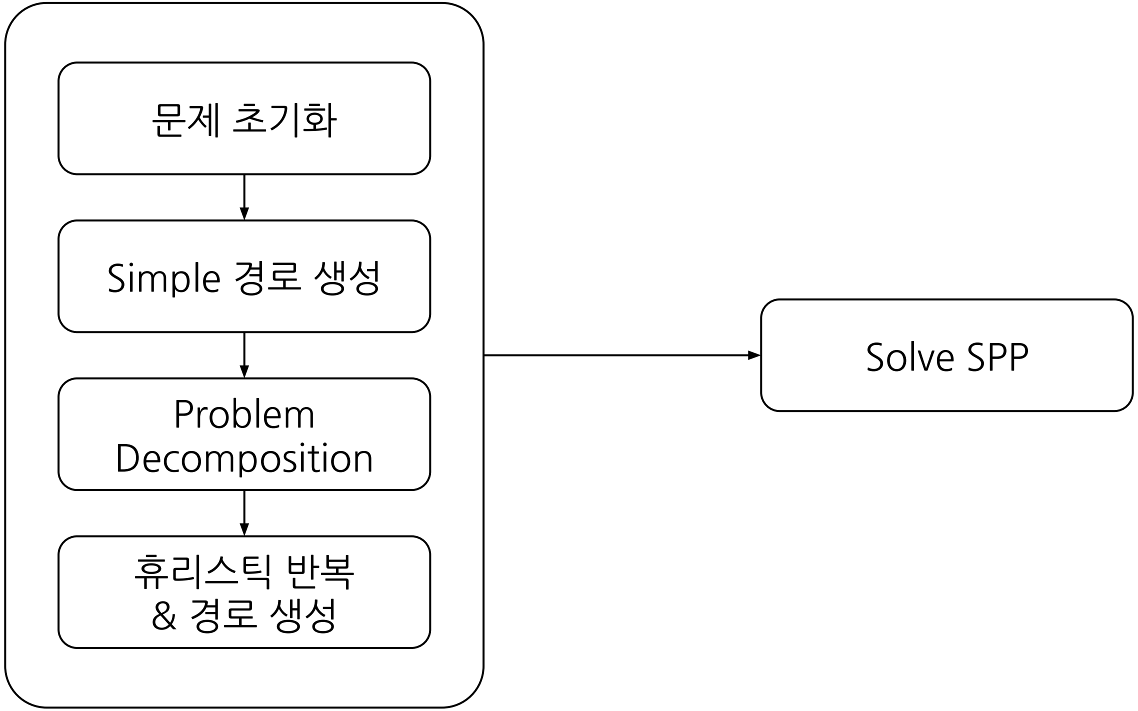
Optimization Grand Challenge 2024

묶음배송 최적화 알고리즘

Team **VIP**
나용수 (서울대학교 산업공학과)

- *Dumas et al. (1991)*
  - PDPTW에 대한 **Set-Partitioning Problem** Formulation
  - **Constrained Shortest Path Problem** as the CG subproblem
  - **Forward-labeling based algorithm** to solve CSPP
- *Ropke and Pisinger (2007)*
  - PDPTW를 풀기 위한 Adaptive **Large Neighborhood Search** Framework
  - **Removal & Insertion Heuristics**
- *Satori and Buriol (2020)*
  - AGES, LNS와 **SPP를 결합**
- *Qi et al. (2012), Tu et al. (2015), Kim et al. (2023)*
  - **Spatio-temporal decomposition of PDPTW instances**


문제 초기화
Simple 경로 생성
Problem Decomposition
휴리스틱 반복 & 경로 생성
Solve SPP

문제 초기화 *Dumas et al. (1991)*

- Tighter Time Windows: 각 라이더마다 계산

$$i = 1, \ldots, n : \text{orders}$$
$$i : \text{index of order } i\text{'s pickup node of order } i$$
$$n + i : \text{index of order } i\text{'s delivery node of order } i$$
$$r_i = \text{readytime}, \; d_i = \text{deadline}, \; s = \text{service time}$$
$$t_{ij} = \text{the travel time of arc } (i, j)$$

$$a_i = r_i$$
$$b_i = d_i - t_{i,n+i} - s$$
$$a_{n+i} = r_i + t_{i,n+i} + s$$
$$b_{n+i} = d_i$$

