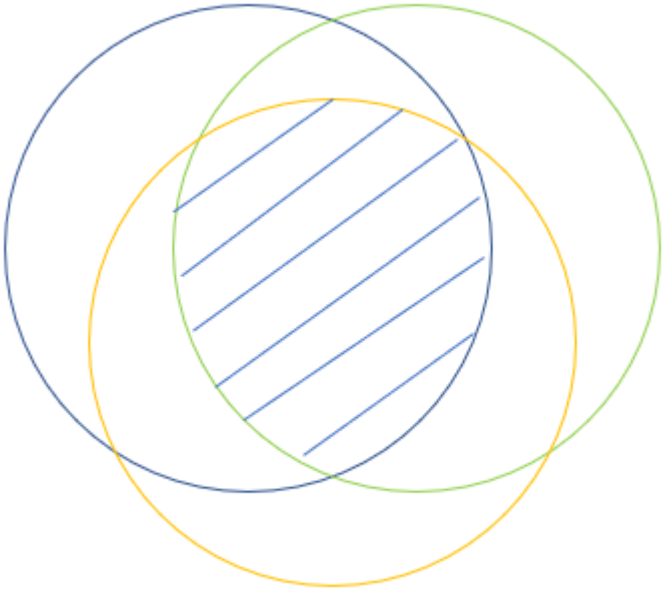
无人机配送路径规划问题

一、问题描述

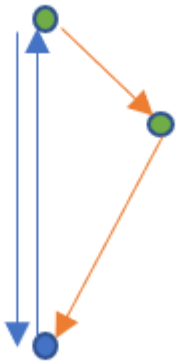
- 根据作业要求，对其中未知量进行合理假设，对无人机配送路径规划问题描述如下：
- 在一个区域中，有$j=3$个无人机配送中心，任意一个配送中心有用户所需要的商品，其数量无限，同时任一配送中心的无人机数量无限。该区域同时有$k=10$个卸货点。每个卸货点会随机生成订单，一个订单只有一个商品，这些订单有优先级别，分为三个优先级别：
    - 一般：180分钟内配送到
    - 较紧急：90分钟内配送到
    - 紧急：30分钟内配送到
- 每隔$t=20$分钟，所有的卸货点会随机生成$0-m(m=3)$个订单，同时每隔t分钟，系统要做成决策，包括：
    - 哪些配送中心出动多少无人机完成哪些订单
    - 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
    - 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，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
- 其他假设条件
    1. 无人机一次最多只能携带$n=4$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

二、算法设计思路

- 由假设条件可以得知，无人机的活动半径为10公里。而且任意一个配送中心都能满足用户的订货需求，所以全部10个卸货点均在3个配送中心无人机活动半径的交集当中。如下图中阴影部分所示：



- 因此对于任一卸货点，我们可以确定让离它最近的配送中心为其配送，这样原问题简化为单个配送中心为其附近卸货点配送的问题。
- 对于决策，由于无人机可以带多个订单货物，且不同卸货点需求的货物数量不同，不能将决策看成普通的旅行商问题。由于目标是在一段时间内，所有无人机的总配送路径最短，同时满足所有订单的优先级别要求，因此目标函数可以看作是最小化送达单个货物的无人机的平均飞行里程：
    - min ∑Distance/∑Cargo
- 因此，在单独考虑单个货物的无人机的平均飞行里程的情况下，要使其最小，需要：
    1. 无人机满载
    2. 无人机中货物全部来自于同一个卸货点
- 其中第二个条件满足三角不等式：如图，当无人机中的货物来自于同一个卸货点时，其往返路程小于货物来自于不同卸货点的往返路程。



- 当上述条件无法满足的情况下，则优先考虑将无人机装满，同时满足无人机航行距离不超过20公里。

- 为了使得上述条件更容易满足，我们可以采用消极的送货方式，让送往同一卸货点的货物尽量地等待，直到攒满4个为止。

- 最后，为了使得约束条件——订单的优先级别要求得到满足，每轮都将剩余送达时间 >=10分钟，且 <=30 分钟的订单加入到需要完成的订单当中，即使送至同一卸货点的订单未达到4个。相应地，要为运送到多个不同卸货点的情况制定相应的运送策略和运送路径。由于各个卸货点的剩余订单数量均小于无人机的载货量，所以为了尽可能少派出无人机，在满足其他约束的条件下，配送这些订单的无人机至少需要向两个卸货点运送。

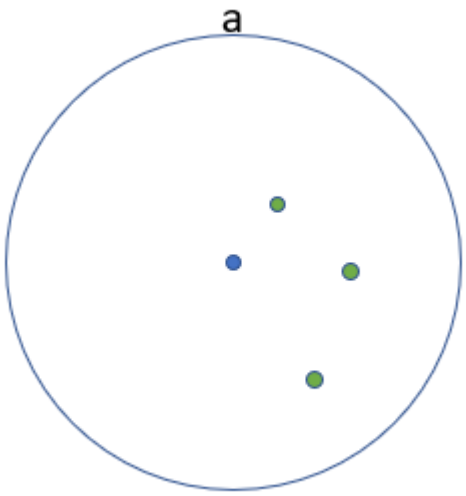
三、具体算法

根据上述思路，该无人机配送路径规划问题的算法具体步骤如下：

- 配送节点集合 P={Pa, Pb, Pc}, 卸货节点集合 X={X1, X2, ..., X10}。这两个顶点集合相加，构成配送图 G 的顶点集 V。

1. P 中的每个节点对应一个 G 的子图 Ga, Gb, Gc，初始时，Ga 的顶点集 Va={Pa}。对于 X 中的每个节点 Xi，分别计算其到 P 中各个节点的距离，将 Xi 加入与之距离最小的节点对应的子图的顶点集中，如果距离相同，则随机加入一个。由此，得到了 G 的3个子图，现在单独对每个子图进行无人机配送路径规划。

