**2021 1학기 연구참여 보고서**

*Pickup and Delivery Problem*

*with time windows*

2021. 02. 22 ~ 2021. 06. 21

Logistic Lab

지도 교수 : 김병인

지도 선배 : 김현준

참여 학생 : 김채영

목차

내용

[목차 2](#_Toc76979123)

[1. Introduction 3](#_Toc76979124)

[1.1 Objectives 3](#_Toc76979125)

[2. Method 4](#_Toc76979126)

[2.1 Mathematical Model 4](#_Toc76979127)

[2.1.1 Sets 4](#_Toc76979128)

[2.1.2 Parameters 4](#_Toc76979129)

[2.1.3 Decision Variables 4](#_Toc76979130)

[2.1.4 Mathematical Model 4](#_Toc76979131)

[2.2 Heuristic Algorithm 5](#_Toc76979132)

[2.2.1 About Problem 5](#_Toc76979133)

[2.2.2 Program Structure 6](#_Toc76979134)

[2.2.3 Algorithm 7](#_Toc76979135)

[2.3 Visualization 9](#_Toc76979136)

[2.3.1 Library 9](#_Toc76979137)

[2.3.2 Program Code 10](#_Toc76979138)

[3. Result 11](#_Toc76979139)

[4. Discussion 12](#_Toc76979140)

[4.1 Optimize taxi driving strategies based on reinforcement learning 12](#_Toc76979141)

[5. Conclusion 14](#_Toc76979142)

[6. Review 15](#_Toc76979143)

[7. Reference 16](#_Toc76979144)

# 1. Introduction

## 1.1 Objectives

일반적인 Vehicle Routing Problem (VRP)은 depot에서 출발한 차량이 node들을 모두 방문하고 돌아오는 데 소요되는 이동거리를 최소화하는 vehicle route를 결정하는 문제이다. 하지만 Pickup and Delivery with Time Window Problem (PDPTW)의 경우, Node들이 2가지 type으로 구성된다. 하나의 request는 Pickup Node와 Delivery Node로 구성되며, 항상 Pickup Node가 Delivery Node보다 선행되어야 한다. 그리고, 같은 request의 Pickup Node와 Delivery Node는 같은 Route 안에 있어야 한다. Time Window는 각 Node마다 지정된 시간대로, 그 시간대 사이에 vehicle이 방문해야 하며, 특정 Node의 Time Window 내에 방문하지 못할 경우, reject을 하거나, penalty가 주어지게 된다.

이러한 PDPTW 문제는 일상 생활에서도 밀접한 연관성이 있다. 정해진 route가 있는 셔틀버스가 아닌, 그 때의 승객의 수요나 요구에 맞춰 셔틀버스를 효율적으로 운영하기 위해서는 PDPTW 문제를 해결함으로써 최적의 셔틀버스 route를 구할 수 있다. 추가적으로 최근 코로나 19로 인해 직접 방문하는 것보다 비대면으로 배달이나 배송을 이용하는 빈도가 늘어나고 있다. PDPTW 문제를 배달에 적용시킨다면, 동일한 시간에 더 많은 배송을 할 수 있어 효율성을 높이고, route를 최적화하여 배달료 인하 효과를 기대할 수 있다.

따라서, 본 연구에서는 PDPTW 문제를 해결하기 위한 heuristic algorithm을 고안하고, 알고리즘의 결과를 지도상에 시간에 따른 vehicle의 route를 시각화하는 것을 목표로 한다.

# 2. Method

## 2.1 Mathematical Model

### 2.1.1 Sets

Set of vehicles

Set of pickup locations,

Set of delivery locations,

Set of all locations,

### 2.1.2 Parameters

time to start service

0 location of depot

travel time between location *i* and location *j*,

opening time of time window at location *i*,

closing time of time window at location *i*,

*M* Big-M constant

### 2.1.3 Decision Variables

1, if the vehicle *k* moves from location *i* to location *j*; 0, otherwise,

1, if location *i* is serviced by vehicle *k*; 0, otherwise,

the arrival time of vehicle *k* at location *i*

the departure time of vehicle *k* at location *i*

### 2.1.4 Mathematical Model

- Objective Function

Total travel time을 최소화한다.

- Constraints

(1) 모든 Vehicle은 Depot에서 출발한다.

(2) Pickup node는 Delivery node보다 같은 vehicle로 선행되어 방문해야 한다.

(3) 모든 Node들에 최대 1번 방문해야 한다. (2번 이상 방문할 필요 없음)

,

(4) Node i에서 다른 Node들로 이동한다면, Node i는 이미 Serve되었다.

(5) Node i에 도착하는 시각은 Serve후, 떠나는 시각보다 빨라야 한다.

(6) Vehicle이 Depot을 출발하는 시각은 전체 Service가 시작하는 시각 이후여야 한다.

(7) Node i에서 출발한 시각에 Node i에서 j까지 이동할 때 걸린 시간을 더한 값이,  
Node j에 도착한 시각보다 짧아야 한다. (만약 Node i에서 j까지 이동하지 않는다면 ( 이라면, Big M으로 제약 조건이 없어진다.)

(8) Node i에 도착한 시각은 Node i의 time window 사이에 있어야 한다.

(9) Node i에서 j까지 이동했다면, Node i와 Node j는 Serve되었다.

(10) Variable 는 binary variable이다.

## 2.2 Heuristic Algorithm

### 2.2.1 About Problem

Pickup과 Delivery로 구성된 Request가 들어올 때, Pickup Node와 Delivery Node를 어떤 Route에 넣어야 하는지, 그 Route에서는 어떤 Position에 각각 Pickup Node와 Delivery Node를 위치시켜야 하는지 구하는 문제이다. Figure 1에서 확인할 수 있듯이, Route의 개수는 Vehicle의 개수로 미리 정해지며, 모든 Vehicle은 depot에서 출발하여, depot으로 도착하도록 설정되어 있다.

