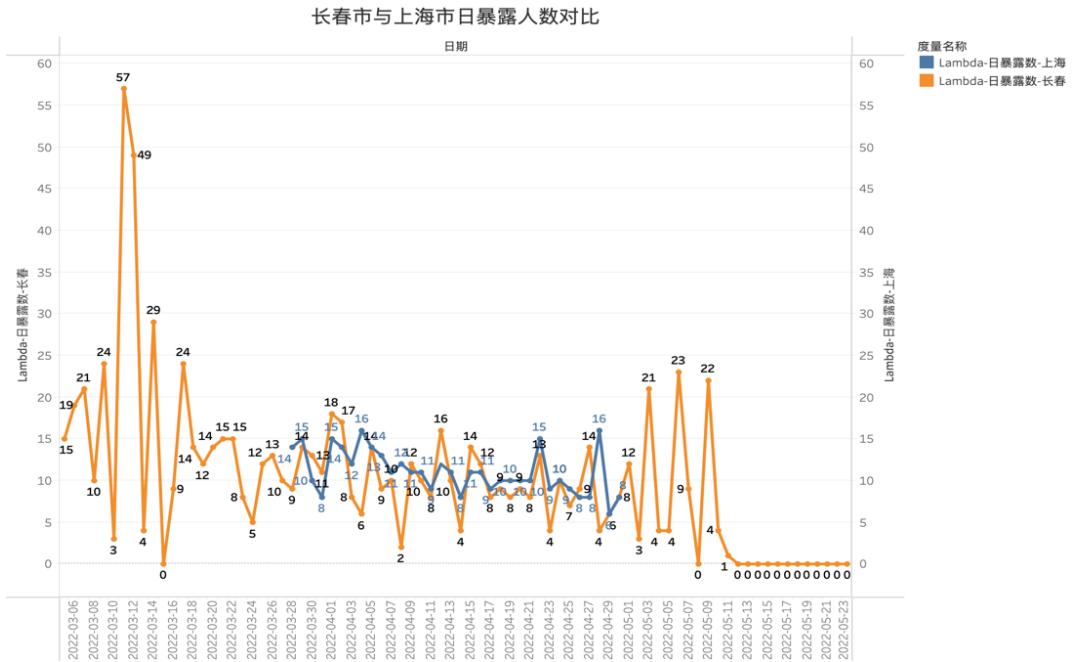


图 4 长春市与上海市日暴露人数对比折线图（单位：人）

5 问题二：生活物资投放点数量与位置问题

5.1 问题背景

聚焦于疫情期间生活物资的科学发放问题，其中之一便是合理规划封控期间包括政府储备物资、外地捐赠物资等在内的大规模物资分拣场所的数量（如各区投放点数量）、规模（如管辖的小区个数、管辖人口数）和位置（如坐标位置、选址半径），以最大程度的提高配送效率、减少人力资源利用成本和减少区域人员流动及接触，提高大规模传染疫情期间的物资调度能力。^[1]

针对问题二中的生活物资投放点数量与位置问题，本文首先进行投放点数量合理性的评估，为此我们构建了一个基于各行政区人口条件、交通运输条件等的公平指数，用以评估在当前长春市投放点分布结果下各区投放点数量的合理性。越靠近公平指标的中位数，则其数量分布越合理，越偏离公平指标的中位数，则其数量分布越不合理。在满足各区基本配送要求基础上，我们对投放点数量过多的行政区进行缩减，对数量不足的予以补足，合理规划现有人力资源和运输能力。

其次，在应急管理主题下，考虑大规模物资分拣场所的数量、规模和位置，给出详实具体的选址信息^[8]。本文建立了基于配送费用最少的选址模型以解决这个多层级配送中心（分拣中心）的问题。

5.2 基于投放点数量合理性的评估模型（EQUI）

STEP 1 建立投放点数量公平指数

借鉴 Vlachopoulou 等(2001)关于仓库选址决策的研究^[9]，本文基于各行政区的隔离人口水平和交通便利水平构建了投放点数量公平指数 *Equity*，认为在考虑交通条件下满足区域人口的日常物资配送要求即为合理。其中，各行政区投放点数量已在附表二给出，各行政区隔离人口数在附件一给出，两者比值为每个投放点服务的隔离人口数量；交通通达度的评价依据中国科学院地理科学与资源研

究所封志明等(2001)构建的的交通通达度评价体系^[10], 以交通密度和交通便捷度作为主要评价内容。由此, 得到的公平指数计算公式如下:

$$Equity = \frac{Number_{drop}}{Population} * Density_{road} * Convenience_{tra}$$

$$\text{即: } Equity = \frac{Number_{drop}}{Population} * \frac{Distance_{road}}{Area} * Number_{road} * Percent$$

其中, $Equity$ 为投放点数量公平指数, $Number_{drop}$ 为当前行政区投放点数量, $Population$ 为该行政区的隔离人口数, $Density_{road}$ 为道路密度, 用道路总长度 ($Distance_{road}$) 与行政区面积 ($Area$) 的比值来衡量, $Convenience_{tra}$ 为交通便利度, 首先通过构建无向图求得当前行政区各个簇节点平均连接的公路数 ($Number_{road}$), 然后应用 PageRank 算法, 综合考虑路口的 PR 值、连接小区数、连接公路数等指标进行聚类, 分别计算每一个行政区在聚类结果中表现较好的路口数占其全部路口数的百分比得出繁忙百分比, 将两者相乘用以衡量该行政区的交通便利度。

STEP 2 计算道路密度

本文根据附件三提供的长春市 9 个行政区交通路口节点数据和路线数据初步绘制了长春市交通网络图, 如图 5 所示。为计算各行政区道路总长度, 我们按路口节点——各小区——各行政区的前后链条匹配各道路所属的行政区, 具体步骤如下:

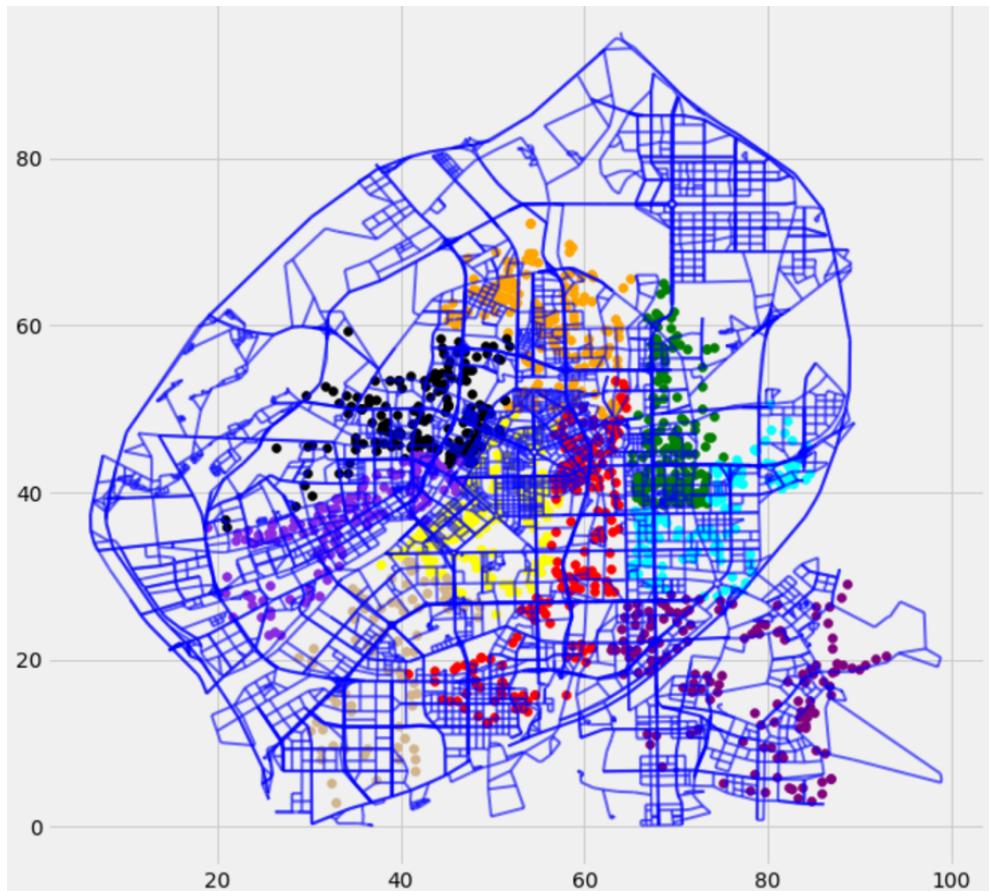


图 5 长春市道路网络图

- 确定各路口节点所属小区。本研究认为在各小区生活物资投放过程中，只需将物资送至距该小区最近的路口节点，即算送达至每一个节点，所以我们将每一路口节点匹配到各个小区。
- 确定各小区所属行政区。因小区的区划属性，根据投票法原则再将各小区归类至9个行政区，即该节点所辖小区主要面积是属于哪个行政区。
- 计算各行政区道路长度。根据路口节点所属的行政区，计算道路路线所属的行政区，如果道路出发点和终止点属于不同的行政区，则该道路路线长度均分，分别属于两个行政区划中，以此得出每个行政区的道路总长度。

在此基础上，计算的道路密度为道路总长度与行政区面积的比值。进一步地，我们将道路网络图细化，将各条道路所属行政区进行可视化，如图6所示。

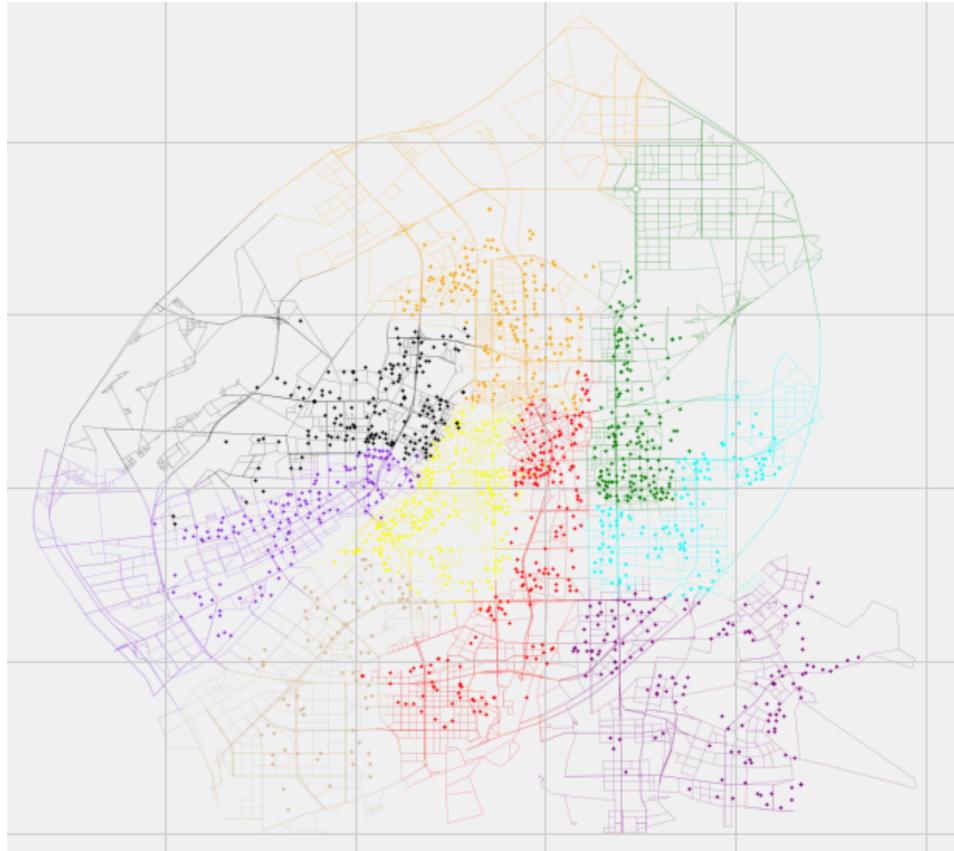


图6 各道路按行政区划分示意图

图中九种不同颜色的点各自代表各行政区的交通路口，不同颜色的线代表各行政区的交通路线，同时我们将其与实际地理地图相比，无明显差距，显示了道路密度计算的准确性。

STEP 3 计算交通便利度

在此步骤中，我们引入评价节点重要性的PageRank算法，该算法是特征向量中心性及其变体的应用，在网页排序领域应用广泛^[11]。PageRank算法认为万维网中一个页面的重要性取决于指向它的其他页面的数量和质量，引申到交通节点中则如果一个节点被很多高质量的节点指向，则这个交通节点的质量也高，我们将每个节点的PR值作为节点重要性排名^[12]。初始时刻，赋予每个节点相同的PR值，然后进行迭代，每一步把每个节点当前的PR值平分给它所指向的所有节点，每个节点的新PR值为它所获得的PR值之和，于是得到节点 v_i 在t时

刻的 PR 值为：

$$PR_i(t) = \sum_{j=1}^n a_{ji} \frac{PR_j(t-1)}{k_j^{\text{out}}} \quad (2)$$

其中 k_j^{out} 为节点 v_i 的出度，迭代直到每个节点的 PR 值都达到稳定为止。

交通便利度的算法本质上是一个节点聚类的问题，该聚类目前有如下特征：

- 节点所辖小区人口数量
- 节点所辖小区数量
- 节点所连接的道路数量
- 节点的 PageRank 值（即 PR 值作为节点重要性排名）

在得到上述三种聚类特征后，对节点进行聚类，将各个簇的节点数占比作为本行政区的交通便利程度的衡量标准，即各个簇节点平均连接的公路数 ($Number_{road}$) 和繁忙百分比 (PR 值) 的乘积，长春市九个行政区的交通便利度在表 2 列示。

