GDP: Greedy-based Dynamic Programming Algorithm
An Effective and Rational Solution to ODRP problems

Hongjie Fang, Peishen Yan, Chunyu Xue
Student ID {518030910150, 518030910094, 518021910698}
`{galaxies, 1050335889, Dicardo} @ sjtu.edu.cn`

Department of Computer Science,
Shanghai Jiao Tong University, Shanghai, China

Abstract. Work on adapting to rapid growth of passenger flow in urban transport system has long been hampered by a shortage of public transportation, since buses, taxis and some other traditional ways have revealed more and more inadequacy. The lack of efficient public transport with reasonable price prevents people from getting a pleasant and relaxed experience. In this paper, we first set several criteria of destination selection and propose our destination selection model after the analyses of several quantified features. Then we formally describe the Order Dispatching and Route Planning problem and prove that it is an NP-Complete problem using NP reduction. In order to relieve stress on passenger flow as well as maximizing total profit, we propose GDP-series algorithms as the solution to the problem, which can build a customized bus-booking platform and provide a relatively efficient solution. Then we design the pricing strategy according to transport cost and comparison with other public transports. The performance testing of our algorithms is based on the real-world trace database, and the numerical experiments and related analyses show the effectiveness and rationality of our algorithms and strategies.
Keywords: ODRP problem, GDP-series algorithms, NP reduction, pricing strategy.

1 Introduction

The performance improvement of public traffic condition has come under heated discussion among traffic administration organs in big cities. A lot of traffic management plans have been carried out, but due to the inevitable disadvantages of the current public transports, such as the inflexibility, relatively low utilization, and sometimes strictly constrained by time period, almost all current solutions can not satisfy the public request well. For instance, if you arrive at Shanghai Pudong International Airport during night period (from 23:00 to 02:00 on the next day), when the public transports have already been not in service, it may cost you quite a long time to wait for a taxi or car-hailing service with relatively high price. There is no doubt that passengers will have inconvenience and even terrible experience in this situation.

As a consequence, we begin to make efforts on designing a relatively optimal customized bus-booking platform, which can provide an efficient and feasible solution to handle problems that conventional public transports are not able to deal with. The main factors we have considered are: departure station and destination selection criteria, order dispatching algorithm, pricing strategy and performance test in numerical experiment.

The main difficulty is that we may have some limitation of the amount of buses or bus capacity, which means that we must selectively give up some orders and make the dispatching algorithm efficient enough to achieve our optimization goal. Meanwhile, we need to guarantee that the solution is not only relatively optimal, but solvable in acceptable time. If the algorithm could give an answer but would take a long time, absolutely it only has a very limited practical use. Therefore, we must try our best to balance both the relative optimization of solution and its running time.

Some restrictions for our scheduling algorithm must be considered. The basic idea is to carry passengers as many as possible to relieve the traffic pressure. However, in order to ensure the long-term running of our platform, we need to maximize the total profit as well, which may require us to ignore some orders that is relatively unreasonable. With Greedy-based Dynamic Programming algorithm, we could dynamically assign a group of passengers with related destinations onto the same bus, and find a relatively optimal route through the orders' destinations chosen from all destinations selected before. Since the Order Dispatching and Route Planning problem is an NP problem, we can only get a local optimal solution to approach global optimal solution as much as possible. Furthermore, we may need to consider the user satisfaction, balance of route assignment and some other more complicated factors.

By observation and research, we can numerically instantiate some parameters, and determine our pricing strategy and other restrictions. Then, we use our algorithm to handle real world data set, and figure out the relatively optimal solution with maximum profits and satisfying majority of passengers.

2 Definitions and Notations

Here we list several major definitions and notations, and you can check Appendix A for more details.

Notations	Definitions and Explanations
i	(usually) the passenger order number
j	(usually) the buses order number
b_j	the bus to which the i-th passenger is dispatched
c_r	the fix cost per bus
c_b	the fuel cost per kilometers
d_0	the departure station
d_i	the destination of passenger i
d	the realistic distance used for pricing
h	the average occupancy rate of a bus
m	the total number of buses
n	the total number of passengers
p_i	the bus fare price for passenger i
p_b	the base price of pricing strategy
p_c	the increasing parameter of bus fare price
B_j	the set of orders dispatched to bus b_j
C_r	the total fixed cost of all buses
C_b	the total fuel cost of all buses
D_j	the set of destinations of orders in B_j
$\mathbb{D}$	the set of all destinations
$\mathbb{D}_c$	the destination candidate set
L	the capacity limitation of each bus
W	the total income of our strategy
$\pi(D_j)$	a visit order for destination set D_j

Table 1. Major Definitions and Notations

3 Model Design

3.1 Assumptions

In our model design, we make the following assumptions.

1. The distance between the two stations is set as the realistic distance on city road network.
2. The bus system departs every half an hour, that is, the orders are processed every half an hour.
3. We assume that 23:00 to 6:00 on the next day is nighttime, since during this time the public transportation is usually out of service, and we assume that the rest of the time is daytime.