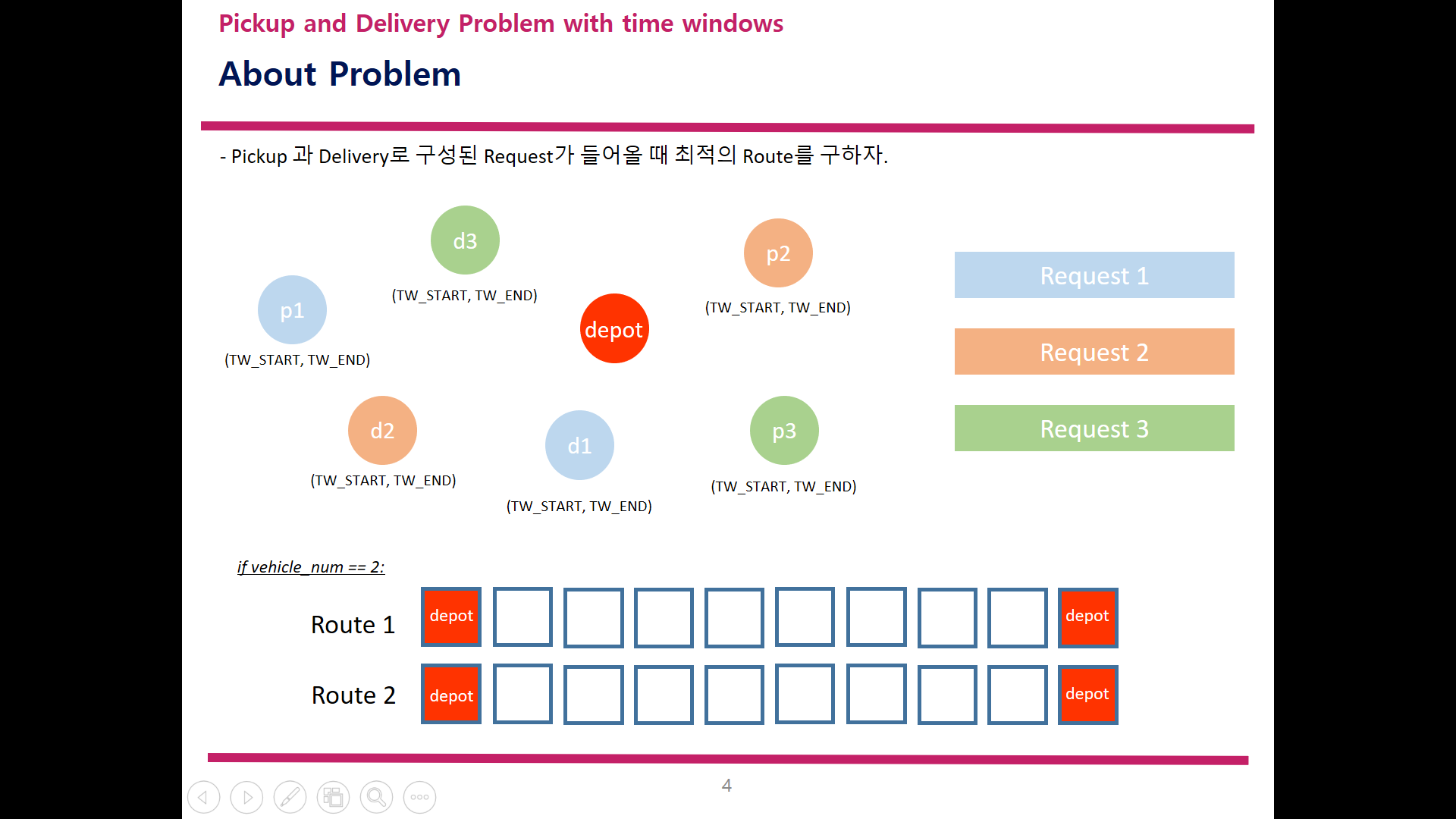


Figure 1. Problem Setting

### 2.2.2 Program Structure

프로그램의 input file은 request file과 parameter file로 구성되어 있다. Request file에는 request가 들어온 순서, request index, 그리고 Pickup Node, Delivery Node 각각의 Time window start와 end 값, 노드의 위경도 값을 받게 된다. Parameter file에는 전체 vehicle 개수를 받아 몇 개의 route를 생성할지 결정하게 된다. 거리를 계산한 후, 시간으로 변환하기 위해 평균 속도에 대한 값도 받게 된다. 또한, depot의 time window와 위경도 값도 입력 받는다.

