# 无人机配送路径规划问题

国家网络安全学院 2023202210030 张正涵

**原始问题陈述：**

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，实现如下图所示区域的无人机配送的路径规划。在此区域中，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，系统要做成决策，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，即先完成那个订单，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；

大作业的图和数据（订单生成）可以自己生成，在第一章描述清楚。

## 第一章 引言

### 1.1 问题分析

配送中心距卸货点距离≤10公里，否则无法回到配送点，所以配送时间肯定≤10分钟。

卸货点都有一个最近的配送中心，但是可能是同一配送中心。

复杂的情况是，配送中心经过多个卸货点回到配送点仍然满足20公里的条件，这种条件就可以大大减少配送路径，因为三角形两边的两倍＞三角形周长。

假设所有的配送中心只会和自己最近卸货点构成上述的关系，而且只要存在多个卸货点最近的情况，就必然存在回路满足。

同时要满足该情况，就必须配送中心发现最近的多个卸货点同时有订单，所以尽可能踩着订货需求的截至时间发货。

对于题目中出现的时间窗问题，配送中心采取的较好的策略就是也采用一个时间窗进行无人机指令的调度。为了达到无人机配送路径最短，就是要在每一次调度分配无人机的时候，为每一个无人机都分配好满足条件的最短配送路径。同时也要规划好任务的优先级。

### 2.1 数据生成和假设条件

由任务需求知：大作业的图和数据（订单生成）可以自己生成，在第一章描述清楚。所以这里以10\*10的网格图为区域图所示，其中生成了两个配送中心，四个卸货点，其具体分布如图1所示。其中红色方块代表配送中心，其坐标为[(0, 0), (10, 10)]；另外的蓝色圆点代表卸货点，它们的坐标为[(3, 4), (5, 6), (7, 8), (9, 10)]。

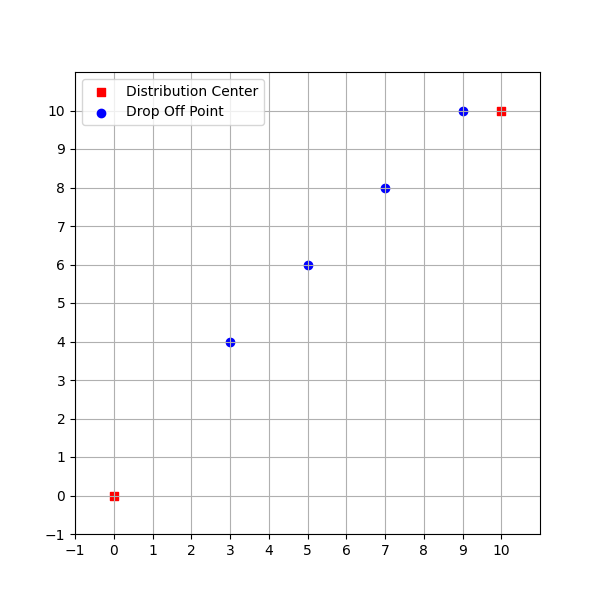


图 1 数据生成可视网格图

生成了问题中配送中心和卸货点的图结构后，目前还缺少订单的具体生成数据以及在1.1节提到的配送中心应采取的调度时间间隔。因此，这里假定订单的生成间隔时间是10分钟，在间隔时间内不会产生新的订单，并且每个卸货点每次只会产生0或1的订单数量，如果产生了1笔订单，那么会在三个优先级中随机生成一种优先级的订单，所以对于所有的m个卸货点，每个时间间隔生成0-m个订单。

其中为了问题计算方便，设计图结构（即节点和节点之间的路径与距离也是必要的），这里将距离全视为整数，并以类似邻接矩阵的形式展示如下：

|  |
| --- |
| graph = {      (0, 0): {(3, 4): 5, (5, 6): 8, (7, 8): 10, (9, 10): 13},      (3, 4): {(0, 0): 5, (5, 6): 3, (7, 8): 6, (9, 10): 9},      (5, 6): {(0, 0): 8, (3, 4): 3, (7, 8): 3, (9, 10): 6},      (7, 8): {(0, 0): 10, (3, 4): 6, (5, 6): 3, (9, 10): 3},      (9, 10): {(0, 0): 13, (3, 4): 9, (5, 6): 6, (7, 8): 3, (10, 10): 1},      (10, 10): {(9, 10): 1}  } |

