Group 13

# Finding Optimal metro route using Q-Learning and compare with A* Algorithm

2020251009 Hyelee Kim

Table of Contents

# Table of Contents

Part 1

# 1. Reference Paper Summary

## <Optimal Metro Route Identification Using Q-Learning>

a system that uses the Q-learning algorithm to identify the most efficient metro routes, minimizing travel time by considering both distance and waiting times at stations.

### Objective

The goal is to find the most time-efficient metro route for a passenger, considering all possible paths to determine the shortest route in terms of both distance and time.

### Methodology

**Q-Learning Algorithm**

Employs a reward-based model-free reinforcement learning to optimize route choices.

**Environment Setup**

Models metro stations as nodes and paths as edges in a graph, including time-related travel variables.

**Reward System**

Rewards time-efficient routes and penalizes longer routes with more stops or delays.

## Key Components

**Agent**: Learns the optimal actions based on rewards received

**Environment**: Consists of all stations and routes in the metro system

**Actions**: Choices the agent can make at each step (e.g., which station to travel to next)

**Reward**: Feedback received after taking an action, aiming to reduce travel time

## Process

The Q-values (quality of actions) are updated using the formula

$$Q(s,a) = Q(s,a) + \alpha \left[ r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

where $s$ and $s'$ are current and next states, $a$ and $a'$ are actions taken in those states, $r$ is the reward received, $\alpha$ is the learning rate, and $\gamma$ is the discount factor.

The system performs multiple iterations to update Q-values, optimizing the route choices based on past experiences.

## Q-learning

**1) Initialization:**

Initializes the Q-table to any value.

**2) Experience:**

Agents gain experience by interacting with the environment.

**3) Q-value update:**

When the agent takes action and is rewarded, the Bellman equation is used to update the Q-value.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a) - Q(s_t, a_t)]$$

Here, $\alpha$ (alpha) is the learning rate, $\gamma$ (gamma) is the discrete factor, $r_{t+1}$ is the received compensation, and $\max Q(s_{t+1}, a)$ is the maximum possible Q-value in the following state.

**4) Policy Development:**

After a certain period of time, the optimal policy is derived based on the Q-table.

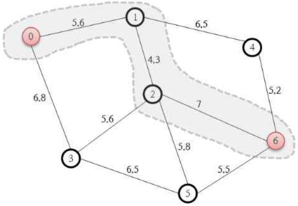
Part 2

# 2. Extend Investigation

## A* Algorithm

A* algorithm is a graph traversal algorithm used to find the shortest path from a starting point to a target point.

It combines Dijkstra's algorithm with heuristic search to traverse the graph more efficiently.




O =

| | | |
|---|---|---|
| Node ID | 5 | 4 |
| F Score | 18.8 | 17.3 |
| G Score | 13.3 | 12.1 |
| H Score | 5.5 | 5.2 |
| Parent Node | 3 | 1 |

C =

| | | | | | |
|---|---|---|---|---|---|
| Node ID | 0 | 3 | 1 | 2 | 6 |
| F Score | 0 | 16.8 | 17.6 | 16.9 | 16.9 |
| G Score | 0 | 6.8 | 5.6 | 9.9 | 16.9 |
| H Score | 0 | 10 | 12 | 7 | 0 |
| Parent Node | - | 0 | 0 | 1 | 2 |

Examples of using A* algorithm

## The fundamental principles of the A* algorithm

The A* algorithm evaluates each node using the following two functions.

1. g(n): The actual cost from the start node to node n (the cost of the path so far)
2. h(n): The estimated cost from node n to the goal node (heuristic function)

http://www.gisdeveloper.co.kr/?p=3897

The A* algorithm proceeds in the direction that minimizes the sum of these functions:

$$f(n) = g(n) + h(n)$$

The efficiency of the algorithm depends on how accurately h(n) estimates the cost to reach the goal.
Common heuristic functions include Manhattan distance and Euclidean distance.

A good reason to find the shortest distance

The A* algorithm is good for finding the shortest distance.
The reasons are as follows.

Efficiency: Reduces unnecessary path exploration using heuristics.
Completeness: Always finds a solution if one exists, given an appropriate heuristic.
Optimality: Guarantees the shortest path if the heuristic is admissible
(i.e., h(n) is less than or equal to the actual cost).

