

OGC 2024

묶음배송 알고리즘 소개

2024.10.24.

TEAM 붉은달

박준석

이민희

최의현

CONTENTS

01

알고리즘 개요

- 문제 정의
- 접근법
- 알고리즘 도식

02

알고리즘 구현 언어

- Python with Gurobi
- Multiprocessing
- Numpy
- Cython

03

알고리즘 구현 기술

- Order Clustering
- Efficient Bundle Generation
- Cost-effective Bundle Generation
- N-M Heuristic

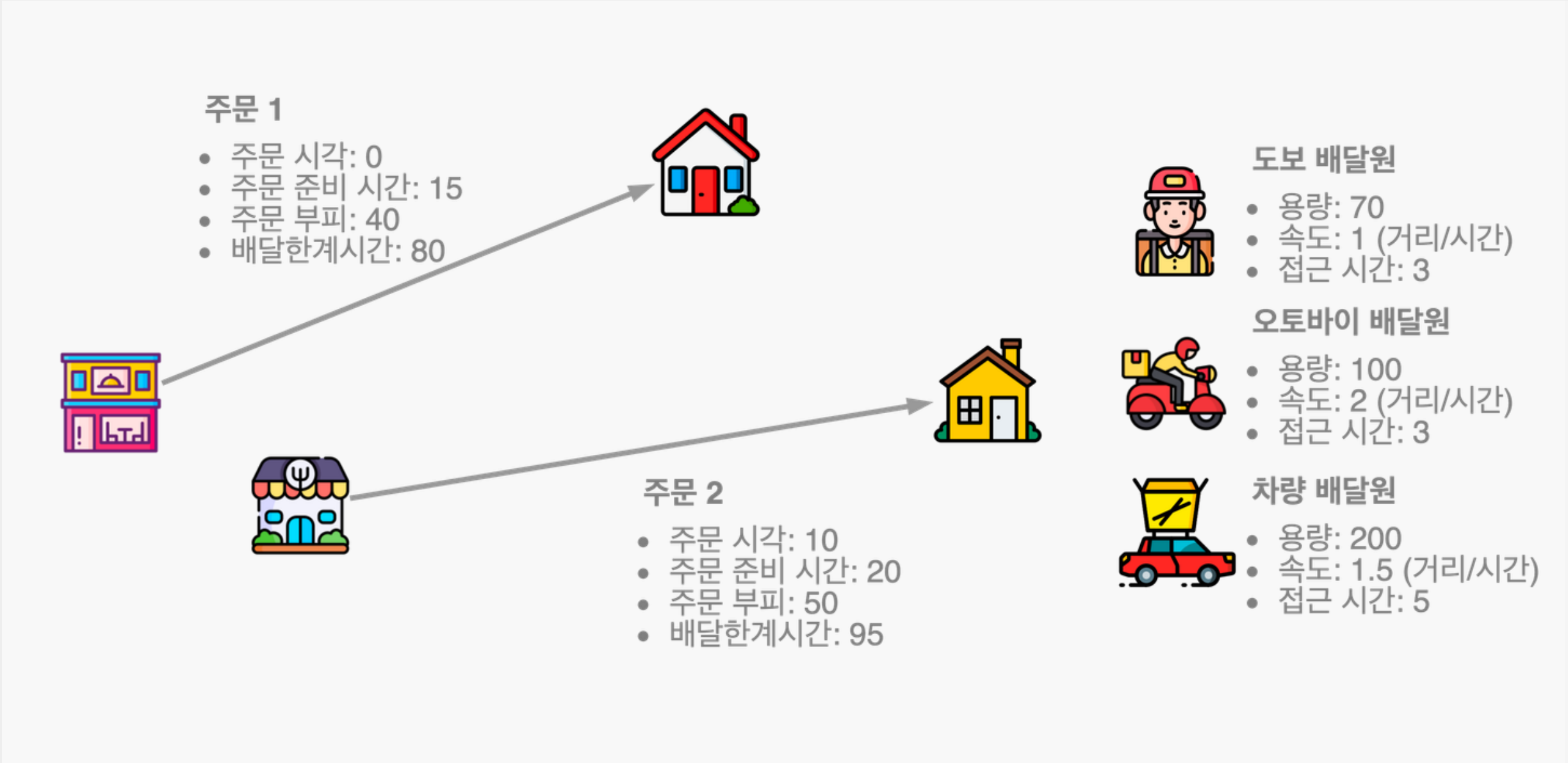
04

알고리즘 특징점 및 발전방향

- Universal Algorithm
- Hyperparameter Optimization
- Algorithm for Sparse data

01 알고리즘 개요

묶음 배송 최적화 문제



주문 리스트가 주어졌을 때 묶음 주문을 활용하여 최소 비용으로 주문을 모두 처리하는 방법을 찾는 문제

최적화 문제 표현

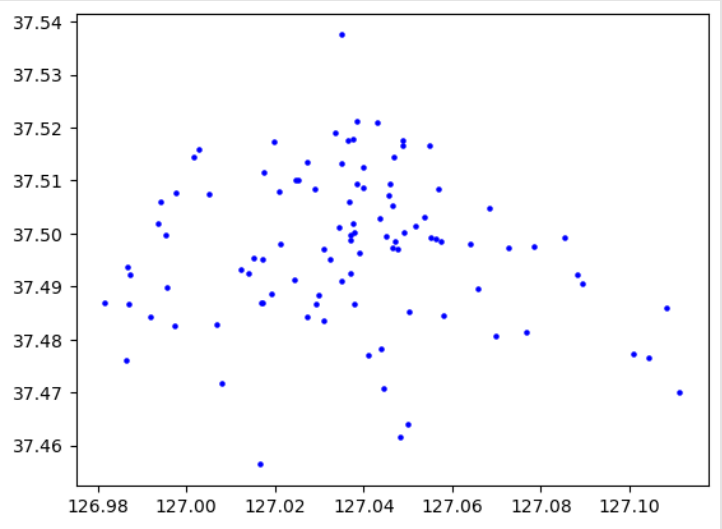
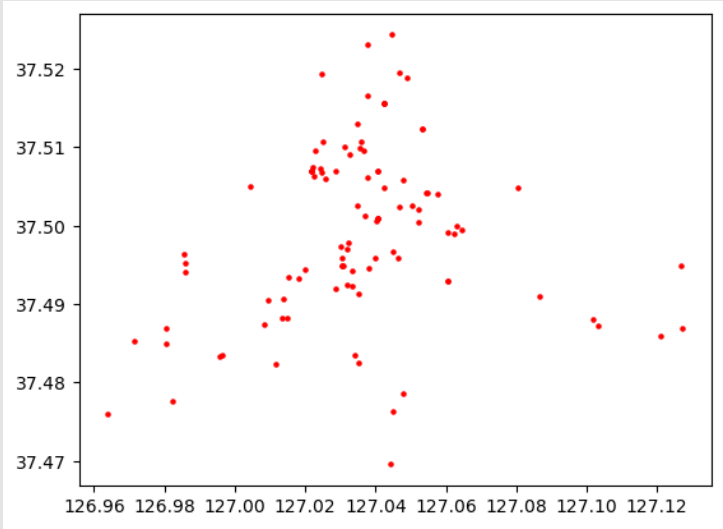
minimize 모든 주문을 처리하기 위한 총 비용
subject to

- (1) **용량 제약** : 묶음 배송 음식의 총 부피 합 \leq 배달원의 한계 용량
- (2) **시간 제약** : 음식은 제 시간 내에 배송 완료
- (3) **방문 순서 제약** : 모든 음식을 수령 받은 뒤 배달 시작
- (4) **주문 만족 제약** : 모든 음식 주문을 만족
- (5) **배달원 할당 제약** : 한 배달원에는 하나의 묶음, 유한 배달원 수

데이터 EDA

[주문정보]

- 주문 id, 주문시간, 픽업/딜리버리 위치, 용량 등



- 평균 용량 20~25
- 평균 거리 3000~4000 / 9000~12000

[배달원 정보]