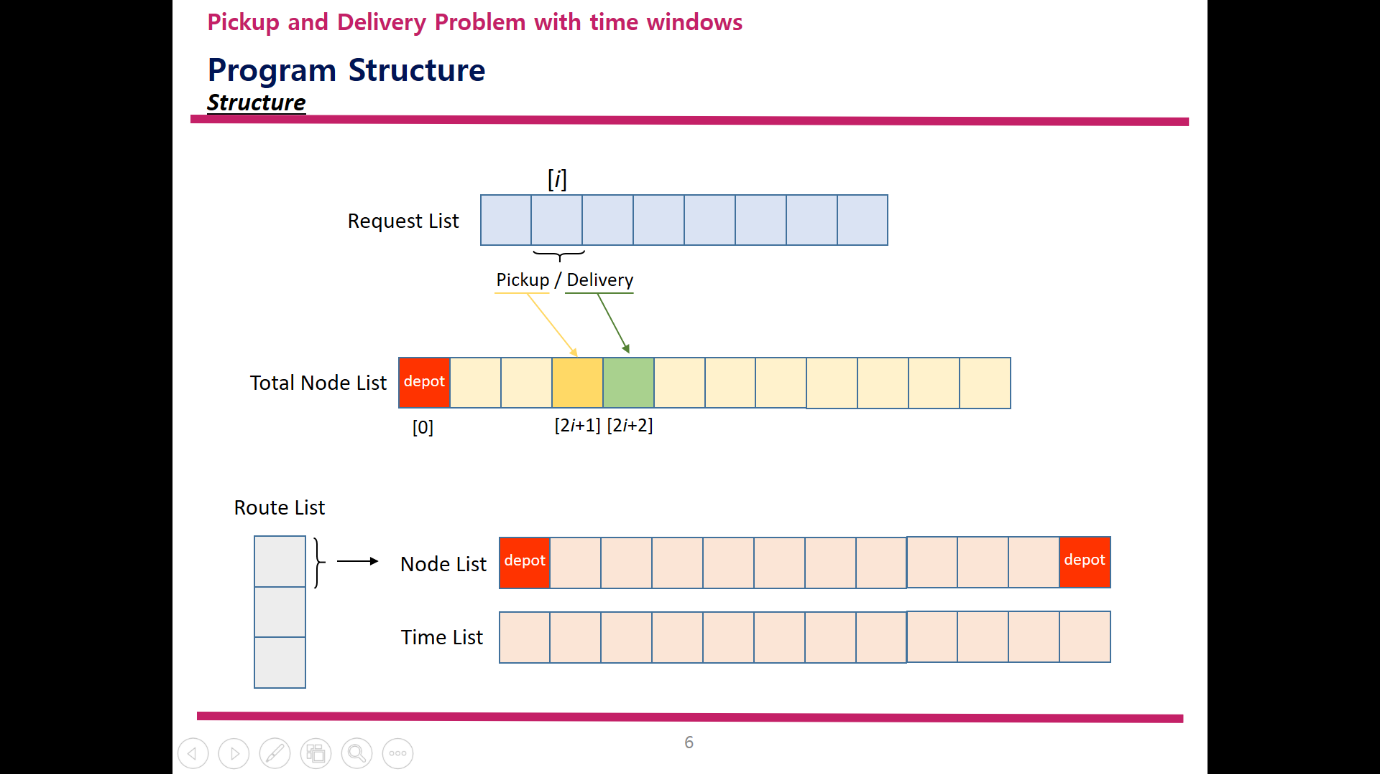


Figure 2. Program Structure

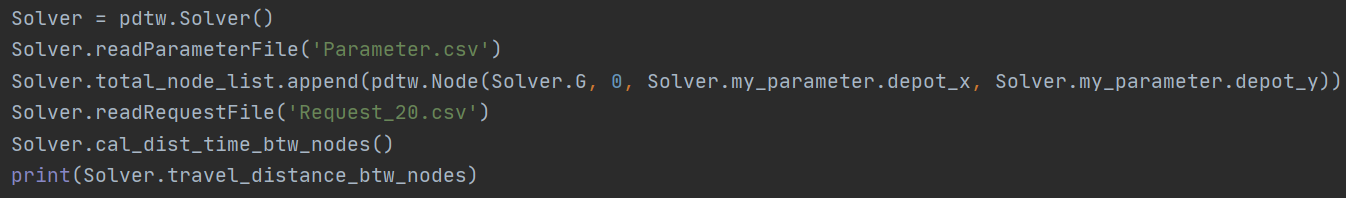


Figure 3. Input Program Code

먼저, request file을 받아 하나씩 request list에 넣는다. Request element에는 Pickup Node에 대한 정보와 delivery node에 대한 정보가 같이 있기 때문에 Total Node List를 선언하여 노드 별 index와 위경도 값을 저장한다. Figure 2과 같이, Depot은 total node list의 0번째 index이고, request i의 Pickup Node는 2i+1번째에, Delivery Node는 2i+2번째에 저장한다. Route List는 vehicle 개수에 따라 몇 개의 Route이 생성되는지 결정되게 되고, 각 Route마다 Node List와 Time List를 갖고 있다. 모든 Vehicle은 depot에서 출발하여, depot으로 도착하여야 하므로, 각 Route 내 Node\_list의 처음과 끝은 depot으로 설정한다. 최종 결과는 Route list에 저장되는데, 각 Route 마다 Node list에는 방문하는 순서대로 node의 index가 저장되고, Time list에는 각 node에 도착 시각이 저장된다.

### 2.2.3 Algorithm

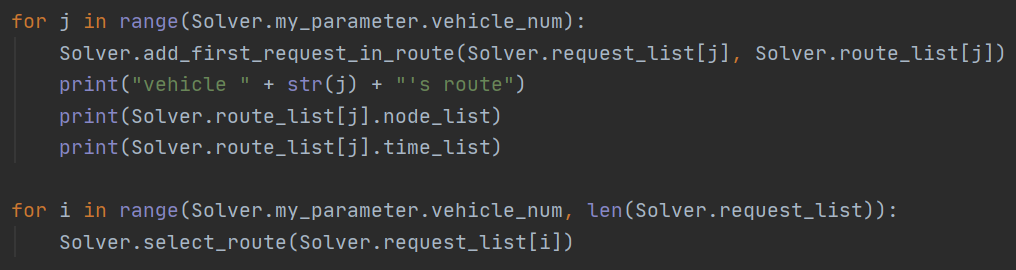


Figure 4. Main function Code

Figure 4의 Code를 보면, 첫번째 for문에서는 해당 Route에 아무 Node도 배정되어 있지 않을 때 추가하는 것이고, 두번째 for문에서는 특정 Request에 대해서 어떤 Route를 선택하게 되는지에 대한 것이다.

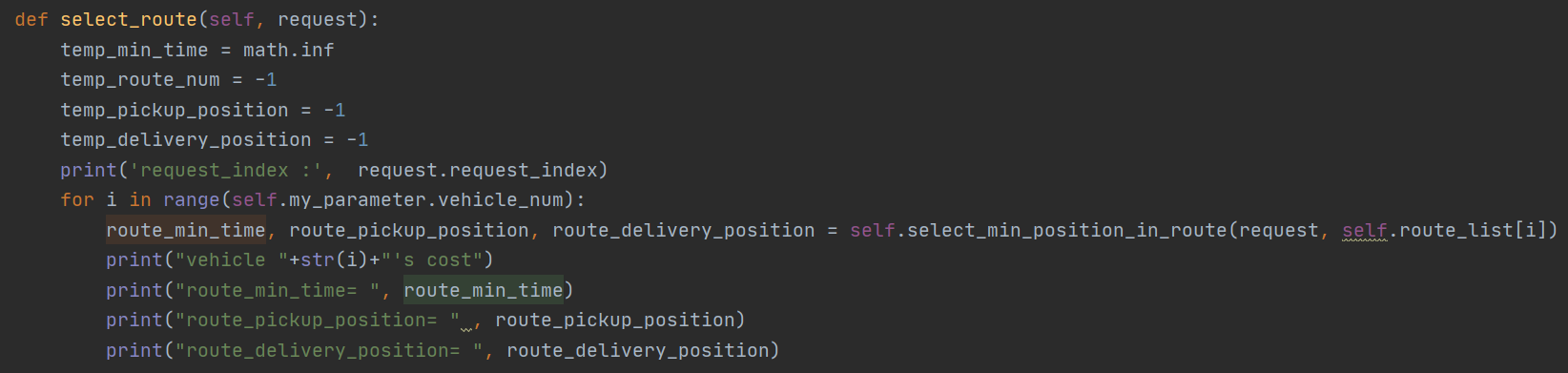


Figure 5. select\_route function – select min position in route

Figure 5는 Figure 4의 두번째 for문의 select\_route 함수에 대한 것이다. 첫 번째로 Request마다 각 Route에서 Pickup Node와 Delivery Node가 어디에 위치하는 것이 최적인지 구하고, 그 Route에서 해당 Request가 들어온 후 늘어난 시간을 계산한다. 해당 route에서 최적인 위치와 Route 종료 시간을 계산하는 함수가 select\_min\_position\_in\_route이고, return 값으로 route\_min\_time, route\_pickup\_position, route\_delivery\_position을 받게된다.

