A 1.5-Approximation Route Finding for a Ride-sharing considering Movement of Passengers

Yonghwan Kim
Nagoya Institute of Technology
Nagoya, Aichi, Japan
kim@nitech.ac.jp

Masato Amano
Nagoya Institute of Technology
Nagoya, Aichi, Japan
m.amano.460@nitech.jp

Daisuke Yamamoto
Nagoya Institute of Technology
Nagoya, Aichi, Japan
daisuke@nitech.ac.jp

ABSTRACT

MaaS (stands for *Mobility as a Service*) is a concept that aims to integrate different transportation services into a unified and seamless mobility solution. It encourages a shift away from personally owned modes of transportation, such as private cars, towards a more comprehensive approach that combines public transportation, such as ride-sharing, bike-sharing, carpooling, and other modes of transport into a single and user-centric service. One of the services offered within *MaaS* is a ride-sharing, which provides several advantages such as cost savings, reduced traffic congestion, or environmental benefits. However, to make a ride-sharing efficient, a sophisticated route finding (*i.e.,* planning) is required to avoid redundantly long routes when picking up or dropping off passengers.

In this paper, we consider an efficient ride-sharing route finding for large-vehicles such as buses that starts at the determined location and returns after visiting all the locations, when a set of the locations of the passengers is given. Moreover, we allow passengers to move within a short distance to find more efficient traversal plan. Note that this problem can be reduced to the traveling salesperson problem (with some constraints) which is a well-known NP-hard problem. Hence, we employ a *1.5-approximation algorithm* to find an efficient route within a reasonable computational time, moreover, use the *Viterbi algorithm* to improve the efficiency of the route when allowing each passenger to move.

CCS CONCEPTS

• **Theory of computation** → *Discrete optimization*; • **Information systems** → Geographic information systems.

KEYWORDS

ride-sharing, route finding, approximation algorithm, TSP

ACM Reference Format:
Yonghwan Kim, Masato Amano, and Daisuke Yamamoto. 2023. A 1.5-Approximation Route Finding for a Ride-sharing considering Movement of Passengers. In *1st ACM SIGSPATIAL International Workshop on Sustainable Mobility (SuMob '23), November 13, 2023, Hamburg, Germany.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3615899.3627933

SuMob '23, November 13, 2023, Hamburg, Germany

ACM ISBN 979-8-4007-0361-4/23/11…$15.00
https://doi.org/10.1145/3615899.3627933

1 INTRODUCTION

Recently, *MaaS*, which stands for "*Mobility as a Service*", attracts much attention of researchers. It is a concept that refers to the integration and provision of various transportation services as a unified and seamless mobility solution. *MaaS* aims to shift the focus from personally owned modes of transportation, such as a private car, to a more comprehensive approach that combines public transportation, such as ride-sharing, bike-sharing, carpooling, and many other modes of transport into a single and user-centric service. A ride-sharing is one aspect of *MaaS*, which shares a transportation with two or more individuals who travel in the same direction or route. A ride-sharing has many advantages including the following: cost savings; passengers can reduce their expense by splitting the cost of transportation, reduce the traffic congestion; it helps decrease the number of vehicles, and environmental benefits.

A ride-sharing can take various forms such as carpooling (*i.e.,* individuals who live or work in close proximity share the transportation), application-based ride-sharing (*i.e.,* connects passengers to drivers), or a community bus (also known as a shuttle bus; public transportation service for specific communities or areas within a city or town). Here we mainly consider a ride-sharing for large-vehicles such as buses which have some different features including the following: (1) *higher capacity*; buses have the advantage of carrying more passengers in a single trip (*e.g.,* a private car usually accommodates less than 10 people, a bus can transport more than 40). This results in better cost sharing and a more positive environmental impact on a per-passenger basis. (2) *lower flexibility*: a bus ride-sharing comes with less flexibility. Bus routes and schedules are usually predetermined and fixed, which means passengers have limited options for pick-up and drop-off locations as well as travel times. Due to the higher capacity of a bus ride-sharing, applying the existing methods has many challenges in terms of computation time and route efficiency. When we consider the optimal route for all the locations of many passengers that starts at a certain location and return to there after visiting all the locations, the total route forms a cycle and this implies that the problem can be reduced to traveling salesperson problem (*TSP*) [7, 10] which is a well-known NP-hard problem. This means that to find the optimal one requires an unacceptable computational cost. Therefore, in many situations, a heuristic or approximate approach is utilized to obtain satisfying (good enough) results, especially from the practical viewpoint. The *hill climbing* [25] is a local search method commonly used to solve an optimization problem such as TSP, which iteratively improves the current solution from an initial one; however, this approach can get stuck in a local optimum, where no further improvements can be made. The *Simulated annealing* [19] is an enhancement to overcome the limitation of being trapped in local optima by allowing for

occasional moves to other solutions to avoid getting trapped in local optima. However, the simulated annealing has several drawbacks including the following: (1) unstable output; it provides different results on each execution, (2) high computational cost; it requires a significant number of iterations to get an acceptable result, and (3) unable to guarantee its performance; even if it executes a huge number of iterations (*i.e.,* high computational cost), it cannot guarantee the efficiency of the result. In buses ride-sharing, a set of passengers is usually fixed due to its *lower flexibility*, hence we find a near-optimal (*i.e.,* approximate) solution.

Moreover, in buses ride-sharing, it can be beneficial to allow passengers to move to nearby desired locations. This can significantly reduce the overall length of the route. Therefore, we allow passengers to move to the location within a short distance to improve the efficiency of the result (*i.e.,* find a shorter route). However, finding an efficient route while considering passenger movement makes the problem more challenging. It involves determining the visiting points among many possible candidate points in the TSP, leading to a large number of combinations. Solving such a problem requires a considerable amount of computational cost, even when using heuristic algorithms. In particular, the simulated annealing algorithm requires a large number of iterations and increases the complexity of each iteration. This makes it even more computationally expensive to obtain an acceptable result. In conclusion, allowing passenger movement in a ride-sharing route planning introduces additional complexity to the problem, therefore, a sophisticated algorithm is necessary to obtain an acceptable result within a reasonable computational time in this problem setting.

Contribution

In this paper, we present an algorithm to find a ride-sharing route traversing all passengers' locations with allowing passenger movement within a small distance.

Our proposed method has the following key features:

(1) We adopt a **1.5-approximation algorithm** to obtain an effective traversal order of the passengers' locations within a reasonable computational time. The algorithm guarantees a result within 1.5 times the optimal solution. Moreover, an approximation algorithm is deterministic, thus it always provides the same result for the same input, whereas the simulated annealing algorithm produces the different result on each execution even for the same input.

(2) We consider the movement of the passengers; for each passenger, the proposed method identifies all the candidate locations where a passenger can move. To determine the most suitable location for each passenger, our method employs the **Viterbi algorithm**. This allows for a swift selection of the appropriate location among all the candidate locations. Therefore, the proposed method effectively generates a ride-sharing route, providing the traversal order for all passengers along with their respective locations.