STEP 4 计算公平指数

根据上文给出的公平指数计算公式：

$$Equity = \frac{Number_{drop}}{Population} * \frac{Distance_{road}}{Area} * Number_{road} * Percent \quad (3)$$

我们得到长春市九个行政区的投放点数量公平指数，详细数据如下表所示。

表 2 各行政区投放点公平指数详细数据

Region	Number _{drop}	Population	Distance _{road}	Area	Number _{road}	Percent	Equity
二道区	9	42.6	456750.51	965	0.3596	21.05%	7.57
净月区	279	22.8	427476.89	479	0.3240	22.23%	786.73
南关区	261	48.9	454041.21	497	0.3557	26.29%	455.99
宽城区	181	32.6	513788.62	877	0.3255	20.91%	221.41
朝阳区	94	57.8	279065.92	379	0.3978	25.11%	119.62
汽开区	10	21.7	451877.48	110	0.3302	22.28%	139.26
经开区	37	20.3	260600.42	112	0.3365	23.98%	342.16
绿园区	470	38.5	383876.90	301	0.3421	26.24%	1397.53
长春新区	215	36.8	372669.34	499	0.3571	21.05%	328.00

其中， $Equity$ 为投放点数量公平指数， $Number_{drop}$ 为当前行政区投放点数量， $Population$ 为该行政区的隔离人口数， $Distance_{road}$ 为道路总长度， $Area$ 为行政区面积，两者比值为道路密度 $Density_{road}$ ， $Number_{road}$ 为当前行政区平均每个路口连接公路数， $Percent$ 为繁忙百分比，两者乘积为交通便利度 $Convenience_{tra}$ 。

可以看到，各行政区的平均度与繁忙百分比并无显著差距，朝阳区、南关区、绿园区和经开区交通便利度相对较高，宽城区交通便利度较低，在固定公平指数情况下，便利度较高的行政区倾向于削减投放点数量以节约资源，反之亦然，这个规律与后文优化结果大致相符。

STEP 5 评估与优化

通过分析表 2 计算得出的投放点数量公平指数，本文对投放点数量的合理

性进行讨论；假定其余变量保持不变的情况下，通过调节公平指数进行投放点数量的优化，使公平指数偏高或偏低的行政区的投放点数量趋于合理水平。

表中的各区投放点平均数量为 173 个，有四个行政区投放点数量在平均水平以下，*Equity* 指数平均值为 422.03，中位数为 328.00，有三个行政区公平指数在平均水平以下，结合各区实际情况，本文选取公平指数的中位数作为基准对各区投放点数量进行优化，优化后的各行政区投放点数量如表 3 所示。

表 3 各行政区优化后投放点数量

<i>Region</i>	优化前 <i>Number_{drop}</i>	优化后 <i>Number_{drop}</i>	投放点增减 数量	优化前 <i>Equity</i>	优化后 <i>Equity</i>
二道区	9	390	381	7.57	328.09
朝阳区	94	258	164	119.62	328.31
汽开区	10	24	14	139.26	334.24
宽城区	181	268	87	221.41	327.83
长春新区	215	215	0	328.00	328.00
经开区	37	36	-1	342.16	332.91
南关区	261	187	-74	455.99	326.71
净月区	279	116	-163	786.73	327.10
绿园区	470	110	-360	1397.53	327.08

优化后总计的投放点数量为 1604 个，相比优化前 1556 个增加 48 个投放点，差距较小。在原有投放点基础上，我们对二道区、朝阳区、汽开区、宽城区分别增加 381、164、14、87 个投放点，对经开区、南关区、净月区、绿园区分别缩减 1、74、163、360 个投放点，优化后的投放点数量在满足各区居民正常生活的情况下节省了人力资源，较少人员流动与接触。

对于优化前后的实际含义，我们对增减的各行政区进行举例解释。对于二道区来说，优化前仅设立 9 个物资投放点，但其地域面积大且隔离人口众多，交通便利程度在 9 个区中处于末位，无法满足隔离居民日常物资运输需求，通过增加投放点数量可以优先解决此问题；对于绿园区来说，区域面积小且交通便利度很高，不需要大量投放点仍能在优异交通条件下满足配送需求，因而可减少投放点以合理分配资源。

5.3 基于配送费用最少的选址模型（MINC）

5.3.1 模型说明

在此选址问题中，我们将其处理为一个多层级配送中心问题，第一层级配送中心指大规模分拣中心，物资在这里分拣完毕后直接配送到各小区。按照题目理解，这里既可以作为储备物资的仓库，也可以作为分拣场所。

因而在生活物资配送物流体系中，配送中心的选址非常关键，利用一定的评估标准（如适宜性、协调性、战略性、经济性等^[13]）评价备选地址，并从中选取最优配送中心，能够缩短配送时间和提高配送效率^[14]。图 6 为一般情况下多层级配送中心连续选址的基本流程。

在构建多层级配送中心连续选址模型过程中，本文以成本极小化为切入点，使配送中心选址满足配送中心与各小区的配送费用总额为极小值，以完成选址模型的构建。

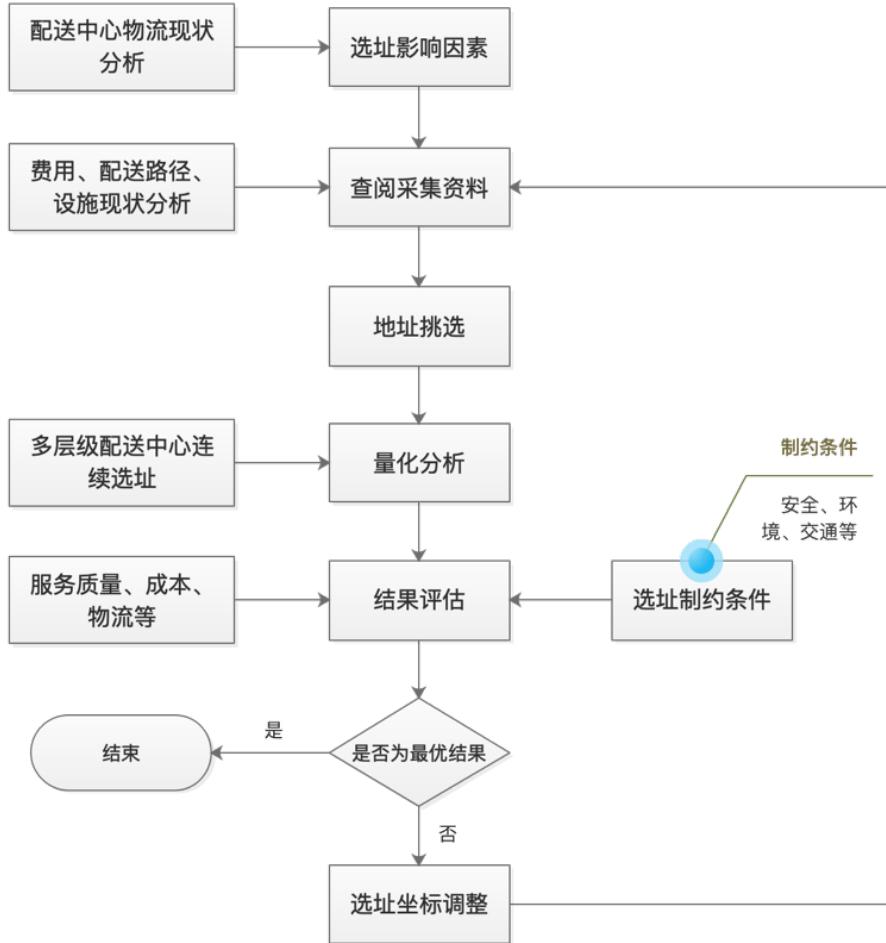


图 7 多层级配送中心连续选址基本流程图

5.3.2 模型建立

构建多层级配送中心选址模型的基本条件如下：

- 根据运输量与运输距离构建运费函数；
- 一个小区只匹配一个配送中心；
- 配送中心符合容量需求；
- 已知各小区的需求量；
- 采用地理坐标对配送中心与小区的距离进行求解，并将其作为模型的配送中心与各小区距离数据；
- 配送中心所供应的人口数不得超过 10 万人；
- 单位运费为 0.2 元/kg/km；
- 每个配送中心的单日最大周转量为 40000kg（此数据由建议供给量乘配送中心最大保障人口计算得出）；
- 小区的需求由该小区的人口建议供给量计算得出。

接下来我们进行模型的构建，模型中涉及的变量如下：将配送中心向各小区配送生活物资的运输费用总额设为 C ；一共有 n 个配送中心，配送中心的坐标表示为 (X_i, Y_i) ，两坐标之间的距离用 d_{ij} 表示；一共有 m 个小区，小区的坐标表示

为 (x_i, y_i) , 其需求量为 p_j , 允许配送的最大距离为 $D_j = 1\text{km}$ 运输单价为 $q = 0.2 \text{ CNY/kg/km}$, 配送中心 i 到各小区 j 的运输费用为 b_{ij} , 配送中心 i 到各小区 j 的运输量为 w_{ij} , 每个分拣中心每天能够分拣的物资量为 v_{ij} 。

表 4 MINC 模型变量表

变量名	单位	描述
C	元	配送中心向各小区配送生活物资的运输费用总额
(X_i, Y_i)	-	配送中心的坐标
d_{ij}	km	两配送中心坐标之间的距离
(x_i, y_i)	-	小区的坐标
p_j	吨	小区生活物资每日需求量
D_j	km	允许配送的最大距离
q	元	运输单价, 为 0.2 元/kg/km
b_{ij}	元	配送中心 i 到各小区 j 的运输费用
w_{ij}	吨	配送中心 i 到各小区 j 的运输量
v_{ij}	吨	每个分拣中心每天能够分拣的物资量

根据上述设定条件, 构建配送费用最少的选址模型, 首先求出配送中心的数量上限, 然后生成配送中心的初始坐标和数量。接着, 准备优化出当前配送中心位置和数量下的最优配送方案, 并且计算出配送费用。最后, 当前配送中心位置和数量下的最优配送方案得出的时候, 通过得出的配送方案和配送费用重新根据迭代条件迭代出新的配送地址。

其目标函数如下所示:

$$minC = \sum_{i=1}^n \sum_{j=1}^m w_{ij} * d_{ij} * q \quad (4)$$

该目标函数的约束条件如下所示:

$$\sum_{i=1}^n w_{ij} = p_j \quad (5)$$

$$(X_i, Y_i) \in \text{长春} \quad (6)$$

$$d_{ij} = \left[(X_i - x_j)^2 + (Y_i - y_j)^2 \right]^{\frac{1}{2}} \leq D_j \quad (7)$$

$$\sum_{i=1}^n w_{ij} \leq v_i \quad (8)$$

其中，约束 5 为运输量需满足各小区的需求量，约束 6 意为配送中心的坐标必须在长春市范围内，约束 7 意为分拣中心每日配送量应小于其承载量，通过上述模型构建我们开始求解配送中心（即政府储备物资和大规模物资分拣场所）的选址。

5.3.3 模型求解

在求解过程中，需要将配送中心的连续选址问题转换成多源 Weber 问题，通过分析选址目标函数，证明了其既非凹函数，也非凸函数，这将导致大量局部最优解的生成^[14, 15]。所以，本文选取启发式算法处理配送中心的连续选址问题^[16]。

首先我们使用聚类的方法求出配送中心上线，共为 71 个。基于已知的 n 个配送中心地址 $(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)$ ，初始化每个配送中心的坐标，设定各小区的需求供应点为距其最近的配送中心，若想令运输费用为极小值，则要求解 X_i 与 Y_i 的偏导数，并使其为 0，可得最优配送中心地址坐标 (X'_i, Y'_i) 的计算公式：

$$X'_i = \frac{\sum_{j=1}^m \frac{w_{ij}q x_j}{d_{ij}}}{\sum_{j=1}^m \frac{w_{ij}q}{d_{ij}}} \quad (9)$$

$$Y'_i = \frac{\sum_{j=1}^m \frac{w_{ij}q y_j}{d_{ij}}}{\sum_{j=1}^m \frac{w_{ij}q}{d_{ij}}}$$

根据上一轮的 (X_i, Y_i) 值和得到的 weight 方案，求解出新的坐标，再反复迭代，直到 (X_i, Y_i) 不再变化为止，图 8 显示了启发式算法的效果，横轴为迭代次数，纵轴为运输成本，启发式算法的迭代过程中，成本与迭代轮数的变化曲线，可以看出，本算法成功让配送费用收敛到较低的值（20 万左右）。

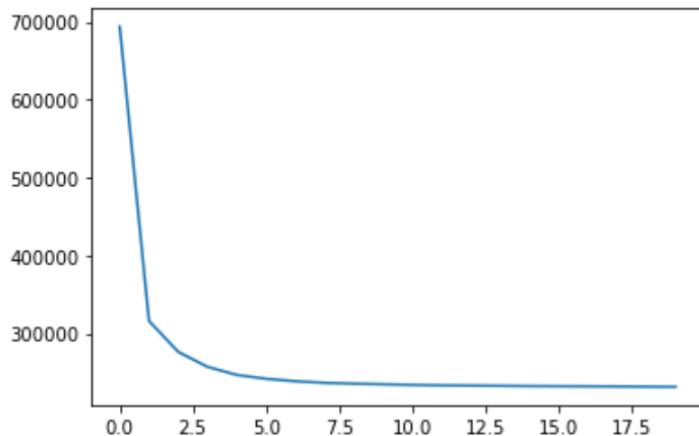


图 8 启发式算法求解效果

此时得到的物流配送中心的最优选址策略，不仅所有配送中心地址均为最优选址，而且运输费用总额也是最少的，各配送中心的坐标、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关键参数如表 5 所示。