문제 초기화 *Dumas et al. (1991)*

- Network Construction (Reduction)
  - (a) priority, (b) pairing, (c) vehicle capacity,
  - (d) time windows

$$\text{If } a_i + s + t_{ij} > b_j,\ i,\ j \in \{1,..,2n\},$$
$$\text{then the arc } (i,j) \text{ is eliminated}$$

- (e) time windows and pairing of requests
- (f) same location
- …

Simple 경로 생성

- Simple 경로: 하나 혹은 두 개의 주문만을 포함하는 (feasible) 최적 묶음 배송 경로
  - e.g. 1) 주문 42에 대해 라이더가 [42 픽업 → 42 배송]
  - e.g. 2) 주문 42, 43에 대해 라이더가 [42 픽업 → 43 픽업 → 43 배송 → 42 배송]
- 라이더에 따라 feasibility, cost 등이 모두 다르므로, 배송 순서가 같은 경로들이 라이더가 다르다면 구분된다.
- 주문 개수 n에 대해 최대 $3(n + \binom{n}{2})$ 개의 simple 경로를 생성한다(in 다항 시간).
- 모든 simple routes를 **경로 pool**에 추가 → 경로 pool의 경로들로 SPP의 열(column)을 구성한다.
- 라이더 k에 대해 어떤 두 주문 i, j을 포함하는 simple route가 있다면 **i and j are bundleable.**

LNS Heuristics *Ropke and Pisinger (2007)*

- Pseudo code of LNS
- Solution: feasible한 경로들의 집합 - 모든 주문을 cover

```python
def LNS(init_sol):
    curr_sol = init_sol
    best_sol = curr_sol
    while stop_condition:
        tmp_sol = curr_sol
        Remove(tmp_sol)
        ReInsert(tmp_sol)

        if cost(tmp_sol) < cost(best_sol):
            best_sol = tmp_sol

        if accept_criteria(tmp_sol, curr_sol):
            curr_sol = tmp_sol

    return best_sol
```

LNS Heuristics *Ropke and Pisinger (2007)*

- Removal & insertion heuristics
- Random Removal: q개의 주문을 randomly 골라 제거한다.
- Greedy Insertion: 1) 모든 경로에 대해 (가능하다면) 최소 삽입 비용(= 삽입 전후 비용 증가)을 계산한다.

```
def FindBestPosition(i, solution):
    min_incr_cost = 10000000
    best_route = None
    for route in solution.current_routes:
        new_route = TryInsert(route, i)
        if new_route is not None:
            incr_cost = cost(new_route) - cost(route)
            if min_incr_cost > incr_cost:
                min_incr_cost = incr_cost
                best_route = new_route

    return min_incr_cost, best_route
```

LNS Heuristics *Ropke and Pisinger (2007)*

- Removal & insertion heuristics
- Random Removal: q개의 주문을 randomly 골라 제거한다.
- Greedy Insertion: 2) 전체 removed 주문 중 최소 삽입 비용이 가장 작은 주문을 최소 삽입 위치에 삽입한다.

→ 모든 removed 주문에 대해 최소 삽입 위치를 계산하고 하나의 주문을 삽입하므로 많은 연산이 필요

```
def InsertGreedy(solution):
    best_incr_cost = 10000000
    route_to_insert = None
    for i in solution.removed_orders:
        incr_cost, new_route = FindBestPosition(i, solution)
        if best_incr_cost > incr_cost:
            best_incr_cost = incr_cost
            route_to_insert = new_route

    if route_to_insert is not None:
        Replace(solution, route_to_insert)
```

LNS Heuristics

- Revised Greedy Insertion
  1) removed orders를 randomly shuffle한다.
  2) random 순서대로 greedy하게 삽입한다.
- 모든 현재 경로에 대해 삽입이 infeasible하다면 새 경로를 create.

```
def RevisedInsertGreedy(solution):
    shuffled = random.shuffle(solution.removed_orders)
    for i in shuffled:
        _, new_route = FindBestPosition(i, solution)

        if new_route is not None:
            Replace(solution, new_route)
```

Full Flow of Heuristics

- LNS as a subprocess
- 새 이웃으로 이동할 때마다 전체경로를 최적 순서로 재정렬한다(=solving CSPP).
- 새 경로를 발견할 때마다 pool에 추가한다.

```
def LNS(init_sol, pool):
    curr_sol = init_sol
    best_sol = curr_sol

    while iter_10_times:
        tmp_sol = curr_sol
        n_to_remove = randint(0.4 * n, 0.8 * n)
        RemoveRandom(tmp_sol, n_to_remove)
        RevisedInsertGreedy(tmp_sol)
        ReorderOptimally(tmp_sol)
        AddNewRoutes(tmp_sol, pool)

        if cost(tmp_sol) < cost(best_sol):
            best_sol = tmp_sol

        if accept(tmp_sol, curr_sol):
            curr_sol = tmp_sol

    return best_sol
```