We implement a prototype system based on proposed method to find a ride-sharing route and visually displays the resultant route on a map to help to verify its result. We also implement another method which is heuristic based on the *simulated annealing* technique [19] to find a ride-sharing route for the evaluation of the proposed method. As a result of experimental evaluations, we show that the proposed system outperforms the one based on the simulated annealing approach. Furthermore, we also present that the proposed system efficiently finds a ride-sharing route within a reasonable computational time, even with an increasing number of passengers or larger allowable distances for passenger movement.

2 RELATED WORKS

Recently, various studies on route planning for ride-sharing are introduced. Many studies focus on the dynamic determination of ride-sharing routes in real-time. J. Alonso-Mora *et al.* [2] propose a system that dynamically assigns passengers to several vehicles with varying passenger capacities and allows for reassignment based on demand. Additionally, Y. Yoshizuka *et al.* [29] evaluate the performance of an algorithm that assigns passengers with individual destinations to Smart Access Vehicles (*SAV*s) used for ride-sharing services in real-time. K. Bathla *et al.* [5] propose a new distributed taxi ride-sharing algorithm to address dynamic scheduling of ride-sharing requests, and S. Ma *et al.* [20] propose a large-scale taxi ride-sharing service handling real-time requests from passengers in the dynamic ride-sharing problem which significantly reduces the total travel distance. Mohammad Asghari *et al.* [3] addresses the real-time ride-sharing problem and emphasizes the importance of designing a scalable framework that matches riders and drivers while considering their constraints and maximizing the platform's profit, and introduce a distributed auction-based framework. The authors show that the proposed framework achieves higher service rates, shorter trips for riders, and increased overall profit for ride-sharing platforms, based on the comparisons with state-of-the-art approaches using real data from New York City's taxi dataset. They also focuses on the evaluation of ride-sharing platforms and addresses the challenge of preventing manipulation for personal gain in [4]. The authors propose a latent space transition model that allows drivers to predict the future availability of drivers.

There are some studies specifically focusing on ride-sharing with a single destination. T. Yoshida *et al.* [28] propose an algorithm to minimize the total travel distance of passengers while ensuring that the total distance traveled by taxis does not increase significantly in the context of shared taxi usage with a common destination. Additionally, R. Massobrio *et al.* [22] and H. E. Ben-Smida *et al.* [8] conduct research on taxi ride-sharing problems with a single origin. C. Tao *et al.* [26] handle the ride-sharing problem with a single origin, considering time constraints, in an online setting. As a ride-sharing planning including multiple destinations, A. K. M. Mustafizur Rahman Khan *et al.* [18] present a ride-sharing plan so that all users visit the destinations where they agree on when a set of points of interest (*POI*s) and a set of users are given. The authors resolve this ride-sharing route planning problem as a construction of minimum Steiner tree problem (a well-known NP-hard problem).

In addition, there are studies on delivery planning problems using quantum annealing. H. Irie *et al.* [17] and K. Saito *et al.* [23] have conducted research on optimization problems that determine the order in which vehicles visit customers in a general delivery planning problem.

Furthermore, research has been conducted on route exploration for ride-sharing that takes passenger movement into account. M. Stiglic *et al.* [24] investigated the benefits of introducing meeting points in ride-sharing, where drivers and passengers with individual destinations are matched. Meeting points refer to locations

within a certain distance from the departure or destination points, where passengers can get on or off. In this approach, they use a $k-d$ tree, which supports Euclidean distance nearest neighbor search, n nearest neighbors search, and fixed-radius near neighbor search in logarithmic time [9], to store meeting points and determined the optimal meeting points through nearest neighbor searches. Additionally, K. Aissat *et al.* [1] proposed exact and heuristic approaches for matching a single driver with multiple passengers in ride-sharing, aiming to minimize the total travel distance of the driver and passengers while identifying meeting points. The exact approach employed enumeration, while the heuristic approach differentiated meeting points as candidate boarding and alighting positions and then searched for routes. Based on the aforementioned research, it was found that utilizing meeting points significantly increases the matching rate of passengers and reduces travel distances. It was also observed that when using meeting points, it is important to secure as many meeting points as possible and choose them carefully.

The problem addressed in this paper is closely related to the traveling salesperson problem (TSP) [7, 10], which finds the shortest path visiting all cities exactly once and returning to the starting city. TSP is known as an NP-hard problem thus an unacceptable computational cost is required for the optimal solution. Various approaches have been introduced to tackle the TSP problem. The Held-Karp algorithm [6], based on the dynamic programming, breaks down the problem into smaller sub-problems and recursively solves them. However, this approach requires an exponential time $\Theta(n^2 2^n)$ and significant space $\Theta(\sqrt{n}2^n)$. Despite this, it is much faster than a brute-force algorithm that requires a superexponential time $\Theta(n!)$. The nearest neighbor [16] repeatedly selects the closest unvisited city, forming a cycle. Although simple and fast, it does not guarantee an optimal solution. The random search [21] generates random solutions by permuting the order of visited points and evaluates the total distance for each solution. Hill climbing [25] is a local search method that iteratively improves a solution until it reaches a local optimum. However, it can get stuck in local optima without further improvements. Simulated annealing [19]is an enhancement that allows occasional moves to solutions with higher distances, avoiding being trapped in local optima and exploring a broader solution space. The k-approximation algorithm provides a solution guaranteed to be within k times the optimal solution for TSP instances. The Christofides' algorithm [11, 15] is a 1.5-approximation algorithm for TSP which terminates within $O(n^3)$ time.

3 PROPOSED METHOD

In this section, we introduce the proposed method to find a ride-sharing route that visit all the locations of passengers. The proposed method consists of four steps; finding all shortest paths, determining the traversal order of the passengers, finding all the candidate locations, and determining the final location of each passenger. Figure 1 illustrates the flow of the proposed method consisting of four steps.

3.1 Find All Shortest Paths

As we mentioned in the previous section, our goal is to find a ride-sharing route that covers all the locations of the passengers. We

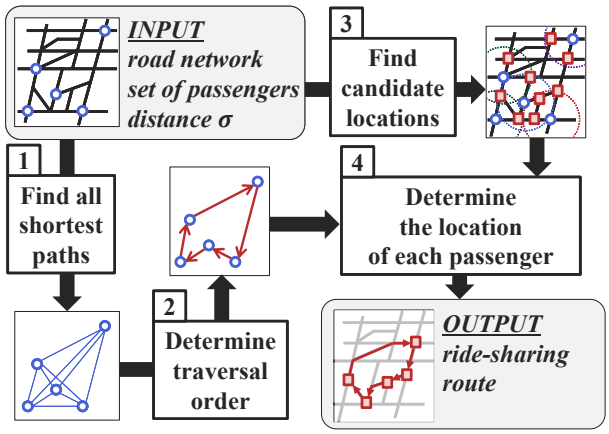


Figure 1: Flow of the proposed method consisting four steps

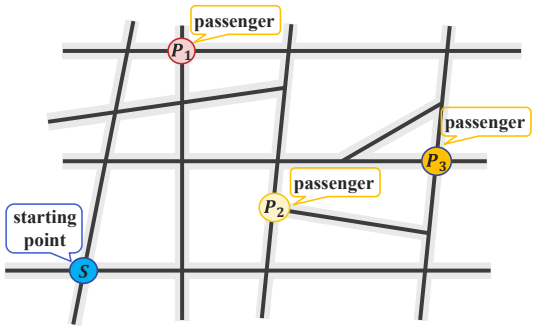


Figure 2: Example of road network and passengers

begin by considering the starting point, denoted by $\mathcal{S}$, from where the transportation starts and returns after visiting all the locations of the passengers. We consider a set of points $\mathbb{P}$, which includes all the passenger locations $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \ldots$. Figure 2 illustrates an example of a road network with starting point $\mathcal{S}$ and the locations of the passengers.