속도: 1.3
용량: 70
접근 시간: 120



속도: 4.2
용량: 100
접근 시간: 120



속도: 5.2
용량: 200
접근 시간: 180

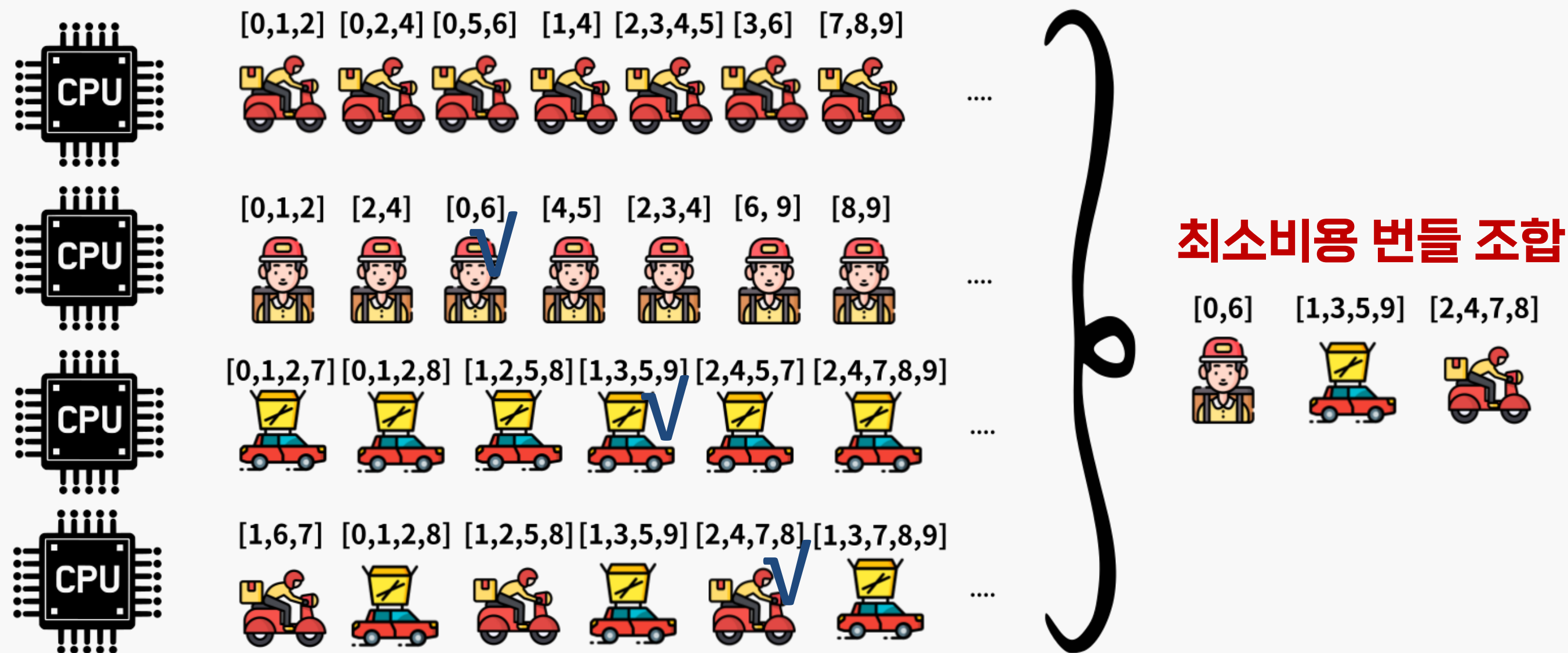
접근법 1. 혼합정수선형계획(Mixed Integer Linear Programming) 모형

- 1 MILP 기반 풀이로 먼저 접근 → 1분 동안 20개의 주문 처리가 불가능
- 2 주문 부분집합 2^N 개 (주문 30개 → 번들 10억 개)
- 3 각 부분집합에 대해 용량 제약, 시간 제약, 방문순서 제약을 모두 만족하는지 확인 필요

접근법 2. Bundle based 모형

- 1 미리 (1)용량 제약, (2)시간 제약, (3) 방문순서 제약을 만족하는 묶음-배달원 번들을 제작
- 2 제작한 번들로 (4) 주문 만족 제약, (5) 배달원 할당 제약을 최소 비용으로 만족하는 최적해 도출

Efficient and cost-effective bundle generation using heuristics and multiprocessing



Multiprocessing
with 4 CPU

Efficient and cost-effective bundle generation

- 어떻게 bundle generation을 efficient하게 할 것인가?
- 어떤 bundle이 cost-effective한 bundle인가?

Minimum Cost Set Partitioning

02 알고리즘 구현 언어

02 알고리즘 구현 언어

예선

- ✓ 대회 기본 언어인 Python으로 구현
- ✓ 1개의 CPU를 사용하는 default Python / 대회는 CPU 사용률 400%까지 허용
→ Python Multiprocessing 모듈을 활용하여 4배 속도로 bundle generation
- ✓ 최적화는 Gurobi-python으로 해결

본선

- ✓ Matrix Calculation 기반 bundle making heuristic 사용을 위해 Numpy 모듈 적극적 사용

결선

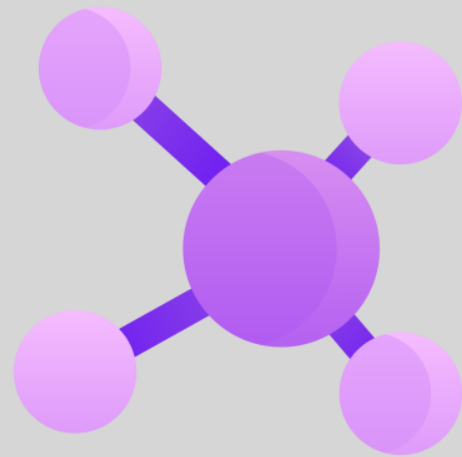
- ✓ Bundle making 속도를 높이기 위해 Cython으로 bundle making code를 C로 컴파일 해서 사용

03 알고리즘 구현 기술

Divergence and Convergence

최종 제출본 알고리즘 아이디어는 간단
발산했던 아이디어의 장점만 채택하여 최종 알고리즘에는 아이디어의 정수만 남김

1



Order Clustering

2



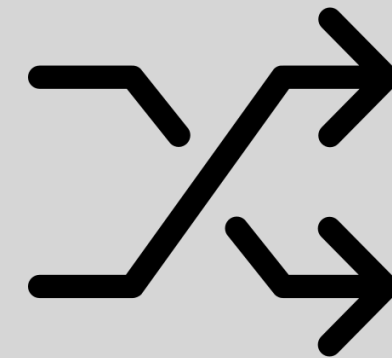
Efficient
Bundle Generation

3



Cost-effective
Bundle Generation

4



N-M heuristic

Order Clustering

- 어떤 주문들을 묶었을 때 Feasible하고 cost-effective할 것인가?
 - Feasible Bundle : 이동 시간을 고려한 시간 제약을 모두 만족
 - Bundle 비용 = 배달원 고정 비용 + 거리 비례 변동 비용
 - 가까운 주문들 묶음 → 주문 간 이동 시간&거리 ↓ , bundle feasibility ↑ , cost ↓
-
- 주문에는 pickup/delivery 두 위치가 존재
 - 가까운 주문이란 무엇인가 - pickup 위치가 가깝고 & delivery 위치가 가까운 주문
 - Pickup 정보로 clustering & delivery 위치로 clustering
→ (pickup cluster, delivery cluster)로 order clustering

Algorithm

INPUT : 모든 오더의 pickup 위치(위도, 경도), delivery 위치(위도, 경도)

K-Means algorithm(K=4)을 활용하여 pickup 위치 clustering

K-Means algorithm(K=4)을 활용하여 delivery 위치 clustering

Cluster Dictionary[pickup cluster, delivery cluster] = [] 생성

For order in all_orders:

 학습된 K-Means algorithm을 활용하여 order의 pickup cluster, delivery cluster 계산

 Cluster Dictionary[pickup cluster, delivery cluster].append(order)

