An Intelligent Parcel Delivery Scheduling Scheme in Road Networks

Junhee Kwon
Department of Computer Science & Engineering
Sungkyunkwan University
Suwon, Republic of Korea
Email: juun9714@skku.edu

Jaehoon (Paul) Jeong
Department of Computer Science & Engineering
Sungkyunkwan University
Suwon, Republic of Korea
Email: pauljeong@skku.edu

***Abstract*—Due to the development of an autonomous vehicle industry, the route finding is the important feature for this industry. This study proposes an intelligent parcel delivery scheduling scheme that utilizes machine learning to optimize delivery schedules for electric vehicles. The simulation involves managing electric vehicles that visit randomly assigned multiple destinations, and each vehicle determines the order of visits based on the algorithm used. Through simulation, the paper shows that the proposed scheme outperforms two legacy schemes (i.e., greedy and branch-and-bound algorithms), considering both route computation time and vehicle travel time.**

***Index Terms*—Machine learning, parcel delivery, road networks, electric vehicles, TSP**

I. Introduction

Similar to how telephones evolved from mere communication devices to smartphones with various functionalities, cars are transitioning from simple transportation means to mobile computers. Intelligent driving systems have garnered significant attention in order to mitigate losses caused by inexperienced driving or mistakes. These systems have capabilities such as autonomous lane recognition [1], distance adjustment, and alerts of approaching vehicles during lane changes. Users who have experienced these features often anticipate fully autonomous driving in near future and express a demand for it. Through the fully automated vehicle, drivers can enjoy in-vehicle activities, such as streaming contents, gaming and just relaxing. Those activities, which are called infotainment, make drivers' in-vehicle time meaningful. For example, an electric device vendor Sony revealed the concept-car with full display inside and outside of the car [2]. As Sony already owns many game contents, industry expects that the gaming experience will be equipped in fully self-driving car. Consequently, automotive manufacturers are considering the infotainment services in autonomous driving environments.

To make these activities work, the vehicle must perform the fundamental tasks of safe and efficient driving. These tasks are to select paths independently, avoid obstacles, recognize the surroundings, and follow determined paths in cooperation with other vehicles. Through software (SW) and hardware (HW), these requirements can be fulfilled. Software technology has demonstrated impressive performance across various domains and is progressively expanding its range of applications. In the automotive industry, SW applications, such as road recognition through computer vision [3] and voice recognition [4], contribute to enhancing the overall in-vehicle experience. The most basic and fundamental task among the mentioned requirements is a route planning task of navigation. Currently, vehicle navigation systems leverage real-time road network information to provide drivers with the fastest routes to their destinations. However, such systems may overlook historical data which could influence driving outcomes. Even if the shortest route is computed, the driving experience may deviate from expectations if such events are not considered.

Machine learning (ML) techniques offer a solution to this problem by leveraging past driving data to learn patterns and predict outcomes in specific situations. In the context of vehicle navigation, ML techniques can be employed to forecast traffic congestion. By employing an ML model trained on historical data, congestion levels reflecting various scenarios can be predicted and used to determine optimized routes. Motivated by the potential and objectives of ML, this paper undertook this research endeavor.

The research focuses on path planning for vehicles engaged in specific missions. Specifically, it involves determining the most costly efficient and fastest routes for delivery service drivers who need to visit multiple destinations while minimizing time and fuel consumption. This scenario shares similarities with a well-known problem in Computer Science (CS) area, the Traveling Salesman Problem (TSP) [5]. TSP addresses the question of the optimal order with which a salesman should visit multiple destinations while minimizing travel time and maximizing efficiency. As an NP-hard problem [6], calculating all possible combinations is theoretically feasible but practically infeasible within a reasonable timeframe. A Greedy algorithm [7] offers a fast and straightforward implementation but does not guarantee an optimal solution. On the other hand, a Branch-and-Bound algorithm computes all possible combinations while pruning some cases. In this study, after implementing a navigation system based on Greedy algorithm, Branch-and-Bound algorithm, and Machine learning, the performance of each technique was evaluated.