由于图上展现不太美观，就以数据的形式展现出来。

## 第二章 问题建模

由1.1节分析可知，该问题实际上是一个较为复杂的无人机路径配送规划问题，和对当前状况分析求解最合适的最短路径异曲同工。原问题涉及到t分钟的时间窗，以及无人机在路径规划时需要考虑到不同优先级订单对实际策略的影响，对应的，应当选择基于优先级的调度策略，并在路径规划问题里选择启发式算法中的A\*算法作为求解最短路径的算法。由于时间窗因素的存在，这里还将引入时间窗来处理订单的优先级问题。

### 2.1 求解策略

根据题目所给指示，一共有两类约束：优先级约束和时间窗约束，所以需要对这两类约束分别给出具体的求解策略，每种策略具体如下。

#### 2.1.1 优先级调度策略

* 配送中心每隔t分钟进行调度
* 高优先级订单优先处理，如果有多个高优先级订单，优先处理最近的订单
* 每架无人机一次最多携带n个物品，优先完成高优先级订单

#### 2.1.2 时间窗策略

* 把每个订单按照优先级分类，计算每个订单的最迟送达时间
* 使用时间窗约束计算哪些订单需要立即处理，哪些订单可以积累后再处理

#### 2.1.3 最短路径策略

* 选择启发式算法A\*算法，用于寻找起点到终点的最短路径

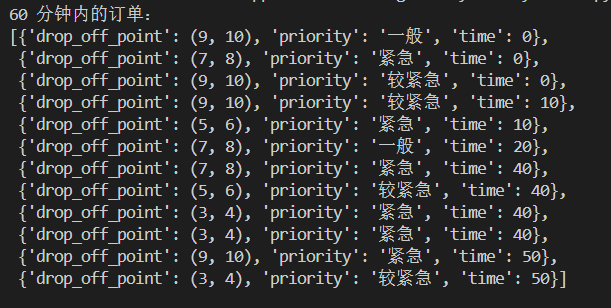
### 2.2 代码实现

首先生成1个小时内的随机订单数据，如下是生成订单数据对应的代码以及订单结果：

|  |
| --- |
| def random\_orders():      for t in range(num\_intervals):          num\_order = random.randint(0, len(drop\_off\_points))          for \_ in range(num\_order):              drop\_off\_point = random.choice(drop\_off\_points)              priority = random.choice(["一般", "较紧急", "紧急"])              orders.append({"time": t \* time\_interval, "drop\_off\_point": drop\_off\_point, "priority": priority})      print(num\_intervals\*time\_interval, "分钟内的订单：")      pprint(orders)  random.seed(0)  random\_orders() |

上述代码的原理最核心的内容：确定时间间隔和间隔次数，在第一章约束的条件下可知每一次间隔生成的订单数量只可能在0-m也就是0-4之间，所以有关订单的其他数据之间使用随机方法生成即可。

固定随机种子0，可以看到此时的订单数据在60分钟内的具体情况：



实现A\*算法寻找最短路径，该算法的启发式代价就用当前点到目标点的曼哈顿距离：

|  |
| --- |
| from queue import PriorityQueue  def heuristic(a, b):  return abs(a[0] - b[0]) + abs(a[1] - b[1]) |

A\*算法的作用是：给定已有的图结构和节点间的距离，对选定的起始点到选定的终点搜寻网络中的最短路径，同时还要记录该路径上无人机到达每个新的卸货点或回到配送中心的到达时间，输出在最短路径中每个节点的辅助信息中。

