基于禁忌搜索和多源最短路径算法的最小权路径覆盖策略

摘要

医疗物资的调度直接影响着患者的生命和健康，高效的调度能够确保患者及时获得必需的医疗资源，提高救治效率，减少疾病蔓延风险。本文通过建立基于禁忌算法的禁忌路径寻优模型，设计了多辆车运送医疗物品给多家医院的最短路径方案。

对于问题一，首先，由于医院之间可直达，故医院节点构成**完全图拓扑**，即至少存在一条**哈密顿回路**。然后，利用**禁忌算法**构建**禁忌路径寻优模型**，最终货车的最短路程巡航方案为：$\mathbf{5 \to 11 \to 14 \to 2 \to 12 \to 4 \to 6 \to 3 \to 7 \to 9 \to 8 \to 1 \to 10 \to 13 \to 5}$；或颠倒其顺序：$\mathbf{5 \to 13 \to 10 \to 1 \to 8 \to 9 \to 7 \to 3 \to 6 \to 4 \to 12 \to 2 \to 14 \to 11 \to 5}$，此时总路程为 **131.5864**, 由于每种最优路径都存在节点相同，顺序颠倒的两种方案，以下仅呈现其中一种方案。

对于问题二，首先，引入**凸包区域划分模型**将 14 个节点分割为上下两个凸包半区。然后，计算每个半区的载货需求量，剔除掉**不满足车辆均衡率**的方案。接着，针对每一个半区，利用禁忌路径寻优模型，得出各个半区内的最短路径。最后，建立**满足均衡率的禁忌路径寻优模型**，选择总路程最短的分割方案：**通过连接节点 5 与节点 2 分割 14 个节点**。其中，**下半区货车的巡航路径** $\mathbf{5 \to 13 \to 10 \to 1 \to 8 \to 9 \to 7 \to 4 \to 5}$；**上半区货车的巡航路径**为：$\mathbf{5 \to 2 \to 3 \to 6 \to 12 \to 14 \to 11 \to 5}$。此时最短路程为：**166.5599**。

对于问题三，首先利用 $Floyd-Warshall$ **算法**更新每个点到其它点的最短路程。然后，利用**深度优先算法**将 14 个节点组成的图分割成哈密顿回路和导出子图两部分。接着，在每个导出子图中添加 5 号节点作为起止点，利用问题一中的禁忌路径寻优模型得出最短巡航路径长度。最后建立**共起点禁忌路径寻优模型**，综合所有哈密顿回路和对应的导出子图得到最优路径规划方案：**哈密顿回路中货车的巡航路线** $\mathbf{5 \to 6 \to 4 \to 3 \to 2 \to 5}$；**在剩余节点中货车的巡航路线** $\mathbf{5 \to 7 \to 8 \to 9 \to 10 \to 14 \to 13 \to 12 \to 11 \to 1 \to 5}$。此时最短路程为：**285.4271**。

对于问题四，利用问题三得到的哈密顿回路集与导出子图集。对导出子图集直接利用禁忌路径寻优模型得出最短巡航路径长度，构建**可变起点禁忌路径寻优模型**，得到最优路径规划方案：**哈密顿回路中货车的巡航路线** $\mathbf{5 \to 2 \to 3 \to 4 \to 6 \to 5}$；**在导出子图中货车的巡航路线**：$\mathbf{1 \to 7 \to 8 \to 9 \to 10 \to 14 \to 13 \to 12 \to 11 \to 1}$。此时的最短路程为：**250.3326**。

本文综合根据哈密顿回路，$Floyd-Warshall$ 算法，禁忌算法，深度优先算法，结合 MATLAB,C++,Excel，对最小权路径覆盖问题进行了多角度分析。为验证模型的稳定性，对问题四中医院节点对之间的道路连通情况进行扰动，求出扰动时两辆货车最优巡航路径的总路程，与问题四中答案进行对比。本文最后对所建模型进行了客观评价，并提出改进意见。

关键词: 哈密顿回路　禁忌搜索算法　$Floyd-Warshall$ 算法　深度优先搜索

一、 问题重述

1.1 背景资料

物流行业的持续发展正逐渐凸显出人力劳动在处理海量订单方面的效率劣势。在这种背景下，越来越多的企业开始转向采用自动化设备，如传送带和机器人，来优化作业流程。特别是在面对重大疫情爆发时，医疗物资的调度成为一项至关重要且紧迫的任务，对物流运营提出了极大的挑战。

1.2 需要解决的问题

某城市有 14 个医院，他们在地理信息系统中的坐标见附件，其中在节点 5 的仓库中存放着医疗物资，各个物资仓和医院的散点分布如图 1 所示，同时，附件中也存放了每个医院的日需求量和坐标位置。以此回答下列问题:

（1）假设任两节点之间都可以直线到达，既不考虑道路和城市布局的限制，设计一个路径规划方法，使一辆能载重 800 吨的货车从仓库出发，单次经过所有医院后返回仓库，驶过的总路程最小。

（2）基于问题（1），由原来的一辆载重 800 吨的车运送变成两辆载重 500 吨的车，设计一个路径规划方法，使得两辆车同时从仓库出发到返回仓库的总路程最小。

（3）现实中，医院之间显然不是都可以两两直达的，附件中列出了各医院间存在的道路，基于图 2，重新设计路径规划方案，让两辆载重 500 吨的货车同时从仓库出发并最终返回仓库的总路程最短。

（4）基于问题（3），进一步选定另一家医院作为第二仓库，以使两辆能载重 500 吨的货车从这两个仓库出发，分别完成周游并返回各自的起点的总路程最短。

二、 模型假设

1. 图是强连通。
2. 货车在运输过程中道路的连通情况不会改变。
3. 对于作为仓库的医院节点来说，其自身的需求不需要货车运载满足。

三、 符号说明

符号	符号说明
$G=(V,E)$	医院节点拓扑图
C_{ij}	i 号医院节点与 j 号医院节点间的欧式距离
D_{ij}	i 号医院节点与 j 号医院节点间的最短路径长度
R_i	i 号医院的物资需求量
$\boldsymbol{v_i}$	第 i 号医院节点的坐标向量
$Degree_i$	第 i 号医院节点的度
Ψ_i	第 i 号车巡航长度
L_i	第 i 号车的装载量
π_i	巡航路径上的第 i 号节点的节点编号
l	$\mathbb{R}^2$ 上线性函数

四、 模型建立与求解

4.1 问题一

4.1.1 问题分析

因医院需求总量（762 吨）不超越货车载重（800 吨），问题一实质为不带容量制约的 TSP 问题。在该问题中，要使货车从仓库出发，经过所有医院，并最终回到仓库，同时保持总路程最短。鉴于医院之间可直达，故医院节点构成完全图拓扑。基于哈密顿回路性质，至少可确保存在一条连通所有节点的哈密顿回路。因此，设计最佳路径方案，即寻找最小的哈密顿回路。而由于禁忌搜索算法是通过维护一个禁忌表来避免在搜索过程中陷入局部最优解，并且在一定程度上能够跳出局部最优解，寻找全局最优解的算法，所以可适用于发现最短的哈密顿回路。

4.1.2 模型建立

（1）每种路径方案均为哈密顿回路

通过图中所有节点一次且仅一次的回路称为哈密顿回路。由哈密顿图的性质可知，若 $G=(V,E)$ 是包含 $n(n\geq 3)$ 个点的无向简单图，若对于 G 中任意不相邻的顶点 v_i，

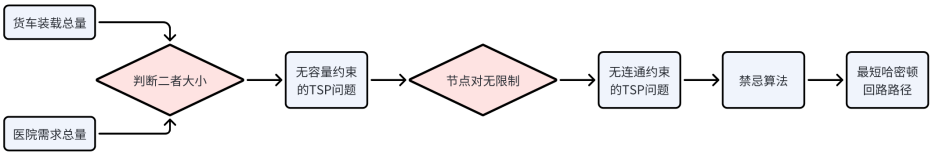

图 1　问题一流程图

v_j，它们的度均满足 $Degree_{v_i} + Degree_{v_j} \geq n$，则 G 中存在哈密顿回路，从而 G 为哈密顿图。

而由题可知医院节点之间两两直达，则任意医院节点 i，其度均满足 $Degree_{v_i} = n-1$，因此其中每两个节点都满足 $Degree_{v_i} + Degree_{v_j} = 2Degree_{v_i} = 2(n-1) > n$。故医院节点拓扑是哈密顿图，存在哈密顿回路。因此，设计最优路径方案即为找出最小哈密顿回路。

（2）所有路径方案的解空间

首先，所有路径方案都是以位于节点 5 的仓库为起点和终点的。若以 $\pi_i = j$ 表示货车第 i 次停靠在第 j 个节点，$C_{\pi_i\pi_{i+1}}$ 表示该路径方案中相邻两个医院之间的距离。由于一共有 14 个医院（节点），所以 $\pi_1 = \pi_{15} = 5$，解空间 S 可表为 $S = \{(\pi_1, \pi_2, \cdots, \pi_{15}) | (\pi_1, \pi_2, \cdots, \pi_{15})为1, 2, \cdots, 15的循环排列\}$，其中每一个循环排列表示经过 14 个目标的一个回路。

因此，在解空间内所有回路中选出总距离最短的，即为最优路径方案：

$$minf(\pi_1, \pi_2, \cdots, \pi_{15}) = \sum_{i=1}^{14} C_{\pi_i\pi_{i+1}} \tag{1}$$

（3）解空间候选集合的选取对于路径：

$$\pi_1 \cdots \pi_{u-1}\pi_u \cdots \pi_{v-1}\pi_v\pi_{v+1} \cdots \pi_{15}$$

交换 u 和 $v(1 < u < v < 15)$ 之间的顺序，得到新的路径为：

$$\pi_1 \cdots \pi_{u-1}\pi_v\pi_{v+1} \cdots \pi_{u+1}\pi_u\pi_{v+1} \cdots \pi_{15}$$

该新路径称为原路径的二领域的一个邻居，则当前解空间的所有邻居一共有 $C_{14}^2 = 91$ 个，这 91 个邻居组成的集合即为候选集合。

（4）候选集合禁忌表的制定

禁忌长度 t 取为二领域中邻居总数的开方，即：

$$t = \sqrt{C_{14}^2} \approx 10 \tag{2}$$

将候选集合禁忌表设计为一个循环队列，初始化禁忌表 $H = \phi$。从候选集合 C 中选择一种路径方案 $\boldsymbol{x}$，如果 $\boldsymbol{x} \notin H$，并且 H 不满，则把 $\boldsymbol{x}$ 添加进禁忌表中；若 H 已满，则最早进入禁忌表的路径出列，$\boldsymbol{x}$ 入列。

（5）所选路径评价函数的选取

由于候选集合中，每一个新路径对比原路径只有两条边发生了变化，因此可将这两边变化前后的值做差作为所选路径的评价函数，从而极大地提高算法的效率[2]

$$\triangle f = (C_{\pi_{u-1}\pi_v} + C_{\pi_u\pi_{v+1}}) - (C_{\pi_{u-1}\pi_u} + C_{\pi_v\pi_{v+1}}) \tag{3}$$