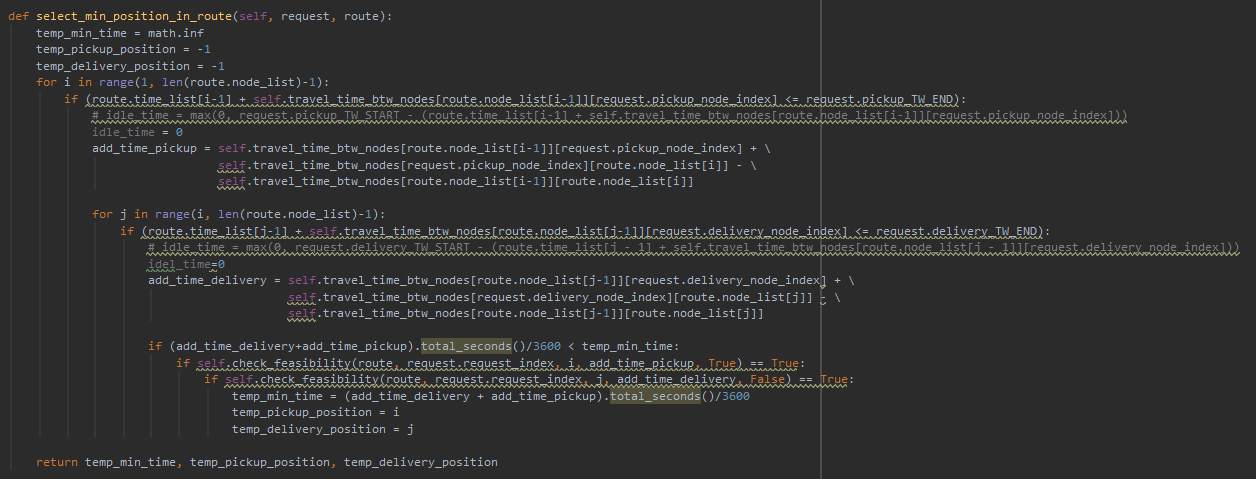


Figure 6. select\_min\_position\_in\_route function

특정 Route에서 Request는 어디에 위치해야 하는지 계산하는 function은 Figure 6의 select\_min\_position\_in\_route function이다. Delivery Node는 Pickup Node과 같은 방법으로 위치를 선정하지만, Delivery Node의 경우 항상 Pickup Node 뒤에 위치해야 하기 때문에 Pickup position이 정한 이후에 pickup position 부터 node\_list의 끝까지 훑으면서 delivery node의 position을 정하게 된다. 하나의 request가 들어왔을 때 해당 route의 add time은 pickup add time과 delivery add time을 합한 값이 된다.

이후에 Figure 7과 같이 하나의 Request에 대해서 모든 Route에 대해 늘어난 시간을 계산한 것을 바탕으로, 가장 적게 늘어난 Route를 선택하고, 그 Route에 배정하게 된다.

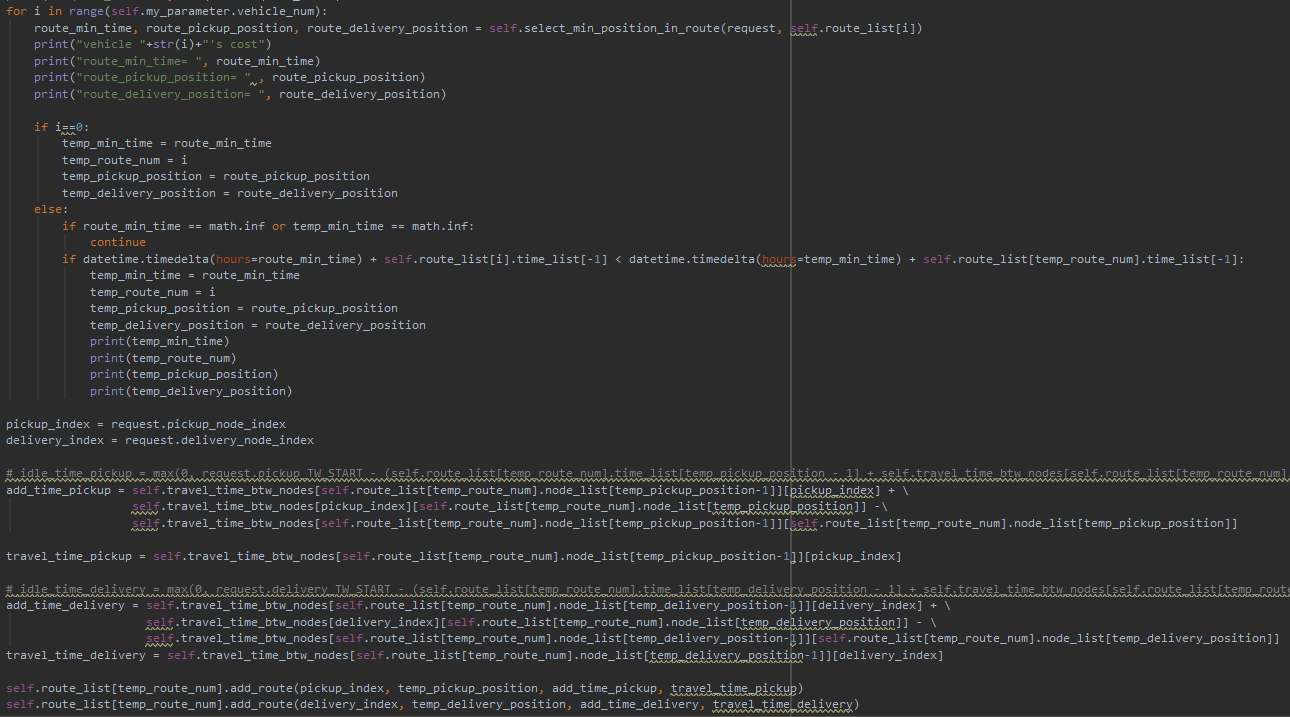


Figure 7. select\_route function – select route

## 2.3 Visualization

|  |  |
| --- | --- |
| Library 명 | 기능 |
| OSMnx | Open Street Map의 거리 네트워크 데이터 다운로드 및 네트워크 분석 |
| Folium | 지리 데이터 시각화, Timestamp로 시간에 따른 위치 표시 기능 |

### 2.3.1 Library

Table 1. 사용한 Library 목록

Table 1은 시각화를 위해 사용한 Library를 정리한 표이다. OSMnx와 Folium를 사용했는데, OSMnx는 Open Street Map의 거리 네트워크를 바탕으로, 최단거리 계산 등 여러가지 분석을 할 수 있다. Folium도 Open Street Map과 같은 지도데이터에 Leaflet.js를 이용하여 시각화 할 수 있다.