Full Flow of Heuristics

- 전체 휴리스틱 과정은 낮은 비용의 해를 찾는 것이 아니라 많은 경로들을 찾는 데 그 목적이 있다.
- 해 공간의 어떤 한 지점에서 반복적으로 이웃을 탐색하면서, **새로운 경로를 충분히 많이 찾지 못하게 될 때** 현재 해를 이웃으로 이동한다.
- 할당된 시간이 초과될 때까지 반복한다.
- 초기해는 모든 주문이 removed된 상태에서 RevisedInsertGreedy로 생성한다.

```
def HEURISTIC(init_sol):
    pool = Pool()

    curr_sol = init_sol
    while not_time_over:
        tmp_sol = curr_sol

        n_routes_before = pool.size()
        tmp_sol = LNS(tmp_sol, pool)
        n_routes_after = pool.size()
        n_newly_added_routes =
            n_routes_after - n_routes_before

        if n_newly_added_routes < threshold:
            curr_sol = tmp_sol # accept
            threshold *= decay_rate

    return pool
```

Decomposition of Problems + Multi-threading

- (제약) 4개의 CPU(스레드)를 이용할 수 있으므로, 문제를 적절히 4개의 작은 문제로 쪼개어 각 스레드에서 작은 문제에 대한 휴리스틱 알고리즘을 실행한다면 효율적으로 더 많은 경로들을 찾을 수 있을 것이다.
  - e.g. 전체 주문은 400개인데 각 thread가 100개씩의 주문만을 다룬다면 연산량이 감소
- Simple routes를 찾을 때 구한 bundleability 정보를 활용하여 문제를 decompose
  - 두 주문의 bundleability: 각 주문의 특성과 라이더의 특성이 모두 어느 정도씩 반영된다.

$$B_{ijk} = \begin{cases} 1 & : \ i \ \& \ j \ \textbf{are bundleable for rider} \ k \\ 0 & : \ \textbf{o.w} \end{cases}$$

$$B_{ij} = \begin{cases} 1 & : \sum_k B_{ijk} \geq 1 \\ 0 & : \ \textbf{o.w} \end{cases}$$

Decomposition of Problems + Multi-threading

- 주문들이 readytime으로 오름차순 정렬되어 있다고 할 때,

$$O_1 = \{1, \ldots, n/4\}, \qquad O_2 = \{n/4+1, \ldots, n/2\},$$
$$O_3 = \{n/2+1, \ldots, 3n/4\}, \; O_4 = \{3n/4+1, \ldots, n\}$$