We find all the shortest paths between every pair of nodes (including all locations in $\mathbb{P}$). To find all these paths, we use the *Floyd-Warshall algorithm* [13] which is a dynamic programming algorithm used to find the shortest paths between all pairs of vertices in a weighted graph. In our case, the weight of each edge represents the distance. The algorithm terminates in $O(V^3)$ time, where V is the number of vertices. Algorithm 1 shows the Floyd-Warshall algorithm for a weighted graph $G = (V, E)$.

3.2 Determine the Traversal Order

In the previous step introduced above, we have the shortest paths between all pairs of vertices. This implies that every pair of points in $\mathbb{P}$, therefore we can consider the complete weighted graph $K_{\mathbb{P}} = (\mathbb{P}, E_{\mathbb{P}})$, where the set of edges $E_{\mathbb{P}}$ consists of the shortest path between two points in $\mathbb{P}$: $E_{\mathbb{P}} = \{(i, j) | \forall i \in \mathbb{P}, \forall j \in \mathbb{P}, i \neq j\}$, and $weight(e = (i, j))$ is defined by the length of the shortest path from i to j ($\in \mathbb{P}$). To find an efficient traversal (*i.e.,* visiting) order, now

Algorithm 1 The Floyd-Warshall algorithm

1: **for** $\forall i \in V$ **do**
2: **for** $\forall j \in V$ **do**
3: $$d[i][j] \leftarrow \begin{cases} weight(e) & \text{edge } e = (i, j) \text{ exists} \\ 0 & i = j \\ \infty & \text{edge } e = (i, j) \text{ does not exist} \end{cases}$$
4: **for** $\forall k \in V$ **do**
5: **for** $\forall j \in V$ **do**
6: **for** $\forall j \in V$ **do**
7: **if** $d[i][j] > d[i][k] + d[k][j]$ **then**
8: $d[i][j] \leftarrow d[i][k] + d[k][j]$

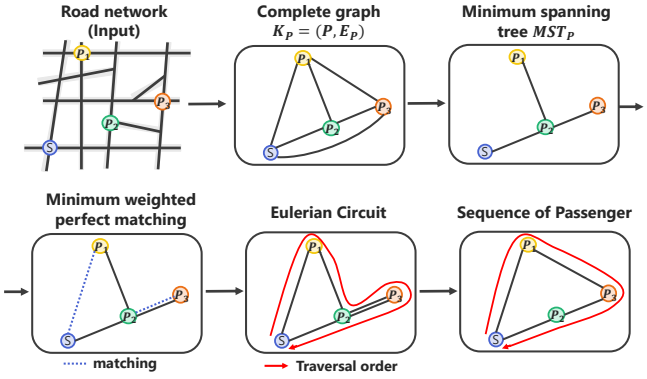


Figure 3: Flow for finding a Hamiltonian circuit

we find a cyclic path that visits all the points in $\mathbb{P}$ as short as possible. This problem setting is the same as the traveling salesperson problem (TSP), therefore, we use the *Christofides' algorithm* which is a 1.5-approximation algorithm for TSP.

3.2.1 Christofides' 1.5-Approximation Algorithm. Here we introduce the 1.5-approximation algorithm, called *Christofides' algorithm* [11, 15], we used in this work as follows.

(1) Construct a minimum spanning tree (MST) on a given graph G.
(2) Find odd-degree vertices and create a minimum weight perfect matching M of these vertices.
(3) Combine MST and M to form a connected multigraph H and find an Eulerian circuit in H.
(4) Shortcut the Eulerian circuit by skipping any repeated vertices, resulting in a Hamiltonian circuit.

The resultant Hamiltonian circuit created by Christofides' algorithm is guaranteed to be at most 1.5 times the optimal solution for TSP instances. This algorithm requires $O(n^3)$ where n is the number of vertices because $O(n^3)$ time is necessary for perfect matching. It is important to note that the Christofides' algorithm does not handle asymetric TSP instances or instances with triangular inequality violations. However, we deal with the actual geographic map data that satisfy all the above conditions.

3.2.2 How to Apply the Christofides' algorithm. Here we describe how to apply the Christofides' algorithm to find a traversal order.

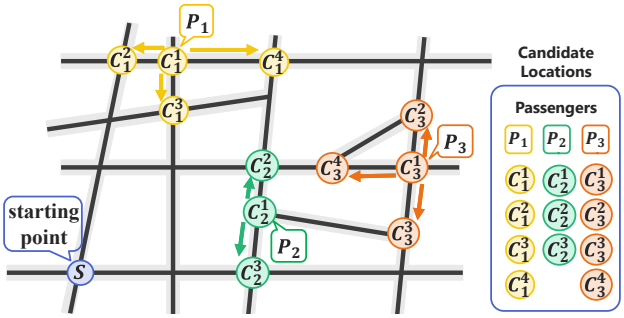


Figure 4: Example of candidate locations for each passenger

(1) Construct complete weighted graph $K_{\mathbb{P}} = (\mathbb{P}, E_{\mathbb{P}})$. The weight of each edge $e = (i, j)$ in $E_{\mathbb{P}}$ is set by the shortest distance between i and j which is calculated by Floyd-Warshall algorithm in the previous step.
(2) Construct a minimum spanning tree ($\text{MST}_{\mathbb{P}}$) on $\mathbb{P}$ using the *Prim's algorithm.*
(3) Find all odd-degree points (called $\mathbb{P}^{odd}$) on $\text{MST}_{\mathbb{P}}$, and find the minimum weighted complete matching $\text{M}_{\mathbb{P}}$ on the induced subgraph by $\mathbb{P}^{odd}$ on $K_{\mathbb{P}}$ using the *Edmonds' algorithm* [12].
(4) Construct a multigraph $\text{H}_{\mathbb{P}}$ by combining $\text{MST}_{\mathbb{P}}$ and $\text{M}_{\mathbb{P}}$ and find a Eulerian cycle on $\text{H}_{\mathbb{P}}$. A Hamiltonian circuit is determined by skipping any repeatedly visited points in the Eulerian cycle.

As a result of these above steps, we can obtain the sequence of points started at $\mathcal{S}$ from the resultant Hamiltonian circuit. Figure 3 shows the flow of finding a cyclic path.

3.3 Find the Candidate Locations

We allow each passenger to move to the different location within some (pre-determined) short distance σ. Here we find all locations that exist within distance σ from any passenger. The details are given in the following.