The remainder of this paper is organized as follows. Section II summarizes the related work about the algorithms. Section III describes the architecture of simulation for the performance

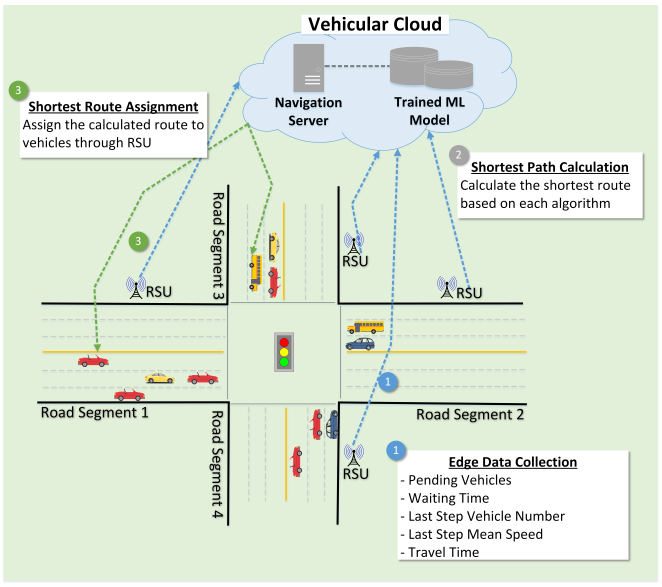

Fig. 1. Road network architecture based on SUMO and TraCI library

evaluation. Section IV shows the implementation of our navigation system and Section V evaluates performance of our simulation. Lastly, Section VI concludes this paper along with future work.

II. Related Work

This section provides an overview of the necessary knowledge to comprehend this paper with the algorithms proposed in recent literature.

The abbreviation TSP stands for the traveling salesman problem, which holds significant popularity within the software domain [8]. It involves identifying a most efficient route while visiting all the assigned destinations. The salesman initiates a journey from the starting point and must return there. Due to the incredibly large number of possible routes, exhaustive calculation within polynomial time becomes impossible to find shortest route. Consequently, this problem is classified as NP-Hard [6]. To address the NP-Hard problem, a heuristic algorithm is employed [9].

The heuristic algorithm includes the greedy algorithm, which makes choices based on a best option at each step, without considering overall optimization. As a result, it does not guarantee the shortest route, but it provides a fast and straightforward implementation that yields a reasonable solution. Another algorithm, known as the Branch and Bound (BnB) algorithm, was developed to solve the time-consuming nature of classic algorithms [10]. To reduce calculation time, this algorithm keeps the minimum value while evaluating cases and compares it with the current calculated value. This mechanism effectively eliminates unnecessary computations [11].

Not only heuristic algorithms, machine learning methods are applied to the navigation area. Polynomial regression [12] is a type of regression analysis that models the relationship between an independent variable and a dependent variable using a polynomial function. Unlike linear regression, which assumes a linear relationship, polynomial regression allows for nonlinear relationships by incorporating higher-order polynomial terms. The polynomial function fits a curve to the data points, enabling it to capture more complex patterns and variations. By including polynomial terms of different degrees, such as quadratic or cubic terms, polynomial regression can provide a more flexible and accurate representation of the data. It is commonly used when the relationship between variables is expected to be curvilinear rather than linear. Curvilinear refers to a shape characterized by curved lines or paths, rather than straight lines or linear trajectories. It describes the deviation from a linear pattern, introducing curves, arcs, or bends in the movement or form.

[13] tried to address the challenges of traffic congestion monitoring and estimation in developing countries. The authors propose using intelligent transport systems and machine learning techniques to analyze and predict real-time traffic conditions. The authors gather data from multiple devices to identify traffic speed and congestion patterns, with the aim of providing adaptable solutions for traffic management. [14] compared the route searching methods, which are greedy algorithm and BnB algorithm. In this research, BnB algorithm resulted better performance in TSP situation. Based on this paper, it is shown that a pruning method of the BnB algorithm is helpful for reducing the calculation time and making the BnB algorithm useful.

III. Architecture

This section discusses the architecture and working flow of our simulation to evaluate the performance of three methods. The simulation in this paper involves the management of electric vehicles that visit multiple destinations assigned randomly. Based on each algorithm, each electric vehicle determines the order of visits and subsequently proceeds to visit the destinations one step at a time. After visiting all the destinations, each vehicle concludes its run and records the total duration of the trip.

A. SUMO and Road Network

We employ the Simulation of Urban Mobility (SUMO) [15] program for road network simulation. It allows researchers to design road networks with customized features. Through this simulator, programmers can simulate the various scenarios of the urban traffic based on SUMO. To make the SUMO simulation dynamic, we made use of the TraCI python library. TraCI allows users to create objects for the simulation and input them into the simulation environment. Moreover, the data is available for users to extract while the vehicles are running within a target road network based on the SUMO.