表 5 政府储备物资和大规模分拣场所的主要参数

序号	横坐标	纵坐标	所在区域	选址半径 (km)	辖区小区数	辖区内 人口数
1	32.5651	38.3133	汽开区	2.9684	61	100,000
2	67.8487	28.3505	净月区	2.0799	43	73,008
3	50.6514	29.2866	朝阳区	1.6083	26	99,999
4	50.6364	64.4072	宽城区	1.7137	45	72,357
5	68.3059	63.8194	二道区	1.3656	11	44,248
6	59.8058	42.5644	南关区	1.7968	57	100,000
7	74.6996	32.3595	经开区	2.0157	34	48,079
8	25.5701	31.0438	汽开区	2.6570	46	95,371
9	67.9372	41.5926	二道区	1.9347	59	100,000
10	57.1856	48.4425	宽城区	1.7857	41	99,999
11	35.6828	50.1325	绿园区	3.0123	36	89,140
12	37.1959	25.0380	长春新区(高新)	2.0596	26	100,000
13	67.7386	11.0340	净月区	1.4939	6	19,757
14	32.7319	15.2500	长春新区(高新)	2.5136	21	94,486
15	54.8042	39.8317	朝阳区	1.2973	36	100,000
16	85.3059	5.0067	净月区	1.3256	11	15,268
17	48.4122	16.9241	南关区	2.4889	45	89,114
18	69.6580	57.4762	二道区	1.8367	29	80,936
19	72.4351	41.4601	二道区	1.6816	51	99,999
20	48.6171	40.9272	朝阳区	1.4373	49	100,000
21	41.5671	51.0914	绿园区	1.8632	35	75,949
22	50.6670	47.5515	绿园区	1.3022	47	100,000
23	58.4063	68.9312	宽城区	1.7696	16	57,147
24	43.1899	30.8613	朝阳区	1.8502	42	99,999
25	44.6695	36.2656	朝阳区	1.6187	56	100,000
26	59.7000	31.4261	南关区	2.3254	52	94,018
27	83.7185	23.0826	净月区	2.9270	27	30,975
28	83.8753	15.7319	净月区	1.7665	26	16,711
29	58.4079	59.5906	宽城区	1.9404	64	95,945
30	69.7598	46.9684	二道区	1.5440	44	100,000
31	61.4481	46.6290	南关区	1.3404	29	99,999
32	79.5218	7.3632	净月区	1.5905	12	10,278
33	81.6525	45.3585	经开区	1.7481	32	67,636
34	43.5040	45.2312	绿园区	1.8327	78	100,000
35	64.6646	51.5227	二道区	1.5037	28	77,432
36	72.1885	17.9956	净月区	1.9337	22	46,998
37	46.9517	56.8684	绿园区	1.7944	35	70,299
38	39.0710	9.3368	长春新区(高新)	2.8118	15	93,582
39	60.5500	21.6940	南关区	2.3289	31	61,090

在地图上显示为如下：

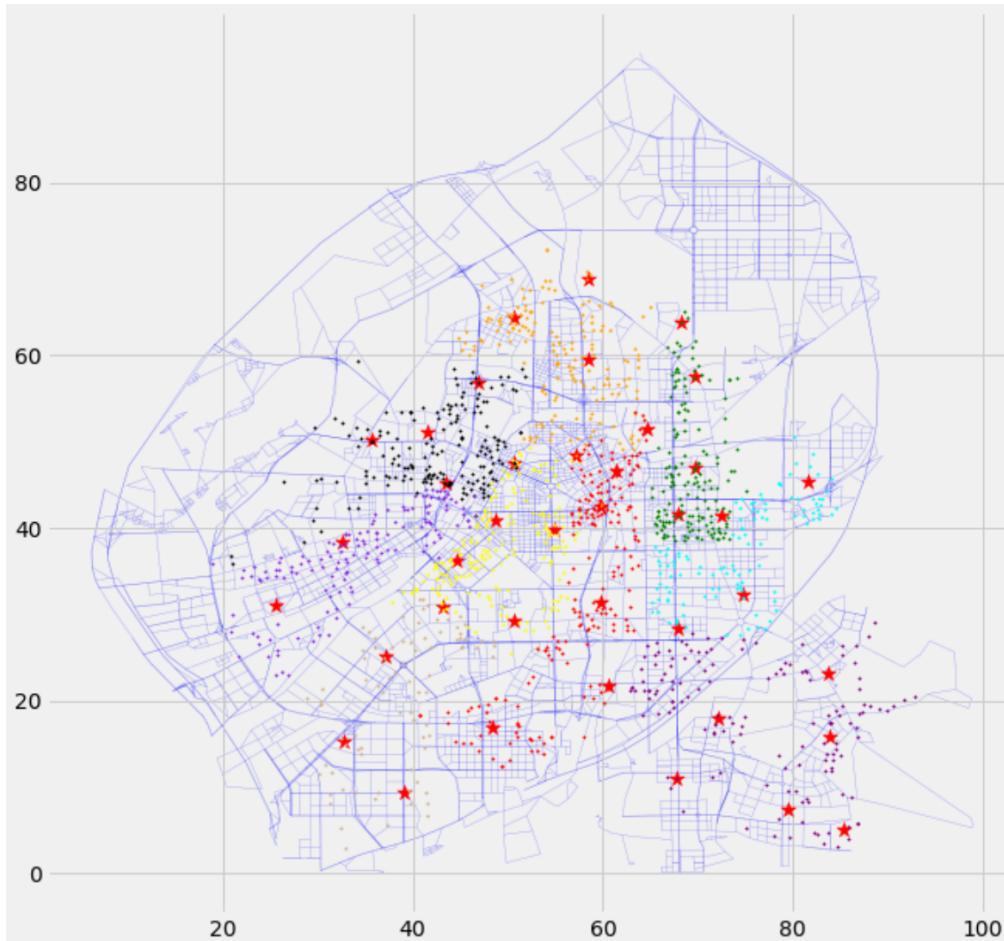


图 9 政府储备物资和大规模分拣场所具体位置

由上述结果可知，在充分考虑未来疫情、自然灾害等特殊事件后，本文认为需要设置 39 个政府储备物资和大规模物资分拣场所，以满足居民生活物资运输的基本需求。

6 问题三：生活物资的科学发放问题

6.1 蔬菜包需求与发放规律

蔬菜作为生活物资中需要程度高且保质期短的特殊产品，需要更多的考虑蔬菜包的发放频率、发放数量、发放重量等要素，以保障居民需求、减少蔬菜浪费、减轻蔬菜打包与发放工作量、减少配送人员对蔬菜的接触等。

为分析长春市此次疫情期间蔬菜包的需求和发放规律，我们依据所提供的蔬菜包数据，提取了各个行政区 4 月 9 日至 4 月 30 日蔬菜包采购袋数、蔬菜包供给重量、蔬菜包发放袋数、蔬菜包发放重量的数据，以分析其发放规律。而对于蔬菜包需求规律，我们假定居民在疫情期间每天所需的蔬菜量是不变的，参考国家卫健委建议的蔬菜摄入量，我们将每人每天所需的蔬菜量固定为 0.4 千克，后续问题也依据此标准。

为探究蔬菜包的发放规律，首先我们统计了各行政区每天的发放袋数，具体情况如图 10 所示，可以看出在蔬菜包发放开始之初数量较少，后随居民需要和

隔离人数增加增加发放数量；随着疫情形势的好转，发放数量日趋减少。部分行政区如南关区、绿园区、汽开区在四月下旬已不需要发放蔬菜包。不同行政区因其人口数量、家庭户数、配送水平、物资数量等的差异，发放袋数显示出较大的差异性，如在4月12日，发放袋数最多为31737袋，最少为5862袋，九区均值为24030袋。

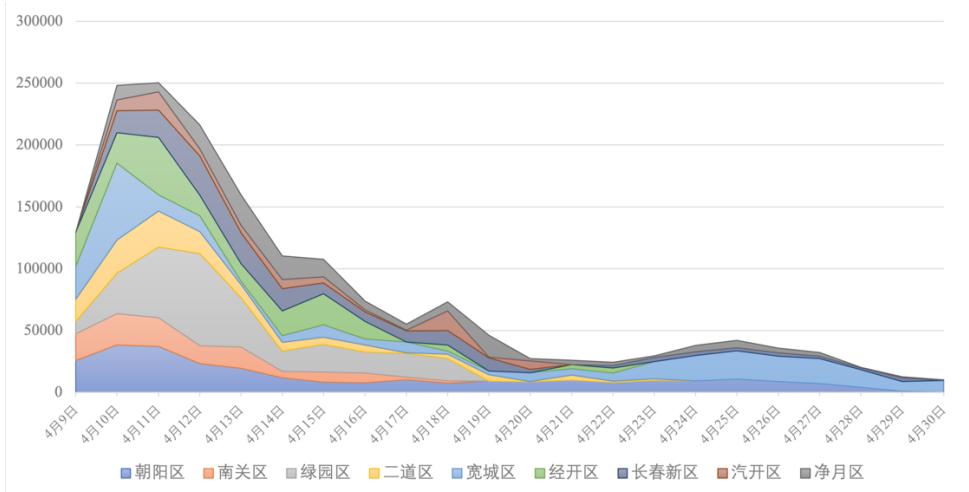


图 10 各行政区每日蔬菜包发放袋数堆积图

其次，我们统计了各行政区每天蔬菜包的发放重量，具体情况如图 11 所示，其变化情况与发放袋数相似，整体均呈下降趋势，不同行政区蔬菜包发放重量有明显差别，如在4月12日，发放重量最大为590.66吨，最小为52.76吨，九区均值为220.16吨。

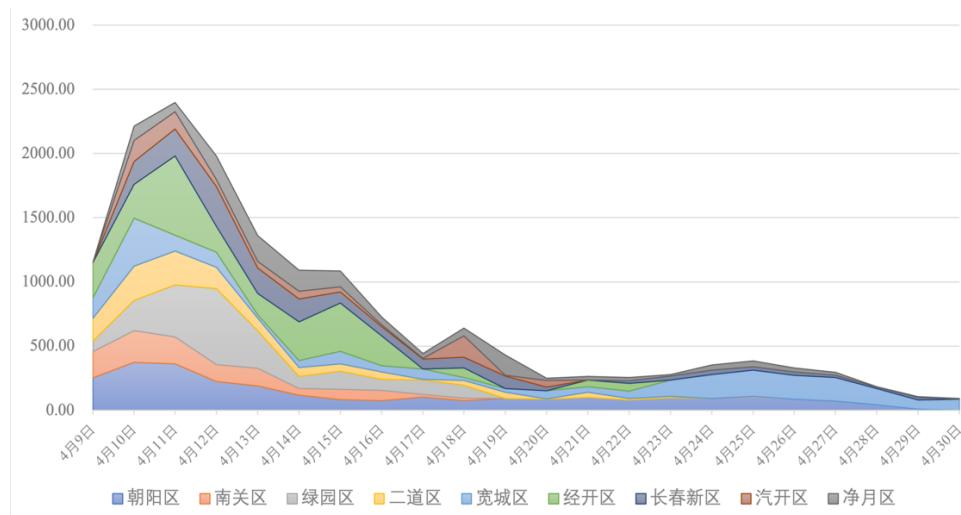


图 11 各行政区每日蔬菜包发放重量堆积图

最后，我们对每袋蔬菜包的平均重量进行分析。每袋蔬菜包的平均重量主要分布在8kg-10kg的范围内，分布较为集中，不同行政区间每袋的平均相差不大，且每个行政区在此时间段内平均重量内部较为一致，如朝阳区每袋蔬菜包平均重量为9.94千克，南关区每袋蔬菜包平均重量为8.98千克。

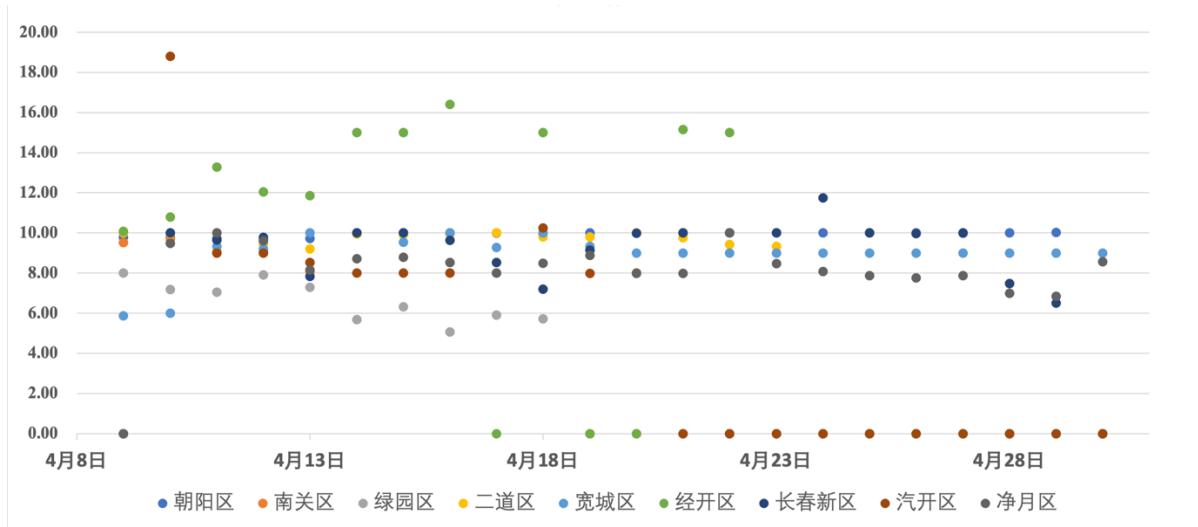


图 12 各行政区每日每袋蔬菜包平均重量

6.2 4月 10-15 日蔬菜包供应方案评价与调整

对于蔬菜包供应方案的评价，首先我们统计了各行政区的人口数、户数、蔬菜包供给袋数、供给重量、发放袋数、发放重量；其次，确定各行政区每日蔬菜需求量，参考国家卫健委建议的蔬菜摄入量，我们将每人每天所需的蔬菜量固定为 0.4 千克，每日需求量则为人口数乘以 0.4 千克。最后，我们求出 4 月 10 日-4 月 15 日每日各行政区的供给需求差与发放需求差，并主要根据供给差与需求差调整每日的蔬菜包供应方案，详细数据如表 6 所示。