当 $\triangle f < 0$ 时，即说明该路径比原路径更短，可以放入禁忌表中。

（6）禁忌路径寻优模型的建立

综合式(1)和(3)，可建立禁忌路径寻优模型如式（5）：

$$\begin{cases} \min f(\pi_1, \pi_2, \cdots, \pi_{15}) = \sum_{i=1}^{14} C_{\pi_i\pi_{i+1}} \\ \triangle f = (C_{\pi_{u-1}\pi_v} + C_{\pi_u\pi_{v+1}}) - (C_{\pi_{u-1}\pi_u} + C_{\pi_v\pi_{v+1}}) \end{cases} \tag{4}$$

同时，为更清晰地描述禁忌算法在本题中的运用，算法执行的具体步骤将通过伪代码的形式来呈现：

Algorithm 1 禁忌搜索路径优化算法

procedure TabuSearch($\pi_{初始}, t_{最大}$)
    初始化禁忌表 H，大小为 t
    将当前解 $\pi_{当前}$ 设置为 $\pi_{初始}$
    将最优解 $\pi_{最优}$ 设置为 $\pi_{当前}$
    初始化迭代计数器 $t_{迭代}$ 为 0
    while $t_{迭代} < t_{最大}$ **do**
        从 $\pi_{当前}$ 生成候选解，通过交换边
        使用公式（5）计算每个候选解的 $\triangle f$
        选择最小 $\triangle f$ 的候选解，且不在 H 中
        更新禁忌表 H，加入选择的候选解
        更新当前解 $\pi_{当前}$ 为选择的候选解
        如果 $\pi_{当前}$ 优于 $\pi_{最优}$，则更新 $\pi_{最优}$
        增加 $t_{迭代}$ 的计数
        if H 已满 **then**
            从 H 中移除最旧的条目
        end if
    end while
    return $\pi_{最优}$
end procedure

4.1.3 问题解答

首先，根据附件中给出的每个医院的的坐标位置，由式（4）可求出 i 号医院与 j 号医院之间的距离 C_{ij}：

$$C_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}, i, j \in 1, 2 \cdots 14 \tag{5}$$

由于篇幅有限，表一仅展示部分医院间的距离:

表 1　各医院之间的距离

医院	1	2	···	13	14
1	0	25.22402622	···	32.81954013	31.50332598
2	25.22402622	0	···	18.08638968	8.751908488
⋮	⋮	⋮		⋮	⋮
13	32.81954013	18.08638968	···	0	11.64582616
14	31.50332598	8.751908488	···	11.64582616	0

于是，可以得到相邻医院的距离：$C_{\pi_i \pi_{i+1}}$。

然后，从候选集合 C 中随机选择一个初始解作为起始路径，并用此来初始化禁忌表 H。接着，对当前路径边交换以产生邻居。使用公式（3）对比新路径与原路径的路程总长度。如果新路径更小，且该路径不在禁忌表 H 内，则更新当前解为这个邻居，并将其加入 H。此过程迭代 100 次后，可以得到一个最短路径巡航方案。该巡航方案用其节点序列来表示如下：$5 \to 11 \to 14 \to 2 \to 12 \to 4 \to 6 \to 3 \to 7 \to 9 \to 8 \to 1 \to 10 \to 13 \to 5$ 或颠倒其顺序：$5 \to 13 \to 10 \to 1 \to 8 \to 9 \to 7 \to 3 \to 6 \to 4 \to 12 \to 2 \to 14 \to 11 \to 5$。此时总路程为 131.5864。具体的路径如图 2:

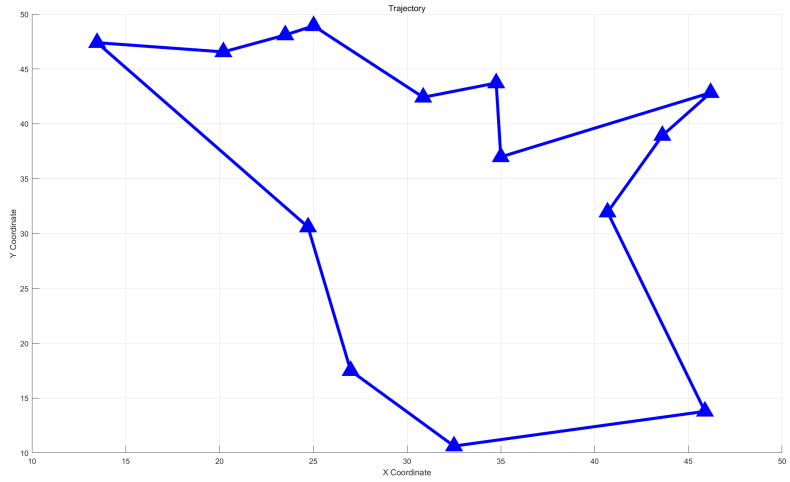

图 2　最优路径

4.2 问题二

4.2.1 问题分析

在问题一的基础上，问题二要求在有容量约束的情况下，重新设计路径规划方案，以实现两辆货车的总行驶距离最小化。由于增加了一辆车，因此必须将原有的 14 个节点划分为两个子区域，以供两辆货车分别承载医疗物资。为此，可以引入凸包区域划分模型，将这 14 个节点分为上下两个凸包半区，从而计算出各半区的总载货需求。同时引入车辆均衡率的概念，只有满足车辆均衡率的分割方案，才值得被考虑。

在分割方案确定后，针对每个半区，通过问题一的禁忌路径寻优模型模型，求出各半区内的最短路径。当两个半区内的路径规划均达到最短时，这两部分路径的结合必然构成了整体的最短路径。

最后，建立满足均衡率的禁忌路径寻优模型，对所有满足车辆均衡率的分割方案进行综合比较，以获取各分割方案中路径方案的总行驶距离, 其中总距离最短的路径方案即为最终的路径规划方案。

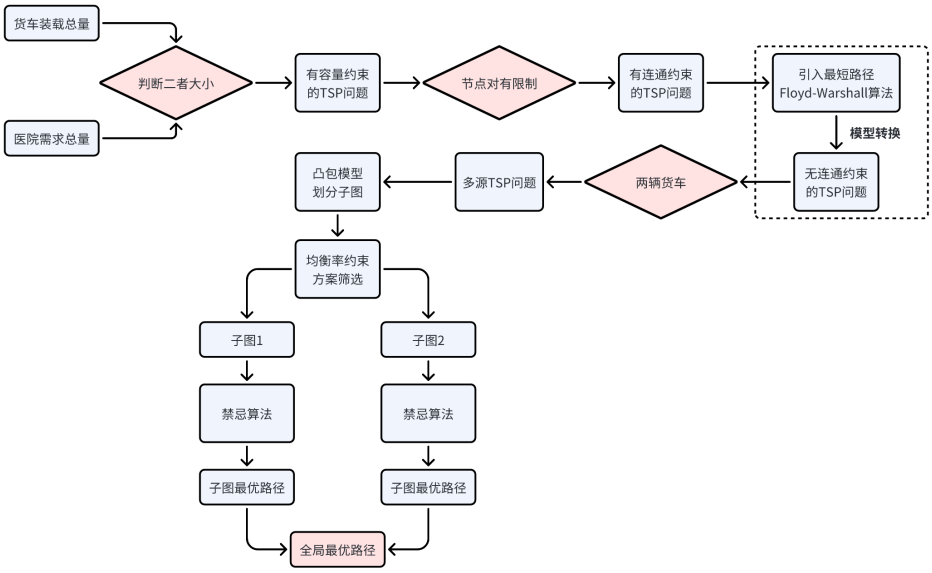

图 3　问题二流程图

4.2.2 模型建立

（1）凸包区域划分模型的建立

①将 14 个点分成上下两个半区的分割方法

考虑总共存在 14 个节点和两辆车，所以需要将这 14 个节点分割成两个均以第五个节点作为起始和终止点的半区，以便两辆车可以分别承担送货的任务。

记第 i 个节点的坐标位置为 $\boldsymbol{v_i} = (x_i, y_i)$，以 $l_{5,i}(i = 1, 2, \cdots 14, i \neq 5)$ 表示第 5 个节点与其他 13 个节点相连的所有直线。

考虑在二维实数空间 $\mathbb{R}^2$ 上的分割，由直线 $[l_{5i}:0]$ 定义，将平面划分为两个部分 (W_1, W_2)（直线上的两个点被归入 W_1）。对于其他 12 个点，必定位于 W_1 或 W_2 之中。我们记 W_1,W_2 中医院节点的集合为 V_1,V_2。由此，这 14 个点被有效地分成了上下两个半区[1]

②证明最优路径方案就在这样的分割方法中

V_1 中点 $\boldsymbol{v_1}, \boldsymbol{v_2}, ..., \boldsymbol{v_{n_1}}$ 的一个凸组合是如下形式的线性组合：

$$p_1\boldsymbol{v_1} + p_2\boldsymbol{v_2} + ... + p_{n_1}\boldsymbol{v_{n_1}}$$
$$st.\begin{cases} p_1 + p_2 + ... + p_{n_1} = 1 \\ p_i \geq 0, \forall i \end{cases}, k \neq 5 \tag{6}$$

同理可用 $\boldsymbol{q} = (q_1, q_2, ..., q_{n_2})$ 表示 V_2 中点的线性组合。则凸包 $convV_1$,$convV_2$ 为 V_1,V_2 中的凸组合的集合。即证明：$[l:0]$ 是最优分割的必要条件是 $l(\boldsymbol{v_5} = 0)$ 分割函数 l 生成的点集 V_1,V_2 满足 $convV_1 \cap convV_2 = \phi$. 以下通过反证法证明。

（a）直线分割线是最优分割方案

若最佳分割情况下 $convV_1 \cap convV_2 \neq \phi$，则存在点 $\boldsymbol{v_c} \in convV_1 \cap convV_2 = V_c$ 为 $convV_1$ 或 $convV_2$ 的一个最小代表元，由凸包的性质可知代表元必为凸包的顶点。又由于 $V_1 \cap V_2 = \phi$，则该点不可能同时为 $convV_1$ 和 $convV_2$ 的代表元，不妨设 $\boldsymbol{v_c}$ 为 $convV_1$ 的顶点。由于每一个点只需要访问一次，因此该点在两条巡航路径中一共出现一次。而该点作为一顶点出现在 V_1 的生成路径中，会增加 $convV_1$ 的直径，反之出现在在 V_2 的生成路径中时则不会增加 $convV_2$ 的直径。因此 $\boldsymbol{v_c}$ 应当出现在 V_2 的生成路径中。从 $convV_1$ 中删除 $\boldsymbol{v_c}$ 后。$convV_1$ 的形态发生改变，面积缩小，因此原分割不是最佳分割。

（b）5 号医院节点一定位于分割线上

若 5 号医院节点不位于分割线上，不妨设 V_5 位于 $convV_1$ 中，由于车辆一定从 5 号医院节点出发，因此 V_2 的生成路径一定包含 5 号医院节点。令 $V_2' = V_2 \cup 5$ 以此构造 $convV_2'$，则 V_5 一定为 $convV_2'$ 的一个顶点。由 (a) 可知当前方案非最优方案。

③凸包区域划分模型的构造

综上，可确定凸包区域划分模型如式（7）所示：

$$\boldsymbol{v_i} \in \begin{cases} V_1 & l_{5i}(x_i) \leq 0 \\ V_2 & l_{5i}(x_i) > 0 \end{cases}, \quad i = 1, 2, ..., 14 \tag{7}$$