As shown in Fig. 1, the road network in our simulation comprises edges, nodes (e.g., intersections), and lanes. Edges connect nodes, and each edge consists of multiple lanes. Traffic lights are present at all nodes and cycle through patterns. Road networks is 12x12 grid pattern. Vehicles start

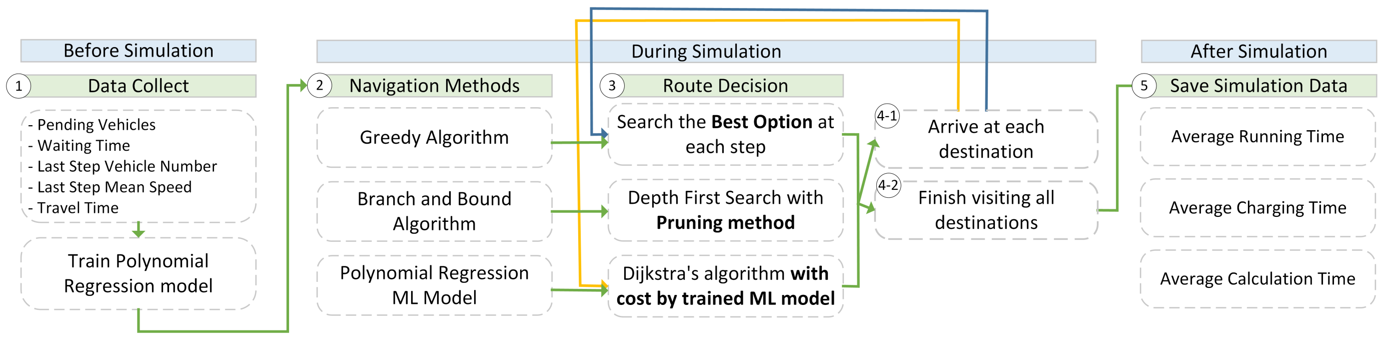
Before Simulation
1 Data Collect
- Pending Vehicles
- Waiting Time
- Last Step Vehicle Number
- Last Step Mean Speed
- Travel Time
Train Polynomial Regression model
During Simulation
2 Navigation Methods
Greedy Algorithm
Branch and Bound Algorithm
Polynomial Regression ML Model
3 Route Decision
Search the Best Option at each step
Depth First Search with Pruning method
Dijkstra's algorithm with cost by trained ML model
4-1 Arrive at each destination
4-2 Finish visiting all destinations
After Simulation
5 Save Simulation Data
Average Running Time
Average Charging Time
Average Calculation Time

Fig. 2. Overall work flow of all the algorithms

TABLE I
MEAN SQUARED ERROR AND R-SQUARED SCORE

Metric	Score
Mean Squared Error	0.0019475
R-squared Score	0.8776582

running from same start point. Vehicles possess various attributes, including an identification (id), state, battery capacity, destination, and the id of the next charging station. Using the vehicle id, the simulation can assign the destinations that need to be visited by a vehicle. Each vehicle is an electric vehicle. It searches for a charging station based on its own algorithm. The road network is equipped with several charging stations.

B. Machine Learning: Polynomial Regression

We utilized a Polynomial Regression model to predict congestion levels for each edge. Polynomial Regression predicts a single output with multiple inputs. In this study, we used four attributes to predict travel time for each edge as specified as step 1 in Fig. 2. Pending Vehicles, Waiting Time, Last Step Vehicle Number, and Last Step Mean Speed for each edge were extracted from the SUMO simulator for input data in advance. Travel Time for each edge was also extracted for output. Since the data were collected at each step of the simulation, it can be predicted in various situations.

Through this process, we extracted approximately 6.77 million data points. We also trained a Polynomial Regression model using these training data as step 1. To assess the model's performance, the Mean Squared Error (MSE) and R-Squared Score (R^2) were calculated. MSE is used to measure the average squared difference between the predicted and actual values, providing a quantitative assessment of the model's accuracy. R^2 is a statistical measure that indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, with higher values indicating a better fit of the model to the data. As shown in Table I, MSE resulted in 0.0019475 and R-squared Score resulted in 0.8776582. After the training process, the model was saved to be loaded for future simulation.

