

武汉大学国家网络安全学院

研究生课程实验报告

基于整数规划和遗传算法解决
带时间窗约束的多配送中心路径规划问题

指导教师： 林海

学生姓名： 沈静远

学生学号： 2023202210156

专业名称： 网络空间安全

二〇二四年六月

目 录

1	问题引入	1
1.1	VRP 问题	1
1.2	CVRP 问题	1
1.3	VRPTW 问题	1
1.4	多中心 VRPTW 问题	1
2	问题建模与算法设计	2
2.1	问题介绍	2
2.2	问题建模	2
2.3	不考虑订单延后	3
2.3.1	整数规划	3
2.3.2	遗传算法	4
2.4	考虑订单延后	4
3	整数规划	5
3.1	数据集准备	5
3.2	Gurobi	6
3.3	重要代码	6
3.4	结果展示	9
4	遗传算法	12
4.1	数据集准备	12
4.2	重要代码	13
4.3	结果展示	14
5	总结	16

1 问题引入

1.1 VRP 问题

车辆路径规划问题 (Vehicle Routing Problem, VRP) 一般指的是：对一系列发货点和收货点，组织调用一定的车辆，安排适当的行车路线，使车辆有序地通过它们，在满足指定的约束条件下（例如：货物的需求量与发货量，交发货时间，车辆容量限制，行驶里程限制，行驶时间限制等），力争实现一定的目标（如车辆空驶总里程最短，运输总费用最低，车辆按一定时间到达，使用的车辆数最小等）。

1.2 CVRP 问题

最基本的 VRP 问题叫做带容量约束的车辆路径规划问题 (Capacitated Vehicle Routing Problem, CVRP)。在 CVRP 中，需要考虑每辆车的容量约束、车辆的路径约束和装载量约束。

1.3 VRPTW 问题

为了考虑配送时间要求，带时间窗的车辆路径规划问题 (Vehicle Routing Problem with Time Window, VRPTW) 应运而生。

VRPTW 不仅考虑 CVRP 的所有约束，还需要考虑时间窗约束，也就是每个顾客对应一个时间窗 $[e_i, l_i]$ ，其中 e_i 和 l_i 分别代表该点的最早到达时间和最晚到达时间。顾客点 $i \in V$ 的需求必须要在其时间窗内被送达。

VRPTW 已经被证明是 NP-hard 问题，其求解复杂度随着问题规模的增加而急剧增加，求解较为困难。

1.4 多中心 VRPTW 问题

多配送中心的 VRP 问题是经典 VRP 的扩展，它研究的是有多个配送中心同时对若干个客户进行服务，每个客户都有一定的货物需求。所要确定的是客户点由哪个配送中心服务，并对服务时的车辆路径进行优化，以便达到成本最低、时间最短等目标。

本结课作业讨论解决的问题就是一个多中心 VRPTW 问题，多配送中心 VRPTW 问题的求解，要比普通 VRP 难度大得多，目前对于该问题的求解大多拆分为两个阶段进行：首先先将客户点分配给配送中心，转化为单配送中心问题；在分别对每一个配送中心的路径进行优化。

2 问题建模与算法设计

2.1 问题介绍

无人机可以快速解决最后 10 公里的配送，本作业要求设计一个算法，实现区域内的无人机配送的路径规划。在区域中，共有 j 个配送中心，任意一个配送中心有用户所需要的商品，其数量无限，同时任一配送中心的无人机数量无限。该区域同时有 k 个卸货点（无人机只需要将货物放到相应的卸货点即可），假设每个卸货点会随机生成订单，一个订单只有一个商品，但这些订单有优先级别，分为三个优先级别（用户下订单时，会选择优先级别，优先级别高的付费高）：

- 一般：3 小时内配送到即可
- 较紧急：1.5 小时内配送到
- 紧急：0.5 小时内配送到

我们将时间离散化，也就是每隔 t 分钟，所有的卸货点会生成订单（0-m 个订单），同时每隔 t 分钟，系统要做出决策，包括：

1. 哪些配送中心出动多少无人机完成哪些订单；
2. 每个无人机的路径规划，即先完成哪个订单，再完成哪个订单，...，最后返回原来的配送中心；

注意：系统做决策时，可以不对当前的某些订单进行配送，因为当前某些订单可能紧急程度不高，可以累积后和后面的订单一起配送。

目标：一段时间内（如一天），所有无人机的总配送路径最短

约束条件：满足订单的优先级别要求

假设条件：

1. 无人机一次最多只能携带 n 个物品；
2. 无人机一次飞行最远路程为 20 公里（无人机送完货后需要返回配送点）；
3. 无人机的速度为 60 公里/小时；
4. 配送中心的无人机数量无限；
5. 任意一个配送中心都能满足用户的订货需求；

2.2 问题建模

VRPTW 可以建模为一个混合整数规划问题，在给出完整数学模型之前，先引入下面的决策变量：

$$x_{ijk} = \begin{cases} 1, & \text{在最优解中, 弧 } (i, j) \text{ 被无人机 } k \text{ 选中} \\ 0, & \text{其他} \end{cases}$$

s_{ik} = 无人机 k 到达 i 的时间

模型中涉及的其他参数为： t_{ij} 表示无人机在弧 (i, j) 上的飞行时间, M 为一个足够大的正数，在本实验中取 10000000。

综合上面引出的决策变量，给出的 VRPTW 问题的标准模型如下：

$$\min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ijk}$$

$$s.t. \sum_{k \in K} \sum_{j \in V} x_{ijk} = 1, \forall i \in C \quad (2.1)$$

$$\sum_{j \in V} x_{0jk} = 1, \forall k \in K \quad (2.2)$$

$$\sum_{i \in V} x_{ihk} - \sum_{j \in V} x_{hjk} = 0, \forall h \in C, \forall k \in K \quad (2.3)$$

$$\sum_{i \in V} x_{i,n+1,k} = 1, \forall k \in K \quad (2.4)$$

$$\sum_{i \in C} q_i \sum_{j \in V} x_{ijk} \leq Q, \forall k \in K \quad (2.5)$$