表 6 4月 10 日-4月 15 日蔬菜包需求、供给与发放情况

所属区域	人口数(人)	户数(户)	每天需求(吨)	供给袋数(袋)	供给重量(吨)	发放袋数(袋)	发放重量(吨)	供给需求差(吨)	发放需求差(吨)
4月 10 日									
二道区	405,784	166,428	162.31	27,005	256.03	26,908	268.00	93.72	105.69
净月区	204,056	85,748	81.62	11,594	110.00	11,594	110.00	28.38	28.38
南关区	462,146	183,877	184.86	27,671	256.00	25,285	245.10	71.14	60.24
宽城区	306,143	128,899	122.46	83,415	642.55	62,226	373.00	520.09	250.54
朝阳区	548,179	219,874	219.27	40,355	395.20	38,244	374.20	175.93	154.93
汽开区	193,462	81,675	77.38	17,136	328.00	8,777	165.00	250.62	87.62
经开区	189,240	76,909	75.70	24,407	263.42	24,360	263.00	187.72	187.30
绿园区	372,541	155,261	149.02	44,772	340.23	32,812	235.50	191.21	86.48
长春新区	338,274	141,780	135.31	9,187	91.87	17,874	178.70	-43.44	43.39
4月 11 日									
二道区	405,784	166,428	162.31	27,957	263.27	29,122	265.20	100.96	102.88
净月区	204,056	85,748	81.62	7,462	74.62	7,462	74.62	-7.00	-7.00

南关区	462,146	183,877	184.86	23,013	219.36	23,224	210.68	34.50	25.82
宽城区	306,143	128,899	122.46	23,126	221.13	13,251	123.69	98.68	1.23
朝阳区	548,179	219,874	219.27	38,981	377.69	37,190	361.57	158.42	142.30
汽开区	193,462	81,675	77.38	16,376	147.38	14,722	132.50	70.00	55.11
经开区	189,240	76,909	75.70	51,347	675.73	46,425	616.41	600.04	540.72
绿园区	372,541	155,261	149.02	69,828	497.44	57,047	402.47	348.42	253.45
长春新区	338,274	141,780	135.31	33,628	325.97	21,894	211.22	190.66	75.91

4月12日

二道区	405,784	166,428	162.31	31,212	296.85	17,698	167.00	134.54	4.69
净月区	204,056	85,748	81.62	23,333	222.00	19,333	186.00	140.38	104.38
南关区	462,146	183,877	184.86	22,668	205.89	14,447	131.90	21.03	-52.96
宽城区	306,143	128,899	122.46	8,549	76.19	12,800	117.87	-46.27	-4.58
朝阳区	548,179	219,874	219.27	29,120	280.24	23,125	223.99	60.97	4.72
汽开区	193,462	81,675	77.38	8,417	75.75	5,863	52.76	-1.63	-24.62
经开区	189,240	76,909	75.70	34,988	462.89	16,664	200.77	387.19	125.07
绿园区	372,541	155,261	149.02	118,057	990.49	74,611	590.66	841.47	441.65
长春新区	338,274	141,780	135.31	13,609	127.48	31,737	310.48	-7.83	175.17

4月13日

二道区	405,784	166,428	162.31	30,281	292.03	10,637	97.92	129.72	-64.39
净月区	204,056	85,748	81.62	23,126	185.00	24,522	200.00	103.38	118.38
南关区	462,146	183,877	184.86	11,857	90.98	17,112	138.27	-93.88	-46.59
宽城区	306,143	128,899	122.46	2,836	28.36	2,737	27.37	-94.10	-95.09
朝阳区	548,179	219,874	219.27	12,010	120.10	19,476	189.27	-99.17	-30.00
汽开区	193,462	81,675	77.38	5,431	40.70	6,045	51.58	-36.68	-25.81
经开区	189,240	76,909	75.70	13,000	189.00	13,994	165.95	113.30	90.25
绿园区	372,541	155,261	149.02	26,809	164.00	39,967	291.53	14.98	142.51
长春新区	338,274	141,780	135.31	20,000	180.00	25,255	198.02	44.69	62.71

4月14日

二道区	405,784	166,428	162.31	12,909	129.00	7,142	71.00	-33.31	-91.31
净月区	204,056	85,748	81.62	13,956	125.12	18,948	165.06	43.50	83.44
南关区	462,146	183,877	184.86	5,070	50.70	5,070	50.70	-134.16	-134.16
宽城区	306,143	128,899	122.46	4,511	45.11	5,385	53.85	-77.35	-68.61
朝阳区	548,179	219,874	219.27	11,874	118.70	11,874	118.70	-100.57	-100.57
汽开区	193,462	81,675	77.38	16,127	129.00	7,500	60.00	51.62	-17.38

经开区	189,240	76,909	75.70	20,000	300.00	20,000	300.00	224.30	224.30
绿园区	372,541	155,261	149.02	16,356	93.10	16,356	93.10	-55.92	-55.92
长春新区	338,274	141,780	135.31	5,000	50.00	17,913	179.13	-85.31	43.82
4月15日									
二道区	405,784	166,428	162.31	5,661	56.00	5,748	57.00	-106.31	-105.31
净月区	204,056	85,748	81.62	11,698	105.22	14,136	124.22	23.60	42.60
南关区	462,146	183,877	184.86	8,040	80.40	8,040	80.40	-104.46	-104.46
宽城区	306,143	128,899	122.46	13,467	129.95	10,244	97.72	7.49	-24.74
朝阳区	548,179	219,874	219.27	8,846	88.50	8,285	82.90	-130.77	-136.37
汽开区	193,462	81,675	77.38	5,000	40.00	5,000	40.00	-37.38	-37.38
经开区	189,240	76,909	75.70	29,000	431.00	25,000	375.00	355.30	299.30
绿园区	372,541	155,261	149.02	22,380	141.30	22,380	141.30	-7.72	-7.72
长春新区	338,274	141,780	135.31	7,405	74.05	8,731	87.31	-61.26	-48.00

为明显直观的看出蔬菜包的供需差异，我们举例绘制了4月10日蔬菜包的当日需求量、当日采购量和当日分发量的条形图，如图13所示。根据图形可以直观看出4月10日蔬菜包的供需不平衡，采购量与分发量均明显大于需求量，这有可能是疫情初期对隔离人数实际需求预估不准确的原因，需减少采购量与分发量以减少资源浪费。

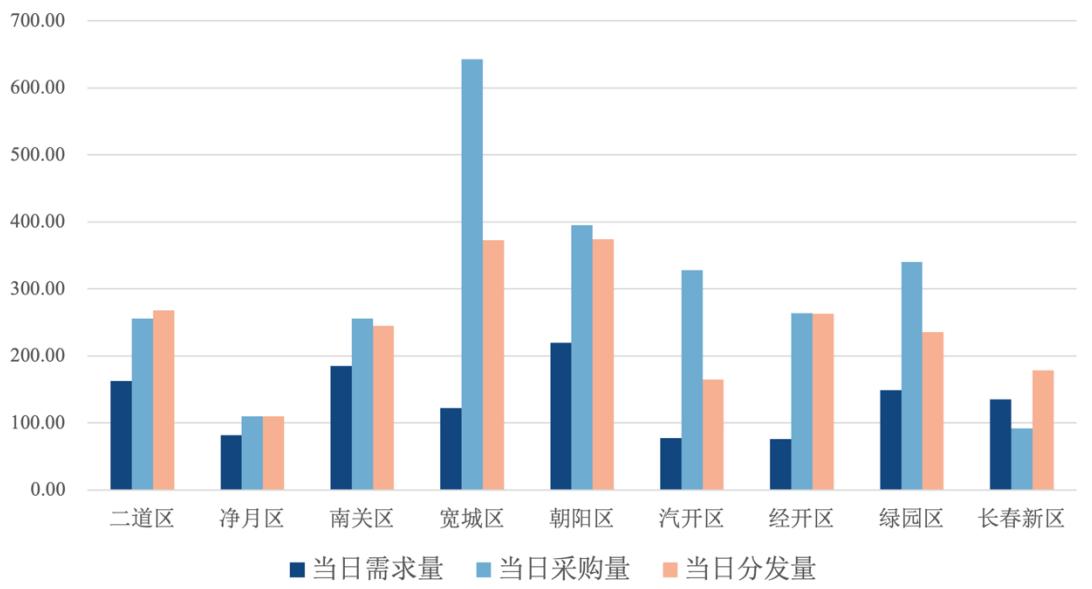


图13 4月10日各行政区蔬菜包供需图 (单位: 吨)

经过上述的数据准备过程，我们对4月10日-4月15日的蔬菜包供应方案进行评价与优化。整体发放过程呈现出供需失衡的状态，4月10日-4月12日供过于求，采购与分发的蔬菜包过多，且因蔬菜的保质期较短可能造成蔬菜浪费，在相同的发放频率下，发放蔬菜过多导致蔬菜不能在一个发放周期内吃完，也造成了发放工作人员人力资源的浪费与工作量的加大。4月13日-4月15日供小

于求，采购与分发的蔬菜包过少，无法满足居民日常生活需求，应适当减少采购与发放数。

对于 4 月 10 日- 4 月 15 日的蔬菜包供应方案调整，表 6 的供给需求差和发放需求差两列已基本说明调整的方向与数量，需求差为正则表明供过于求，需减少对应数量的蔬菜包，需求差为负则表明供不应求，需增加对应数量的蔬菜包，如 4 月 10 日二道区应减少采购 93.72 吨蔬菜包，并减少发放 105.69 吨蔬菜包，4 月 15 日长春新区应增加 61.26 吨的采购量，并多分发 48.00 吨的蔬菜包，表中具体数值可做参考，需根据实际情况加以调整。

此外，对于蔬菜包每袋重量、发放频率、每户发放重量等问题本文也有一定思考，如蔬菜包每袋重量可以保持在 9kg-10kg 之间，发放频率视情况 2-3 天发放一次，每户每天可发放 1.5kg-2.5kg 蔬菜，更为细节的方案可以再做探讨。

7 问题四：疫情封控下的生活物资发放预案

7.1 问题背景

此问题要求在第二、三问的基础上做出特殊时期保障居民生活物资供应的详细预案（有序网络图），网络上游是各项物资来源，中游是各项物资的集散地，网络下游是长春市所有小区。在完成有序网络图后我们进一步考虑用卡车运送物资。而本文以混合整数线性规划为主，在假定了物资配送中心上限的情况下，解决了这个问题。

7.2 基于混合整数线性规划的多层次物流网络选址模型（MILP）

在第二问中，本文已经通过最短路距离构建了网络图，在第四问中要求基于真实路线进行规划，构建有序网络图，同时在构建完成后要求考虑使用货车来进行运输，对比两种条件下分别构成的有序网络图。故而在题目要求下，本文采用了基于混合整数线性规划（MILP）的多层次物流网络选址模型。

7.2.1 模型说明

由题可知，每个区都应该有一个物资来源仓库，一共九个仓库。在长春市的保供网络中，一共存在三个级别的节点，分别是：

- 一级节点：物资来源仓库（9 个），题目要求每个区都得有一个。
- 二级节点：物资集散中心。
- 三级节点：需要被运输蔬菜的小区，一共有 1409 个。

模型的核心在于从给定的网络节点中选择物资集散中心的选址和数量，同时确定其服务的小区，使得总成本最小。

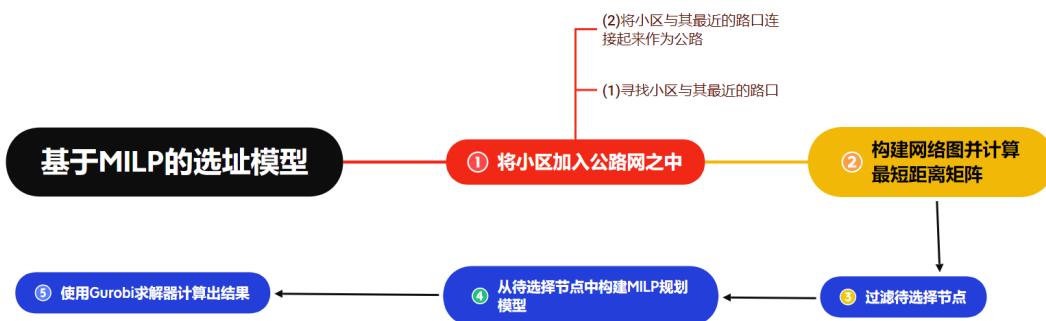


图 14 基于 MILP 的选址模型

本文所建立的模型基于真实路线，因而首先将小区节点加入公路网之中，然后构建了网络图并且计算了公路网中节点之间的最短距离。在构建了公路网为基础的图上，构建 MILP 规划模型，求出模型结果。

