基于遗传算法的快递公司送货策略选择

摘要

近年来，快递行业正蓬勃发展，为我们的生活带来更多方便。随着快递业务量的增加，快递公司的送货策略也变得越来越重要。本文将利用遗传算法，为快递公司设计一个合理的送货策略。

针对问题一，经过分析可知，该题属于VRP问题的变种。送货策略的影响因素有以下几点：送货时间，送货快递员人数，送货里程，总费用等。其中，还需要满足平均送货时间、单次运输货物重量等约束。由于涉及多个决策变量，送货的所有可能性过于庞大，采用普通的线性，非线性规划方法求解仍有计算量过大的问题。因此，我们采用遗传算法来求解该VRP问题，求得各个快递员的送货路线，以及总的送货里程数等信息，并利用路线图表进行可视化，且数据与图表均证明效果良好。其中，送货的总时间为3小时15分37秒，总行驶里程为383.81公里。

针对问题二，在第一问的基础上，快递员的速度和相应的费用，以及货物情况都有一定的变化。因此需要为不同情况分配不同的权重，制定不同优先级，其中关键目标是总费用最小化，即需要尽可能少的业务员，使得总成本最小。在此基础上，我们还需要在遗传算法的基础上考虑贪心策略，即先送进的，先送重的快递，同时还要考虑送远处的快递相对承重要少一些。否则在前段时间中会承载更重的快递，导致总成本增大。最终求解得出的总成本为10505元，送货的总时间为3小时16分29秒，总行驶里程为496.13公里。

针对问题三，相对于原题只是送货时间延长，这时可以使用更少的派送员，派送更多的快件，因此送货策略会有所变化。在使用遗传算法的基础上，我们需要重新考虑送货策略，为送货员数量分配更大的权重，同时结合第二问的贪心策略，考虑派送所需总费用问题。我们重新计算送货员的送货路线，以及总的送货里程数等信息，并利用路线图表进行可视化，最终求出送货所消耗的总时间为3小时11分46秒，总行驶里程为381.49公里。

关键词：VRP问题；遗传算法；送货策略；优化

一、 问题重述

现有一间快递公司需要在规定条件下安排业务员派送完所有快件。所有快件在早上 7 点钟到达，早上 9 点钟开始派送，要求于当天17点之前必须派送完毕，每个业务员每天平均工作时间不超过 6 小时，在每个送货点停留的时间为 10 分钟，途中速度为 25km/h，每次出发最多能带 25 千克的重量。为计算方便，本文将快件一律用重量来衡量，平均每天收到总重量为 184.5 千克。

根据以上信息完成下面的问题解答：

问题一:请运用有关数学建模的知识，给该公司提供一个合理的送货策略（即需要多少个业务员，每个业务员的运行线路，以及总的运行公里数）;

问题二:如果业务员携带快件时的速度是 20km/h，获得酬金3 元/km · kg；而不携带快件时的速度是30km/h，酬金2 元/km，请为公司设计一个费用最省的策略;

问题三:如果可以延长业务员的工作时间到 8 小时，公司的送货策略将有何变化。

二、 问题分析

2.1 问题一的分析

由于需要计算各个派送点之间的距离，我们需要先算出各个派送点之间的距离，即得到距离矩阵。当快递员去较远的地方时，一次来回就大概要花费4小时，而且还要排派送多个点，因此最终时间已经接近六小时，无法在中途回总部再继续前往派送。然后，我们采取了遗传算法，首先定义快递员类，送货节点类还有其他工具类等，对多个功能进行模块化封装。然后采用遗传算法，用一个基因序列来代表结果。其中序列以0进行分隔，每个子数组都代表一个快递员走过的节点的集合（有按行程顺序）。首先初始化1000个基因序列，每个序列代表种群中的一个个体。然后每次迭代都算出其中最好的基因序列作为候选最优解，且每次迭代末尾都有对后50%的个体进行进化和变异处理，以确保总群的整体水平不断接近最优解。最终当连续迭代500次仍没有更新时认为十分接近最优解，排序后拿到种群首个个体的基因序列作为最终答案，且可视化后也可再次证明该送货策略的合理性。

2.2 问题二的分析

问题二增加了业务员的约束条件：业务员携带快件时的速度是 20km/h，获得酬金3 元/km · kg；而不携带快件时的速度是30km/h，酬金2 元/km。

为了设计一个费用最省的策略，本文将目标函数定为成本，使其最小化，因此需要尽可能少的业务员，在满足约束条件的情况下，使得总成本最小。因此，针对问题二，本文在使用遗传算法的基础上，巧用贪心策略，来解决成本最小化的问题，即先送先进的，先送重的快递。否则在前段时间中会承载更重的快递，导致总成本增大。

2.3 问题三的分析

问题三延长了业务员的工作时间，意味着可以使用更少的派送员，派送更多的快件，因此送货策略会有所变化。在问题一和问题二的解决方案的基础上，本文为快递员数量、总里程分配了更多权重，并重新计算适应度来对不同策略进行评估。

三、 模型假设

考虑到真实道路的分布情况，以及模型构建的合理性，我们做出以下假设：

1. 由于题目未给出具体道路情况和距离值，只给出各个派送点的坐标，因此各个派送点之间的距离用欧式距离来计算，计算公式如下：

$$D_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{1}$$

2. 送货平均时间不能超过6小时，且快递员到达送货点需要停留10分钟。即有些快递员可以送货超过6小时，但需要有其他快递员的送货时间短来作为平均代价。若不加以约束，在迭代过程中要反复计算平均送货时间并作合法校验，若校验不通过还需反向传播更新或者直接抛弃该选择，直接考虑下一个可行解，计算效率较低且效果不明显。因此假定每个快递员的送货时间均不超过6小时，将其作为约束条件。在简化模型，提高效率的同时，还对送货员的送货约束进行了合理化。

3. 遗传算法种群大小为1000，最大重复结果迭代次数为500，即若连续500次迭代结果没有更新，则认为已经收敛，取排序后种群的首元素作为最优解。

四、 符号说明

符号	表示含义
$\boldsymbol{D_{ij}}$	节点i和j之间的距离
$\boldsymbol{fitness}$	适应度，值越小说明结果越好
$\boldsymbol{time_i}$	快递员i派送所需的总时间（单位是分钟）

$mileage_i$	快递员i的行驶里程
$load_i$	快递员i的总载荷
$money_i$	快递员i获得的酬金
$allTime$	完成所有派送任务所需的时间
$allMileage$	总里程
n	有派送任务的快递员人数
$allCost$	交付给快递员的总费用
$gene$	基因编码序列
$path_i$	快递员i派送的送货点集合
$valid$	有效性标志
cnt	迭代次数计数器
$nodeInfo$	节点信息，包含送货点的相关数据
nid	送货点编号
x_j	送货点j的 x 坐标
y_j	送货点j的 y 坐标
$demand_j$	送货点j的需求
$courierInfo$	快递员信息，包含快递员的相关数据
cid	快递员编号
$capacity$	快递员送货承载量
$timeLimit$	行驶时间限制
w_t	时间指标的权重
w_s	总里程指标的权重
w_n	快递员人数指标的权重

五、 模型的建立与求解

5.1 模型一的建立与求解

5.1.1 快递员派送策略模型

本题可视为经典的VRP问题，即车辆路径优化问题，是一种典型的组合优化问题。其目标是找到一组最优的送货路线，使得送货时间最短，送货里程最短，送货快递员人数最少。Kim [1]等人对VRP问题进行了综合介绍，包括最新进展，使用的模型，潜