(1) Obtain the distance of all points with respect to each passenger. Note that the shortest distance from each passenger to every point is already calculated by the Floyd-Warshall algorithm in the first step.
(2) Find all points located within distance σ from any passenger.
(3) Store all points located within distance σ from one passenger as the set of its candidate locations. Note that the current location of the passenger is also stored because $0 \leq \sigma$ always holds.

Figure 4 shows an example of the candidate locations of each passenger.

3.4 Determine the Location of Each Passenger

Here we determine the final location of each passenger among the candidate locations to obtain the ride-sharing route as a result (refer to Figure 5). We use the *Viterbi algorithm* [27], which is a dynamic programming algorithm to find the most likely sequence of hidden states in a hidden Markov model (HMM). In an HMM,

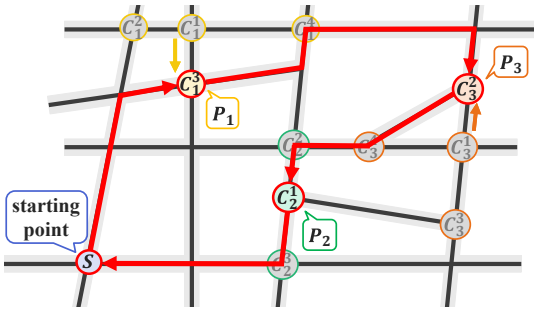


Figure 5: Example of a ride-sharing route

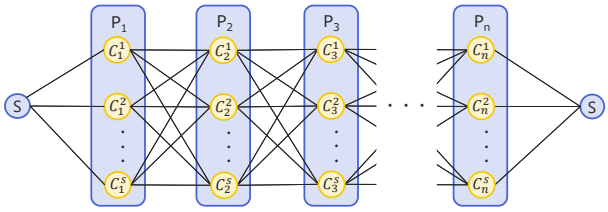


Figure 6: Graphical representation of Viterbi algorithm

we have a sequence of observations, but the underlying states that generated these observations are hidden. The Viterbi algorithm aims to determine the most probable sequence of hidden states that could have generated the given observations. When this technique is applied to find a cyclic path, we can find a more effective path based on the costs between two consecutive visit points among many possible paths. In particular, we find a more effective path by minimizing the objective function 1.

$$\underset{(k_1,k_2,k_3,\ldots,k_n)}{\operatorname{argmin}} \left\{ L\left(S \to C_1^{k_1}\right) + \sum_{i=2}^{n} L\left(C_{i-1}^{k_{i-1}} \to C_i^{k_i}\right) + L\left(C_n^{k_n} \to S\right) \right\} \tag{1}$$

In function 1, $L\left(u \to v\right)$ means the shortest distance from u to v, $P_1, P_2, P_3, \ldots, P_n$ denote each passenger, and $C_1^k, C_2^k, C_3^k, \ldots, C_n^k$ represent the set of candidate locations of passenger i. Figure 6 gives an overview of the Viterbi algorithm; here we determine from k_1 to k_n so that minimize the total length of the path, which implies that the location of every passenger is determined.

Algorithm 2 represents the Viterbi algorithm to determine the location of each passenger in the proposed method, where $C_i = \{C_i^1, C_i^2, C_i^3, \ldots\}$ is the set of candidate locations of i-th passenger appeared in the Hamiltonian circuit in Step 2. In Algorithm 2, we use some variables as follows: matrix variable $E[u][v]$ stores the shortest distance from u to v obtained by the Floyd-Warshall algorithm, and array variable $L[u]$ is the total distance of the path from $\mathcal{S}$ to u. As an exception, $L[\mathcal{S}]$ means the total distance of the cyclic path started at $\mathcal{S}$ visiting one candidate location of every passenger and returning to $\mathcal{S}$. This implies that $L[\mathcal{S}]$ becomes the output of the proposed method.

Algorithm 2 The Viterbi algorithm

```
for i ∈ {1, 2, 3, ..., n} do
    L[i] ← ∞
for C_1^i ∈ C_1 do
    L[C_1^i] ← E[S][C_1^i]
for i ∈ {2, 3, ..., n} do
    for C_{i-1}^j ∈ C_{i-1} do
        for C_i^k ∈ C_i do
            if L[C_i^k] > L[C_{i-1}^j] + E[C_{i-1}^j][C_i^k] then
                L[C_i^k] ← L[C_{i-1}^j] + E[C_{i-1}^j][C_i^k]
for C_n^i ∈ C_n do
    if L[S] > L[C_n^i] + E[C_n^i][S] then
        L[S] ← L[C_n^i] + E[C_n^i][S]
```

4 PROTOTYPE SYSTEM

To evaluate the proposed method introduced in Section 3, we implement a prototype system that executes the proposed algorithm. Moreover, for an evaluation the proposed method by comparing another algorithm, the system also includes another heuristic algorithm (simulated annealing) in the system for a comparison with the proposed method.

4.1 Overview of the Prototype System

Figure 7 shows the overview of the prototype system.

As a preprocess of the system, we find all the shortest paths between every pair of nodes within a given area (by the method given in Section 3.1) using a Road DB which contains road data obtained from OpenStreetMap [14]. The result is maintained in the *shortest path DataBase* (DB).

When the required input, the starting point, set of the passengers' locations, and the maximum distance each passenger can move, are given, the system executes two methods presented in Section 3.2 and 3.3 respectively. First, it finds a Hamiltonian circuit that visits all the locations exactly once to determine the traversal order of the passengers. And the system also finds all candidate locations of every passenger. These two results are then used to determine the

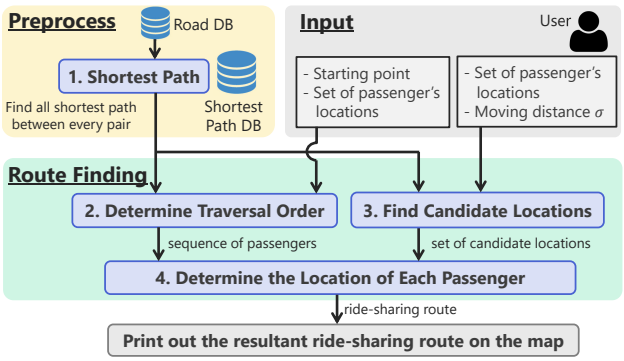


Figure 7: Overview of the prototype system

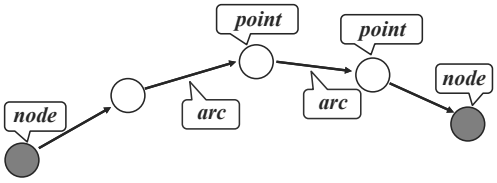


Figure 8: Structure of road data

location of each passenger and find a ride-sharing route. Finally, the system displays the resultant route on a map as the final output.

4.2 Data of the Prototype System

We use *OpenStreetMap* [14] to obtain the actual road data. Figure 8 illustrates the structure of the road data; each road data consists of nodes, points, and arcs. A point is a point on the road having location information presented by a coodinates of latitude and longitude. A node is a special point which is located on the intersection of the roads. Ar arc is a sequence of the points from one node to another. A shape of each road can be approximately illustrated by connecting of every two adjacent points on the road. When an area is determined, we construct a road network by obtaining necessary road data, nodes, points, and arcs, from OpenStreetMap, and store them in DataBase as *Road DataBase* (DB). When we draw the resultant ride-sharing route on the map, we also use OpenStreetMap to display a map.

