2020 年武汉理工大学大学生数学建模竞赛

题 目：基于改进的蚁群算法解决车辆路径规划问题

摘 要

本文是在食品运输实际问题背景下对于车辆调度问题的研究，该文针对题目中层层递进的车辆调度问题进行建模与分析，从单车辆最短运输时间问题、单车场单型号多车辆总体调度效率问题，再到单车场多型号多车辆总体调度问题，不断修改和完善算法，从而寻找车辆调度的最佳方案。

针对问题一单车场单车辆运输最短时间问题，我们经过分析可知，该路线街道是平行于方向轴的，而且任意两点之间是互通可达的，首先用曼哈顿距离表示任意两节点之间的距离，从而将该问题转化为一个 TSP 问题来建模求解，通过传统的蚁群算法，通过 100 次的迭代，最终得到了稳定清晰的路线结果，算出最短运输距离为 116km，最短运输时间为 8.65 小时，运输费用约为 3598 元。

针对问题二求解单车场单型号车辆总体调度效率问题,我们首先通过需求分析，定义了调度效率的标准，充分考虑时间、费用、车辆承载利用率的影响，对这个车辆调度问题中的变量和路径进行抽象和建模，构建该情况下路径，距离，服务规则，工作时间的约束条件，并提炼出反映调度效率的指标公式，并计算权重进行综合评价，利用蚁群算法将 VRP 问题转化为 TSP 问题，最终得到第二问的调度方案为用 6 辆 6t 车运输货物，距离为 222km，时间为 1.39h,费用为 1105.4 元，车辆承载利用率为：86.53%。

针对第三问求解单车场多型号车辆总体调度问题，经过分析可知，该问题是在问题二的基础上增加了另外一种车型来辅助运载，我们将第二问的 VRP 模型进一步拓展为 FSMVRP 模型，通过组合优化和动态规划的方式，对多种车型在最短规划路线上的效率进行筛选，从而优化蚁群算法，最后得出总体调度效率最高的方案是：用 5 辆 4t 的车和 3 辆 6t 的车运输货物，对应的距离 224km，时间为 1.21h,费用是 935.99 元，车辆承载率 75.6%。

通过以上模型和算法的建立优化，在建立模型所需的约束条件，本文目前基本实现了对于车辆调度的多种情况的求解，且结果均满足需求基本合理。

关键词： 蚁群算法；VRP ；路径规划；车辆调度问题

目录

一、问题重述

1.1 问题背景

该问题是食品的配送问题，具体描述种中提到了配送的地点，数量，以及费用详情和工作时间约束，具体描述为 19 个不同地区的销售点和 1 个配送中心，配送中心根据需求将来规划最优路径使得以最高效率来实现调度和生产，每台运输车车每日工作 4 小时，运输车重载运费 2 元/吨公里，且假定街道方向均平行于坐标轴，任意两站点间都可以通过一次拐弯到达。那么选取怎样的运输方式以实现成本最小化是公司最为关注的。本文根据车辆的承载量、车型、运输时间、运输费用等实际条件来建立数学模型来选取最优调度方案。

1.2 问题描述

本文模型主要解决以下问题：

1.在运输车载重为 100t,平均速度为 40 公里/小时，每个销售点需要用 20 分钟的时间下货，空载费用 0.6 元/公里的条件下，它送完所有食品并回到仓库，最少需要多少时间？

2.在运输车载重为 6t，平均速度为 50 公里 / 小时，每个销售点需要用 5 分钟的时间下货，空载费用 0.4 元/公里的条件下；要使它们送完所有食品并回到仓库，运输车应如何调度使总体调度效率最高？

3.在有两种车型分别为载重量为 4 吨、6 吨两种运输车，空载费用分别为 0.2、0.4 元/公里，平均速度为 50 公里 / 小时，每个销售点需要用 5 分钟的时间下货的条件下，如何安排车辆数和调度方案？

二、问题分析

针对问题一，通过分析我们可以清楚的发现，在该题中，仅有一辆配送车而配送路线点任意两点都是互通可达的，且配送车的容量远远大于所有的需求之和，因此第一问是一个典型的 TSP 问题，因其车辆移动的时候只能是通过横竖的方式来移动，所以我们可以将这个问题中各个点的距离抽象出来成空间中曼哈顿距离，然后建立出每个点的正确距离，然后再来利用蚁群算法对该问题进行迭代和运算，最终得到迭代稳定的结果并画出迭代稳定的结果图用于验证和计算。

针对问题二，通过分析我们可以了解到，问题一和问题二之间最大的差别是，问题一中只有一辆大车，需要配送所有的配送点，而问题二中在汽车配送的数量上没有进行限制，但是配送的车辆的容量减小了，从而使得我们需要考虑配送过程中的费用和时间，以及车辆容量利用率的 CVRP 问题最终得出一个最佳的方案。

针对问题三，第三问是在第二问的基础上的一种多车型单车场配送问题的优化，根据题意我们发现，第三问就是在第二问的基础上增加了一种新的车型，所以说第二问是第三问的简化形式，因此我们可以在第二问的基础上对算法进行优化利用动态规划的思想对路径进行迭代和运算，从而得出将部分解空间用新车型进行替换和优化，从而得到最终的结果多车优化方案。

三、模型假设

结合本题的实际情况，为确保模型求解的准确性和合理性，排除一些因素的干扰，提出以下假设：

1. 每辆车的载重不会影响速度。
2. 运输车在行驶过程中不考虑红灯、堵车等因素，其速度始终保持不变。

3. 不考虑运输车加速、制动的速度变化及时间影响。
4. 该食品公司可以提供足够多的车辆。
5. 不考虑每辆车派出去所产生的人工费用（驾驶费）以及装卸货等成本。
6. 车辆的折旧费用忽略不计。

四、符号说明

符号	意义
(x_i, y_i)	配送点 i 的坐标
s_{ij}	地区 i 和地区 j 之间的曼哈顿距离
q_i	地区 i 的需求量
m	蚂蚁的数量
n	地区的总数量
l_{ij}	地区 i 到地区 j 的路径
η_{ij}	边l_{ij}上的能见度
$\tau_{ij}(t)$	t 时刻下在边l_{ij}的信息素。
$\Delta\tau_{ij}$	边l_{ij}的信息素增量
$\Delta\tau_{ij}^{k}$	在一次迭代中，第k只蚂蚁在边l_{ij}的留下的信息素
$P_{ij}^{k}(t)$	在第 t 时刻第 k 只蚂蚁从城市 i 转移到 j 城市的概率
tabu_k	蚂蚁 k 的城市禁忌表
α	信息素重要程度因子
β	期望启发因子
ρ	信息素的蒸发系数
Q	信息素释放总量
x_{ij}	表示 i 到 j 的路径
V_0	表示配送点的集合
E	表示赋权图中边的集合
$minC$	最小费用
minS	最短路径
y_{ik}	客户 i 的产品配送由车辆 k 来完成
e	配送效率
Q_m	第 m 车型的最大配送运载量
n_m	第 m 种车的数量
x_{ijkm}	m 类型的第 k 辆车经过路径 i, j
y_{ikm}	客户 i 的产品配送由 m 车型的第 k 辆车来完成
δ	车载利用率

五、模型建立求解

5.1 问题一的模型

针对问题一的题目要求，我们可以看到，该问题中确定了大型运输车的数量，载重，平均车速，而街道方向均为平行于坐标轴的方向，而且满足任意站点都是互相通达的，那么该问题则可以视为一个 TSP 问题通过蚁群算法来进行建模求解。

5.1.1 模型的建立

在问题一中，由于平均速度已知，而题目要求解的是需要通过最少的时间来完成这个问题，那么其实就可以转化为求出最短的运输路径，同时我们还可以了解到的是这个问题中街道是沿着平行于坐标轴的方向，那么该图中，任意两点之间的距离则需要通过曼哈顿距离表示：

$$s_{ij} = |x_i - x_j| + |y_i - y_j|$$

同时，在我们看到汽车的载重量是 100 吨，而经过计算$\sum_i^l q_i = 31.15 < 100$，因此这一辆汽车是完全满足送货要求的，且任意送货点之间是具有互通性的。

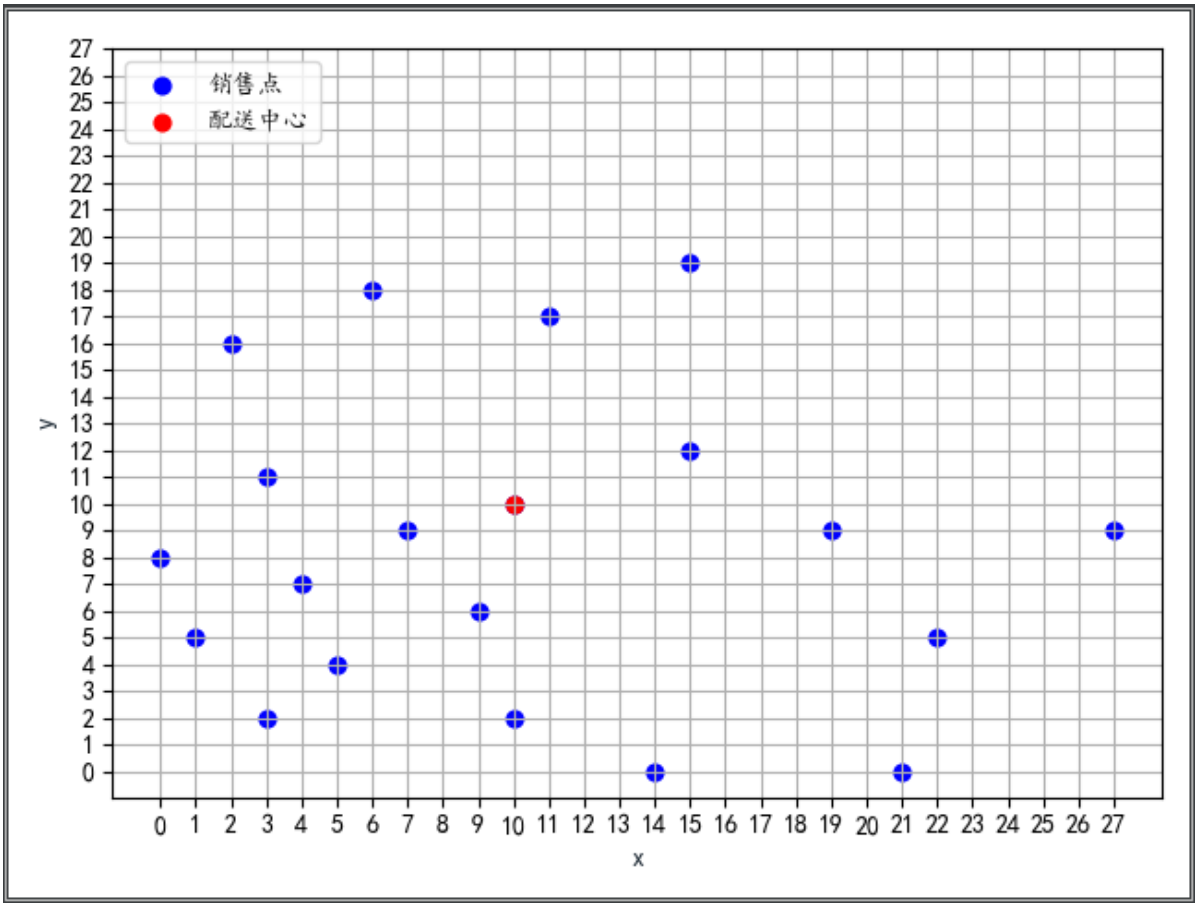

图 1.1 销售点和配送中心分布图

通过曼哈顿距离计算得到了任意两点之间的距离之后，下面我们将利用蚁群算法对上述问题进行解析。

为不失一般性，这里设置蚂蚁的数量为 m，各点的数量为 n 并设置η_{ij}表示为

边l_{ij}上的能见度，在这边主要定义为s_{ij}的倒数，$\tau_{ij}(t)$定义为 t 时刻下在边l_{ij}的信息素。

$\Delta\tau_{ij}$在一次迭代中，边l_{ij}的信息素增量，$\Delta\tau_{ij}^{k}$表示在一次迭代中，第 k 只蚂蚁在边l_{ij}的留下的信息素，$P_{ij}^{k}(t)$表示在第 t 时刻第 k 只蚂蚁从城市 i 转移到 j 城市的概率。