在探知领域等。张 [7]以及Zirour [5]等人也对VRP问题进行了详细介绍以及求解分析。本文采用遗传算法来构建快递员派送策略模型，具体步骤如下：

step1: 初始化时要输入快递员数量，送货点数量，送货时间，送货点坐标等信息。

step2: 由于需要计算各个派送点之间的距离，我们需要先算出各个派送点之间的距离，即得到距离矩阵。距离矩阵如下图 1 所示。

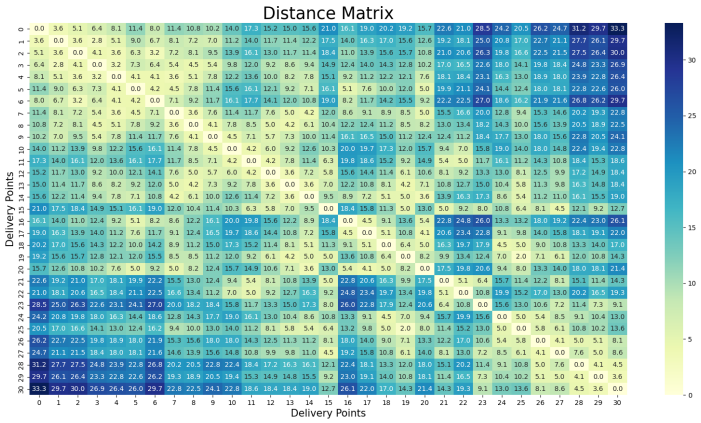


图 1: 距离矩阵

step3: 确定目标函数，以其函数值的最小化为目标进行求解。其目标评价指标由三个因素组成：送货时间、送货里程，以及送货快递员人数。公式如下：

$$fitness = w_t \cdot allTime + w_s \cdot allMileage + w_n \cdot n \tag{2}$$

其中，w_t、w_s、w_n 分别为送货时间、送货里程，以及送货快递员人数的权重系数，$allTime$、$allMileage$、n 分别为送货时间、送货里程，以及送货快递员人数。由于送货时间和里程的数值较大，而快递员人数只是一个很小的整数，所以需要分配给送货员人数一个较大的权重，以保证其在目标函数中的影响。在本题中，我们先取$w_t = 1$，$w_s = 1$，$w_n = 10$。

step4: 确定各方面的约束条件，如单个快递员送货时间耗时不能超过6小时（360分钟）等。王 [10]等人以及郭 [12]等人都构建了带软时间窗的车辆路径优化问题模型，由于本题中的顾客没有时间窗限制，因此不需要考虑时间窗约束条件。

里程约束条件如下：

$$time_i \leq 360, \quad \forall i = 1, \ldots, n \tag{3}$$

货物装载约束条件如下：

$$load_i = \sum_{j \in path_i} demand_j \leq 25, \quad \forall i = 1, \ldots, n \tag{4}$$

其中，快递员i经过的送货点集合为$path_i$，j表示送货点编号，$demand_j$表示送货点j的需求。

step5: 由于参数过多，求解前需对相关信息进行不同类的封装，并根据求解目标确定好相关的等式约束。相关变量关系以及类的封装信息如下：

总时间的计算公式如下：

$$allTime = \sum_{i=1}^{n} time_i \tag{5}$$

总里程数的计算公式如下：

$$allMileage = \sum_{i=1}^{n} mileage_i \tag{6}$$

送货点对象信息如下：

$$nodeInfo = \begin{cases} nid & \text{送货点编号} \\ x & \text{送货点的 } x \text{ 坐标} \\ y & \text{送货点的 } y \text{ 坐标} \\ demand & \text{送货点的需求} \end{cases} \tag{7}$$

快递员对象信息如下：

$$courierInfo = \begin{cases} cid & \text{快递员编号} \\ capacity & \text{快递员送货承载量} \\ timeLimit & \text{行驶时间限制} \end{cases} \tag{8}$$

5.1.2 遗传算法优化求解模型

1. 遗传算法简介

遗传算法（Genetic Algorithm, GA）是一种基于自然选择和遗传机制的优化算法。[2]、[3]等人对遗传算法的发展历史，概念等做了详细的介绍。它模仿了生物进化过程，利用选择、交叉和变异等操作在解决复杂优化问题上具有显著效果。遗传算法最早由

约翰·霍兰德（John Holland）在20世纪70年代提出，并被广泛应用于工程、科学、经济等领域。

2. 遗传算法的基本概念

- **个体与种群：**
  - **个体**：遗传算法中的个体代表一个可能的解。
  - **种群**：多个个体组成一个种群，每个种群代表一组可能的解。
- **基因与染色体：**
  - **基因**：个体的特征参数。
  - **染色体**：由基因构成的字符串或数组，表示一个完整的解。
- **适应度函数：**
  - 适应度函数用于评估个体的优劣，适应度值越高表示该个体越优。

3. 遗传算法的基本步骤

遗传算法的流程图如图 2 所示。

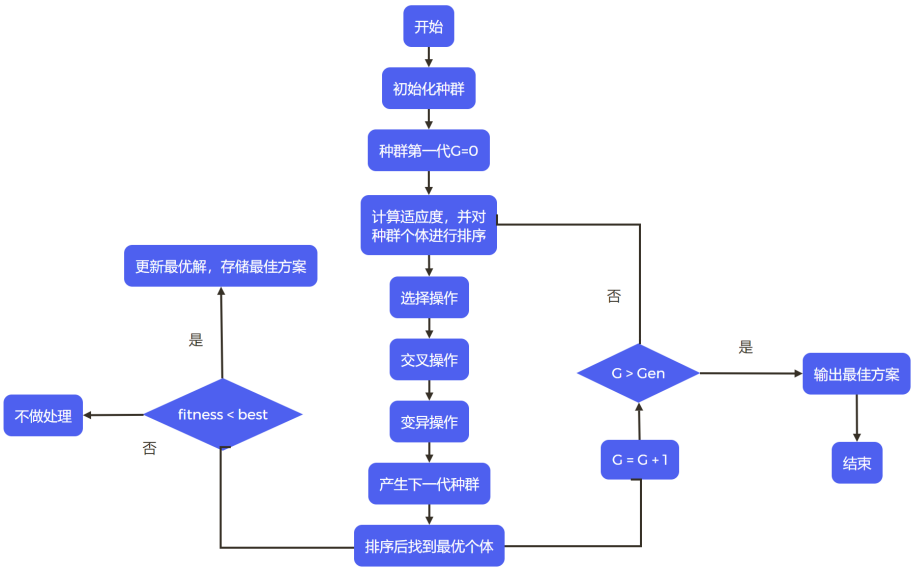


图 2: 遗传算法流程图

Step1：初始化

随机生成一个初始种群。

Step2: 适应度评估

计算种群中每个个体的适应度值。

Step3: 选择（Selection）

根据适应度值选择较优的个体用于繁殖，常用的方法有轮盘赌选择、锦标赛选择等。

Step4: 交叉（Crossover）

选择的个体进行交叉操作，产生新的个体（子代），常见的交叉方法有单点交叉、多点交叉等。

Step5: 变异（Mutation）

对新个体的基因进行随机变异，以增加种群的多样性。

Step6: 生成新种群

用选择、交叉和变异操作生成的新个体组成新的种群。

Step7: 终止条件

判断是否满足终止条件，如达到预设的适应度值或迭代次数。

Step8: 最优解输出

输出种群中的最优个体作为最终解。

4. 算法流程

下面是一个简单的遗传算法伪代码：

Algorithm 1 遗传算法

```
初始化种群
评估种群适应度
while not termination_condition() do
  选择操作
  交叉操作
  变异操作
  生成新种群
  评估种群适应度
end while
输出最优解
```

遗传算法因其简单有效，被广泛应用于各种优化问题中，成为求解复杂问题的重要工具之一。在求解VRP这类问题中，王 [9]等人提及了求出精确解的局限性，并强调了遗传算法作为启发式算法，在实际计算量上的合理性与可行性。同时，黄 [14]等人进一步详细说明了相比于普通的优化算法，遗传算法求解过程的合理性。通过多点切入，

全局搜索，而不是直接从单个初始点进行迭代，从而避免过早陷入局部最优解。因此，我们采取遗传算法应用于该VRP问题，得到一个合理的送货策略。

首先确定约束条件，根据模型假设进行约束，然后定义快递员类，送货节点类还有其他工具类等，对多个功能进行模块化封装。

对于迭代中间的具体过程，刘 [6]等人提出了自适应遗传算法，能够根据迭代过程中的实际情况，自动调整交叉概率和变异概率，在增加优秀个体的同时还能保证物种的多样性，提高算法的收敛速度和全局搜索能力。不过计算的复杂度较高，且需要较多的实验验证其自动调整的有效性，因此本文采用了普通的遗传算法，根据题目需求选择相对固定的交叉概率和变异概率，以保证算法的稳定性和可靠性。

当模型求得的结果逐步逼近最优解，我们设定持续不更新最优解时，其最大迭代次数约束条件如下：

$$\text{cnt} \leq 500 \tag{9}$$

基于以上分析，建立基于遗传算法求解VRP问题的模型如下：

目标函数：

$$\textbf{min}\ fitness = f(x) \tag{10}$$

其中x表示最终种群的最优个体，代表着送货策略。

约束条件：

$$\begin{cases} time_i \leq 360, \quad \forall i = 1, \ldots, n \\ load_i = \sum_{j \in path_i} demand_j \leq 25, \quad \forall i = 1, \ldots, n \\ allTime = \sum_{i=1}^{n} time_i \\ allMileage = \sum_{i=1}^{n} mileage_i \\ \text{cnt} \leq 500 \end{cases} \tag{11}$$