### 2.3.2 Program Code

Figure 8. plot\_on\_map function Code

지도 상에 시간에 따른 route를 나타내기 위한 코드는 위의 Figure 8과 같다. 먼저 Depot의 위치를 기준으로 Folium 상에서 지도를 갖고 온다. Lines는 route가 시간에 따라 어디로 이동하는지를 나타낸다. Lines는 Line의 list로 구성되어 있는데, line은 coordinates(출발 위도, 출발 경도, 도착 위도, 도착 경도), dates(출발 시각, 도착 시각), color, weight로 구성되어 있다.

따라서 add\_path에 따라 Route내의 Node List 순서대로 line을 설정하여 lines list에 추가하게 된다. 이후에 map 상에 path를 어떻게 나타낼 것인지에 대한 feature를 설정하고, timestamp가 지날 때마다 Map 상에 띄워주게 된다.

# 3. Result

Heuristic Algorithm으로 구한 최적의 Route는 Figure 9과 같다.

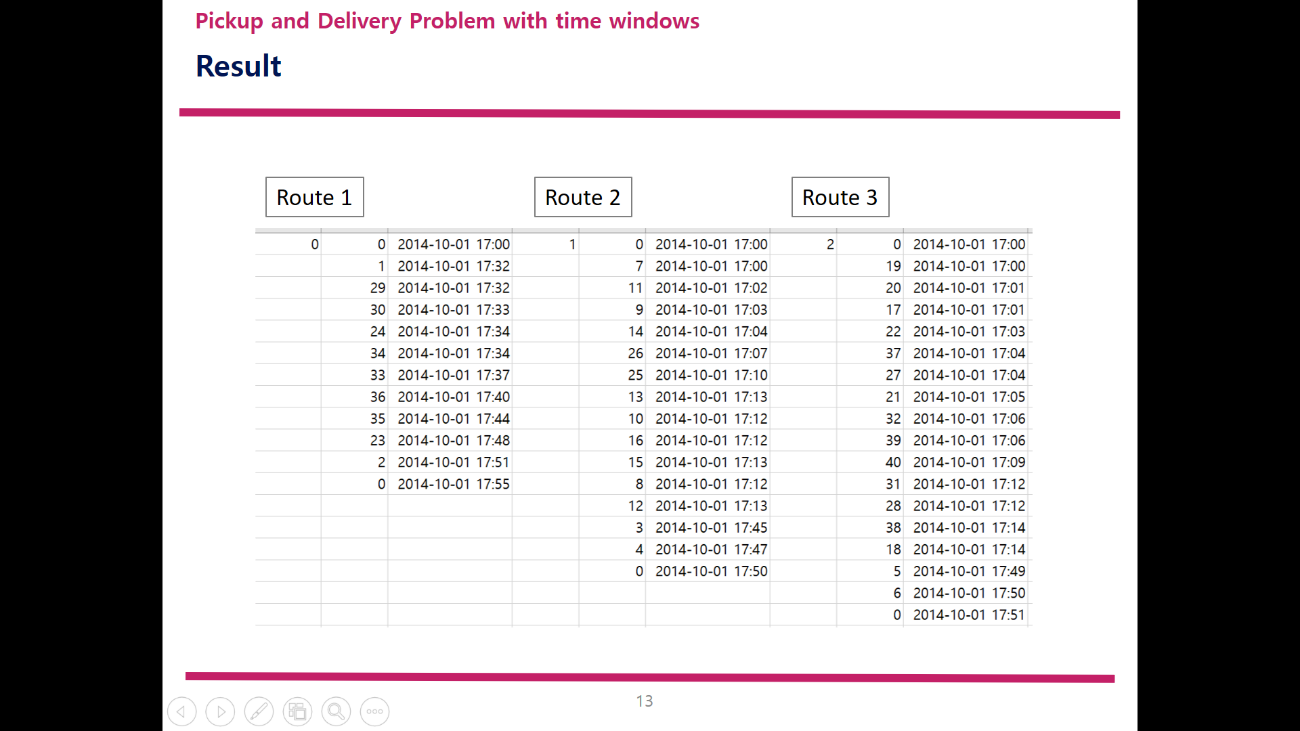


Figure 9. Heuristic Result

총 3개의 vehicle이 있다고 설정했기 때문에 Route는 3가지로 나오고, 각 Route마다 Total Node List의 index로 방문하는 Node의 순서를 알 수 있으며, 각 Node에 도착하는 시각도 그 옆 행에 출력하도록 했다. Route마다 다시 Depot으로 돌아오는 시각이 비슷한 것으로 보았을 때, Node들이 비교적 잘 배치가 되었음을 알 수 있다.

각 Route 별 결과를 지도 상에 띄웠을 때는 Figure 10과 같다.

