报名序号： 2664

赛题题目：B 题 5G 网络环境下应急物资配送问题

基于遗传算法的车辆与无人机 VRP 问题求解

摘　要

随着 5G 与无人机技术的发展，使用“车辆+无人机”的组合配送模式逐渐成为应急物资配送的高效率方案。但在这一策略中，无人机因承载能力和电量限制需要经常性与车辆会合以更换电池和储备物资。如何合理规划车辆与无人机在路线网中的行进方案是一个经典的 VRP 问题，而对于这一问题，我们使用启发式方法求解。

对于问题一，在仅使用车的情况下，这是一个单一车辆的 VRP 问题。注意到节点需求总量满足车载最大容量，故抽象为节点遍历要求路径最短的问题。对这一问题我们将其抽象为一个 0-1 规划并使用遗传算法的策略进行求解，最终解得最优路径长为 582km。

对于问题二，引入无人机后车与无人机的路径由若干子路径组合而成，我们通过引入对无人机的 0-1 规划与节点运输变量后得到双对象 VRP 问题，并同样应用遗传算法进行了求解。结果表明，车辆仅经过...个节点而无人机经过...个节点，总的最短时间为 6.24h。

对于问题三，将车载最大容量下调意味着需要运输两次才能遍历完所有节点。这里每一次遍历都可以视作一个单独的 VRP 问题，但两次的运输节点划分策略对整体寻优起到关键作用。我们基于动态规划将其抽象为一个集合二分的搜索问题，通过负载约束等对状态空间进行剪枝并成功找到了最优策略，花费时间仅需 6.72h。

对于问题四，将路径图进行了扩展，而继续沿用问题三的思想，发现货物需要运送四次，于是将节点集划分为四个子集并进行 VRP，通过搜索的方式确定两个中心位置后对四分 VRP 进行状态空间的搜索。这里通过节点度的大小进行优先搜索，得到最优运送时间为 7.927h。

模型应用启发式算法进行求解，所得结果在保证速度的同时较为准确地对问题进行了合理解答。而利用动态规划对状态空间的搜索与剪枝，则避免了直接搜索带来的不必要时间与空间开销。综合评估，通过启发式算法对 VRP 问题的求解是奏效的。

关键词：VRP 问题；启发式算法；遗传算法；动态规划

一、 问题重述

1.1 问题背景

为了应对一些重特大突发事件造成的道路阻断、损坏、封闭等意想不到的情况，伴随着科技水平的提升及 5G 网络的逐渐普及，无人机的应用越来越广泛，“配送车辆+无人机”的配送模式已经渐渐成为一种新的有效的配送方式。配送车辆对某地点进行配送的同时，无人机也可向周围可行的地点进行配送，并于配送完成后返回配送车辆重新装载物资、更换电池。

无人机和车辆都可以一次运输多个地点，但在无人机电量耗尽之前二者必须在同一地点会合让无人机更换电池和重新装载物资。配送车辆和无人机合作完成所有地点应急物资配送任务，返回到出发地点，此时称为完成一次整体配送。

而在整体配送任务中配送时间是主要的研究因素。按照配送车辆和无人机从出发开始至全部返回到出发地点的时间来计算。在配送过程中，不考虑配送车辆及无人机装卸物资的时间，同时不考虑配送车辆和无人机在各个配送点的停留时间。

1.2 问题提出

为了尽快完成物资配送任务，请根据附件所给数据解决以下几个问题：

1. 附件 1 给出了所有地点的连接关系网络。若目前所有应急物资集中在地点 9，唯一的配送车辆的最大载重量为 1000 千克，采取仅使用配送车辆建立完成一次整体配送的数学模型，并给出最优方案。

2. 附件 2 在附件 1 的基础上增加了无人机专用路线。应急物资仍然集中在第 9 个地点，配送车辆的最大载重量为 1000 千克，采取“配送车辆+无人机”的配送模式。请结合附件 2，建立完成一次整体配送的数学模型，并给出最优方案。

3. 若问题 2 中的配送车辆的最大载重量为 500 千克，其他条件不变。请结合附件 2，建立完成一次整体配送的数学模型，并给出最优方案。

4. 附件 3 的网络有 30 个地点，计划设置两个应急物资集中地点，若配送车辆的最大载重量为 500 千克，采取“配送车辆+无人机”的配送模式。请结合附件 3，建立完成一次整体配送的数学模型，确定两个应急物资集中地点的最佳位置。

二、 问题分析

2.1 问题一的分析

从问题一中给定的网络图可以看出，例如 2、3、5 构成一个闭合回路，4 构成了一个悬挂点等。这样的网络如果将其抽象为一个 TSP 问题是无解的。为此，我们认为可以重复遍历节点，将其抽象为一个 VRP 问题进行求解。将其抽象为图遍历问题，优化目标为总路程，因为在仅有一辆车的情况下优化路程即优化了时间。由于 14 个节点的需求求和小于 1000 所以是符合要求的。VRP 问题的求解可以使用遗传算法。

2.2 问题二的分析

问题二在问题一的基础上增加了无人机，使得车辆不需要遍历所有节点即可完成任务。但在每个子段上的时间是取无人机和车辆之间最大的一个。问题被抽象为一个二阶段 VRP，使用遗传算法同样可以对问题进行设计求解，也可以使用动态规划的方式决策但搜索空间较大。

2.3 问题三的分析

问题三将配送车的容量限制为 500，这使得运输的轮数必须两轮才能满足要求。本质上可以将问题抽象为一个节点的完全二分问题，使得对节点集的划分，两个子集上的 VRP 之和最优。在问题二中我们可以得到给定交通网络的情况下 VRP 最优值的求解，那么这个问题本质上的难点在于节点划分策略的搜索。需要注意的是，网络整体的节点在两轮运输中都可以遍历，但二分策略提供每一轮需要运输的节点序号。