3.2 Destination Selection Model

In this section, we will introduce our destination selection model which can select the destinations effectively and efficiently based on given information, and we will make a detailed explanation to it.

First of all, we mainly regard the destinations of history order sets for taxis and online car-hailing services, as the main basis for the selection of the destinations. We can divide the map into several blocks and count the number of orders destined in each area. The more the number of orders is, the more likely this area is the destination of the passengers. In this way, we can directly choose the top k areas with the largest number of orders destined. This method is simple and efficient, but there are also some problems that can not be ignored:

– The destinations of taxi orders are mainly grouped in downtown area, so almost all of the top k areas with the largest number of taxi orders destined are in city center, which makes our destinations less representative;
– The destinations selected in this way may be too dense in some places, especially in downtown, which is not practical for the airport bus to have such a high destination station density;

– This method does not make full use of our information about the given situation, that is, we mainly focus on the passengers who arrive at airport during midnight period, when the public transports have already been not in service.

To consider more comprehensively, when a passenger arrives at airport at midnight when there is no available public transport service, what he (or she) wants to do might be directly heading for the hotels or homes and then having a good rest. Since the density of the residential area is almost the same in the city, we mainly focus on the hotel density of the city. Therefore, if we take the hotel density into the consideration of destination selection, the model will become more humanized and practical. Furthermore, if conditions permit, we might also consider other criteria of the selected destinations, such as road density nearby, congested status of roads and daily inflow of passengers.

Limited by the available data, our model only take the destinations of history order sets of taxis and the hotel density into considerations. Here is the formal statement of our Destination Selection Model.

First, we divide the map into many blocks with the size of about 200m × 200m, then we will collect the information about each block and estimate whether we will select it as a destination. Suppose there are totally s blocks, namely $1, 2, \cdots, s$. About the history taxi order data, we will count the number of orders o_i destined in block i, then we define the order weight W_i^o of the block i as follows.

$$W_i^o = \frac{1}{\alpha_o} \cdot \min(o_i, \alpha_o)$$

where α_o is a limitation of the number of orders determined by experiences in the experiments. About the hotel density data, we will count the number of hotels h_i destined in block i, then we define the hotel weight W_i^h of the block i as follows.

$$W_i^h = \text{Sigmoid}(h_i) = \frac{1}{1 + e^{-h_i}}$$

Here we explain why we use the sigmoid function[1] as the weight function. Consider about an given area, comparing to no hotels in this area, at least one hotel might make a great difference; but if there are already some hotels, more hotels can only make small differences. This is the reason of using sigmoid function, because it can create a great divergence between area that does not have a hotel and area that has a few hotels, and the differences between areas that has at least one hotels are not too big.

Finally, we combine this two weight linearly as follows and get the final weight of each block.

$$W_i = \beta_o W_i^o + \beta_h W_i^h, \qquad \text{for } i = 1, 2, \cdots, s$$

where β_o and β_h are coefficients determined in the experiments.

Now we can set a lower bound W_0 to the final weight of the blocks. The blocks whose weights are too small can not be the destination candidates. Only those blocks whose weights are greater than W_0 are the destination candidates. Therefore, we can express the destination candidate set $\mathbb{D}_c$ as follows.

$$\mathbb{D}_c = \{i \mid W_i > W_0, \quad i = 1, 2, \cdots, s\}$$

Then, we can apply K-Means Algorithm [1] on the geographic coordinates of the destinations in the candidate destination set $\mathbb{D}_c$ to merge some close destinations and lower the destination station density. Finally we can get our destination station set $\mathbb{D}$ containing exactly k destinations.

3.3 ODRP: The Order Dispatch and Route Planning Problem

In this section, we will propose the formal definitions of *the order dispatch and route planning problem* (also known as ODRP problem). Totally four versions of the ODRP problems are listed as follows.

Suppose there are n passengers numbered from 1 to n and m buses numbered from 1 to m at the departure station d_0, while the destination of passenger i is $d_i \in \mathbb{D}$ and the capacity limitation of each bus is L. Assume the distance between two station d_i and d_j can be represented as $dist(d_i, d_j)$. The bus fuel cost per kilometer is c_b, and the cost per route of a bus is a fixed number c_r. The bus fare price for passenger i from the departure station to station d_i is p_i. Let b_i denote the bus to which the order of passenger i is dispatched. Specially, if the order of passenger i is ignored, we will set $b_i = 0$.

[1] Sigmoid function: mathematical function which has a characteristic "S"-shaped curve. Here we choose the logistic function $f(x) = (1 + e^{-x})^{-1}$.