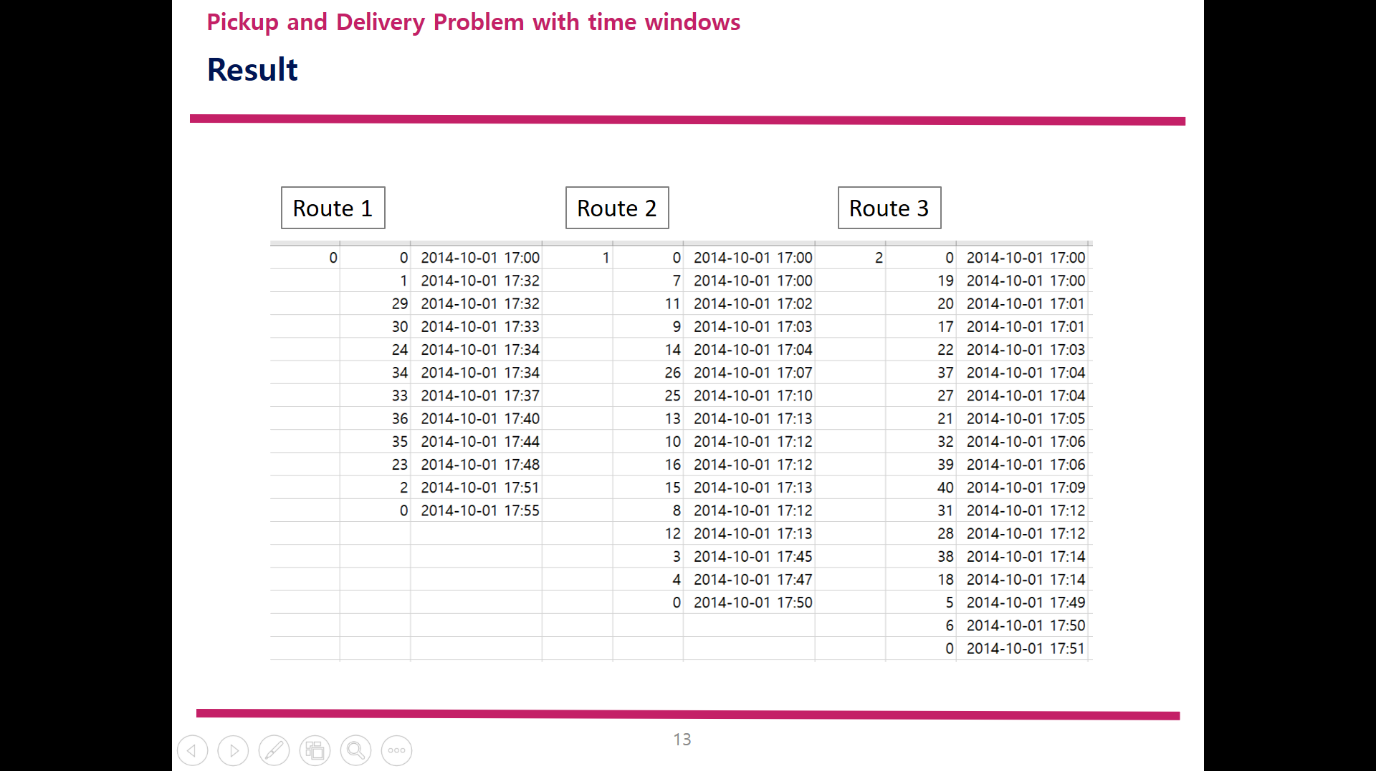
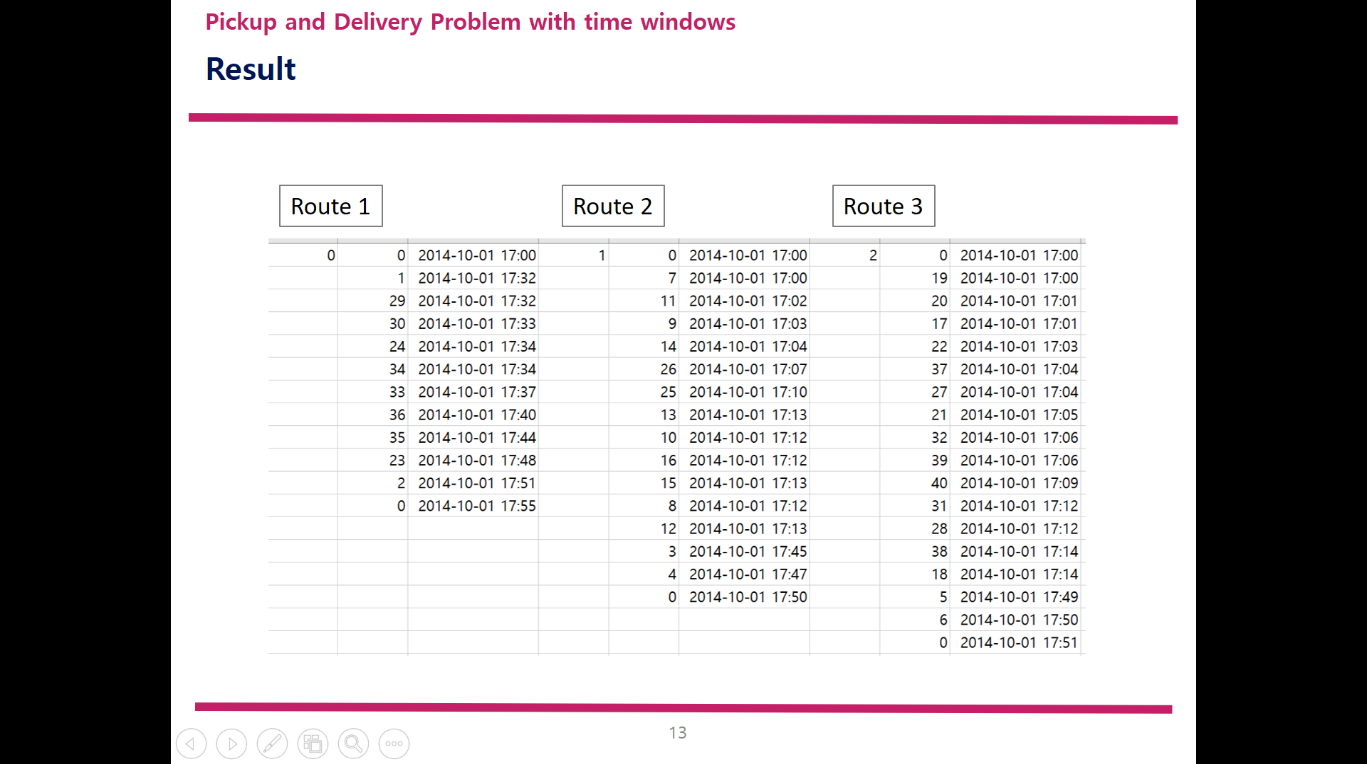
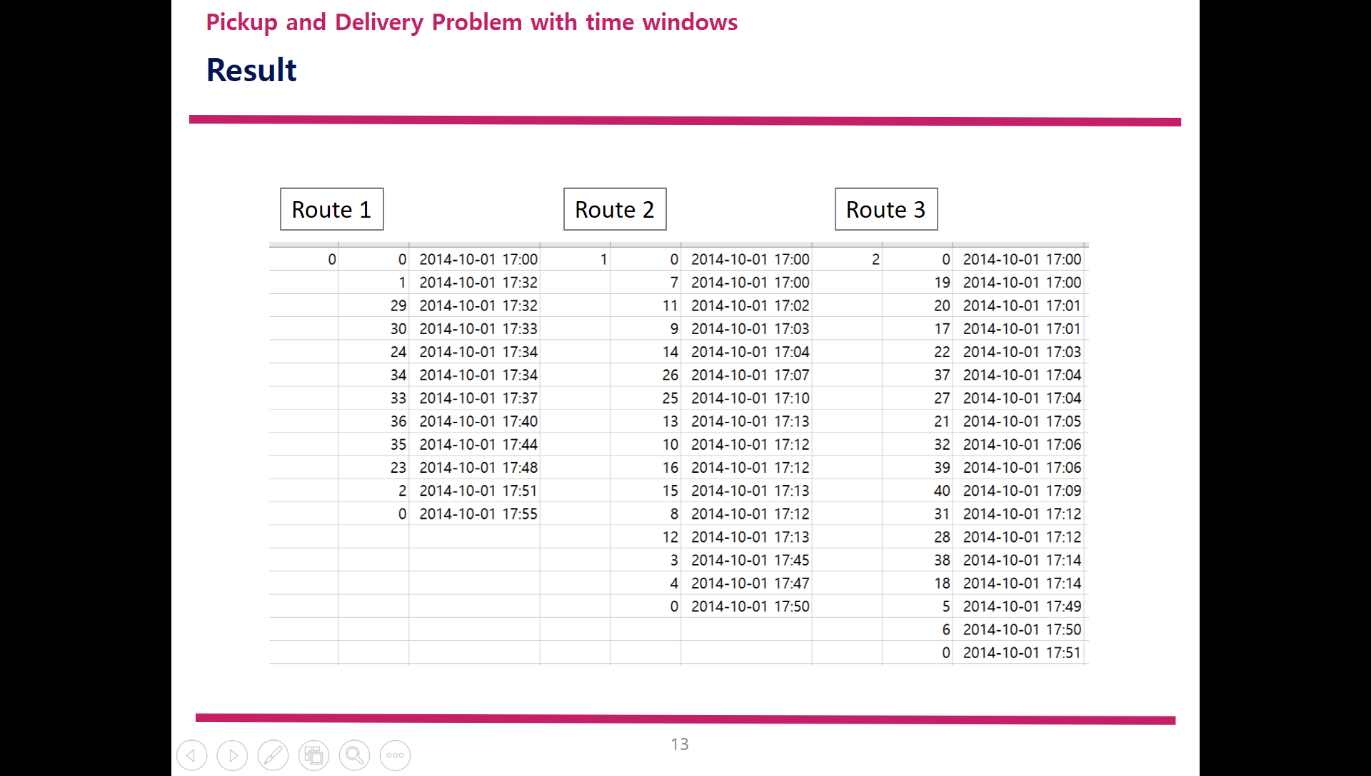
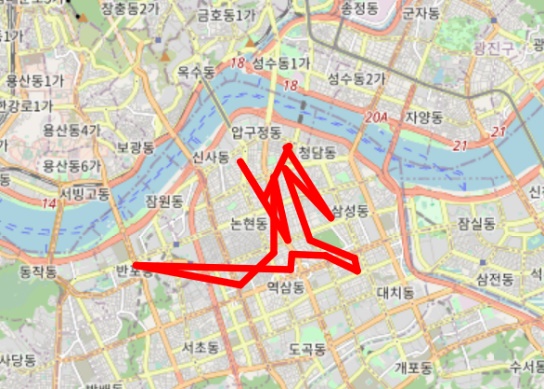