C. Working Flow

Using the TraCI library, the SUMO simulator can be executed with customized conditions based on Python code. We aim to determine driving routes for electric vehicles with multiple destinations using three approach such as Greedy algorithm, BnB algorithm and Machine Learning (ML) technique. To ensure consistent experimentation in the same environment, we implemented all the methods with identical destination information.

The overall process is shown in Fig. 2. Upon the execution of the TraCI code, the simulation commences, and the TraCI code generates vehicles and their corresponding destination lists randomly. Subsequently, each vehicle is assigned a destination list. Then, code executes the iteration to run all three algorithms with same destination list for vehicles as steps 2 and 3 in Fig. 2. When each algorithm ends, driving records are saved as a file containing the running time, charging time, and calculation time for each vehicle in step 5. Based on the saved information, the efficiency of each algorithm can be compared.

Even if the biggest difference among algorithms is the way it finds the efficient route to visit all the destinations. In the case of the greedy algorithm and ML, the complete sequence of visiting orders is not predetermined prior to the initial departure. As a result, methods go back to step 3 after step 4-1 to select an optimal choice at each destination. ML utilizes the trained model and current edge information to predict the travel time of the edges when determining the next destination.

In the case of the BnB algorithm, the entire route is determined before the initial departure unlike the greedy algorithm. Consequently, additional search time for finding the next destination is unnecessary at each destination. Based on the calculated information, the fastest route can be explored. This process employs Depth-First Search (DFS) and the BnB algorithm.

IV. IMPLEMENTATION

In this section, detailed implementations of the Greedy algorithm, BnB algorithm and Machine learning are described.

A. Greedy Algorithm

In Greedy algorithm, once a vehicle reaches its targeted destination, it searches for the next destination to visit. In

Algorithm 1 Greedy Algorithm

```
function FINDNEXTGREEDY($from$)
    for i to $length(D)$ do
        if $distance(from, D[i]) < min$ then
            $min \leftarrow distance(from, D[i])$
            $T \leftarrow i$
        end if
    end for
end function

$T \leftarrow 0$
$D \leftarrow DestinationList$
$S \leftarrow StartingPoint$
while $length(D) > 0$ do
    $min \leftarrow sys.maxsize$
    if Not Running then
        FINDNEXTGREEDY($S$)
        RUN()
    else if Running & Arrived then
        REMOVEFROMD($T$)
        FINDNEXTGREEDY($T$)
        RUN()
    end if
end while
```

Algorithm 2 Branch and Bound Algorithm

```
function DFS($visited$, $value$, $min$, $final$, $destNum$)
    if $min < value$ then
        return                                  ▷ Branch and Bound part
    end if
    if $length(visited) == length(destNum)$ then
        $final \leftarrow visited$
        return
    end if
    for $i$ to $length(destNum)$ do
        if $i$ not in $visited$ then
            VISITED.APPEND($i$)
            $value \leftarrow value + distanceTo(i)$
            DFS($visited$, $value$, $min$, $final$, $destNum$)
            VISITED.POP()
        end if
    end for
end function
```

Algorithm 1, T represents the road ID of the current destination, D is the list of assigned destinations, and S denotes the starting point. The algorithm updates minimum value (denoted as min), and finds the nearest destination using $FindNext()$ function. This process continues until the vehicle finish visiting all assigned destinations.

If vehicle is at start point, the vehicle calculates a closest destination prior to departure. After that, the vehicle begins driving towards the first destination. If it is running, it needs to determine the next destination to proceed with. The vehicle removes the destination which it has already visited from the list through $RemoveFromD()$ function. Subsequently, it searches for the next destination to reach based on its current location, utilizing $FindNext()$ function. This iteration continues until there are no more destinations to visit, indicating that D is empty. The time complexity is $O(n)$ when n is the number of destination.

