Optimization Grand Challenge 2024

OPTMATE

October 24, 2024

목차

알고리즘 개발방향

- 문제에 따른 특성 구분
  - 노드간 거리의 가까움 정도
  - 한계시간(deadline)의 범위
- 불필요한 중복 계산 최소화
- 가능한 다양한 주문 조합 탐색

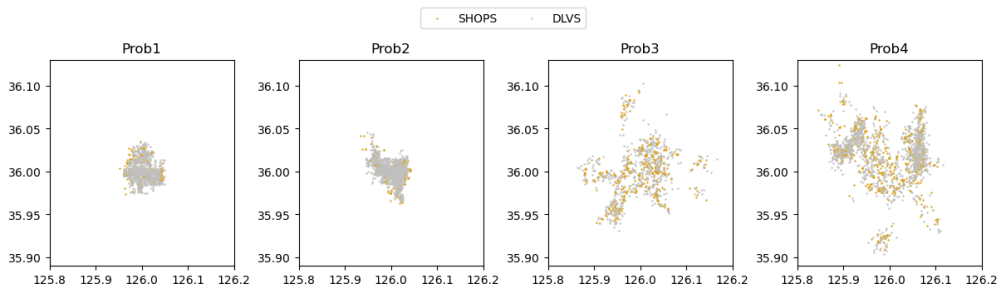


Figure 1: 문제 예시

알고리즘 로직

1. feasible bundle search	2. bundle drop	3. math model
- numpy array 사용	- 주문 조합별 경로 고려	- 최소 배달 비용
- labeling 알고리즘과 유사	- 묶음별 주문 고려	- 주문 할당
- 6개 묶음까지 탐색	- 가능한 다양한 조합 포함	- 배달원 가용수 고려

- Bundle Search Rule
  - 2개부터 6개 묶음까지 순서대로 extension
  - 주문별 SHOP, DLV 거리간 합의 최소부터 탐색
    0: [(30+2, 1), (12+10, 2), (5+8, 3)] → [3,2,1]
  - 주문별 extension 개수는 문제 특성, 주문 수, 시간 제한에 따라 달라지는 parameter

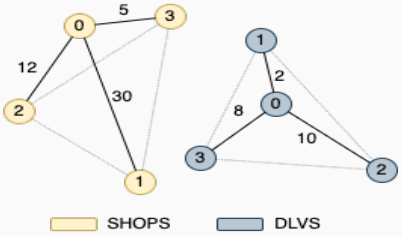
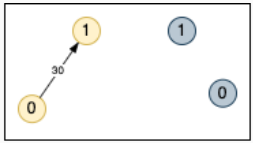


Figure 2: 주문 0에 대한 거리 예시

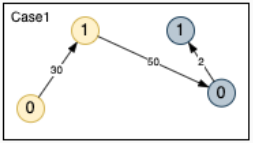
알고리즘 로직 - feasible bundle search

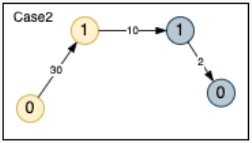
- Label 구성 요소
  [남은 용량, 최소 한계시간, SHOP 거리합, 마지막 SHOP과 첫 DLV 거리, DLV 거리합, 배달원 인덱스, 배달 비용, 방문순서]



주문 ID	0	1
부피	10	20
한계시간	70	80

배달원	BIKE	WALK	CAR
Idx	0	1	2
용량	100	50	200





Case1.

[70, 70, 30, 50, 2, 0, c_1, 0, 1, 0, 1]
[20, 70, 30, 50, 2, 1, c_2, 0, 1, 0, 1]
[170, 70, 30, 50, 2, 2, c_3, 0, 1, 0, 1]

Case2.