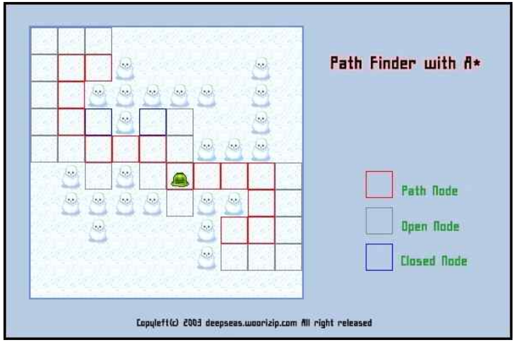
## Applications of A* algorithms

**Game Development**: Calculating movement paths for game characters

**Robotics**: Path planning for robots

**Navigation Systems**: Finding the shortest path on maps

**Artificial Intelligence**: Solving puzzles and searching problem spaces




Game Development


Navigation Systems

http://aidev.co.kr/game/501
https://unsplash.com/ko/%EC%82%AC%EC%A7%84/2-00%EC%9D%98-%EB%94%94%EC%A7%80%ED%84%B8-%EC%9E%A5%EC%B9%98-4IqOnX2sdyM

## Q-learning Algorithm vs A* Algorithm

| Feature | Q-learning Algorithm | A* Algorithm |
|---|---|---|
| **Algorithm Type** | Reinforcement learning algorithm | Heuristic search algorithm |
| **Purpose** | To learn the optimal policy in a state-action space | To find the shortest path from the start point to the goal point in a graph |
| **Learning/Search Method** | Agent interacts with the environment and learns through reward signals | Proceeds in the direction that minimizes the sum of the heuristic and path cost |
| **Advantages** | Model-free, can learn without an environment model, applicable to various environments | Efficient and guarantees the optimal path with an appropriate heuristic |
| **Disadvantages** | Can take a long time to learn, requires many trials during exploration | Requires a well-defined graph and cost function, can use a lot of memory |

https://ai.stackexchange.com/questions/23072/what-are-the-differences-between-q-learning-and-a

## Comparison of Strengths and Weaknesses

**Q-learning**

**Strengths**: Applicable to various environments, model-free, capable of learning policies.

**Weaknesses**: Long learning time, may be inefficient in the initial stages.

**A***

**Strengths**: Fast and efficient pathfinding, guarantees optimal path with an appropriate heuristic.

**Weaknesses**: Requires a clear graph and cost function, high memory usage.

These two algorithms are suitable for different types of problems, so choosing the appropriate algorithm depends on the nature of the problem.

Q-learning is suitable for reinforcement learning problems, while A* is suitable for pathfinding problems.

Part 3

# 3. Data

## Data

**Seoul Metropolitan Rapid Transit Corporation Line 1 - 8 Station Coordinates (Latitude and Longitude) Information**

https://www.data.go.kr/data/15099316/fileData.do?recommendDataYn=Y

| | 연번 | 호선 | 고유역번호(외부역코드) | 역명 | 위도 | 경도 | 작성일자 |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | 150 | 서울 | 37.553150 | 126.972533 | 1974-02-28 |
| **1** | 2 | 1 | 151 | 시청 | 37.563590 | 126.975407 | 1974-08-15 |
| **2** | 3 | 1 | 152 | 종각 | 37.570203 | 126.983116 | 1974-08-15 |
| **3** | 4 | 1 | 153 | 종로3가 | 37.570429 | 126.992095 | 1974-08-15 |
| **4** | 5 | 1 | 154 | 종로5가 | 37.570971 | 127.001900 | 1974-03-31 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **271** | 272 | 8 | 2823 | 남한산성입구 | 37.451568 | 127.159845 | 1996-10-31 |
| **272** | 273 | 8 | 2824 | 단대오거리 | 37.445057 | 127.156735 | 1996-12-28 |
| **273** | 274 | 8 | 2825 | 신흥 | 37.440952 | 127.147590 | 1996-12-28 |
| **274** | 275 | 8 | 2826 | 수진 | 37.437575 | 127.140936 | 1996-12-28 |
| **275** | 276 | 8 | 2827 | 모란 | 37.433888 | 127.129921 | 1996-11-30 |