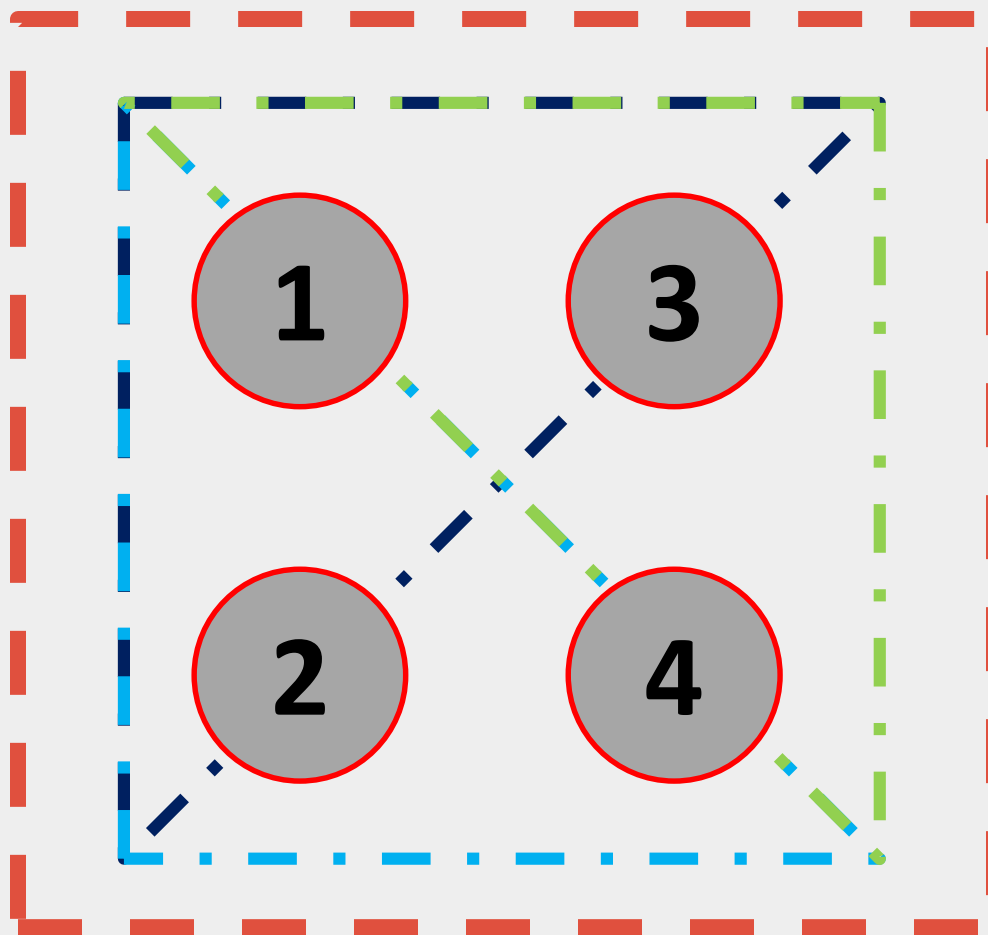
OUTPUT : 16개 cluster로 나뉜 order list

Efficient Bundle Generation – Enumeration

- Order Clustering을 통해 가까운 주문 집합을 얻음
- TODO : 주문 집합을 가지고 만들 수 있는 모든 번들을 빠르게 생성
- Naïve Idea : full enumeration and check
 - 3개의 주문으로 번들을 제작 : $3! \text{ pickup sequence} \times 3! \text{ delivery sequence} = 36$
 - 4개 : $4! \times 4! = 576$
 - 5개 : $5! \times 5! = 14400$
- 30개 주문 집합에서 주문 4개까지의 번들을 모두 만들기 위한 계산량
 - $$\binom{30}{1} * 1 + \binom{30}{2} * 4 + \binom{30}{3} * 36 + \binom{30}{4} * 576 = 1593,3210$$

Efficient Bundle Generation - Subset feasibility check

- KEY IDEA : 주문[1,2,3,4]로 번들을 만들기 위한 필요 조건
 - 주문[1,2,3], 주문[1,2,4], 주문[1,3,4], 주문[2,3,4] 번들 생성 가능
- 활용 : Sequence enumeration 전에, subset check를 먼저 수행
- Enumeration을 수행하는 번들 조합 획기적 감소



Algorithm

INPUT : 주문 집합

Feasibility_set = {}

모든 2개 번들 조합에 대해 optimal bundle 생성

생성 번들에 대해 Feasibility_set.add(orders in bundle)

For (ord1, ord2, ord3) in combination(주문집합, 3):

 if (ord1, ord2) & (ord1, ord3) & (ord2, ord3) in feasibility_set:

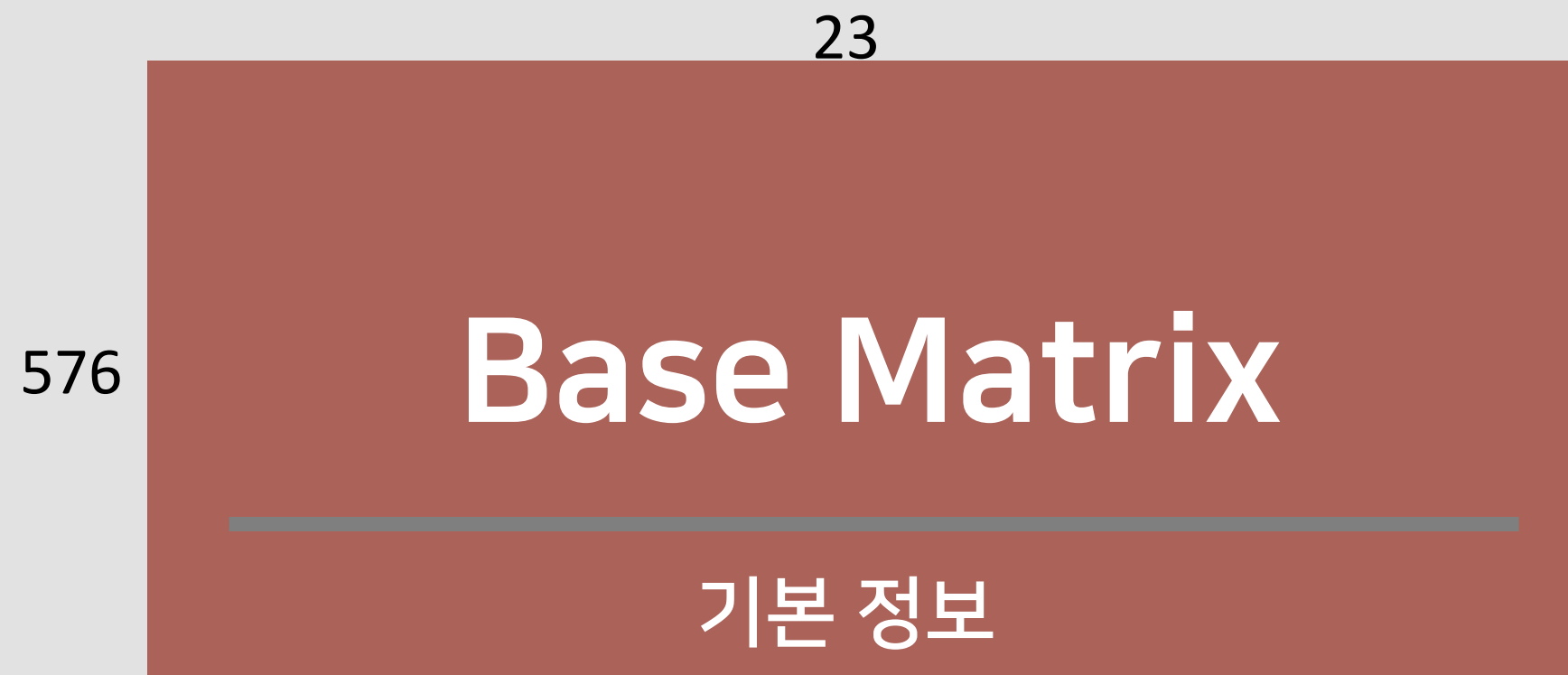
 make optimal bundle with (ord1, ord2, ord3)

...similar way...

OUTPUT : 주문 집합에서 만들 수 있는 모든 4개 이하의 최적 번들

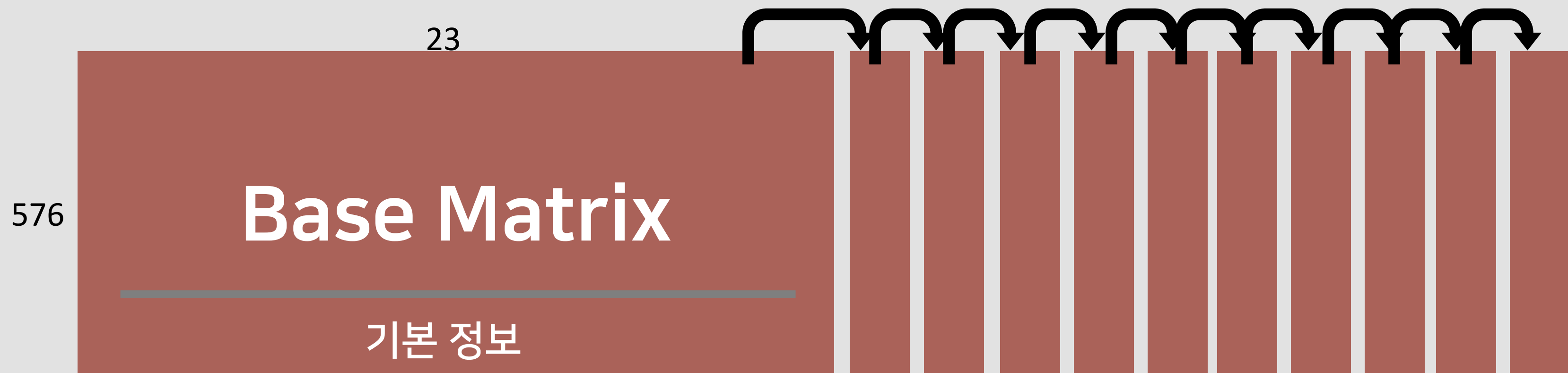
Efficient Bundle Generation – Matrix Calculation

- KEY IDEA : 주문 4개 번들을 만들기 위한 576번 loop를 matrix calculation으로 대체
- Make base matrix with size (576, 23)
 - Row : 576번 enumeration sequence
 - Column : 각 주문 준비 완료 시각, 배달 한계 시각, 배달 위치간 이동시간을 기록