We define set B_j for bus j as the set of orders that are dispatched to bus j, that is,

$$B_j = \{i \mid b_i = j \ (i = 1, 2, \cdots, n)\}, \qquad \text{for } j = 0, 1, 2, \cdots, m$$

Since we set the b_i value of all ignored order to 0, B_0 is the set of ignored order.
Then we can define set D_j for bus j as the set of destinations of orders in B_j, that is,

$$D_j = \{d_i \mid i \in B_j\}, \qquad \text{for } j = 1, 2, \cdots, m$$

After that, we need to define a visit order for each destination set D_j, say $\pi(D_j)$. Then the specific route of bus j is basically defined as follows.

$$\mathit{Route}(\pi(D_j)) = \text{departure station} \to \pi_1(D_j) \to \pi_2(D_j) \to \cdots \to \pi_{|D_j|}(D_j) \to \text{departure station}$$

The income of the plan includes the ticket price of each passenger on the bus, and the cost of the plan includes the fuel cost of each bus and the fixed management fee of each bus that carries passengers. Therefore, we propose our formal definition of ODRP problems as follows.

Definition 1 (ODRP Version 1). *Suppose all the passengers need to be satisfied. Then the ODRP problem is to find a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j to maximize the total net profit.*

Definition 2 (ODRP Version 2). *Suppose we only need to satisfy a subset of the set of all passengers, and we can ignore no more than n_l orders without any loss, where n_l is a given integer. Then the ODRP problem is to find a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j to maximize the total net profit.*

Definition 3 (ODRP Version 3). *Suppose we only need to satisfy a subset of the set of all passengers, and we can ignore no more than n_l orders, where n_l is a given integer. Since ignoring order could bring passengers' dissatisfaction, we define the dissatisfaction cost of the plan as follows:*

$$c_d(n_i) = \alpha_{cd} n_i$$

where α_{cd} is a given non-negative constant coefficient and n_i stands for the number of ignored order. Then the ODRP problem is to find a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j to maximize the total net profit.

Definition 4 (ODRP Version 4). *Suppose we only need to satisfy a subset of the set of all passengers, and we can ignore no more than n_l orders, where n_l is a given integer. Since ignoring order could bring passengers' dissatisfaction, we define the dissatisfaction cost of the plan as follows:*

$$c_d(n_i) = \alpha_{cd} n_i$$

where α_{cd} is a given non-negative constant coefficient and n_i stands for the number of ignored order. Since the passenger will be displeased if the distance of the route is much longer than he (or she) expected, we also define a route length cost of each passenger i whose order is satisfied as follows:

$$c_l(i) = \alpha_{cl} \cdot [b_i \neq 0] \cdot \max\left(\frac{\textit{the distance of the real route of passenger } i}{\textit{the distance of the shortest route of passenger } i} - \alpha_p, 0\right)$$

where α_{cl} is a given non-negative constant coefficient and $\alpha_p = 1.5$ is a constant proportion determined by experiences. Then the ODRP problem is to find a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j to maximize the total net profit.

Here we also provide the decision versions of each ODRP problem as follows.

Definition 5 (ODRP Version 1, Decision Version). *Suppose all the passengers need to be satisfied. Given a profit argument k, then the ODRP problem is to find whether there is a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j such that the total net profit is no less than k.*

Definition 6 (ODRP Version 2, Decision Version). *Suppose we only need to satisfy a subset of the set of all passengers, and we can ignore no more than n_l orders without any loss, where n_l is a given integer. Given a profit argument k, then the ODRP problem is to find whether there is a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j such that the total net profit is no less than k.*

Definition 7 (ODRP Version 3, Decision Version). *Suppose we only need to satisfy a subset of the set of all passengers, and we can ignore no more than n_l orders, where n_l is a given integer. Since ignoring order could bring passengers' dissatisfaction, we define the dissatisfaction cost of the plan as follows:*

$$c_d(n_i) = \alpha_{cd} n_i$$

where α_{cd} is a given non-negative constant coefficient and n_i stands for the number of ignored order. Given a profit argument k, then the ODRP problem is to find whether there is a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j such that the total net profit is no less than k.

Definition 8 (ODRP Version 4, Decision Version). *Suppose we only need to satisfy a subset of the set of all passengers, and we can ignore no more than n_l orders, where n_l is a given integer. Since ignoring order could bring passengers' dissatisfaction, we define the dissatisfaction cost of the plan as follows:*

$$c_d(n_i) = \alpha_{cd} n_i$$

where α_{cd} is a given non-negative constant coefficient and n_i stands for the number of ignored order. Since the passenger will be displeased if the distance of the route is much longer than he (or she) expected, we also define a route length cost of each passenger i whose order is satisfied as follows:

$$c_l(i) = \alpha_{cl} \cdot [b_i \neq 0] \cdot \max\left(\frac{\textit{the distance of the real route of passenger } i}{\textit{the distance of the shortest route of passenger } i} - \alpha_p, 0\right)$$

where α_{cl} is a given non-negative constant coefficient and $\alpha_p = 1.5$ is a constant proportion determined by experiences. Given a profit argument k, then the ODRP problem is to find whether there is a valid order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j such that the total net profit is no less than k.

3.4 Programming Formulation of ODRP Problems

In this section, we will formulate all the ODRP problems as programmings. In the following statement, we will use Iverson bracket[1] $[P]$ many times. It will return 1 if the logistic statement P is true, and it will return 0 if the logistic statement P is false. Here are the specific details of the programmings.