(2) 满足均衡率的禁忌路径寻优模型的建立

第 k 辆车巡航长度为：

$$\psi_k = \sum_{i=1}^{n_k} C_{\pi_{k,i}\pi_{k,i+1}} 其中 k = 1, 2, \pi_{k,1} = \pi_{k,n_k} = 5 \tag{8}$$

其中 n_k 表示第 k 辆车经过的节点总数，满足:

$$\sum_{k=1}^{2}(n_k - 2) + 1 = 14 \tag{9}$$

记第 k 辆车经过的所有节点（除了 5 号医院节点）组成的集合为 V_k，则:

$$\begin{cases} \cap_{k=1}^{2} V_k = \phi \\ \cup_{k=1}^{2} V_k = V \end{cases} \tag{10}$$

因为每辆车经过的节点的需求量之和就是第 k 辆车的装载量，即:

$$L_k = \sum_{i \in V_k} R_i, k = 1, 2 \tag{11}$$

其中 R_i 表示第 i 个节点的物资需求量。

而从实际考虑角度出发，为了最大程度地满足需求，每辆车应当被合理规划，以便使其在路径设计中尽可能多地涵盖医疗资源点，而不仅限于其中一两个医院，从而实现资源分配的高效利用。所以，可定义车辆的装载均衡率 η 如式（11）:

$$\begin{cases} \eta = \dfrac{\min\limits_{k=1}^{2} L_k}{\max\limits_{k=1}^{2} L_k} \geq \dfrac{\sum\limits_{i=1}^{14} R_i - 500}{500} \\ L_k \leq 500 \end{cases} \tag{12}$$

所以，在两辆车所载物品重量相差过大时，可排除这种装载组合。

在所有装载组合中选择总路程最短的，即为最优路径。以此，可建立满足均衡率的禁忌路径寻优模型如下:

$$\min f = \min \sum_{k=1}^{2} \psi_k = \min \sum_{k=1}^{2} \sum_{i=1}^{n_k} C_{\pi_{k,i} \pi_{k,i+1}} \tag{13}$$

4.2.3 问题解答

（1）依据车辆均衡率将 14 个点分成两个部分

首先，将第 5 个节点与其他节点连线将图分为两部分，其次根据每个部分医院的总需求，依据式（11），可得到所有分割方案下的车辆均衡率（如表 2 所示）。

表 2　每种分割方法的车辆均衡率

分割方案	具体分割方法	上半部分需求	下半部分需求	均衡率%
1	通过连接节点 5 与节点 1 来分割	650	42	6.46
2	通过连接节点 5 与节点 2 来分割	420	272	64.76
⋮	⋮	⋮	⋮	⋮
13	通过连接节点 5 与节点 13 来分割	91	601	15.14
14	通过连接节点 5 与节点 14 来分割	55	637	8.63

在剔除所有不满足车辆装载率（即均衡率不大于 52.8%）的分割方案后，仅有以下四种分割方案被保留：

方案 2: 通过连接节点 5 与节点 2 来分割

方案 4: 通过连接节点 5 与节点 4 来分割

方案 7: 通过连接节点 5 与节点 7 来分割

方案 8: 通过连接节点 5 与节点 8 来分割

每种分割方法都能把 14 个点分成满足车辆均衡率的两部分。

（2）找出每种分割方法中的最短路径

对每种分割方法里的每一部分，都可以随机生成一个路径作为第一问建立的基于禁忌搜索的路径优化模型的初始解，放入禁忌表 H 中，然后通过交换边得到邻居，使用公式（3）对比新路径与原路径的路程总长度。如果新路径更小，且该路径不在禁忌表 H 内，则更新当前解为这个邻居，并将其加入 H。此过程迭代 100 次后，可以得到这 4 种分割方案的每一部分的最短路径如表 3：

表 3　4 种分割方案中各部分的最短路径

分割方案	上半部分的最短路径	下半部分的最短路径
2	5→ 13 → 10 → 1 → 8 → 9 → 7 → 4 → 5	5 → 2 → 3 → 6 → 12 → 14 → 11 → 5
4	5 → 7 → 9 → 8 → 1 → 10 → 13 → 5	5 → 11 → 14 → 12 → 6 → 3 → 4 → 2 → 5
7	5 → 9 → 8 → 1 → 10 → 13 → 5	5 → 11 → 14 → 12 → 6 → 3 → 7 → 4 → 2 → 5
8	5 → 2 → 4 → 7 → 9 → 3 → 6 → 12 → 14 → 11 → 5	5 → 10 → 8 → 1 → 13 → 5

（3）最优路径的求解

由（2）可得，这 4 种分割方法的最短路径的总路程见表 4：

表 4　4 种分割方案的最短路径总路程

分割方案	分割方法	最短路径总路程
2	通过连接节点 5 与节点 2 来分割	164.1804
4	通过连接节点 5 与节点 4 来分割	166.5599
7	通过连接节点 5 与节点 7 来分割	170.4825
8	通过连接节点 5 与节点 8 来分割	188.9968

由表 4 可得，总路程最短的是分割方案 2，即通过连接节点 5 和节点 2 来分割。

通过连接节点 5 和节点 2 分割后，上下两个半区货车的最优巡航方案路径轨迹如图 4 所示：

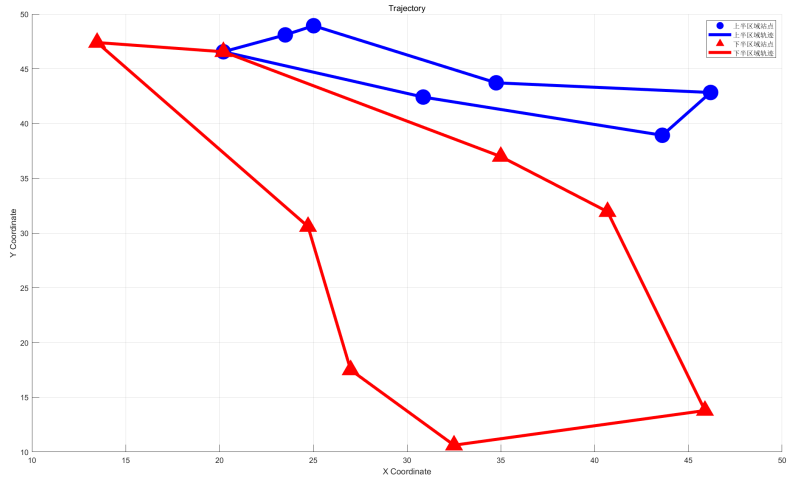

图 4　连接节点 5 和节点 2 来分割的最优路径

其中，蓝色为上半区域医院站点和货车最优轨迹；红色为下半区域医院站点和货车最优轨迹。

因此，最优路径规划的划分方法为：

通过连接节点 5 与节点 2 将 14 个节点分割为上下两部分。

货车的最优路径巡航方案为：

下半部分货车的巡航路线为： $5 \to 13 \to 10 \to 1 \to 8 \to 9 \to 7 \to 4 \to 5$；

上半部分货车的巡航路线为： $5 \to 2 \to 3 \to 6 \to 12 \to 14 \to 11 \to 5$ 。

此时最短路程为：166.5599。

在具体运货方案中，可以任意选择车辆的负责区域，对车辆编号没有要求。

4.3 问题三

4.3.1 问题分析

针对问题三，需要考虑到道路连通性限制和容量约束，重新设计路径规划方案以最小化两辆货车从仓库出发返回仓库的总路程。因此本题从 TSP 问题退化为一般的共起点可相交最小权路径覆盖问题。然而由于限制了覆盖的路径数量，本题无法转换为其对偶问题最大匹配问题使用 KM 算法求解，因此依然采用启发式算法求解。

首先，利用 $Floyd-Warshall$ 算法计算每个点到其他点的最短路径距离，同时，在计算每个节点对之间的最短路径时，要考虑到借道的情况。即货车在两两不能直达的节点对之间可以向其他能直达的节点对借道而不卸货，因此 $Floyd-Warshall$ 算法会更新距离矩阵中的距离值，将道路不连通的情况转化为虚拟的直达距离加长的情况。因此，问题变成了与问题二类似的情况，只是节点之间的距离发生了改变。

然后，将 14 个节点视为完全图，分成哈密顿回路构成的第一子图和对应导出子图。针对哈密顿回路，采用深度优先算法创建哈密顿回路集合，每个回路代表第一子图的最

佳巡航路径，即从仓库出发到返回仓库。对导出子图，首尾添加节点 5 作为定点仓库，并运用问题一中的禁忌路径寻优模型，获取导出子图最优巡航路径。

最后，计算每个哈密顿回路及其对应导出子图最优巡航路径的总路程，以选出总路程最短的最优路径方案。

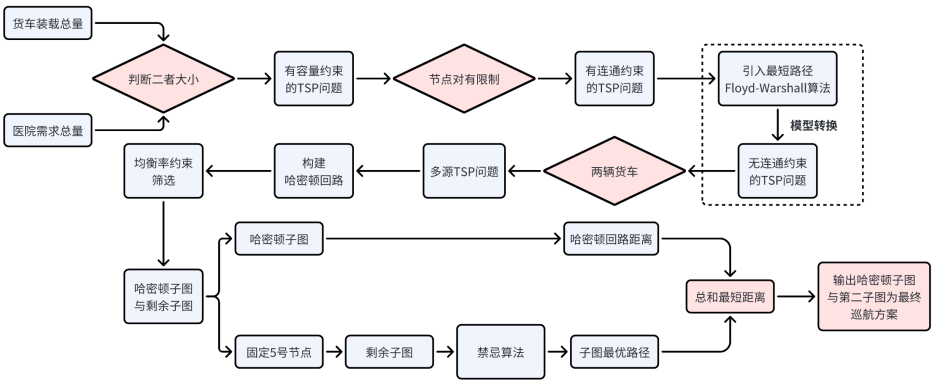

图 5　问题三流程图

4.3.2 模型建立

（1）最短路径邻接矩阵的构建

$Floyd-Warshall$ 算法是一种用于计算图中所有节点之间最短路径的动态规划算法。因此，为排除道路连通性对算法的干扰，需要结合医院的距离矩阵，医院间的连通性矩阵以构建医院间的邻接矩阵，从而更新医院间的最短距离矩阵。

①医院间的距离矩阵 C 的构建

在不考虑某些节点对没有直达道路的前提下，可以得到节点对之间的距离矩阵 $\boldsymbol{C}$，由于篇幅有限仅展示部分医院间的距离:

$$\boldsymbol{C}=\begin{bmatrix} 0 & 25.2240 & \cdots & 32.8195 & 31.5033 \\ 25.2240 & 0 & \cdots & 18.0863 & 8.7519 \\ \vdots & \vdots & & \vdots & \vdots \\ 32.8195 & 18.0863 & \cdots & 0 & 11.6458 \\ 31.5033 & 8.7519 & \cdots & 11.6458 & 0 \end{bmatrix}$$

其中，$C_{i,j}$ 表示第 i 个医院与第 j 个医院之间在不考虑有无直达道路情况下的距离，INF 表示两个医院站点之间无法直达。

②路网连通关系矩阵 $\boldsymbol{\sigma}$ 的构建

根据附件提供材料，可以得到每个节点之间的连通关系，即路网连接关系矩阵 $\boldsymbol{\sigma}$，由于篇幅有限，矩阵 $\boldsymbol{\sigma}$ 仅展示部分医院间的连通关系:

$$\boldsymbol{\sigma} = \begin{bmatrix} 1 & 1 & \cdots & \infty & \infty \\ 1 & 1 & \cdots & \infty & \infty \\ \vdots & \vdots & & \vdots & \vdots \\ \infty & \infty & \cdots & 1 & 1 \\ \infty & \infty & \cdots & 1 & 1 \end{bmatrix}$$