$$s_{ik} + t_{ij} - M(1 - x_{ijk}) \leq s_{jk}, \forall (i, j) \in A, \forall k \in K \quad (2.6)$$

$$e_i \leq s_{ik} \leq l_i, \forall i \in V, \forall k \in K \quad (2.7)$$

$$c_{ij} x_{ijk} \leq 20, \forall (i, j) \in A, \forall k \in K \quad (2.8)$$

$$x_{ijk} \in \{0, 1\}, \forall (i, j) \in A, \forall k \in K \quad (2.9)$$

其中，Q 为无人机的容量， q_i 为第 i 个卸货点生成的订单需求量：

- 目标函数是为了最小化所有无人机的总行驶成本（距离）
- 约束 1-4 保证了每架无人机必须从配送中心出发，经过一个卸货点就离开那个卸货点，最终返回原来的配送中心
- 约束 5 为无人机的容量约束
- 约束 6 7 是时间窗约束，保证了无人机到达每个卸货点的时间均在要求时间窗内，即**满足订单的优先级别要求**，点 $n+1$ 是点 0 的一个备份，即**从配货中心出发且返回配货中心**，是为了方便实现。
- 约束 8 是保证无人机一次的飞行路程不超过最远路程 20 公里。

2.3 不考虑订单延后

这种情况下，将每一时刻视为一个静态的路径规划问题，每次做出决策将所有订单均派发给无人机，每次路径规划将会完成所有的订单。因此该问题为带时间窗的多配送中心路径规划问题，可以用以下算法进行求解。

2.3.1 整数规划

使用整数规划（Integer Programming, IP）来求解车辆路径问题是一种有效方法。整数规划求解 VRP 的主要优势在于，它可以通过全面搜索问题的所有可能解空间，从而保证找到全局最优解。

许多优化工具和库可以有效地求解整数规划问题，为 VRP 提供精确的解，例如 Gurobi、CPLEX、PuLP、OR-Tools 等等。

2.3.2 遗传算法

遗传算法 (Genetic Algorithm, GA) 是一种基于自然选择和遗传学原理的搜索算法, 属于进化计算领域。它模拟了生物进化过程中的遗传、突变、杂交 (交叉) 和选择等过程, 以此解决优化和搜索问题。

遗传算法的主要特点和步骤如下:

1. 初始化: 随机生成一组候选解, 这些解构成了初始种群。
2. 评估: 使用适应度函数来评估每个个体 (解) 的性能。
3. 选择: 根据适应度的高低, 选择性能较好的个体, 以便它们更有可能繁衍后代。
4. 交叉: 通过交叉操作, 父母个体的部分信息被结合在一起, 产生新的后代。
5. 变异: 对后代个体进行小的随机变化, 以引入新的遗传信息。
6. 新一代种群: 使用交叉和变异产生的新个体替换掉原始种群中的部分或全部个体, 形成新一代种群。
7. 终止条件: 如果达到一定的代数 (迭代次数) 或者适应度满足要求, 算法结束。

遗传算法的优点包括对搜索空间的广泛适用性、不依赖问题的具体数学模型、容易实现等。但它也存在一些缺点, 如可能早熟收敛、计算量大、可能难以找到全局最优解等。遗传算法广泛应用于函数优化、模式识别、机器学习、工程设计等领域。

2.4 考虑订单延后

最优解: 如果考虑订单延后, 每隔 t 时刻会生成一批新的订单, 那么对于一段时间来说, 则可以通过考虑该时间段内所有的卸货点订单生成情况来计算获得最优解。例如该特定时间段为 $3t$, 则同时考虑 3 份订单地图, 此时的时间窗约束会发生变化, 例如, 0 时刻产生的优先级为 x_0 的订单, 时间窗为 $[0, x_0]$, t 时刻产生的优先级为 x_1 的订单时间窗为 $[t, t + x_1]$ 。

但并不符合实际, 因为它相当于一段时间之后, 得知了每个后续时刻的订单情况, 再去考虑是否将当前时刻的某些订单延后处理, 这是不合理的。

近似解: 因此对于在每个时刻都需要做出决策的现实情况, 我的想法是通过学习大量的真实数据, 计算出一个**距离阈值** α 和一个**时间阈值** β , 如果在当前时刻配送某个订单比不配送所需要多花费的距离大于 α , 且距离该订单时间窗结束的剩余时间大于 β , 则将其延迟配送, 否则在当前时刻配送。