|  |
| --- |
| def a\_star\_search\_with\_time(start, goal, graph, start\_time, speed):  frontier = PriorityQueue()  frontier.put((0, start))  came\_from = {}  cost\_so\_far = {}  came\_from[start] = (None, start\_time) # 记录前驱节点和到达时间  cost\_so\_far[start] = 0  while not frontier.empty():  current\_priority, current = frontier.get()  current\_time = came\_from[current][1] # 当前节点的到达时间  if current == goal:  break  for next in graph[current]:  travel\_time = graph[current][next] / speed # 计算飞行时间  new\_cost = cost\_so\_far[current] + graph[current][next]  next\_time = current\_time + travel\_time # 计算到达下一个节点的时间  if next not in cost\_so\_far or new\_cost < cost\_so\_far[next]:  cost\_so\_far[next] = new\_cost  priority = new\_cost + heuristic(goal, next)  frontier.put((priority, next))  came\_from[next] = (current, next\_time)  return came\_from, cost\_so\_far |

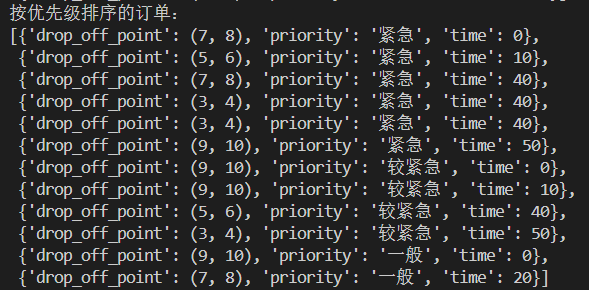
该算法的输入是起点坐标，终点坐标，图结构，起点坐标调度的时间，无人机每分钟的速度；算法的输出是起点到重点的路径，路径的每个点记录了对应的坐标和对应的到达时间，还有每个路径点对应的飞行代价。

有了包含最短路径和路径点到达时间的列表，我们可以设计函数对该路径进行打印，设计函数如下：

|  |
| --- |
| # 重建路径，并包含时间信息  def reconstruct\_path\_with\_time(came\_from, start, goal):  current = goal  path = []  while current != start:  path.append((current, came\_from[current][1]))  current = came\_from[current][0]  path.append((start, came\_from[start][1]))  path.reverse()  return path |

下一步是t分钟执行的调度策略和路径规划算法，最简单就是根据优先级调单，并使用上述的A\*算法进行路径规划，记录每个路径信息的时间信息。由此可知，对于t\*num\_intervals=10\*6=60分钟生成的所有随机订单需要根据优先级优先重排，“紧急”优先级第1，“较紧急”优先级第2，“一般”优先级第3，订单产生时间作为排序的第二优先级，最早的订单比晚发生的订单优先级要高。对应的排序代码很简单，如下所示：

|  |
| --- |
| # 按优先级排序订单  priority\_order = {"紧急": 1, "较紧急": 2, "一般": 3}  orders.sort(key=lambda x: (priority\_order[x["priority"]], x["time"]))  print("按优先级排序的订单：")  pprint(orders) |



虽然我们考虑了t\*num\_intervals=10\*6=60分钟的所有随机订单。但是，我们按照题目要求给出对应的不同时间段的num\_intervals=6次的调度策略。对于这种情况处理起来也非常简单，因为调度策略的具体时间已知，对每次的调度策略发生的时间进行遍历，寻找优先级重排后的订单列表，筛选出订单发生小于等于当前调度策略时间的所有订单，并且对总订单进行更新，只保留还未开始的订单。

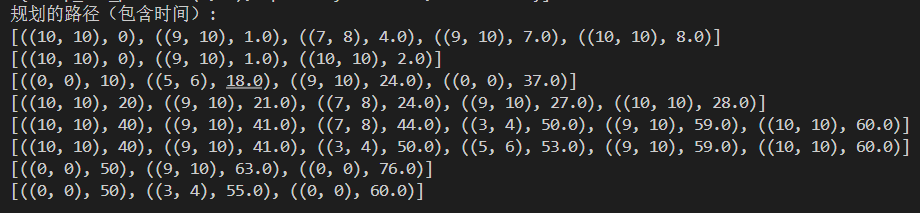
已知网结构，当前的订单列表，就可以寻找合适的配送路径。具体的方法为：随机选取配送中心，依次一辆一辆增加无人机的数量，直到无人机达到最大负载或者订单为空就确定下这个无人机需要运载的货物以及规划路线。代码如下：