We must make a restriction to the number of passengers on a bus, that is,

$$|B_j| \leq L, \quad \text{for } j = 1, 2, \cdots, m \tag{1}$$

Then, we can calculate our income W as follows.

$$W = \sum_{i=1}^{n} [b_i \neq 0] \cdot p_i$$

The first part of our cost is the fixed cost C_r, we can calculate it as follows.

$$C_r = c_r \cdot \sum_{j=1}^{m} [B_j \neq \varnothing]$$

The second part of our cost is the fuel cost C_b, which is related to the the route of each bus. We can represent is as follows.

$$C_b = c_b \cdot \sum_{j=1}^{m} \sum_{(d_x,d_y) \in Route(\pi(D_j))} dist(d_x, d_y)$$

[1] Iverson bracket: a notation that converts any logical proposition into 0 or 1.

1. **(ODRP Version 1)** In ODRP Version 1, every passenger needs be satisfied, so we must make the following restriction.
$$b_i \neq 0, \quad \text{for } i = 1, 2, \cdots, n \tag{2}$$
Therefore, we can formulate the following programming.
$$\begin{aligned} \max \quad & W - C_r - C_b \\ \text{s.t.} \quad & \text{Constraint } (1)(2) \\ & b_i \in \{0, 1, \cdots, m\}, \quad \text{for } i = 1, 2, \cdots, n \\ & \pi(D_j) \text{ permutation of } D_j, \quad \text{for } j = 1, 2, \cdots, m \end{aligned}$$

2. **(ODRP Version 2)** In ODRP Version 2, we can ignore the passenger without any cost, but we can ignore at most n_l order, so we can make the following restriction.
$$\sum_{i=1}^{n} [b_i = 0] \leq n_l \tag{3}$$
Therefore, we can formulate the following programming.
$$\begin{aligned} \max \quad & W - C_r - C_b \\ \text{s.t.} \quad & \text{Constraint } (1)(3) \\ & b_i \in \{0, 1, \cdots, m\}, \quad \text{for } i = 1, 2, \cdots, n \\ & \pi(D_j) \text{ permutation of } D_j, \quad \text{for } j = 1, 2, \cdots, m \end{aligned}$$

3. **(ODRP Version 3)** In ODRP Version 3, we can ignore no more than n_l orders, but ignoring order could bring dissatisfaction cost, which is the third part of our cost. We can represent it as follows.
$$C_d = c_d\left(\sum_{i=1}^{n} [b_i = 0]\right) = \alpha_{cd} \cdot \sum_{i=1}^{n} [b_i = 0]$$
Therefore, we can formulate the following programming.
$$\begin{aligned} \max \quad & W - C_r - C_b - C_d \\ \text{s.t.} \quad & \text{Constraint } (1)(3) \\ & b_i \in \{0, 1, \cdots, m\}, \quad \text{for } i = 1, 2, \cdots, n \\ & \pi(D_j) \text{ permutation of } D_j, \quad \text{for } j = 1, 2, \cdots, m \end{aligned}$$

4. **(ODRP Version 4)** In ODRP Version 4, we should take the route length cost of each passenger into consideration, which is the last part of our cost. We can represent it as follows.
$$C_r = \sum_{i=1}^{n} c_l(i) = \alpha_{cl} \cdot \sum_{i=1}^{n} [b_i \neq 0] \cdot \max\left(\frac{\text{the distance of the real route of passenger } i}{\text{the distance of the shortest route of passenger } i} - \alpha_p, 0\right)$$
Therefore, we can formulate the following programming.
$$\begin{aligned} \max \quad & W - C_r - C_b - C_d - C_r \\ \text{s.t.} \quad & \text{Constraint } (1)(3) \\ & b_i \in \{0, 1, \cdots, m\}, \quad \text{for } i = 1, 2, \cdots, n \\ & \pi(D_j) \text{ permutation of } D_j, \quad \text{for } j = 1, 2, \cdots, m \end{aligned}$$

It's also worthy to mention the method to transform $\pi(D_j)$ into a series of variables and constraints. It's actually a special case of TSP problem[1]. Take TSP problem[1] as an example, we use binary variables $x_{i,j}$ to denote whether the path between city i and city j is in the route. Then we can make the following constraints. The first constraint guarantees that the route is a cycle, and the second constraint guarantees that there is only one cycle.

$$\begin{aligned} & \sum_j x_{i,j} = \sum_j x_{j,i} = 1, && \text{for all city } i \\ & \sum_{i \in S} \sum_{j \in S} x_{i,j} \leq |S| - 1, && \text{for all subset } S \text{ of the city set satisfying } 2 \leq |S| \leq \text{number of cities} \end{aligned}$$

[1] TSP problem: The travelling salesman problem is to find a minimum weight Hamilton cycle in a graph.

3.5 Difficulty of ODRP Problems

Here we can claim that these four versions of ODRP problem are all NP-Complete problems. And their proof processes are as follows.