2.4 问题四的分析

问题四相对而言更加开放，将图的节点数增加到 30 的同时设置两个送货点。那么问题实际上同样可以抽象为节点划分问题，由于无人机和车的容量限制可知需要运输四次才能运输完毕，但两个送货点允许了我们可以同时运输两轮。将节点划分为两对集合，每对集合中的两个有共同的起点。四个节点的 VRP 解之和最优即可完成要求。

三、 模型假设

针对这些问题，我们的模型假设如下：

1. 观察到车辆和无人机只有速度定义，我们假设车辆和无人机都是匀速运动状态并且一方先到需要等待另一方。

2. 假设每个节点都可以重复遍历但只能运输一次，不失一般性，规定只有在运输后的节点才可以再次遍历。

3. 在配送过程中，不考虑配送车辆及无人机装卸物资的时间，同时不考虑配送车辆和无人机在各个配送点的停留时间。

4. 在配送过程中每个节点只允许一次性配送完毕。

四、 符号说明

表 1　各变量符号说明及解释

符号	说明
x	车辆经过的路径决策变量
y	无人机经过的路径决策变量
S	节点集合
D	路径网络的邻接矩阵
T	总体时间
$D_{i\to k}$	从 i 到 k 这一子路径的路程
$S_{i,k}$	从 i 到 k 这一子路径的节点路径
$P(y)$	被无人机运输的节点
v_y	无人机速度
$P(x)$	节点被车辆运输的节点
v_x	车辆速度
F	节点需求
W	无人机负载
$\underset{x,y}{VRP}(S)$	针对节点集 S 的 VRP 过程
$Circle(S)$	判断节点集 S 是否存在回路

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 模型的建立

图 1 为问题的路径图，可以看到在这一个问题中 TSP 问题是无解的，原因在于节点 5 和节点 6 必定会重复遍历。而作为 TSP 问题的扩展，VRP 问题能对此类问题进行良好的求解。

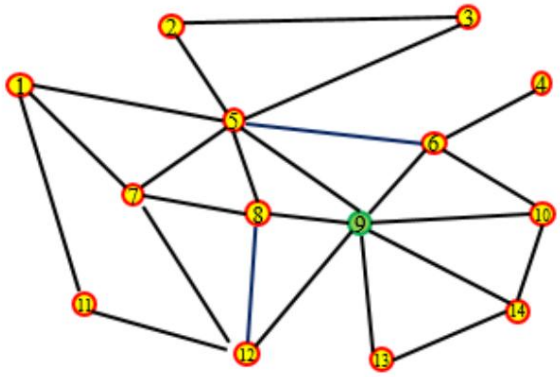

图 1. 问题一的网络结构

车辆路径问题最早由 Dantzig 和 Ramser 于 1959 年首次提出，是运筹学中一个经典问题[1]。VRP 问题主要研究物流配送中的车辆路径规划问题，是当今物流行业中的基础问题。在 VRP 问题中，假设有一个供求关系系统，车辆从仓库取货，配送到若干个顾客处。车辆受到载重量的约束，需要组织适当的行车路线，在顾客的需求得到满足的基础上，使代价函数最小。代价函数根据问题不同而不同，常见的有车辆总运行时间最小，车辆总运行路径最短等[2]。

由于问题一中所有节点的需求量之和小于 1000，故暂时不需要考虑负载问题。

对于这一 VRP 问题，我们将其抽象为一个 0-1 规划问题求解。决策变量为节点 i 与节点 j 之间的路径是否经过，若经过则变量取值为 1，不经过则变量取值为 0。决策目标为使得总路径最短，那么，问题可以抽象为：

$$\min_{x} \ F = D^T X$$

$$s.t. \begin{cases} \sum_{i=1}^{14} x_{i,j} = 1 \\ \sum_{j=1}^{14} x_{j,i} = 1 \\ x_{i,j} = x_{j,i} \\ x_{i,j} = 0, 1 \end{cases} \tag{1}$$

而对于无法直接到达的节点对，我们定义其距离为无穷大，这时对应的 x 必须取 0。问题被抽象为一个 0-1 规划问题。

离散优化的 0-1 问题求解策略有很多，但即使是节点数较小的 VRP，由于组合爆炸现象其解空间仍然可能较大[3]。对于此类 NP 完全问题，使用传统穷举策略以及优化算法例如匈牙利法、蒙特卡洛方法已经难以求解因为问题的变量维数太高。随着进化计算与群体智能算法的成熟，有越来越多的智能算法被用于函数优化问题的求解，其中，遗

传算法是一类借鉴生物界自然选择和自然遗传机制的随机搜索算法，通过模拟生物的遗传、变异等自然现象搜索函数极值[4]。其主要特点是直接对结构对象进行操作，不存在求导和函数连续性的限定；具有更好的全局寻优能力；不需要确定的规则就能自动获取和指导优化的搜索空间，自适应地调整搜索方向[5,6]。

遗传算法借鉴了生物学的概念，首先需要对问题进行编码，通常是将函数编码为二进制代码以后，随机产生初始种群作为初始解。随后是遗传算法的核心操作之一——选择，通常选择首先要计算出个体的适应度，根据适应度不同来采取不同选择方法进行选择，常用方法有适应度比例法、期望值法、排位次法、轮盘赌法等[7]。

在自然界中，基因的突变与染色体的交叉组合是常见现象，这里也需要在选择以后按照一定的概率发生突变和组合。不断重复上述操作直到收敛，得到的解即最优。遗传算法基本思想如图 2 所示：