下面是需要设置的一些超参数，α表示信息启发因子表示信息的相对重要程度，β表示期望启发因子，ρ表示信息素的蒸发系数，用禁忌表tabu_k表示蚂蚁 k 在经过地区 i 后，就将该地区划入到自己的禁忌表中从而使得下一次不能选择这个城市。

根据蚁群算法的常规假设，我们可以建立如下建模计算过程：

1. 首先初始化模型参数，将 m 只蚂蚁放到 n 个点上，同时将该点加入蚂蚁的禁忌表中tabu_k，并讲各个曼哈顿距离对应的边信息素设置为同一常数。
2. 每只蚂蚁通过各边的信息素和能见度独立选择下一个地区，并将转移成功的地区存入到禁忌表了，其转移概率公式为

$$P_{ij}^{k}(t)=\begin{cases}\dfrac{[\tau_{ij}(t)]^{\alpha}\cdot[\eta_{\mathrm{ij}}(t)]^{\beta}}{\sum_{x\in J_k(i)}[\tau_{is}(t)]^{\alpha}\cdot[\eta_{is}(t)]^{\beta}}, if\ j\in J_k(i)\\ 0\qquad\qquad\qquad\qquad\qquad\qquad, else\end{cases}$$

3. 当所有城市都加入了禁忌表则构成了一次迭代可行解
4. 当完成一次迭代以后，我们利用下列信息素的更新规则进行更新。

$$\tau_{ij}(t+1)=(1-\rho)\cdot\tau_{ij}(t)+\Delta\tau_{ij}$$

$$\Delta\tau_{ij}=\sum_{k=1}^{n}\Delta\tau_{ij}^{k}$$

对于信息素增量的求解，我们采用蚁周系统系统，从而更加快速的得到全局最优解。

$$\Delta\tau_{ij}^{k}=\begin{cases}Q/L_k, if\ antk\ across\ l_{ij}\\ 0\end{cases}$$

得到全局最优路径以后然后再进行费用计算，得到最终的费用结果。

5.1.2 模型求解

针对问题一模型求解时主要时分为一下几个基本环节，主要是初始化参数，构建解空间，更新信息素，迭代返回，获得稳定结果。

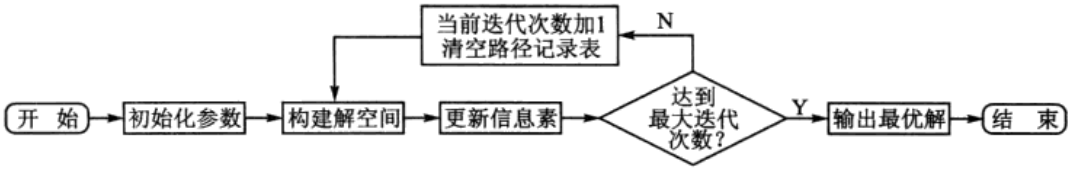

图 1.2 蚁群算法求解流程

1. 初始化参数

我们在计算之初,需要对相关的参数进行初始化，如蚁群规模(蚂蚁数量)m、信息素重要程度因子α、启发函数重要程度因子β、信息素挥发因子 ρ 、信息素释

放总量 Q、最大迭代次数 NC_max 进行相关设置

```
m=16;     %% m 蚂蚁个数
Alpha=1;  %% Alpha 表征信息素重要程度的参数
Beta=5;   %% Beta 表征启发式因子重要程度的参数
Rho=0.5; %% Rho 信息素蒸发系数
NC_max=100; %%最大迭代次数
Q=100;          %%信息素增加强度系数

C=[
3 2;
1 5;
5 4;
4 7;
0 8;
3 11;
7 9;
9 6;
10 2;
14 0;
2 16;
6 18;
11 17;
16 12;
```

图 1.3 蚁群算法参数设定

2. 构建解空间

将各个蚂蚁随机地置于不同出发点,对每个蚂蚁 k(k= 1,2,.,m),利用上述转移方程进行迭代和计算其下一个待访问的城市,直到所有蚂蚁访问完所有的城市，从而获得一个可行解的路径长度。

3. 更新信息素

更新信息素主要是通过上述的更新迭代规则方式来进行计算和迭代计算各个蚂蚁经过的路径长度 L,(k=1,2,.. ,m),记录当前迭代次数中的最优解(最短路径)。同时,根据对各个城市连接路径上的信息素浓度进行更新。

4. 结果输出

最终我们通过在跌打 100 次后得到了稳定的输出结果

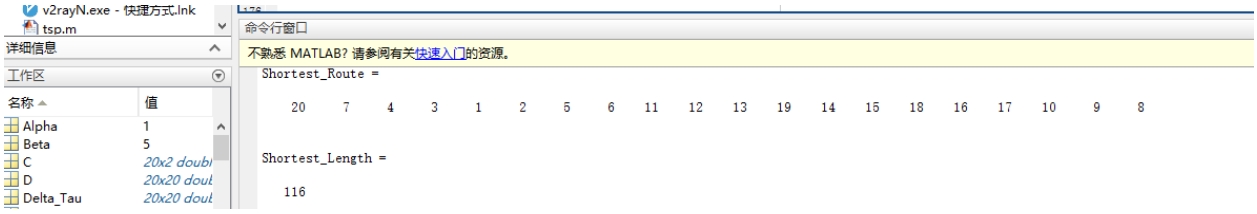

图 1.4 matlab 蚁群算法计算结果

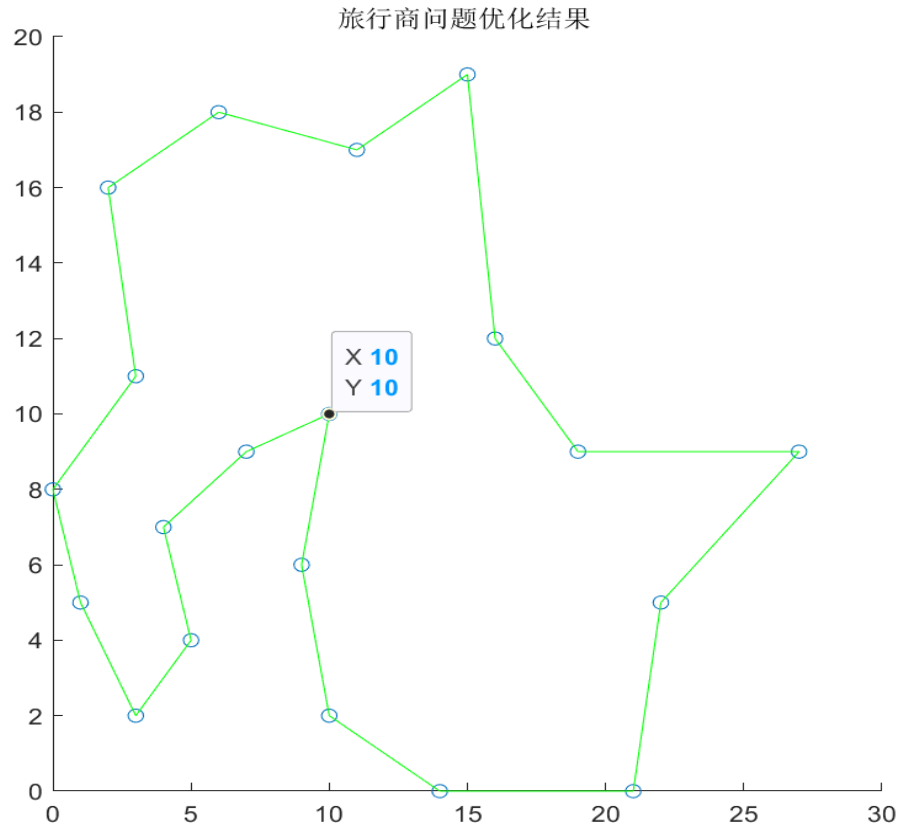

图 1.5 优化路线图

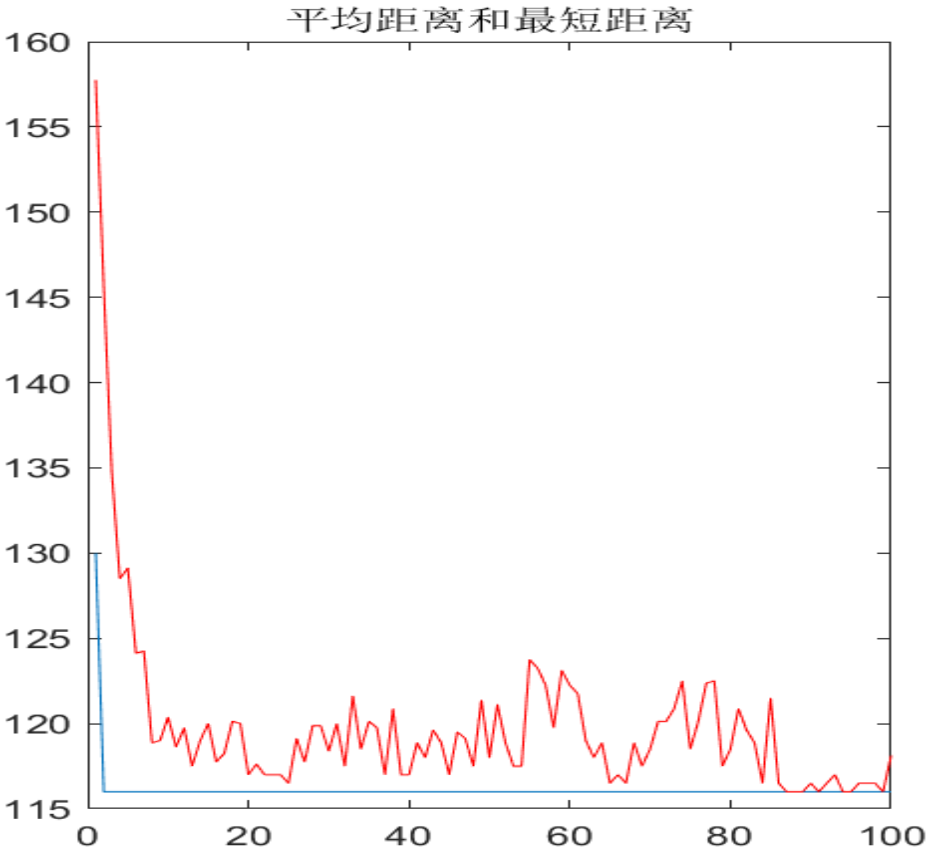

图 1.6 迭代过程的平均距离和最短距离变化

5.1.3 结论

通过上述蚁群算法的迭代过程计算，我们可以知道，该运输路线图为

表 1 问题一车载路线

序号	20	7	4	3	1	5	6	11	12	13
需求量/t	0	1.2	1.2	1.5	2.5	0.85	1.3	2.5	0.8	1.5
累计距离/km	0	4	9	13	17	26	32	38	44	50
序号	19	14	15	18	16	17	10	9	8	20
需求量/t	1.6	0.9	1.4	1.4	2.2	1.8	1.8	1.4	3.3	0
累计距离/km	56	63	70	78	87	93	100	106	111	116