5.1.3 模型求解具体步骤

step1: 初始化种群

种群初始时是第0代种群，个体总数为1000，表示方式如下：

$$P(0) = \{x_1, x_2, \ldots, x_{1000}\} \tag{12}$$

其中每个个体都有一个基因序列gene，存储着快递员运输路线信息。如：

[2, 3, 4, 0, 1, 5, 0, 8, 7, 6, 0, 9, 10]

表示有4个快递员，第一个快递员的送货路线为2 3 4，第二个快递员的送货路线为1 5，第三个快递员的送货路线为8 7 6，第四个快递员的送货路线为9 10。（其中0作为分隔符标志）

step2: 根据适应度排序

根据式子10，计算每个个体的适应度值并排序，评估种群适应度。其中第一个个体为当前代种群中的最优解。

step3: 迭代更新策略

选择后半部分个体，将其基因序列用前半部分个体进行替换，确保基因优良性；同时将这部分个体进行交叉和变异操作，以增加种群的多样性。交叉变异操作：将该基因序列中的两个基因位点进行交换，使得基因序列改变，但总元素的构成不变，以增加种群的多样性。

种群排序后，具体迭代更新策略关系如下：

$$\text{更新} = \begin{cases} \text{保持不变，正常情况} & \text{种群前半部分个体} \\ \text{变为前半部分的基因序列，模拟进化并进行突变} & \text{种群后半部分个体} \end{cases} \tag{13}$$

step4: 突变操作分析

由于突变操作会使得基因序列不满足题目要求，如：位点2变成1，生成的基因序列中有两个1而没有2，即快递员到1号送货点送两次货，而没有快递员往2号点送货，显然不符合题目要求，因此本题不采用基因突变操作。

step5: 终止条件

根据式9作为迭代终止条件，最终对基因序列和其他信息进行整合，得到送货策略，并绘制出不同快递员的具体行驶路线图，如图 3 所示。由该图明显看出，各个快递员的送货路线合理，每个快递员负责的送货区域很少出现重叠，且不会出现重复送货，遗漏送货点的情况，同时总体送货时间和里程数较少，符合题目要求。

利用公式5，6，将求得的具体结果数据进行整合，得到的结果如下表 2 所示。

由上表可知，完成派送任务总耗时为195.62分钟，总行驶里程为383.81千米。且所有快递员的负载均小于25kg，行驶里程均较少，符合题目要求。

step6: 参数调整

由于初始时，设定好了权重系数$w_t = 1$，$w_s = 1$，$w_n = 10$，为了更好地优化送货策略，可以对权重系数进行动态调整，以寻求更好的送货效果。

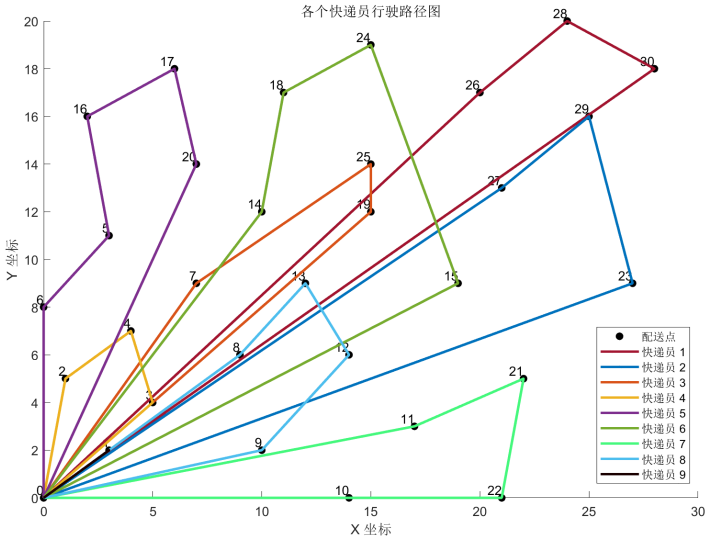


图 3: 快递员送货路线图

表 2: 送货策略

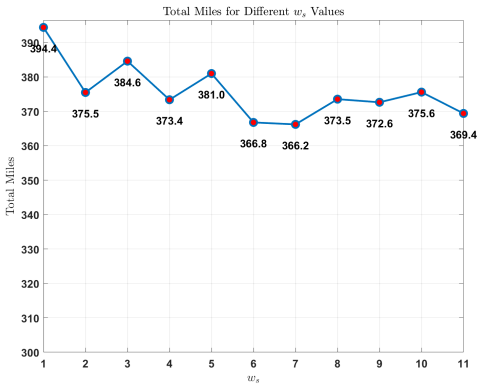
快递员	路径(不包括总部点)	负载(kg)	行驶里程(km)
1	30 28 26	20.20	69.01
3	27 29 23	22.50	65.44
4	19 25 7	24.60	42.05
5	3 4 2	19.70	18.27
6	20 17 16 5 6	21.40	41.59
7	14 18 24 15	22.30	56.99
8	11 21 22 10	23.60	48.75
10	9 12 13 8	22.20	34.52
22	1	8.00	7.21
总时间(minute)			195.62
总行驶里程(km)			383.81

在保持其他两个参数不变的情况下，分别对w_s和w_n进行[1,101]范围内的参数调整，都设置为10个等长步长，对送货策略进行优化。由于送货时间满足约束即可，对于送货策略的影响较小，因此不进行参数调整，而是始终$w_t = 1$作为参数对照。最终计算求解并进行图表可视化，比较不同因素的权重对送货策略的影响。

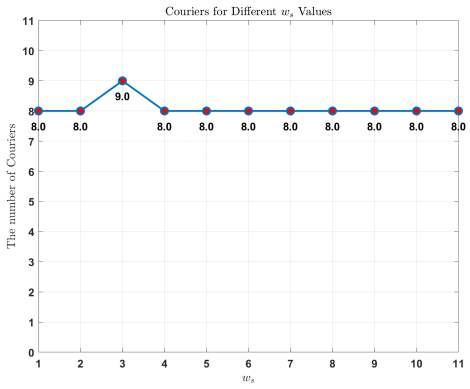
先保持$w_n = 10$不变，对w_s进行调整，得到的结果如图 4a 和图 4b 所示。

由上图可见，不同的里程权重w_s下，总里程和快递员数量的总体区别不大，不过随着w_s的增大，总里程有减小的趋势。说明里程权重w_s能使模型更偏向于取总里程小的作为优化目标，但是整体对模型的影响较小。

随后保持$w_s = 1$不变，对w_n进行调整，得到的结果如图 5a 和图 5b 所示。

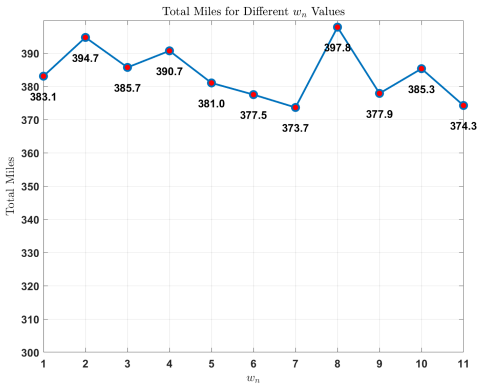


(a) w_s对总里程的影响

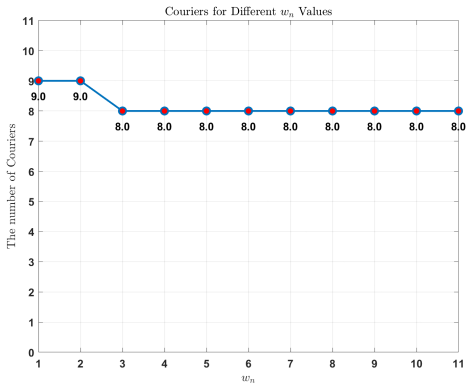


(b) w_s对快递员数量的影响

图 4: 不同里程权重w_s的影响情况



(a) w_n对总里程的影响



(b) w_n对快递员数量的影响

图 5: 不同快递员数量权重w_n的影响情况

由上图可见，不同的快递员数量权重w_n下，总里程和快递员数量的总体区别不大，随着w_n的增大，总快递员人数由9减到8后保持不变，而总里程的变化没有明显的规律，说明快递员数量权重w_n对模型的影响较小，在一定参数范围内基本可以忽略该权重对模型的影响。

step7: 结果分析

利用遗传算法求解VRP问题，效果良好。在权重参数的调整过程中，里程权重w_s对总里程有一定的影响，而快递员数量权重w_n对送货策略的影响较小。这是由于遗传算法作为一种启发式算法，本身求得的最优解便会受种群初始化等因素的影响，具有一定的随机性，因此对于设置不同的权重参数，模型受到的影响较小。由于加大送货员数量的w_n权重，刚开始确实能减少送货员数量，但后面不变，因此可以在分配给w_n一定权重的基础上，适当加大w_s的权重，以达到更好的送货策略。