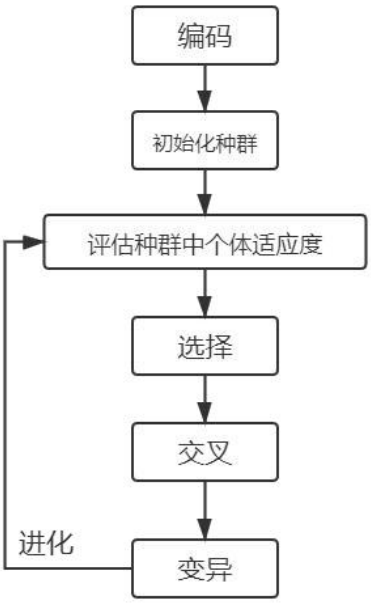

图 2　遗传算法的基本思想

设计思路如下：

Step 1: 设置优化算法基本参数，设置最大路线数 N，设置惩罚系数。

Step 2: 生成初始个体，每个个体长度为 14，随机生成路径和每个点所在的路径编号，并检验是否存在没有经过点的路线（检验时排除第 9 个点），否则重新分配，这样得到了一组路径样本。

Step 3: 计算初始个体的目标函数，有了前面的距离矩阵，可以计算每个路径下的总路程或总时间。同时计算每条路线的运载量（需求量之和）

Step 4: 遗传算法的迭代，进行交叉和变异操作。交叉过程就是 rand<交叉率时，随便选择两个点的顺序交换或者路线编号交换，但交叉和变异都需要经过检验。

Step 5: 最后输出最优路线及路径。

5.1.2 模型的求解

通过程序设计的方式，我们对遗传算法中变量进行了合理编码、变异与交叉。规定对角线上的值为 0，而无法连通的节点之间直接距离定义为 1e+8。设置迭代轮数为 3000 轮，算法求得的最短距离随迭代轮数变化如图 3 所示：

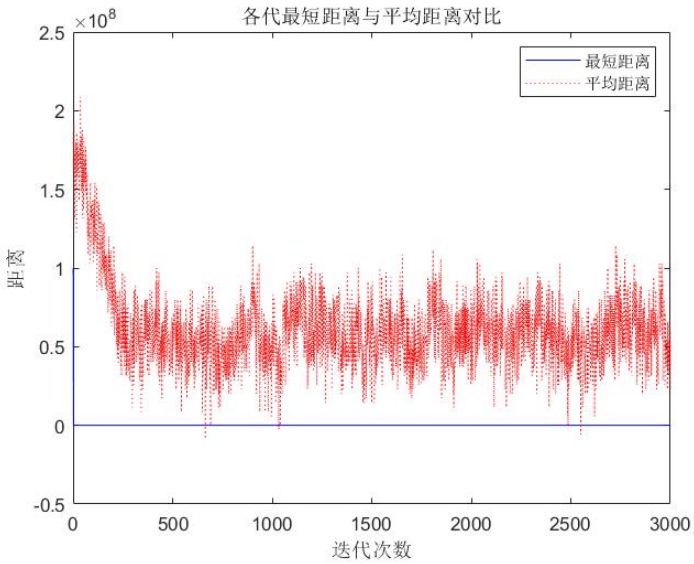

图 3. 遗传算法迭代曲线

从图中可以看到，最短距离随着迭代轮数的增加逐渐趋向收敛，而在每一代中的最优解已经没有那么高的数量级，很可能是一个最优解或近似最优解。通过遗传算法最终的迭代结果，整体的路径如图 4 所示：

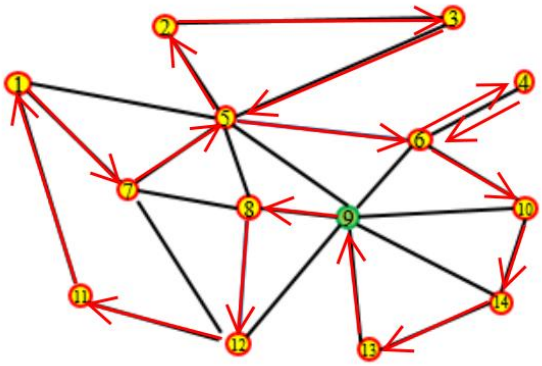

图 4. 路径图

根据我们的算法，算得的最短路径如图 4 所示，配送车经过的路径依次为 9-8-12-11-1-7-5-2-3-5-6-4-6-10-14-13-9,其中地点 6 经过了两次，这是由于地点 4 只有 4-6 一条路径连接所致，计算得到配送车经过的总路径为 582km，配送车花费的最短总时长为 11.64h.

综合分析，模型求解时间仅需 27s，在保证速度较快的同时我们也以枚举的方法列出了所有的可能，发现确实是最优解，说明使用启发式方法对于这一规模下的 VRP 问题求解是完备的，模型应用初步成功。

5.2 问题二模型的建立与求解

5.2.1 模型的建立

问题二在问题一的基础上增加了无人机，也新增了仅有无人机能行动的路径。如图 5 所示，虚线为仅有无人机能运行的路径。

经过路径增广以后我们通过计算问题一和问题二中节点的平均度数和聚类系数，发现问题二形成的增广网络在拓扑结构上更具有稳定性，节点度也更高，使得节点形成

TSP 形式下的回路是可行的。其中，节点 5 度数为 7，节点 9 度数为 7，节点 8 度数为 5，节点 6 度数也为 5，是需要注意的四个节点。

在问题一的基础上我们引入无人机的矩阵 y，使得：

$$\min_{x,y} T$$

$$s.t.\begin{cases} \sum_{i=1}^{14} x_{i,j} \leqslant 1 \\ \sum_{j=1}^{14} x_{j,i} \leqslant 1 \\ \sum_{i=1}^{14} y_{i,j} \leqslant 1 \\ \sum_{j=1}^{14} y_{i,j} \leqslant 1 \\ x_{i,j} = x_{j,i},\ y_{i,j} = y_{j,i} \\ x_{i,j}, y_{i,j} = 0, 1 \\ S(x) \cup S(y) = S \end{cases} \tag{2}$$

