Bài toán 1. Hệ thống lấy hàng động truck và drone hoạt động song song và ràng buộc time window

Input: Hệ thống lấy hàng gồm: 1 tập K truck đồng nhất, D drone đồng nhất thực cùng xuất phát từ Depot 0 phục vụ tập C. Trong đó, tập khách hàng C được chia làm 2 phần

- C1 - bắt buộc phải phục vụ xe tải do khối lượng quá lớn hoặc nằm ngoài khả năng bay của drone
- C2- có thể phục vụ bởi cả drone hoặc truck

Constraints:

- **Xe tải** có tải trọng tối đa M_T
- **Drone** có tải trọng tốt đa M_D và bán kính bay tối đa là L_D
- **Khách:**
  - Thời gian đợi tối đa của một khách hàng là L_w (thời gian đợi = thời điểm gói hàng về đến kho hàng – thời điểm gói hàng được lấy xong)
  - Khách hàng i xuất hiện một cách dynamic (không biết trước) vào thời điểm r_i và yêu cầu phục vụ trong khoảng thời gian $[e_i, l_i]$
  - Mỗi gói hàng của khách hàng i có tải trọng yêu cầu: d_i
- **Hành trình:** Xe tải/drone được phép thực hiện nhiều hành trình, mỗi hành trình có thể phục vụ nhiều khách hàng sao cho không vi phạm các ràng buộc ở bên trên
- Thời gian sạc/thay pin drone, thời gian phục vụ khách hàng coi như = 0

Output:

Hành trình của các Truck, Drone

Objectives:

- Objective 1: tối thiểu hoá thời gian hoàn thành nhiệm vụ
- Objective 2: tối đa hoá số khách được phục vụ

Hướng tiếp cận:

1. **RL**
2. **GP**

Sử dụng Genetic Programming. Thuật toán sẽ chia nhỏ thời gian của hệ thống thành các time slot a1, a2,..., an. Thời điểm ai sẽ chạy thuật toán để lập lịch cho xe vận hành vào thời điểm a(i+1)

Tham khảo: Tại đây

Thuật toán sẽ xây dựng 2 luật routing rule và sequencing rule.

Routing rule xác định k xe có độ ưu tiên lớn nhất để thực hiện request req

Tại thời điểm a(i+1) phương tiện k sẽ có reqQueue các khách hàng chờ được phục vụ

Các yếu tố (nút lá của cây GP) ảnh hưởng tới routing rule là:

- Số lượng request trong reqQueue chia cho tổng số request của bài toán
- Chênh lệch giữa capacity của vehicle với tổng demand từ các request trong reqQueue chia cho tổng demand của cả bài toán
- Thời gian di chuyển từ median của các request trong reqQueue tới req đang xét chia cho thời gian close của bài toán
- Thời gian di chuyển từ vị trí hiện tại của xe tới req đang xét chia cho thời gian close của bài toán
- Ở demand của req đang xét chia cho tổng số demand của cả bài toán
- Kiểm tra xem vehicle là drone hay truck

```
switch (opt)
{
    case 0: return vehicle.reqQueue.size() / (double)(p.requests.size());
    case 1: return (vehicle.capacity - vehicleSumOfReqDemand()) / p.sumOfReqDemand();
    case 2: return vehicle.movingTime(vehicle.medianOfReqLoc(), req.location) / p.depot.timeWindow.second;
    case 3: return -vehicle.movingTimeTo(req.location) / p.depot.timeWindow.second;
    case 4: return req.demand / p.sumOfReqDemand();
    case 5: return vehicle.type == vehicleState::DRONE ? 1.0 : 0.0;

    default:
        break;
}
```

Sequencing rule: sequencing rule thì sẽ tính giá trị chỉ số cho vehicle với từng request trong reqQueue của vehicle đó để tìm ra request có chỉ số min để ưu tiên thực hiện.

Các yếu tố ảnh hưởng tới Sequencing rule

- Thời gian di chuyển từ vị trí hiện tại của xe tới req đang xét chia cho thời gian close của bài toán
- Chênh lệch giữa thời gian hiện tại của bài toán và thời gian request được đánh giá để xử lý chia cho thời gian close của bài toán
- TimeUnit Close là khoảng thời gian giữa lúc request đóng và thời gian vehicle hoàn thành request hiện tại đang làm.
- Demand của request chia cho tổng demand của bài toán

- Chênh lệch giữa thời gian hiện tại của bài toán và thời gian mở window của request chia cho thời giand đóng của bài toán
- Thời gian request xuất hiện chia cho thời gian đóng của bài toán

```
switch (opt)
{
    case 0: return vehicle.movingTimeTo(req.location) / p.depot.timeWindow.second;
    case 1: return (curTime - readyTime) / p.depot.timeWindow.second;
    case 2:
    {
        double timeUntilClose = req.timeWindow.second - vehicle.busyUntil;
        return timeUntilClose <= 0.0001 ? 1.0 : (timeUntilClose - vehicle.movingTimeTo(req.location)) / timeUntilClose;
    }
    case 3: return req.demand / p.sumOfReqDemand();
    case 4: return (curTime - req.timeWindow.first) / p.depot.timeWindow.second;
    case 5: return req.appearedTime / p.depot.timeWindow.second;

    default:
        break;
}
```