5.2 模型二的建立与求解

5.2.1 模型的建立

目标函数是成本最小化，因此需要尽可能少的业务员，在满足约束条件的情况下，使得总成本最小。因此相比于第一问，目标函数有所变化，公式如下14所示：

$$fitness = allCost = \sum_{i=1}^{n} money_i \tag{14}$$

其中，n 是总共所需快递员的数量，$money_i$ 是第 i 个快递员所获得的酬金。具体一个快递员的酬金计算公式如下：

首先是从快递公司出发到每个送货点的酬金：

$$money_i = money_i + 3 \cdot mileage_i \cdot demand_j \tag{15}$$

上式中，$mileage_i$ 是快递员 i 当前的行驶里程，$demand_j$ 是快递员 j 的送货点j的送货量。式中累加的部分，表示的是快递员 i 从快递公司出发，携带货物$demand_j$ 到送货点 j 的酬金。

每当计算完一个送货点后，last 会更新为当前送货点 j，以便为下一个送货点计算距离：

$$last = j \tag{16}$$

此外，当前快递员的总里程也需要更新：

$$mileage_i = mileage_i + D_{last,j} \tag{17}$$

然后是快递员从最后一个送货点返回快递公司的酬金：

$$money_i = money_i + 2 \cdot D_{last,1} \tag{18}$$

上式中，$D_{last,1}$ 表示从最后一个送货点 last 返回快递公司（节点 1）的距离。

此外，在本问的优化求解过程中也体现出贪心策略，即需要先送进的，先送重的快递，否则在前段时间中快递员会承载更重的快递，导致总成本增大。

5.2.2 模型的求解

step1: 种群初始化

在模型一的基础上，对种群初始化的合法判断作了修改。由于快递员在不同情况下的行驶速度不同，因此分不同情况来计算耗时，对随机序列的合法性进行判断：若生成的某一个快递员行驶路线所需时间超过6小时，则认为该个体不合法，需要重新生成基因序列。

step2: 优化目标

初始化以及种群个体发生变异后，都会执行更新update函数，在写入最终数据的同时校验变异的合法性。在更新函数中，由于此时的基因序列已经确定，各个快递员的行驶路径已知，因此可以根据新的目标函数14来计算总成本，并以其值最小化作为优化目标，对种群个体的优劣程度进行排序，进而不断迭代筛选选出接近最优解的送货策略。

step3: 结果展示

仍然使用第一问的迭代更新策略，在不同约束以及目标函数的驱动下，不断产生逼近最优解的结果。最终得到的送货策略如图 6 所示。

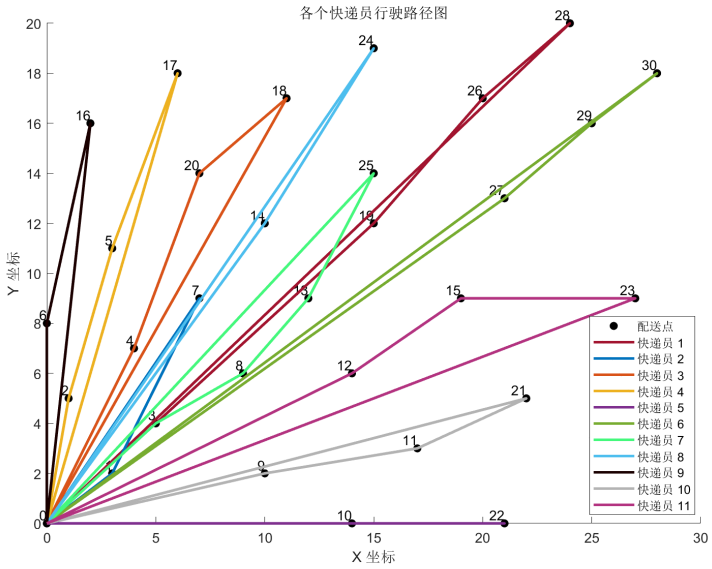


图 6: 快递员送货路线图

利用公式5，6和14，将求得的具体结果数据进行整合，得到最终结果数据如下表 3 所示。

由图示可知：该最优策略体现了贪心的思想，即先送进的，先送重的快递，最终再送远处的快递并不承载货物返回。体现在图中就是多条折线向远处走，最终直线返回原点的形状。若不采取这种方案，则会导致返回公司时还有货物需要承载，行驶速度慢且费用较高，增加了额外的成本。

step4: 模型比较

表 3: 送货策略

快递员	路径(不包括总部点)	负载(kg)	行驶里程(km)
4	19 26 28	23.80	62.52
10	1 7	15.20	23.07
12	4 20 18	17.60	40.93
13	2 5 17	18.50	38.01
18	10 22	13.30	42.00
21	27 29 30	24.30	66.59
24	3 8 13 25	23.70	41.47
25	14 24	11.40	48.43
27	6 16	6.50	32.37
29	9 11 21	11.70	45.22
30	12 15 23	18.50	57.52
总时间(minute)			196.48
总行驶里程(km)			498.13
总成本(元)			10505

利用Python的deap库，对本小问再次进行遗传算法的实现，得到最终的送货策略，如图 7 所示。

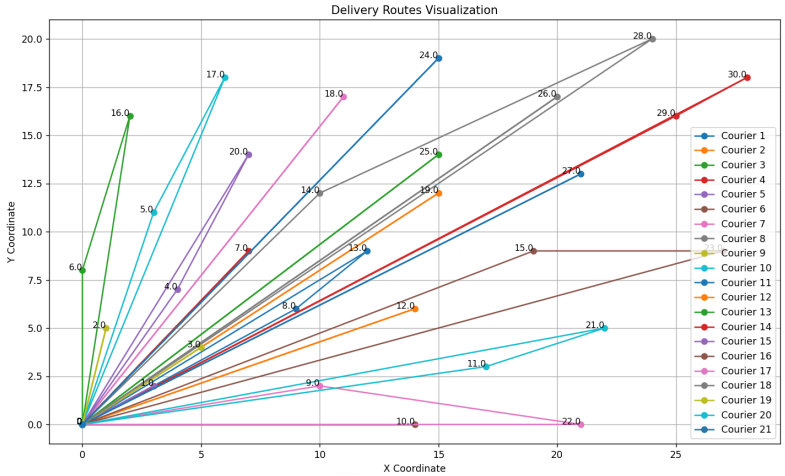


图 7: 快递员送货路线图

将该实现方法与之前独自实现的遗传算法进行对比，结果数据如下：

由表4可见，两种实现方法得到的结果基本一致，但deap库实现的总成本略高于独自实现的总成本，总时间也较高,行驶里程明显更高。这是因为使用deap库时，传的参数较为固定，导致计算出来的快递员数量过多。而独自实现的遗传算法可以根据题目

表 4: 遗传算法实现结果对比

指标	独自实现	deap库实现
总时间(minute)	196.48	186.43
总行驶里程(km)	498.13	761.33
总成本(元)	10505	10769

需求进行更灵活的调整，因此计算得出的总成本较低。

step5: 结果分析

相较于第一问，本问题求解出分配的快递员数量较多，这是因为快递员携带快件时的速度较慢，且所获得的酬金较高。若安排的快递员较少，则平均每个快递员所需经过的派送点就越多，派送过程中会大大增加成本。虽然相比第一问，本问求解的总里程明显增加，但主要是多个快递员都需要从远处返回快递公司造成的，而在这部分时间段内，快递员手中已无货物，回总部的时间更短，酬金更少，对总成本的消耗影响较小。因此在总成本最小化的情况下，需要增加快递员的数量，才能解得更好的结果。