4.3 Functions of the System

The prototype system includes the following features:

- **Change properties of map.** A user can change the scale of the map and the displayed area by moving the map.

Figure 9: The prototype system

- **Set the points on the map.** A user can set the starting point $\mathcal{S}$ and the locations of the passengers by clicking any location on the map. We prepare an option so that all points are set on arbitrary locations on the map (only the number of points is required as an input).
- **Set the distance σ.** A user can set the distance that each passenger can move. In the system, a user can set this distance from 0 meters to 800 meters, with increments of 100 meters.
- **Method selecting.** We implement the proposed method to find a ride-sharing route in the system. Moreover, we also implement another heuristic algorithm based on the simulated annealing for comparison.

Figure 9 presents the prototype system we implemented displaying the resultant ride-sharing route on the map.

5 EXPERIMENTAL EVALUATION

In this section, we give the experimental results by comparing the proposed method with a heuristic method (simulated annealing) using the prototype system.

5.1 Preparation

We consider a rectangular area of Nagoya city in Japan; with each side measuring approximately 5.4km. To simulate the locations of the passengers, We randomly select a set of points within this area. Moreover, we also choose one point as the starting point $\mathcal{S}$.

We implement the following two algorithms.

(1) **1.5-approximation algorithm (**1.5AA**)** is our proposed method, which provides a ride-sharing route that is guaranteed to be within 1.5 times the optimal solution. This algorithm is deterministic, which means that it always gives the same output for the same input set. Therefore, we execute the method only once for each input set.
(2) **Simulated annealing (**SA**)** is a general heuristic algorithm commonly used in optimization problems. This method involves a randomized approach, which provides different results on each execution. To account for this randomness, we execute the algorithm 20 times and calculate the average overall distance of the resultant ride-sharing routes.

It is important to note that the simulated annealing algorithm produces more efficient results when the number of iterations increases, meaning more time is spent on computation. This suggests that if we allocate a significant amount of computational time to the simulated annealing algorithm, it will yield much more efficient result compared to the proposed method. Consequently, we explore two approaches for executing the simulated annealing: a *fixed-time* execution (denoted by FTSA) and a *fixed-iteration* execution (denoted by FISA). The former executes the algorithm within the same execution time as the proposed method, while the latter executes the algorithm with a predetermined number of iterations, which is determined by the preliminary experiment (will be described later).

Equation 2 is the acceptance probability P_k used in method SA, where ΔE is the difference between the length of the new route e_{new} and the length of current route e_{cur}; $\Delta E = e_{new} - e_{cur}$. In a fixed-time execution, we use Equation 3 for T in Equation 2, where $\mathcal{T}_{\mathrm{AA}}$ is the execution time spent by the proposed method with the

same input and t_k is it until k-th iterations by SA. In a fixed-iteration execution, we use Equation 4 for T, where $\mathcal{L}_{max}$ is the maximum number of iterations (determined by preliminary experiment) and k is the current number of iterations.

$$P_k = \exp\left(-\frac{\Delta E}{T}\right) \tag{2}$$

$$T = \frac{\mathcal{T}_{\mathsf{AA}} - t_k}{\mathcal{T}_{\mathsf{AA}}} \times 100 \tag{3}$$

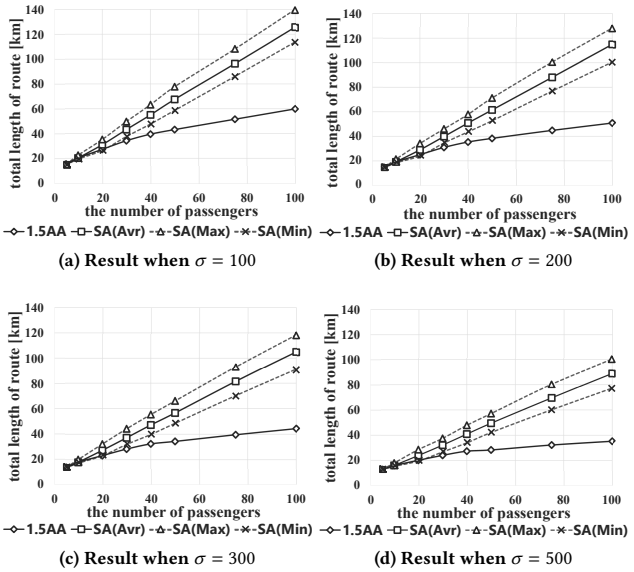
$$T = \frac{\mathcal{L}_{max} - k}{\mathcal{L}_{max}} \times 100 \tag{4}$$

5.2 Comparison with a Fixed-time Simulated Annealing

In this experiment, we evaluate two algorithms with varying numbers of passengers: 5, 10, 20, 30, 40, 50, 75, and 100. For each case, we randomly generated 20 sets of locations for the passengers. Additionally, we adjusted the distance (denoted by σ) that the passengers could move. The value of σ is set from 100 meters to 500 meters, with increments of 100 meters. In each experiment, we first execute the proposed algorithm 1.5AA and measure the time $\mathcal{T}_{\mathsf{AA}}$ required in 1.5AA for each execution. Afterward, we execute the method SA within the same time frame $\mathcal{T}_{\mathsf{AA}}$ (we call this execution FTSA).

Figure 10 shows the total lengths of the routes obtained from the two algorithms when the number of passengers varies from 5 to 100. Note that we exclude the result for $\sigma = 400$ due to space constraints. However, the same pattern can be observed in the results. The results also include the maximum and minimum route lengths provided by FTSA for reference.

When the number of passengers is 5 or 10, the two algorithms produce very similar results. However, when the number of passengers is 20 or more, algorithm 1.5AA yields shorter ride-sharing



(a) Result when $\sigma = 100$ (b) Result when $\sigma = 200$

(c) Result when $\sigma = 300$ (d) Result when $\sigma = 500$

Figure 10: Total length of ride-sharing routes by FTSA

Table 1: Standard deviations of fixed-time SA **(**FTSA**)**

Passengers	5	10	20	30	50	100
$\sigma = 100$	0.152	0.959	2.418	3.093	4.937	6.936
$\sigma = 200$	0.137	0.954	2.680	3.082	4.708	6.927
$\sigma = 300$	0.137	0.909	2.379	3.213	4.660	6.957
$\sigma = 400$	0.143	0.792	2.320	2.897	4.491	7.038
$\sigma = 500$	0.122	0.859	2.467	2.999	4.282	6.453
Average	0.138	0.895	2.453	3.057	4.616	6.862

routes compared to algorithm FTSA. In particular, when σ is set to 300, with 20 passengers, there is a reduction of approximately 15% in the length of the route. With 30 passengers, the reduction is around 24%; with 40 passengers, it is approximately 31%; and with 50 passengers, it is about 39%. Overall, algorithm 1.5AA consistently produces shorter routes than the average route length obtained by FTSA. Moreover, for cases with 30 or more passengers, algorithm 1.5AA yields shorter routes than the minimum route length obtained by FTSA. This implies that even if FTSA occasionally produce shorter route lengths, the ride-sharing route length tends to be longer compared to those produced by 1.5AA. Similar results are obtained even when $\sigma \neq 300$, although there may be slight variations.

