使用近似算法求解无人机配送问题

《高级算法》结课作业

姓　　名：赵峪祺

学　　号：2023202210133

指导教师：林海

编写日期：Jun 30, 2024

目　录

1 问题描述

1.1 问题概括

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，实现一个特定区域内的无人机配送问题的路径规划。

1.2 问题详情

假设在一区域中，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，等等。最后需要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 算法设计

2.1 具体假设

首先，题目并未给出具体的配送中心和配货点数量，及其坐标和距离等。所以为了简化问题，这里假设所有的配送中心和配货点都位于二维平面，且每两点之间都可达。每个配送中心和配货点都有一个坐标(x, y)，两个点之间的距离就是欧氏距离：

$$dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

所以，可以很轻松地构建出一个邻接矩阵A，其中A_{ij}表示表示点i到点j的距离。

接着，我们来假设其他未给出的变量。设它们为以下值：

- 配送中心数量$j = 3$；
- 卸货点数量$k = 6$；
- 生成订单的时间间隔$t = 30(min)$；
- 每次生成的订单数量$m = 6$；
- 无人机一次最多携带数量$n = 3$。

这样就构成了一个完整的问题。我们将单位统一，则已知的其他变量为：

- 三个优先级$Pri_1, Pri_2, Pri_3 = 30, 90, 180(min)$；
- 无人机一次飞行最远路程$d = 20(km)$；
- 无人机一次最大飞行距离$s = 1(km/min)$；
- 总计时间长度$T = 180(min)$

2.2 设计思路

根据题意可得，每一个无人机在配送完后都需要回到原来的配送中心，且目标为路径最短，这让我们想到了 TSP（旅行商）问题，即求经过所有需要经过节点的最短回路。假如题目只有一个配送中心、且同时只能配出一架无人机，那其就是一个旅行商问题，我们可以将产出订单的配货点作为需要经过的节点，配送中心作为起点，用解 TSP 的思路去求解。

但很明显此题目不能直接化为旅行商问题，因为：

1. 题目不仅有一个配送中心；
2. 每个配送中心不止可以配出一架无人机；
3. 每架无人机有最远距离和携带物品限制；
4. 每个订单有优先级和时间限制。

但我们可以这样考虑：假如每架无人机按照此问题的最优解进行配送，那么其走过的轨迹上所有的点（配货点）和起点（配送中心）构成的图的 TSP 回路就是其最优路径。所以理论上来说，我们可以通过穷举所有组合，并一一检查合理的节点组合，单独地对每个组合求其 TSP 回路并进行比较，来获得精确的最优解。但这样做的复杂度非常高，光是穷举就需要$O(n!)$的时间复杂度，而计算和比较又要消耗额外的时间，所以穷举法虽可得到精确解，但基本上不可行。

我们来换个思路：采用近似算法和缩减法。假设系统在某一个时间点要处理一些订单，则我们可以分别以每个配送中心为出发点、所有的持有订单的配货点作为其他节点构成一个图，来求解一条近似最优的 TSP 回路。这里考虑：

- 对于配送中心来说，这几条回路就是一次性配送所有订单的较优解；
- 但对于持有订单的配货点而言，它的最优解是直接从最近的配送中心出发，到达配货点再返回。

所以：

- 如果贸然取配送中心的几个 TSP 回路，则必然有更好的解，因为不是所有的配货点都必须等待一次性配送；
- 如果直接取配货点的最短回路，则不能保证其全局最优性，因为没有考虑到配货点之间的距离。

那么我们的算法怎么做？很简单，只需在此基础上加以限制来缩减路径即可：

1. 对所有待配送订单进行排序，让其中的订单按照其持有者到最近配送中心的距离的升序进行排序（贪心选择，从代价最小的订单开始）。
2. 对已排序的订单进行遍历，依代价由小到大选择未配送的订单，然后求得距离此订单持有者（配货点）最近的配送中心所对应的 TSP 回路；
3. 遍历此回路的每个节点，如果节点有更优解（直接到当前节点的最近配送中心

的回路的总距离更短）或不满足要求（包括无人机最大里程限制、最大携带限制和优先级要求），则在路径中删除此节点；

4. 如果节点所持有的所有订单都满足条件，则都标记为已配送；否则只将满足条件的订单标记为已配送；
5. 如果路径中的节点经此路径后其所有订单都被标记为已配送，在可选节点集中删除它（确保其不会被重复选择来构建图）；
6. 回到 2，重复，直至所有的订单都已得到配送。