,其运输距离为 116km，同时运算距离解决，通过运算得到了其运输费用约为 3598，运输时间为 8.65 小时。

5.2 问题二的模型

在问题二中，站点数量一定且有不同的货物需求，配送中心向站点提供货物且保证满足客户需求，与第一问有所不同的是载重量有明显限制，由一个同类型车队负责运输货物，组织适当的行车路线，达到运输车总体调度最高的目的，是经典的车辆路径问题(VRP)，可以将其转换为 TSP 问题，再利用蚁群算法求解[8]。通过对对蚁群算法的分析，针对有容量约束的车辆路径问题，提出蚁群算法的改进算法，为快速、有效地求解有容量约束的车辆路径问题开辟了途径[5]。在本题中约束时间以及每个站点只由一辆车配送，综合考虑车的数量以及车辆承载利用率，定义调度效率E，将多目标问题转换为单目标问题，在优先考虑时间的前提下，使得费用最小，以此得到总体调度最优的方案。

5.2.1 模型的建立

对问题二中的站点重新编号，假设原编号记为a,修正后的编号为$i=-a+20$,即配送中心的编号为 0，站点编号为 1,2,…,19，均以 i 来表示。G(V,E)为全联通赋权图，$\{0\}\cup V_0$ 表示所有节点的集合,0 表示配送中心,$V_0=\{1,2,\cdots,19\}$表示站点集合；$E=\{(i,j)\ i,j\in V\}$表示边集合，s_{ij}表示从站点 i 到站点 j 的距离,$k=\{1,2,\ldots,n\}$表示配送中心可用车辆集合，为同类型车，第 i 个站点的需求量q_i（i=1,2,…,19）,M 表示费用，t 表示时间，车辆承载利用率δ。

$$x_{ij}=\begin{cases}1, & \text{车辆 k 经过路径}(i,j)\\ 0, & \textit{否则}\end{cases}$$

$$y_{ik}=\begin{cases}1, & \text{客户 i 的产品配送由车辆 k 来完成}\\ 0, & \textit{否则}\end{cases}$$

VRP 的数学模型[4]：从一个配送中心向多个客户送货，配送中心位置确定不变，每个客户需求量一定，车辆载重量不能 超过载荷量。假定配送中心能满足所有

客户的需求，要求合理安排车辆的配送路线，使完成所有配送任务的总路径长度最小。假设条件如下:

1）在每条配送路线上的客户需求量的总和不能超过车辆的载荷量；

2）每个客户的需求由一辆送货车配送，且只能配送一次；

3）每次配送任务完成后，配送车辆必须回到配送中心。

目标函数：

有关所求费用公式如下所示：

$$minC = \sum_{k=1}^{n}\sum_{i=0}^{19}\sum_{j=0}^{19} s_{ij}x_{ijk}(w - y_{ik}q_i) * 2 + \sum_{k=1}^{n}\sum_{i=0}^{19} s_{i0}x_{i0k} * 0.4$$

$$\mathrm{minS} = \sum_{k=1}^{n}\sum_{i=0}^{19}\sum_{j=0}^{19} s_{ij}x_{ijk}$$

初始时：$w = \sum_{i=0}^{19} y_{ik}\, q_i$　每次循环后:$w = \sum_{i=0}^{19} y_{ik}\, q_k - y_{ik}q_i$带入上式

目标函数约束条件：

$$\sum_{\mathrm{i=0}}^{19} \mathrm{y_{ik}\, q_i} \le 6 \qquad (1)$$

$$\sum_{\mathrm{k=1}}^{\mathrm{n}} \mathrm{y_{ik}} = 1 \qquad (2)$$

$$\sum_{\mathrm{i=0}}^{19} \mathrm{x_{ijk}} = \mathrm{y_{jk}} \qquad \forall \mathrm{k,j} = 1,2,\dots,19 \qquad (3)$$

$$\sum_{\mathrm{j=0}}^{19} \mathrm{x_{ijk}} = \mathrm{y_{ik}} \qquad \forall \mathrm{k,i} = 1,2,\dots,19 \qquad (4)$$

$$\sum_{\mathrm{i,j\in U\times U}} \mathrm{x_{ijk}} \le |\mathrm{U}| - 1 \quad 2 \le |\mathrm{U}| \le 19 \qquad (5)$$

$$\sum_{\mathrm{k=1}}^{\mathrm{n}} \mathrm{y_{0k}} = \mathrm{n} \qquad (6)$$

$$\frac{\sum_{\mathrm{i=0}}^{19}\sum_{\mathrm{j=0}}^{19} \mathrm{s_{ij}x_{ijk}}}{50} + \frac{5\sum_{\mathrm{i=0}}^{19} \mathrm{y_{ik}}}{60} \le 4 \qquad (7)$$

$$\mathrm{x_{ijk}} = 0 \text{ 或 } 1 \quad \forall \mathrm{i,j,k} \qquad (8)$$

$$\mathrm{y_{ik}} = 0 \text{ 或 } 1 \quad \forall \mathrm{i,k} \qquad (9)$$

（1）式表示表示配送车辆载重约束，配送车辆所服务的客户需求不大于车辆载重约束；（2）式表示每个站点只能由一辆车配送完成；（3）（4）式表示变量$\mathrm{x_{ijk}}$与$\mathrm{y_{ik}}$之间满足的关系，即保证客户被配送车辆服务时，一定存在与其相连的路径

（5）式表示未避免出现与配送中心相分离的线路，需要支路消去约束条件；（6）式表示 n 辆车都是由配送中心 0 出发；（7）每辆车运作时间不超过 4h；（8）（9）式表示x_{ijk}与y_{ik}的取值范围。

其次还要再充分考虑车辆承载利用率δ，定义δ如下：

$$\delta = \frac{\sum_{k=1}^{n}\sum_{i=1}^{19} y_{ik} q_i}{6n}$$

由于调度效率与时间和费用也有关，因此，我们定义调度效率的公式如下所示：

$$e = M * t + minC + 1 - \delta$$

其中，M表示一个相当大的一个数，通过在目标函数中引入参数 M，能够保证算法在求解车辆路径问题时以车辆数为第一优化目标，以车辆旅行费用作为第二优化目标，也就是一个具有较少车辆数的解比一个具有较大车辆数但是较小车辆旅行距离的解好,其次还充分考虑承载利用率。约束条件表示每个顾客点被一辆车辆服务。

我们认为，调度效率首要考虑的就是时间，因此时间在调度效率中为首要考虑因素，根据最短时间对应的状态再去求所得费用，还要综合考虑车辆承载利用率。

6.2.2 模型求解

显然，由建立的模型可知，该题是一个明显的 VRP。考虑到实际情况中存在生鲜配送等需要明显考虑时间因素大于费用因素的情况以及车辆速度不变的情况下，我们仍然只需要将该问题转化为 TSP 问题求出最短路径，然后根据蚁群算法求出对应路径的最短时间即可。由于车辆数目并不确定，且模型中有很多约束条件需要满足，我们考虑将原有的 TSP 需要求解遍历每个销售点构成的大的哈密顿通路，转化成一个具有多个哈密顿通路的哈密顿图，而因此哈密顿通路的数量即为运输车辆的数量。

本题转化后的 TSP 问题不同于传统的用蚁群算法求解的 TSP。传统用蚁群算法求解的 TSP 需要在不同的地点上随机放置，然后使得蚂蚁能够在迭代过程中通过信息素找到最优路径。然而，本题本质上依然是一个 VRP，并且有很多 TSP 中没有的约束条件，所以我们借鉴了一些求解 VRP 的蚁群算法的特点来对传统里求 TSP 蚁群算法来进行改进，我们的蚁群算法实现流程如下：

1. 蚁群参数状态初始化。包括但不限于最大迭代次数、蚂蚁数量、信息素重要程度、启发式因子重要程度、信息蒸发系数、信息素增强系数等参数。
2. 确定每次迭代里蚂蚁下一步可以前往的地点集合。
3. 在这些地点中，根据如下的概率公式选择下一个地点：

$$P_{ij} = \begin{cases} T_{ij}^{\alpha} * \dfrac{\left(\frac{1}{T_{ij}}\right)^{\beta}}{\sum_{s \in allowed} T_{ij}^{\alpha} * \left(\frac{1}{T_{ij}}\right)^{\beta}} & \text{如果} j \in allowed \\ 0 & \text{否则} \end{cases}$$