其中，$\sigma_{i,j}$ 表示第 i 个医院与第 j 个医院之间是否存在直达道路，1 表示有，∞ 表示没有。

如图 6 所示是每个医院站点间的道路连通情况。

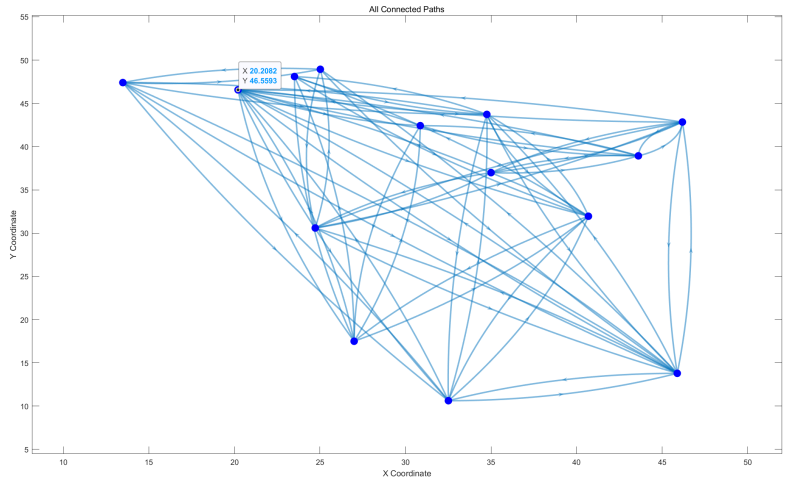

图 6　所有医院的道路连通情况

其中蓝色站点表示每个医院站点，有向箭头轨迹表示每个医院站点存在的出入道路，标点表示作为仓库的 5 号医院。

计算 $\boldsymbol{C}$ 与 $\boldsymbol{\sigma}$ 的 Hadamard 积，得到有道路限制下 G 的邻接矩阵 $\boldsymbol{C'}$：

$$\boldsymbol{C'} = \boldsymbol{C} \circ \boldsymbol{\sigma} = \begin{bmatrix} 0 & 25.2240 & \cdots & \infty & \infty \\ 25.2240 & 0 & \cdots & \infty & \infty \\ \vdots & \vdots & & \vdots & \vdots \\ \infty & \infty & \cdots & 0 & 11.6458 \\ \infty & \infty & \cdots & 11.6458 & 0 \end{bmatrix}$$

③*Floyd-Warshall* 算法求多源最短路径

Floyd-Warshall 算法是一个求多源最短路径的动态规划算法。动态规划方程如下：

$$D_{ij}^{(k)} = \begin{cases} C'_{ij} & k = 0 \\ min\{D_{ij}^{(k-1)}, D_{ik}^{(k-1)} + D_{kj}^{(k-1)}\} & k \geq 1 \end{cases} \tag{14}$$

其中 D_{ij}^{k} 表示第 i 号医院节点与第 j 号医院节点间的经过第 k 号医院节点的最短路径。

该算法考虑的是一条最短路径的中间节点，利用了路径 p 和从 i 到 j 之间中间节点均取自集合 $\langle \pi_1, \pi_2, ..., \pi_n \rangle$ 的最短路径之间的关系。该关系依赖于节点 k 是否为路径 p 上的一个中间节点。因为对于任何路径，所有中间节点均属于集合 $\langle \pi_1, \pi_2, ..., \pi_n \rangle$，因此 $\boldsymbol{D}^n$ 就是最终的最短路径矩阵 $\boldsymbol{D}$。

例如，由矩阵 C 中读出节点 1 到节点 j 的直达距离分别为 0、25.224、$\cdots$、∞、∞。以节点 1 为例，显然节点 1 无法直接到达节点 13、14，但是，此时让节点 1 经过中转节点 k (k=2、$\cdots$、13/14) 再走到节点 13/14。同样的，节点 1 到节点 $l, (l = 2, 3, \cdots, 12)$ 虽然有直达路径，但不一定这些直达路径就是最短路径，仍可以为节点 1 选择中转节点 m，通过比较节点 1 到节点 m 再到节点 l 的距离能否比节点 1 直接到节点 l 小，来确定节点 1 到节点 l 的最短路径。

由此，最终得到了每个节点对之间的最短路径距离矩阵 $\boldsymbol{D}$ 如下：

$$D = \begin{bmatrix} 0 & 25.2240 & \cdots & 54.6920 & 66.3378 \\ 25.2240 & 0 & \cdots & 64.1489 & 52.5030 \\ \vdots & \vdots & & \vdots & \vdots \\ 54.6920 & 64.1489 & \cdots & 0 & 11.6458 \\ 66.3378 & 52.5030 & \cdots & 11.6458 & 0 \end{bmatrix}$$

（2）哈密顿回路集合的构建

深度优先搜索（DFS）是一种在图遍历中广泛应用的搜索方法，用于找到以 5 号医院节点为起点的长度为 n 的哈密顿回路，并随机生成每一个点邻居的访问顺序，在进行大量嗅探后该算法可以寻找到全部哈密顿回路，组成哈密顿回路集合 $\boldsymbol{P}$。为了更加清晰的展示该算法的一次搜索过程，下面是按照步骤来分点写的描述：

Step 1: 创建 $FindHamiltonianCycle$ 过程以及三个参数：图 G，当前节点 u，和当前路径 p。

Step 2: 在过程开始时，首先检查路径 p 的长度是否等于设定的节点数 n，并且当前节点 u 是否与路径 p 的第一个节点相邻。如果满足这两个条件，说明已经找到了哈密顿回路，于是返回路径 p 作为结果。

Step 3: 如果不满足上述条件，进入循环，遍历当前节点 u 的未被标记的邻居节点 v。

Step 4: 对于每个未被标记的邻居节点 v，将节点 v 添加到路径 p 的末尾，并标记节点 v 为已访问。

Step 5: 递归调用 $FindHamiltonianCycle$ 过程，传递图 G、节点 v 和更新后的路径 p。

Step 6: 如果递归调用返回的结果 $result$ 不为空，说明找到了哈密顿回路，于是将 $result$ 作为结果返回。

Step 7: 如果递归调用没有找到哈密顿回路，需要回溯。首先，将路径 p 的最后一个节点移除，然后取消标记节点 v 为已访问。

Step 8: 继续循环，查找下一个未被标记的邻居节点。

Step 9: 如果遍历完所有未被标记的邻居节点仍然没有找到哈密顿回路，最终返回“未找到哈密顿回路”。

（3）共起点禁忌路径寻优模型的建立

记 $\boldsymbol{P} = (\boldsymbol{p_1}, \boldsymbol{p_2}, ..., \boldsymbol{p_m})$, 其中 m 指哈密顿回路的数量，$\boldsymbol{p_i} = \langle V_{i1}, V_{i2}, ..., V_{iN_i} \rangle$, V_{N_i} 为第 i 条哈密顿回路的第 N_i 个节点。

对于任一条哈密顿回路 $\boldsymbol{p_i}$, 定义从图 G 中剔除掉 $\boldsymbol{p_i}$ 后的导出子图为 $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$ 其中

$$V_{\pi,i} = \{v : v \in V, v \notin \boldsymbol{p_i}\} \cup \{5\}, E_{\pi,i} = \{e = (u, v) : e \in E \cap e \in (V_{\pi,i} \times V_{\pi,i})\}$$

对于任一导出子图，利用问题一中的禁忌路径寻优模型，将每条哈密顿回路与其对应导出子图的最短路径相结合，总路程最短的路径即为最优路径。

因此，可得共起点禁忌路径寻优模型如下：

$$\begin{cases} \min\limits_{k=1}^{m} f_k = \psi_{k,1} + \psi_{k,2} = \sum_{i=1}^{N_k - 1} D_{V_{ki}V_{k(i+1)}} + \sum_{i=1}^{n-N_k} D_{\pi_{ki}\pi_{k(i+1)}} \\ \eta \geq \frac{\sum\limits_{i=1}^{14} R_i - 500}{500} \end{cases} \tag{15}$$

其中，$\psi_{k,1}$ 指的是第 i 条哈密顿回路 $\boldsymbol{p_k}$ 的长度，$\psi_{k,2}$ 指的是导出子图中最优巡航长度。

因此，此问题可归结为与问题二相同的问题，即已知节点之间距离（即 D 中的最短距离）的条件下，实现两辆货车的总行驶距离最小化。

4.3.3 问题解答

（1）求解哈密顿回路集合

一共有 14 个节点，以第 5 个节点同时作为起点和终点，则哈密顿回路可以由最多 15 个节点，最少 3 个节点构成，通过深度优先搜索，在进行大量嗅探后可得到所有的哈密顿回路。由于篇幅优先，这里仅选取了前三条哈密顿回路如表 5 所示：

表 5　所有哈密顿回路

起点	途径节点													终点
5	3	6	4	10	14	9	13	8	7	12	11	1	2	5
5	6	10	4	3	2	1	11	7	8	12	13	14	9	5
5	9	6	4	10	14	13	12	8	7	11	1	2	3	5

根据问题二中的车辆均衡率要求，对哈密顿回路集合进行筛选，仅保留那些满足均衡率的回路，于是，最终得到了 m =878 条满足车辆均衡率的哈密顿回路，由于篇幅有限，表 6 仅展示前 3 条回路：

表 6　满足车辆均衡率所有哈密顿回路

起点	途径节点										终点
5	8	12	13	14	9	6	4	3	2	1	5
5	9	14	13	8	12	11	1	2	3	6	5
5	6	3	2	1	11	12	8	13	14	9	5

（2）最优路径的求解

在每一条哈密顿回路所对应的导出子图中添加 5 号节点作为起止点，利用问题一中建立的禁忌路径寻优模型，可以寻找导出子图中的最短路径。由于篇幅有限，表 7 仅展示导出子图中所有路径的其中 5 条路径：

表 7　导出子图的最短路径

起点	途径节点										终点
5	2	3	4	6	9	10	11	12	13	14	5
5	1	2	3	4	6	8	10	11	13	14	5
5	1	2	3	4	6	8	10	11	13	14	5
5	2	3	4	6	8	9	10	12	13	14	5
5	1	2	3	4	7	8	11	12	13	14	5

综合计算哈密顿回路和导出子图的最短路径，得到最优路径规划方案如下：

哈密顿回路中的货车巡航路线 $5 \to 6 \to 4 \to 3 \to 2 \to 5$；

导出子图中的货车巡航路线：$5 \to 7 \to 8 \to 9 \to 10 \to 14 \to 13 \to 12 \to 11 \to 1 \to 5$。