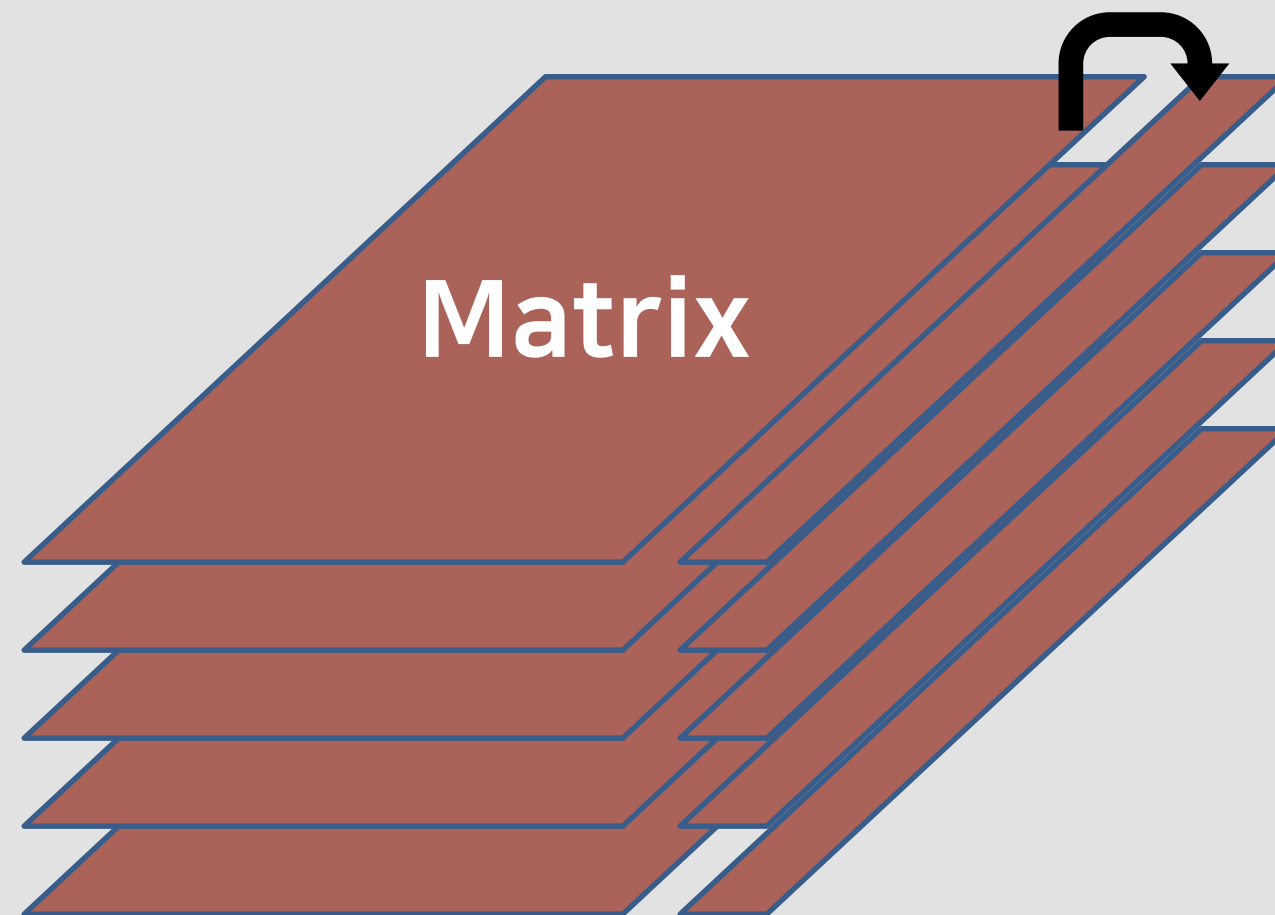
Efficient Bundle Generation - Matrix Calculation

- Pickup sequence : [1,2,3,4] , delivery sequence : [1,2,3,4]
- 주문 2 픽업 완료시간 : $\max(\text{주문 1 픽업 완료 시각} + 1\sim 2 \text{ 이동시간}, \text{주문 2 준비 완료 시각})$
- 주문 3 픽업 완료시간 : $\max(\text{주문 2 픽업 완료 시각} + 2\sim 3 \text{ 이동시간}, \text{주문 3 준비 완료 시각})$
- 주문 1의 배달 완료시간 : $\text{주문 4 픽업 완료 시간} + 4\sim 1 \text{ 이동시간} < \text{주문 1 배달 한계 시각?}$
- 주문 1,2,3,4의 배달 한계 시각을 모두 맞추면 feasible bundle



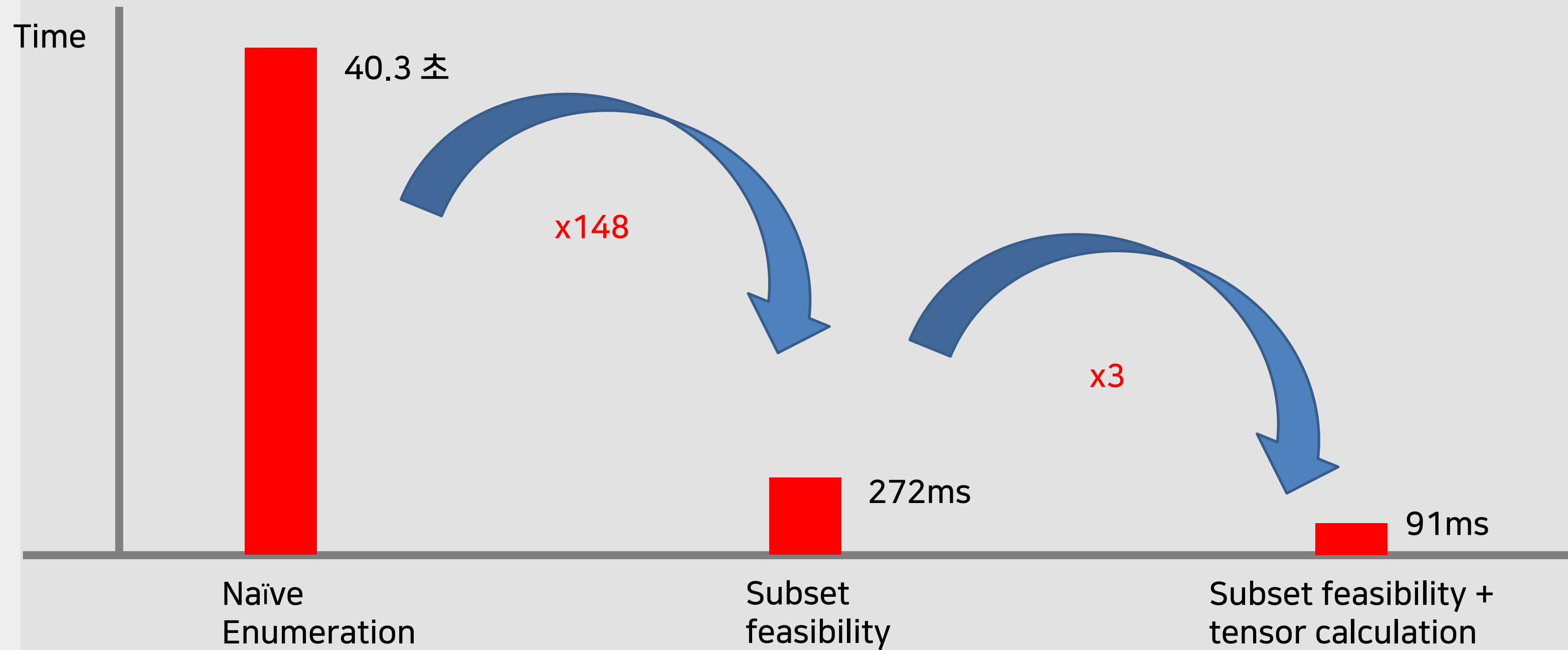
Efficient Bundle Generation - Tensor Calculation

- 576번의 loop가 필요했던 optimal bundle making with 4 orders
→ (576, 30) size matrix calculation
- 여러 주문 조합에 대해서도 할 수 있을까? → matrix를 위로 쌓아 tensor 연산으로 가능