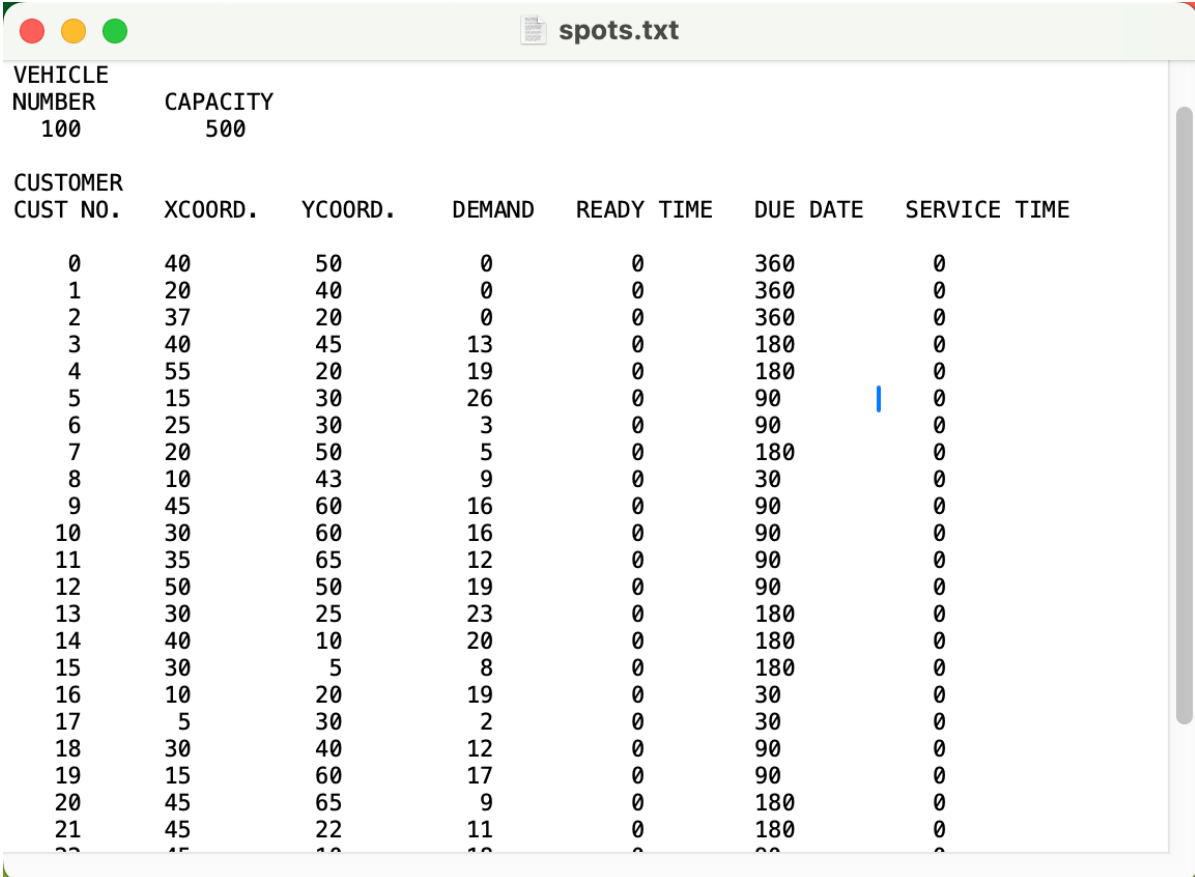
3 整数规划

方法采用两阶段方法进行求解：

1. 将卸货点分配给最近的配送中心；
2. 再对分配好的卸货点和对应的配送中心进行路线优化。

3.1 数据集准备

下面是定义问题实例（假设有 50 个卸货点）的文本文件的格式说明。所有常量均为整数。完整数据见文件夹中 spots（问题实例数据）.txt。



VEHICLE NUMBER	CAPACITY	CUSTOMER CUST NO.	XC00RD.	YC00RD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
100	500	0	40	50	0	0	360	0
		1	20	40	0	0	360	0
		2	37	20	0	0	360	0
		3	40	45	13	0	180	0
		4	55	20	19	0	180	0
		5	15	30	26	0	90	0
		6	25	30	3	0	90	0
		7	20	50	5	0	180	0
		8	10	43	9	0	30	0
		9	45	60	16	0	90	0
		10	30	60	16	0	90	0
		11	35	65	12	0	90	0
		12	50	50	19	0	90	0
		13	30	25	23	0	180	0
		14	40	10	20	0	180	0
		15	30	5	8	0	180	0
		16	10	20	19	0	30	0
		17	5	30	2	0	30	0
		18	30	40	12	0	90	0
		19	15	60	17	0	90	0
		20	45	65	9	0	180	0
		21	45	22	11	0	180	0
		22	45	10	10	0	30	0

- 编号 0、1、2 表示配送中心，所有无人机都必须从这里出发。后续节点皆为卸货点。
- NUMBER 是车辆的最大数量，假设条件中配送中心的无人机数量无限，这里设定为一个足够大的整数 100。
- CAPACITY 为每架无人机的容量，假设条件中无人机一次最多只能携带 n 个物品，这里设定为 200。
- READYTIME 是指可到达指定卸货点的最早时间。
- DUETIME 是指可到达指定卸货点的最晚时间。

- DEMAND 是指卸货点生成的订单数量。

注意, 在本解决方案中, 将 i 到 j 所需的时间 t_{ij} 和成本 c_{ij} 都换算成了 i 到 j 所需的距离来看待。根据题目, 无人机的速度为 60 公里/小时, 所以将优先级别为一般 (3h 内)、较紧急 (1.5h 内) 和紧急 (0.5h 内) 的订单的 DUE TIME 对应地设置为 180、90 和 30。

本作业分别展示了在卸货点数量为 17、21、24 三种不同情况下的路径规划。

3.2 Gurobi

Gurobi 是由美国 Gurobi Optimization 公司开发新一代大规模优化器, 提供 C++, Java, Python, .Net, Matlab 和 R 等多种接口, 支持求解以下模型:

- 连续和混合整数线性问题;
- 凸目标或约束连续和混合整数二次问题;
- 非凸目标或约束连续和混合整数二次问题;
- 含有对数、指数、三角函数、高阶多项式目标或约束, 以及任何形式的
- 分段约束的非线性问题;
- 含有绝对值、最大值、最小值、逻辑与或非目标或约束的非线性问题。

Gurobi 具有强大的求解能力和高效的算法, 可以处理大规模复杂问题。它的求解器使用先进的优化技术, 如线性规划的单纯形法、内点法以及整数规划的分支定界法和割平面法等。Gurobi 还提供了丰富的 API (应用程序编程接口), 可以与各种编程语言 (如 Python、C++、Java 等) 进行集成, 使开发者能够方便地使用它的功能来解决实际问题。

Gurobi 的应用领域广泛, 包括物流和运输优化、生产计划、资源分配、金融风险管理和电力系统调度等。通过使用 Gurobi, 用户可以对复杂的决策问题建立数学模型, 并通过求解器获得最优或接近最优的解决方案, 从而提高效率、降低成本或优化资源利用。