此时，最短路径为：285.4271。

为更直观地呈现最优路径规划方案，将其表示在图 7：

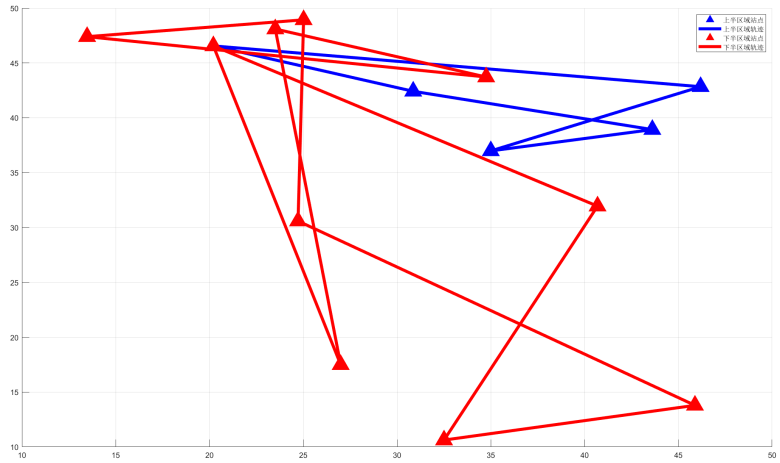

图 7　问题三最优路径轨迹

4.4 问题四

4.4.1 问题分析

鉴于允许在另一节点增设新仓库，使得两辆运输车可以以不同仓库为起点出发并最终回到各自的起点，相较于共享同一起点的情况，具备更高的操作自由度。因此，该问题可以进一步推广为最小可相交路径覆盖问题。值得注意的是，在非二分图上的最小权路径覆盖问题通常属于典型的 NP-hard 问题。因此，其求解步骤与问题三相似，唯一的区别在于在对导出子图求解最短路径时，不需要加入第 5 节点作为起止点。后续步骤保持不变，以此得到最优路径规划方案。

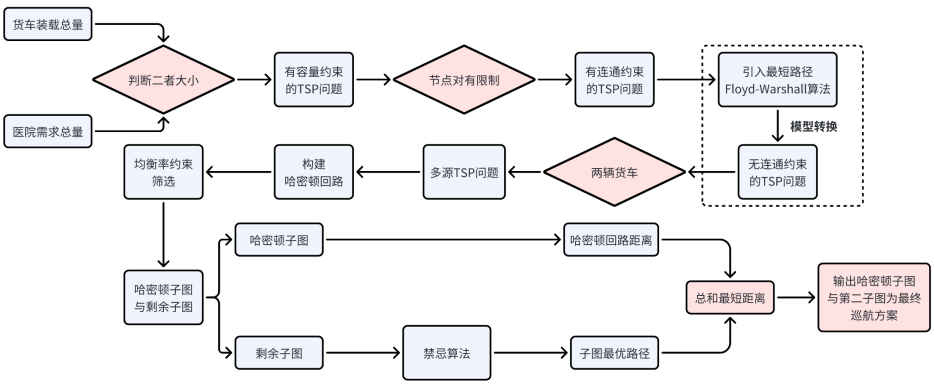

图 8　问题四流程图

4.4.2 模型建立

（1）$Floyd-WWarshall$ 最短路径邻接矩阵的构建

同问题三，仍利用道路连通矩阵和节点距离矩阵构建最短路径距离矩阵 D：$\boldsymbol{D}$ 如下：

$$D=\begin{bmatrix} 0 & 25.2240 & \cdots & 54.6920 & 66.3378 \\ 25.2240 & 0 & \cdots & 64.1489 & 52.5030 \\ \vdots & \vdots & & \vdots & \vdots \\ 54.6920 & 64.1489 & \cdots & 0 & 11.6458 \\ 66.3378 & 52.5030 & \cdots & 11.6458 & 0 \end{bmatrix}$$

（2）哈密顿回路集合与剩余子图集合的构建

利用 DFS 构建哈密顿回路集合，并计算对应的导出子图。

（3）可变起点禁忌路径寻优模型的建立

记当前选择哈密顿回路为路径 $\boldsymbol{p}$。由问题二对凸包划分模型证明的第二部分可知，车辆出发的仓库节点 v_s 一定位于其运货目标节点集合，即该路径 $\boldsymbol{p}$ 的导出子图的点集 V_π 构成的凸包中，因此在最优情况下必有 $v_s \notin \boldsymbol{p}$，从而 $V_\pi \cap \boldsymbol{p} = \phi$。因此只需要在 V_π 中选择某一节点作为仓库节点即可。与问题三不同的是，第二辆车的出发与终止节点可以为任意节点，因此前文在搜索解空间的邻居时交换位置 u 与 v 的选择范围由 $(1 < u < v < 15)$ 变化为 $(1 \leq u < v \leq n - N)$，其中 N 为的 $\boldsymbol{p}$ 长度。

4.4.3 问题解答

利用问题一中建立的禁忌路径寻优模型，直接通过导出子图寻找最短路径。这些节点存在车辆在导出子图中巡航的最短路径，那么仓库在其中任意一个节点都可以存在。由于篇幅有限，表 8 仅展示导出子图中所有路径的其中 5 条路径：

表 8　导出子图的最短路径

起点	途径节点								终点
2	3	4	6	9	10	11	12	13	14
1	2	3	4	6	8	10	11	13	14
1	2	3	4	6	8	10	11	13	14
2	3	4	6	8	9	10	12	13	14
1	2	3	4	7	8	11	12	13	14

综合计算哈密顿回路和导出子图的最短路径，得到最优路径规划方案如下：

哈密顿回路中的货车巡航路线：$5 \to 2 \to 3 \to 4 \to 6 \to 5$ ；

导出子图中的货车巡航路线：$1 \to 7 \to 8 \to 9 \to 10 \to 14 \to 13 \to 12 \to 11 \to 1$ 。

此时的最短路程为：**250.3326**。

为更直观地呈现最优路径规划方案，将其表示在图 9：

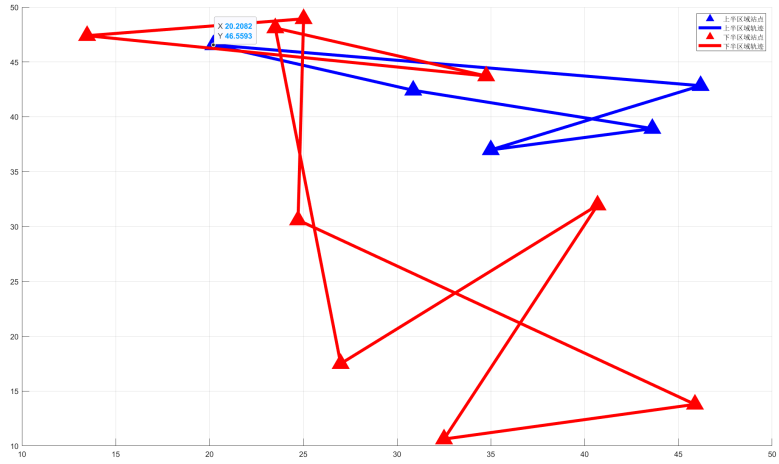

图 9　问题四最优路径轨迹

五、 模型的稳定性分析

问题三中探究了道路连通性对最优巡航路径影响的数学模型。而问题三中所建立的模型是否稳定，不仅影响了问题四求解的精确性，也是问题四求解的基础。

为深入分析道路连通性对最优巡航路径数学模型的稳定性，需要特别关注在扰动某一项指标情况下模型的稳定性。以关闭 5 号和 9 号医院节点对之间的连通性关系为例，其出入度和客流需求量高，在关闭了这条道路后，模型的稳定性如下：

当关闭 5 号和 9 号节点对之间的道路时，模型最短巡航路径为 250.3326，劣化 0%。

观察到在扰动情况下，所得结果的仍然和原结果相同，模型稳定性强。

六、 模型的评价

6.1 模型的优点

1. 本文使用禁忌搜索算法得到问题的近似最优解，在保证结论优势度的前提下极大提升了算法效率。

2. 在第四问中采用随机节点访问顺序的方法构建哈密顿回路集合，可以通过修改嗅探次数平衡策略的执行效率与解的近似度，具有较好的弹性和可拓展性。

6.2 模型的缺点

1. 采用深度搜索算法和禁忌搜索算法，matlab 的运行效率较低。

七、 模型的推广

在一般图上的最小边权覆盖问题和最小路径覆盖问题都是经典的 NP-hard 问题，本文提供了在约定路径覆盖数量的情况下最小边权覆盖的启发式算法，并在方案中考虑了作业对象的负载均衡率。在物流调度，工厂工作调度和集群编队调度等问题中有广泛的应用。

参考文献

[1] 马学森, 宫帅, 朱建, 等. 2018. 动态凸包引导的偏优规划蚁群算法求解 TSP 问题 [J]. 通信学报, 39(10): 59-71.

[2] 姜启源，谢金星，叶俊. 数学模型（第四版）[M]. 北京：高等教育出版社，2011.

附录清单

附录一　哈密顿回路集合构建

附录二　对问题一的哈密顿回路禁忌搜索最优巡航路线

附录三　对所有可行的分割方案可视化

附录四　计算所有分割方案的均衡率需求

附录五　对所有分割方案进行禁忌搜索寻找最优巡航路径

附录六　筛选满足均衡率的哈密顿回路和补集

附录七　计算哈密顿回路和禁忌搜索最优补集路径距离

附录八　计算哈密顿回路补集的最优巡航路径

附录 一　哈密顿回路集合构建

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
using std::ifstream;
using std::ofstream;
using std::cout;
constexpr int MAXN=20;
constexpr int MAXM=300000;
constexpr int MOD=1000000007;
struct Edge
{
    int v{0};
    int next{0};
}edge[MAXM];

int n, m, edgeNum{0},PathTry{MAXM};
int head[MAXN], Path[MAXN],degree[MAXN];
long long hamitonHash[MAXM];
bool vis[MAXN];
ifstream fin("Path.txt");
//ofstream cout("Hamilton.csv",std ::ios::app);

void DFS(int u,int t,int PathLen)
{
    vis[u] = t;
    Path[t] = u;
    if (t >= PathLen)
    {
        return;
    }
    int indV=rand()%degree[u];
    int e=head[u];
    for (int i=0; i<indV; ++i)
        e = edge[e].next;
    int v = edge[e].v;
    for (int count=0; count<100; ++count)
    {
        if (!vis[v])
            break;
        indV=rand()%degree[u];
        e=head[u];
        for (int i=0; i<indV; ++i)
```

```
            e = edge[e].next;
        v = edge[e].v;
    }
    if (!vis[v])
    {
        vis[v] = t;
        DFS(v, t + 1,PathLen);
        vis[v] = 0;
    }
    return;
}
void AddEdge(int u, int v){
   //Add edge
    edgeNum++;
    edge[edgeNum].v = v;
    edge[edgeNum].next = head[u];
    head[u] = edgeNum;
    degree[u]++;
    return;
}

void Print(int PathLen)
{
    for (int i=1; i<=PathLen; ++i)
        cout << Path[i] << ",";
    cout << 5 << std::endl;
    return;
}
long long CalcHash(int PathLen)
{
    long long res = 0;
    int len=0;
    for (int i=1; Path[i]; ++i){
        res =res*MAXN%MOD + Path[i];
        ++len;
    }
    if (len != PathLen)
        return -1;
    int u=Path[len];
    for (int e=head[u]; e!=-1; e=edge[e].next)
    {
        int v=edge[e].v;
        if (v == Path[1])
            return res;
    }
    return -1;
}
```