Efficient Bundle Generation - Experiments

- 30개 주문 집합에서 4개까지의 번들 모두 generation

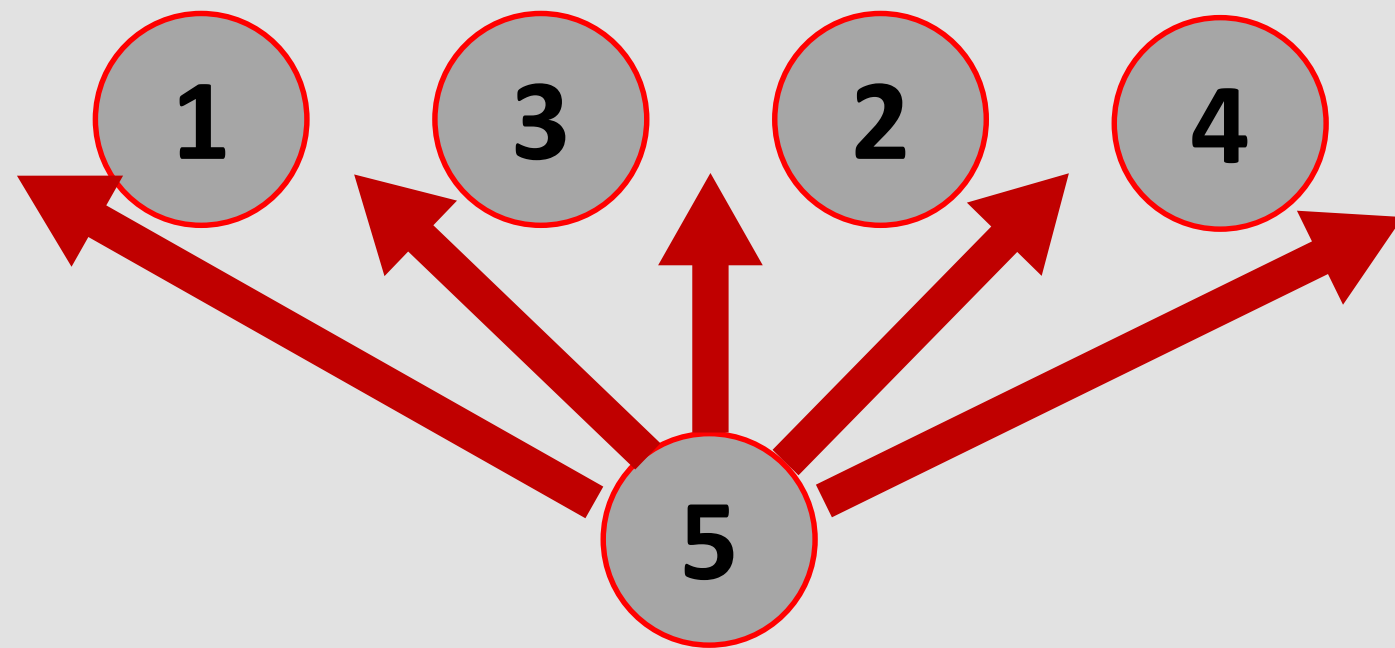


Efficient Bundle Generation - Heuristic for bundle with 5,6,7,8 orders

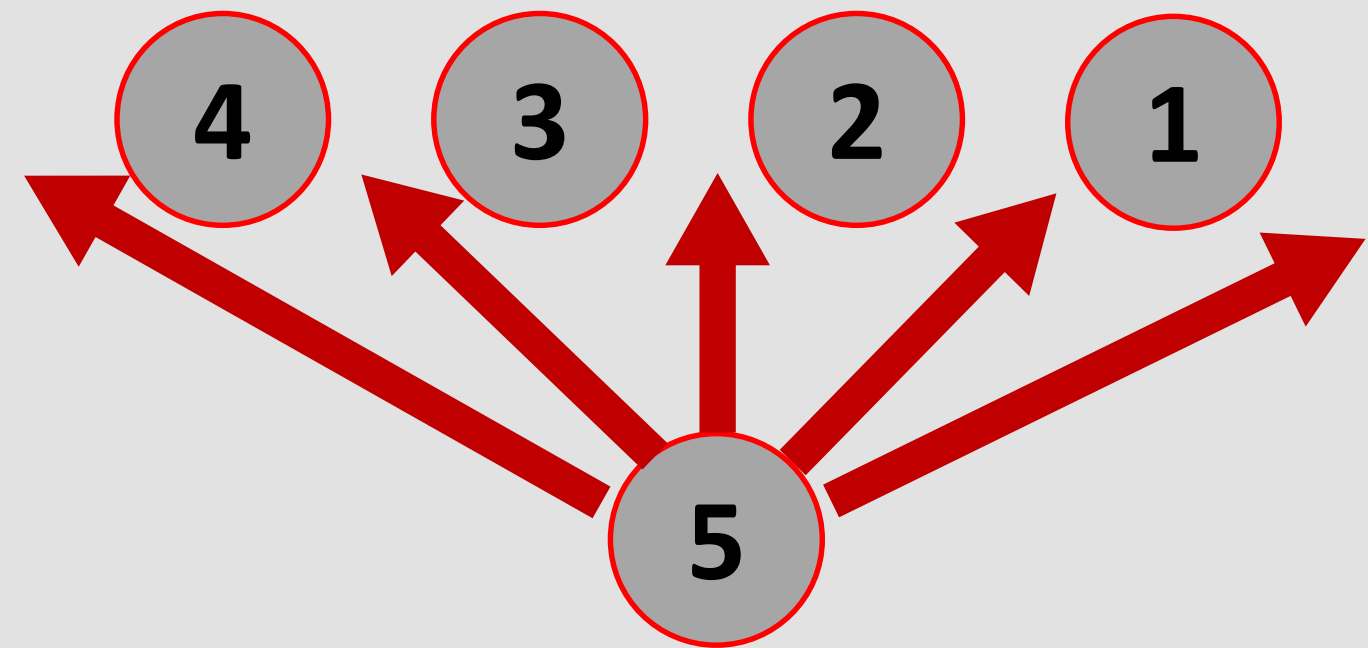
- KEY IDEA

4개 주문 번들에 주문을 하나 더 끼워넣을 때, 기존 optimal sequence를 유지할 것이다.

Optimal pickup sequence



Optimal delivery sequence






- 각 25번, 36번, 49번, 64번만의 연산이 필요하다

Efficient Bundle Generation - Exploit Rider Information

- KEY IDEA

모든 데이터에서 배달원 정보는 일치하며, 배달원 간의 위계가 존재

	속도: 1.3 용량: 70 접근 시간: 120		속도: 4.2 용량: 100 접근 시간: 120		속도: 5.2 용량: 200 접근 시간: 180
--	---------------------------------	--	----------------------------------	--	----------------------------------

- Bike는 Walk에 비해 속도, 용량 ↑, 접근 시간 동일 → 모든 walk 번들은 bike로 변경 가능
- Car는 Walk에 비해 속도, 용량 ↑, 접근 시간 ↑ → 대부분 walk 번들은 car로 변경 가능
- Bike는 Car에 비해 속도 ↑, 접근 시간 ↓ → 용량 100 이하 car 번들 bike 변경 가능
- Car는 Bike에 비해 속도, 용량 ↑, 접근 시간 ↑ → bike 번들은 높은 확률로 car 변경 가능

Cost Effective Bundle Generation

- '가까운' 주문 집합은 cost-effective bundle 생성
- '가까운' 주문 집합을 위한 Order Clustering : even한 분배 X, 주문 수 높을 때 X
- KEY IDEA : 주문 간의 거리를 정의
- Distance(오더1, 오더2)
 - = (오더1, 오더2의 픽업 위치 거리)
 - + (오더1, 오더2의 딜리버리 위치 거리)
 - + 0.05*(오더1의 픽업 위치, 오더2의 딜리버리 위치 거리)
 - + 0.05*(오더1의 딜리버리 위치, 오더2의 픽업 위치 거리)