本解决方案使用了 Gurobi+python 实现。

3.3 重要代码

- 将卸货点分配给最近的配送中心

```

1 data = readData(data_path, customerNum)
2 r0.append(0)
3 r1.append(1)
4 r2.append(2)
5 for i in range(customerNum):
6     if data.distanceMatrix[0][3+i]<=data.distanceMatrix[1][3+i] and data.
       distanceMatrix[0][3+i]<=data.distanceMatrix[2][3+i]:
7         r0.append(3+i)
8     elif data.distanceMatrix[1][3+i]<=data.distanceMatrix[0][3+i] and data.
       distanceMatrix[1][3+i]<=data.distanceMatrix[2][3+i]:
9         r1.append(3+i)
10    elif data.distanceMatrix[2][3+i]<=data.distanceMatrix[0][3+i] and data.
       distanceMatrix[2][3+i]<=data.distanceMatrix[1][3+i]:
11        r2.append(3+i)
12    r0.append(data.nodeNum-3)

```



```

13 r1.append(data.nodeNum-2)
14 r2.append(data.nodeNum-1)

```

readDsata 将 txt 中的数据读到数组中，并计算出距离矩阵。遍历卸货点将其分配给距离最近的配货中心。在最后添加上配货中心的备份，为了方便实现从配货中心出发且返回配货中心。

这样便将多中心的 VRPTW 问题拆分成了三个单配送中心的 VRPTW 问题。

- 再对分配好的卸货点和对应的配送中心分别进行路线规划。

代码主要逻辑见注释：

```

1  # 声明模型
2  model = Model("VRPTWO")
3  # 关闭输出
4  model.setParam('OutputFlag', 0)
5  # 定义变量
6  X = [[[[] for _ in range(data.vehicleNum)] for _ in range(data.nodeNum)] for
       _ in range(data.nodeNum)]
7  S = [[[[] for _ in range(data.vehicleNum)] for _ in range(data.nodeNum)]
8  for i in r0:
9      for k in range(data.vehicleNum):
10         S[i][k] = model.addVar(data.readyTime[i], data.dueTime[i], vtype=GRB.
            CONTINUOUS, name=f'S_{i}_{k}')
11         for j in r0:
12             X[i][j][k] = model.addVar(vtype=GRB.BINARY, name=f'X_{i}_{j}_{k}')
13  # 目标函数
14  obj = LinExpr(0)
15  for i in r0:
16      for j in r0:
17          if i != j:
18              for k in range(data.vehicleNum):
19                  obj.addTerms(data.distanceMatrix[i][j], X[i][j][k])
20  model.setObjective(obj, GRB.MINIMIZE)
21  # 约束1: 车辆只能从一个点到另一个点
22  for i in r0[1:-1]:
23      expr = LinExpr(0)
24      for j in r0:
25          if i != j:
26              for k in range(data.vehicleNum):
27                  if i != 0 and i != len(r0) - 1:
28                      expr.addTerms(1, X[i][j][k])
29      model.addConstr(expr == 1)
30  # 约束2: 车辆必须从仓库出发
31  for k in range(data.vehicleNum):
32      expr = LinExpr(0)
33      for j in r0[1:]:
34          expr.addTerms(1, X[0][j][k])
35      model.addConstr(expr == 1)
36  # 约束3: 车辆经过一个点就必须离开一个点
37  for k in range(data.vehicleNum):
38      for h in r0[1:-1]:
39          expr1 = LinExpr(0)
40          expr2 = LinExpr(0)
41          for i in r0:
42              if h != i:

```

```

43         expr1.addTerms(1, X[i][h][k])
44     for j in r0:
45         if h != j:
46             expr2.addTerms(1, X[h][j][k])
47     model.addConstr(expr1 == expr2)
48 # 约束4: 车辆最终返回仓库
49 for k in range(data.vehicleNum):
50     expr = LinExpr(0)
51     for i in r0[:-1]:
52         expr.addTerms(1, X[i][r0[-1]][k])
53     model.addConstr(expr == 1)
54 # 约束5: 车辆容量约束
55 for k in range(data.vehicleNum):
56     expr = LinExpr(0)
57     for i in r0[1:-1]:
58         for j in r0:
59             if i != 0 and i != data.nodeNum - 1 and i != j:
60                 expr.addTerms(data.demand[i], X[i][j][k])
61     model.addConstr(expr <= data.capacity)
62 # 约束6: 时间窗约束
63 for k in range(data.vehicleNum):
64     for i in r0:
65         for j in r0:
66             if i != j:
67                 model.addConstr(S[i][k] + data.distanceMatrix[i][j] - S[j][k] <=
                    M - M * X[i][j][k])
68 # 约束7: 无人机一次最多飞行20公里
69 for k in range(data.vehicleNum):
70     for i in r0:
71         for j in r0:
72             if data.distanceMatrix[i][j] < 20:
73                 # 只有当弧ij的长度小于20时, 才需要添加约束
74                 model.addConstr(X[i][j][k] <= 1)
75             else:
76                 # 如果弧ij的长度大于或等于20, 车辆k不能选择这条弧
77                 model.addConstr(X[i][j][k] == 0)
78
79 # 记录求解开始时间
80 start_time = time.time()
81 # 求解
82 model.optimize()
83 if model.status == GRB.OPTIMAL:
84     print("-" * 20, "配送中心OSolvedSuccessfully", '-' * 20)
85     # 输出求解以及总用时
86     print(f"Solve_Time:{time.time()-start_time}s")
87     print(f"Total_Travel_Distance:{model.ObjVal}")

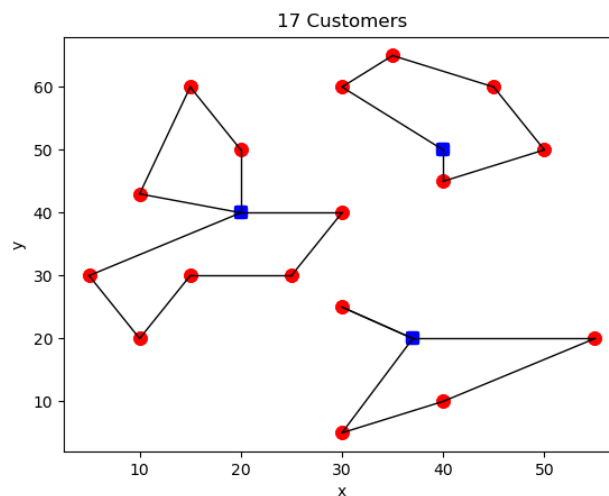
```