求解 TSP 问题难度较高，但我们可以通过近似算法来求得一个可接受的较优解。因为我们假设问题的场景位于二维欧氏空间，所以节点间的距离必定满足三角不等式，这就可以使用上课提到的最小生成树近似法来求解 TSP 的较优回路，具体算法设计请见下一小节。因为我们选的是距离配货点最近的配送中心，即在其回路中可以较快遍历到此配货点，可以保证解较优；而且在选择订单前，首先根据订单持有者到最近配送中心的距离来进行排序，贪心选择更优的代价开始算法，减少随机性；同时，在遍历回路时，若某节点有更优解，则舍弃路径中的此节点，使其在别的更优回路或是自己对应的最优回路被选择，确保得到更好的解。结束后，算法会输出一系列的路径，其数量小于等于待处理的订单数量。

关于调度，因为即使是最优的订单都有 30 分钟的等待时间，而无人机受限于速度和最大距离，最多只能飞 20 分钟。为了确保每个订单都能送达，我们设每个配货点到配送中心的距离都不大于 10km 以保证无人机可送达。我们的假定是每 30 分钟生成一次订单，所以一定可以在下一次生成订单前配送完毕所有优先级最高的订单。那么对于调度的设计就简单了，只需在每个时间片仅处理优先级最高的订单，然后在下一个时间片将上一个时间片的其他优先级订单的优先级集体提高一级即可。最后一个时间片会将所有未处理的订单直接提高到最高优先级进行处理。

2.3 算法流程

首先是对指定配送中心通过最小生成树来近似求解最优回路，使用 Prim 算法。设构成的图的总节点数为n，则获得最小生成树的时间复杂度是$O(n^2)$，而先序遍历此树的时间复杂度是$O(n)$，所以算法 1 总的时间复杂度为$O(n^2) + O(n) = O(n^2)$。

Algorithm 1 Get loop by MST

s ← chosen distribution center;
R ← distribution points that have orders;
G ← build graph by s and R;
T ← generate MST by **Prim Algorithm**, with root s;
L ← prior-order traversal of T;
return L;

然后是对于每个订单，获得其近似最优配送路径，一条路径可能会处理多个订单。这里会根据具体的限制条件和是否有更好的解来进行路径缩减。

我们假设系统每批次生成k个订单，每个图的总节点数为n。对订单的排序使用快速排序，其时间复杂度为$O(klogk)$，且算法 1 的时间复杂度为$O(n^2)$，则算法 2 的总计时间复杂度为$O(kn^2) + O(kn) + O(klogk) = O(kn^2)$。

Algorithm 2 For each orders, reduce and limit loops to get path

O ← orders to handle;
P ← path results to output;
k ← set to 0, count of P;
Quick Sort O by each O_i's owner's distance to nearest distribution center;
for i in length of O do:
    continue if O_i is handled;
    s ← nearest distribution center of O_i's owner;
    P_k ← the minimal loop path, using **Algorithm 1**, with argument s;
    for p in P_k do:
        continue if p is s;
        o ← p's order;
        s' ← p's nearest distribution center;
        p^{-1} ← The previous node of p in P_k;
        if o's priority is not satisfied then:
            delete p from P_k;
        endif
        if $distance(p^{-1} \rightarrow p \rightarrow s) > distance(p^{-1} \rightarrow s + s' \rightarrow p \rightarrow s')$ then:
            delete p from P_k;

endif

if $distance(\{s \to \cdots \to p^{-1}\} \to p \to s)$ > drone's max fly distance then:

delete p from P_k;

endif

if orders' number until p^{-1} = drone's max carry then:

delete p from P_k;

endif

if p still in P_k then:

if p's orders can all be handled then:

set all orders of p to handled;

remove p from choosing set;

else

set o to handled;

endif

endif

endfor

$k \leftarrow k + 1$;

endfor

return P;

经过算法 2 求解得到的回路可以认为是每一个批次订单的较优配送回路。接着再使用如上所说的简单调度进入接下来的时间片进行操作。

Algorithm 3 Schedule

$LO \leftarrow$ list including high, mid and low priority orders;

for t in time slices do:

$LO \leftarrow LO$ with new generated orders;

if t is last time slice then:

$LO_{high} \leftarrow \{LO_{high}, LO_{mid}, LO_{low}\}$;

endif

handle all orders in LO_{high} using **Algorithm 2**;

$LO_{high} \leftarrow LO_{mid}$;

$LO_{mid} \leftarrow LO_{low}$;