这里我们没有再选择路径为优化目标，而是换用时间为优化目标。因为总路径最短并不一定能保证时间最短。若对于某一段子路径，配送车和无人机都在节点 i 出发，在节点 k 会合，那么这一段的时间优化为：

$$\begin{gathered} T_i = \max\left\{\frac{D_{i\to k}(x)}{v_x}, \frac{D_{i\to k}(y)}{v_y}\right\} \\ S_{i,k}(x) \cup S_{i,k}(y) = \{i,k\} \end{gathered} \tag{3}$$

在这一段子路径中，无人机和配送车除了起点和终点没有共有节点，需要使得它们的经过时间取更大的一方。而对于对应的路径，我们即取对应子路径 u,v 及其对应的 0-1 决策变量 x,y，使得：

$$\begin{aligned} D_{i\to k}(x) &= \sum_{(u,v)} x_{u,v} D_{u,v},\ (u,v) \in S_{i,k}(x) \\ D_{i\to k}(y) &= \sum_{(u,v)} y_{u,v} D_{u,v},\ (u,v) \in S_{i,k}(y) \end{aligned} \tag{4}$$

在子路径上车辆和无人机的路径规划方式如图 6 所示，可以看到，对于给定子路径来讲车辆和无人机路径是没有除起始点和终点外的公共点的。

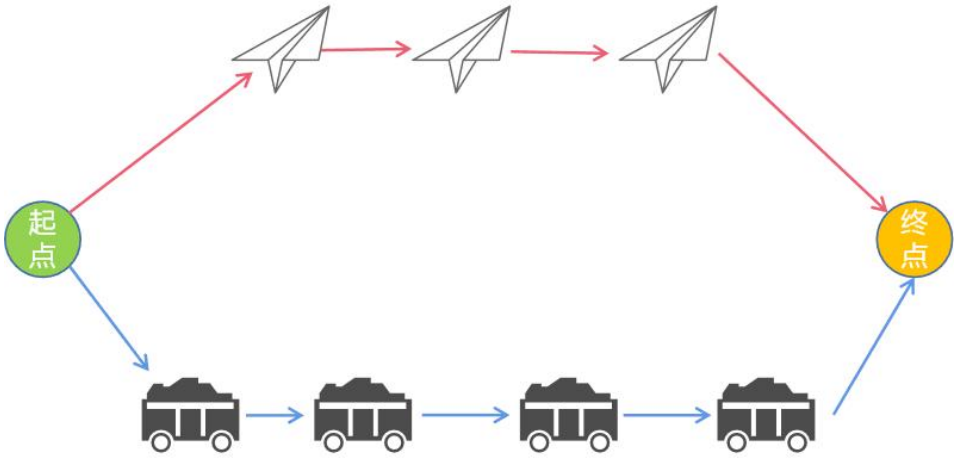

图 6. 车辆和无人机路径图

那么决策目标是所有子路径的最优时间求和。即：

$$T=\sum_{i\in N}\max\left\{\frac{D_{i\to i+1}(x)}{v_x},\ \frac{D_{i\to i+1}(y)}{v_y}\right\},\quad N\subseteq S \tag{5}$$

其中 N 为所有子路径划分的会合点集。

而在问题中两个小于的约束条件无法保证每个节点都能运送到。为了保证这一点，我们引入两个 14 维的 0-1 向量，表示节点由无人机配送还是由车辆配送。这两个向量满足条件：

$$\forall i\leqslant 14,\ P_i(x),\ P_i(y)=\ 0,1$$

$$s.t.\begin{cases}\sum_{i=1}^{14}x_{i,j}=P_i(x)\\ \sum_{j=1}^{14}x_{j,i}=P_i(x)\\ \sum_{i=1}^{14}y_{i,j}=P_i(y)\\ \sum_{j=1}^{14}y_{i,j}=P_j(y)\\ P(x)+P(y)=1\\ P(x)\cdot P(y)=0\end{cases} \tag{6}$$

除此以外，向量 P 还需要满足负载条件，但我们已经知道车载负载是一定可以满足的，于是考虑无人机在每个子路径上的负载条件：

$$\sum_{i\in S_{i,k}(y)}P_i(y)F_i\leqslant W_y \tag{7}$$

于是，我们建立起了一个较为完备的带约束的优化模型。对于这一模型，约束条件过多且有两个对象，所以在变量编码过程中需要重新进行约束，所需要的变量量也是原有的两倍以上。对于这一问题，使用遗传算法的求解将远远优于传统优化。