3.4 结果展示

卸货点设置为 17 时，各配送中心对应的路径、行驶距离以及各无人机携带订单的数量如下图：

```
/Users/simonlion/anaconda/anaconda3/envs/test1/bin/python /Users/simonlion/Pyd
配送中心0:[0, 3, 9, 10, 11, 12, 20]
配送中心1:[1, 5, 6, 7, 8, 16, 17, 18, 19, 21]
配送中心2:[2, 4, 13, 14, 15, 22]
----- Problem Information -----
Node Num: 23
Customer Num: 17
Vehicle Num: 100
Vehicle Capacity: 500.0
Set parameter Username
Academic license - for non-commercial use only - expires 2025-06-05
----- 配送中心0 Solved Successfully -----
Solve Time: 0.6557540893554688 s
Total Travel Distance: 59.75422309809326
[[0, 10, 11, 9, 12, 3, 0]]
Route-1 : [0, 10, 11, 9, 12, 3, 0] , distance: 59.75422309809326 , load: 76.0
----- 配送中心1 Solved Successfully -----
Solve Time: 1.9123859405517578 s
Total Travel Distance: 120.90946758289563
[[1, 8, 19, 7, 1], [1, 17, 16, 5, 6, 18, 1]]
Route-1 : [1, 8, 19, 7, 1] , distance: 49.34069154307885 , load: 31.0
Route-2 : [1, 17, 16, 5, 6, 18, 1] , distance: 71.5687760398168 , load: 62.0
----- 配送中心2 Solved Successfully -----
Solve Time: 0.12818193435668945 s
Total Travel Distance: 80.965692156151
[[2, 4, 14, 15, 2], [2, 13, 2]]
Route-1 : [2, 4, 14, 15, 2] , distance: 63.76104162206574 , load: 47.0
Route-2 : [2, 13, 2] , distance: 17.204650534085253 , load: 23.0
```

可视化后如图：



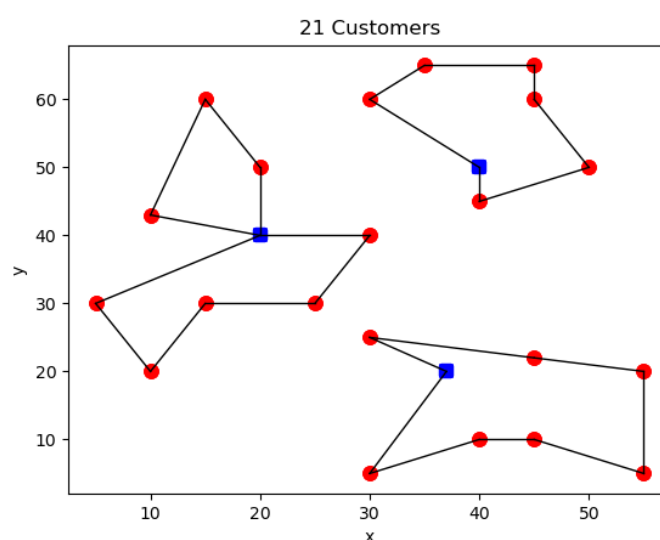
卸货点设置为 21 时，各配送中心对应的路径、行驶距离以及各无人机携带订单的数量如下图：

```

/Users/simonlion/anaconda/anaconda3/envs/test1/bin/python /Users/simonlion/PycharmProj
配送中心0:[0, 3, 9, 10, 11, 12, 20, 24]
配送中心1:[1, 5, 6, 7, 8, 16, 17, 18, 19, 25]
配送中心2:[2, 4, 13, 14, 15, 21, 22, 23, 26]
----- Problem Information -----
Node Num: 27
Customer Num: 21
Vehicle Num: 100
Vehicle Capacity: 500.0
Set parameter Username
Academic license - for non-commercial use only - expires 2025-06-05
----- 配送中心0 Solved Successfully -----
Solve Time: 0.8625321388244629 s
Total Travel Distance: 63.57388321059432
[[0, 3, 12, 9, 20, 11, 10, 0]]
Route-1 : [0, 3, 12, 9, 20, 11, 10, 0] , distance: 63.573883210594325 , load: 85.0
----- 配送中心1 Solved Successfully -----
Solve Time: 1.8958089351654053 s
Total Travel Distance: 120.90946758289563
[[1, 8, 19, 7, 1], [1, 17, 16, 5, 6, 18, 1]]
Route-1 : [1, 8, 19, 7, 1] , distance: 49.34069154307885 , load: 31.0
Route-2 : [1, 17, 16, 5, 6, 18, 1] , distance: 71.5687760398168 , load: 62.0
----- 配送中心2 Solved Successfully -----
Solve Time: 0.7926630973815918 s
Total Travel Distance: 93.01104796725129
[[2, 15, 14, 22, 23, 4, 21, 13, 2]]
Route-1 : [2, 15, 14, 22, 23, 4, 21, 13, 2] , distance: 93.0110479672513 , load: 128.0