[70, 70, 30, 10, 2, 0, c_4, 0, 1, 1, 0]
[20, 70, 30, 10, 2, 1, c_5, 0, 1, 1, 0]
[170, 70, 30, 10, 2, 2, c_6, 0, 1, 1, 0]

Figure 3: 2개 묶음 배송에 대한 label 생성 예시

알고리즘 로직 - feasible bundle search

2개 묶음 배송 label을 생성하면서 Possible Matrix도 함께 만듦

Possible

배달원 d가 주문 o_1과 주문 o_2의 묶음 배송이 가능하면 1 아니면 0 값을 가지는 Matrix

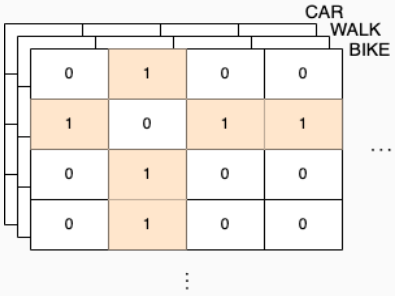


Figure 4: Possible Matrix 예시

- Possible Matrix의 초기값은 모두 0
- 묶음 용량과 한계시간 제약을 모두 만족하면 1
- 2개 묶음이 불가능한 주문 조합은 3개 이상 묶음도 불가능하므로 extension할 때 제외함
- 문제의 난이도는 α를 사용하여 유추함

$$\alpha = \frac{\sum Possible}{K * (K-1) * 3}$$

- α값이 커질수록 탐색 범위가 늘어남
  Sample 문제들은 보통 0.3 이하

알고리즘 로직 - feasible bundle search

α를 사용하여 다음과 같이 extension할 주문 개수를 결정

if $K \leq 300$ **then**
$cut = min(default_k + (Timelimit/\beta_k * (1-\alpha)), K-1)$
else if $K \leq 500$ **then**
$cut = default_k + (Timelimit/\beta_k * (1-\alpha))$
else if $K \leq 750$ **then**
$cut = default_k + (Timelimit/\beta_k * (1-\alpha))$
else if $K \leq 1000$ **then**
$cut = (default_k * (1-\alpha)) + (Timelimit/\beta_k * (1-\alpha))$
else
$cut = (default_k * (1-\alpha)) + (Timelimit/\beta_k * (1-\alpha))$

$\Rightarrow$ $default_k$값은 K가 커질수록 작아지고 30-40개 정도일 때 적당하게 좋은 솔루션을 찾음
β_k는 K가 커질수록 커지고 3-10 사이값으로 설정함

알고리즘 로직 - feasible bundle search

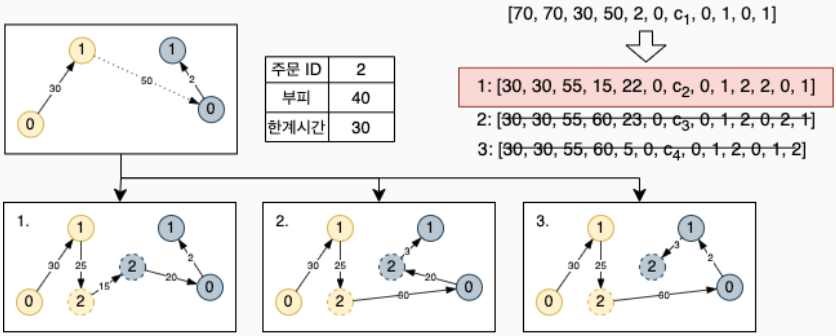


Figure 5: 3개 묶음 배송에 대한 label 생성 예시

- 3개 이상 묶음 배송 경로부터는 계산량이 점점 많아지기 때문에 이전 label에서 extension 한 후 최소 비용인 label만 남겨두고 나머지는 더이상 extension하지 않음