Next, we focus on the variability (*i.e.,* unevenness) of the resultant routes obtained by FTSA. Table 1 shows the standard deviations of FTSA for each passenger count. It is obvious that as the number of passengers increases, the standard deviation also increases. This indicates that the accuracy of algorithm FTSA becomes more unstable as the number of passengers grows. For instance, with 5 passengers, the standard deviation is 0.138 km, while with 10 passengers, it is 0.895 km. These values suggest relatively low variability but unstable solution accuracy. Furthermore, with 20 passengers, the standard deviation jumps to 2.453 km, indicating a significant difference between good and poor solutions. Examining the results in Figure 10(c), we can see that the difference between the maximum and minimum values increases as the number of passengers rises, pointing to unstable solution accuracy. Based on these findings, it can be concluded that the proposed algorithm 1.5AA outperforms FTSA in terms of both result efficiency and stability.

Figure 11 shows the difference of the execution time (*i.e.,* the time taken from receiving the input data until the completion of the ride-sharing route search) between 1.5AA and FTSA using the ratio calculated as the execution time of FTSA dividing by it of 1.5AA. This means that when the ratio is larger (resp. smaller) than 100%, FTSA spent more (resp. less) execution time than 1.5AA. Even the difference of the execution time between two algorithms is at most approximately 7.64% (when the number of passengers is 30 and $\sigma = 500$), these results shows that our method for adjusting the execution time ($\mathcal{T}_{\mathsf{AA}}$) of these two algorithms, 1.5AA and FTSA, equal operates correctly.

Now we check the average number of iterations in algorithm FTSA when it spent similar execution time to 1.5AA. Figure 12 illustrates the number of iterations in FTSA when its execution time is restricted by time $\mathcal{T}_{\mathsf{AA}}$ spent in 1.5AA. The result shows that the number of iterations decreases for up to 40 passengers. This is

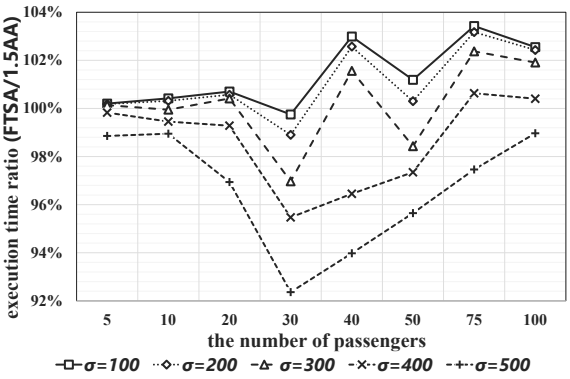


Figure 11: Execution time of FTSA **compared to** 1.5AA

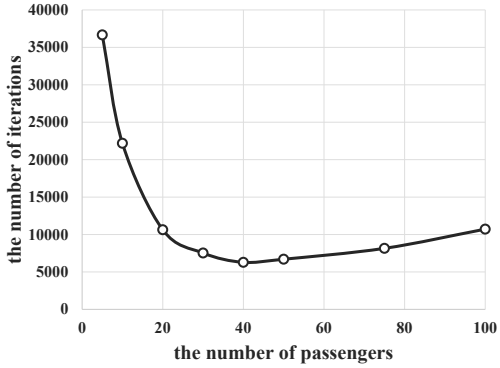


Figure 12: Number of iterations in fixed-time SA

because that the computational complexity per iteration of FTSA increases with the growing number of passengers. In other words, although the execution times are similar, the increased computational complexity per iteration leads to a decrease in the overall iteration count. However, the number of iterations gradually increases for more than 40 passengers; When the number of passengers becomes larger, even algorithm 1.5AA requires a significant execution time. As a result, FTSA has ample execution time to execute more iterations, even with the increased complexity per iteration.

In this experiment, since the number of iterations depends on the restricted execution time $\mathcal{T}_{\mathsf{AA}}$, the number of iterations may be insufficient for a certain number of passengers, leading to inefficient ride-sharing routes. Therefore, investigating the appropriate iteration count for FTSA through preliminary experiments is important to improve its efficiency.

5.3 Preliminary Experiment for a Fixed-iteration Execution

For the comparison with a fixed-iteration SA, here we conduct a preliminary experiment to find an appropriate number of iterations in SA when 20 sets of the locations of 50 passengers are given.

Table 2 presents the results of the experiment showing the average length of the resultant routes and the average of the execution time when the number of iterations varies from 100 to 200,000. We also provide the corresponding *t-test* results as Table 3. In each result, if the calculated p-value is below the threshold (we assume 0.05 generally used), then it indicates a significant difference (*i.e.,* the null hypothesis is rejected in favor of the alternative hypothesis).

Table 2: Average length of routes and average execution times in various number of iterations (from 100 to 200K)

iterations	100	1K	10K	100K	200K
length (km)	59.125	71.905	66.831	52.880	55.375
time (ms)	39.1	235.4	2172.8	21542.4	43055.0

Table 3: *t-test* result for the experiment in Table 2

	100	1K	10K	100K	200K
100	-	2.938e-4	0.002	0.009	0.024
1K	-	-	0.016	3.107e-8	1.681e-9
10K	-	-	-	6.134e-8	1.315e-11
100K	-	-	-	-	0.605

From the result presented in Table 2, the total length of the routes becomes the minimum when the number of iterations is 100,000. And from the *t-test* result in Table 3, the p-value between 100K and 200K is 0.605 which is larger than the threshold 0.05, which means that the difference between 100K and 200K is not significant. Hence, we limit the number of iterations to around 100,000. Considering the execution time becomes about 21.5 seconds when the number of iterations is 100,000, the less number of iterations is desired to find a ride-sharing route within a reasonable execution time.

Next, we present the experimental results as Table 4 when the number of iterations varies from 10,000 to 100,000. And the corresponding *t-test* result is presented in Table 5 (as the above, the threshold level for the *t-test* is set at 0.05).