```

可视化后如图：



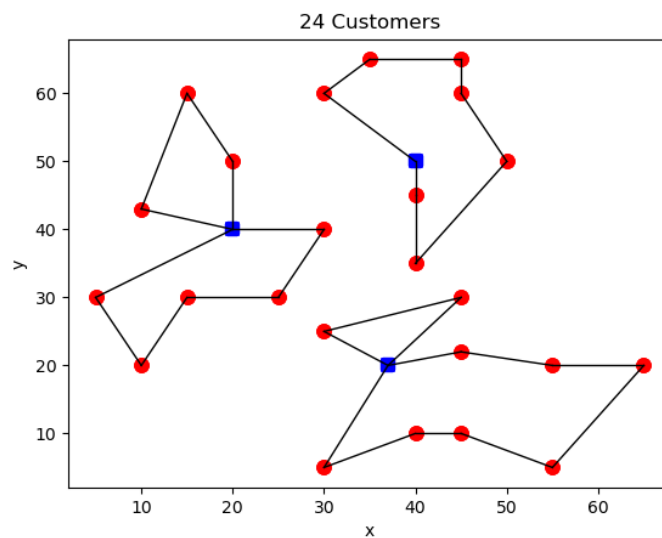
卸货点设置为 24 时，各配送中心对应的路径、行驶距离以及各无人机携带订单的数量如下图：

```

配送中心0:[0, 3, 9, 10, 11, 12, 20, 24, 27]
配送中心1:[1, 5, 6, 7, 8, 16, 17, 18, 19, 28]
配送中心2:[2, 4, 13, 14, 15, 21, 22, 23, 25, 26, 29]
----- Problem Information -----
Node Num: 30
Customer Num: 24
Vehicle Num: 100
Vehicle Capacity: 500.0
Set parameter Username
Academic license - for non-commercial use only - expires 2025-06-05
----- 配送中心0 Solved Successfully -----
Solve Time: 2.2171337604522705 s
Total Travel Distance: 80.42129970041532
[[0, 10, 11, 20, 9, 12, 24, 3, 0]]
Route-1 : [0, 10, 11, 20, 9, 12, 24, 3, 0] , distance: 80.42129970041532 , load: 88.0
----- 配送中心1 Solved Successfully -----
Solve Time: 1.8846542835235596 s
Total Travel Distance: 120.90946758289563
[[1, 8, 19, 7, 1], [1, 17, 16, 5, 6, 18, 1]]
Route-1 : [1, 8, 19, 7, 1] , distance: 49.34069154307885 , load: 31.0
Route-2 : [1, 17, 16, 5, 6, 18, 1] , distance: 71.5687760398168 , load: 62.0
----- 配送中心2 Solved Successfully -----
Solve Time: 1.4806199073791504 s
Total Travel Distance: 127.6055938307358
[[2, 13, 26, 2], [2, 21, 4, 25, 23, 22, 14, 15, 2]]
Route-1 : [2, 13, 26, 2] , distance: 37.21996204275022 , load: 40.0
Route-2 : [2, 21, 4, 25, 23, 22, 14, 15, 2] , distance: 90.38563178798557 , load: 111.0

```

可视化后如图：



4 遗传算法

对问题进行 python 实现如下：

1. 数据结构设计：设计了三个主要的类：DeliveryCenter, DropOffPoint, 和 Order，用于表示配送中心、卸货点和订单。
2. 订单生成：在每个时间间隔，根据设置的概率随机生成不同优先级的订单。
3. 配送中心分配：根据配送中心与卸货点之间的距离分配卸货点，以确保每个卸货点都被分配到最近的配送中心。
4. 遗传算法路径优化：使用遗传算法来求解每个配送中心的最优配送路径，目标是最小化路程且满足负载和距离限制。
5. 动态可视化：通过 matplotlib 库实现动态可视化，展示无人机从配送中心到卸货点的配送过程。

4.1 数据集准备

本解决方案中的配送中心和卸货点位置信息由代码随机生成。

```
1 # 生成随机位置
2 def generatePosition(count, size1, size2):
3     return [(random.randint(size1, size2), random.randint(size1, size2)) for _ in
4             range(count)]
5 # 位置距离计算
6 def distance(pos1, pos2):
7     return ((pos1[0] - pos2[0]) ** 2 + (pos1[1] - pos2[1]) ** 2) ** 0.5
8 # 生成的位置
9 centers = generatePosition(centersNum, 3, 7)
10 customers = generatePosition(customerNum, 0, 10)
```

此外，每个卸货点生成订单的具体信息（包括订单的优先级，截止时间，产生时间等基本信息）由订单生成函数随机生成。

```
1 # 订单类
2 class Order:
3     def __init__(self, priority, drop_point_id, time_generated):
4         self.priority = priority # 1: 一般, 2: 较紧急, 3: 紧急
5         self.drop_point_id = drop_point_id
6         self.time_generated = time_generated
7         self.deadline = self.set_deadline(time_generated, priority)
8         self.processed = False
9     def set_deadline(self, time_generated, priority):
10         if priority == 1:
11             return time_generated + 180 # 一般订单, 3小时内送达
12         elif priority == 2:
13             return time_generated + 90 # 较紧急订单, 1.5小时内送达
14         else:
15             return time_generated + 30 # 紧急订单, 0.5小时内送达
16 # 生成订单
```

```

17 def generateOrders(current_time):
18     num_orders = random.randint(0, ordersMax)
19     return [Order(random.randint(1, 3), random.randint(0, customerNum - 1),
20                 current_time) for _
                in range(num_orders)]