Cost Effective Bundle Generation

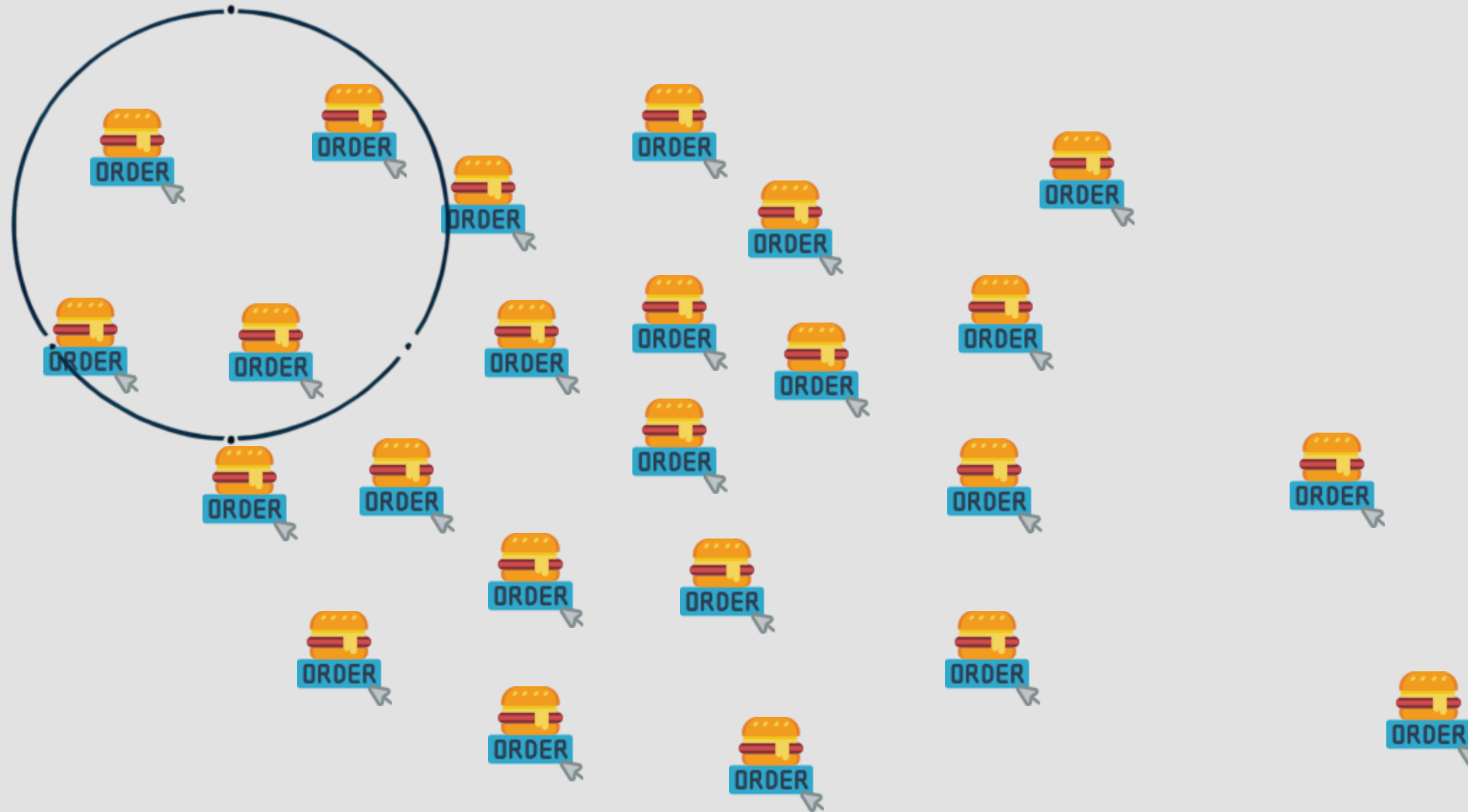
접근법 1 : 거리 기반 Clustering

- 이제 주문은 좌표가 아닌 서로 간의 거리로 표현 가능
- 인스턴스 간 거리가 주어졌을 때 균일한 클러스터링이 가능한 ML Technique
Multidimensional scaling(MDS) 사용

접근법 2 : Nearest Neighbor

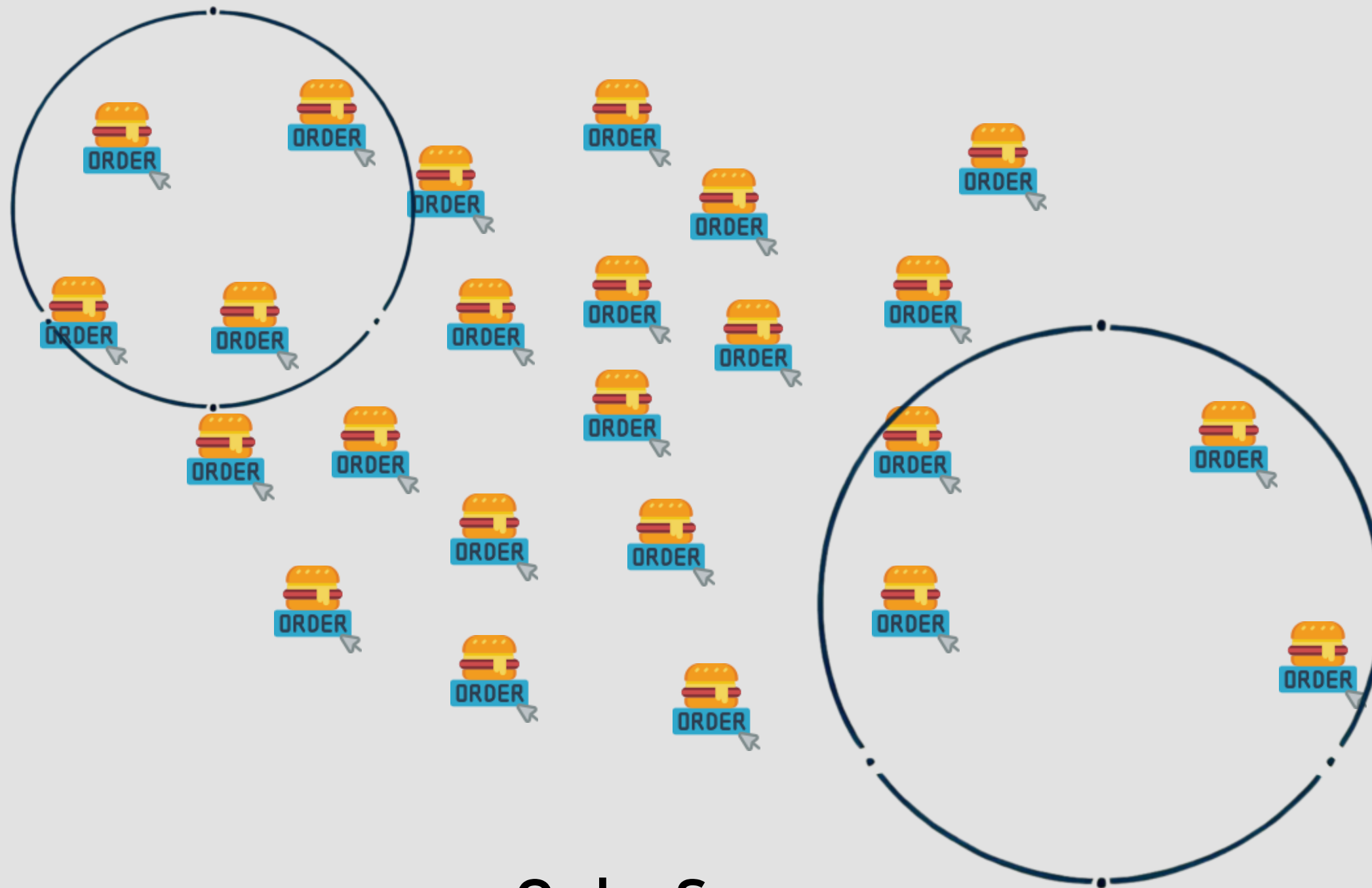
- 한 주문을 선택해서 가장 가까운 N개를 선정 → N개의 주문 집합 생성 반복
- 모든 주문에 균일하게 번들이 만들어 질 수 있도록 count_list 생성, minimum count 주문 선택