4. 然后根据下一个地点的条件（如需求量、是否已经被遍历等）确定是否能够满足对应的约束条件，不能满足回到起点，补充载重。

5. 状态更新。
6. 在所有蚂蚁走出的路径上，根据蚂蚁释放的信息素，随着迭代次数增加，不断更新其路径上的信息素。
7. 进行下一次迭代。

最终求得的哈密顿图结果如图所示：

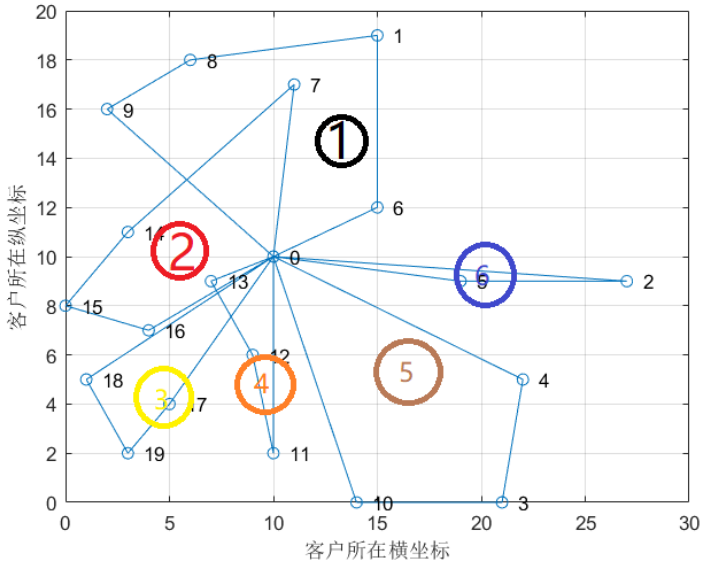

图 2.1 配送路线在哈密顿下的路线图

根据图中的标识可以得知，图中共有 6 个哈密顿通路，即可以安排 6 辆 6 吨车运输来跑出最短路径长度。与此同时，可以得到蚁群算法的迭代结果如图所示：

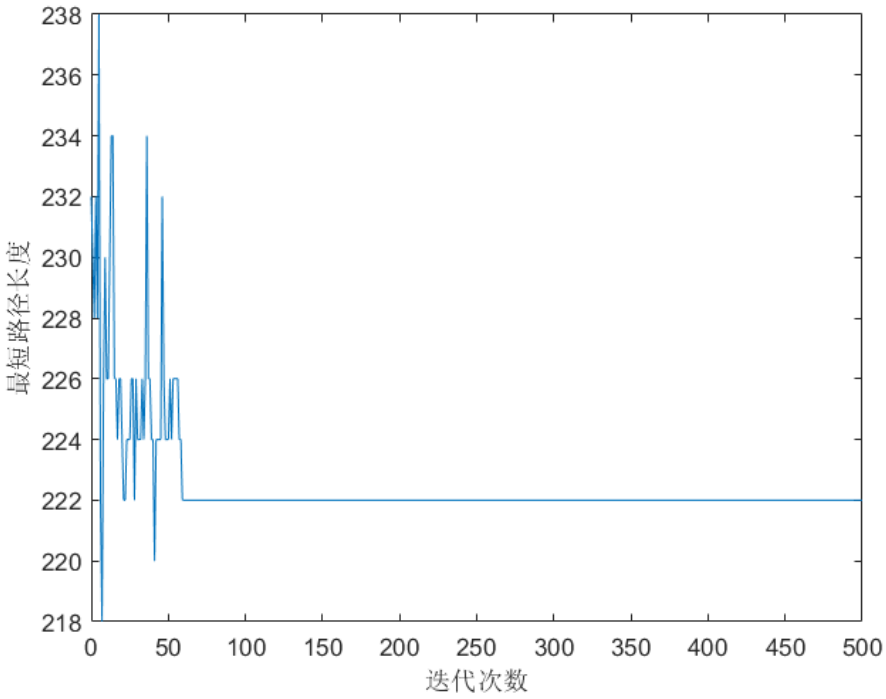

图 2.2 蚁群算法迭代结果

可以看到，在 500 次迭代后，路径长度的结果趋于稳定，即$minL = 222$km，

对应的时间根据速度和时间的公式可得约为t =1.39s，根据前面所建立的数学模型，我们可以得到$minM = 1105.4$元。

5.2.3 结论

根据前面有关调度效率的公式，我们可以得知，在当时间达到最小时，调度效率就得到最高，此时可得$e = 1.39M + 1105.4$。

与此对应的调度方案如下表格所示：

表 2 问题二运输方案

车辆编号	载重	路线
1	5.9	1->14->13->12->1
2	5.8	1->7->2->9->10->1
3	6	1->18->20->19->1
4	4.85	1->17->16->15->8->1
5	5.8	1->11->4->5->1
6	2.8	1->3->6->1

由上面的表格我们可以得知，我们的车辆承载利用率δ为

$$\delta = 86.53\%$$

注意：这里的路线编号是配送中心的编号重定义为 1 了，该路径的编号是倒序排列的

5.3 问题三的模型

第三问与第二问相比，最大的区别在于第三问中有两种不同承载量的车，分别为可以载重 6t 和 4t 的车，是一种多车型多约束多目标的路径规划问题多车型问题。多车型问题是对单车型运输问题的一种扩展，对于多车型问题，通过约束转换，转化为每辆车的 TSP 问题[错误!未找到引用源。]。因此我们在第二问的基础上增加车型的变量，将第二问的模型进一步延伸。

5.3.1 模型的建立

定义载重 4t 的车的车型编号为 1,载重 6t 的车的车型编号为 2，以 m(m=1,2)来表示，各车型的载重量为Q_m ，各各车型的数量分别为n_1，n_2 以n_m来表示。定义变量如下：

$$x_{ijkm} = \begin{cases} 1, & \text{m 类型的第 k 辆车经过路径(i,j)} \\ 0, & \text{否则} \end{cases}$$

$$y_{ikm} = \begin{cases} 1, & \text{客户 i 的产品配送由 m 车型的第 k 辆车来完成} \\ 0, & \text{否则} \end{cases}$$

VRP 的数学模型为

目标函数：

$$minC = \sum_{m}^{12}\sum_{k=1}^{n}\sum_{i=0}^{19}\sum_{j=0}^{19} s_{ij}x_{ijkm}(w - y_{ikm}q_i) * 2 + \sum_{k=1}^{n_1}\sum_{i=0}^{19} s_{i0}x_{i0k1} * 0.4 + \sum_{k=1}^{n_2}\sum_{i=0}^{19} s_{i0}x_{i0k2}$$

初始时：$w = \sum_{i=0}^{19} y_{ikm}\, q_i$　每次循环后:$w = \sum_{i=0}^{19} y_{ikm}\, q_k - y_{ik}q_i$带入上式

目标函数约束条件：

$$\sum_{i=0}^{19} y_{ikm}\, q_i \le Q_m \quad (1)$$

$$\sum_{m=1}^{n2} y_{ikm} = 1 \quad (2)$$

$$\sum_{i=0}^{19} x_{ijkm} = y_{jkm} \qquad \forall k, m, j = 1,2,\dots,19 \quad (3)$$

$$\sum_{j=0}^{19} x_{ijk} = y_{ikm} \qquad \forall k, m, i = 1,2,\dots,19 \quad (4)$$

$$\sum_{i,j \in U \times U} x_{ijk} \le |U| - 1 \quad 2 \le |U| \le 19 \quad (5)$$

$$\sum_{k=1}^{n_1} y_{0k1} = n_1 \qquad \sum_{k=1}^{n_2} y_{0k2} = n_2 \quad (6)$$

$$\frac{\sum_{m=1}^{2}\sum_{k}^{n_m}\sum_{i=0}^{19}\sum_{j=0}^{19} s_{ij} x_{ijkm}}{50} + \frac{5\sum_{i=0}^{19} y_{ikm}}{60} \le 4 \quad (7)$$

$x_{ijkm} = 0$ 或 $1 \quad \forall i, j, k, m$ (8)

$y_{ikm} = 0$ 或 $1 \quad \forall i, k, m$ (9)

(1)式表示表示配送车辆载重约束，配送车辆所服务的客户需求不大于车辆载重约束；(2)式表示每个站点只能由一辆车配送完成；(3)(4)式表示变量x_{ijkm}与y_{ikm}之间满足的关系，即保证客户被配送车辆服务时，一定存在与其相连的路径；(5)式表示未避免出现与配送中心相分离的线路，需要支路消去约束条件；(6)式表示n_m辆车都是由配送中心 0 出发；(7)每辆车的运作时间不超过 4h；(8)(9)式表示x_{ijkm}与y_{ikm}的取值范围。

其次车辆承载利用率δ'，定义δ'如下：

$$\delta' = \frac{\sum_{m}^{2}\sum_{k}^{n_m}\sum_{i=0}^{19} y_{ikm} q_i}{4n_1 + 6n_2}$$

定义调度效率的公式如下所示：

$$e' = M * t + minC + 1 - \delta'$$

5.3.2 模型求解

问题三实际上是一个在问题二的基础上基于车辆的解空间的动态规划问题。而且，两种车之间只有空载费用不同，其余条件均相同。实际上，问题二就是问题三的一种极端情况，即问题二是全部用 6 吨车去运输，4 吨车数量为 0，而问题三是可能要同时使用 4 吨运输车和 6 吨运输车。那么，再考虑另一种极端情

况，即全部用 4 吨运输车去满足，可以直接采用问题二的模型和算法求解。此时，应该会有比问题二时间花费更少的状态产生。那么，问题三的最优解实际上就应该为这两种极端状态之间中间状态的一种，并且这种状态能够较好的平衡时间和费用的关系。因而，很容易就能想到的是，可以通过一定数量的 4 吨车来走原有每辆 6 吨车的路线达到最终费用和时间都变小，即每次取一定数量的 6 吨车，用 4 吨车去遍历这些 6 吨车的路线，然后保存每次费用和时间，一旦费用和时间变小，就更新状态。

算法流程如下：

1. 提取和计算运算所需数据。
2. 随机抽取两个 6 吨车的路线。
3. 通过蚁群算法用 4 吨车去跑提取出来的 6 吨车的路线。
4. 将现有状态和原有状态比较，如果现有状态效率更高，则更新，否则进入下一次迭代。

最终，可得 4 吨车和 6 吨车的路径图如图所示：

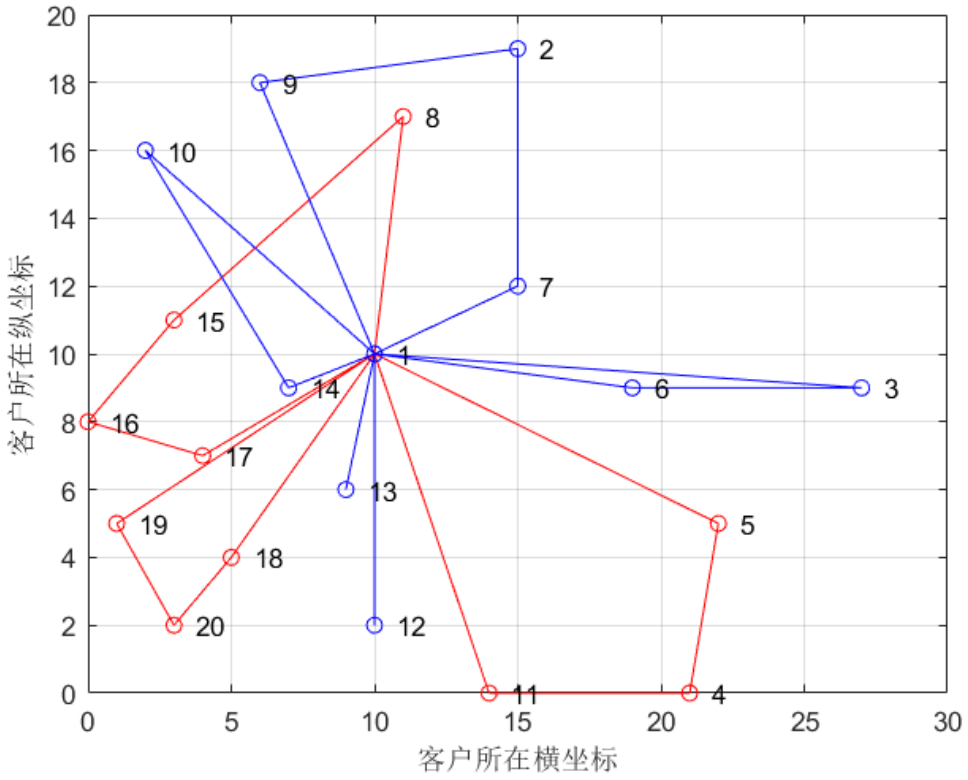

图 3.1 两种车的路径图(蓝:4 吨运输车,红:6 吨运输车)

5.3.3 结论

经过计算可得，需要 5 辆 4 吨运输车辆和 3 辆 6 吨运输车辆，对应的时间为 $t = 1.21h$，最小费用为$minM = 935.99$, 路程为$minL$ =224。