7.2.2 模型假设与变量

- 集散中心最大数量为第二题中的结果，即 71 个。
- 集散中心的位置只能从路口节点进行选择，故而已知所有待选位置。
- 一个集散中心可以服务多个小区。
- 所有的物资必须先从仓库到达集散中心，然后才能经集散中心到小区。
- 每个集散中心单日所能中转的最大货运量与第二题相同，为 40000kg(40 吨)。

表 7 MILP 模型变量表

变量名	单位	描述
K	-	备选的二级节点位置，某一个备选点用 k 来表示
L	-	级节点集合，某一个三级节点用 l 表示
Q _{kl}	-	从二级节点 k 送到三级节点 l 的供货量
S _{kl}	-	从二级节点 k 到小区 l 的距离
R _k	-	是否要选择 k 作为集散中心，0 表示不选，1 表示选
D _l	-	小区 l 的单日需求

7.2.3 模型建立与求解

本文建立的基于 MILP 的选址模型如下：

$$\min Z = \sum_{k=1}^{1106} \sum_{l=1}^{1409} Q_{kl} S_{kl} R_k \quad (10)$$

$$\sum_{k=1}^{1106} Q_{kl} R_k = D_l (l = 1, 2, 3, \dots, 1409) \quad (11)$$

$$\sum_{l=1}^{1409} Q_{kl} \leq 40000 (k = 1, 2, 3, \dots, 1106) \quad (12)$$

$$\sum_{k=1}^{1106} R_k \leq 71 \quad (13)$$

约束 1 表示必须满足小区需求，约束 2 表示不能超出集散中心的中转能力，约束 3 表示选择站点不得超过 71 个，使用 Gurobi 优化器对模型进行求解，求解得出的模型结果如表所示。

表 8 MILP 的选址模型求解结果

节点编号	横坐标	纵坐标	所在区域
50	36.0777	26.9145	长春新区
832	35.38287	39.9084	汽开区
114	30.52418	19.87043	长春新区
2170	74.59688	28.5144	经开区

259	42.6475	26.87423	长春新区
2311	74.49263	33.21283	经开区
8489	83.92503	4.111467	净月区
2368	73.71915	39.7196	二道区
8523	67.75328	43.14067	二道区
341	54.89843	26.58453	南关区
2402	66.40535	39.63407	二道区
361	54.50023	46.8974	朝阳区
2416	70.02693	42.07753	二道区
2473	67.3589	44.8662	二道区
464	50.30985	32.82167	朝阳区
2525	62.21218	48.2646	南关区
8694	53.618	56.29123	宽城区
2555	69.93763	48.28693	二道区
523	41.30598	43.7594	汽开区
573	30.286	35.78437	汽开区
4771	72.47325	16.8627	净月区
2725	21.12865	35.06737	汽开区
4856	81.60525	23.5711	净月区
4881	35.67723	49.9512	绿园区
4908	81.1949	41.76777	经开区
835	44.40128	31.08427	朝阳区
6981	43.3606	34.74203	朝阳区
7007	46.5579	36.51817	朝阳区
7047	65.3883	18.3269	净月区
916	47.38085	43.99027	绿园区
2991	58.73483	20.56367	南关区
964	55.4905	39.66487	朝阳区
987	49.42955	38.1282	朝阳区
3056	49.80903	62.94903	宽城区
9230	46.80702	56.84633	绿园区
9262	32.6463	15.6609	长春新区

在求解出二级节点（集散中心）的位置后，通过同区域二级节点的位置加权平均求出仓库的位置如表所示。

表 9 同区域二级节点位置加权平均求得的仓库位置

节点编号	区域	仓库中心横坐标	仓库中心纵坐标
1	二道区	68.83	48.48
2	净月区	77.27	16.44
3	南关区	54.53	30.24
4	宽城区	56.56	60.22
5	朝阳区	49.66	37.17

6	汽开区	30.70	36.26
7	经开区	73.22	34.97
8	绿园区	39.45	50.13
9	长春新区(高新)	35.58	17.28

在地图中显示为如下位置：

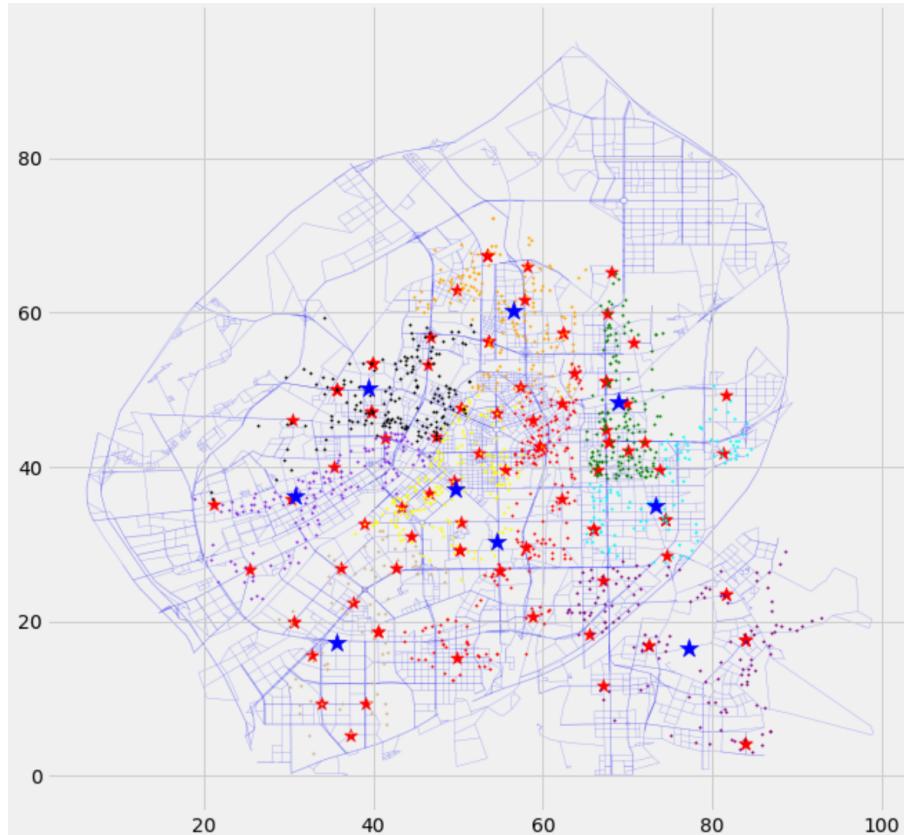


图 15 仓库具体位置

7.2.4 模型改进（考虑货车运输）

题目中要求，考虑两种货车进行运输，大货车一次载重 10000kg，小货车一次载重 4000kg。相比起上文的模型，这里需要补充变量。补充假设为大车出动一次的成本是小车出动一次的 2 倍，则模型修改为：

$$\min Z = \sum_{k=1}^{1106} \sum_{l=1}^{1409} Q_{kl} S_{kl} R_k + 2 \sum_{k=1}^{1106} R_k B_k + \sum_{k=1}^{1106} R_k L_k \quad (14)$$

$$\sum_{l=1}^{1409} Q_{kl} R_k = D_l \quad (l = 1, 2, 3, \dots, 1409) \quad (15)$$

$$\sum_{k=1}^{1106} Q_{kl} \leq 40000 \quad (k = 1, 2, 3, \dots, 1106) \quad (16)$$

$$\sum_{l=1}^{1106} Q_{kl} \leq 71 \quad (17)$$

$$\leq B_k * 10000 + L_k * 4000 \quad (k = 1, 2, \dots, 1106) \quad (18)$$

约束 1 表示必须满足小区需求，约束 2 表示不能超出集散中心的中转能力，约束 3 表示选择站点不得超过 71 个，约束 4 表示车辆出动车次必须把该有的任务送完。

求解后结果如下：

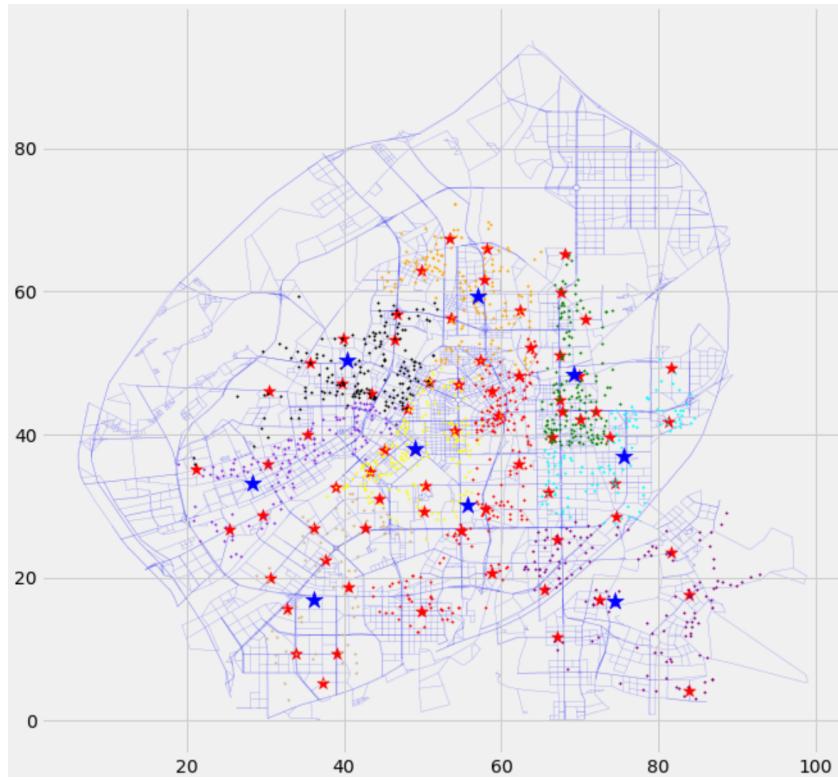


图 16 改进后模型求解结果

有图可以看出，集散中心的位置变化了一小部分，而物资仓库的位置大致维持不变，侧面也证明了本模型的稳健性。

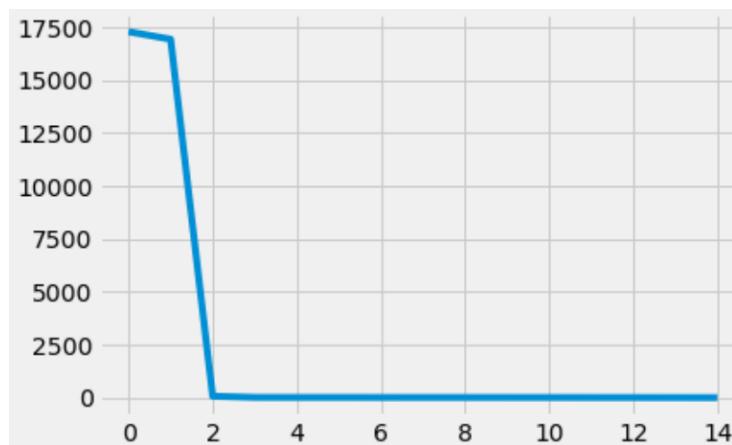


图 17 成本变化图

本模型是以成本最小化为目标函数，综合考量了汽车运输的实际情况，本分配方案的最大优势就是贴近公路网实际和成本较低。

8 模型总结与评价

8.1 模型建立优点

思路清晰，多方面对比。本文在第一问中的 SEIR 模型推算出了长春市本地的感染接触能力后，还搜寻上海市的数据加以对比计算，佐证我们在第一问的结论。

充分考虑了多方面的指标进行评价。在第二问中的合理性评价中，本文既通过计算图指标得出道路因素，还考虑了人口因素和聚类因素等多种因素来综合评价一个区的禀赋条件，并最终得出结论。同时，本文在第二问中采用的启发式算法的可学性较强，迭代条件源自参考文献中的迭代方式进行修改。收敛速度快，科学依据强。

第三问中考虑全面，论述清晰。

第四问模型的稳健性较高。在问题四中无论是否考虑卡车运送，二级配送节点出现变化不大，一级配送节点（仓库）的位置都没有发生剧烈变化。

8.2 模型建立缺点

考虑条件仍然不够全面，本文没有考虑自然灾害和交通拥堵对物资调配带来的影响。构建小区到公路的道路仍然值得深入研究。

参考文献

- [1] 金卫健, 徐浩, 黄传峰, et al. 基于前景理论的大规模传染疫情应急管理决策研究 [J]. 中国管理科学: 1-13.
- [2] 范如国, 王奕博, 罗明, et al. 基于 SEIR 的新冠肺炎传播模型及拐点预测分析 [J]. 电子科技大学学报, 2020, 49(03): 369-74.
- [3] 刘小惠, 何阳, 麻先恩, et al. 有关新冠肺炎潜伏期和疑似期的统计数据: 基于湖北省外 2172 条确诊数据 [J]. 应用数学学报, 2020, 43(2): 278-94.
- [4] 张奕, 孙瑛, 段玮. 新型冠状病毒肺炎患者出院后核酸复阳发生率的 Meta 分析 [J]. 国际病毒学杂志, 2021, 28(1): 6-10.
- [5] 张泽娜, 高玮, 路滟, et al. 新冠肺炎病例密切接触者的感染情况分析 [J]. 现代预防医学, 2020, 47(24): 4.
- [6] 谭索怡, 曹自强, 秦砾, et al. 基于密切接触者人数推断新冠肺炎疫情发展趋势 [J]. 电子科技大学学报, 2020, 49(5).
- [7] BADR H S, DU H, MARSHALL M, et al. Association between mobility patterns and COVID-19 transmission in the USA: a mathematical modelling study [J]. The Lancet Infectious Diseases, 2020, 20(11): 1247-54.
- [8] 姜凯凯, 高混尘, 孙洁. 依托便利店构建生活物资应急配送终端体系——以日本便利店的灾后救援经验为例 [J]. 国际城市规划, 2021, (036-005).
- [9] VLACHOPOULOU M, SILLEOS G, MANTHOU V. Geographic information systems in warehouse site selection decisions [J]. International Journal of Production Economics, 2001, 71(1): 205-12.
- [10] 封志明, 刘东, 杨艳昭. 中国交通通达度评价:从分县到分省 [J]. 地理研究, 2009, 28(02): 419-29.
- [11] BRIN S, PAGE L. The anatomy of a large-scale hypertextual Web search engine [J]. Computer

- Networks and ISDN Systems, 1998, 30(1): 107-17.
- [12] 任晓龙, 吕琳媛. 网络重要节点排序方法综述 [J]. 科学通报, 2014, 59(13): 1175-97.
- [13] 杨舒慧, 何德贵, 梁艳梅. 广州大学城快递物流共同配送中心选址模型构建与仿真实验 [J]. 物流技术, 2017, 36(09): 92-6.
- [14] 李桂娥. 冷链物流多层级配送中心连续选址模型构建 [J]. 计算机仿真, 2022, 39(04): 423-7.
- [15] 张苏, 吴晨晨, 蒋建林, et al. 连续设施选址:模型、方法与应用 [J]. 运筹与管理, 2020, 29(05): 84-95.
- [16] 杨沐明, 黄亚魁, 戴彧虹. 一类多商品设施选址问题的基于线性松弛解的启发式方法 [J]. 运筹学学报, 2019, 23(03): 15-26.