```

4.2 重要代码

使用 `deap` 库，定义问题对应的遗传算法实例，然后进行路径规划

```

1 def createRoute(individual, center, customers):
2     # 根据遗传算法个体生成路径
3     route = [center] + [customers[i] for i in individual] + [center]
4     return route
5 def eval_route(individual, center, customers):
6     # 计算路径的总距离
7     route = createRoute(individual, center, customers)
8     total_distance = sum(distance(route[i], route[i + 1]) for i in range(len(
9         route) - 1))
10    return (total_distance,)
11 def setupGA(center, drop_point_ids, customers):
12     creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
13     creator.create("Individual", list, fitness=creator.FitnessMin)
14     # 创建一个名为 FitnessMin 的适应度类，权重为-1.0，表示我们希望最小化适应度函数
15     toolbox = base.Toolbox()
16     # 确保随机抽样的元素数量不超过列表大小
17     sample_size = min(len(drop_point_ids), len(drop_point_ids))
18     toolbox.register("indices", random.sample, range(len(drop_point_ids)),
19                     sample_size)
20     toolbox.register("individual", tools.initIterate, creator.Individual, toolbox
21                     .indices)
22     toolbox.register("population", tools.initRepeat, list, toolbox.individual)
23     toolbox.register("evaluate", eval_route, center=center, customers=customers)
24     toolbox.register("mate", tools.cxOrdered)
25     toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.05)
26     toolbox.register("select", tools.selTournament, tournsize=3)
27     # 函数最后返回 toolbox，它包含了运行遗传算法所需的所有组件
28     return toolbox
29 def optimizeRoutes(center, orders, customers):
30     drop_point_ids = [order.drop_point_id for order in orders if not order.
31                     processed]
32     if len(drop_point_ids) < 2: # 如果小于2个订单，则无法进行有效的路径规划
33         return [center] + [customers[idx] for idx in drop_point_ids] + [center]
34     toolbox = setupGA(center, drop_point_ids, customers)
35     pop = toolbox.population(n=POP_SIZE)
36     hof = tools.HallOfFame(1, similar=np.array_equal)
37     # 创建一个“名人堂” (Hall of Fame)，用来存储最好的个体。similar参数用于比较个体
38     # 是否相同
39     stats = tools.Statistics(lambda ind: ind.fitness.values)
40     stats.register("min", np.min)
41     stats.register("avg", np.mean)
42     algorithms.eaSimple(pop, toolbox, cxpb=CX_PB, mutpb=MUT_PB, ngen=GEN, stats=
43         stats, halloffame=hof, verbose=False)
44     best_route = createRoute(hof[0], center, customers)

```

```

39  #遍历名人堂中最佳个体的索引，将对应订单的processed属性设置为 True，表示这些订单
    已经被包含在最佳路线中处理了
40  for idx in hof[0]:
41  for idx in hof[0]:
42      orders[idx].processed = True
43  return best_route

```

4.3 结果展示

我们以 1 个配送中心，5 个卸货点，单位时间最多生成 5 个订单，每 60 分钟生成一次订单，运行 240 分钟为例，绘制实验结果图：



图 4.1 (1)

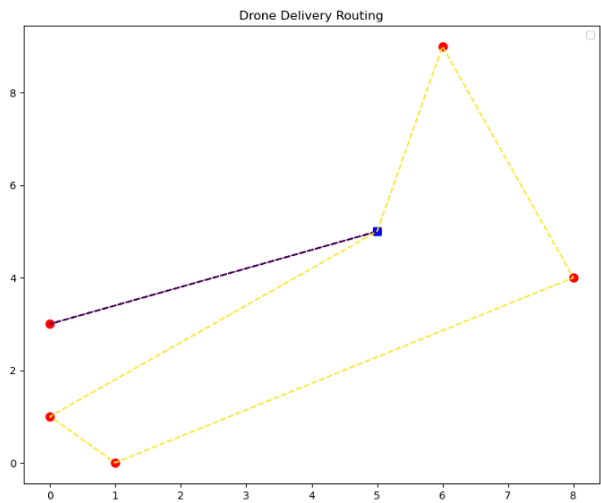


图 4.2 (2)

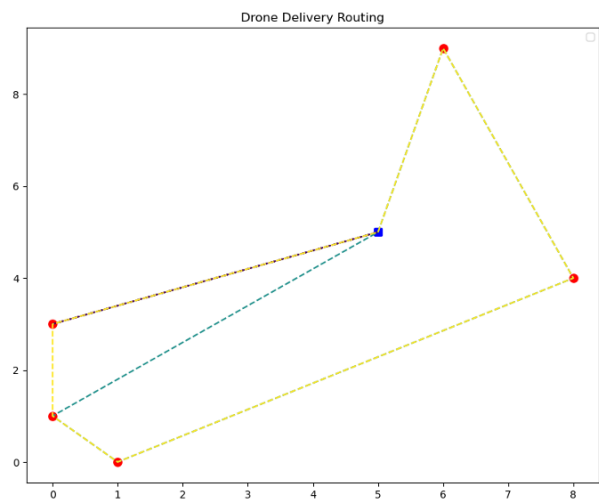


图 4.3 (3)

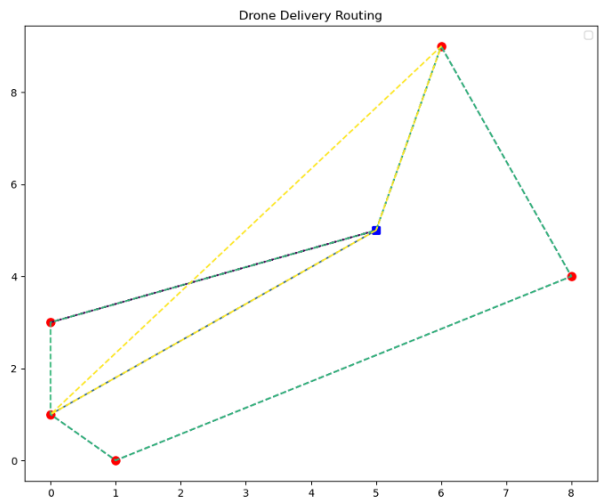


图 4.4 (4)

我们以 4 个配送中心，10 个卸货点，单位时间最多生成 5 个订单，每 60 分钟生成一次订单，运行 300 分钟为例，绘制实验结果图 (顺序为从左往右)：

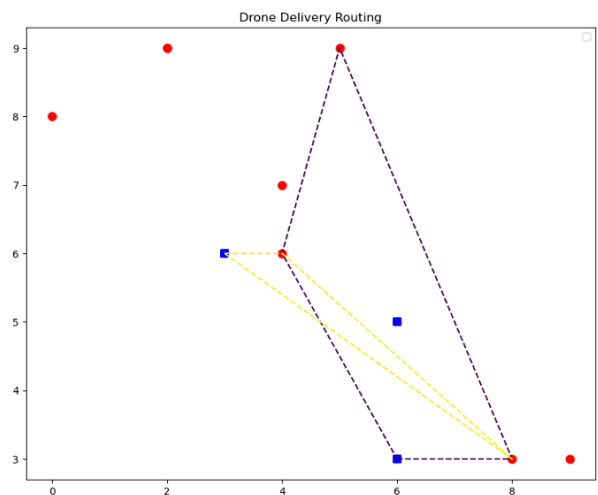


图 4.5 (1)