Figure 10. Results on the map

각 Route를 지도 상에서 보면, Path가 교차되어 있는 것도 있지만 Pickup Node 와 Delivery Node의 선행 관계와 각 Node 별 Time Window로 인해 교차가 필수적인 경우도 있다고 추측할 수 있다. 추가적으로 feasibility를 유지하면서 2-OPT 알고리즘을 적용시킨다면 더 좋은 결과를 얻을 수 있을 것이라 기대한다.

# 4. Discussion

조합 최적화 문제 (Combinatorial Optimization)에 일반적인 최적화 방법론인 수리 모델링, 휴리스틱, 메타 휴리스틱 방법론 외, 강화 학습 (Reinforcement Learning) 을 결합한 연구가 주목을 받고 있다. 본 연구의 문제에 강화 학습을 적용하기 위해, 기존 연구 중 택시 운행 전략 (taxi driving strategies) 에 강화 학습을 적용한 논문에 대해 소개하고자 한다.

## 4.1 Optimize taxi driving strategies based on reinforcement learning

택시 서비스는 공급 측면에서도 비어 있는 택시들이 많고, 수요 측면에서도 짧은 거리에 있는 탑승 가능한 택시를 찾기 어렵기 때문에 비효율적인 서비스이다. 또한, 택시 운전사들은 그들의 경험에만 의존하여 운전 Strategy를 결정한다.

이전 연구에서는 실시간으로 도심의 교통량 예측, 다음 승객을 더 효율적으로 찾는 방법, 공유 택시 시스템 등의 연구를 통해 택시 서비스의 효율성을 높이려고 했다. 하지만, 이러한 방법들은 short term profit에만 집중되어 있어, 본 연구에서는 long term profit을 최대로 하여 local이 아닌 global profit과 efficiency를 증가시키는 것에 중점을 두었다.

Global profit과 efficiency를 위해 강화 학습을 통해 택시 운전사들의 long term profit을 최대화하기 위한 최적의 운전 strategy를 찾는 것을 목표로 한다. 강화 학습이란 Agent가 Environment의 State를 보고 특정한 Action을 취한 뒤, Environment로부터 Reward를 얻게 되고, 또 State를 보고, Action을 취하고, Reward를 얻는 것이 반복이 된다. 이렇게 학습을 하면서 최대의 Reward를 얻기 위해 Agent가 어떤 Action을 취해야 하는지, 즉 Optimal Policy를 구하는 것이 강화 학습의 목적이 된다.