```
void InitGraph()
{
    //init
    fin >> n >> m;
    for (int i=0; i<MAXN; ++i)
        head[i] = -1;
    //input graph by stars
    for (int e=1; e<=m; ++e){
        int u, v;
        fin >> u >> v;
        AddEdge(u, v);
        AddEdge(v, u);
    }
    fin.close();
    return;
}
void Solve(int root)
{
    //serach Hamilton Path from the root
    for (int PathLen=n; PathLen>3; --PathLen)
    {
        int cnt=0;
        for (int i=0; i<PathTry; ++i)
            hamitonHash[i] = 0;
        for (int k=0; k<PathTry; ++k)
        {
            for (int i=0; i<MAXN; ++i)
                vis[i] = 0;
            for (int i=0; i<MAXN; ++i)
                Path[i] = 0;
            DFS(root,1,PathLen);
            long long tmpHash=CalcHash(PathLen);
            if (tmpHash<0)
                continue;
            bool exist=false;
            for (int i=1; i<k; ++i)
            {
                if (hamitonHash[i] == tmpHash)
                {
                    exist=true;
                    break;
                }
            }
            if (!exist)
            {
                hamitonHash[++cnt] = tmpHash;
```

```
            if (tmpHash > 0)
            {
                Print(PathLen);
                cout.flush(); // 将数据刷新到文件中
            }
        }
    }
    return;
}

int main()
{
    freopen("Hamilton.csv", "w", stdout);
    std::ios_base::sync_with_stdio(false);
    srand(time(NULL));
    InitGraph();
    Solve(5);
    return 0;
}
```

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
using std::ifstream;
using std::ofstream;
using std::cout;
constexpr int MAXN=20;
constexpr int MAXM=100000;
constexpr int MOD=1000000007;
struct edge
{
    int v{0},u{0};
    int next{0};
}Edge[MAXM],resEdge[MAXM];

int n, m, edgeNum{0}, edgeResNum{0},PathTry{MAXM};
int Head[MAXN], Path[MAXN], Degree[MAXN], nodeNum { 0 },node[MAXN];
int resHead[MAXN], resPath[MAXN], resDegree[MAXN];
long long hamitonHash[MAXM];
bool vis[MAXN];
ifstream fin("Path.txt");
//ofstream cout("Hamilton.csv",std ::ios::app);
```

```
void DFS(int u,int t,int PathLen)
{
    vis[u] = t;
    Path[t] = u;
    if (t >= PathLen)
        return;
    int indV=rand()%Degree[u];
    int e=Head[u];
    for (int i=0; i<indV; ++i)
        e = Edge[e].next;
    int v = Edge[e].v;
    for (int count=0; count<100; ++count)
    {
        if (!vis[v])
            break;
        indV=rand()%Degree[u];
        e=Head[u];
        for (int i=0; i<indV; ++i)
            e = Edge[e].next;
        v = Edge[e].v;
    }
    if (!vis[v])
    {
        vis[v] = t;
        DFS(v, t + 1,PathLen);
        vis[v] = 0;
    }
    return;
}
void resDFS(int u, int t, int PathLen)
{
    vis[u] = t;
    resPath[t] = u;
    if (t >= PathLen)
        return;
    for (int e=resHead[u]; e!=-1; e=resEdge[e].next)
    {
        int v = resEdge[e].v;
        if (!vis[v])
        {
            vis[v] = t;
            resDFS(v, t + 1, PathLen);
            vis[v] = 0;
        }
    }

    return;
```

```
}
void AddEdge(int u, int v){
   //Add Edge
    edgeNum++;
    Edge[edgeNum].v = v;
    Edge[edgeNum].u = u;
    Edge[edgeNum].next = Head[u];
    Head[u] = edgeNum;
    Degree[u]++;
    return;
}
void AddResEdge(int u, int v)
{
    // Add Edge
    edgeResNum++;
    resEdge[edgeResNum].v = v;
    resEdge[edgeResNum].u = u;
    resEdge[edgeResNum].next = resHead[u];
    resHead[u] = edgeResNum;
    resDegree[u]++;
    return;
}
long long CalcHash(int PathLen)
{
    long long res = 0;
    int len = 0;
    for (int i = 1; Path[i]; ++i)
    {
        res = res * MAXN % MOD + Path[i];
        ++len;
    }
    if (len != PathLen)
        return -1;
    int u = Path[len];
    for (int e = Head[u]; e != -1; e = Edge[e].next)
    {
        int v = Edge[e].v;
        if (v == Path[1])
            return res;
    }
    return -1;
}
long long CalcResHash(int PathLen)
{
    long long res = 0;
    int len = 0;
    for (int i = 1; resPath[i]; ++i)
```

```
    {
        res = res * MAXN % MOD + resPath[i];
        ++len;
    }
    if (len != PathLen)
        return -1;
    int u = resPath[len];
    for (int e = resHead[u]; e != -1; e = resEdge[e].next)
    {
        int v = resEdge[e].v;
        if (v == resPath[1])
            return res;
    }
    return -1;
}

int ChechResHamiton(int PathLen)
{
    bool exist[MAXN];
    nodeNum=0;
    edgeResNum = 0;
    for (int i = 0; i < MAXN; ++i) exist[i] = false;
    for (int i = 0; i <MAXN; ++i) node[i]=0;
    for (int i = 0; i < MAXN; ++i) resHead[i] = -1;
    for (int i = 0; i < MAXN; ++i) resPath[i] = 0;
    for (int i = 0; i < MAXN; ++i) resDegree[i] = 0;
    for (int i = 1; i <= PathLen; ++i) exist[Path[i]] = true;
    exist[5] = false;
    for (int i=0; i<MAXN; ++i) vis[i]=false;
    node[nodeNum++] = 5;
    for (int i=1; i<=n; ++i)
        if (!exist[i])
            node[nodeNum++] = i;
    --nodeNum;
    for (int e=1; e<=m; ++e)
    {
        int u = Edge[e].u, v = Edge[e].v;
        if (!exist[u]&&!exist[v])
            AddResEdge(u, v);
    }
    for (int i=0; i<PathTry; ++i)
    {
        resDFS(node[0], 1, nodeNum);
        if (CalcResHash(nodeNum) > 0)
            return node[0];
    }
    return 0;
```

```
}

void Print(int PathLen)
{
    for (int i=1; i<=PathLen; ++i)
        cout << Path[i] << ",";
    cout << 5 << std::endl;
    return;
}
void PrintRes(int nodeNum,int root)
{
    for (int i=1; i<=nodeNum; ++i)
        cout << resPath[i] << ",";
    cout << root << std::endl;
    return;
}

void InitGraph()
{
    //init
    fin >> n >> m;
    for (int i=0; i<MAXN; ++i)
        Head[i] = -1;
    //input graph by stars
    for (int e=1; e<=m; ++e){
        int u, v;
        fin >> u >> v;
        AddEdge(u, v);
        AddEdge(v, u);
    }
    fin.close();
    return;
}
void Solve(int root)
{
    //serach Hamilton Path from the root
    for (int PathLen=n-2; PathLen>3; --PathLen)
    {
        int cnt=0;
        for (int i=0; i<PathTry; ++i)
            hamitonHash[i] = 0;
        for (int k=0; k<PathTry; ++k)
        {
            for (int i=0; i<MAXN; ++i)
                vis[i] = 0;
            for (int i=0; i<MAXN; ++i)
                Path[i] = 0;
```

```
            DFS(root,1,PathLen);
            long long tmpHash=CalcHash(PathLen);
            if (tmpHash<0)
                continue;
            bool exist=false;
            for (int i=1; i<k; ++i)
            {
                if (hamitonHash[i] == tmpHash)
                {
                    exist=true;
                    break;
                }
            }
            if (!exist)
            {
                hamitonHash[++cnt] = tmpHash;
                if (tmpHash > 0)
                {
                    int resRoot = ChechResHamiton(PathLen);
                    if (resRoot)
                    {
                        Print(PathLen);
                        PrintRes(nodeNum,resRoot);
                    }
                    cout.flush(); // 将数据刷新到文件中
                }
            }
        }
    }
    return;
}

int main()
{
    freopen("Hamilton3.csv", "w", stdout);
    std::ios_base::sync_with_stdio(false);
    srand(time(NULL));
    InitGraph();
    Solve(5);
    return 0;
}
```

附录 二　对问题一的哈密顿回路禁忌搜索最优巡航路线

```
%%
clc;close all;clear
load("Coordinate_point.mat")
%%
% 数据预处理，保存为 waypoints
waypoints = data;

% 距离矩阵，存储各个点之间的距离
num_points = size(waypoints, 1);
dist_matrix = zeros(num_points, num_points);

for i = 1:num_points
    for j = 1:num_points
        dist_matrix(i, j) = sqrt((waypoints(i, 2) - waypoints(j, 2))^2 + (waypoints(i, 3) -
            waypoints(j, 3))^2);
    end
end


%%
bestall_solution = [];
for o = 1:100
% 禁忌搜索参数
num_iterations = 1000; % 迭代次数
tabu_length = 10;   % 禁忌表长度
% 初始化
remaining_points = setdiff(1:num_points, 5);
current_solution = [5, randperm(13), 5]; % 初始解
% 假设 current_solution 已经生成
current_solution(2:14) = current_solution(2:14) + (current_solution(2:14) >= 5);

best_solution = current_solution;
best_distance = calculate_total_distance(dist_matrix, current_solution);

%%
% 迭代搜索
tabu_list = zeros(1, num_points+1);

for iter = 1:num_iterations
    best_neighbor = [];
    best_neighbor_distance = Inf;

    for i = 1:num_points
        for j = i+1:num_points
            new_solution = current_solution;
            new_solution(i) = current_solution(j);
            new_solution(j) = current_solution(i);
```

```
            if ~ismember(new_solution, tabu_list, $*$rows$*$)
                distance = calculate_total_distance(dist_matrix, new_solution);
                if distance < best_neighbor_distance
                    best_neighbor_distance = distance;
                    best_neighbor = new_solution;
                end
            end
        end
    end

    if best_neighbor_distance < best_distance
        current_solution = best_neighbor;
        best_distance = best_neighbor_distance;

        if size(tabu_list, 1) >= tabu_length
            tabu_list(1, :) = [];
        end
        tabu_list = [tabu_list; best_neighbor];

        if best_distance < calculate_total_distance(dist_matrix, best_solution)
            best_solution = best_neighbor;
        end
    end
end

% 在最优解中添加起始点并计算总距离
best_solution = [best_solution]; % 添加出发点
best_distance = calculate_total_distance(dist_matrix, best_solution);

disp($*$最优路径: $*$);
disp(best_solution);
disp([$*$最短总路程: $*$, num2str(best_distance)]);
bestall_solution(o,1) = best_distance;
end
Newbest = min(bestall_solution);
%%

% 计算路径总距离的函数
function total_distance = calculate_total_distance(dist_matrix, solution)
    total_distance = 0;
    num_points = length(solution);
    for i = 1:num_points-1
        total_distance = total_distance + dist_matrix(solution(i), solution(i+1));
    end
    total_distance = total_distance + dist_matrix(solution(num_points), solution(1)); %
        返回起始点
```

```
end
```