276 rows × 7 columns

**We used the latitude and longtitude data of the stations to calculate the euclidean distance between the origin and the destination, and used this as the heuristic function in the a* algorithm.**

**We will create a dictionary using the name and location (latitude and longitude) information of public data subway stations provided by the government.**

# Data

```
station_dist = station_dist.loc[:,['역명','위도','경도']]  #station name, latitude, longtitude
station_dist = station_dist[station_dist['역명'].isin(station_names)]    #station name
station_dist
```

-> After extracting subway station names, latitude, and longitude from the DataFrame, data for selected 40 stations was extracted

```
d_list = station_dist.values.tolist()
```

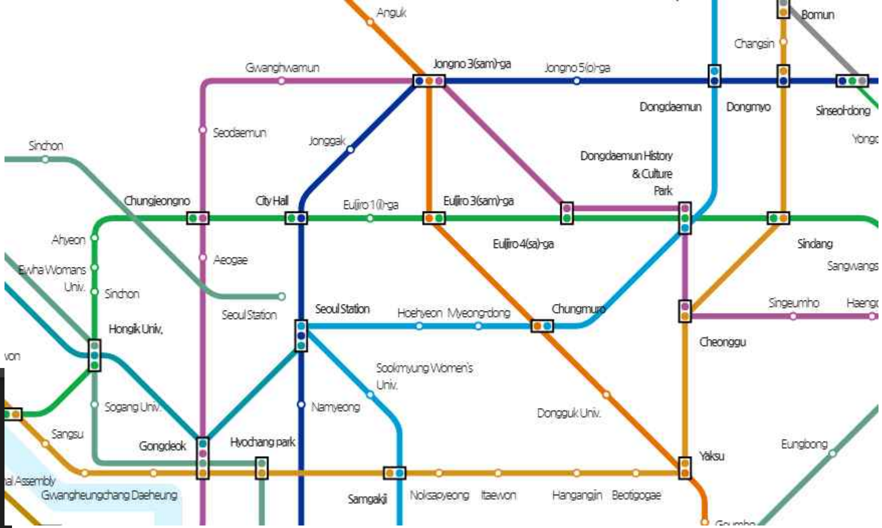
```
k_dist = {}
for i in d_list:
  k_dist[i[0]] = [i[1],i[2]]
```

-> Created a 2D list d_list from each row of the DataFrame to create a dictionary where subway station names are keys and their corresponding latitude and longitude are values

```
dist = {station_e[key]: value for key, value in k_dist.items()}
print(dist)
```

-> Created a dictionary dist where each subway station name, translated into English, serves as a key