1. ODRP Version 1 is an NP-Complete problem.

   Lemma 1. *ODRP Version 1 is an NP problem.*

   Proof. Here are the certificate and certifier of the ODRP Version 1.
   – **Certificate.** An order dispatch plan b_i and the corresponding visit orders $\pi(D_j)$ of each bus j.
   – **Certifier.** Check if the plan is valid and comparing the total net profit of the plan with k.
   Given a certificate t, we can implement an polynomial time certifier by calculating income and cost and then check whether the total net profit of the plan is no less than k with the formula $profit = income - cost$. Therefore, ODRP Version 1 is an NP problem. □

   Lemma 2. *TSP $\leq_p$ ODRP Version 1.*

   Proof. Given an instance $\varPhi$ of TSP which contains n cities, we construct an instance $\varSigma$ of ODRP Version 1 containing n orders whose destinations are $d_1, d_2, \cdots, d_n$ and corresponding bus fare price is $p_1, p_2, \cdots, p_n$, m buses with capacity limitation L and certain fixed cost c_r, fuel cost per kilometer c_b, which is satisfying if and only if $\varPhi$ is satisfiable.
   Construction. First we construct n destinations $d_1, d_2, \cdots, d_n$ corresponding to n cities $ct_1, ct_2, \cdots, ct_n$ in $\varPhi$ respectively, and let $m = 1$ and $L = n$. At the same time, we announce that the fuel cost per kilometer $c_b = 1$, the distance between d_i and d_j is equal to the distance between ct_i and ct_j, and bus fix cost $c_r = 0$. We also set the ticket price of every passenger to 0, that is, $p_i = 0$ for all i. Finally, we set the limitation k of $\varSigma$ to $-k'$, where k' is the corresponding argument in $\varPhi$ (in TSP problem we should check if the total of path length is no more than k').
   Claim and Proof. $\varSigma$ is satisfiable if and only if $\varPhi$ is satisfiable. Here is the specific proof.
   – ($\Longleftarrow$) Given a satisfying Path selection of $\varPhi$, we construct an order dispatch plan and visit orders as follows:
     - Since $m = 1$ and $L = n$, we dispatch all orders to the only bus.
     - And the visit order of the only bus $\pi(D_1)$ corresponds to the visit order $\pi(\{ct_i | 1 \leq i \leq n\})$ of the instance $\varPhi$ of TSP. In other words, if $(ct_i, ct_j) \in Path_{\varPhi}$ then $(d_i, d_j) \in Route_{\varSigma}(\pi(D_1))$.

     Define $P_{i,j}$ and $C_{i,j}$ as follows:

$$P_{i,j} = \begin{cases} dist(ct_i, ct_j), & \text{if } (ct_i, ct_j) \in Path_{\varPhi} \\ 0, & \text{if } (ct_i, ct_j) \notin Path_{\varPhi} \end{cases}$$

$$C_{i,j} = \begin{cases} -dist(d_i, d_j), & \text{if } (d_i, d_j) \in Route_{\varSigma}(\pi(D_1)) \\ 0, & \text{if } (d_i, d_j) \notin Route_{\varSigma}(\pi(D_1)) \end{cases}$$

     Then if path selection is satisfying, we have $\sum_{i \neq j} P_{i,j} \leq k'$, and because we construct the visit Order $\pi(D_1)$ corresponding to the visit order $\pi(\{ct_i | 1 \leq i \leq n\})$ of TSP, then $(ct_i, ct_j) \in Path_{\varPhi}$ if and only if $(d_i, d_j) \in Route_{\varSigma}(\pi(D_1))$. Thus, we have:

$$\begin{aligned} \text{Total profit} &= \sum_{i \neq j} C_{i,j} \\ &= \sum_{(d_i, d_j) \in Route_{\varSigma}(\pi(D_1))} -dist(d_i, d_j) \\ &= -\sum_{(ct_i, ct_j) \in Path(\varPhi)} dist(ct_i, ct_j) \\ &= -\sum_{i \neq j} P_{i,j} \\ &\geq -k' = k \end{aligned}$$

     Hence, $\varSigma$ is satisfiable if $\varPhi$ is satisfiable.

- ($\Longrightarrow$) Given a satisfying solution to ODRP Version 1 instance Σ constructed by Φ, then according to those information we mentioned above, if $(ct_i, ct_j) \in Path_\Phi$ then $(d_i, d_j) \in Route_\Sigma(\pi(D_1))$, and we have $\sum_{i\neq j} C_{i,j} \geq k$. Thus, we have:

$$\begin{aligned}\text{Total path length} &= \sum_{i\neq j} P_{i,j} \\ &= \sum_{(ct_i,ct_j)\in Path(\Phi)} dist(ct_i, ct_j) \\ &= -\sum_{(d_i,d_j)\in Route_\Sigma(\pi(D_1))} -dist(d_i, d_j) \\ &= -\sum_{i\neq j} C_{i,j} \\ &\leq -k = k'\end{aligned}$$

  Hence, Φ is satisfiable if Σ is satisfiable.