알고리즘 로직 - feasible bundle search

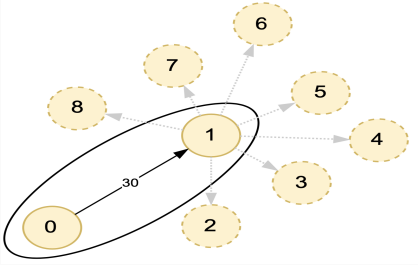


Figure 6: 0→1 묶음 SHOP 경로

- label 경로에서 shop 마지막 주문 기준으로 shop, dlv 거리간합이 작은 주문 순서대로 extension함
- 이 때 묶음 배송하는 다른 주문과도 거리간 합이 작은 주문일 경우에만 extension함
- 마지막 주문만 고려하는 것보다 모든 묶음 배송 주문과의 거리를 고려하는게 결과가 좀 더 좋았음

- 묶음 배송을 extension하면서 추가 생성되는 label수는 30-60개 사이로 제한을 둠
- 다른 주문들과 묶음 배송이 덜 되는 주문들을 최대한 많이 고려하는 것이 중요함
- 특히 3개, 4개 묶음 배송에 대한 label 수에 따라 탐색 범위가 기하급수적으로 증가하기 때문에 적절하게 잘 cut하여 extension할 label 수를 줄이는 것이 중요함
- 이와 같은 방식으로 6개 묶음 배송 경로까지 탐색

알고리즘 로직 - bundle drop

- Step.1 묶음별 주문 조합에 대해 최소 비용 경로 선택
  - 3개 묶음 배송 예시 (주문 : 0, 1, 2)

  ({0, 1, 2}, *rider*) : [[..,50, 1, 2, 0, 0, 1, 2],[..,20, 1, 0, 2, 2, 0, 1],[..,40, 0, 2, 1, 1, 2, 0],..]

⇒ ({0, 1, 2}, *rider*) : [~~[..,50, 1, 2, 0, 0, 1, 2]~~, [..,20, 1, 0, 2, 2, 0, 1] ,~~[..,40, 0, 2, 1, 1, 2, 0]~~,..]

⇒ ({0, 1, 2}, *rider*) : [...,20, 1, 0, 2, 2, 0, 1]

- Step.2 각 주문별 최소 비용 조합 20-40개 선택

주문 번호	(배달 비용, 주문 조합) List
0	[(20, {0,1,2}), (80, {0,14,20}), (10, {0,10,78}),···]
1	[(20, {0,1,2}), (75, {1,4,30}), (50, {1,44,64}), ···]
2	[(20, {0,1,2}), (45, {2,16,22}), (50, {2,60,99}), ···]
···	···

주문 0번 예시 : $[\cdots, (10, \{0, 10, 78\}), \cdots, (20, \{0, 1, 2\}), \cdots, (80, \{0, 14, 20\}), \cdots]$

주문별로 비용을 기준으로 sorting 후 20-40개 조합 선택

알고리즘 로직 - math model

Parameters

K : 주문 개수
R : 배송 가능한 묶음 배송경로 집합
O : 주문 집합
D : 배달원 집합
RB_d : 배달원 d가 배송 가능한 묶음 배송경로 집합
RO_r : 묶음 배송경로 r에 포함된 주문 집합
OR_r : 주문 o가 포함된 묶음 배송경로 집합
ro_r : 묶음 배송경로 r에 포함된 주문 개수
a_d : 배달원 d의 가용 수
c_r : 묶음 배송경로 r의 배달 비용

Decision variables

x_r : 묶음 배송경로 r이 선택되면 1 아니면 0인 binary variable
y_{ro} : 묶음 배송경로 r에 주문 o가 선택되면 1 아니면 0인 binary variable

알고리즘 로직 - math model

Objective function 평균 배달 비용 최소화

$$\min \sum_{r \in R} c_r x_r / K \tag{1}$$

Constraints

$$\text{s.t.} \sum_{o \in RO_r} y_{ro} = ro_r x_r \quad \forall r \in R, \tag{2}$$