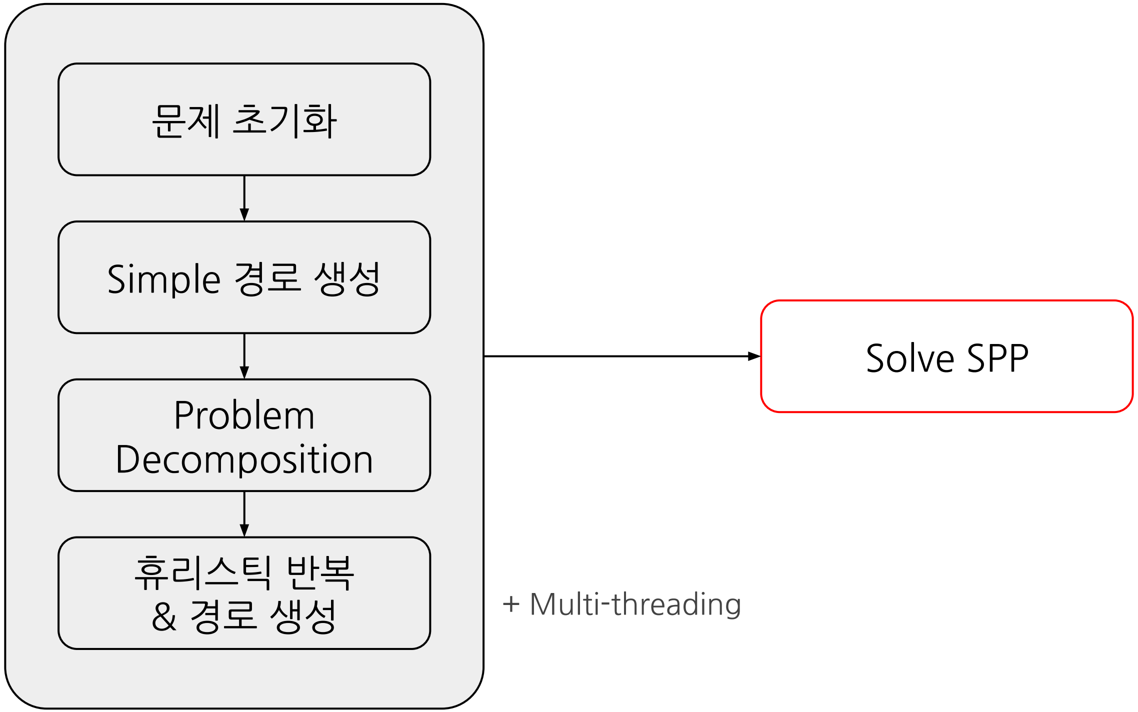
ORDs : a set of all orders

S_t : a set of splited orders (for thread t)

1. $O_t \subset S_t$
2. $\forall j \in \text{ORDs}, \; \exists\, i \in O_t \text{ s.t. } B_{ij} = 1 \;\Leftrightarrow\; j \in S_t$

Decomposition of Problems + Multi-threading

- 4개의 S_t를 각 스레드에 분배하여 병렬 처리
- 만약 휴리스틱 과정이 (시간이 충분할 때) 주어진 주문 집합에 대해 가능한 모든 경로를 탐색 가능하다면, 분할된 주문 집합들로부터 생성 불가능한 경로는 없으므로 전역 최적성을 손실하지 않는다.
- S_t중 최대 집합(혹은 최소 집합)의 크기는 문제 특성에 대한 정보가 될 수 있다.
    - 또는 bundleability matrix B를 활용할 수 있다.
- 주문들의 B.sum(axis=0)가 작을수록 S_t들의 크기가 작아져 decomposition의 효과가 극대화된다.
- 가용 스레드 수가 클수록 각 스레드에 적은 수의 주문이 할당되어 더 많은 경로를 찾을 수 있겠지만, 스레드가 일정 수 이상이 되면 오히려 비효율적이게 될 수 있다.
- 어떤 경로는 여러개의 스레드에서 중복되어 생성될 수 있다. → SPP 이전에 중복 제거 진행


문제 초기화
Simple 경로 생성
Problem Decomposition
휴리스틱 반복 & 경로 생성
Solve SPP
+ Multi-threading

Set-Partitioning Problem

- 휴리스틱 과정의 결과로 pool에 존재하는 모든 경로를 SPP의 column으로 활용

$$\Omega : \text{a set of all generated routes}$$

$$\Omega_k \subset \Omega : \text{routes where its rider is } k$$

$$a_{ir} = \begin{cases} 1 & : \text{ route } r \text{ includes } i \\ 0 & : \text{ o.w} \end{cases}$$

$$\begin{array}{rlr} \text{min.} & \sum_{r\in\Omega} c_r X_r & \\ \text{subject to} & \sum_{r\in\Omega} a_{ir} X_r = 1 & \forall i \in \text{ORDs} \\ & \sum_{r\in\Omega_k} X_r \leq \#\text{Available}_k & k = 1, 2, 3 \\ & X_r \in \{0, 1\} & \forall r \in \Omega \end{array}$$

Implementations

- Language: Python3 & C++
    - ctypes를 통해 C/C++ 코드를 link
    - 사용한 Python Libs/APIs: Numpy
    - 사용한 C++ Libs/APIs: STL containers, OpenMP, GCC Policy-based Data Structure Extensions
    - C++ 추가 빌드 옵션(GCC): O3, Ofast, unroll-loops