□

Theorem 1. *ODRP Version 1 is an NP-Complete problem.*

Proof. According to Lemma 1, we know that ODRP Version 1 is an NP problem. Then according to Lemma 2, we know that TSP can be reduced to ODRP Version 1. Since TSP is an NP-Complete problem, ODRP Version 1 should be an NP-complete problem. □

2. ODRP Version 2 is an NP-Complete problem.

Lemma 3. *ODRP Version 2 is an NP problem.*

Proof. Here are the certificate and certifier of the ODRP Version 2.
- **Certificate.** An order dispatch b_i and the corresponding visit orders $\pi(D_j)$ of each bus j.
- **Certifier.** Check if the plan is valid, compare the total net profit of the plan with k and make sure that there are no more than n_l orders ignored.

Given a certificate t, we can implement an polynomial time certifier by calculating income and cost and then check whether the total net profit of the plan is no less than k with the formula $profit = income - cost$. And then, check whether the number of ignored orders is no more than n_l, which is $O(1)$ time cost. Therefore, ODRP Version 2 is a NP problem. □

Lemma 4. *ODRP Version 1 $\leq_p$ ODRP Version 2.*

Proof. Given an instance Φ of ODRP Version 1 which contains n orders whose destinations are $d_1, d_2, \cdots, d_n$ and corresponding bus fare price is $p_1, p_2, \cdots, p_n$, m buses with capacity limitation L and certain fixed cost c_r, fuel cost per kilometer c_b. Then we construct an instance Σ of ODRP Version 2 which is satisfying if and only if Φ is satisfiable.

Construction. First we construct Σ whose parameters included in Version 1 are consistent, and let $n_l = 0$, which means no order can be ignored. And finally, we set the limitation k' of Σ to k, where k is the corresponding argument in Φ (in ODRP Version 1 we should check if the total of net profit is no less than k).

Claim and Proof. Σ is satisfiable if and only if Φ is satisfiable. Here is the specific proof.
- ($\Longleftarrow$) Given an order dispatch plan and visit orders of Φ which is satisfying, we construct an order dispatch plan b'_i and visit orders $\pi'(D'_j)$ of Σ as follows:
  - Let $b'_i = b_i$ for all i, which means the i-th order is dispatched to the exactly same bus in Σ as it is dispatched to in Φ. Therefore, we can derive that $D'_j = D_j$ for all j.
  - Let $\pi'(D'_j) = \pi(D_j)$ for all j, which means the visit order of the j-th bus in Σ is the same as the visit order of the corresponding bus in Φ.

  Obviously, when $n_l = 0$, all the things are so identical that the income and cost are the same in both Σ and Φ, which means the total net profit in Σ is no less than k' if the total net profit in Φ is no less than k. Hence, Σ is satisfiable if Φ is satisfiable.
- ($\Longrightarrow$) Given an order dispatch plan and visit orders of Σ which is satisfying, we construct an order dispatch plan b_i and visit orders $\pi(D_j)$ of Φ as follows:

- Let $b_i = b'_i$ for all i, which means the i-th order is dispatched to the exactly same bus in Φ as it is dispatched to in Σ. Therefore, we can derive that $D_j = D'_j$ for all j.
- Let $\pi(D_j) = \pi'(D'_j)$ for all j, which means the visit order of the j-th bus in Φ is the same as the visit order of the corresponding bus in Σ.

Similar to the reason mentioned above, we have that the total net profit in Φ is no less than k if the total net profit in Σ is no less than k'. Hence, Φ is satisfiable if Σ is satisfiable.

□

Theorem 2. *ODRP Version 2 is an an NP-Complete problem.*

Proof. According to Lemma 3, we know that ODRP Version 2 is an NP problem. We know that ODRP Version 1 can be reduced to ODRP Version 2 from Lemma 4. Since ODRP Version 1 is an NP-Complete problem according to Theorem 1, ODRP Version 2 is an NP-complete problem. □

3. ODRP Version 3 is NP-Complete problem.

Lemma 5. *ODRP Version 3 is an NP problem.*

Proof. Similar to the previous processes, here are the certificate and certifier of ODRP Version 3.
- **Certificate.** An order dispatch b_i and the corresponding visit orders $\pi(D_j)$ of each bus j.
- **Certifier.** Check if the plan is valid, compare the total net profit of the plan with k and make sure that there are no more than n_l orders ignored.

Given a certificate t, we can implement an polynomial time certifier by calculating income and cost (here we should consider the dissatisfaction cost in it) and then check whether the total net profit of the plan is no less than k with the formula $profit = income - cost$. And then, check whether the number of ignored orders is no more than n_l, which is $O(1)$ time cost. Therefore, ODRP Version 3 is a NP problem. □