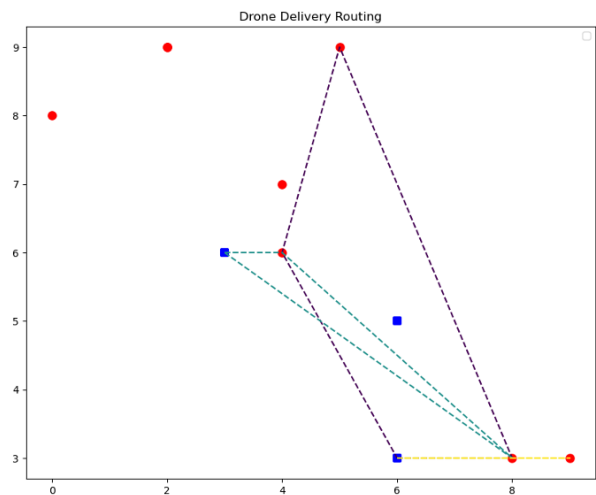


图 4.6 (2)

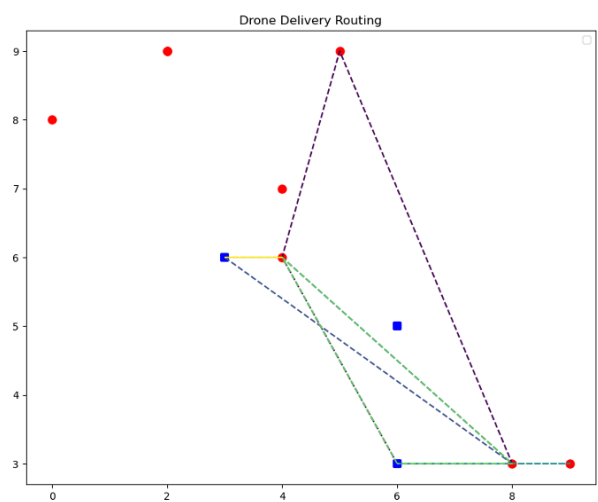


图 4.7 (3)

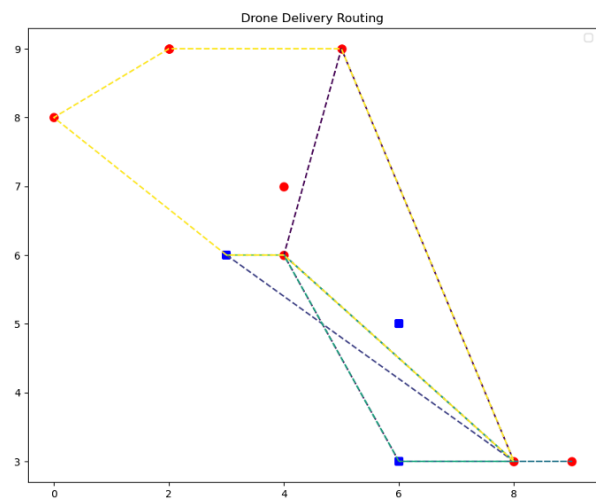


图 4.8 (4)

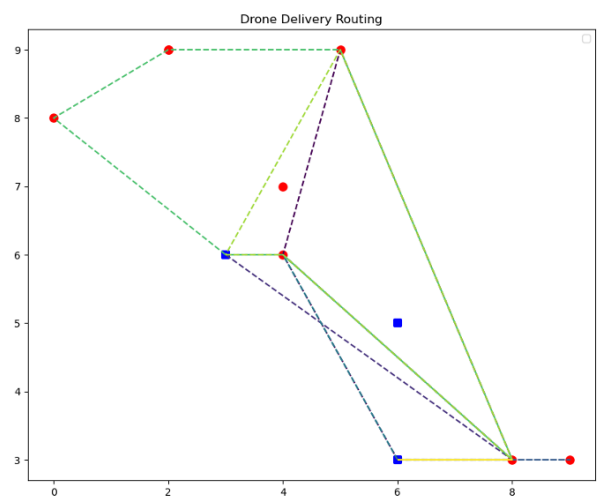


图 4.9 (5)

5 总结

解决 VRPTW 问题比较常见地可以利用蚁群算法、遗传算法等启发式算法和线性规划法，它们在原理、实现和性能上存在一些区别：

- 蚁群算法 (Ant Colony Optimization, ACO) 通常能够找到较好的解但不保证最优，但可能需要较长的计算时间。它适合于解决大规模的 VRPTW 问题，但解的质量和计算效率受算法参数和信息素更新策略的影响较大。
- 遗传算法 (Genetic Algorithm, GA) 通常能够找到全局最优解或近似最优解，但同样可能需要较长的计算时间。它适用于解决复杂的 VRPTW 问题，但解的质量和计算效率也受算法参数的影响。
- 线性规划法 (Linear Programming, LP) 是一种数学方法，用于在一组线性不等式约束下求解线性目标函数的最大值或最小值问题。在 VRPTW 中，可以将问题转化为线性规划模型，通过求解线性规划问题来找到最优解。线性规划法能够保证在一定条件下找到最优解，但适用性受限于问题的规模和复杂性。对于大规模的 VRPTW 问题，线性规划可能因为计算复杂度过高而难以取得好的效果，这时候启发式算法就要更加适合。

总结来说，蚁群算法和遗传算法属于启发式算法，它们通过模拟自然现象来寻找问题的近似解，适合于解决大规模和复杂的问题。线性规划法则是一种精确算法，能够保证在一定条件下找到最优解，但其适用性受限于问题的规模和复杂性。在实际应用中，可以根据问题的具体情况和需求选择合适的算法。