CJ 대한통운 미래기술챌린지 2023

실시간 주문 대응 Routing 최적화

내 택배 언제와 팀

임승환, 김윤아, 허제욱

0. 구동 환경

운영체제 및 환경: 윈도우 환경

프로그래밍 언어: 파이썬 3.10

필요 라이브러리

numpy == 1.25.2

pandas == 2.0.3

python-dateutil == 2.8.2

pytz == 2023.3

six == 1.16.0

tzdata == 2023.3

환경 설정 방법

① 가상 환경 생성

python --m venv .venv

② 가상 환경 활성화

call .venv/Scripts/activate

③ pip 업그레이드

python --m pip install --upgrade pip

④ 필요 라이브러리 설치 (requirements.txt 기반)

pip install -r requirements.txt

입력 데이터

data/raw 폴더 안에 아래 이름의 csv파일들을 입력

- od_matrix.csv
- orders.csv
- terminals.csv
- vehicles.csv

출력 데이터

results 폴더 안에 아래 이름의 csv 파일 출력

- order_result.csv
- vehicle_result.csv

실행 방법

① Input 데이터 전처리 과정 포함

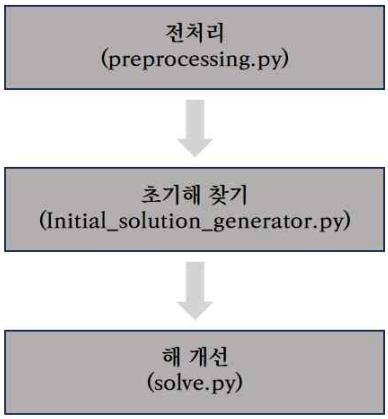
python main.py -r

② Input 데이터 전처리 과정 제외

python main.py

1. 알고리즘 구조

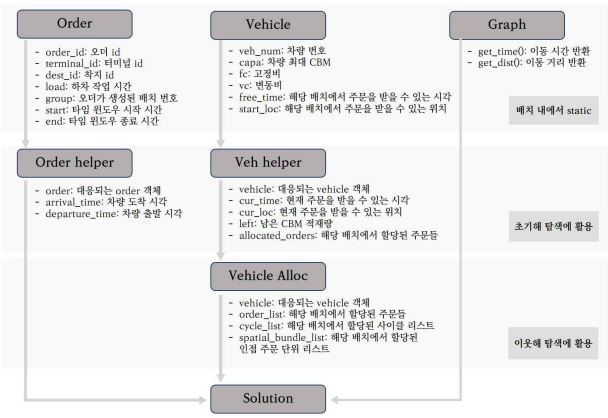
Fig 1. 알고리즘 전체 구조



본 알고리즘은 [Fig 1]과 같이 3단계로 이루어진다. 가장 먼저 전처리 단계에서 입력받은 데이터를 효율적인 연산을 위한 적절한 자료구조와 값으로 변환한다. csv형식으로 입력받은 데이터를 txt 형식으로 저장하여, 전처리 단계는 최초에만 실행하면 된다. 다음으로, 초기해 찾기 단계에서는 문제의 모든 제약 조건을 만족하는 초기 가능해를 찾는다. 마지막으로 해 개선 단계에서 초기해의 주문 배정 결과와 차량 이동 경로를 수정하여 최적의 목적함수 값을 갖는 가능해를 찾는다.

알고리즘에 사용되는 객체와 매서드의 세부 구조는 아래 [Fig 2]와 같다.

Fig 2. 객체와 매서드



크게는 배치 내에서 변화가 없는 클래스와, 배치 내에서 해 탐색에 활용되는 변수들로 나뉘어있다. 배치 내에서 해 탐색에 활용되는 변수 역시 초기해 탐색과 이웃해 탐색을 구분하여 정의하고 있으며, order_helper와 Vehicle_Alloc, Graph로 초기화된 Solution 객체가 완성되면 다음 배치를 위해 Vehicle의 free_time과 start_loc가 업데이트된다.

2. Initial_Solution_Generator

2-1) Initial_Solution_Generator 알고리즘 구조

Algorithm 1 Get initial Solution

```
foreach terminal in terminals do
    terminal_orders ← orders departing
    from this terminal
    call terminal_alloc function
end for
vehicle_alloc_list = []
foreach vehicle in vehicle_list do
    add new Vehicle_Alloc instance to
    vehicle_alloc_list
end for
create a new Solution object
return solution object
```

2-2) Initial_Solution_Generator 알고리즘 설명

Initial Solution Generator는 주어진 주문과 차량 목록에 따라 초기해를 생성하는 클래스입니다.

주요 내용 및 기능은 다음과 같습니다.

초기화 함수(__init__)에서는 트래픽 네트워크 정보를 담고 있는 graph, 해당 배치의 주문 목록, 현재 배치 번호를 입력으로 받습니다.

get_init_solution 함수는 해당 터미널의 주문들을 모아 terminal_alloc 함수를 호출하여 초기 솔루션을 생성하고, 솔루션 객체를 반환하는 메인 함수입니다.

terminal_alloc 함수는 주어진 터미널과 주문 목록에 대해 주문을 할당하는 함수로, next_order에서 어떤 주문을 할당할지를 결정합니다.

next_order 함수는 현재 차량이 자유로워지는 시간과 현재 위치, 남은 용량을 기반으로 다음으로 할당할 주문을 선택합니다.

주문과 차량의 할당 로직을 구체적으로 살펴보면 다음과 같습니다.

주문 분류 우선 각 터미널에 대하여 터미널에 할당된 주문을 분류하고, 주문이 많은 터미널 순서대로 터미널을 정렬합니다.