5.3 模型三的建立与求解

5.3.1 模型的建立

延长平均快递员的工作时间至8小时，意味着可以使用更少的派送员，派送更多的快件，因此送货策略会有所变化。

修改目标函数，即适应度评价指标，由于延长了快递员的工作时间，因此可将时间影响因子减小，而适当增大总里程以及快递员数量的影响因子，从而使得方案的总成本较小，在规定时间内高效地派送完所有快递件。公式如下19所示：

$$fitness = allTime + 20 \cdot allMileage + 10 \cdot n \tag{19}$$

其中，$w_t = 1$，$w_s = 20$，$w_n = 10$。由于延长了快递员的工作时间，因此可将时间影响因子减小，而适当增大总里程以及快递员数量的影响因子。由第一问的权重分析可知，w_n对模型结果的影响很小，而w_s的权重适当增大可以使得行驶总里程有小幅度的降低，进而使得方案的总成本较小，在规定时间内高效地派送完所有快递件。所以，本问我们仍在满足时间约束的情况下，分配给w_s较大的权重。

5.3.2 模型的求解

step1: 目标函数的确定

利用公式19，调整模型的目标函数，重新计算适应度值，评估种群适应度。

step2: 遗传算法求解过程

仍然采用问题1的遗传算法，对种群进行初始化，评估适应度，选择、交叉、变异等操作，通过新的目标函数来筛选优良个体，不断迭代更新种群，直至满足终止条件。

step3: 结果展示

最终将数据整合并进行图表可视化，得到的送货策略如下图 8 所示。可见快递员数量较少，单个快递员负责的派送点更多，因此图中不同路线区域重叠的情况较多。

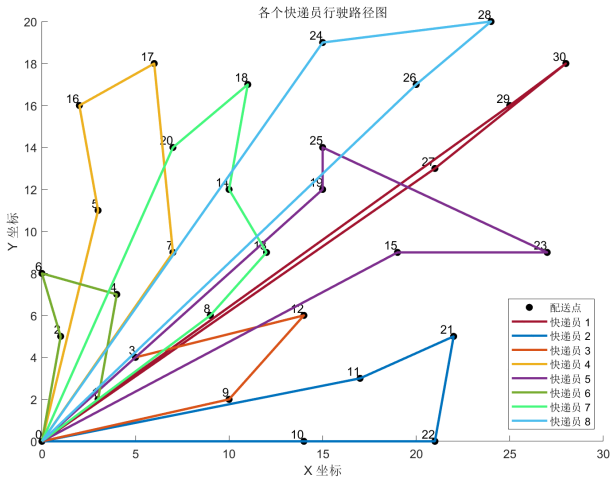


图 8: 快递员送货路线图

利用公式5，6和19，将求得的具体结果数据进行整合，得到最终结果数据如下表 5 所示。

表 5: 送货策略

快递员	路径(不包括总部点)	负载(kg)	行驶里程(km)
1	27 30 29	24.30	66.59
2	11 21 22 10	23.60	48.75
3	3 12 9	20.10	31.48
4	5 16 17 7	21.00	41.43
5	19 25 23 15	23.20	63.23
6	2 6 4 1	24.70	21.09
7	20 18 14 13 8	24.00	44.42
8	26 28 24	23.60	64.51
总时间(minute)			191.76
总行驶里程(km)			381.49

由上表可知，完成派送任务总耗时为191.76分钟，总行驶里程为381.49千米。且所有快递员的负载均小于25kg，行驶里程均较少，符合题目要求。

step4: 模型比较

与第一问求得表2中的数据相比，本问的送货策略中快递员数量较少，总耗时与总行驶里程也均略小于第一问。这是因为延长了快递员的工作时间，使得每个快递员可以负责更多的送货点，因此需要的快递员数量较少。同时，由于目标函数的调整，使得总成本较小，总时间和总里程也相应减少，使得送货策略更加高效。

六、 模型的评价与推广

6.1 模型的优点

1. 采用遗传算法这种启发式算法，可以有效解决VRP问题，且计算量相比于普通的线性规划方法要小得多，能够有效解决实际派送问题。若采用传统的规划方法，即使变种很多但计算量仍过大，在一定时间内难以收敛，不适合解决这种问题。

2. 遗传算法的使用更灵活，可根据实际情况调整参数，如种群大小，迭代次数，突变的情况等，以达到更好的效果。

3. 遗传算法通过模拟生物进化过程，能够在解空间中进行广泛搜索，有效避免陷入局部最优解，从而具有更强的全局搜索能力。通过生成一组初始解（即不同的配送路径方案），作为搜索的起点。通过不断地进行选择、交叉和变异操作，不对迭代优化解。

4. 遗传算法在搜索过程中，同时对多个解进行评估和选择，这种并行处理机制提高了算法的搜索效率。

6.2 模型的缺点

1. 遗传算法的效果受到参数的影响，参数若初始化时较为不切实际，后续迭代容易陷入局部最优解而与全局最优解差距过大，因此需要根据实际情况进行调整，谨慎选择参数以及后续的调整策略。

2. 遗传算法属于启发式算法，相比于传统规划模型，缺少足够的理论支撑，因此在一些特殊情况下，可能无法得到接近最优解的答案。需要根据实际情况进行调整，如提供先验知识，增加约束条件等。赵 [11]等人提出了一种新的解决方案，可以采用双种群的遗传算法来进行交叉求解。在一次迭代完成后，通过交换两个种群间的优秀个体所携带的遗传信息，以打破种群内的平衡态，从而避免过早陷入局部最优解，最终能达到更好的效果。

3. 遗传算法在搜索过程中没有能够及时利用网络的反馈信息，这导致算法的搜索速度相对较慢。为了获得较精确的解，往往需要较多的训练时间。

4. 在遗传算法的进化过程中，如果适应度值较高的个体大量繁殖，可能导致种群的多样性丧失，出现近亲繁殖现象。这将导致算法陷入局部最优解，无法找到全局最优解，即出现早熟收敛问题。

6.3 模型的推广

1. 遗传算法可以借助蒙特卡洛等其他启发式算法的思想，通过先验知识，概率分配等方法，进行模型的优化，有助于使用更大的参数，更多的迭代次数同时不对时间产生过多影响，提高模型的效率，达到更好的效果。

2. 对于这种启发式算法，可以推广到其他类似的交通问题上，如物流配送，货物装载等问题。郭 [13]等人便提及了遗传算法在外卖路径优化方面的应用，徐 [8]等人也使用了遗传算法来运河智能选线研究。只需要根据实际情况进行调整，更换适合的约束条件，即可解决其他问题。

3. 在模式识别领域，遗传算法可以被用于手写数字识别、图像识别和语音识别等任务。通过将模式识别问题转化为优化问题，遗传算法可以在搜索空间中找到最优解，提高识别的准确性。

4. 在其他领域也有遗传算法的应用，如金融领域的投资组合优化、电力系统的优化调度、机器学习领域的特征选择等。 [4]等人讲述了遗传算法在信号处理领域中的应用。因此，遗传算法具有广泛的应用前景，理解好模型的启发式等性质，可以在不同领域中发挥重要作用。

参考文献

[1] Gitae Kim, Yew-Soon Ong, Chen Kim Heng, Puay Siew Tan, and Nengsheng Allan Zhang. City vehicle routing problem (city vrp): A review. *IEEE Transactions on Intelligent Transportation Systems*, 16(4):1654–1666, 2015.

[2] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm-a literature review. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 380–384. IEEE, 2019.

[3] Tom V Mathew. Genetic algorithm. *Report submitted at IIT Bombay*, 53, 2012.

[4] Kit-Sang Tang, Kim-Fung Man, Sam Kwong, and Qianhua He. Genetic algorithms and their applications. *IEEE signal processing magazine*, 13(6):22–37, 1996.

[5] Mourad Zirour. Vehicle routing problem: models and solutions. *Journal of Quality Measurement and Analysis JQMA*, 4(1):205–218, 2008.

[6] 刘祥坤. 基于改进遗传算法的物流配送路径优化研究. Master's thesis, 长春工业大学, 2023.

[7] 张子为. 基于遗传算法的vrp扩展模型求解方法研究. Master's thesis, 安庆师范大学, 2020.

[8] 徐翔宇. 基于遗传算法的运河智能选线研究. Master's thesis, 重庆交通大学, 2024.

[9] 王刚. 遗传算法在vrp中的应用与研究. Master's thesis, 重庆交通大学, 2011.

[10] 王雪兵. 基于遗传算法的y物流公司配送路径优化研究. Master's thesis, 中北大学, 2021.

[11] 赵燕伟, 吴斌, 蒋丽, 董红召, and 王万良. 车辆路径问题的双种群遗传算法求解方法. *计算机集成制造系统*, 10(3):0, 2004.

[12] 郭庆腾. 改进遗传算法求解车辆路径问题研究. Master's thesis, 青岛大学, 2024.