Algorithm 3 Machine Learning Method

```
function FINDNEXTML($edgeList$, $PRmodel$, $Dests$)
    $ExpTime \leftarrow$ COLLECTEDGEDATA($edgeList, PRmodel$)
    $Distances, Predec \leftarrow$ DIJKSTRA($ExpTime$)
    $Costs \leftarrow 0$                                  ▷ Initialize Costs
    for $dest$ in $Dests$ do
        $Costs \leftarrow Costs + Distances[dest]$
    end for
    $NextDest \leftarrow$ MIN($Costs$)
    $Route \leftarrow$ SHORTESTPATH($Predec, NextDest$)
    return $NextDest$ and $Route$
end function

function COLLECTEDGEDATA($edgeList$, $PRmodel$)
    $finalData \leftarrow []$
    for $edge$ in $edgeList$ do
        $finalData \leftarrow finalData + pendingVeh$
        $finalData \leftarrow finalData + waitingTime$
        $finalData \leftarrow finalData + meanSpeed$
        $finalData \leftarrow finalData + vehNum$
    end for
    $TravelTime \leftarrow$ PRMODEL($finalData$)
    return $TravelTime$
end function
```

B. Branch and Bound Algorithm

The BnB algorithm includes a mechanism to reduce computation time while considering all possible cases. The BnB algorithm utilizes a DFS [16] algorithm. It also utilizes parameters such as min and D. The visited list (denoted as $visited$) sequentially records the visited destinations, and the $final$ list contains a path that $DFS()$ function will ultimately return. $destNum$ represents the number of destinations to be visited, and $value$ records the accumulated time during the path exploration.

In Algorithm 2, $DFS()$ function has a recursive structure. When the function is called, $value$ is updated by adding the additional time to reach the next destination. The $visited$ list is updated by adding the next destination. If the length of the $visited$ list is equal to $destNum$, the path is stored in $final$, and returned. The min value is continuously updated. If the intermediate cost is already greater than the min value, that case is not proceeded more. The time complexity is $O(n^n)$ when n is the number of destination.

C. Machine Learning Method

In ML method, it determines the nearest destination based on not only the data provided by the simulation but also a pretrained Polynomial Regression model. This model calculates the anticipated congestion level for each edge. Then, employing Dijkstra's algorithm, which considers the congestion level

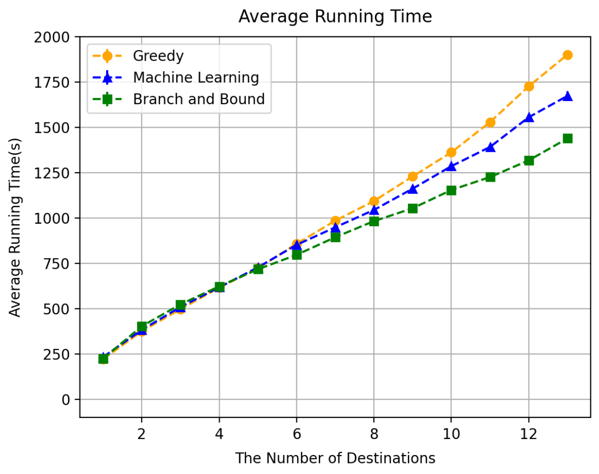

Fig. 3. Average running time according to the number of destinations

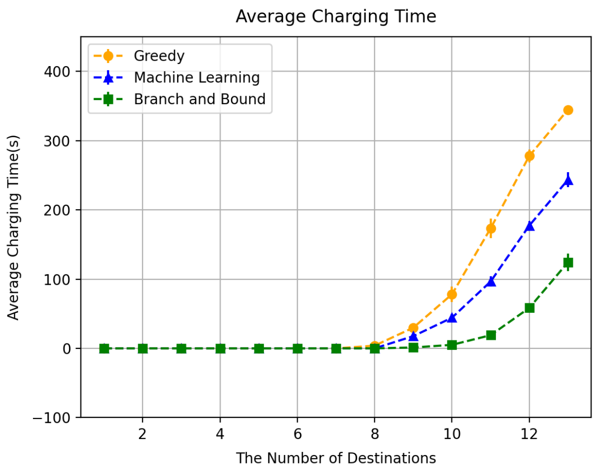

Fig. 4. Average charging time according to the number of destinations

as the cost, it determines the minimum cost required to reach each edge in the road network.

In Algorithm 3, $FindNextML()$ requires the pre-trained Polynomial Model and the list of remaining destinations for the vehicle as input. It then passes the $edgeList$ and Polynomial model to $CollectEdgeData()$. This function uses the Polynomial Model with the current simulation data to calculate the anticipated Travel Time for each edge and returns it. $FindNextML()$ utilizes the estimated Travel Time to determine the shortest cost as $Distances$ for each edge and the corresponding shortest path information as $Predec$.