Cost Effective Bundle Generation - Nearest Neighbor

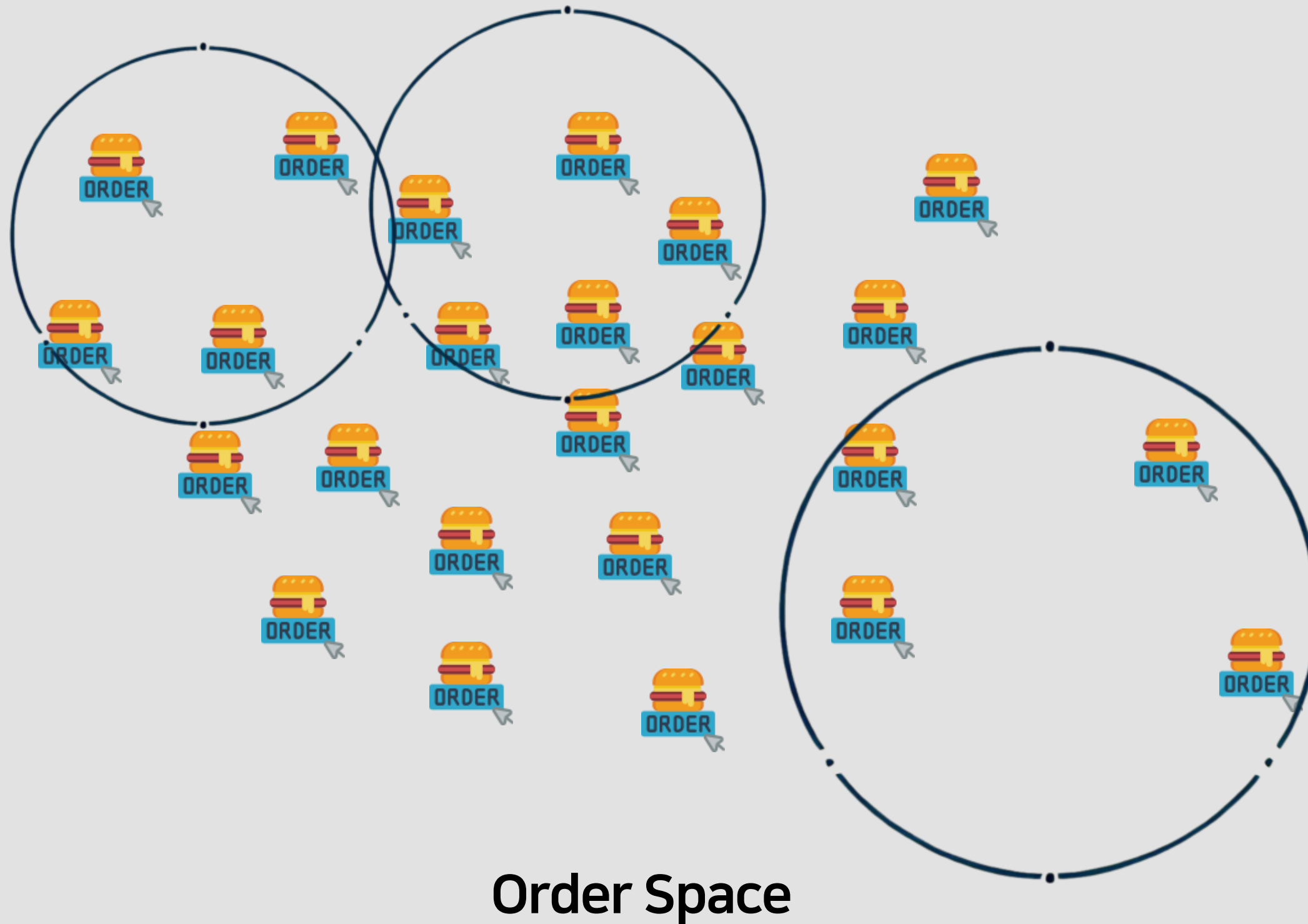


Order Space

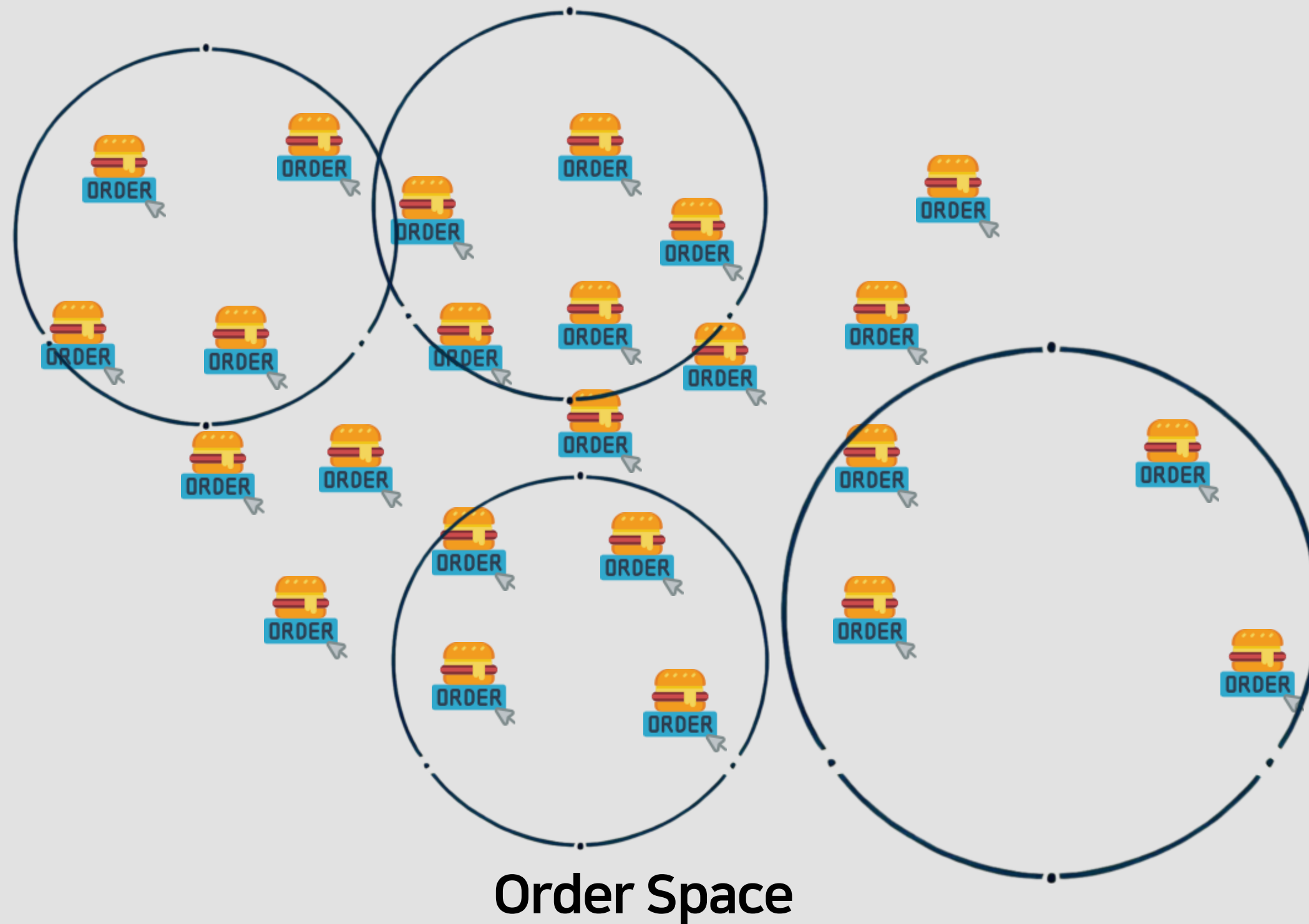
Cost Effective Bundle Generation - Nearest Neighbor



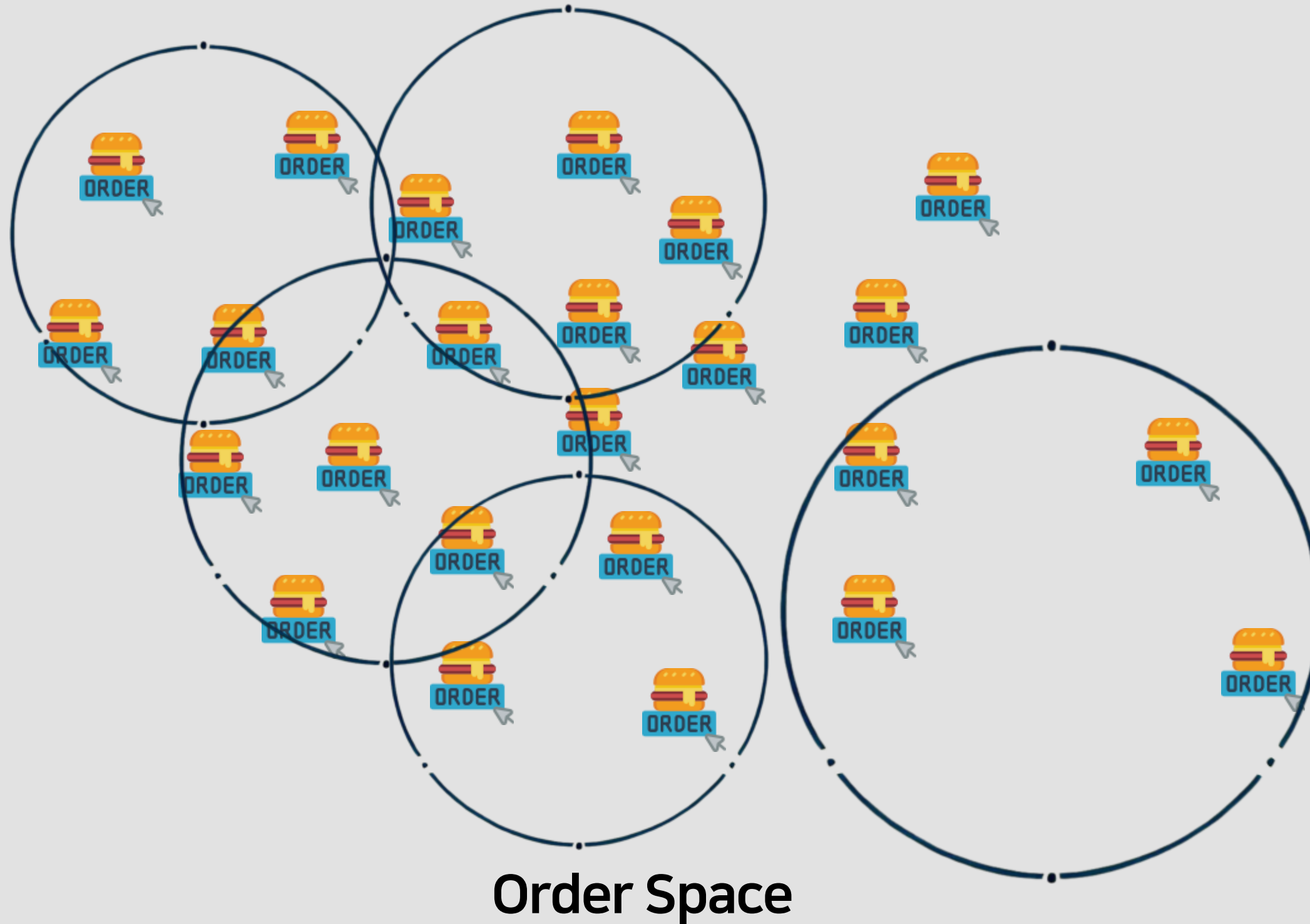
Cost Effective Bundle Generation - Nearest Neighbor