[13] 郭靖文. 基于遗传算法的外卖路径优化研究. Master's thesis, 辽宁工程技术大学, 2023.

[14] 黄尚. 遗传算法在物流配送中的研究与应用. Master's thesis, 广东工业大学, 2013.

附录

问题一主函数代码

```
function main()
  % 从excel里面读取数据
  filename = '../data/dp.xlsx';
  cNode = 30;  % 送货节点数量（不包括快递公司）
  % 30个送货点、1个总公司，共31个
  x = zeros(cNode + 1, 1);
  y = zeros(cNode + 1, 1);
  demand = zeros(cNode + 1, 1);
  % 配送中心的坐标
  x(1) = 0;
  y(1) = 0;
  demand(1) = 0;
  % 读取送货目标点的坐标和需求量，共30行数据
  targets = readmatrix(filename, 'Range', sprintf('B2:D%d', 2 + cNode -
      1));
  x(2:end) = targets(:, 1);
  y(2:end) = targets(:, 2);
  demand(2:end) = targets(:, 3);
  % 读取快递员数量
  cCourier = 30;
  % 读取快递员的容量和距离限制
  capacity = 25 * ones(cCourier, 1);
  timeLimit = 360 * ones(cCourier, 1);
  % 求解 VRP
  [path] = Export(cNode + 1, x, y, demand, cCourier, capacity,
     timeLimit, 1, 1, 10, @Callback);
  % 调用画图函数
  img(x, y, path);
end

% 类似接口函数，提供VRP接口
function [path] = Export(numNode, x, y, demand, numCourier, capacity,
    timeLimit, k1, k2, k3, Callback)
  % 创建 VRP 对象
  vrp = VRP();
  % 添加节点信息
  for i = 1:numNode
    vrp.addNode(x(i), y(i), demand(i));
  end
  % 添加快递员信息
  for i = 1:numCourier
    vrp.addCourier(capacity(i), timeLimit(i));
  end
  % 设置权重
  vrp.setWeights(k1, k2, k3);
  % 调用vrp中的函数求解
  res = vrp.solve();
  % 提取结果
  path = cell(numCourier, 1);  % 每个元素都是一个路径集合，而不是一个值
  pathLen = zeros(numCourier, 1); % 记录该辆车经过的节点个数
  load = zeros(numCourier, 1);
  mileage = zeros(numCourier, 1);
```

```
  time = res.time;
  for i = 1:numCourier
    cNode = length(res.path{i});
    path{i} = res.path{i};
    pathLen(i) = cNode;
    load(i) = res.load(i);
    mileage(i) = res.mileage(i);
  end
  % 调用回调函数
  Callback(numCourier, path, pathLen, load, mileage, time);
end

function Callback(numCourier, path, pathLen, load, mileage, time)
  % 回调函数
  allMile = 0.0;  % 总里程
  % 输出每辆车的路径、负载和行驶里程
  for i = 1:numCourier
    if pathLen(i) > 0
      fprintf('time %d:\t', i);
      fprintf('%d ', path{i});
      fprintf('load: %.2f mileage: %.2f\n', load(i), mileage(i));
      allMile = allMile + mileage(i);
    end
  end
  fprintf('allTime: %.2f allMile: %.2f\n', time, allMile);
end
```

VRP问题求解代码

```
classdef VRP < handle
  properties
    cNode
    cCourier
    k1
    k2
    k3
    nodeInfo
    courierInfo
    dis
    update_num
  end

  methods
    function obj = VRP()
      obj.cNode = 0;
      obj.cCourier = 0;
      obj.k1 = 1.0;
      obj.k2 = 1.0;
      obj.k3 = 1.0;
      obj.nodeInfo = [];
      obj.courierInfo = [];
      obj.update_num = 0;
    end

    function obj = addNode(obj, x, y, demand)
      obj.nodeInfo = [obj.nodeInfo, Node(obj.cNode, x, y, demand)];
```

```
    obj.cNode = obj.cNode + 1;
  end

  function obj = addCourier(obj, capacity, disLimit)
    obj.courierInfo = [obj.courierInfo, Courier(obj.cCourier,
        capacity, disLimit)];
    obj.cCourier = obj.cCourier + 1;
  end

  function obj = setWeights(obj, k1, k2, k3)
    obj.k1 = k1;
    obj.k2 = k2;
    obj.k3 = k3;
  end

  function res = solve(obj)
    % 预处理所有边的距离（欧式距离）
    obj.dis = zeros(obj.cNode, obj.cNode);
    for i = 1:obj.cNode
      for j = 1:obj.cNode
        obj.dis(i, j) = sqrt((obj.nodeInfo(i).x - obj.nodeInfo(j).x)
            ^2 + (obj.nodeInfo(i).y - obj.nodeInfo(j).y)^2);
      end
    end
    % 初始化种群，每个个体就是一个基因序列，代表着一个可能的结果
    chroms = [];
    while length(chroms) < 1000
      c = Chrom(obj);
      if c.valid
        chroms = [chroms, c];
      end
    end
    % 遗传算法
    % cnt 是一个计数器，用于记录遗传算法中连续未找到更优解的代数
    % 当 cnt 达到某个预设的阈值时，算法会终止，用于避免算法陷入无限循环
    cnt = 0;
    numGeneration = 0;  % 代数：种群繁衍到第几代
    best = Chrom(obj);
    while true
      numGeneration = numGeneration + 1;
      % 重载了 lt < ，因此内置的sort函数能根据fitness进行升序排序
      chroms = sort(chroms);
      % 对1000个个体进行排序后，第一个就算当前代的最优解
      if chroms(1) < best
        obj.update_num = obj.update_num + 1;
        best = chroms(1);
        cnt = 0;
      else
        cnt = cnt + 1;
      end
      % 无效种群迭代过多，终止迭代（可能已经逼近最优解了，无法再继续了）
      if cnt >= 500
        break;
      end
      % 后面一半的个体，赋值为前一半那些优良个体，同时加入基因突变
      % 这样确保优良迭代的同时，也模拟了不确定性
      halfSize = floor(length(chroms) / 2);
      for i = halfSize+1 : length(chroms)
```

```
            chroms(i) = chroms(i - halfSize);
            chroms(i) = chroms(i).mutation();
        end
        % 重新生成（双种群初始化可以参考借鉴）
        % for i = halfSize+quarterSize:length(chroms)
        %  chroms(i) = Chrom(obj);
        % end
      end
      % 解码结果
      res = best.decode();
      res.numGeneration = numGeneration;
    end
  end
end
```