$LO_{low} \leftarrow \{\}$;

endfor

3 代码实现

本设计的代码使用 Rust 1.78.0 实现，随附在提交的压缩包中，代码同样也上传了 GitHub：https://github.com/CircuitMurderer/drone_delivery_solve。运行方式：

```
cd [repo dir]
cargo run
```

以下是一些运行结果。

邻接矩阵：

```
======
AdjMat:
0.0000 4.4721 5.0990 6.0828 7.6158 7.6158 6.3246 6.4031 11.3137
4.4721 0.0000 4.2426 2.2361 5.0990 3.1623 4.4721 2.2361 7.2111
5.0990 4.2426 0.0000 6.4031 2.8284 6.3246 1.4142 4.1231 7.6158
6.0828 2.2361 6.4031 0.0000 6.7082 2.2361 6.4031 3.1623 7.2801
7.6158 5.0990 2.8284 6.7082 0.0000 5.6569 1.4142 3.6056 5.0990
7.6158 3.1623 6.3246 2.2361 5.6569 0.0000 5.8310 2.2361 5.0990
6.3246 4.4721 1.4142 6.4031 1.4142 5.8310 0.0000 3.6056 6.3246
6.4031 2.2361 4.1231 3.1623 3.6056 2.2361 3.6056 0.0000 5.0000
11.3137 7.2111 7.6158 7.2801 5.0990 5.0990 6.3246 5.0000 0.0000
```

路径和 cost：（1、4、7 为配送中心）

```
======
Time slice 1
[New orders] 6-Mid 3-Low 8-Low 0-Low 2-High 5-High
[High-pri orders] 2-High 5-High
[Process queue] 5-High 2-High
[Routes and costs]
        Min cost: 4.4721        Route: [7, 5, 7]
        Min cost: 5.6569        Route: [4, 2, 4]
[Total min cost] 10.1290
======
Time slice 2
[New orders] 6-Low 3-High 8-High 0-High 2-Mid 5-Mid
[High-pri orders] 6-High 3-High 8-High 0-High
[Process queue] 6-High 3-High 0-High 8-High
[Routes and costs]
        Min cost: 2.8284        Route: [4, 6, 4]
        Min cost: 12.7910       Route: [1, 3, 0, 1]
        Min cost: 10.0000       Route: [7, 8, 7]
[Total min cost] 25.6194
======
Time slice 3
[New orders] 6-High 3-Low 8-High 0-High 2-High 5-Mid
[High-pri orders] 3-High 8-High 0-High 2-High 5-High 6-High 8-High 0-High 2-High
[Process queue] 6-High 3-High 5-High 2-High 2-High 0-High 0-High 8-High 8-High
[Routes and costs]
        Min cost: 5.6569        Route: [4, 6, 2, 4]
        Min cost: 16.7823       Route: [1, 3, 5, 8, 1]
        Min cost: 8.9443        Route: [1, 0, 1]
        Min cost: 10.0000       Route: [7, 8, 7]
[Total min cost] 41.3834
======
Time slice 4
[New orders] 6-Low 3-High 8-High 0-Mid 2-Low 5-Low
[High-pri orders] 6-High 5-High 3-High 8-High
[Process queue] 6-High 5-High 3-High 8-High
[Routes and costs]
        Min cost: 2.8284        Route: [4, 6, 4]
        Min cost: 16.7522       Route: [7, 5, 3, 8, 7]
[Total min cost] 19.5807
```

```
======
Time slice 5
[New orders] 6-Low 3-High 8-High 0-High 2-High 5-Mid
[High-pri orders] 3-High 0-High 3-High 8-High 0-High 2-High
[Process queue] 3-High 3-High 2-High 0-High 0-High 8-High
[Routes and costs]
        Min cost: 12.7910       Route: [1, 3, 0, 1]
        Min cost: 15.5432       Route: [4, 2, 8, 4]
        Min cost: 8.9443        Route: [1, 0, 1]
[Total min cost] 37.2785
======
Time slice 6
[New orders] 6-Mid 3-High 8-Low 0-Low 2-Low 5-High
[High-pri orders] 6-High 2-High 5-High 5-High 3-High 5-High 6-High 6-High 8-High 0-High 2-High
[Process queue] 6-High 6-High 6-High 5-High 5-High 3-High 5-High 2-High 2-High 0-High 8-High
[Routes and costs]
        Min cost: 2.8284        Route: [4, 6, 4]
        Min cost: 4.4721        Route: [7, 5, 7]
        Min cost: 12.7910       Route: [1, 3, 0, 1]
        Min cost: 15.5432       Route: [4, 2, 8, 4]
[Total min cost] 35.6347
======
[Overall cost] 169.6256
```