From the result in Table 4, the minimum length is achieved when the number of iterations is 100,000, however, from the *t-test* result in Table 5, the difference between 70K and 100K is not significant because the p-value is less than the threshold. This implies that there is no significant difference between the results obtained at

Table 4: Average length of routes and average execution times in various number of iterations (from 10K to 100K)

iterations	10K	30K	50K	70K	100K
length (km)	65.800	59.795	57.090	54.611	53.123
time (ms)	2434.1	7244.7	12063.0	16883.2	24125.1

Table 5: *t-test* result for the experiment in Table 4

	10K	30K	50K	70K	100K
10K	-	1.994e-4	4.884e-6	3.549e-10	3.858e-9
30K	-	-	0.030	1.035e-4	3.017e-6
50K	-	-	-	0.045	4.569e-4
70K	-	-	-	-	0.193

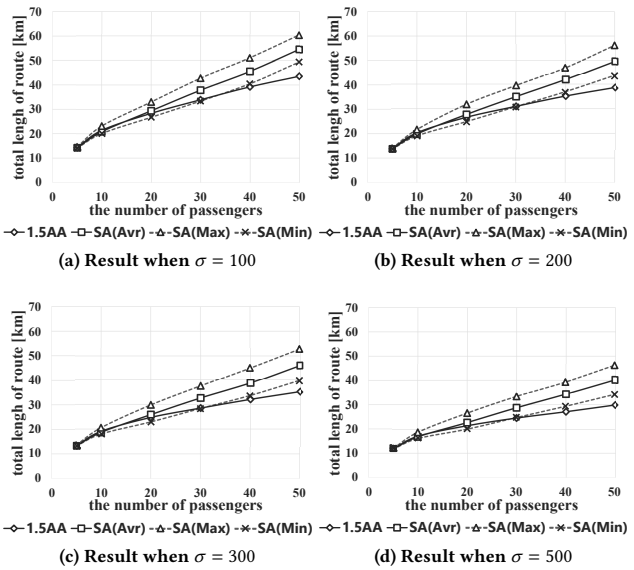
70,000 and 100,000 iterations. Therefore, we conclude that the appropriate number of iterations for the simulated annealing method in this experiment is 70,000.

5.4 Comparison with a Fixed-iteration Simulated Annealing

Now we compare the proposed method with a fixed-iteration SA, when the number of iterations is fixed to 70,000 obtained by the preliminary experiment in Section 5.3. All the other settings are the same as them in the previous experiments presented in Section 5.2.

Figure 13 presents the total length of the routes obtained by two algorithms; 1.5AA and a fixed-iteration SA. Note that also in these results, we exclude the result for $\sigma = 400$ due to space constraints as Section 5.2. In every result, the difference between two algorithms become smaller than the previous experiment, because the number of iterations is enough to obtain an effective ride-sharing route regardless its execution time. When the number of passengers is 10 or less, the results of two algorithms are similar, however, in the case of 20 or more passengers, the proposed method 1.5AA provides more efficient routes than SA's. In particular, at 20 passengers, it is about 4% shorter, at 30 passengers about 13% shorter, at 40 passengers about 17% shorter, and at 50 passengers about 24% shorter. Note that when the number of passengers is 40 or more, the results obtained by 1.5AA yields shorter length of ride-sharing route compared to the minimum case of SA.

Table 6 shows the standard deviations of fixed-iteration SA. As the same as the previous experiments, the result present that as the number of passengers increases, the standard deviation values become larger even the values become smaller compared to fixed-time SA. However, although the stability of the result is improved compared to the previous experiment, SA still exhibits some variability.



(a) Result when $\sigma = 100$ (b) Result when $\sigma = 200$

(c) Result when $\sigma = 300$ (d) Result when $\sigma = 500$

Figure 13: Total length of ride-sharing routes by FISA

Table 6: Standard deviations of fixed-iteration SA

Passengers	5	10	20	30	50	100
$\sigma = 100$	0.210	0.881	1.793	2.435	2.806	3.116
$\sigma = 200$	0.171	0.801	1.872	2.362	2.855	3.252
$\sigma = 300$	0.188	0.807	1.973	2.543	3.035	3.369
$\sigma = 400$	0.175	0.777	1.872	2.406	2.883	3.109
$\sigma = 500$	0.139	0.809	2.006	2.371	2.717	3.369
Average	0.176	0.815	1.903	2.423	2.859	3.243

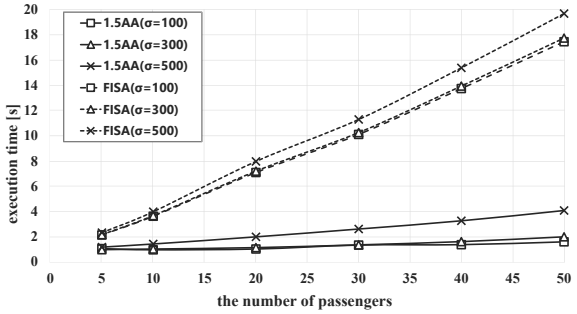


Figure 14: Time comparing with fixed-iteration SA

Figure 14 shows the execution time of the proposed method (1.5AA) compared to the fixed-iteration simulated annealing (SA); solid lines present the execution times of 1.5AA, and dotted lines show them of SA. Note that we omit the results when $\sigma = 200$ and $\sigma = 400$ for the readability of the results (they show the same patterns as the other results). Obviously, SA requires a huge execution time compared to 1.5AA, because the number of iterations is fixed to 70,000 to obtain good results regardless its execution time. However, as the results depicted in Figure 13, the total length of the route by 1.5AA becomes shorter than it obtained by SA, even the difference between their computational costs is significant.

6 CONCLUSION

In this study, we proposed an approximation algorithm to find a ride-sharing route considering the movement of passengers. Moreover, we have implemented the proposed system that finds and visually displays the ride-sharing route obtained by the proposed method (1.5AA) and evaluated the proposed algorithm comparing with a heuristic algorithm based on simulated annealing (SA).

In the evaluation experiments, we examined two variations of the execution of algorithm SA; a fixed-time execution (denoted by FTSA) and a fixed-iteration execution (denoted by FISA). As a result of the comparison with algorithm FTSA, even the two algorithm spent similar execution time, the proposed algorithm 1.5AA outperformed FTSA. Specifically, the total length of the ride-sharing route generated by 1.5AA was less than 40% of the average route length obtained by FTSA. The result also represented a significant improvement compared to both the worst case (about 65% reduction) and even the best case (about 54% reduction) of FTSA. Next we conducted the preliminary experiments to find the appropriate

number of iterations in SA for getting a good result within a reasonable computational time, and we found that 70,000 is the best balanced value between efficiency and computational time. Thus we fixed the number of iterations to 70,000 for algorithm FISA, and compared 1.5AA with FISA. As a result, even the execution time of FISA is at most about 11 times longer than it of 1.5AA, the resultant route length is reduced more than 20%. Compared to the best case of FISA, the ride-sharing route generated by 1.5AA is more than 10% shorter than it obtained by FISA.

In conclusion, our proposed method outperforms a heuristic algorithm in terms of both the efficiency (*i.e.,* the length of the resultant ride-sharing route) and the computational time. In particular, even when a large instance, 50 passengers who can move to the other locations within 500m, is given as an input set, the proposed method generates a efficient ride-sharing route within approximately 4 seconds. This execution time is fast enough from the practical viewpoint.

Our future works include an improvement by integrating nearby boarding positions of the passengers and consideration of the existence of one-way roads. Since pick-up and drop-off require a certain amount of time in ride-sharing, integrating nearby passenger boarding positions could reduce the total amount of time spent on these activities. Moreover, the proposed method consider a road network as a simple undirected graph. Hence, it is necessary to consider one-way roads (*i.e.,* arc) to realize more practical application.