2. 以 Ga 为例，设 Va={Pa, X1, X2, X3}，其中蓝色点代表配送中心，绿色点代表卸货点。



   3个卸货点的订单数量为 C1, C2, C3，所有卸货点的顶点集合为 D，对于某一订单 Di，其剩余送达时间为 Ti(min)。对于新的订单，优先级别为一般的剩余时间为180分钟，较紧急90分钟，紧急30分钟。每隔20分钟对订单数量以及订单剩余送达时间进行更新，更新后的剩余送达时间 Ti=Ti-20min。

3. 检查 C1>=4，如果为真，选择剩余时间最短的4个订单的集合 Df 装入一架无人机运送，其路径为 Pa -> X1 -> Pa，C1=C1-4，D=D-Df，以此类推对各个卸货点进行相同操作。

4. 将剩余订单 Dr=D 中剩余送达时间 T，>=10min，且 <=30min 的订单加入该轮必须配送的订单集合 Dm 中。

5. 通过 Dm 找到对应的卸货节点集合 Xm, Vm={Pa}。将 Dm 的配送问题看作一个装箱问题，使用贪心算法确定所需无人机最小数量的近似解：对卸货节点按订单数量排序，不断选取小于无人机剩余空间的最多的

订单装入一架无人机，同时将对应的卸货节点加入集合 Vm，用最小生成树近似算法求 Vm 旅行商回路的近似解，若加入卸货节点后旅行商回路 >20(km)，则跳过该节点订单，尝试装如比它小下一个节点的订单，直到没有节点订单可装入该无人机。重复上述步骤，直到将 Dm 中的订单货物全部装如无人机，得到装箱问题的贪心算法近似解。而无人机的路径即为 Vm 旅行商回路的近似解。

6. 每隔20分钟重复上述步骤，更新无人机的路径规划。

四、算法伪代码

```
Arguments: 无人机配送中心数量 J，卸货点数量 K，时间间隔 T，订单数量 M，无人机载货量 N
Input: 配送节点集合 P，卸货节点集合 X，所有节点构成的完全图 G
Output: 无人机运送路径规划 W = {D: R}
/*
* D为派出的无人机集合
* R为各无人机的路径以及路径上各卸货点的送货数量
*/

Initialize W <- {}
for p ∈ P do:
    Initialize p.V <- {p}

for x ∈ X do:
    Initialize temp <- Infinity
    for p ∈ P do:
        if distance(x, p) < temp then:
            // Distance()通过图的邻接矩阵计算节点间距离
            temp <- Distance(x, p, G)
            nearest <- p
    p.V <- p.V ∪ {x}

for every T minutes do:
    // 各卸货点生成订单
    Generate_Orders()
    // 更新各卸货点订单列表 v.orders、订单数量 v.n和各订单剩余时间 order.t
    Update()
    Initialize Dm <- {}
    Initialize Vm <- []
    for p ∈ P do:
        for v in p.V do:
            if v == p then:
                continue
            if v.n >= N then:
                // 获取剩余时间最少的N个订单集合
                orders <- Get_N_Most_Urgent_Orders(v)
                W <- W ∪ {d: ([p, v, p], [N])}
                v.orders <- v.orders - orders
                v.n <- v.n - N
            for order ∈ v.orders do:
                if order.t <= 30 and order.t >= 10 then:
                    Dm <- Dm ∪ {order}

        // 对必须配送订单列表包含的卸货点依据订单数量进行降序排序
```

```
        Vm <- Sort(Dm)
        While true do:
            Initialize destinations <- [p]
            Initialize quantity <- []
            Initialize room <- N
            /*
            * Find_Most_Orders_Under_Room()寻找小于等于无人机剩余空间的订单量最大的卸
货点
            * 如果找到返回该卸货点在Vm中的索引，如果找不到返回-1
            */
            While Find_Most_Orders_Under_Room(Vm) >=0 do:
                v = Vm[Find_Most_Orders_Under_Room(Vm)]
                /*
                * TSP_MST()为旅行商问题最小生成树近似算法
                * 返回旅行商回路数据结构
                * lenth成员为回路的长度
                * route成员为经过节点的顺序列表
                */
                tsp <- TSP_MST(destinations ∪ [v], G)
                if tsp.lenth <= 20 then:
                    destinations <- destinations ∪ [v]
                    quantity <- quantity ∪ [v.n]
                    Vm <- Vm - v
                    room <- room - v.n

            W <- W ∪ {d: (tsp.route, quantity)}
            if Vm == ∅ then:
                break

    return W
```