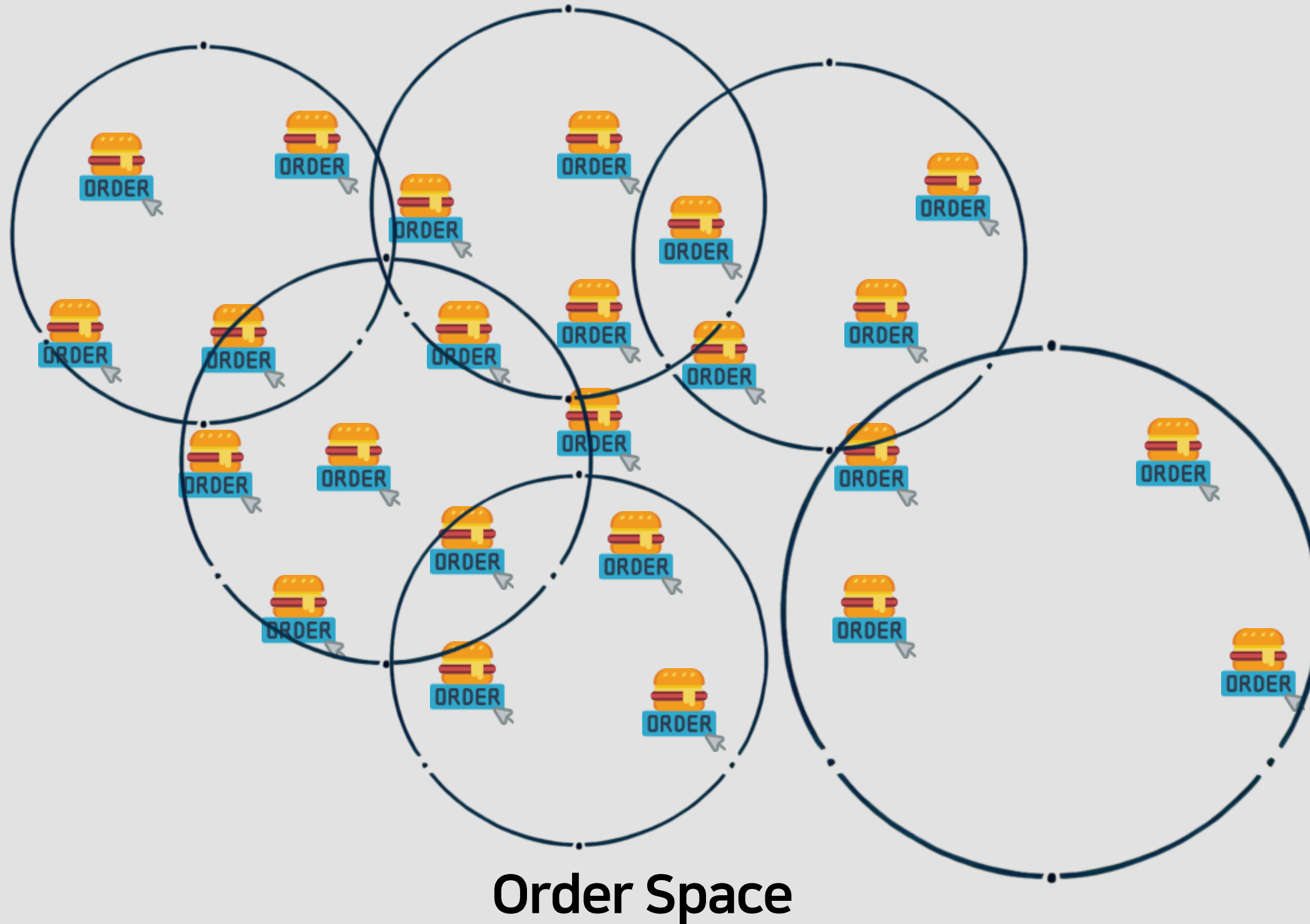
Cost Effective Bundle Generation - Nearest Neighbor



Cost Effective Bundle Generation - Nearest Neighbor



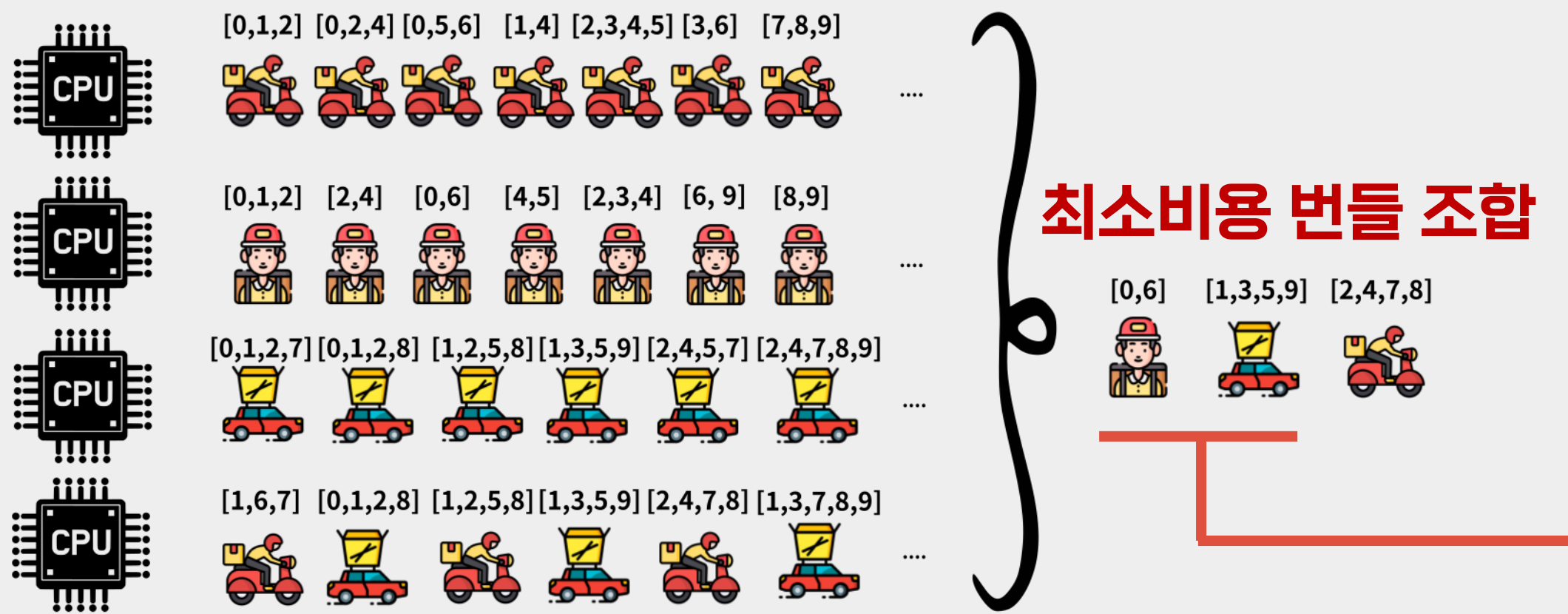
Cost Effective Bundle Generation - Nearest Neighbor



NM Heuristic

- 번들 생성 → Minimum Cost Set Partitioning을 통해 근사 최적해를 구함
- KEY IDEA : 해를 개선해보자
 - 번들 1개를 더 좋은 번들 1개로 대체(X)
 - 번들 2개를 합쳐서 번들 1개로 대체(X)
 - 번들 N개를 합쳐서 번들 M개로 대체
- 가지고 있는 강력한 무기 : 주문 집합이 주어졌을 때, make every profitable bundles

NM Heuristic



- 최적해의 번들 N개를 다른 번들 M개로 대체
- 새로운 주문집합 [0,1,3,5,6,9]를 활용하여 new bundle generation
- [0,1,3,5,6,9]를 Cover하는 small size minimum set partitioning problem 풀이 후 최적 번들 대체

Iterative Improvement Algorithm

04 알고리즘 특징점 및 발전 방향

알고리즘 특징점 및 발전 방향

POINT 01	<div>Universal Algorithm</div> <div>주어진 데이터에 대해 알고리즘을 나누지 않음, Hyperparameter 2개를 조절하면서 대응</div>
POINT 02	<div>Hyperparameter Optimization</div> <div>실무에서 요구되는 계산 한계 시간과 데이터의 형태를 미리 아는 상황이라면 파라미터를 미리 최적화하여 성능을 훨씬 올릴 수 있을 것</div>
POINT 03	<div>Efficient Bundle Generation</div> <div>주문 집합에 대해 모든 bundle을 생성하는 알고리즘의 성능은 가장 우수할 것이라 생각함, 다른 팀의 알고리즘에도 쓰일 수 있는 부분이 많을 것이라 예상, CPU 수가 늘어나거나 Matrix Calculation에 유리한 GPU활용 방안을 고안한다면 더 발전할 수도 있을 것</div>
POINT 04	<div>Algorithm for sparse data, NM Heuristic</div> <div>주어진 데이터 중 특징적으로 위치 간 거리가 먼 데이터들이 있었음, 이를 위한 알고리즘은 따로 개발하는 것이 훨씬 성능적으로 좋을 것이라 예상, NM heuristic의 발전 가능성이 있어보임</div>

THANK YOU FOR ATTENTION

TEAM 붉은달

박준석

이민희

최의현