其分配的路线表如图所示：（路线编号为倒过来的编号，即现在的编号=21-原来的编号，例：1->20, 2->19…，20->1，以此类推）

表 3 问题三运输方案

运载类型	编号	路线总需求量	路线
4	1	2.7	1->14->10->1
4	2	1.5	1->13->1
4	3	3.2	1->7->2->1
4	4	2.2	1->9->12->1
4	5	2.35	1->3->6->1
6	1	6	1->18->20->19->1
6	2	4.85	1->17->16->15->8->1
6	3	5.8	1->11->4->5->1

由上面的表格我们可以得知，我们的车辆承载利用率δ为

$$\delta = 75.26\%$$

六、结果分析

6．1 模型一

模型一通过蚁群算法求解出的路径完全符合题目要求。由于运载车的载重大于各个销售地点的需求量之和，所以只需要根据蚁群算法的框架最终使得运载车不返回挨个遍历各个销售点即可得到最优路径。从图中我们可以很清晰的看出这个最优路径是符合要求的。

6.2 模型二和模型三

1.由于模型三是建立在模型二基础上的，所以我们这里将模型二和模型三合在一起分析。我们从题目中分析可以知道，调度效率与费用和时间有关，而费用与路程和载重有关。载重可以通过车辆经过的路径来进行调控，因此费用主要与路径有关。与此同时，由于运输车速度不变，时间也与路径相关，综合起来看，调度效率仅与路径直接相关。因此，模型二就可以将该 VRP 转化成 TSP。但是因为问题二本质上其实还是一个 VRP，并且有许多约束条件，需要对模型一采用的蚁群算法进行改进。改进后的蚁群算法获取的结果是有着多个哈密顿圈的哈密顿图，而当多个车辆同时沿着不同的哈密顿圈进行遍历时，花费时间最少，由此可以确定运输车辆的数目。最终可得我们的车辆承载利用率$\delta = 86.53\%$

2.从题目中可以直接看出，题目二是题目三的一种极端情况，即 4 吨运输车数量为 0，而题目三是题目二的一种一般情况——加入了另一种车辆运输。因此，我们将题目三转化为一种基于题目二的解的一种动态规划问题。最优解的状态必然在 4 吨运输车数量为 0 或 6 吨运输车数量为 0 两种极端状态中间的某一个状态。因此，我们用 4 吨车逐步取代题目二中所得结果的 6 吨车，并在约束条件下使得时间和费用再度变小，从而获得更加优化的方案。

3.在求解问题三时，我们通过逐步抽取不同情况下两辆 6 吨车的路线来作为 4 吨车的部分解空间。在这个过程中，我们总是尽可能多的让 4 吨车遍历更多的点，达到局部最优解，进而组成全局最优解。

七、模型评价

7.1 模型优点

本文首先很巧妙的将 VRP 问题转换为 TSP 问题进行求解，极大的简化了计算，约束条件严格，每一个站点只有一辆车运输且车辆必须送完所有食品并回到仓库，并且考虑运送时间、车辆数量、承载利用率等方面，将最少费用选为最优衡量指标，提供总体调度最优方案。并且在第三问中，将模型进一步拓展，从单车型调度问题拓展到多车型调度问题，以解决更为实际的问题。

7.2 模型缺点

在模型假设部分，假设该食品公司可以提供足够多的车辆，其次，在计算最小费用时忽略人工成本的影响，路径与时间成正比，这与实际情况有一定的差距。此外，虽然我们运用具体模型求解时，完全根据题目中所给的约束条件，但是交通状况、车速变化等客观因素仍然对模型的紧缺度有较大的影响。

7.3 模型改进

由于我们模型假设中运输车辆始终匀速前行，食品公司每派出一辆车所花费的人工成本和车辆在行驶过程中的耽搁时间直接忽略不计与实际情况不符。因此我们可以假设在运输车在行驶过程中平均耽搁时间为T_0,人工的工资为 z 元/每小时。因此在费用还应加入人工费用，综合考虑得出符合实际的最优解。原模型中计算费用为 C,时间为 t，记改进后记费用为 M，其表达式为：$M = C + z(T_0 + t)$,这样便可求出更真实的解。

参考文献

[1] 王晓东,张永强,薛红.基于改进蚁群算法对 VRP 线路优化[J].吉林大学学报(信息科学版),2017,35(02):198-203.

[2] 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.

[3] 陈萍,黄厚宽,董兴业.求解多车型车辆路径问题的变邻域搜索算法[J].系统仿真学报,2011,23(09):1945-1950.

[4] 贾立双,李静.基于一种改进算法的单车场多车型车辆调度研究[J].中国制造业信息化,2008(19):8-11.

[5] 王书勤. 车辆路径问题的蚁群算法研究[D].重庆大学,2008.

[6] 程满中,王江晴.基于蚂蚁算法的车辆路径问题应用研究[J].计算机与数字工程,2007(05):146-148+210.

[7] 叶志坚,叶怀珍,周道平,易海燕.多车型车辆路径问题的算法[J].公路交通科技,2005(05):147-151.

[8] 刘云忠，宣慧玉．蚂蚁算法在车辆路径问题中的应用研究[J]．信息制,2004(02):249-252.

附录

1. tsp.m

```
%%%蚁群算法解决TSP问题%%%%%%%

clear all; %清除所有变量
close all; %清图
clc ;      %清屏
m=16;    %% m 蚂蚁个数
Alpha=1;  %% Alpha 表征信息素重要程度的参数
Beta=5;  %% Beta 表征启发式因子重要程度的参数
Rho=0.5; %% Rho 信息素蒸发系数
NC_max=100; %%最大迭代次数
Q=100;          %%信息素增加强度系数
C=[
3 2;
1 5;
5 4;
4 7;
0 8;
3 11;
7 9;
9 6;
10 2;
14 0;
2 16;
6 18;
11 17;
16 12;
19 9;
22 5;
21 0;
27 9;
15 19;
10 10
];

%%---------------------------------------------------------------------
%% 主要符号说明
%% C n个城市的坐标，n×2的矩阵
%% NC_max 最大迭代次数
%% m 蚂蚁个数
%% Alpha 表征信息素重要程度的参数
%% Beta 表征启发式因子重要程度的参数
```

```
%% Rho 信息素蒸发系数
%% Q 信息素增加强度系数
%% R_best 各代最佳路线
%% L_best 各代最佳路线的长度
%%============================================================================
%%第一步：变量初始化
n=size(C,1);%n表示问题的规模（城市个数）
D=zeros(n,n);%D表示完全图的赋权邻接矩阵
for i=1:n
    for j=1:n
        if i~=j
            D(i,j)=(abs(C(i,1)-C(j,1))+abs(C(i,2)-C(j,2)));
        else
            D(i,j)=eps;        %i=j时不计算，应该为0，但后面的启发因子要取倒数，用eps（浮点相
对精度）表示
        end
        D(j,i)=D(i,j);    %对称矩阵
    end
end
Eta=1./D;             %Eta为启发因子，这里设为距离的倒数
Tau=ones(n,n);        %Tau为信息素矩阵
Tabu=zeros(m,n);    %存储并记录路径的生成
NC=1;                  %迭代计数器，记录迭代次数
R_best=zeros(NC_max,n);        %各代最佳路线
L_best=inf.*ones(NC_max,1);    %各代最佳路线的长度
L_ave=zeros(NC_max,1);         %各代路线的平均长度



while NC<=NC_max         %停止条件之一：达到最大迭代次数，停止
    %%第二步：将m只蚂蚁放到n个城市上
    Randpos=[];    %随即存取
    for i=1:(ceil(m/n))
        Randpos=[Randpos,randperm(n)];
    end
    Tabu(:,1)=(Randpos(1,1:m))';
    %%第三步：m只蚂蚁按概率函数选择下一座城市，完成各自的周游
    for j=2:n      %所在城市不计算
        for i=1:m
            visited=Tabu(i,1:(j-1)); %记录已访问的城市，避免重复访问
            J=zeros(1,(n-j+1));        %待访问的城市
            P=J;                       %待访问城市的选择概率分布
            Jc=1;
            for k=1:n
```

```
            if length(find(visited==k))==0    %开始时置0
                J(Jc)=k;
                Jc=Jc+1;                              %访问的城市个数自加1
            end
        end
        %下面计算待选城市的概率分布
        for k=1:length(J)
            P(k)=(Tau(visited(end),J(k))^Alpha)*(Eta(visited(end),J(k))^Beta);
        end
        P=P/(sum(P));
        %按概率原则选取下一个城市
        Pcum=cumsum(P);      %cumsum，元素累加即求和
        Select=find(Pcum>=rand); %若计算的概率大于原来的就选择这条路线
        to_visit=J(Select(1));
        Tabu(i,j)=to_visit;
    end
end
if NC>=2
    Tabu(1,:)=R_best(NC-1,:);
end
%%第四步：记录本次迭代最佳路线
L=zeros(m,1);      %开始距离为0，m*1的列向量
for i=1:m
    R=Tabu(i,:);
    for j=1:(n-1)
        L(i)=L(i)+D(R(j),R(j+1));     %原距离加上第j个城市到第j+1个城市的距离
    end
    L(i)=L(i)+D(R(1),R(n));       %一轮下来后走过的距离
end
L_best(NC)=min(L);            %最佳距离取最小
pos=find(L==L_best(NC));
R_best(NC,:)=Tabu(pos(1),:); %此轮迭代后的最佳路线
L_ave(NC)=mean(L);            %此轮迭代后的平均距离
NC=NC+1                       %迭代继续


%%第五步：更新信息素
Delta_Tau=zeros(n,n);         %开始时信息素为n*n的0矩阵
for i=1:m
    for j=1:(n-1)
        Delta_Tau(Tabu(i,j),Tabu(i,j+1))=Delta_Tau(Tabu(i,j),Tabu(i,j+1))+Q/L(i);
        %此次循环在路径（i，j）上的信息素增量
    end
    Delta_Tau(Tabu(i,n),Tabu(i,1))=Delta_Tau(Tabu(i,n),Tabu(i,1))+Q/L(i);
```

```
        %此次循环在整个路径上的信息素增量
    end
    Tau=(1-Rho).*Tau+Delta_Tau; %考虑信息素挥发，更新后的信息素
    %%第六步：禁忌表清零
    Tabu=zeros(m,n);                %%直到最大迭代次数
end
%%第七步：输出结果
Pos=find(L_best==min(L_best)); %找到最佳路径（非0为真）
Shortest_Route=R_best(Pos(1),:) %最大迭代次数后最佳路径
Shortest_Length=L_best(Pos(1)) %最大迭代次数后最短距离

figure(1)
plot(L_best)
xlabel('迭代次数')
ylabel('目标函数值')
title('适应度进化曲线')


figure(2)
subplot(1,2,1)                     %绘制第一个子图形
   %画路线图
%%=====================================================================
%% DrawRoute.m
%% 画路线图
%%---------------------------------------------------------------------
%% C Coordinate 节点坐标，由一个N×2的矩阵存储
%% R Route 路线
%%=====================================================================
N=length(R);
scatter(C(:,1),C(:,2));
 hold on
 plot([C(R(1),1),C(R(N),1)],[C(R(1),2),C(R(N),2)],'g')
 hold on
for ii=2:N
    plot([C(R(ii-1),1),C(R(ii),1)],[C(R(ii-1),2),C(R(ii),2)],'g')
     hold on
end
title('旅行商问题优化结果 ')

subplot(1,2,2)                     %绘制第二个子图形
plot(L_best)
hold on                            %保持图形
plot(L_ave,'r')
title('平均距离和最短距离')      %标题
```