5.2.2 模型的求解

根据上述模型，我们对遗传算法中的变量编码进行重构，引入新的对象构成双对象遗传，所得到的迭代曲线如图 7 所示：

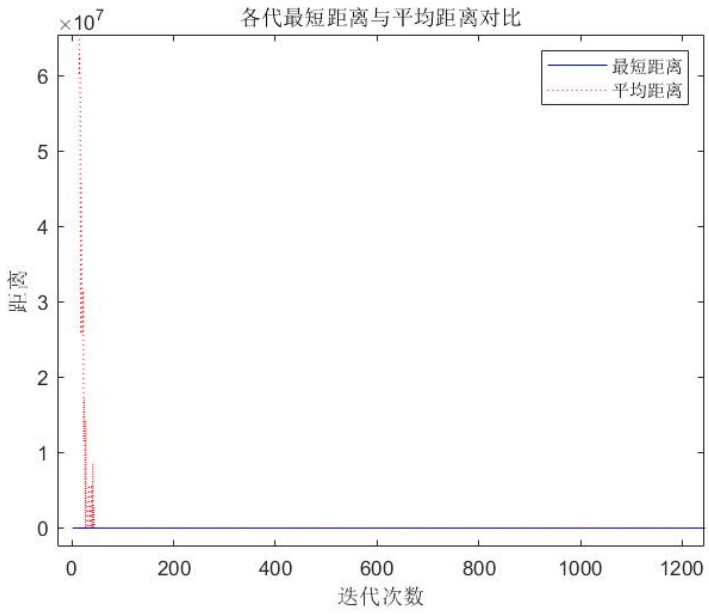

图 7. 问题二的遗传迭代

可以发现相比于问题一，问题二在经过路径增广以后其解更明显收敛更快，迅速得到了最优解。这是因为节点度的平均值增高提高了网络稳定性与连通性。

经过计算，我们发现当无人机集合 S={1,3,4,11,12,13,14}时，我们算得的路径是最短的，路径图如图 8 所示，配送车经过的路径依次为 9-8-7-5-2-5-6-10-9，配送车分别依次在地点 9,8,7,6,10 释放无人机，最终配送车和无人机同时从地点 10 出发回到初始点 9。从图中我们发现配送车走过的路径基本都处于内圈当中，结合附件 2 所给数据，我们认为这是由于处于图形边缘的点{1,3,4,11,12,13,14}能由配送车经过的路径较少，而这些点的当日物资需求量又不是很大，因而最短路径会解得这样的结果。配送车和无人机完成整体配送的最短总时长为 6.24h。

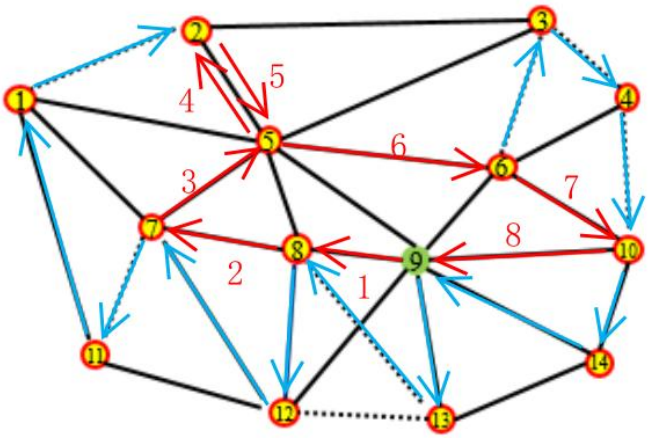

图 8. 问题二路径图

5.3 问题三模型的建立与求解

5.3.1 模型的建立

问题三对车辆的负载进行了进一步约束，将原有的 1000 变为 500。在问题二当中我们已经提出了对 14 节点的求解方案，将其进行封装并记作 VRP(S)。我们的封装结果可以描述为：

$$T = \underset{x,y}{VRP}(S) \tag{8}$$

而事实上，对于任意一个节点子集都可以使用上述过程得到一个最优解。现在由于车辆负载能力减半，我们需要对节点进行二分化处理。主要是将 S 划分为两个子集，每个子集代表两轮运输过程中运输的节点。两轮过程中运输的节点除了节点 9 不可以重复，于是我们有：

$$\begin{aligned} S_1 \cup S_2 &= S \\ S_1 \cap S_2 &= \{9\} \end{aligned} \tag{9}$$

将集合表示为向量的形式，那么，原有条件的无人机负载条件不变，但对车载和每个节点的负载向量而言，需要满足：

$$\begin{aligned} S_1(P(x)+P(y))F &\leqslant 500 \\ S_2(P(x)+P(y))F &\leqslant 500 \end{aligned} \tag{10}$$

那么这一问题本质上是将第二问的 VRP 问题二分为：

$$\begin{aligned} \min\ T(S_1,S_2) &= \underset{x,y}{VRP}(S_1) + \underset{x,y}{VRP}(S_2) \\ s.t.\ &\begin{cases} S_1 \cup S_2 = S \\ S_1 \cap S_2 = \{9\} \end{cases} \end{aligned} \tag{11}$$

并且对于这一节点划分形成的两个子集中，两个子集都能形成一条闭合回路。即：能够保证子集中每个节点都能与该集合中其它至少一个节点相接。若判定一个子图的闭合回路函数为 Circle，那么需要：

$$Circle(S_1) = Circle(S_2) \geqslant 1 \tag{12}$$

对于 VRP 的子问题我们已经用遗传算法进行了封装。现在核心问题为 S 的二分策略与最优状态的搜索。这一问题通常最纯粹的方法为遍历式搜索，而我们为了节约状态空间，选择采用动态规划的方式动态剪枝。

我们将节点按照负载量从大到小排序，若对于某一节点 i 而言，此前节点的划分序列已经明确，那么在保证节点负载的情况下，由于 P(x)+P(y)=1，后续问题会被二分为：

$$\begin{aligned} T(S_1,S_2) &= T(S_1,S_2|\{S_{i-1}\},S_i \in S_1) + T(S_1,S_2|\{S_{i-1}\},S_i \in S_2) \\ S_1 F &\leqslant 500 \\ S_2 F &\leqslant 500 \end{aligned} \tag{13}$$