附录 三　对所有可行的分割方案可视化

```
%%
% clc;close all;clear
load("Coordinate_point.mat")

%% 问题一的最短路径
% order = [ 5  11   14    2   12    4    6    3    7    9    8    1   10   13    5];
%order = [   5    13   10    1    8    9    7    3    6    4   12    2   14   11    5];
%% 问题二 58最短路径
order = [5   2    4    7    9    3    6   12   14   11    5];%ABove
order2 = [    5   10    8    1   13    5 ];%Below
clear order order2
%% 问题二 54最短路径
order2 = [5   7    9    8    1   10   13    5];%下班红
order = [ 5 11 14 12  6  3  4  2  5];%上半蓝
clear order order2
%% 问题二 57最短路径
order2 = [ 5   9    8    1   10   13    5];%下班红
order = [ 5   11   14   12    6    3    7    4    2    5 ];%上半蓝
clear order order2
%% 问题二 52最短路径
order2 = [ 5  13   10    1    8    9    7    4    5];%下班红
order = [ 5   2   3    6   12   14   11    5 ];%上半蓝
clear order order2
%%
order = [ 5  9  6  3  4  10 14 13 12 11 7 8 5];
order2 = [5 1 2 5];
% 计算总路程
total_distance = 0;

for i = 1:length(order)-1
    point1 = data(order(i), 2:3);
    point2 = data(order(i+1), 2:3);
    distance = norm(point2 - point1);
    total_distance = total_distance + distance;
end

disp([$*$Total distance: $*$, num2str(total_distance)]);

% 绘制轨迹图像
figure;
% plot(data(:, 2), data(:, 3), $*$bo$*$,$*$MarkerSize$*$,10,$*$MarkerFaceColor$*$,$*$b$*$)
```

```
% plot(data(:, 2), data(:, 3), $*$bo$*$,$*$MarkerSize$*$,20,$*$MarkerFaceColor$*$,[.5 .5 .5])
hold on;
plot(data(order, 2), data(order, 3),
    $*$b^$*$,$*$MarkerSize$*$,20,$*$MarkerFaceColor$*$,$*$b$*$)
plot(data(order, 2), data(order, 3), $*$b-$*$,$*$LineWidth$*$,4); % 绘制按顺序的点
xlabel($*$X Coordinate$*$);
ylabel($*$Y Coordinate$*$);
title($*$Trajectory$*$);

grid on;
hold on;
clear order
%%
% plot(data(:, 2), data(:, 3), $*$bo$*$,$*$MarkerSize$*$,10,$*$MarkerFaceColor$*$,$*$b$*$)
plot(data(order2, 2), data(order2, 3),
    $*$r^$*$,$*$MarkerSize$*$,20,$*$MarkerFaceColor$*$,$*$r$*$); % 绘制按顺序的点
plot(data(order2, 2), data(order2, 3), $*$r-$*$,$*$LineWidth$*$,4); % 绘制按顺序的点
legend($*$上半区域站点$*$,$*$上半区域轨迹$*$,$*$下半区域站点$*$,$*$下半区域轨迹$*$)
total_distance_order2 = 0;
clear distance
for i = 1:length(order2)-1
   point1 = data(order2(i), 2:3);
   point2 = data(order2(i+1), 2:3);
   distance = norm(point2 - point1);
   total_distance_order2 = total_distance_order2 + distance;
end

disp([$*$Total distance: $*$, num2str(total_distance_order2)]);
hold off

Totaldistance = total_distance_order2+total_distance;
disp($*$Alldistance=$*$)
disp(Totaldistance)
```

附录 四　计算所有分割方案的均衡率需求

```
clc;close all;clear
load($*$Cargoneedpreday.mat$*$)
Cargoperday = data;
clear data
load($*$Coordinate_point.mat$*$)
Pointdata = data;
clear data
Pointdata(:,4)=Cargoperday(:,2);
```

```
%% 计算货物需求均衡率
% 假设你已经加载了Cargoperday和Pointdata数据

% 遍历从点1到点14
for end_point = 1:14
    if end_point == 5
        continue; % 跳过点5
    end

    % 选择点5和另一个点作为连线的两个点
    point5 = Pointdata(5, 2:3);
    point_end = Pointdata(end_point, 2:3);

    % 计算连线的斜率和截距
    slope = (point_end(2) - point5(2)) / (point_end(1) - point5(1));
    intercept = point5(2) - slope * point5(1);

    % 根据连线方程，判断每个点在连线的上方还是下方，并计算上下两侧的货物需求
    demandAboveLine = 0;
    demandBelowLine = 0;
    % 初始化上半区域和下半区域的点
pointsAboveLine = [];
pointsBelowLine = [];
needpoint = [];
    for i = 1:size(Pointdata, 1)
        x = Pointdata(i, 2);
        y = Pointdata(i, 3);
        demand = Pointdata(i, 4);

        if y >= slope * x + intercept
            demandAboveLine = demandAboveLine + demand;
             pointsAboveLine = [pointsAboveLine; Pointdata(i, :)];
        else
            demandBelowLine = demandBelowLine + demand;
             pointsBelowLine = [pointsBelowLine; Pointdata(i, :)];
        end
    end
    demandAboveLine=demandAboveLine-70;
    minDemand = min(demandAboveLine, demandBelowLine);
    maxDemand = max(demandAboveLine, demandBelowLine);
    balanceRate = (minDemand / maxDemand) * 100;
    fprintf($*$点%d为连线另一端时，上半部分的货物需求总量：%.2f\n$*$, end_point, demandAboveLine);
    fprintf($*$点%d为连线另一端时，下半部分的货物需求总量：%.2f\n$*$, end_point, demandBelowLine);
    fprintf($*$点%d为连线另一端时，货物需求均衡率：%.2f%%\n$*$, end_point, balanceRate);
    fprintf($*$-----------------------------------------------\n$*$);
```

```
%    figure(end_point)
%    % 绘制上半区域的点
% scatter(pointsAboveLine(:, 2), pointsAboveLine(:, 3), $*$filled$*$, $*$DisplayName$*$,
    $*$上半区域$*$);
% hold on;
%
% % 绘制下半区域的点
% scatter(pointsBelowLine(:, 2), pointsBelowLine(:, 3), $*$filled$*$, $*$DisplayName$*$,
    $*$下半区域$*$);
%
% % 绘制连线
% plot([point5(1),point_end(1)], [point5(2), point_end(2)], $*$r$*$, $*$LineWidth$*$, 2,
    $*$DisplayName$*$, $*$连线$*$);
%
% % 设置坐标轴标签等
% xlabel($*$横坐标$*$);
% ylabel($*$纵坐标$*$);
% title($*$上半区域和下半区域点的分布$*$);
% grid on;
% legend;
end
%% 以点4为例计算上下两个半区

close all;clc

% 选择点5和点8作为连线的两个点，一共需要2 4 7 8 9
point5 = Pointdata(5, 2:3);
point4 = Pointdata(8, 2:3);

% 计算连线的斜率和截距
slope = (point4(2) - point5(2)) / (point4(1) - point5(1));
intercept = point5(2) - slope * point5(1);

% 初始化上半区域和下半区域的点
pointsAboveLine = [];
pointsBelowLine = [];

for i = 1:size(Pointdata, 1)
    x = Pointdata(i, 2);
    y = Pointdata(i, 3);

    if y >= slope * x + intercept
        pointsAboveLine = [pointsAboveLine; Pointdata(i, :)];
    else
        pointsBelowLine = [pointsBelowLine; Pointdata(i, :)];
    end
```

```
end

% 绘制上半区域的点
scatter(pointsAboveLine(:, 2), pointsAboveLine(:, 3), $*$filled$*$, $*$DisplayName$*$,
    $*$上半区域$*$);
hold on;

% 绘制下半区域的点
scatter(pointsBelowLine(:, 2), pointsBelowLine(:, 3), $*$filled$*$, $*$DisplayName$*$,
    $*$下半区域$*$);

% 绘制连线
plot([point5(1), point4(1)], [point5(2), point4(2)], $*$r$*$, $*$LineWidth$*$, 2,
    $*$DisplayName$*$, $*$连线$*$);

% 设置坐标轴标签等
xlabel($*$横坐标$*$);
ylabel($*$纵坐标$*$);
title($*$上半区域和下半区域点的分布$*$);
grid on;
legend;

% 保持图形窗口打开
hold off;
```

附录 五　对所有分割方案进行禁忌搜索寻找最优巡航路径

```
clc;close all;clear
load($*$PointAbove5_4.mat$*$)
load($*$PointBelow5_4.mat$*$)
load("Coordinate_point.mat")

% 数据预处理，保存为 waypoints
waypoints = data;
clear data

%  data = pointsAboveLine;
data = pointsBelowLine;

% 距离矩阵，存储各个点之间的距离
num_points = size(waypoints, 1);
dist_matrix = zeros(num_points, num_points);

for i = 1:num_points
   for j = 1:num_points
```

```
        dist_matrix(i, j) = sqrt((waypoints(i, 2) - waypoints(j, 2))^2 + (waypoints(i, 3) -
            waypoints(j, 3))^2);
    end
end
clear i
 %%
bestall_solution = [];
for o = 1:100
% 禁忌搜索参数
num_iterations = 1000; % 迭代次数
tabu_length = 10;   % 禁忌表长度
% 初始化
remaining_points = setdiff(1:num_points, 5);

randdata = length(data(:,1));
current_solution = [5, randperm(randdata), 5]; % 初始解


idx=current_solution(2:randdata+1);
% 上半区域
% for i = 1:randdata
%     current_solution(1,i+1)=pointsAboveLine(idx(1,i),1);
% end
% clear i

% 下半区域
for i = 1:randdata
    current_solution(1,i+1)=pointsBelowLine(idx(1,i),1);
end
clear i
L = length(current_solution(1,:));
%%
best_solution = current_solution;
best_distance = calculate_total_distance(dist_matrix, current_solution);
%%
% 迭代搜索
tabu_list = zeros(1, L);

for iter = 1:num_iterations
    best_neighbor = [];
    best_neighbor_distance = Inf;

    for i = 1:L
        for j = i+1:L
            new_solution = current_solution;
            new_solution(i) = current_solution(j);
            new_solution(j) = current_solution(i);
```

```
            if ~ismember(new_solution, tabu_list, $*$rows$*$)
                distance = calculate_total_distance(dist_matrix, new_solution);
                if distance < best_neighbor_distance
                    best_neighbor_distance = distance;
                    best_neighbor = new_solution;
                end
            end
        end
    end

    if best_neighbor_distance < best_distance
        current_solution = best_neighbor;
        best_distance = best_neighbor_distance;

        if size(tabu_list, 1) >= tabu_length
            tabu_list(1, :) = [];
        end
        tabu_list = [tabu_list; best_neighbor];

        if best_distance < calculate_total_distance(dist_matrix, best_solution)
            best_solution = best_neighbor;
        end
    end
end

% 在最优解中添加起始点并计算总距离
best_solution = [best_solution]; % 添加出发点
best_distance = calculate_total_distance(dist_matrix, best_solution);

disp($*$最优路径: $*$);
disp(best_solution);
disp([$*$最短总路程: $*$, num2str(best_distance)]);
bestall_solution(o,1) = best_distance;
 end
Newbest = min(bestall_solution);
%%

% 计算路径总距离的函数
function total_distance = calculate_total_distance(dist_matrix, solution)
    total_distance = 0;
    num_points = length(solution);
    for i = 1:num_points-1
        total_distance = total_distance + dist_matrix(solution(i), solution(i+1));
    end
    total_distance = total_distance + dist_matrix(solution(num_points), solution(1)); %
        返回起始点
```

```
end
```