2. good.ipynb

```
#%%

import numpy as np
import matplotlib as plt
import pandas as pd



data=pd.read_excel('data.xlsx')
# x=np.ndarray([])
print(data)

#%%

print(data['X'][1])

#%%

print(data['X'])

#%%

def distance(x1,y1,x2,y2):
    return abs(x1-x2)+abs(y1-y2)
distance(1,1,2,2)

#%%

c=np.zeros((20,20))


# print(c)
# print(c[1][1])
for i in range(len(data['X'])):
    for j in range(len(data['X'])):
        # print(data['X'][i],data['Y'][i],data['X'][j],data['X'][j])
        # print(distance(data['X'][i],data['Y'][i],data['X'][j],data['Y'][j]))
        c[i][j]=distance(data['X'][i],data['Y'][i],data['X'][j],data['Y'][j])

#%%

print(c)
```

```
#%%

import pandas as pd
data=pd.read_excel('data3.xlsx')
# x=np.ndarray([])
print(data)

#%%

max=31.15

print("time:",116/50+19*(1/3))
lis=np.array([ 20,7,4  ,   3   ,  1  ,   2   ,  5   ,  6  ,  11  ,  12  ,
13  ,  19  ,  14   , 15   , 18   , 16 ,   17  ,  10    , 9   ,  8    ])
lis = lis-1
print(lis)
waste=0
weight=max
for i in range(1,len(lis)):
    # print(lis[i])
    bf=lis[i-1]
    index=lis[i]
    # print(index)
    dis=distance(data['X'][bf],data['Y'][bf],data['X'][index],data['Y'][index])
    print(dis)
    waste+=2*weight*dis
    weight-=data['T'][index]
# print(waste)
print(0.6*distance(data['X'][19],data['Y'][19],data['X'][7],data['Y'][7]))
waste+=0.6*distance(data['X'][19],data['Y'][19],data['X'][7],data['Y'][7])
print(waste)

#%%

#%%



#%%

import pandas as pd
data=pd.read_excel('data3.xlsx')
# print(data)
def getans(lis,data):
```

```
    waste=0
    maxx=0
    for i in range(1,len(lis)):
        maxx+=data['T'][lis[i]-1]
        # print(data['T'][lis[i]-1])
    print(maxx<=6)
    length=0
    for i in range(1,len(lis)):
        # print(lis[i])
        bf=lis[i-1]-1
        index=lis[i]-1
        # print(index)

dis=distance(data['X'][bf],data['Y'][bf],data['X'][index],data['Y'][index])
        length+=dis
        # print(dis)
        waste+=2*maxx*dis
        maxx-=data['T'][index]
    # print("waste:",waste)
    #               print(0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0]))
    waste+=0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    length        +=        distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    print("waste:",waste)
    print("time:",length/50+5/60*(len(lis)-1))
    return waste
# x=np.ndarray([])
# print(data)
#lis=[[1,7,2,6,3],[1,8,9,10,14],[1,11,4,5],[1,12,20,18],[1,13],[1,17,19,16,15]]
# lis=[[1,14,13,12],[1,15,17,    18,
    19],[1,7,2,6,3],[1,8,9,10,16],[1,11,4,5],[1,20]]
# lis=[[1,14,13,12],[1,17,15,10,16],[1,18,20,19],[1,7,2,8,9],[1,6,3,5],[1,11,4]]
lis=[[1,14,13,12],[1,7,2,9,10],[1,18,20,19],[1,17,16,15],[1,11,4,5],[1,3,6]]
w=0
for l in lis:
    print(l)
    w+=getans(l,data)

print("totalcost:",w)

#%%
```

```
# fourlis=[[1,7],[1,3,2,4],[1,6,8,5]]
# fourlis=[[1,4,7],[1,3,2],[1,6,8],[1,5]]
# fourlis=[[1,3,2,4],[1,6,8],[1,5,7],[1,6,8]]
# fourlis=[[1,4,7],[1,3,2],[1,6,8],[1,5]]
# fourlis=[[1,7,5],[1,6],[1,2],[1,4],[1,3]]
# fourlis=[[1,5,2,6],[1,4],[1,7],[1,8,3]]

fourlis=[[1,14,10],[1,13],[1,7,2,9],[1,12]] #254
# fourlis=[[1,14,17,12],[1,13],[1,8,15,16]] 260
# fourlis=[[1,18,17],[1,7,2,9],[1,20],[1,10]] 323
# fourlis=[[1,7,2,8],[1,15,16,17],[1,9,10]] 328
# fourlis=[[1,7,2,8],[1,17,16,15],[1,9,10]] 336

def getansfour(lis,data):
    waste=0
    maxx=0
    for i in range(1,len(lis)):
        maxx+=data['T'][lis[i]-1]
        # print(data['T'][lis[i]-1])
    print(maxx<=4)
    length=0
    for i in range(1,len(lis)):
        # print(lis[i])
        bf=lis[i-1]-1
        index=lis[i]-1
        # print(index)

dis=distance(data['X'][bf],data['Y'][bf],data['X'][index],data['Y'][index])
        length+=dis
        # print(dis)
        waste+=2*maxx*dis
        maxx-=data['T'][index]
    # print("waste:",waste)
    #               print(0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0]))
    waste+=0.2*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    length        +=        distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    print("waste:",waste)
    print("time:",length/50+5/60*(len(lis)-1))
    return waste
wf=0
for l in fourlis:
```

```
    print(l)
    wf+=getansfour(l,data)

print("4lcost:",wf)
# 1 7   1   3   2   4   1   6   8   5   1
# 1 4   7   1   3   2   1   6   8   1   5
# 1 3   2   4   1   6   8   1   5   7   1
# 1 3   2   4   1   5   7   1   6   8   1
#%%



import pandas as pd
data=pd.read_excel('data3.xlsx')
# print(data)
def getans(lis,data):
    waste=0
    maxx=0
    for i in range(1,len(lis)):
        maxx+=data['T'][lis[i]-1]
        # print(data['T'][lis[i]-1])
    print(maxx<=6)
    length=0
    for i in range(1,len(lis)):
        # print(lis[i])
        bf=lis[i-1]-1
        index=lis[i]-1
        # print(index)

dis=distance(data['X'][bf],data['Y'][bf],data['X'][index],data['Y'][index])
        length+=dis
        # print(dis)
        waste+=2*maxx*dis
        maxx-=data['T'][index]
    # print("waste:",waste)
    #               print(0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0]))
    waste+=0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    length        +=        distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    print("waste:",waste)
    print("time:",length/50+5/60*(len(lis)-1))
    return waste
```

```
# x=np.ndarray([])
# print(data)
#lis=[[1,7,2,6,3],[1,8,9,10,14],[1,11,4,5],[1,12,20,18],[1,13],[1,17,19,16,15]]
# lis=[[1,14,13,12],[1,15,17,    18,
    19],[1,7,2,6,3],[1,8,9,10,16],[1,11,4,5],[1,20]]
# lis=[[1,14,13,12],[1,17,15,10,16],[1,18,20,19],[1,11,4]]



#lis=[[1,14,13,12],[1,7,2,9,10],[1,18,20,19],[1,17,16,15],[1,11,4,5],[1,3,6]]
lis=[[1,18,20,19],[1,17,16,15],[1,11,4,5]] #685.8
# lis=[[1,7,2,9,10],[1,18,20,19],[1,11,4,5],[1,3,6]] 832.6
# lis=[[1,14,13,12],[1,7,2,9,10],[1,11,4,5],[1,3,6]] 752.8
# lis=[[1,14,13,12],[1,18,20,19],[1,11,4,5],[1,3,6]] 696.6
# lis=[[1,14,13,12],[1,18,20,19],[1,11,4,5],[1,3,6]] 696.6
w=0
for l in lis:
    print(l)
    w+=getans(l,data)

print("6cost:",w)
print(getansfour([1,3,6],data))
#%%

# 939.8
# 1092.6
# 1,075.8
# 1024.6
# 1032.6
3.  yichuan.ipynb
#%%

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#%%

data=pd.read_excel('data3.xlsx')

#%%

data
car=[[4,0.2],[6,0.4]]
```

```
#%%

q=data['T']
q

#%%

def distance(x1,y1,x2,y2):
    return abs(x1-x2)+abs(y1-y2)



#%%

dis=np.zeros((20,20))
for i in range(len(data['X'])):
    for j in range(len(data['X'])):
        dis[i][j]=distance(data['X'][i],data['Y'][i],data['X'][j],data['Y'][j])

dis

#%%

V1=np.arange(1,20)
V2=V1[::-1]
V2

#%%

def find(arr,g):
    for i in range(len(arr)):
        if arr[i] == g :
            return True
    return False

def encode(v1,v2):
    # 8,9,10,11
    v1_example=v1.copy()
    v2_example=v2.copy()
    y1=[v1[8],v1[9],v1[10],v1[11]]
    # print(y1)
    y2=[v2[8],v2[9],v2[10],v2[11]]
    # print(y2)
    for  id in v1:
```

```
        # print(id)
        if find(y1,id) == False:
            y1.append(id)
    for im in v2:
        # print(im)
        if find(y2,im) == False:
            # print(1)
            y2.append(im)
            # print(y2)
    # print(y1)
    print(y2)
    return y1,y2

#%%

encode(V1,V2)


#%%

# print(np.random.random(1))
# # print(np.random.randint(low=1,high=20,size=1))
# m=np.random.randint(low=1,high=20,size=1)
# print(m.item())
def change(V1):
    r=np.random.random(1)
    # print(r)
    if r>0.66:
        m1=np.random.randint(low=1,high=20,size=1)
        m2=np.random.randint(low=1,high=20,size=1)
        t=V1[m1]
        V1[m1]=V1[m2]
        V1[m2]=t
    return V1
# print(V1)
# V2=change(V1)
# print(V2)


#%%

def divide(m):
    lis=[]
    print(car[0])
```

```
    counts=0
    for i in m:
        if counts<=6
        counts+=q[i]

divide(V1)

#%%

def getans(lis,data):
    waste=0
    maxx=0
    for i in range(1,len(lis)):
        maxx+=data['T'][lis[i]-1]
        # print(data['T'][lis[i]-1])
    print(maxx<=6)
    length=0
    for i in range(1,len(lis)):
        # print(lis[i])
        bf=lis[i-1]-1
        index=lis[i]-1
        # print(index)

dis=distance(data['X'][bf],data['Y'][bf],data['X'][index],data['Y'][index])
        length+=dis
        # print(dis)
        waste+=2*maxx*dis
        maxx-=data['T'][index]
    # print("waste:",waste)
    #               print(0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0]))
    waste+=0.4*distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    length        +=        distance(data['X'][len(lis)-1],data['Y'][len(lis)-
1],data['X'][0],data['Y'][0])
    print("waste:",waste)
    print("time:",length/50+5/60*(len(lis)-1))
    return waste


#%%
import pandas as pd
data=pd.read_excel('data.xlsx')
import numpy as np
```

```
route=np.array([20  ,7  ,4  ,3  ,1  ,5, 6,  11, 12  ,13 ,19 ,14 ,15 ,18 ,16 ,17
    ,10 ,9  ,8  ,20])
route=route-1
# print(route)
dist=0
for i in range(1,len(route)):

dist+=distance(data['X'][route[i]],data['Y'][route[i]],data['X'][route[i-
1]],data['Y'][route[i-1]])
        print(dist)
```