种群个体类代码

```
classdef Chrom
  properties
    nodeInfo
    courierInfo
    dis
    gene
    time          % 时间矩阵
    mileage       % 里程矩阵
    load          % 负载矩阵
    allTime       % 总时间
    allMileage    % 总里程
    n             % 快递员人数
    valid
    k1
    k2
    k3
  end

  methods
    % 构造方法
    function obj = Chrom(vrp)
      % 初始化节点信息和快递员信息
      obj.nodeInfo = vrp.nodeInfo;
      obj.courierInfo = vrp.courierInfo;
      obj.dis = vrp.dis;
      obj.k1 = vrp.k1;
      obj.k2 = vrp.k2;
      obj.k3 = vrp.k3;
      cNode = length(obj.nodeInfo) - 1;
      cCourier = length(obj.courierInfo);
      % 记录各辆车的行驶时间、里程和装载货物重量，共cCourier辆车
      obj.time = zeros(cCourier, 1);
      obj.mileage = zeros(cCourier, 1);
      obj.load = zeros(cCourier, 1);
      % 初始化基因序列，由所有节点编号组成
      obj.gene = 1:cNode;
      % rng('shuffle'); obj.gene = obj.gene(randperm(cNode));
      % 对基因序列进行结果的初始化，得到一个满足题目要求但不是最优解的结果
      % 序列之间用0隔开，代表不同车运输的情况
```

```
    index = 1;
    for i = 1:cCourier - 1  % 循环cCourier-1次就插入这么多次分隔符0
      % 而且每次车都尽可能装满货物，不可能出现最后一辆车还装不下的情况
      sum = 0.0;
      last = 1;
      allTime = 0.0;
      while true
        % 约束：不能越界、不能超重、总里程不能超过约束!
        % 1、不能越界
        if index > length(obj.gene)
          break;
        end
        % 2、不能超重
        sum = sum + obj.nodeInfo(obj.gene(index)+1).demand;
        if sum > obj.courierInfo(i).capacity
          break;
        end
        % 3、不能超时间（停留一次送货点10分钟）
        allTime = allTime + obj.dis(last, obj.gene(index)+1)*2.4 +
            10;
        last = obj.gene(index)+1;
        if (allTime + obj.dis(last, 1)) > obj.courierInfo(i).
            timeLimit
          break;
        end
        % 该辆车没超重，没超出总里程，就尝试前往下一个城市运输货物
        index = index + 1;
      end
      % 最终退出循环时，快递员只能装到index-1这个地方的货物
      % index这个节点装不下的/总里程过长/越界的，所以才退出循环
      % 因此基因要在index处插入0，作为快递员运输货物的节点集合分隔符
      obj.gene = [obj.gene(1:index-1), 0, obj.gene(index:end)];
      % 跳过分隔符，由下一辆车开始，运输下一座城市的货物
      index = index + 1;
    end
    % 第一次更新不会超重，前面初始化已经做过处理了
    obj = obj.update();
  end

  function obj = update(obj)
    % 更新时间、里程和载荷
    for i = 1 : length(obj.mileage) % 其实就是快递员数量
      obj.time(i) = 0.0;
      obj.mileage(i) = 0.0;
      obj.load(i) = 0.0;
    end
    iCourier = 1;    % 快递员索引
    last = 1;    % 节点索引，1代表快递公司处，其他代表送货点
    % 对基因序列进行遍历
    for i = 1 : length(obj.gene)
      % 遇到分隔符
      if obj.gene(i) == 0
        % 送完了，要返回快递公司，因此需要加上当前位置last到快递公司1的距离
        obj.mileage(iCourier) = obj.mileage(iCourier) + obj.dis(last,
            1);
        % 耗时约束:  当前耗时 + 回到公司耗时 < 总时间约束
        % 这样才符合条件，否则直接 false 并 return
```

```
            obj.time(iCourier) = obj.time(iCourier) + obj.dis(last, 1) *
                2.4;
            if obj.time(iCourier) > obj.courierInfo(iCourier).timeLimit
              obj.valid = false;
              return;
            end
            iCourier = iCourier + 1;  % 下一辆车来送
            last = 1;          % 从快递公司出发
          % 没遇到分隔符
          else
            % 之前承载加上当前送货点的重量
            obj.load(iCourier) = obj.load(iCourier) + obj.nodeInfo(obj.
                gene(i)+1).demand;
            % 超重，标记valid为false，直接return掉，后面不做处理
            if obj.load(iCourier) > obj.courierInfo(iCourier).capacity
              obj.valid = false;
              return;
            end
            % 不满足先return，满足了才进行下面一系列操作
            % 当前快递员耗时
            obj.time(iCourier) = obj.time(iCourier) + obj.dis(last, obj.
                gene(i)+1)*2.4 + 10;
            % 从上次位置last出发，到当前点的距离
            % （距离矩阵中第一个点是快递公司，所以要+1）
            obj.mileage(iCourier) = obj.mileage(iCourier) + obj.dis(last,
                 obj.gene(i)+1);
            % 迭代，位置更新
            last = obj.gene(i)+1;
          end
        end
        % 最后一辆任务车还要加上回快递公司的距离
        obj.mileage(iCourier) = obj.mileage(iCourier) + obj.dis(last, 1);
        % 最后一辆车的耗时约束
        obj.time(iCourier) = obj.time(iCourier) + obj.dis(last, 1) * 2.4;
        if obj.time(iCourier) > obj.courierInfo(iCourier).timeLimit
          obj.valid = false;
          return;
        end
        % time length cnt 就是三个优化评价指标
        % 总时间就是所有车之中耗时最久的
        obj.allTime = max(obj.time);
        % 所有快递员行驶总里程
        obj.allMileage = sum(obj.mileage);
        % 有进行运输的快递员数（有行驶里程的就是有运输的）
        obj.n = sum(obj.mileage > 0);
        % 前面false已经return掉了，此处肯定为true
        obj.valid = true;
      end

      % 基因变异操作
      function obj = mutation(obj)
        obj.valid = false;  % 变异合法且不变坏，才可以退出
        while ~obj.valid
          % 重新随机交换基因
          % rng('shuffle'); i = randi(length(obj.gene));
          j = randi(length(obj.gene));
          temp = obj.gene(i);
          obj.gene(i) = obj.gene(j);
```

```
        obj.gene(j) = temp;
        % 更新后验证合法性
        temp_time = obj.allTime;
        temp_length = obj.allMileage;
        temp_cnt = obj.n;
        obj = obj.update();  % 会为valid赋值的，更新成功才退出循环
        if ~obj.valid
          % 交换无效，恢复原状，但是valid还是false，因为根本没更新，不能退出循环
          temp = obj.gene(i);
          obj.gene(i) = obj.gene(j);
          obj.gene(j) = temp;
          obj.allTime = temp_time;
          obj.allMileage = temp_length;
          obj.n = temp_cnt;
        end
      end
    end

    function f = fitness(obj)
      % 计算适应度
      if ~obj.valid
        f = 1e8;    % 很大的数，说明不适应
      else
        % 根据三个指标，分配权重后，计算出最终评价指标值，越小越好!
        f = obj.k1 * obj.allTime + obj.k2 * obj.allMileage + obj.k3 *
            obj.n;
      end
    end

    % 重载比较运算符，这样sort就知道要根据fitness的大小进行升序排序了
    % 最小的就算是所有个体的最优解
    function b = lt(obj, c)
      % 比较适应度
      b = obj.fitness() < c.fitness();
    end

    % 上面重载还不够，需要再指定一个排序函数
    function sortedArr = sort(objArray)
      [~, idx] = sort(arrayfun(@(x) x.fitness(), objArray));
      sortedArr = objArray(idx);
    end

    function res = decode(obj)
      % 解码结果
      res.path = cell(length(obj.courierInfo), 1);
      res.load = obj.load;
      res.mileage = obj.mileage;
      res.time = obj.allTime;
      % 结果就是一个字典，索引是快递员，值是该快递员经过的节点集合
      iCourier = 1;
      for i = 1:length(obj.gene)
        if obj.gene(i) == 0
          iCourier = iCourier + 1;
        else
          res.path{iCourier} = [res.path{iCourier}, obj.gene(i)];
        end
      end
    end
```

```
  end
end
```