따라서 본 연구에서의 Environment는 특정 지역을 일정한 크기의 grid로 나누어 하나의 grid를 cell이라 하고, 이 cell들을 environment로 정의하였다. State는 occupied, vacant, parking 3가지로 구분이 되는데, occupied는 택시에 승객이 탑승해 있는 상태(택시가 움직이고 있고, 목적지가 있는 상태), vacant는 택시에 승객이 탑승해 있지 않은 상태로 목적 없이 돌아다니는 상태 (택시는 움직이고 있지만, 목적지는 없는 상태), parking은 택시 정류장에서 승객을 기다리고 있는 상태 (택시가 승객을 태우기 위해 움직이지 않고 대기하고 있는 상태)를 말한다.

Action은 State와 대응된다. empty\_driving, carrying\_passengers, waiting 총 3가지로 구성되어 있다. 따라서 Action은 승객이 taxi에 탄 상태와 타지 않은 상태 2가지 조건에 따라 다르게 구성되어 있다

.

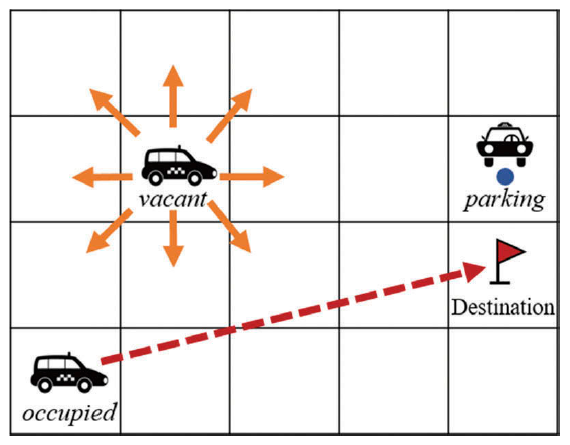


Figure 11. Action set according to states

(1)

(2)

Cannot choose the next policy until it drops off its passengers

위의 Environment, State, Action을 바탕으로 Q-learning algorithm을 통해 최적의 Strategy를 도출한다. Value Function Q는 모든 cell에서의 3가지 action의 reward의 기대값이고, Q matrix는 N\*3 matrix가 된다. 따라서 Q-learning algorithm을 통해 Q matrix의 값을 계속해서 update하게 되고, 특정 cell에서 가장 큰 Q 값을 갖는 action을 선택하게 된다.

# 5. Conclusion

PDPTW(Pickup and Delivery Problem with time windows)에서 request가 들어온 순서대로 add time이 가장 적게 늘어나는 route에 배정하는 휴리스틱 알고리즘을 고안해보았다. Time Window 제약조건이 있는 문제라, Route에 방문하는 Node의 순서뿐만 아니라, Node에 방문했을 시각을 함께 저장하여 Time Window의 feasibility 계산을 용이하게 했다. 추가적으로 휴리스틱 알고리즘으로 낸 결과를 OSMnx와 Folium 라이브러리를 활용하여 지도상에 띄울 수 있는 것에 본 연구의 의의를 둔다.

본 연구에서는 Request가 들어오는 순서대로 해당 Request의 최적 Route에 배정했다. 따라서 앞서 들어온 Request가 후에 들어온 Request의 결과에 따라 바뀔 수 없다. 하지만, 특정 시간동안 Request를 받고, 들어온 순서 상관없이 Route에 배정한다면 더 나은 Route 결과를 기대할 수 있을 것이다.

최근 다양한 조합 최적화 문제에 강화 학습을 적용하는 연구들이 많이 이루어지고 있다. 각 문제들의 특성에 맞게 강화 학습 모델의 구성 요소인 state space, action space, reward, transition probability 를 적절하게 정하는 것이 중요하다. TSP, VRP, PDPTW 등 경로 결정 문제에 강화 학습을 적용한 연구들을 더 조사하여, 본 연구의 문제에 적합한 강화 학습 모델을 고안한다면 좋은 연구가 될 것이라 기대한다.

# 6. Review

먼저 코로나로 인해 대면 연구 참여에 어려움이 많은 상황임에도 불구하고, 대면 연구 참여를 허락해주신 김병인 교수님께 큰 감사를 드립니다. 면담과 랩미팅에서 해주신 진심 어린 말씀들은 공부나 연구뿐만 아니라 살아가는데 있어서 큰 도움이 될 것 같습니다.

대면 연구 참여를 하며 연구실에 출근하면서 랩미팅에 참석 및 세미나 발표 등 소중한 경험들을 할 수 있었습니다. 항상 잘 챙겨 주시고 도와 주신 연구실 선배님들 감사합니다.

한 학기동안 Anylogic, Cplex 등을 사용해보고, 실제로 최적화 문제를 해결하기 위해서 어떻게 문제를 정의해야 하는지, 수리 모델을 만들어야 하는지 등 많은 것들을 배울 수 있었습니다.

특히 마지막에는 PDPTW 문제를 해결해보고자, VRP 관련한 논문들을 공부해보고, 수리 모델을 만들고, 휴리스틱 알고리즘 고안 및 시각화를 해보았습니다. 비록 좋은 성능은 아니지만, 전반적인 과정들을 직접 해보면서, 하나의 최적화 문제를 해결하기 위해 어떤 과정들을 거쳐야 하는지 깨달을 수 있었습니다.

마지막으로 김현준 사수님께, 항상 질문이나 고민에 대해서 여쭤보면 성심껏 답해 주셔서 정말 감사했습니다. 여러 방면으로 많은 도움이 되었습니다. 부사수로서 선배님을 뵐 수 있어서 좋았고, 저도 계속해서 공부를 하게 된다면 선배님 같은 연구자가 되고 싶습니다.

한 학기동안 많은 경험을 할 수 있도록 도움을 주신 김병인 교수님과 물류연구실 선배님들께 다시 한번 감사하다는 말씀 전하며 후기를 마치겠습니다.

감사합니다.

# 7. Reference

1. Gao, Y., Jiang, D., & Xu, Y. (2018). Optimize taxi driving strategies based on reinforcement learning. *International Journal of Geographical Information Science*, 32(8), 1677–1696.
2. Boeing, G. (2017). OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. *Computers, Environment and Urban Systems*, 65, 126-139.
3. OSMnx: Python for Street Networks (<https://geoffboeing.com/2016/11/osmnx-python-street-networks/>)
4. Folium 0.12.1 documentation (<https://python-visualization.github.io/folium/>)
5. folium의 plugins 패키지 샘플 살펴보기 (https://dailyheumsi.tistory.com/85)