|  |
| --- |
| # 送达时间限制  priority\_time\_limit = {"紧急": 30, "较紧急": 90, "一般": 180}  for current\_time in range(0, time\_interval \* num\_intervals, time\_interval):  current\_orders = [order for order in orders if order["time"] <= current\_time]  orders = [order for order in orders if order not in current\_orders]  while current\_orders:  current\_load = 0  current\_center = random.choice(distribution\_centers) # 随机挑选配送中心  route = [(current\_center, current\_time)]  total\_distance = 0  while current\_load < max\_load and current\_orders:  order = current\_orders.pop(0)  start = route[-1][0]  start\_time = route[-1][1]  goal = order["drop\_off\_point"]  delivery\_deadline = order["time"] + priority\_time\_limit[order["priority"]]  came\_from, cost\_so\_far = a\_star\_search\_with\_time(start, goal, graph, start\_time, speed)  if total\_distance + cost\_so\_far[goal] <= max\_distance and came\_from[goal][1] <= delivery\_deadline:  path = reconstruct\_path\_with\_time(came\_from, start, goal)  route.extend(path[1:]) # 加入路径，去掉起点  total\_distance += cost\_so\_far[goal]  current\_load += 1  else:  current\_orders.insert(0, order)  # 放回订单队列  break  if(len(route)>1):  if route[-1][0] != current\_center:  came\_from, cost\_so\_far = a\_star\_search\_with\_time(route[-1][0], current\_center, graph, route[-1][1], speed)  path\_back = reconstruct\_path\_with\_time(came\_from, route[-1][0], current\_center)  route.extend(path\_back[1:]) # 加入返回路径，去掉起点  routes.append(route)    return routes |

代码的逻辑就是优先遍历订单，对每个订单随机选取配送地点，根据配送点和订单的卸货点之间的关系，计算最大载重下的最优路线，同时考虑最晚送达时间因素。如果发现不满足条件就退出最短路径寻找，把已经找到的最短路径弹出记录到策略当中。

|  |
| --- |
| # 调度和路径规划  max\_load = 2 # 无人机一次最多携带2个物品  max\_distance = 20 # 无人机一次飞行最远路程为20公里  speed = 60 / 60 # 速度，单位公里/分钟  routes = schedule\_and\_plan\_with\_time(orders, distribution\_centers, graph, max\_load, max\_distance, speed)  print("规划的路径（包含时间）:", routes) |

用以上设计的算法，输入已经规定好的参数，输出对应的结果为：



每一行代表配送中心发出的一架无人机的飞行策略，其中每个子元素的第一个元素是坐标，第二个元素是到达时间，每行中第一个和最后一个元素代表配送中心，中间都代表着卸货点，其中不同卸货点的个数代表了此次无人机承载的货物的个数。从图上也能看出，无人机优秀的完成了任务并回到了出发点。

## 第三章 问题规约

从第二章我们解决了t分钟k个卸货点生成一次不同等级和数量的订单过程，给出t\*num\_intervals分钟内j个配送中心在t个对应的时间该做出如何的无人机调度策略，同时也给出了每个无人机该执行的具体飞行路径。原理类似，在建模问题中，以上的t值、m值和经历的时间间隔的个数num\_intervals，已经网络结构和节点数量都可以合理范围的更改，解决问题的套路是不变的，遵循以上的方式就能给出每个t时刻配送中心该执行何种策略。即使时间扩大到一天，解决问题的思路也如前者所述：

1、生成随机订单数据。

2、使用A\*算法计算最短路径。

3、根据优先级和时间窗调度无人机。

4、规划无人机的最短配送路径。

综上所述，通过合理调整t值、m值以及时间间隔的个数num\_intervals，并结合网络结构和节点数量的变化，可以在不同的场景下灵活应用上述方法，制定出每个时刻配送中心的无人机调度策略。这种方法不仅具有普适性，还能在实际应用中确保高效的配送和资源优化。只要遵循这一思路，我们就能在各种复杂的配送任务中找到最优解，提高无人机配送系统的整体效率，优化无人机配送路径的最小值。