附录 六　筛选满足均衡率的哈密顿回路和补集

```
%%
clc;close all;clear

PointHamilton = xlsread($*$Hamilton3.csv$*$);
load("Coordinate_point.mat")

Coordinate_point = data;
clear data

load($*$Cargoneedpreday.mat$*$)
Coordinate_point(:,4) = data(:,2);
clear data

% 创建新的空数组来存储符合要求的行数据

% valid_rows=[];
H0=[];
% 循环遍历PointHamilton的每一行
for i = 1:size(PointHamilton, 1)
    % 提取出现的站点号（假设PointHamilton的每一行表示一组站点组合）
    current_combination = PointHamilton(i, :);

    % 初始化总需求
    total_demand = 0;

    % 遍历当前组合中的站点号
    for j = 1:length(current_combination)
        % 提取站点号
        current_point = current_combination(j);

        % 使用站点号从Coordinate_point中检索相应的站点需求
        % 假设Coordinate_point的第一列是站点号，第四列是需求
        idx = find(Coordinate_point(:, 1) == current_point);

        % 如果找到了站点，累加需求
        if ~isempty(idx)
            total_demand = total_demand + Coordinate_point(idx, 4);
        end
    end

    % 检查总需求是否满足条件（大于500且小于246）
```

```
    disp($*$排列方式$*$)
    disp(i)
    disp($*$的总需求为$*$)
    disp(total_demand)
    disp($*$-----------------$*$)

   if total_demand < 500 && total_demand > 246
       % 将当前行数据存储到valid_rows中
       valid_rows =PointHamilton(i,:) ;
       H0=[H0;valid_rows];
    end


end
clear i j total_demand valid_rows current_point current_combination idx
%%
% 假设 H0 是一个包含站点排列组合的矩阵，Coordinate_point 是一个包含站点需求的矩阵，第一列是站点号
% 创建一个空的 H1 矩阵来存储未出现的站点号
PointStation = Coordinate_point(:,1)$*$;
H1 = nan(size(H0, 1), 14); % 初始化H1为NaN

for i = 1:size(H0, 1)

current_combination = H0(i, :);
nan_indices = isnan(current_combination);
current_combination(nan_indices) = [];

% 从Coordinate_point的第一列中查找第一行的元素
elements_found = ismember(PointStation,current_combination );

% 创建H1变量来存储不在Coordinate_point中出现的元素
valid_rows = PointStation(~elements_found);
%H1(i) =valid_rows;
  H1(i, 1:numel(valid_rows)) = valid_rows;
end
clear current_combination nan_indices current_combination i
```

附录 七　计算哈密顿回路和禁忌搜索最优补集路径距离

```
clc;close all;clear

load($*$Ha059.mat$*$)
load($*$Ha159.mat$*$)
load("Coordinate_point.mat")
```

```
Pointdata = data;
clear data
load($*$Cargoneedpreday.mat$*$)
Pointdata(:,4) = data(:,2);
clear data
load($*$Roadconnect.mat$*$)
Roadconnect=Roadconnectrelation(2:end,2:end);
% Roadconnect(5,9)=0;%灵敏度分析增加道路不连通下
clear Roadconnectrelation;
load("Coordinate_point.mat")
% 数据预处理，保存为 waypoints
waypoints = data;
%clear data
load($*$alldistance59.mat$*$)
% 距离矩阵存储各个点之间的距离
num_points = size(waypoints, 1);
dist_matrix = zeros(num_points, num_points);
for i = 1:num_points
    for j = 1:num_points
        dist_matrix(i, j) = sqrt((waypoints(i, 2) - waypoints(j, 2))^2 + (waypoints(i, 3) -
            waypoints(j, 3))^2);
    end
end
clear i
n = size(dist_matrix, 1);

% 根据连通关系矩阵设置初始距离矩阵
for i = 1:n
    for j = 1:n
        if Roadconnect(i, j) == 1
            dist_matrix(i, j) = dist_matrix(i, j); % 设置为实际距离
        else
            dist_matrix(i, j) = inf; % 设置为无穷大表示不可达
        end
    end
end
for i=1:n
    dist_matrix(i,i)=0;
end
% Floyd-Warshall算法
for k = 1:n
    for i = 1:n
        for j = 1:n
            if dist_matrix(i, k) + dist_matrix(k, j) < dist_matrix(i, j)
                dist_matrix(i, j) = dist_matrix(i, k) + dist_matrix(k, j);
            end
        end
```

```
    end
end
clear i j k n
%%
Area1 = [];
for k = 1:length(H0(:,1))
order = H0(k,:);
nan_indices = isnan(order);
order(nan_indices) = [];
% 计算总路程
total_distance = 0;
for i = 1:length(order)-1
    point1 = data(order(:,i), 2:3);
    point2 = data(order(:,i+1), 2:3);
    distance = norm(point2 - point1);
    total_distance = total_distance + distance;
end

disp([$*$Total distance: $*$, num2str(total_distance)]);
Area1(k) = total_distance;

clear order
end
Area1 = Area1$*$;
Tsst = min(Area1);
disp(Tsst)

%%
clear H0 H1 i k Area1
Hdata = xlsread($*$Hamilton2.csv$*$);
H0 = [];
H1 = [];
for i = 1:2:length(Hdata(:,1))
h0=Hdata(i,:);
H0=[H0;h0];
end
clear i h0
for i = 2:2:length(Hdata(:,1))
h1=Hdata(i,:);
H1=[H1;h1];
end
clear h1 i

Area1 = [];
Area2 = [];
for k = 1:length(H0(:,1))
orderh0 = H0(k,:);
```

```
nan_indicesh0 = isnan(orderh0);
orderh0(nan_indicesh0) = [];
% 计算总路程
h0_distance = 0;

for i = 1:length(orderh0)-1
    point1 = data(orderh0(:,i), 2:3);
    point2 = data(orderh0(:,i+1), 2:3);
    h0distance = norm(point2 - point1);
    h0_distance = h0_distance + h0distance;

end

disp([$*$H0 distance: $*$, num2str(h0_distance)]);
Area1(k) = h0_distance;
clear order
end
Area1 = Area1$*$;
Tsst = min(Area1);
disp(Tsst)
clear k i
%%
for k = 1:length(H0(:,1))
orderh1 = H1(k,:);
nan_indicesh1 = isnan(orderh1);
orderh1(nan_indicesh1) = [];
% 计算总路程
h1_distance = 0;

for i = 1:length(orderh1)-1
    point3 = data(orderh1(:,i), 2:3);
    point4 = data(orderh1(:,i+1), 2:3);
    h1distance = norm(point4 - point3);
    h1_distance = h1_distance + h1distance;
end
disp([$*$H1 distance: $*$, num2str(h1_distance)]);
Area2(k) = h1_distance;
clear order
end
Area2 = Area2$*$;
```

附录 八　计算哈密顿回路补集的最优巡航路径

```
clc;close all;clear

load($*$Ha0.mat$*$)
load($*$Ha1.mat$*$)
load("Coordinate_point.mat")
Pointdata = data;
clear data
load($*$Cargoneedpreday.mat$*$)
Pointdata(:,4) = data(:,2);
clear data
load($*$Roadconnect.mat$*$)
Roadconnect=Roadconnectrelation(2:end,2:end);



clear Roadconnectrelation;
load("Coordinate_point.mat")
% 数据预处理，保存为 waypoints
waypoints = data;
clear data

%% 道路不连通性
 %Roadconnect(5,9)=0;%灵敏度分析增加道路不连通下
%%
% 距离矩阵存储各个点之间的距离
num_points = size(waypoints, 1);
dist_matrix = zeros(num_points, num_points);
for i = 1:num_points
    for j = 1:num_points
        dist_matrix(i, j) = sqrt((waypoints(i, 2) - waypoints(j, 2))^2 + (waypoints(i, 3) -
            waypoints(j, 3))^2);
    end
end
clear i
n = size(dist_matrix, 1);

% 根据连通关系矩阵设置初始距离矩阵
for i = 1:n
    for j = 1:n
        if Roadconnect(i, j) == 1
            dist_matrix(i, j) = dist_matrix(i, j); % 设置为实际距离
        else
            dist_matrix(i, j) = inf; % 设置为无穷大表示不可达
        end
    end
end
for i=1:n
    dist_matrix(i,i)=0;
```

```
end
% Floyd-Warshall算法
for k = 1:n
    for i = 1:n
        for j = 1:n
            if dist_matrix(i, k) + dist_matrix(k, j) < dist_matrix(i, j)
                dist_matrix(i, j) = dist_matrix(i, k) + dist_matrix(k, j);
            end
        end
    end
end
clear i j k n


% data = pointsAboveLine;
% data = pointsBelowLine;
num_iterations = 1000; % 迭代次数
tabu_length = 10;   % 禁忌表长度

remaining_points = setdiff(1:num_points, 5);
%% 迭代
Allbestdistance = [];
%for m = 1:length(H1(:,1))
for m = 834
data = H1(m,:);
nan_indices = isnan(data);
data(nan_indices) = [];
randdata = length(data(1,:))-1;
for o = 1 :10


%    idp = data(1,1);
%    idx = find(data(1,: ) == idp);
%    elements_found = ismember(data,idp);
%
%    newdata = data(~elements_found);
    current_solution=[5,data,5];

    L = length(current_solution(1,:));
    best_solution = current_solution;
    best_distance = calculate_total_distance(dist_matrix, current_solution);

%%
tabu_list = zeros(1, L);

for iter = 1:num_iterations
    best_neighbor = [];
```

```
    best_neighbor_distance = Inf;

    for i = 1:L
        for j = i+1:L
            new_solution = current_solution;
            new_solution(i) = current_solution(j);
            new_solution(j) = current_solution(i);

            if ~ismember(new_solution, tabu_list, $*$rows$*$)
                distance = calculate_total_distance(dist_matrix, new_solution);
                if distance < best_neighbor_distance
                    best_neighbor_distance = distance;
                    best_neighbor = new_solution;
                end
            end
        end
    end

    if best_neighbor_distance < best_distance
        current_solution = best_neighbor;
        best_distance = best_neighbor_distance;

        if size(tabu_list, 1) >= tabu_length
            tabu_list(1, :) = [];
        end
        tabu_list = [tabu_list; best_neighbor];

        if best_distance < calculate_total_distance(dist_matrix, best_solution)
            best_solution = best_neighbor;
        end
    end
end

% 在最优解中添加起始点并计算总距离
best_solution = [best_solution]; % 添加出发点
best_distance = calculate_total_distance(dist_matrix, best_solution);

disp($*$最优路径: $*$);
disp(best_solution);
disp([$*$最短总路程: $*$, num2str(best_distance)]);
bestall_solution(o,1) = best_distance;
disp($*$----------$*$)

end
disp($*$下一组哈密顿闭环最短为$*$)
Allbestdistance(m,1) = min(bestall_solution);
end
```

```
%% 自定义计算距离

% 计算路径总距离的函数
function total_distance = calculate_total_distance(dist_matrix, solution)
    total_distance = 0;
    num_points = length(solution);
    for i = 1:num_points-1
        total_distance = total_distance + dist_matrix(solution(i), solution(i+1));
    end
    total_distance = total_distance + dist_matrix(solution(num_points), solution(1)); %
        返回起始点
end
```