Lemma 6. *ODRP Version 2* $\leq_p$ *ODRP Version 3.*

Proof. Given an instance Φ of ODRP Version 2 which contains n orders whose destinations are $d_1, d_2, \cdots, d_n$ and corresponding bus fare price is $p_1, p_2, \cdots, p_n$, m buses with capacity limitation L and certain fixed cost c_r, fuel cost per kilometer c_b, the largest number of ignored orders n_l. Then we construct an instance Σ of ODRP Version 3 which is satisfying if and only if Φ is satisfiable.
Construction. First we construct Σ whose parameters included in Version 2 are consistent, and let $\alpha_{cd} = 0$, which means ignoring orders won't cost. And finally, we set the limitation k' of Σ to k, where k is the corresponding argument in Φ (in ODRP Version 1 we should check if the total of net profit is no less than k).
Claim and Proof. Σ is satisfiable if and only if Φ is satisfiable. Here is the specific proof.
- ($\Longleftarrow$) Given an order dispatch plan and visit orders of Φ which is satisfying, we construct an order dispatch plan b'_i and visit orders $\pi'(D'_j)$ of Σ as follows:
  - Let $b'_i = b_i$ for all i, which means the i-th order is dispatched to the exactly same bus in Σ as it is dispatched to in Φ or both of them are ignored. Therefore, we can derive that $D'_j = D_j$ for all j.
  - Let $\pi'(D'_j) = \pi(D_j)$ for all j, which means the visit order of the j-th bus in Σ is the same as the visit order of the corresponding bus in Φ.

  Obviously, when $\alpha_{cd} = 0$, all the things are so identical that the income and cost are the same in both Σ and Φ, which means the total net profit in Σ is no less than k' if the total net profit in Φ is no less than k. Hence, Σ is satisfiable if Φ is satisfiable.
- ($\Longrightarrow$) Given an order dispatch plan and visit orders of Σ which is satisfying, we construct an order dispatch plan b_i and visit orders $\pi(D_j)$ of Φ as follows:
  - Let $b_i = b'_i$ for all i, which means the i-th order is dispatched to the exactly same bus in Φ as it is dispatched to in Σ or both of them are ignored. Therefore, we can derive that $D_j = D'_j$ for all j.
  - Let $\pi(D_j) = \pi'(D'_j)$ for all j, which means the visit order of the j-th bus in Φ is the same as the visit order of the corresponding bus in Σ.

  Similar to the reason mentioned above, we have that the total net profit in Φ is no less than k if the total net profit in Σ is no less than k'. Hence, Φ is satisfiable if Σ is satisfiable.

□

Theorem 3. *ODRP Version 3 is an NP-Complete problem.*

Proof. According to Lemma 5, we know that ODRP Version 3 is an NP problem. We know that ODRP Version 2 can be reduced to ODRP Version 3 from Lemma 6. Since ODRP Version 2 is an NP-Complete problem according to Theorem 2, ODRP Version 3 is an NP-complete problem. □

4. ODRP Version 4 is an NP-Complete problem.

Lemma 7. *ODRP Version 4 is an NP problem.*

Proof. Similar to the previous processes, here are the certificate and certifier of ODRP Version 4.
- **Certificate.** An order dispatch b_i and the corresponding visit orders $\pi(D_j)$ of each bus j.
- **Certifier.** Check if the plan is valid, compare the total net profit of the plan with k and make sure that there are no more than n_l orders ignored.

Given a certificate t, we can implement an polynomial time certifier by calculating income and cost (here we should consider the dissatisfaction cost $c_d(n_i)$ and route length cost $c_l(i)$ in it) and then check whether the total net profit of the plan is no less than k with the formula $profit = income - cost$. And then, check whether the number of ignored orders is no more than n_l, which is $O(1)$ time cost. Therefore, ODRP Version 4 is a NP problem. □

Lemma 8. *ODRP Version 3 $\leq_p$ ODRP Version 4.*

Proof. Given an instance Φ of ODRP Version 3 which contains n orders whose destinations are $d_1, d_2, \cdots, d_n$ and corresponding bus fare price is $p_1, p_2, \cdots, p_n$, m buses with capacity limitation L and certain fixed cost c_r, fuel cost per kilometer c_b, the largest number of ignored orders n_l, the dissatisfaction coefficence α_{cd}. Then we construct an instance Σ of ODRP Version 4 which is satisfying if and only if Φ is satisfiable.
Construction. First we construct Σ whose parameters included in Version 2 are consistent, and let the coefficence of route length cost $\alpha_{cl} = 0$, which means passengers' displeasement because of long route length won't cost. And finally, we set the limitation k' of Σ to k, where k is the corresponding argument in Φ (in ODRP Version 1 we should check if the total of net profit is no less than k).
Claim and Proof. Σ is satisfiable if and only if Φ is satisfiable. Here is the specific proof.
- ($\Longleftarrow$) Given an order dispatch plan and visit orders of Φ which is satisfying, we construct an order dispatch plan b'_i and visit orders $\pi'(D'_j)$ of Σ as follows:
  - Let $b'_i = b_i$ for all i, which means the i-th order is dispatched to the exactly same bus in Σ as it is dispatched to in Φ or both of them are ignored. Therefore, we can derive that $D'_j = D_j$ for all j.
  - Let $\pi'(D'_j) = \pi(D_j)$ for all j, which means the visit order of the j-th bus in Σ is the same as the visit order of the corresponding bus in Φ.