As the many previous works, we can also consider the dynamic changes of passengers. This means that even in bus ride-sharing, unexpected situations can arise, like some passengers not being able to board or new passengers joining. Our proposed algorithm can rapidly recalculate the entire route from the current state in a short time by some revision; one valid approach is to change the bus's starting point to its current location and exclude places where passengers are already on board. This allows us to swiftly recalculate a new solution that's almost optimal for the current scenario.

ACKNOWLEDGEMENTS

This work was supported by JSPS KAKENHI Grant Numbers 21K 19766 and 23H03403.

REFERENCES

[1] Kamel Aissat and Ammar Oulamara. 2014. A Priori Approach Of Real-Time Ridesharing Problem With Intermediate Meeting Locations. *J. Artif. Intell. Soft Comput. Res.* 4, 4 (2014), 287–299. https://doi.org/10.1515/jaiscr-2015-0015

[2] Javier Alonso-Mora et al. 2017. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proc. Natl. Acad. Sci. USA* 114, 3 (2017), 462–467. https://doi.org/10.1073/pnas.1611675114

[3] Mohammad Asghari et al. 2016. Price-aware real-time ride-sharing at scale: an auction-based approach. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016*. ACM, 3:1–3:10. https://doi.org/10.1145/2996913.2996974

[4] Mohammad Asghari and Cyrus Shahabi. 2017. An On-line Truthful and Individually Rational Pricing Mechanism for Ride-sharing. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017*. ACM, 7:1–7:10. https://doi.org/10.1145/3139958.3139991

[5] Kanika Bathla et al. 2018. Real-Time Distributed Taxi Ride Sharing. In *21st International Conference on Intelligent Transportation Systems, ITSC 2018, Maui, HI, USA, November 4-7, 2018*. IEEE, 2044–2051. https://doi.org/10.1109/ITSC.2018.8569315

[6] Richard Bellman. 1962. Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM* 9, 1 (1962), 61–63. https://doi.org/10.1145/321105.321111

[7] Mandell Bellmore and George L. Nemhauser. 1968. The Traveling Salesman Problem: A Survey. *Oper. Res.* 16, 3 (1968), 538–558. https://doi.org/10.1287/opre.16.3.538

[8] Houssem E. Ben-Smida et al. 2016. Mixed Integer Linear Programming Formulation for the Taxi Sharing Problem. In *Smart Cities - First International Conference, Smart-CT 2016 (Lecture Notes in Computer Science, Vol. 9704)*. Springer, 106–117. https://doi.org/10.1007/978-3-319-39595-1_11

[9] Jon Louis Bentley. 1990. K-d Trees for Semidynamic Point Sets. In *Proceedings of the Sixth Annual Symposium on Computational Geometry, Berkeley, CA, USA, June 6-8, 1990*. ACM, 187–197. https://doi.org/10.1145/98524.98564

[10] Rainer E. Burkard et al. 1998. Well-Solvable Special Cases of the Traveling Salesman Problem: A Survey. *SIAM Rev.* 40, 3 (1998), 496–546. https://doi.org/10.1137/S0036144596297514

[11] Nicos Christofides. 2022. Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem. *Oper. Res. Forum* 3, 1 (2022). https://doi.org/10.1007/s43069-021-00101-z

[12] Jack Edmonds. 1965. Paths, Trees, and Flowers. *Canadian Journal of Mathematics* 17 (1965), 449–467. https://doi.org/10.4153/CJM-1965-045-4

[13] Robert W. Floyd. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5, 6 (1962), 345. https://doi.org/10.1145/367766.368168

[14] OpenStreetMap Foundation. 2004-2023. OpenStreetMap. https://openstreetmap.jp/map.

[15] Michael T. Goodrich and Roberto Tamassia. 2014. *Algorithm Design and Applications*. Wiley. https://www.wiley.com/en-us/Algorithm+Design+and+Applications-p-9781118335918

[16] Gregory Z. Gutin, Anders Yeo, and Alexey Zverovich. 2002. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. *Discret. Appl. Math.* 117, 1-3 (2002), 81–86. https://doi.org/10.1016/S0166-218X(01)00195-0

[17] Hirotaka Irie et al. 2017. Quantum Annealing of Vehicle Routing Problem with Time, State and Capacity. In *Quantum Technology and Optimization Problems - First International Workshop, QTOP@NetSys 2019 (Lecture Notes in Computer Science, Vol. 11413)*. Springer, 145–156. https://doi.org/10.1007/978-3-030-14082-3_13

[18] A. K. M. Mustafizur Rahman Khan et al. 2017. Ride-sharing is About Agreeing on a Destination. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017*. ACM, 6:1–6:10. https://doi.org/10.1145/3139958.3139972

[19] P. J. M. Laarhoven and E. H. L. Aarts. 1987. *Simulated Annealing: Theory and Applications*. Kluwer Academic Publishers, USA.

[20] Shuo Ma et al. 2013. T-share: A large-scale dynamic taxi ridesharing service. In *29th IEEE International Conference on Data Engineering, ICDE 2013*. IEEE Computer Society, 410–421. https://doi.org/10.1109/ICDE.2013.6544843

[21] L.A. Rastrigin. 1963. The Convergence of the Random Search Method in the External Control of Many-Parameter System. *Automation and Remote Control* 24 (1963), 1337–1342.

[22] Massobrio Renzo et al. 2014. A parallel micro evolutionary algorithm for taxi sharing optimization. In *VIII ALIO/EURO Workshop on Applied Combinatorial Optimization*. https://doi.org/10.13140/RG.2.1.3047.7925

[23] Kazuhiro Saito et al. 2021. Evaluation of Quantum Annealing for Vehicle Routing Problem (in Japanese). *IPSJ-TOD* 14, 1 (03 2021), 8–17.

[24] Mitja Stiglic et al. 2015. The benefits of meeting points in ride-sharing systems. *Transportation Research Part B: Methodological* 82 (2015), 36–53. https://doi.org/10.1016/j.trb.2015.07.025

[25] C. Storey. 1962. Applications of a hill climbing method of optimization. *Chemical Engineering Science* 17, 1 (1962), 45–52. https://doi.org/10.1016/0009-2509(62)80005-0

[26] Chi-Chung Tao and Chun-Ying Chen. 2007. Heuristic Algorithms for the Dynamic Taxipooling Problem Based on Intelligent Transportation System Technologies. In *Fourth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007, Volume 3*. IEEE Computer Society, 590–595. https://doi.org/10.1109/FSKD.2007.346

[27] Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Trans. Inf. Theory* 13, 2 (1967), 260–269. https://doi.org/10.1109/TIT.1967.1054010

[28] Takato Yoshida et al. 2019. Modeling and evaluating taxi ride-sharing for event trips (in Japanese). *Transaction on Mathematical Modeling and its Applications: TOM* 12, 2 (07 2019), 1–11. https://cir.nii.ac.jp/crid/1050845763318147456

[29] Yuki Yoshizuka et al. 2018. Performance Evaluation of Path Finding Algorithm for Ride-sharing Service (in Japanese). *The 32nd Annual Conference of the Japanese Society for Artificial Intelligence* JSAI2018 (2018), 4F1OS11c02–4F1OS11c02. https://doi.org/10.11517/pjsai.JSAI2018.0_4F1OS11c02