The $Cost$ variable stores the minimum cost for the destinations. Among these costs, the destination with the least cost is selected as $NextDest$. Then $Route$ is returned, which represents a shortest path to reach $NextDest$. Ultimately, $FindNextML()$ computes and returns the next destination and the corresponding route for the vehicle is provided with $Route$ and $NextDest$. The time complexity is $O(n^2)$ when n is the number of destination.

V. Performance Evaluation

This section describes the experiment environment and the performance results of the three methods.

A. Experiment

As shown in Figs. 3, 4, and 5, this study conducted performance measurement under various conditions. A vehicle number is fixed as 30, and the number of destinations varies from 1 to 13 and parcels are equally distributed on every vehicle at first. Authors did not limit the delivery time. Each experiment was repeated 10 times, and an error bar was used as a 95% confidence interval for performance evaluation.

Average travel time, average charging time, and average computation time were set as performance evaluation metrics. Average charging time represents the time spent at the charging station. In the case where the designated route is inefficient, the vehicle may travel a greater distance, resulting in charging during operation. Average computation time refers to the time to search for routes toward the target. While the most efficient routes are important in real-world, guaranteeing moderate computation time for real-world is also a crucial factor.

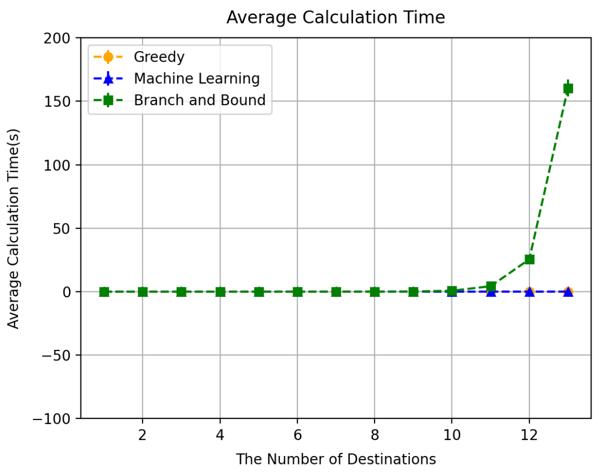

Fig. 5. Average calculation time according to the number of destinations

B. Testing Results

In Fig. 3, the horizontal axis is the number of destinations and the vertical axis is average running time. As the number of destinations increases, the average running time of all the methods and gaps between them are getting bigger. A bigger curve means inefficient. Based on the graph, the greedy algorithm shows the worst performance. As greedy algorithm just consider the status of present and do not consider the optimization of route efficiency, it resulted the worst performance. Between the other two methods, BnB algorithm resulted in the lower running time. From five destinations, it kept resulting in the best performance among all the methods.

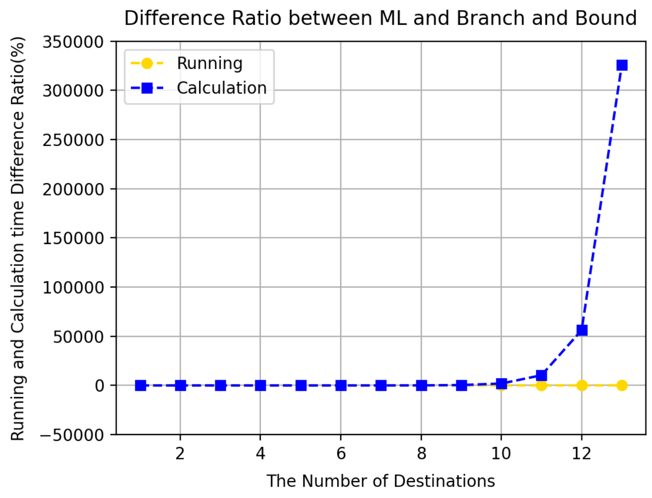

Fig. 6. Difference ratio between ML and BnB methods in terms of running time and calculation time

In Fig. 4, the vertical axis is average charging time. When the efficiency of the route is low, a vehicle has no choice but to wander the road network and consumes more energy. Based on this logic, the charging time of the vehicle can be used as a performance metric for the efficiency of the determined route. Similar to Fig. 3, the greedy algorithm showed the worst performance and the BnB algorithm resulted in best performance, too.