基于此，遍历到一个最优策略并根据负载条件进行剪枝即可循环利用遗传算法对问题进行求解。

二分策略的示意图如图 9 所示：

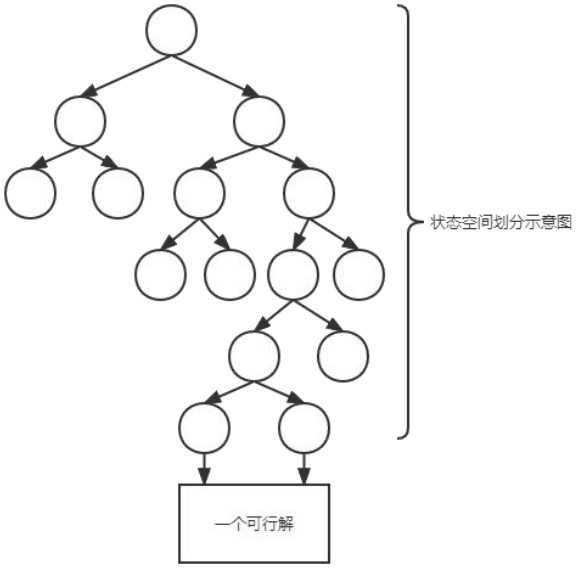

图 9. 二分策略

5.3.2 模型的求解

经过遗传迭代和规划剪枝计算，我们发现当无人机经过点的集合取为S={1,3,4,11,12,13,14}时，我们迭代得到的路径是最短的，路径图如图 10 所示，配送车经过的路径依次是 9-8-7-5-2-5-9-6-10-14-9，配送车分别依次在地点 9,8,7,6 释放无人机，最终无人机在配送车的携带下从地点 10 途径 14 返回初始点。我们发现配送车经过的路径和第二问求得的结果有较多的重叠，只是第二问 5-6 部分的路径变为 5-9-6，这也是受到配送车最大载重量的限制导致必须返程补充物资所致。通过和第二问相比较，这也反映了结果的可靠性，最终配送车和无人机完成一次整体配送所用总时长为 6.72h

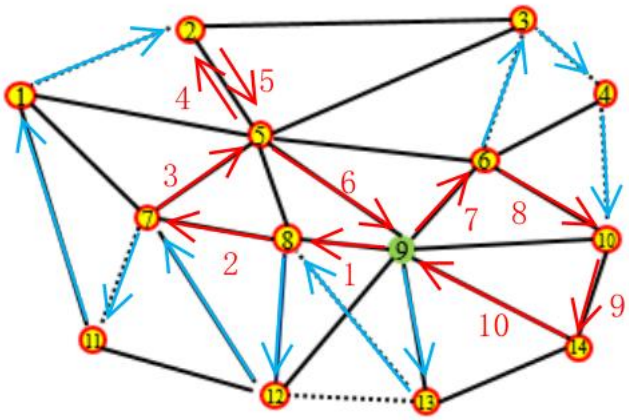

图 10. 问题三的配送路径

5.4 模型四的建立与求解

5.4.1 模型的建立

问题四是对问题三的扩展，将节点扩展到 30 个节点，如图 11 所示：

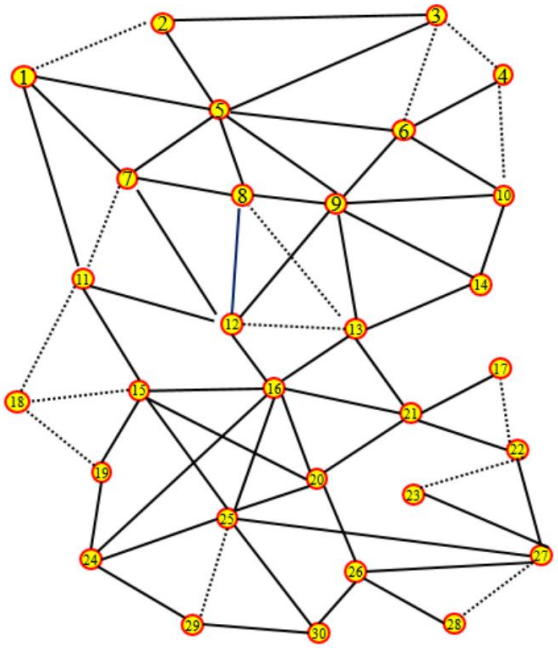

图 11. 问题四的网络

问题四的网络可以在节点 11、12、13 下方进行二分从而划分为两个子网络，子网络的稳定性都是较高的。但整体的最优划分策略仍然需要量化方法得到。

那么同样的，类比问题三的想法，我们经过统计，所有节点的负载至少需要运送四次才能运送完成。我们仍然以节点划分的策略解决这一问题，将节点划分为四个子集，若运送点分别为 u,v，则问题形式为：

$$\min\ T(S_{11},S_{12},S_{21},S_{22}) = \underset{x,y}{VRP}(S_{11}) + \underset{x,y}{VRP}(S_{12}) + \underset{x,y}{VRP}(S_{21}) + \underset{x,y}{VRP}(S_{22})$$

$$s.t.\begin{cases} S_{11}\cup S_{12}\cup S_{21}\cup S_{22} = S \\ S_{11}\cap S_{12} = \{u\} \\ S_{21}\cap S_{22} = \{v\} \\ S_{1i}\cap S_{2j} = \varnothing \end{cases} \tag{14}$$

四个子集的并集能够包括所有 30 个节点，而对同一运输中心的两轮运输其交集只能包括该中心，而对于不同中心的每一轮运输都不会产生交集，也就是重复运输的点。同样地，问题仍然需要满足负载条件限制。

此时我们仍然通过动态规划的方式对问题进行求解。首先是中心的选择策略搜索，我们将增广图中每个节点的度从大到小进行排序，按照从大到小的方式遍历节点序列(u,v)，而对于每一对(u,v)，其他节点的划分会处于一个四分的状态。同样根据回路约束和负载约束对问题进行剪枝。

5.4.2 模型的求解

为了提高运算速度，我们优先选择度数较大的地点作为可能出现的物资配送点，经过计算得出不同的物资配送中心完成配送的最短时间分布如表 1 所示

表 1. 配送物资的时间分布

集中点	5	8	9	20	25
最短时间	7.173h	7.227h	6.72h	7.927h	8.58h
覆盖地点	{1,2,3,4,5,6,7,8,9,10,11,12,13,14}			{15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30}	