第1题

```
from scipy.integrate import odeint # 导入 scipy.integrate 模块
import numpy as np
import pandas as pd

def dySEIR(y, t, lamda, theta, mu): # SEIR 模型, 导数函数
    s, e, i = y
    ds_dt = -lamda*i

    de_dt = lamda*i - theta*e
    di_dt = theta*e - mu*i
    return np.array([ds_dt, de_dt, di_dt])

class LambdaSolver:
    def __init__(self) -> None:
        self.actually_injected = pd.read_csv(
            'Data\Day_Population.csv')[['合计']].to_list()
        # self.actually_injected = pd.read_excel(
        #     'Data\Day_Shanghai.xlsx')[['合计']].to_list()

    def loss(self, lamda, pre_value, after_value):
        """计算当前lamda造成的损失值
        lamda:当前的Lambda值,
        pre_value:前一天的确诊数,
        after_value:后一天的确诊数"""
        population = 9066906 # 总人数
        theta = 0.15 # 日发病率, 每天密接转阳的比率
        mu = 0.75 # 日治愈率, 每天治愈的患者人数占患者总数的比例
        tEnd = 2 # 预测日期长度
        i0 = pre_value/population # 患病者比例的初值
        e0 = i0*8 # 潜伏者比例的初值
        s0 = 1-i0-e0 # 易感者比例的初值
        Y0 = (s0, e0, i0) # 微分方程组的初值
        # odeint 数值解, 求解微分方程初值问题
        t = np.arange(0.0, tEnd, 1) # (start,stop,step)
        ySEIR = odeint(dySEIR, Y0, t, args=(lamda, theta, mu)) # SEIR 模型
        pre_injected = ySEIR[:, 2] * population
        pre_injected = pre_injected[1] - pre_value
        ture_injected = after_value
        return (pre_injected-ture_injected)**2

    def found_best_lambda(self):
        """
        使用简单的for循环来优化每天的lambda
        """
        actually_injected = self.actually_injected
        lambda_result = []
        for i in range(len(actually_injected) - 1):
            pre_value = actually_injected[i] # 前一天的数据
            after_value = actually_injected[i+1] # 后一天的数据
            result = self.loss(lamda=1, pre_value=pre_value, after_value=after_value)
```

```

    loss_list = list()
    for j in range(100):
        loss_list.append(self.loss(j, pre_value, after_value))
    lambda_result.append(np.argmin(loss_list))
    return lambda_result

```

第2题前半部分：

```

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
plt.style.use('fivethirtyeight')

class Solver:
    def __init__(self) -> None:
        self.community = pd.read_excel('Data/community.xlsx')
        self.intersection_node = pd.read_excel('Data\intersection_node.xlsx')
        self.intersection_route = pd.read_excel('Data\intersection_route.xlsx')

    def draw_map(self):
        intersection_route = self.intersection_route
        intersection_node = self.intersection_node
        community = self.community
        routes = intersection_route[['路线起点', '路线终点']]
        x_index = intersection_node['路口横坐标'].tolist()
        y_index = intersection_node['路口纵坐标'].tolist()
        area = list(community['所属区域'].value_counts().index)
        colors = ['red', 'yellow', 'black', 'green',
                  'orange', 'purple', 'blueviolet', 'cyan', 'tan']
        area_color_dict = dict(zip(area, colors))
        plt.figure(figsize=(10, 10))
        for i in range(routes.shape[0]):
            try:
                start = routes.iloc[i, 0]-1
                end = routes.iloc[i, 1]-1
                plt.plot([x_index[start], x_index[end]], [y_index[start],
                                              y_index[end]],
                        color='b', alpha=0.6, lw=0.2, zorder=0.2)
            except:
                print(start, end)
        for i in range(community.shape[0]):
            x_index = community.iloc[i, 4]
            y_index = community.iloc[i, 5]
            area = community.iloc[i, 7]
            plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
        plt.show()

    def community_connect_intersection(self):
        community = self.community
        intersection_node = self.intersection_node

```

```

community_intersection = list()
for i in range(community.shape[0]):
    community_node_x_index = community.iloc[i, 4]
    community_node_y_index = community.iloc[i, 5]
    distance = list()
    for j in range(intersection_node.shape[0]):
        intersection_node_x = intersection_node.iloc[j, 1]
        intersection_node_y = intersection_node.iloc[j, 2]
        di = np.sqrt((community_node_x_index - intersection_node_x)**2 + (community_node_y_index - intersection_node_y)**2)
    distance.append(di)
    community_intersection.append(np.argmin(distance) + 1)
community['最近路口节点'] = community_intersection
community.to_csv('TempResult\社区路口连接数据.csv', index=False)

def intersection_area(self):
    """本函数用于计算有小区连着的路口节点所属的行政区"""
    community_intersection = pd.read_csv('TempResult\社区路口连接数据.csv')
    a = community_intersection.groupby(
        ['最近路口节点', '所属区域'], as_index=False).count()
    groups = a.groupby('最近路口节点')
    intersection_node = list()
    intersection_areas = list()
    intersection_communities = list()
    for i in groups:
        node, node_data = i
        areas = node_data['所属区域'].tolist()
        counts = node_data['小区编号'].tolist()
        intersection_node.append(node)
        intersection_areas.append(areas[np.argmax(counts)])
        intersection_communities.append(np.sum(counts))
    result = pd.DataFrame({
        '路口节点': intersection_node,
        '路口节点所属行政区': intersection_areas,
        '路口节点连接社区数': intersection_communities})
    result.to_csv('TempResult\路口连接行政区数据.csv', index=False)

def intersection_area_update(self):
    """本函数通过从上一个函数得出的结果，给所有路口节点赋予行政区的标签"""
    label_data = pd.read_csv('TempResult\路口连接行政区数据.csv')
    unlabel_data = pd.read_excel('Data\intersection_node.xlsx')
    train_data = pd.merge(left=unlabel_data,
                          right=label_data, how="right", left_on='节点编号',
                          right_on='路口节点')[['路口节点', '路口节点所属行政区', '路口节点连接社区数', '路口横坐标',
                          '路口纵坐标']]
    # train_data.to_csv('TempResult\路口训练集.csv', index=False)
    areas = list(set(train_data['路口节点所属行政区'].tolist()))
    areas_number = list(range(9))
    areas2number = dict(zip(areas, areas_number))
    number2areas = dict(zip(areas_number, areas))
    train_data['路口节点所属行政区'] = train_data['路口节点所属行政区'].apply(
        lambda x: areas2number[x])
    x_train = train_data[['路口横坐标', '路口纵坐标']]
    y_train = train_data['路口节点所属行政区']

```

```

    test_data = pd.merge(left=unlabel_data,
                         right=label_data, how="left", left_on='节点编号',
                         right_on='路口节点')[['节点编号', '路口横坐标', '路口纵坐标', '路口节点连接社区数']]
    test_data['路口节点连接社区数'] = test_data['路口节点连接社区数'].fillna(0)
    test_data = test_data[test_data['路口节点连接社区数'] == 0]
    knn = KNeighborsClassifier().fit(x_train, y_train)
    x_test = test_data[['路口横坐标', '路口纵坐标']]
    y_test_pre = knn.predict(x_test)
    test_data['路口节点所属行政区'] = y_test_pre
    train_data['节点编号'] = train_data['路口节点']
    train_data = train_data[['节点编号', '路口横坐标',
                            '路口纵坐标', '路口节点连接社区数', '路口节点所属行政
区']]
    res = pd.concat([train_data, test_data])
    res = res.sort_values(by='节点编号', ascending=True)
    res['路口节点所属行政区'] = res['路口节点所属行政区'].apply(
        lambda x: number2areas[x])
    res.to_csv('TempResult\路口行政区小区数据.csv', index=False)

def route_area(self):
    """本函数用于计算路线的行政区划,同时计算公路密度"""
    route_data = pd.read_excel('Data\intersection_route.xlsx')
    intersection_data = pd.read_csv('TempResult\路口行政区小区数据.csv')
    start_area = []
    end_area = []
    for i in range(route_data.shape[0]):
        start = route_data.iloc[i, 1]
        end = route_data.iloc[i, 2]
        start_area.append(intersection_data.iloc[start-1, 4])
        end_area.append(intersection_data.iloc[end-1, 4])
    route_data['起点行政区'] = start_area
    route_data['终点行政区'] = end_area
    # route_data.to_csv('TempResult\带行政区的公路线.csv', index=False)
    route_data = route_data[['路线距离(m)', '起点行政区']]
    area_route_length = route_data.groupby(
        ['起点行政区'], as_index=False)[['路线距离(m)']].sum()
    area_route_length.to_csv('TempResult\各行政区公路线总长度.csv', index=False)
    area_route_length.to_excel('TempResult\各行政区公路线总长度.xlsx',
                               index=False)

def draw_map_with_area(self):
    intersection_route = pd.read_csv('TempResult\带行政区的公路线.csv')
    intersection_node = self.intersection_node
    community = self.community
    routes = intersection_route[['路线起点', '路线终点', '起点行政区']]
    x_index = intersection_node['路口横坐标'].tolist()
    y_index = intersection_node['路口纵坐标'].tolist()
    area = list(community['所属区域'].value_counts().index)
    colors = ['red', 'yellow', 'black', 'green',
              'orange', 'purple', 'blueviolet', 'cyan', 'tan']
    area_color_dict = dict(zip(area, colors))
    plt.figure(figsize=(10, 10))
    for i in range(routes.shape[0]):
        try:
            start = routes.iloc[i, 0]-1

```

```

        end = routes.iloc[i, 1]-1
        area = routes.iloc[i, 2]
        color = area_color_dict[area]
        plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                    y_index[end]],
                  color=color, alpha=0.6, lw=0.2, zorder=0.2)
    except:
        print(start, end)
for i in range(community.shape[0]):
    x_index = community.iloc[i, 4]
    y_index = community.iloc[i, 5]
    area = community.iloc[i, 7]
    plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
plt.show()

def cluster(self):
    """计算该路口所连接的路线数"""
    route_data = pd.read_excel('Data\intersection_route.xlsx')
    G = nx.DiGraph()
    for i in range(route_data.shape[0]):
        start_idx = route_data.iloc[i, 1]
        end_idx = route_data.iloc[i, 2]
        weight = route_data.iloc[i, 3]
        G.add_edge(start_idx, end_idx, weight=weight)
    a = nx.pagerank(G)
    node_list = list(a.keys())
    pr_list = list()
    for i in node_list:
        pr_list.append(a[i])
    pr_list = np.array(pr_list)
    pr_list = (pr_list-np.min(pr_list))/(np.max(pr_list)-np.min(pr_list))
    aa = pd.DataFrame({
        '节点编号': node_list,
        'pr值': pr_list
    })
    b = dict(G.degree())
    node_list = list(b.keys())
    degree_list = list()
    for i in node_list:
        degree_list.append(b[i])
    degree_list = np.array(degree_list)
    degree_list = (degree_list-np.min(degree_list)) / \
        (np.max(degree_list)-np.min(degree_list))
    bb = pd.DataFrame({
        '节点编号': node_list,
        '度值': degree_list
    })
    res = pd.merge(aa, bb, on='节点编号', how="inner")
    left = pd.read_csv('TempResult\路口行政区小区数据.csv')
    cluster_data = pd.merge(left, res, on='节点编号', how='left')
    right = pd.read_csv('TempResult\社区路口连接数据.csv')
    right = right.groupby(['最近路口节点'], as_index=False)[['小区人口数
    (人)']].sum()
    right['节点编号'] = right['最近路口节点']
    right = right.drop(columns=['最近路口节点'])[['节点编号', '小区人口数(人)']]