However, what we need to consider is the feasibility in real-world. If its calculation time is too long, drivers cannot use the method even if it results in the most effective route for vehicles. Based on this concern, we recorded the calculation time of the ML and BnB algorithms as shown in Fig. 5. Ratio between the two methods in terms of Running time and Calculation time is shown in Fig. 6 in a percentage format. The running time difference ratio between two methods are not so big, from 2% to 15%. However, the calculation time difference ratio between two methods get bigger. It resulted in $325,894\%$ when the number of destinations is 13. As this calculation time is not acceptable to be used in the real world, we concluded that the ML method is better choice even if BnB results in an optimal route.

VI. Conclusion

This paper researched on which algorithm results in best efficient route for TSP situation. Through this research, delivery service vehicles can be run based on an efficient route and make logistics industry more efficient.

Three performance metrics were compared, such as running time, charging time, and calculation time. As a result, the BnB algorithm resulted in the best route for this navigation problem and the worst calculation time. Through this, we concluded that the calculation time is also important feature in terms of a navigation method. As a result, the BnB method can be more efficient when there are a small number of destination as its calculation time gets bigger when there are a large number of destinations. When there are a large number of destination, the ML method is better than the BnB algorithm as ML provides a moderate route for vehicles and takes shorter time than the BnB algorithm.

As a future work, only Polynomial Regression was applied, but we will consider other machine learning models or another method to spread the traffic through road networks.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2023R1A2C2002990), and also by the IITP grant funded by the Korea MSIT (No. 2022-0-01199-002). Note that Jaehoon (Paul) Jeong is the corresponding author.

References

[1] S. Waykole, N. Shiwakoti, and P. Stasinopoulos, "Performance evaluation of lane detection and tracking algorithm based on learning-based approach for autonomous vehicle," *Sustainability*, vol. 14, no. 19, p. 12100, 2022.

[2] P. Valdes-Dapena, "Sony and honda reveal their new car brand," *CNN Business*. [Online]. Available: https://www.cnn.com/2023/01/04/tech/sony-honda-afeela

[3] J. Janai, F. Güney, A. Behl, A. Geiger *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Foundations and Trends® in Computer Graphics and Vision*, vol. 12, no. 1–3, pp. 1–308, 2020.

[4] E. Okur, S. H. Kumar, S. Sahay, A. Arslan Esme, and L. Nachman, "Natural language interactions in autonomous vehicles: Intent detection and slot filling from passenger utterances," in *International Conference on Computational Linguistics and Intelligent Text Processing*. Springer, 2019, pp. 334–350.

[5] K. L. Hoffman, M. Padberg, G. Rinaldi *et al.*, "Traveling salesman problem," *Encyclopedia of operations research and management science*, vol. 1, pp. 1573–1578, 2013.

[6] G. J. Woeginger, "Exact algorithms for np-hard problems: A survey," in *Combinatorial Optimization—Eureka, You Shrink! Papers Dedicated to Jack Edmonds 5th International Workshop Aussois, France, March 5–9, 2001 Revised Papers*. Springer, 2003, pp. 185–207.

[7] J. Bang-Jensen, G. Gutin, and A. Yeo, "When the greedy algorithm fails," *Discrete optimization*, vol. 1, no. 2, pp. 121–127, 2004.

[8] G. A. Croes, "A method for solving traveling-salesman problems," *Operations research*, vol. 6, no. 6, pp. 791–812, 1958.

[9] D. S. Hochba, "Approximation algorithms for np-hard problems," *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997.

[10] M. E. Pfetsch, "Branch-and-cut for the maximum feasible subsystem problem," *SIAM Journal on Optimization*, vol. 19, no. 1, pp. 21–38, 2008.

[11] W. Zhang, "Truncated branch-and-bound: A case study on the asymmetric tsp," in *Proc. Of AAAI 1993 Spring Symposium on AI and NP-hard problems*, vol. 160166, 1993.

[12] E. Ostertagová, "Modelling using polynomial regression," *Procedia Engineering*, vol. 48, pp. 500–506, 2012.

[13] S. J. Kamble and M. R. Kounte, "Machine learning approach on traffic congestion monitoring system in internet of vehicles," *Procedia Computer Science*, vol. 171, pp. 2235–2241, 2020, third International Conference on Computing and Network Communications (CoCoNet'19). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920312321

[14] J. Kwon and J. Jeong, "A parcel delivery scheduling scheme in road networks," in *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*. IEEE, 2022, pp. 1750–1755.

[15] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker, "Recent development and applications of sumo-simulation of urban mobility," *International journal on advances in systems and measurements*, vol. 5, no. 3&4, 2012.

[16] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.