4. ANT_Test.m

```
clc;clear all
%% ==============提取数据==============
[xdata,textdata]=xlsread('data.xlsx'); %加载 20 个城市的数据，数据按照表格中位置
保存在 Excel 文件 exp12_3_1.xls 中
x_label=xdata(:,3); %第三列为横坐标
y_label=xdata(:,4); %第四列为纵坐标
Demand=xdata(:,2);  %第二列为需求量
C=[x_label y_label];      %坐标矩阵
n=size(C,1);         %n 表示节点（客户）个数

%修改下编号
x_changed=zeros(n,1);
y_changed=zeros(n,1);
Demand_changed=zeros(n,1);
%调整下编号，方便运算
for i=1:n
    x_changed(n-i+1,1)=x_label(i,1);
    y_changed(n-i+1,1)=y_label(i,1);
    Demand_changed(n-i+1,1)=Demand(i,1);
end
C=[x_changed y_changed];
%% ==============计算距离矩阵==============
D=zeros(n,n);        %D 表示完全图的赋权邻接矩阵，即距离矩阵 D 初始化
for i=1:n
   for j=1:n
       if i~=j
           D(i,j)=abs((C(i,1)-C(j,1)))+abs((C(i,2)-C(j,2))); %计算两城市之间的
距离
       else
           D(i,j)=0;   %i=j, 则距离为 0;
       end
       D(j,i)=D(i,j);  %距离矩阵为对称矩阵
   end
```

```
end
Cap=6;
%num_Vehicle = sum(Demand(1:n,1))/Cap;
Alpha=1;Beta=5;Rho=0.75;iter_max=500;Q=10;m=20;  %Cap 为车辆最大载重
[R_best,L_best,L_ave,Shortest_Route,Shortest_Length]=ANT_VRP(D,Demand_changed,C
ap,iter_max,m,Alpha,Beta,Rho,Q); %蚁群算法求解
Shortest_Route    %提取最优路线
Shortest_Length                         %提取最短路径长度

%% ==============作图==============
figure(1)   %作迭代收敛曲线图
x_changed=linspace(0,iter_max,iter_max);
y_changed=L_best(:,1);
plot(x_changed,y_changed);
xlabel('迭代次数'); ylabel('最短路径长度');

figure(2)   %作最短路径图
plot([C(Shortest_Route,1)],[C(Shortest_Route,2)],'o-');
grid on
for i =1:size(C,1)
    text(C(i,1),C(i,2),['    ' num2str(i-1)]);
end
xlabel('客户所在横坐标'); ylabel('客户所在纵坐标');
%xlswrite('shortestRoute.xlsx',Shortest_Route_1);
xlswrite('shortestRoute.xlsx',Shortest_Route);
```

5. ANT_VRP.m

```
function
[R_best,L_best,L_ave,Shortest_Route,Shortest_Length]=ANT_VRP(D_new,Demand_new,C
ap,iter_max,m,Alpha,Beta,Rho,Q)

%% R_best 各代最佳路线
%% L_best 各代最佳路线的长度
%% L_ave 各代平均距离
%% Shortest_Route 最短路径
%% Shortest_Length 最短路径长度
%% D_new 城市间之间的距离矩阵，为对称矩阵
%% Demand_new 客户需求量
%% Cap 车辆最大载重
%% iter_max 最大迭代次数
%% m 蚂蚁个数
%% Alpha 表征信息素重要程度的参数
%% Beta 表征启发式因子重要程度的参数
%% Rho 信息素蒸发系数
%% Q 信息素增加强度系数
```

```
%% num_Vehicle 车辆最大数量

n=size(D_new,1);                %地点数量
T=zeros(m,2*n);             %匹配的路线距离
Eta=ones(m,2*n);            %启发因子
Tau=ones(n,n);              %信息素
Tabu=zeros(m,n);            %禁忌表
Route=zeros(m,2*n);         %路径
L=zeros(m,1);               %总路程
L_best=zeros(iter_max,1);   %各代最佳路线长度
R_best=zeros(iter_max,2*n); %各代最佳路线
nC=1;

while nC<=iter_max                      %停止条件
    Eta=zeros(m,2*n);
    T=zeros(m,2*n);
    Tabu=zeros(m,n);
    Route=zeros(m,2*n);
    L=zeros(m,1);

    %%%%%%=============初始化起点城市（禁忌表）==================
    for i=1:m
        Cap_1=Cap;      %最大装载量
        %禁忌表指定城市编号
        j=1;

        j_r=1;          %路线城市指定编号
        while Tabu(i,n)==0
            T=zeros(m,2*n);     %装载量加载矩阵
            Tabu(i,1)=1;        %禁忌表起点位置为1
            Route(i,1)=1;       %路径起点位置为1
            visited=find(Tabu(i,:)>0);   %已访问城市
            num_v=length(visited);          %已访问城市个数
            J=zeros(1,(n-num_v));           %待访问城市加载表
            P=J;                            %待访问城市选择概率分布
            Jc=1;                           %待访问城市选择指针
            for k=1:n                       %城市
                if length(find(Tabu(i,:)==k))==0     %如果k不是已访问城市代号，
就将k加入矩阵J中
                    J(Jc)=k;
                    Jc=Jc+1;
                end
            end
```

```
        %%%%%%%============ 每 只 蚂 蚁 按 照 选 择 概 率 遍 历 所 有 城 市
=================

            for k=1:n-num_v                 %待访问城市

                if Cap_1-Demand_new(J(1,k),1)>=0      %如果车辆装载量大于待访
问城市需求量

                    if Route(i,j_r)==1            %如果每只蚂蚁在起点城市
                        T(i,k)=D_new(1,J(1,k));
                        P(k)=(Tau(1,J(1,k))^Alpha)*((1/T(i,k))^Beta);  %概率计
算公式中的分子
                    else                          %如果每只蚂蚁在不在起点城市
                        T(i,k)=D_new(Tabu(i,j),J(1,k));

P(k)=(Tau(Tabu(i,visited(end)),J(1,k))^Alpha)*((1/T(i,k))^Beta); %概率计算公式
中的分子
                    end

                else                %如果车辆装载量小于待访问城市需求量
                    T(i,k)=0;
                    P(k)=0;
                end
            end

             % 运输车到下一个地点距离为 0 时，下一个选择在起点 1
            if length(find(T(i,:)>0))==0     %%%当车辆装载量小于待访问城市时，
选择起点为 1
                % 重新装满货物
                Cap_1=Cap;
                j_r=j_r+1;
                Route(i,j_r)=1;
                L(i)=L(i)+D_new(1,Tabu(i,visited(end)));
            %没有回仓库
            else
                P=P/(sum(P));                 %按照概率原则选取下一个城市
                Pcum=cumsum(P);               %求累积概率和：cumsum（[1;2;3])=1
3 6,目的在于使得 Pcum 的值总有大于 rand 的数
                Select=find(Pcum>rand);        %按概率选取下一个城市：当累积概
率和大于给定的随机数，则选择求和被加上的最后一个城市作为即将访问的城市
                o_visit=J(1,Select(1));       %待访问城市
                j=j+1;
                j_r=j_r+1;
                Tabu(i,j)=o_visit;             %待访问城市
```

```
                Route(i,j_r)=o_visit;
                Cap_1=Cap_1-Demand_new(o_visit,1);  %车辆装载剩余量
                L(i)=L(i)+T(i,Select(1));        %路径长度
            end
        end
         L(i)=L(i)+D_new(Tabu(i,n),1);                %%路径长度
    end

    L_best(nC)=min(L);              %最优路径为距离最短的路径
    pos=find(L==min(L));            %找出最优路径对应的位置：即为哪只蚂蚁
    R_best(nC,:)=Route(pos(1),:);   %确定最优路径对应的城市顺序
    L_ave(nC)=mean(L)';             %求第 k 次迭代的平均距离

    Delta_Tau=zeros(n,n);               %Delta_Tau(i,j)表示所有蚂蚁留在第 i 个城市
到第 j 个城市路径上的信息素增量
    L_zan=L_best(1:nC,1);
    post=find(L_zan==min(L_zan));
    Cities=find(R_best(nC,:)>0);
    num_R=length(Cities);

    for k=1:num_R-1           %建立了完整路径后在释放信息素

Delta_Tau(R_best(nC,k),R_best(nC,k+1))=Delta_Tau(R_best(nC,k),R_best(nC,k+1))+Q
/L_best(nC);
    end

Delta_Tau(R_best(nC,num_R),1)=Delta_Tau(R_best(nC,num_R),1)+Q/L_best(nC);
        Tau=Rho*Tau+Delta_Tau;

        nC=nC+1;
end
Shortest_Route=zeros(1,2*n);            %提取最短路径
Shortest_Route(1,:)=R_best(iter_max,:);
Shortest_Route=Shortest_Route(Shortest_Route>0);
Shortest_Route=[Shortest_Route Shortest_Route(1,1)];
Shortest_Length=min(L_best);            %提取最短路径长度
%L_ave=mean(L_best);
```