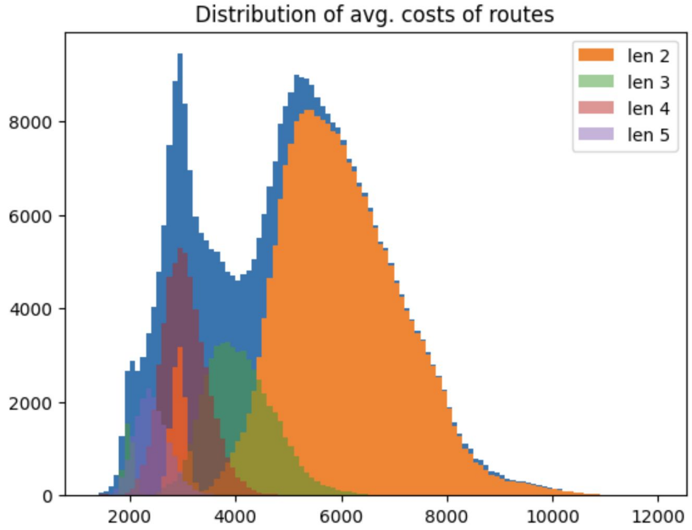
- MIP Solver: Gurobi (used in Python)

Implementations - Parameters & Strategies

- LNS iterations: 매 호출마다 10회 반복
- threshold of HEURISTIC: 전체 주문 개수의 20%
- threshold decay: 95%
- ReorderOptimally()에서 hashing을 통해 이미 정렬된 경로는 정렬 시도하지 않음
- 시간 할당: 전체 timelimit의 35%를 휴리스틱 과정에, 나머지는 MIP solver에 할당
    - bundleability가 특히 낮은 경우 SPP가 쉽게 수렴하므로 휴리스틱 과정에 timelimt의 50% 할당
    - e.g. STAGE2_05.json, STAGE2_06.json
- LNS 반복의 일부(10%)는 임의의 주문들이 아니라, 임의의 경로 자체를 지워서 Guided Ejection Search와 유사한 효과를 유도했다. (minimize the number of vehicles)
- 병렬화할 수 있는 로직은 최대한 병렬화 구현

Implementations - Parameters & Strategies

- 일부 큰 규모의 문제의 경우, 빠른 수렴을 위해 2개 주문을 포함하는 simple routes들 중 평균 비용이 비싼 상위 10%를 제거하였다(for bike and car rider).



특장점

- 다양한 경로를 탐색한다는 목적에 알맞게 LNS 휴리스틱을 수정하여 활용하였다.
- 주문들의 bundleability 정보를 활용해 problem decomposition을 수행하였으며, 이는 병렬화에 용이하다.
- Bundleability 정보는 문제 특성을 파악하는 지표로 활용될 수 있다.

발전 방향

- SPP를 풀 때 solver를 활용하는 것 이상의 내용이 없어, 해당 부분을 보완할 경우 성능 향상이 기대된다.
  - 대회 기간 중 다양한 **Valid Inequality/Cut Generation**을 시도했으나 Gurobi의 자체 휴리스틱이 충분히 좋아 크게 도움되지 못했음.
- 경로들을 평균비용을 기준으로 일부 버림한 것과 같이, 최적해에 포함될 여지가 적은 경로들을 제거하는 과정을 추가할 수 있을 것 ← 다만 Gurobi가 자동으로 column & row elimination을 진행하는 것으로 보임.
  - 특히 휴리스틱 과정과 (Stabilized) CG 과정을 병렬화하는 방법도 시도해 볼 법함

4. 참고문헌

Dumas, Y., Desrosiers, J., & Soumis, F. (1991). The pickup and delivery problem with time windows. European journal of operational research, 54(1), 7-22.

Kim, Y., Edirimanna, D., Wilbur, M., Pugliese, P., Laszka, A., Dubey, A., & Samaranayake, S. (2023, June). Rolling horizon based temporal decomposition for the offline pickup and delivery problem with time windows. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 37, No. 4, pp. 5151-5159).

Qi, M., Lin, W. H., Li, N., & Miao, L. (2012). A spatiotemporal partitioning approach for large-scale vehicle routing problems with time windows. Transportation Research Part E: Logistics and Transportation Review, 48(1), 248-257.

Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation science, 40(4), 455-472.

Sartori, C. S., & Buriol, L. S. (2020). A study on the pickup and delivery problem with time windows: Matheuristics and new instances. Computers & Operations Research, 124, 105065.

Tu, W., Li, Q., Fang, Z., & Zhou, B. (2015). A novel spatial-temporal Voronoi diagram-based heuristic approach for large-scale vehicle routing optimization with time constraints. ISPRS International Journal of Geo-Information, 4(4), 2019-2044.

후기

- 최적화 이론이 현실에서 가질 수 있는 힘을 확인
- 다양한 문헌을 조사하면서 관련 이론을 공부할 수 있는 기회
- 향후 진로 계획을 세우는 데 도움

건의사항&발전 방향

- 규모가 커진다면 참가팀의 Division을 나눠볼 수 있을 것