```

```

cluster_data = pd.merge(cluster_data, right, on='节点编号', how='left')
cluster_data['小区人口数(人)'] = cluster_data['小区人口数(人)'].fillna(0)
temp = np.array(cluster_data['小区人口数(人)'])
cluster_data['小区人口数(人)'] = (
    temp-np.min(temp))/(np.max(temp)-np.min(temp))
temp = np.array(cluster_data['路口节点连接社区数'])
cluster_data['路口节点连接社区数'] = (
    temp-np.min(temp))/(np.max(temp)-np.min(temp))
train_data = cluster_data[['路口节点连接社区数', 'pr值', '度值', '小区人口数(人)']]
# k = list(range(2,10))
# pinggu =list()
# for ki in k:
#     kmeans = KMeans(n_clusters=ki, random_state=42)
#     y_pred = kmeans.fit_predict(train_data)
#     pinggu.append(kmeans.inertia_)
# plt.plot(k,pinggu)
kmeans = KMeans(n_clusters=5, random_state=42)
y_pred = kmeans.fit_predict(train_data)
cluster_data['节点类别'] = y_pred
label_translate = dict(
    zip([0, 1, 2, 3, 4], ['一般', '冷清', '较冷清', '繁忙', '较繁忙']))
cluster_data['节点类别'] = cluster_data['节点类别'].apply(
    lambda x: label_translate[x])
cluster_data.to_csv('TempResult\行政区统计数据.csv', index=False)

def compute_area_info(self):
    data = pd.read_csv('TempResult\行政区统计数据.csv')
    groups = data.groupby(['路口节点所属行政区'])
    area_list = list()
    mean_degree = list()
    busy_percent = list()
    for i in groups:
        area, group_data = i
        area_list.append(area)
        mean_degree.append(group_data['度值'].mean())
        length = group_data.shape[0]
        busy_number = group_data[group_data['节点类别'] == '繁忙'].shape[0] + \
            group_data[group_data['节点类别'] == '较繁忙'].shape[0]
        busy_percent.append(busy_number/length)
    res = pd.DataFrame({
        '区域名称': area_list,
        '平均度': mean_degree,
        '繁忙百分比': busy_percent
    })
    data = pd.read_excel('TempResult\行政区指标.xlsx')
    result = pd.merge(res, data, on='区域名称', how='inner')
    result['指标'] = result['生活物资投放点数量'] / \
        result['隔离人口数(万人)'] * result['公路距离'] / \
        result['行政面积'] * result['平均度']*result['繁忙百分比']
    result.to_csv('Result/first_result.csv', index=False)
    result.to_excel('Result/first_result.xlsx', index=False)

```

第2题后半部分:

```

from cnames import cnames
import pandas as pd
import gurobipy as gb
from numpy import linalg as LA
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt
import numpy as np
import os
plt.style.use('fivethirtyeight')
colors = list(cnames.keys())

```



```

class ProblemSolver:
    def __init__(self) -> None:
        self.xiaoQu = pd.read_excel('Data\xiaoQu.xlsx') # 读取小区的坐标和需求数据

    def run(self):
        sortcenter_number = self.get_number_of_sortingCenter()
        # self.draw_cluster_xiaoqu_data()
        return sortcenter_number

    def get_number_of_sortingCenter(self):
        """本函数用于对小区进行聚类从而确定配送中心的数量"""
        XiaoQu = pd.read_excel('Data\xiaoQu.xlsx')
        x_index = np.array(XiaoQu['横坐标'])
        y_index = np.array(XiaoQu['纵坐标'])
        train_data = pd.DataFrame({
            '横坐标': x_index,
            '纵坐标': y_index
        })
        if os.path.exists('Temp\xiaoQu_Cluster.csv'):
            XiaoQu = pd.read_csv('Temp\xiaoQu_Cluster.csv')
        else:
            for i in range(2, 100):
                clust = AgglomerativeClustering(
                    n_clusters=i, linkage='complete')
                y_pred = clust.fit_predict(train_data)
                train_data['节点类别'] = y_pred
                train_data['人口数'] = XiaoQu['小区人口数']
                cluster_population = np.array(
                    train_data.groupby(['节点类别'])['人口数'].sum().tolist())
                decide = all(cluster_population <= 100000)
                if decide:
                    break
            XiaoQu['节点类别'] = y_pred
            XiaoQu.to_csv('Temp\xiaoQu_Cluster.csv', index=False)
        return len(list(set(XiaoQu['节点类别'])))

```



```

    def draw_cluster_xiaoqu_data(self):
        """在确定配送中心数量后为地图上色"""
        XiaoQu = pd.read_csv('Temp\xiaoQu_Cluster.csv')
        sortcenter = list(set(XiaoQu['节点类别']))

```

```

color_dict = dict(zip(sortcenter, colors[:len(sortcenter)]))
x_index = np.array(XiaoQu['横坐标'])
y_index = np.array(XiaoQu['纵坐标'])
for i in range(xiaoQu.shape[0]):
    x_i = x_index[i]
    y_i = y_index[i]
    color_i = XiaoQu.iloc[i, 6]
    color_i = color_dict[color_i]
    plt.scatter(x_i, y_i, color=color_i, s=2)
plt.show()

def draw_result(self):
    sort_intersection = pd.read_csv('Result\物资分发中心坐标.csv')
    intersection_route = pd.read_excel('Data\intersection_route.xlsx')
    intersection_node = pd.read_excel('Data\intersection_node.xlsx')
    community = pd.read_excel('Data\community.xlsx')
    routes = intersection_route[['路线起点', '路线终点']]
    x_index = intersection_node['路口横坐标'].tolist()
    y_index = intersection_node['路口纵坐标'].tolist()
    area = list(community['所属区域'].value_counts().index)
    colors = ['red', 'yellow', 'black', 'green',
              'orange', 'purple', 'blueviolet', 'cyan', 'tan']
    area_color_dict = dict(zip(area, colors))
    plt.figure(figsize=(10, 10))
    for i in range(routes.shape[0]):
        try:
            start = routes.iloc[i, 0]-1
            end = routes.iloc[i, 1]-1
            plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                       y_index[end]],
                     color='b', alpha=0.6, lw=0.2, zorder=0.2)
        except:
            print(start, end)
    for i in range(sort_intersection.shape[0]):
        xi = sort_intersection.iloc[i, 0]
        yi = sort_intersection.iloc[i, 1]
        plt.scatter(xi, yi, marker='*', color='red', s=100)
    for i in range(community.shape[0]):
        x_index = community.iloc[i, 4]
        y_index = community.iloc[i, 5]
        area = community.iloc[i, 7]
        plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)

def optimize(self):
    number_sort = 71
    sorting_index = 100 * \
        np.random.RandomState(152).random(size=(number_sort, 2)) # 分拣中心的
坐标
    """计算路径并且确认"""
    chengben = list()
    for xun in range(20):
        xiaoqu = pd.read_csv('Temp\XiaoQu_Cluster.csv')
        p = xiaoqu['需求'].tolist()
        q = 0.2
        v = 100000 * 0.4

```

```

xiaoqu_index = np.array(xiaoqu[['横坐标', '纵坐标']])
distance = list()
for i in range(sorting_index.shape[0]):
    sorting_i = np.array([sorting_index[i]])
    res = LA.norm(np.array(
        [sorting_i[:, 0] - xiaoqu_index[:, 0], sorting_i[:, 1] -
        xiaoqu_index[:, 1]]), axis=0)
    distance.append(res.tolist())
distance = np.array(distance)*0.32 # 比例尺问题
model = gb.Model()
w = model.addVars(number_sort, 1409, lb=0.0, ub=40000,
                   vtype=gb.GRB.CONTINUOUS, name='w') # 添加变量
model.setObjective(gb.quicksum(w[i, j] * distance[i, j] * q for i in
range(number_sort)
                                for j in range(1409)),
gb.GRB.MINIMIZE) # 添加目标函数
model.addConstrs(w.sum('*', j) == p[j] for j in range(1409))
model.addConstrs(w.sum(i, '*') <= v for i in range(number_sort))
model.Params.LogToConsole = True # 显示求解过程
model.Params.TimeLimit = 100 # 限制求解时间为 100s
model.optimize()
cost = model.objVal
"""更新坐标"""
for i in range(number_sort):
    sorting_index[i, 0] = (sum(w[i, j].X * q * xiaoqu_index[j,
0])/distance[i, j] for j in range(
        1409))+0.0000001) / (sum(w[i, j].X * q / distance[i, j] for
j in range(1409))+0.0000001) + 0.1*np.random.RandomState(152).random()
    sorting_index[i, 1] = (sum(w[i, j].X * q * xiaoqu_index[j,
1])/distance[i, j] for j in range(
        1409))+0.0000001) / (sum(w[i, j].X * q / distance[i, j] for
j in range(1409))+0.0000001) + 0.1*np.random.RandomState(152).random()
    chengben.append(cost)
print("总花费为", cost)
all_renkou = pd.read_csv('Temp\\XiaoQu_Cluster.csv')['小区人口数'].tolist()
all_xuqiu = pd.read_csv('Temp\\XiaoQu_Cluster.csv')['需求'].tolist()
all_quyu = pd.read_csv('Temp\\XiaoQu_Cluster.csv')['所属区域'].tolist()
give_plan = list()
row_index = list()
column_index = list()
distance_plan = list()
Population = list()
area = list()
for i in range(number_sort):
    give = sum(w[i, j].X for j in range(1409))
    if give != 0:
        row_index.append(sorting_index[i, 0])
        column_index.append(sorting_index[i, 1])
        give_plan.append([w[i, j].X for j in range(1409)])
        distance_plan.append([distance[i, j] for j in range(1409)])
    give_plan = np.array(give_plan)
    distance_plan = np.array(distance_plan)
    xiaogueshu = list()
    xuanzhibanjing = list()
    for i in range(39):

```

```

count = 0
banjing = list()
renkou = list()
quyu = list()
for j in range(1409):
    give = give_plan[i, j]
    if give != 0:
        count += 1
        banjing.append(distance_plan[i, j])
        renkou.append(all_renkou[j]*give/all_xuqiu[j])
        quyu.append(all_quyu[j])
xiaoqugeshu.append(count)
xuanzhibanjing.append(max(banjing))
Population.append(int(sum(renkou)))
area.append(max(set(quyu), key=quyu.count))
result = pd.DataFrame({
    '横坐标': row_index,
    '纵坐标': column_index,
    '选址半径(km)': xuanzhibanjing,
    '辖区小区数': xiaoqugeshu,
    '辖区内人口数': Population,
    '所在区域': area
})
result.to_csv('Result/物资分发中心坐标.csv', index=False)
result.to_excel('Result/物资分发中心坐标.xlsx', index=False)

```

第3题:

```

import pandas as pd
xiaoqu = pd.read_excel('Data\小区.xlsx')
import matplotlib.pyplot as plt
plt.rcParams["font.sans-serif"] = ["SimHei"] # 设置字体
plt.rcParams["axes.unicode_minus"] = False # 正常显示负号

april_10 = pd.read_excel('Data/每天蔬菜包数据/4月10日.xlsx')
april_11 = pd.read_excel('Data/每天蔬菜包数据/4月11日.xlsx')
april_12 = pd.read_excel('Data/每天蔬菜包数据/4月12日.xlsx')
april_13 = pd.read_excel('Data/每天蔬菜包数据/4月13日.xlsx')
april_14 = pd.read_excel('Data/每天蔬菜包数据/4月14日.xlsx')
april_15 = pd.read_excel('Data/每天蔬菜包数据/4月15日.xlsx')

area_population = xiaoqu.groupby(['所属区域'], as_index=False)[[
    '小区人口数(人)', '小区户数(户)']].sum()
area_population.iloc[-1, 0] = '长春新区'
area_population['每天需求'] = 0.4*area_population['小区人口数(人)']/1000
area_population = pd.merge(area_population, april_10,
                           left_on='所属区域', right_on='行政区', how='inner')
demand = area_population['每天需求'].tolist()
buy = area_population['蔬菜包供给重量'].tolist()
give = area_population['蔬菜包发放重量'].tolist()
areas = area_population['所属区域'].tolist()
area_population['供给需求差'] = area_population['蔬菜包供给重量'] -
area_population['每天需求']

```

```

area_population['发放需求差'] = give = area_population['蔬菜包发放重量'] -
area_population['每天需求']
area_population.to_excel('Result/' + str('april_10') + '.xlsx', index=False)
# plt.figure(figsize=(14, 8))

# #设置刻度及刻度标签
# x_demand = list(range(len(areas)))
# x_buy = [i+0.2 for i in x_demand]
# x_give = [i + 0.4 for i in x_demand]

# plt.bar(x_demand, demand, width=0.2, label="当日需求量")
# plt.bar(x_buy, buy, width=0.2, label="当日采购量")
# plt.bar(x_give, give, width=0.2, label="当日分发量")
# plt.legend()
# plt.xticks(x_buy, areas, rotation=45)
# plt.show()

```

第4题:

```

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import networkx as nx
from numpy import linalg as LA
import os
from sklearn.cluster import KMeans
from sklearn.neighbors import KNeighborsClassifier
plt.style.use('fivethirtyeight')

class Solution:
    def __init__(self) -> None:
        self.community = pd.read_excel('Data/community.xlsx')
        self.intersection_node = pd.read_excel('Data/intersection_node.xlsx')
        self.intersection_route = pd.read_excel('Data/intersection_route.xlsx')

    def run(self):
        # self.draw_map() # 画出全市地图
        pass

    def build_graph(self):
        """本函数用于构建无向图"""
        community_route = self.community_neraest_intersection()
        intersection_route = self.intersection_route
        G = nx.Graph()
        intersection_edges = list()
        community_edges = list()
        for i in range(intersection_route.shape[0]):
            start = intersection_route.iloc[i, 1] # 路线起点
            end = intersection_route.iloc[i, 2]
            weight = intersection_route.iloc[i, 3]
            intersection_edges.append((start, end, weight))
        for i in range(community_route.shape[0]):