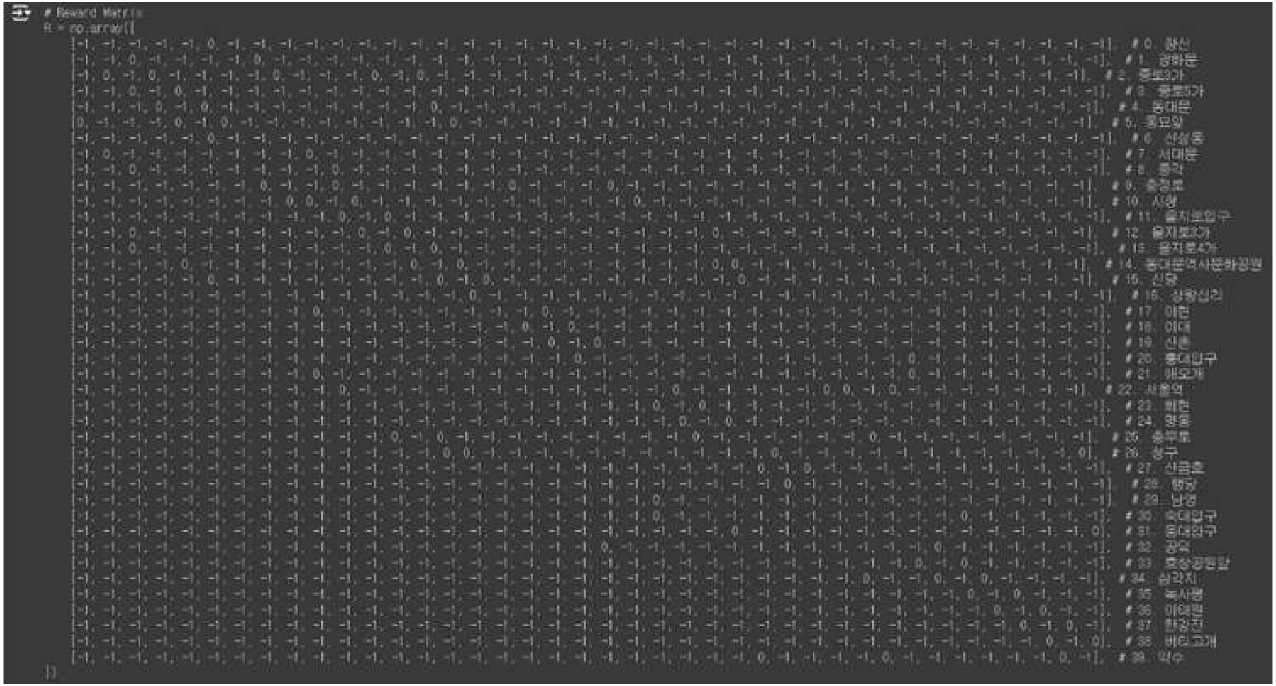
```
station_names = [
    "Changsin", "Gwanghwamun", "Jongno 3-ga", "Jongno 5-ga",
    "Dongdaemun", "Dongmyo", "Sinseol-dong", "Seodaemun",
    "Jonggak", "Chungjeongno", "City Hall", "Euljiro 1-ga",
    "Euljiro 3-ga", "Euljiro 4-ga", "Dongdaemun History & Culture Park",
    "Sindang", "Sangwangsimni", "Ahyeon", "Ewha Womans University",
    "Sinchon", "Hongik University", "Aeogae", "Seoul Station",
    "Hoehyeon", "Myeong-dong", "Chungmuro", "Cheonggu",
    "Singumho", "Haengdang", "Namyeong", "Sookmyung Women's University",
    "Dongguk University", "Gongdeok", "Hyochang Park", "Samgakji",
    "Noksapyeong", "Itaewon", "Hangangjin", "Beotigogae",
    "Yaksu"
]
```




http://www.seoulmetro.co.kr/en/cyberStation.do?menuIdx=337

In the first project, **13 subway stations** were selected.
In this second project, 40 subway stations were selected with a significant increase.

Q-learning slows down processing as the dimension increases.
More subway stations were selected to observe the difference from A*, which quickly finds the optimal route.




**40*40 matrix** was made through the dictionary.

**Each row** means a **subway station**.

**0** is an adjacent subway station (movable subway station), and **-1** is a non-movable subway station.

```python
start_time = time.time()
def heuristic(station, goal):
    station = station_names[station]
    goal = station_names[goal]
    start_x = dist[station][0]
    dest_x = dist[goal][0]
    start_y = dist[station][1]
    dest_y = dist[goal][1]
    return math.sqrt((start_x - dest_x)**2 + (start_y - dest_y)**2)
```

* Heuristic function: calculate Euclidean distance between current station and the goal station using longtitude and latitude for estimate to guide the search

```python
def astar(start, goal):
    open_list = [(0, start)]
    closed_list = set()
    came_from = {}

    g_score = {station: float('inf') for station in range(len(station_names))}
    g_score[start] = 0

    while open_list:
        current_cost, current_station = heapq.heappop(open_list)

        if current_station == goal:
            path = []
            while current_station in came_from:
                path.append(current_station)
                current_station = came_from[current_station]
            path.append(start)
            path.reverse()
            return path

        closed_list.add(current_station)

        for neighbor, cost in enumerate(R[current_station]):
            if cost < 0:
                continue

            tentative_g_score = g_score[current_station] + cost

            if tentative_g_score < g_score[neighbor]:
                came_from[neighbor] = current_station
                g_score[neighbor] = tentative_g_score
                f_score = tentative_g_score + heuristic(neighbor, goal)
                heapq.heappush(open_list, (f_score, neighbor))

    return None
end_time = time.time()
print('time spent:',end_time - start_time)
```

*astar function: The main loop processes each station in the open list.
g_score dictionary holds the cost of the cheapest path from the start to each station, initialixed to infinity for all stations except the start.
came_from dictionary is for tracking the path.
if the goal station is reached,
the path is reconstructed by following the came_from dictionary.
For each neighbor of the current station, we calculate a tentative g_score.
If this is better than the previously known g_score, we update the scores, the path, and push the neighbor onto the open list with the updated cost.