6. Problem3.m

```
clc;clear all;
%% ==============提取数据==============
[xdata,textdata]=xlsread('data.xlsx'); %加载 20 个城市的数据，数据按照表
格中位置保存在 Excel 文件 exp12_3_1.xls 中
x_label=xdata(:,3); %第三列为横坐标
y_label=xdata(:,4); %第四列为纵坐标
```

```
Demand=xdata(:,2);  %第二列为需求量
C=[x_label y_label];      %坐标矩阵
n=size(C,1);         %n 表示节点（客户）个数
%修改下编号
x_changed=zeros(n,1);   %改变后的坐标
y_changed=zeros(n,1);   %改变后的坐标
Demand_changed=zeros(n,1);
%调整下编号，方便运算
for i=1:n
    x_changed(n-i+1,1)=x_label(i,1);
    y_changed(n-i+1,1)=y_label(i,1);
    Demand_changed(n-i+1,1)=Demand(i,1);
end
C=[x_changed y_changed];
Shortest_Length_6t=xlsread('shortestLength_6t.xlsx'); %6 吨获取最短路径长度
D=zeros(n,n);
for i=1:n
   for j=1:n
       if i~=j
           D(i,j)=abs((C(i,1)-C(j,1)))+abs((C(i,2)-C(j,2))); %计算两城市之间的距离
       else
           D(i,j)=0;   %i=j, 则距离为 0;
       end
       D(j,i)=D(i,j);  %距离矩阵为对称矩阵
   end
end
shortest_Route = xlsread('shortestRoute.xlsx');                    %获取问题 2 中得到的最佳路径
%num_R = size(shortest_Route,2);         %有多少个节点
num_Vehicle_6t = size(find(shortest_Route(1,:)==1),2)-1;        %获取 6 吨车辆数目
Route_6t = zeros(num_Vehicle_6t,n);    %题目二中各辆 6 吨车的路线
Route_4t = zeros(ceil(num_Vehicle_6t*6/4),n);       %4 吨运输车的运输线路
number_orign=find(shortest_Route(1,:)==1);       %找到每个 6 吨车起始点在路线中的编号

%% ============================数据运算=================

tmp_1=zeros(n,n);
size_Route=zeros(num_Vehicle_6t,1);%每条路线结点数目
for i=1:num_Vehicle_6t
```

```
tmp_1(i,number_orign(1,i):number_orign(1,i+1))=shortest_Route(1,number_o
rign(1,i):number_orign(1,i+1));
    size_Route(i,1)=size(find(tmp_1(i,:)),2);
end

for i=1:num_Vehicle_6t

Route_6t(i,1:size_Route(i,1))=shortest_Route(1,number_orign(1,i):number_
orign(1,i+1));
end
%获取 6 吨车每条路线的长度
Length_6t =zeros(num_Vehicle_6t,1);
for i=1:num_Vehicle_6t
    for s=1:n
        if Route_6t(i,s)==0
            break;
        else


Length_6t(i,1)=Length_6t(i,1)+D(Route_6t(i,s),Route_6t(i,s));
        end

    end
end

Replaced = zeros(1,num_Vehicle_6t);%可以被取代的列表

sum_Demand = zeros(num_Vehicle_6t,1); %获取每条路线的总需求
for i=1:num_Vehicle_6t

sum_Demand(i,1)=sum(Demand_changed(Route_6t(i,1:size_Route(i,1)),1));
end

%先找每条线路的总需求量小于等于 4 吨的
for i=1:num_Vehicle_6t
    if sum_Demand(i,1)<=4
        Replaced(1,i)=i;
    end
end
Replaced_1 = find(Replaced);%找到要替代的车辆编号
num_Vehicle_4t=0;%4 吨车数量

if size(Replaced_1,2)>0 %有可以直接替代的车辆
    num_Vehicle_4t = size(Replaced_1,2); %更新 4 吨车辆数量
```

```
    num_Vehicle_6t = num_Vehicle_6t-1;

    iter_flag=0; %迭代次数标志

    tmp_L = zeros(1,10);    %保存每次选取两个不同6吨车辆后最佳路径的最短长度
    tmp_R = zeros(10,n);    %保存每次选取两个不同6吨车辆后最佳路径
    All_Route = zeros(15,20);%保存每种情况下的映射
    set_selectedVehicle=[1,2;1,3;1,4;1,5;2,3;2,4;2,5;3,4;3,5;4,5];
    while iter_flag<1000 %随机抽取1000次
        tmp_Vehicle=randperm(5,2); %随机抽取两个车辆

        %将这些6吨车遍历的地点放入到4吨车即将遍历的地点里
        Route_4t(tmp_Vehicle(1,1),:) = Route_6t(tmp_Vehicle(1,1),:);
        Route_4t(tmp_Vehicle(1,2),:) = Route_6t(tmp_Vehicle(1,2),:);
        %需要遍历的点
        tmp_Route = [Route_4t(tmp_Vehicle(1,1),:) Route_4t(tmp_Vehicle(1,2),:)];
        tmp_Route(find(tmp_Route==0))=[];
        tmp_Route=unique(tmp_Route);
        %4吨车需要遍历的地点的数量
        num_4t = size(tmp_Route,2);
        %构建新的距离矩阵
        D_new=zeros(size(tmp_Route,2),size(tmp_Route,2));
        for i=1:size(tmp_Route,2)
            for j=1:size(tmp_Route,2)
                if tmp_Route(1,j)~=tmp_Route(1,i)
                    D_new(i,j)=abs((C(tmp_Route(1,i),1)-C(tmp_Route(1,j),1)))+abs((C(tmp_Route(1,i),2)-C(tmp_Route(1,j),2))); %计算两城市之间的距离
                else
                    D_new(i,j)=0;   %i=j, 则距离为0;
                end
                D_new(j,i)=D_new(i,j);  %距离矩阵为对称矩阵
            end
        end
        %构建新的需求矩阵
        Demand_new=zeros(num_4t,1);
        for i=1:num_4t

            Demand_new(i,1)=Demand_changed(tmp_Route(1,i),1);

        end
        Alpha=1;Beta=5;Rho=0.75;iter_max=500;Q=10;m=20;Cap=4;  %Cap为车
```

辆最大载重

```
[R_best,L_best,L_ave,Shortest_Route,Shortest_Length]=ANT_VRP(D_new,Demand_new,Cap,iter_max,m,Alpha,Beta,Rho,Q); %蚁群算法求解 4 吨车最佳路径
        %将迭代结果保存下来

        %一次抽取后的长度
        for o=1:10  %判断抽取的是哪种情况
            if
tmp_Vehicle(1,1)==set_selectedVehicle(o,1)||tmp_Vehicle(1,1)==set_selectedVehicle(o,2)&&(tmp_Vehicle(1,2)==set_selectedVehicle(o,1)||tmp_Vehicle(1,2)==set_selectedVehicle(o,2))
                tmp_L(1,o)=Shortest_Length;
                tmp_R(o,1:size(Shortest_Route,2))=Shortest_Route;
                All_Route(o,1:size(tmp_Route,2))=tmp_Route;
                tmp_Length_1 = Shortest_Length_6t-
Length_6t(tmp_Vehicle(1,1),1)-Length_6t(tmp_Vehicle(1,2),1);%临时长度
                tmp_Length_2 = Shortest_Length_6t-Shortest_Length;
                if tmp_Length_1 < tmp_Length_2

                    tmp_num_Vehicle_4t =
size(find(Shortest_Route(1,:)==1),2)-1;%获取 4 吨运输车数量
                    num_Vehicle_4t = num_Vehicle_4t+tmp_num_Vehicle_4t;
                    num_Vehicle_6t = num_Vehicle_6t-2;
                end
            end
        end
        iter_flag=iter_flag+1;
    end
else
    iter_flag=0; %迭代次数标志

    tmp_L = zeros(1,15);     %保存每次选取两个不同 6 吨车辆后最佳路径的最
短长度
    tmp_R = zeros(15,n);     %保存每次选取两个不同 6 吨车辆后最佳路径
    tmp_Y = zeros(15,n);     %保存每次选取的映射

set_selectedVehicle=[1,2;1,3;1,4;1,5;1,6;2,3;2,4;2,5;2,6;3,4;3,5;3,6;4,5
;4,6;5,6];
    while iter_flag<100 %随机抽取 100 次
        tmp_Vehicle=randperm(6,2); %随机抽取两个车辆

        %将这些 6 吨车遍历的地点放入到 4 吨车即将遍历的地点里
        Route_4t(tmp_Vehicle(1,1),:) = Route_6t(tmp_Vehicle(1,1),:);
```

```
        Route_4t(tmp_Vehicle(1,2),:) = Route_6t(tmp_Vehicle(1,2),:);
        %需要遍历的点
        tmp_Route = [Route_4t(tmp_Vehicle(1,1),:)
Route_4t(tmp_Vehicle(1,2),:)];
        tmp_Route(find(tmp_Route==0))=[];
        tmp_Route=unique(tmp_Route);
        %4 吨车需要遍历的地点的数量
        num_4t = size(tmp_Route,2);
        %构建新的距离矩阵
        D_new=zeros(size(tmp_Route,2),size(tmp_Route,2));
        for i=1:size(tmp_Route,2)
            for j=1:size(tmp_Route,2)
                if tmp_Route(1,j)~=tmp_Route(1,i)
                    D_new(i,j)=abs((C(tmp_Route(1,i),1)-
C(tmp_Route(1,j),1)))+abs((C(tmp_Route(1,i),2)-C(tmp_Route(1,j),2))); %
计算两城市之间的距离
                else
                    D_new(i,j)=0;   %i=j, 则距离为 0;
                end
                D_new(j,i)=D_new(i,j);  %距离矩阵为对称矩阵
            end
        end
        %构建新的需求矩阵
        Demand_new=zeros(num_4t,1);
        for i=1:num_4t

            Demand_new(i,1)=Demand_changed(tmp_Route(1,i),1);

        end
        Alpha=1;Beta=5;Rho=0.75;iter_max=500;Q=10;m=20;Cap=4;  %Cap 为车
辆最大载重

[R_best,L_best,L_ave,Shortest_Route,Shortest_Length]=ANT_VRP(D_new,Deman
d_new,Cap,iter_max,m,Alpha,Beta,Rho,Q); %蚁群算法求解 4 吨车最佳路径

        %一次抽取后的长度
        for o=1:15  %判断抽取的是哪种情况
            if
tmp_Vehicle(1,1)==set_selectedVehicle(o,1)||tmp_Vehicle(1,1)==set_select
edVehicle(o,2)&&(tmp_Vehicle(1,2)==set_selectedVehicle(o,1)||tmp_Vehicle
(1,2)==set_selectedVehicle(o,2))
                tmp_L(1,o)=Shortest_Length;
                tmp_R(o,1:size(Shortest_Route,2))=Shortest_Route;
                tmp_Length_1 = Shortest_Length_6t-
```

```
    Length_6t(tmp_Vehicle(1,1),1)-Length_6t(tmp_Vehicle(1,2),1);%临时长度
                    tmp_Length_2 = Shortest_Length_6t-Shortest_Length;
                    if tmp_Length_1 < tmp_Length_2

                        tmp_num_Vehicle_4t =
    size(find(Shortest_Route(1,:)==1),2)-1;%获取 4 吨运输车数量
                        num_Vehicle_4t = num_Vehicle_4t+tmp_num_Vehicle_4t;
                        num_Vehicle_6t = num_Vehicle_6t-2;
                    end
                end
            end
            iter_flag=iter_flag+1;
        end
    end
    num_Vehicle_6t
    num_Vehicle_4t
    xlswrite('tmp_4t.xlsx',tmp_R);
    xlswrite('tmp_4tL.xlsx',tmp_L);
```

7. Problem3_1.m

```
%% =============提取数据=============
[xdata,textdata]=xlsread('data.xlsx'); %加载 20 个城市的数据，数据按照表格中位置
保存在 Excel 文件 exp12_3_1.xls 中
x_label=xdata(:,3); %第三列为横坐标
y_label=xdata(:,4); %第四列为纵坐标
Demand=xdata(:,2);  %第二列为需求量
C=[x_label y_label];      %坐标矩阵
n=size(C,1);         %n 表示节点（客户）个数

%修改下编号
x_changed=zeros(n,1);
y_changed=zeros(n,1);
Demand_changed=zeros(n,1);
%调整下编号，方便运算
for i=1:n
    x_changed(n-i+1,1)=x_label(i,1);
    y_changed(n-i+1,1)=y_label(i,1);
    Demand_changed(n-i+1,1)=Demand(i,1);
end
C=[x_changed y_changed];
%% =====第三问画图
figure(2)   %作最短路径图
tmp_1=[1,14,10,1,13,1,7,2,9,1,12,1,3,6,1];%4 吨车
tmp_2=[1,18,20,19,1,17,16,15,8,1,11,4,5,1];%6 吨车
plot([C(tmp_2,1)],[C(tmp_2,2)],'o-r');
```

```
hold on;
plot([C(tmp_1,1)],[C(tmp_1,2)],'o-b');
grid on

for i =1:size(C,1)
    text(C(i,1),C(i,2),['   ' num2str(i)]);
end
xlabel('客户所在横坐标'); ylabel('客户所在纵坐标');
%xlswrite('shortestRoute.xlsx',Shortest_Route_1);
%xlswrite('shortestRoute.xlsx',Shortest_Route);
```