由此，我们得出配送车和无人机完成整体配送所需时间最短对应的路径图如图 12 所示。通过观察我们发现配送中心覆盖的地点从中间被分割成了两部分，通过分析附件 3 给出的数据和图形分布，我们观察到图形的上半部分和下半部分连通的度数相对较少，从中间分割恰好可以使得配送效率达到最优。9 号配送中心对应的最短时长为 6.72h，20 号配送中心对应得最短时长为 7.927h，我们选择 9 号和 20 号地点作为我们的配送中心，两个配送中心完成所有地点整体配送任务所需的最短时长为 7.927h。

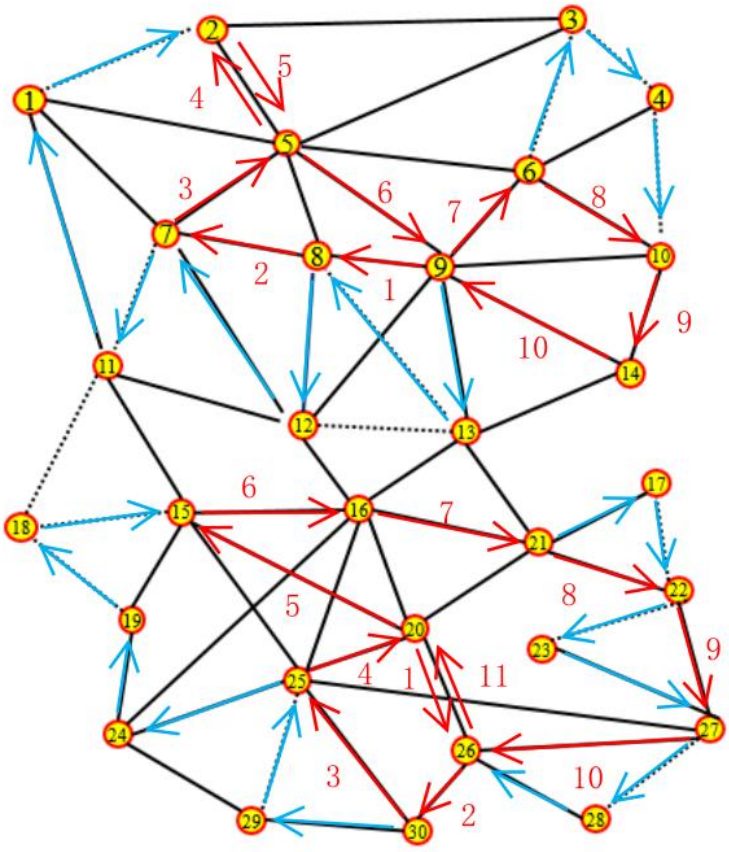

图 12. 配送路径

六、 模型的评价、改进与推广

6.1 模型的优点

该模型从运筹优化的角度出发，结合启发式的遗传算法，保证了结果的可靠性，随后我们根据一些模型评价指标评估，认为模型在本数据集上取得了较好的效果。我们认为，我们在这一系列问题中提出的模型有以下优点：

1.使用动态规划与遗传算法，结果准确可靠，模型表现好。

2.利用排序以及大量约束降低了搜索空间，有力节约了程序运行资源从而实现快速、高效的迭代与搜索。

3.通过遗传算法对 VRP 问题求解，相比于传统优化算法对节点数量增高造成的代价增长更小，在小规模问题上基本能够做到最优解。

4.对 VRP 问题的求解进行了封装，即使节点和图结构发生变化仍然能够快速迁移应用到其它问题，可移植性强。

6.2 模型的缺点

尽管模型整体表现良好，但我们认为还存在一些不足之处，例如：

1.大规模数据集上的遗传算法求解 VRP 问题结果将可能不是全局最优而是局部最优或近似最优，因为 VRP 是一个 NP 完全问题很难求解绝对最优解。

2.VRP 问题中列举的约束条件仍然过多，未能进一步进行简化。同时，变量维度引入较多也使得问题的求解更复杂，需要进一步压缩变量个数节约搜索空间。

6.3 模型的改进

该模型除了可以使用遗传算法求解以外，同样属于启发式算法框架下的粒子群算法、蚁群算法、模拟退火算法、禁忌搜索等作为更快的启发式搜索算法也可以用于尝试。另外，对于动态规划算法对网络节点划分的状态空间仍然过大，尤其是在问题四中划分为四分问题，算法的时间复杂度较大，实际上这一系列循环是可以通过更改算法结构简省的。通过将循环合并或利用空间换时间等能够提升算法的工作效率。

七、 参考文献

[1] 史春燕,黄辉.车辆路径问题:研究综述及展望[J].物流科技,2014,37(12):75-77.DOI:10.13714/j.cnki.1002-3100.2014.12.024.
[2] 方金城,张岐山.物流配送车辆路径问题(VRP)算法综述[J].沈阳工程学院学报(自然科学版),2006(04):357-360.
[3] 唐坤.车辆路径问题中的遗传算法设计[J].东华大学学报(自然科学版),2002(01):66-70.
[4] 李向阳.遗传算法求解VRP问题[J].计算机工程与设计,2004(02):271-273+276.DOI:10.16208/j.issn1000-7024.2004.02.034.
[5] 马永杰,云文霞.遗传算法研究进展[J].计算机应用研究,2012,29(04):1201-1206+1210.
[6]. 吴渝, 唐红, 刘洪涛. 网络群体智能与突现计算[M]. 科学出版社, 2012.
[7]. 李士勇，李研，林永茂. 智能优化算法与涌现计算[M]. 清华大学出版社，2020，P115-118.