问题二种群个体类代码

由于问题二的目标函数以及约束条件不同，因此需要重新定义种群个体类，代码如下：

```
classdef Chrom
  properties
    nodeInfo
    courierInfo
    dis
    gene
    time          % 时间矩阵
    mileage       % 里程矩阵
    load          % 负载矩阵
    money         % 酬金矩阵
    allTime       % 总时间
    allMileage    % 总里程
    n   % 快递员人数
    allCost       % 总费用
    valid
    k1
    k2
    k3
  end

  methods
    % 构造方法
    function obj = Chrom(vrp)
      % 初始化节点信息和快递员信息
      obj.nodeInfo = vrp.nodeInfo;
      obj.courierInfo = vrp.courierInfo;
      obj.dis = vrp.dis;
      obj.k1 = vrp.k1;
      obj.k2 = vrp.k2;
      obj.k3 = vrp.k3;
      cNode = length(obj.nodeInfo) - 1;
      cCourier = length(obj.courierInfo);
      % 记录各辆车的行驶时间、行驶里程、装载货物重量、酬金，共cCourier辆车
      obj.time = zeros(cCourier, 1);
      obj.mileage = zeros(cCourier, 1);
      obj.load = zeros(cCourier, 1);
      obj.money = zeros(cCourier, 1);
      % 初始化基因序列，由所有节点编号组成
      obj.gene = 1:cNode;
      obj.gene = obj.gene(randperm(cNode));
      % 对基因序列进行结果的初始化，得到一个满足题目要求但不是最优解的结果
      % 序列之间用0隔开，代表不同车运输的情况
      index = 1;
      for i = 1:cCourier - 1   % 循环cCourier-1次就插入这么多次分隔符0
        % 而且每次车都尽可能装满货物，不可能出现最后一辆车还装不下的情况
        sum = 0.0;
        last = 1;
```

```
        allTime = 0.0;
        while true
          % 约束：不能越界、不能超重、总里程不能超过约束
          % 1、不能越界
          if index > length(obj.gene)
            break;
          end
          % 2、不能超重
          sum = sum + obj.nodeInfo(obj.gene(index)+1).demand;
          if sum > obj.courierInfo(i).capacity
            break;
          end
          % 3、不能超时间（此处必不是最后一次，因此20km/h）
          % 公里数/20*60 = *3 得到时间，+ 10 是要加上每个送货点的停留时间
          allTime = allTime + obj.dis(last, obj.gene(index)+1)*3 + 10;
          last = obj.gene(index)+1;
          % 最后空手回总部，速度30km/h，公里数/30*60得到时间
          if (allTime + obj.dis(last, 1)*2) > obj.courierInfo(i).
              timeLimit
            break;
          end
          % 该辆车没超重，没超出总时间，就尝试前往下一个城市运输货物
          index = index + 1;
        end
        % 最终退出循环时，快递员只能装到index-1这个地方的货物
        % 因此基因要在index处插入0，作为快递员运输货物的节点集合分隔符
        obj.gene = [obj.gene(1:index-1), 0, obj.gene(index:end)];
        % 跳过分隔符，由下一辆车开始，运输下一座城市的货物
        index = index + 1;
      end

      % 第一次更新不会超重，前面初始化已经做过处理了
      obj = obj.update();
    end

    function obj = update(obj)
      % 更新时间、里程、载荷、酬金
      for i = 1 : length(obj.mileage) % 其实就是快递员数量
        obj.time(i) = 0.0;
        obj.mileage(i) = 0.0;
        obj.load(i) = 0.0;
        obj.money(i) = 0.0;
      end

      iCourier = 1;    % 快递员索引
      last = 1;    % 节点索引，1代表快递公司处，其他代表送货点
      % 对基因序列进行遍历
      for i = 1 : length(obj.gene)
        % 遇到分隔符
        if obj.gene(i) == 0
          % 送完了，要返回快递公司，因此需要加上当前位置last到快递公司1的距离
          obj.mileage(iCourier) = obj.mileage(iCourier) + obj.dis(last,
              1);
          % 耗时约束：  当前耗时 + 回到公司耗时 < 总时间约束，才符合条件
          obj.time(iCourier) = obj.time(iCourier) + obj.dis(last, 1) *
              2;
          if obj.time(iCourier) > obj.courierInfo(iCourier).timeLimit
            % disp('变异产生的新个体不合格')
```

```
            obj.valid = false;
            return;
          end
          % 最后加钱：当前快递员跑完一圈所获得的酬金
          obj.money(iCourier) = obj.money(iCourier) + 2 * obj.dis(last,
              1);
          iCourier = iCourier + 1;  % 下一辆车来送
          last = 1;          % 从快递公司出发
        % 没遇到分隔符
        else
          % 之前承载加上当前送货点的重量
          obj.load(iCourier) = obj.load(iCourier) + obj.nodeInfo(obj.
              gene(i)+1).demand;
          % 超重，标记valid为false，直接return掉，后面不做处理
          if obj.load(iCourier) > obj.courierInfo(iCourier).capacity
            obj.valid = false;
            return;
          end
          % 不满足先return，满足了才进行下面一系列操作
          % 当前快递员耗时
          obj.time(iCourier) = obj.time(iCourier) + obj.dis(last, obj.
              gene(i)+1)*3 + 10;
          % 从上次位置last出发，到当前点的距离（距离矩阵中第一个点是快递公司，所以要+1）
          obj.mileage(iCourier) = obj.mileage(iCourier) + obj.dis(last,
               obj.gene(i)+1);
          % 加酬金：先结算送到这个送货点的货物，所带来的酬金
          obj.money(iCourier) = obj.money(iCourier) + 3 * obj.mileage(
              iCourier) * obj.nodeInfo(obj.gene(i)+1).demand;
          % 迭代，位置更新
          last = obj.gene(i)+1;
        end
      end
      % 最后一辆任务车还要加上回快递公司的距离
      obj.mileage(iCourier) = obj.mileage(iCourier) + obj.dis(last, 1);
      % 最后一辆车的耗时约束
      obj.time(iCourier) = obj.time(iCourier) + obj.dis(last, 1) * 2;
      if obj.time(iCourier) > obj.courierInfo(iCourier).timeLimit
        obj.valid = false;
        return;
      end
      % time length cnt 就是三个优化评价指标
      % 总时间就是所有车之中行驶距离最长的，车速都一样，距离映射时间
      obj.allTime = max(obj.time);
      % 所有快递员行驶总里程
      obj.allMileage = sum(obj.mileage);
      % 有进行运输的快递员数（有行驶里程的就是有运输的）
      obj.n = sum(obj.mileage > 0);
      % 总费用：所有快递员的酬金
      obj.allCost = sum(obj.money);
      % 前面false已经return掉了，现在到这里的肯定为true
      obj.valid = true;
    end

    function obj = mutation(obj)
      obj.valid = false;  % 变异合法且不变坏，才让你退出
      while ~obj.valid
        % 重新随机交换基因
        % rng('shuffle'); i = randi(length(obj.gene));
```

```
        j = randi(length(obj.gene));
        temp = obj.gene(i);
        obj.gene(i) = obj.gene(j);
        obj.gene(j) = temp;
        % 更新后验证合法性
        temp_time = obj.allTime;
        temp_length = obj.allMileage;
        temp_cnt = obj.n;
        obj = obj.update();  % 会为valid赋值的，更新成功才退出循环
        if ~obj.valid
          % 交换无效，恢复原状，但是valid还是false，因为根本没更新，不能退出循环
          temp = obj.gene(i);
          obj.gene(i) = obj.gene(j);
          obj.gene(j) = temp;
          obj.allTime = temp_time;
          obj.allMileage = temp_length;
          obj.n = temp_cnt;
        end
      end
    end

    function f = fitness(obj)
      % 计算适应度
      if ~obj.valid
        f = 1e8;    % 很大的数，说明不适应
      else
        % 根据三个指标，分配权重后，计算出最终评价指标值，越小越好!
        % f = obj.k1 * obj.allTime + obj.k2 * obj.allMileage + obj.k3 * obj.n;
        % 总目标只有一个：总费用最少
        f = obj.allCost;
      end
    end

    % 重载比较运算符
    function b = lt(obj, c)
      % 比较适应度
      b = obj.fitness() < c.fitness();
    end

    function sortedArr = sort(objArray)
      % 根据元素中的函数计算出来的评价值不是属性，不能直接比较! 要用arrayfun
      [~, idx] = sort(arrayfun(@(x) x.fitness(), objArray));
      sortedArr = objArray(idx);
    end

    function res = decode(obj)
      % 解码结果
      res.path = cell(length(obj.courierInfo), 1);
      res.load = obj.load;
      res.mileage = obj.mileage;
      res.time = obj.allTime;
      res.cost = obj.allCost;
      % 结果就是一个字典，索引是快递员，值是该快递员经过的节点集合
      iCourier = 1;
      for i = 1:length(obj.gene)
        if obj.gene(i) == 0
          iCourier = iCourier + 1;
        else
```

```
            res.path{iCourier} = [res.path{iCourier}, obj.gene(i)];
          end
        end
      end
      function s = toString(obj)
        % 转为字符串，遍历基因中每个节点数字，转为字符串后再接空格不断拼接
        s = 'gene: ';
        for i = 1:length(obj.gene)
          s = [s, num2str(obj.gene(i)), ' '];
        end
      end
    end
end
```

问题三主函数代码

由于问题三只改变了业务员工作时间这个约束条件，因此只需要修改主函数中的参数以及评价指标的权重分配即可，主函数代码如下：

```
function main()
  % 从excel里面读取数据
  filename = '../data/dp.xlsx';
  cNode = 30;  % 送货节点数量（不包括快递公司）
  % 30个送货点、1个总公司，共31个
  x = zeros(cNode + 1, 1);
  y = zeros(cNode + 1, 1);
  demand = zeros(cNode + 1, 1);
  % 配送中心的坐标
  x(1) = 0;
  y(1) = 0;
  demand(1) = 0;
  % 读取客户的坐标和需求量，30行数据
  targets = readmatrix(filename, 'Range', sprintf('B2:D%d', 2 + cNode -
      1));
  x(2:end) = targets(:, 1);
  y(2:end) = targets(:, 2);
  demand(2:end) = targets(:, 3);
  % 读取快递员数量
  cCourier = 30;
  % 读取快递员的容量和距离限制
  capacity = 25 * ones(cCourier, 1);
  timeLimit = 480 * ones(cCourier, 1);  % 此处约束变为8小时，单位是分钟
  % 求解 VRP，权重与之前不同，此处分配为 1:2:10
  [path] = Export(cNode + 1, x, y, demand, cCourier, capacity,
     timeLimit, 1, 2, 10, @Callback);
  % 调用画图函数
  img(x, y, path);
end
```