*Time function: it was used for calculating the time spent for the whole code to run, which also means the time spent for computer to calculate and find the optimal route

Part 4

# 4. Conclusion

## Result

**Case1 : Yaksu - Jonggak**

```
time spent: 0.4152791500091553
Optimal path:
Yaksu -> Cheonggu -> Sindang -> Dongdaemun History & Culture Park -> Euljiro 4-ga -> Euljiro 3-ga -> Jongno 3-ga -> Jonggak
```

**Q-learning**

```
time spent: 0.0009264945983886719
shortest path: ['Yaksu', 'Dongguk University', 'Chungmuro', 'Euljiro 3-ga', 'Jongno 3-ga', 'Jonggak']
```

**A***

It passes 6 stops in the Q-learning method and 4 stops in the A* method.
The processing time of the A* method is much faster.

**➔case 1, efficient results were obtained in a short time using the A* algorithm.**

## Result

**Case2 : Euljiro 3-ga - Cheonggu**

```
time spent: 0.2520482540130615
Optimal path:
Euljiro 3-ga -> Euljiro 4-ga -> Dongdaemun History & Culture Park -> Sindang -> Cheonggu
```

**Q-learning**

```
time spent: 0.0005445480346679688
shortest path: ['Euljiro 3-ga', 'Euljiro 4-ga', 'Dongdaemun History & Culture Park', 'Cheonggu']
```

**A***

It passes 3 stops in the Q-learning method and 2 stops in the A* method.
The processing time of the A* method is much faster.

➔ **case 2, efficient results were obtained in a short time using the A* algorithm.**

**In both cases, the A* algorithm was superior to Q-learning.**
**Therefore, the A* algorithm is suitable for use in the path search problem.**

Part 5

# 5. References

# References

**Optimal metro route identification using Q-Learning**
https://ieeexplore.ieee.org/document/9573726

**Research on Optimal Path based on Dijkstra Algorithms**
https://www.atlantis-press.com/proceedings/icmeit-19/55917280

file:///C:/Users/USER/Downloads/Optimal_metro_route_identification_using_Q-Learning.pdf
https://mangkyu.tistory.com/61
https://kau-algorithm.tistory.com/7
https://velog.io/@hamkua/%EC%A7%80%ED%95%98%EC%B2%A0-%EC%B5%9C%EB%8B%A8%EA%B2%BD%EB%A1%9C-a
https://blog.naver.com/codnjs9999/220367824625

https://www.youtube.com/watch?app=desktop&v=wbWUJjD2oCo
https://corporatefinanceinstitute.com/resources/wealth-management/heuristics/
https://brilliant.org/wiki/dijkstras-short-path-finder/
https://chaudharysrasthy1528.medium.com/ospf-routing-protocol-implemented-using-dijkstras-algorithm-7c40a8d11d8c
https://data.seoul.go.kr/dataList/OA-12034/S/1/datasetView.do
https://www.sisul.or.kr/open_content/skydome/introduce/pop_subway.jsp
https://develop-dream.tistory.com/89
https://map.naver.com/p?c=15.00,0,0,0,dh
--------------------------------------------------------------------------------------------------------------------------------------------------------
http://www.gisdeveloper.co.kr/?p=3897
http://aidev.co.kr/game/501
https://unsplash.com/ko/%EC%82%AC%EC%A7%84/2-00%EC%9D%98-%EB%94%94%EC%A7%80%ED%84%B8-%EC%9E%A5%EC%B9%98-4IqOnX2sdyM
https://ai.stackexchange.com/questions/23072/what-are-the-differences-between-q-learning-and-a
https://develop-dream.tistory.com/89
https://www.data.go.kr/data/15099316/fileData.do?recommendDataYn=Y

# E.O.D.

Thanks very much!