차량 배열 각 터미널에 대해 차량의 현재 위치, 현재 시간(이미 할당된 주문을 완료한 시간)을 고려하여 터미널에 가장 빨리 올 수 있는 차량부터 배정을 시작합니다. 정렬된 차량 리스트를 돌면서 각 차량에 하나의 사이클을 배정하고, 모든 차량을 다 돌았을 때 주문이 한 번도 배정되지 않을 때까지 차량 루프를 반복합니다.

차량에 주문 배정 각 차량에 대하여 할당할 주문을 더 할당할 주문이 없을 때까지 할당합니다. next_order에서는 이 차량에 다음으로 할당할 주문을 결정하는데, 이 오더가 이미 할당되었는지, 차량의 잔여 용량이 주문의 용량보다 큰지를 검사합니다. 그리고 기본적으로는 차량이 도착할 수 있는 시간과 타임 윈도우를 고려하여 가장 빨리 처리할 수 있는 주문이 먼저 할당되지만, 주문 리스트 중에 이미 1~2일이 지난 주문에 대해서는 점수에 패널티를 부여하여 제약을 어기지 않는 선에서 먼저 할당이 이루어지도록 합니다.

3. solver

3-1) solver 알고리즘 구조

Algorithm 2 Search Neighborhood

```
while no improvement do
    distribute_cycle
    swap_vehicles
    swap_orders
    swap_spatial_bundles
    swap_cycles
end while
foreach methods do
    foreach combinations do
        check cbm
        check time (72hrs & leftover)
        check cost reduction
    end do
end do
```

각 이웃해 탐색 과정은 구체적으로 다음과 같다.

- distribute_cycle: 해당 배치에서 오더가 할당되지 않은 차량에 사이클 단위로 나눠주는 개선
- swap vehicle: 차량에 배정된 오더 전체를 교환
- swap orders: 오더 단위로 교환
- swap spatial bundle: 인접 주문 단위로 교환
- swap cycle: 사이클 단위로 교환

3-2) solver 알고리즘 설명

해당 코드는 초기해에 대하여 이웃해를 탐색하며 해를 개선하는 Solver 클래스를 정의합니다.

초기화 함수(__init__)는 초기해에 해당하는

solution, 트래픽 네트워크 정보를 담고 있는 graph, 그리고 현재 처리중인 데이터 배치 번호를 입력 변수로 받습니다. 그리고 cur_batch는 현재 배치가 마지막 배치인지 여부를 반환하는 last라는 속성을 결정합니다.

최적화 메인 루프 solve 함수는 주요 최적화 루프를 실행하는 메인 함수로, 먼저 다양한 최적화 기법을 포함하는 함수 목록 funs를 정의합니다. 이후, 설정된 반복 횟수(NUM_ITER)에 따라 주요 최적화 루프를 실행하고, 각 반복에서는 funs에서 정의된 모든 최적화 기법을 순서대로 시도합니다. 만약 특정 시간(TIMELIMIT_SEC)을 초과하거나 어떤 최적화도 발생하지 않았다면 루프가 종료됩니다.

인접해 탐색 방법 distribute_cycle 메서드는 사이클을 새로운 차량에 재분배하는 것에 초점을 맞추고 있습니다. distribute_cycles 함수에서 모든 차량 쌍의 조합을 생성하고, 각 쌍에 대하여 차량의 용량과, 시간 제약, 비용 개선 등의 조건을 검사하여 가능하다면 veh1과 veh2의 주문 리스트를 업데이트하고, 그렇지 않다면 False를 반환합니다.

교체 과정은 차량 단위, 사이클 단위, 인접 주문 단위, 그리고 오더 단위로 이루어집니다. 메서드는 교체 조합을 생성하는 swap_vehicles, swap_orders, swap_spatial_bundle, 그리고 swap_cycles 함수가 있고, 각 함수명 앞에 'do_'라는 prefix가 붙어있는 메서드에서 해당 교체를 수행했을 때 제약 조건을 위반하지 않는지 여부와 해 개선 여부를 판단하여 swap을 진행합니다. 각 제약조건 판단 방식은 다음 문단에서 다루고 있습니다. _try가 붙어있는 distribute_cycle_try, spatial_bundle_try, swap_cycle_try의 경우 한 번 swap이 진행됐을 때에 묶음 단위가 바뀌어서 묶음의 인덱스가 바뀌는 경우에 정확한 교체 조합 생성을 위해 사용하는 함수입니다.

교체 여부 판단 'do_'라는 prefix가 붙어있는 교체 여부를 결정하는 함수에서는 CBM 제약과 시간 제약, 그리고 비용 개선을 판단하여 교체 여부를 결정합니다. '타겟 오더'를 차량A에서 차량B로 옮기려고 할 때를 가정해봅니다. CBM 제약 판단을 위해 타겟 오더들 중 가장 CBM이 가장 큰 오더가 차량B의 CBM 제약을 어기는지를 확인합니다. 타겟 오더가 옮겨진 후 사이클은 CBM까지 고려하여 재생성되기 때문에 CBM 판단에서 이전 차량A에서의 사이클 단위를 고려할 필요는 없습니다. 시간 제약 판단을 위해 타겟 오더를 차량 B로 옮겼을 때, 해당 배치의 각 오더들이 72시간 제약을 어기는지 여부와 타겟 오더를 포함한 차량 B에 할당된 오더 중 마지막 오더의 시작 시간이 다음 배치로 넘어가는지 여부를 판단합니다. 이는 초기해를 할당할 때 다음 배치로 넘어가는 주문을 이월시켰기 때문에, 동일한 맥락에서 solver를 작동시키기 위한 장치입니다. 그리고 마지막으로, 비용 개선을 판단하여 개선의 교체되는 차량들 기준으로 비용 개선의 delta값만 우선적으로 판단하여 교체 여부를 결정하고, 비용이 개선된다면 교체를 진행합니다.