경로 r이 할당되면 r에 포함된 모든 주문 o도 할당

$$\sum_{r \in OR_o} y_{ro} = 1 \quad \forall o \in O, \tag{3}$$

모든 주문 o는 무조건 1개씩 할당

$$\sum_{r \in RB_d} x_r \leq a_d \quad \forall d \in D \tag{4}$$

배달원 d의 가용수 제약

알고리즘 구현

- 알고리즘 구현 언어
  - Python 3.10
- 사용 패키지
  - numba 0.59.1
  - gurobipy 11.0.1
  - defaultdict
  - util.py
- 구현 기술
  - OGC2024 환경에서 별도의 compile 없이 사용 가능
  - Python 속도의 한계를 보완하기 위해 numba 사용
    - nopython - 순수한 기계어로 변환된 코드를 실행하려고 시도 (정적인 코드)
    - cache - 컴파일된 함수를 디스크에 저장하여, 다시 실행될 때 이미 컴파일된 결과를 재사용
    - fastmath - 더 빠른 수학 연산 가능
    - parallel - 다중 코어를 활용하여 병렬 처리 활성화
  - 상용 최적화 Solver Gurobi 사용

*numba: 파이썬 코드를 Just-In-Time(JIT) 컴파일하여 계산이 많은 작업을 빠르게 수행함

알고리즘 특징 및 장점

- 알고리즘 특징 및 장점
  - 결과를 저장하며 중복 계산을 피하는 DP 기반 알고리즘
  - 불필요한 label 생성을 피하기 위해 정의한 extension rule들이 꽤 효과적임
  - label 생성시 4개 CPU CORE가 병렬 처리하기 때문에 빠르게 탐색 가능
  - Python이라는 쉬운 언어로 구현되어 있어 코드가 상대적으로 짧고 간결함
  - 구조가 복잡하지 않기 때문에 문제 목적에 맞게 추가적인 수정이 쉽고 빠름
  - Python에 다양한 라이브러리를 이용하여 추후 머신러닝 적용 등에도 용이함

알고리즘 발전방향

- 알고리즘 속도 개선
  - feasible bundle을 만드는 로직만 numba로 구현되어 있음
  - 만들어진 label을 전처리 하는 과정도 시간이 꽤 소요됨
  - 전처리시 dictionary를 사용하는 방법을 선택하여 numba를 사용하지 않았는데 다른 방식으로 좀 더 빠르게 처리할 수 있는 방법을 찾으면 좋을듯함
  - Python이 아닌 C나 C++ 언어로 변경해보는 것도 방법
- Extension Parameter를 결정하는 로직 개선
  - 경험적으로 Sample 문제들에 대해 어느 정도 수준이면 적당한 parameter값이 계산되는 로직을 만들었지만 제한 시간을 넘기는 경우가 발생함
  - 문제 특성, 주문수, 제한 시간에 따라 적절한 cut을 계산하는 함수를 새롭게 만들어도 좋을 듯함
- Column Generation 적용
  - 현재 방법에서 extension을 줄여서 빠르게 적당히 좋은 초기해를 찾은 후 Column Generation 방식으로 경로 column을 추가하면서 목적함수를 개선하는 방식도 가능할듯함

경진대회 참여 후기

- 도전해볼만한 난이도?
  - 직관적인 문제와 예시
  - 비교적 간단한 가정
  - 상용 최적화 Solver 사용없이 할당 문제는 구현하기 쉽지 않을듯함
  - 예선, 본선에 비해 결선 난이도가 꽤 높았다고 느낌
- Python 코딩 공부
  - 속도 향상을 위해 numba 라이브러리 사용
  - numba 병렬 처리 구현을 위해 구글링하면서 코딩 공부가 많이 되었음
  - 알고리즘 개선을 위해 고민해보는 시간을 가지는 계기가 됨
- 그 외
  - 다른 참가팀의 좋은 알고리즘과 비교하며 성능이 어느 정도인지 가늠할 수 있었음
  - 주 언어로 Python을 사용하기 때문에 이번 대회는 Python으로 구현하였는데 이번 기회에 C나 C++언어에도 흥미가 생겨서 기초부터 배워보려고 함

Thank you