```

```

        start = community_route.iloc[i,0] # 路线起点
        end = community_route.iloc[i,8]
        weight = community_route.iloc[i,9]
        community_edges.append((c+str(start), i+str(end), weight))
G.add_weighted_edges_from(intersection_edges)
G.add_weighted_edges_from(community_edges)
return G

def community_neraest_intersection(self):
    """本函数用于计算每个小区最近的路口"""
    if os.path.exists('TempResult/社区路口连接数据.csv'):
        return pd.read_csv('TempResult/社区路口连接数据.csv')
    else:
        intersection = self.intersection_node
        community = self.community
        intersection_index = np.array(intersection[['路口横坐标','路口纵坐标']])
        community_index = np.array(community[['小区横坐标','小区纵坐标']])
        nearest_intersection = list()
        distance = list()
        for i in range(community_index.shape[0]):
            community_index_i = np.array([community_index[i]])
            res = LA.norm(np.array([community_index_i[:, 0] -
intersection_index[:, 0],
                           community_index_i[:, 1] -
intersection_index[:, 1]]), axis=0)
            nearest_intersection.append(np.argmin(res)+1)
            distance.append(np.min(res)* 320)
        community['最近路口节点'] = nearest_intersection
        community['路口路线距离'] = distance
        community.to_csv("TempResult/社区路口连接数据.csv", index=False)
        return community

def draw_map(self):
    intersection_route = self.intersection_route
    intersection_node = self.intersection_node
    community = self.community
    routes = intersection_route[['路线起点', '路线终点']]
    x_index = intersection_node['路口横坐标'].tolist()
    y_index = intersection_node['路口纵坐标'].tolist()
    area = list(community['所属区域'].value_counts().index)
    colors = ['red', 'yellow', 'black', 'green',
              'orange', 'purple', 'blueviolet', 'cyan', 'tan']
    area_color_dict = dict(zip(area, colors))
    plt.figure(figsize=(10, 10))
    for i in range(routes.shape[0]):
        try:
            start = routes.iloc[i, 0]-1
            end = routes.iloc[i, 1]-1

```

```

        plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                y_index[end]],
                  color='b', alpha=0.6, lw=0.2, zorder=0.2)
    except:
        print(start, end)
    for i in range(community.shape[0]):
        x_index = community.iloc[i, 4]
        y_index = community.iloc[i, 5]
        area = community.iloc[i, 7]
        plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
    plt.show()

def draw_solution(self):
    sort_intersection = pd.read_csv('无货车结果(集散中心).csv')['节点编号'].tolist()
    intersection_route = self.intersection_route
    intersection_node = self.intersection_node
    community = self.community
    routes = intersection_route[['路线起点', '路线终点']]
    x_index = intersection_node['路口横坐标'].tolist()
    y_index = intersection_node['路口纵坐标'].tolist()
    area = list(community['所属区域'].value_counts().index)
    colors = ['red', 'yellow', 'black', 'green',
              'orange', 'purple', 'blueviolet', 'cyan', 'tan']
    area_color_dict = dict(zip(area, colors))
    plt.figure(figsize=(10, 10))
    for i in range(routes.shape[0]):
        try:
            start = routes.iloc[i, 0]-1
            end = routes.iloc[i, 1]-1
            plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                       y_index[end]],
                      color='b', alpha=0.6, lw=0.2, zorder=0.2)
        except:
            print(start, end)
    for i in sort_intersection:
        xi = x_index[i-1]
        yi = y_index[i-1]
        plt.scatter(xi,yi,marker='*', color='red', s=100)
    for i in range(community.shape[0]):
        x_index = community.iloc[i, 4]
        y_index = community.iloc[i, 5]
        area = community.iloc[i, 7]
        plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
    cangku = pd.read_csv('无货车结果(仓库).csv')
    for i in range(cangku.shape[0]):
        x_index = cangku.iloc[i, 2]
        y_index = cangku.iloc[i, 3]
        plt.scatter(x_index, y_index, color='blue', marker='*', s=200)
    plt.show()

def draw_solution_car(self):
    sort_intersection = pd.read_csv('有货车结果(集散中心).csv')['节点编号'].tolist()
    intersection_route = self.intersection_route

```

```

intersection_node = self.intersection_node
community = self.community
routes = intersection_route[['路线起点', '路线终点']]
x_index = intersection_node['路口横坐标'].tolist()
y_index = intersection_node['路口纵坐标'].tolist()
area = list(community['所属区域'].value_counts().index)
colors = ['red', 'yellow', 'black', 'green',
          'orange', 'purple', 'blueviolet', 'cyan', 'tan']
area_color_dict = dict(zip(area, colors))
plt.figure(figsize=(10, 10))
for i in range(routes.shape[0]):
    try:
        start = routes.iloc[i, 0]-1
        end = routes.iloc[i, 1]-1
        plt.plot([x_index[start], x_index[end]], [y_index[start],
                                                    y_index[end]],
                 color='b', alpha=0.6, lw=0.2, zorder=0.2)
    except:
        print(start, end)
for i in sort_intersection:
    xi = x_index[i-1]
    yi = y_index[i-1]
    plt.scatter(xi, yi, marker='*', color='red', s=100)
for i in range(community.shape[0]):
    x_index = community.iloc[i, 4]
    y_index = community.iloc[i, 5]
    area = community.iloc[i, 7]
    plt.scatter(x_index, y_index, color=area_color_dict[area], s=2)
cangku = pd.read_csv('有货车结果(仓库).csv')
for i in range(cangku.shape[0]):
    x_index = cangku.iloc[i, 2]
    y_index = cangku.iloc[i, 3]
    plt.scatter(x_index, y_index, color='blue', marker='*', s=200)
plt.show()

```

有货车:

```

from Solution import *
solve = Solution()
G = solve.build_graph()
import pandas as pd
data = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection = list(set(data['最近路口节点']))
community = pd.read_excel('Data\community.xlsx')
demand = (community['小区人口数(人)']*0.4).tolist()
distance = list()
for i in range(len(intersection)):
    start = intersection[i]
    shortest_distance_dict = nx.shortest_path_length(G,
                                                       source='i'+str(start), weight='weight')
    distance.append([shortest_distance_dict['c'+str(i)] for i in
                    range(1, 1410)])
distance = np.array(distance)
import gurobipy as gb
intersection_length = len(intersection)

```

```

community_length=1409
model=gb.Model()
Q =
model.addVars(intersection_length,community_length,lb=0.0,ub=40000,vtype=gb.GRB.
CONTINUOUS, name='Q') # 添加变量
R = model.addVars(intersection_length,vtype=gb.GRB.BINARY,name='R') # 添加变量
B = model.addVars(intersection_length,vtype=gb.GRB.INTEGER,name='B',lb=0,ub=4)
L = model.addVars(intersection_length,vtype=gb.GRB.INTEGER,name='L',lb=0,ub=10)
print("变量添加完毕")
model.addConstrs(
    gb.quicksum(Q[k,l]*R[k] for k in range(intersection_length)) == demand[l]
for l in range(community_length)
)
print('约束1添加完毕')
model.addConstrs(
    gb.quicksum(Q[k,l] for l in range(community_length)) <= 40000 for k in
range(intersection_length)
)
print("约束2添加完毕")
model.addConstr(
    gb.quicksum(R[k] for k in range(intersection_length)) <= 71
)
print("约束3添加完毕")
model.addConstrs(
    B[k]*10000 + L[k]*4000 - gb.quicksum(Q[k,l] for l in
range(community_length)) >= 0 for k in range(intersection_length)
)
print("约束4添加完毕")

model.setObjective(
    gb.quicksum(
        Q[k,l] * distance[k,l] * R[k] for l in range(community_length) for k in
range(intersection_length)
    ) + 2*gb.quicksum(
        R[k]*B[k] for k in range(intersection_length)
    ) + gb.quicksum(R[k]*L[k] for k in range(intersection_length)),
gb.GRB.MINIMIZE
) # 添加目标函数
print("目标函数设置完毕")
model.Params.TimeLimit=3600
model.optimize()
intersection_node = list()
intersection_x = list()
intersection_y = list()
for i in range(len(intersection)):
    if round(R[i].X) != 0:
        intersection_node.append(intersection[i])

res = pd.DataFrame({'intersection_node':intersection_node})
res.to_csv('有货车.csv',index=False)
old_data = pd.read_excel('Data\intersection_node.xlsx')
intersection_node = pd.read_csv('有货车.csv')[['intersection_node']]
intersection_node =
pd.merge(intersection_node,old_data,left_on='intersection_node',right_on='节点编
号',

```

```

                how='left')[['节点编号','路口横坐标','路口纵坐标']]
areas = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection_node = pd.merge(intersection_node,areas, left_on='节点编号',right_on='最近路口节点',how='left')
intersection_node = intersection_node[['节点编号','路口横坐标','路口纵坐标','所属区域']]
intersection_node.drop_duplicates(subset=['节点编号'], keep='first', inplace=True)
intersection_node.reset_index(drop=True, inplace=True)
intersection_node.to_csv('有货车结果(集散中心).csv',index=False)
intersection_node.to_excel('有货车结果(集散中心).xlsx',index=False)
groups = intersection_node.groupby(['所属区域'])
areas=list()
center_x = list()
center_y = list()
for i in groups:
    ar,group_data = i
    areas.append(ar)
    center_x.append(group_data['路口横坐标'].mean())
    center_y.append(group_data['路口纵坐标'].mean())
cangku = pd.DataFrame({
    '节点编号':[i+1 for i in range(len(areas))],
    '区域':areas,
    '仓库中心横坐标':center_x,
    '仓库中心纵坐标':center_y
})
cangku.to_csv('有货车结果(仓库).csv',index=False)
cangku.to_excel('有货车结果(仓库).xlsx',index=False)
from Solution import *
solve = Solution()
solve.draw_solution_car()

```

没有货车:

```

from Solution import *
solve = Solution()
G = solve.build_graph()
import pandas as pd
data = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection = list(set(data['最近路口节点']))
community = pd.read_excel('Data\community.xlsx')
demand = (community['小区人口数(人)']*0.4).tolist()
distance = list()
for i in range(len(intersection)):
    start = intersection[i]
    shortest_distance_dict = nx.shortest_path_length(G,
    source='i'+str(start),weight='weight')
    distance.append([shortest_distance_dict['c'+ str(i)] for i in
    range(1,1410)])
distance = np.array(distance)
import gurobipy as gb
intersection_length=len(intersection)
community_length=1409
model=gb.Model()

```

```

Q =
model.addVars(intersection_length,community_length,lb=0.0,ub=40000,vtype=gb.GRB.
CONTINUOUS, name='Q') # 添加变量
R = model.addVars(intersection_length,vtype=gb.GRB.BINARY,name='R')
print("变量添加完毕")
model.addConstrs(
    gb.quicksum(Q[k,l]*R[k] for k in range(intersection_length)) == demand[l]
for l in range(community_length)
)
print('约束1添加完毕')
model.addConstrs(
    gb.quicksum(Q[k,l] for l in range(community_length)) <= 40000 for k in
range(intersection_length)
)
print("约束2添加完毕")
model.addConstr(
    gb.quicksum(R[k] for k in range(intersection_length)) <= 71
)

model.setObjective(
    gb.quicksum(
        Q[k,l] * distance[k,l] * R[k] for l in range(community_length) for k in
range(intersection_length)
    ), gb.GRB.MINIMIZE
) # 添加目标函数
print("目标函数设置完毕")
model.Params.TimeLimit=3600
model.optimize()
intersection_node = list()
intersection_x = list()
intersection_y = list()
for i in range(len(intersection)):
    if round(R[i].X) != 0:
        intersection_node.append(intersection[i])
res = pd.DataFrame({'intersection_node':intersection_node})
res.to_csv('无货车.csv',index=False)
import pandas as pd
old_data = pd.read_excel('Data\intersection_node.xlsx')
intersection_node = pd.read_csv('无货车.csv')[['intersection_node']]
intersection_node =
pd.merge(intersection_node,old_data,left_on='intersection_node',right_on='节点编
号',
           how='inner')[['节点编号','路口横坐标','路口纵坐标']]
areas = pd.read_csv('TempResult\社区路口连接数据.csv')
intersection_node = pd.merge(intersection_node,areas,left_on='节点编
号',right_on='最近路口节点',how='inner')
intersection_node = intersection_node[['节点编号','路口横坐标','路口纵坐标','所属区
域']]
intersection_node.drop_duplicates(subset=['节点编号'], keep='first', inplace=True)
intersection_node.reset_index(drop=True, inplace=True)
intersection_node.to_csv('无货车结果(集散中心).csv',index=False)
intersection_node.to_excel('无货车结果(集散中心).xlsx',index=False)
groups = intersection_node.groupby(['所属区域'])
areas=list()
center_x = list()

```

```
center_y = list()
for i in groups:
    ar,group_data = i
    areas.append(ar)
    center_x.append(group_data['路口横坐标'].mean())
    center_y.append(group_data['路口纵坐标'].mean())
cangku = pd.DataFrame({
    '节点编号':[i+1 for i in range(len(areas))],
    '区域':areas,
    '仓库中心横坐标':center_x,
    '仓库中心纵坐标':center_y
})
cangku.to_csv('无货车结果(仓库).csv',index=False)
cangku.to_excel('无货车结果(仓库).xlsx',index=False)
from Solution import *
solve = solution()
solve.draw_solution()
loss =
[17281,16928,66,12,10.5,10.3,9.43,8.83,8.58,8.21,7.72,5.82,5.04,3.06,1.14]
plt.plot(loss)
```