附录

利用 Ortools 求解 VRP 问题 Python 代码

```
from ortools.constraint_solver import pywrapcpfrom ortools.constraint_solver import
routing_enums_pb2

# 定义这个VRP 问题的数据，使用字典结构存储
def create_data_model():
    """Creates the data for the example."""
    data = {}
    # Array of distances between locations.
    _distances = D
    data["distances"] = _distances
    data["num_locations"] = len(_distances)   # 有几个节点
    data["num_vehicles"] = 4                 # 有几辆车
    data["depot"] = 0                        # 仓库索引，我们假设所有的车辆都从同一地点出发，也就
是车场。
                                             # 或者，你可以允许车辆在任何位置启动和结束。
    return data

# 计算距离的回调函数，和之前的routing 问题一样的
def create_distance_callback(data):
    """Creates callback to return distance between points."""
    distances = data["distances"]

    def distance_callback(from_node, to_node):
        """Returns the manhattan distance between the two nodes"""
        return distances[from_node][to_node]

    return distance_callback

def add_distance_dimension(routing, distance_callback):
    """Add Global Span constraint"""
    distance = 'Distance'
    maximum_distance = 3000  # 每辆车能形式的最大距离
    routing.AddDimension(
        distance_callback,
        0,  # null slack
        maximum_distance,
        True,  # 从累积到零，意思应该和“走了这么久还剩多少汽油”差不多吧
        distance)
    distance_dimension = routing.GetDimensionOrDie(distance)
    # Try to minimize the max distance among vehicles.
    # 尽量减少车辆之间的最大距离。
    distance_dimension.SetGlobalSpanCostCoefficient(100)
```

```
# 打印每辆车的路线(访问的位置)，以及路线的距离。# 请注意，这些距离包括从仓库到路线中第一个位置的距离以及从最后一个位置返回到仓库的距离。# IndexToNode, NextVar 函数和前面的 tsp 问题是相同的意思
def print_solution(data, routing, assignment):
    """Print routes on console."""
    total_distance = 0
    for vehicle_id in range(data["num_vehicles"]):
        index = routing.Start(vehicle_id)
        plan_output = 'Route for vehicle {}:\n'.format(vehicle_id)
        route_dist = 0
        while not routing.IsEnd(index):
            node_index = routing.IndexToNode(index)
            next_node_index = routing.IndexToNode(
                assignment.Value(routing.NextVar(index)))
            route_dist += routing.GetArcCostForVehicle(node_index, next_node_index,
vehicle_id)
            plan_output += ' {0} ->'.format(node_index)
            index = assignment.Value(routing.NextVar(index))
        plan_output += ' {}\n'.format(routing.IndexToNode(index))
        plan_output += 'Distance of route: {}m\n'.format(route_dist)
        print(plan_output)
        total_distance += route_dist
    print('Total distance of all routes: {}m'.format(total_distance))
# 主函数def main():
    # 创建数据集
    data = create_data_model()

    # Create Routing Model
    # 创建路由模型
    routing = pywrapcp.RoutingModel(
        data["num_locations"],
        data["num_vehicles"],
        data["depot"])

    # 提供距离回调，以便解决程序可以计算位置之间的距离。
    distance_callback = create_distance_callback(data)
    routing.SetArcCostEvaluatorOfAllVehicles(distance_callback)

    # 添加距离维度
    add_distance_dimension(routing, distance_callback)

    # Setting first solution heuristic (cheapest addition).
    # 必须指定启发式方法来找到第一个解决方案
    search_parameters = pywrapcp.RoutingModel.DefaultSearchParameters()
    search_parameters.first_solution_strategy = (
```

```
        routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)  # pylint:
disable=no-member

    # 求解问题
    assignment = routing.SolveWithParameters(search_parameters)
    if assignment:
        print_solution(data, routing, assignment)

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

```
clc
clear
close all
data1=xlsread('附件 1.1.xlsx','问题 1 各地点之间距离（单位　公里）','B2:K11');
idx=find(isnan(data1));
data1(idx)=inf;
data=data1;
DDCS=100000;
popsize=200;
chromlength=10;
pc=0.5;
pm=0.005;
sl1=2 ;
A=zeros(popsize,chromlength);
a=zeros(chromlength,1);
newpop=initpop(popsize,chromlength);
for i=1:DDCS
       fitvalue=calfitvalue(newpop,data);
       [newpop]=selection(newpop,fitvalue);
       [newpop]=crossover(newpop,pc,sl1);
       [newpop]=mutation(newpop,pc);
         fitvalue=calfitvalue(newpop,data);
       [bestindividual,bestfit]=best(newpop,fitvalue);
     a(i)=bestfit;
       A(i,:)=bestindividual;
end
plot(a);
k=find(max(a))
A(k,:)

function[newpop]=selection(pop,fitvalue)
totalfit=sum(fitvalue);
fitvalue=fitvalue/totalfit;
```

```
fitvalue=cumsum(fitvalue);
[px,py]=size(pop);

for j=1:px
    a=rand;
    sd=fitvalue>=a;
    es=find(sd);
    newpop(j,:)=pop(es(1),:);
End

function[newpop]=crossover(pop,pc,sl1)
[px,py]=size(pop);
newpop=ones(size(pop));
for i=1:2:px-1
    if(rand<pc)
        a=randperm(py);
        a=a(1:sl1);
        for j=1:sl1
            c=pop(i,a(j));
            d=pop(i+1,a(j));
            e=pop(i,c);
            pop(i,c)=pop(i,d);
            pop(i,d)=e;
            e1=pop(i+1,c);
            pop(i+1,c)=pop(i+1,d);
            pop(i+1,d)=e1;
        end
          newpop(i,:)=pop(i,:);
        newpop(i+1,:)=pop(i+1,:);
    else
    newpop(i,:)=pop(i,:);
    newpop(i+1,:)=pop(i+1,:);
    end
end
```