  Obviously, when $\alpha_{cl} = 0$, all the things are so identical that the income and cost are the same in both Σ and Φ, which means the total net profit in Σ is no less than k' if the total net profit in Φ is no less than k. Hence, Σ is satisfiable if Φ is satisfiable.
- ($\Longrightarrow$) Given an order dispatch plan and visit orders of Σ which is satisfying, we construct an order dispatch plan b_i and visit orders $\pi(D_j)$ of Φ as follows:
  - Let $b_i = b'_i$ for all i, which means the i-th order is dispatched to the exactly same bus in Φ as it is dispatched to in Σ or both of them are ignored. Therefore, we can derive that $D_j = D'_j$ for all j.
  - Let $\pi(D_j) = \pi'(D'_j)$ for all j, which means the visit order of the j-th bus in Φ is the same as the visit order of the corresponding bus in Σ.

  Similar to the reason mentioned above, we have that the total net profit in Φ is no less than k if the total net profit in Σ is no less than k'. Hence, Φ is satisfiable if Σ is satisfiable.

□

Theorem 4. *ODRP Version 4 is an NP-Complete problem.*

Proof. According to Lemma 7, we know that ODRP Version 4 is an NP problem. We know that ODRP Version 3 can be reduced to ODRP Version 4 from Lemma 8. Since ODRP Version 3 is an NP-Complete problem according to Theorem 3, ODRP Version 4 is an NP-complete problem. □

3.6 Pricing Strategy

In this section, we will introduce our pricing strategy which can dynamically determine the price of each order based on the destination selected. More details are as follows.

First, the main factor we price is the linear distance from the departure station to the destination the passenger has selected. That is, the further the destination is, the higher the price will be, and if some passengers select the same destination, the price they should pay will be the same.

The main principles that we build our pricing design are:

– To maximize the total income as much as possible.
– To make our bus-booking system more competitive on pricing strategy compared to taxi or sharing car, which can make our system more attractive and welcomed.
– Dynamically adjusted to to better improve the experience of passengers according to the feedback offered, since our initial pricing strategy may not be totally suitable in the transport market.
– Our pricing strategy should be able to differ the different time period (day time or night time).

To meet these principles, we have researched the taxi pricing strategies in various cities, and build our bus-booking pricing strategy based on it, which will linearly increased based on the as-the-crow-flies distance. Meanwhile, to better satisfy the **internal cost**, like the driver cost, vehicle maintenance cost and platform operation cost, we set a base price of our pricing function. Of course, we make our pricing lower than the price of taking taxi, but in the same time not much lower in order to get more income.

What's more, according to the definition of economic principles, the **external cost** can be divided into three aspects:

– External influences related to actual transport activity, like air, noise and water pollution.
– Vehicle related externalities, like the traffic jam caused by the bus.
– Externalities related to infrastructure, like the barrier effect to community.

In more symbolic expression, our price of each passenger i is only determined by the linear distance between the departure station and destination station, that is,

$$p_i = p(dist(d_0, d_i))$$

where the price function $p(\cdot)$ should be an increasing function. For convenience, we assume the price of taxi $p_t(d)$ for distance d is as follows.

$$p_t(d) = \begin{cases} p_{tb}, & 0 < d \leq d' \\ p_{tb} + p_{tc} \cdot (d - d'), & d > d' \end{cases}$$

where p_{tb}, p_{tc} and d' are given arguments, representing taxi's base price, the increasing parameter of taxi's price and the distance suitable for taxi's base price respectively. Therefore, our function $p(\cdot)$ should meet the following requirement.

$$p(d) \leq P_r \cdot p_t(d), \qquad 0 < P_r < 1, \text{for all } d > d' + \delta$$

where δ is a small constant and P_r is relative pricing ratio between taxi and bus, which can be considered with many factors from both internal and external cost. According to a price research [2], the prices of taxi, bus and subway has a corresponding relation, as Table 2 shown.

taxi	bus	subway
10 yuan	2.5 yuan ∼ 3.4 yuan	2.9 yuan ∼ 4.2 yuan

Table 2. The corresponding relation among prices of taxi, bus and subway

The relative pricing ratio between taxi and bus is around 4, that is, $P_r = 4$. According to the principles above, we can assume that the pricing ratio p_{bat} between traditional bus and our bus-booking system is 2, which means the price per kilometer when using our platform is twice as expensive as the price per kilometer when taking a traditional bus. Furthermore, the price of taking taxi per kilometer is also twice as expensive as the price per kilometer when using our platform.

To make the the price function $p(d)$ easy to understand, we can set it to this form.

$$p(d) = p_b + p_c \cdot d, \qquad d > 0$$

where p_b is our base price and p_c is the increasing parameter of our bus.

We can dynamically adjust these arguments to make it more suitable in the market. According to the traditional bus average internal cost research [3], we can define the following parameters in Table 3.

Notations	Descriptions
c_b	the fuel cost per kilometer
c_r	the total fixed cost per bus per time
c_{mtc}	the maintenance cost per (traditional) bus per hour
h	the average bus occupancy rate
d_{avg}	the average distance per hour for each bus
p_{bat}	the pricing ratio between traditional bus and our bus-booking system
p_{pr}	the profit rate

Table 3. The price parameters and their initial settings

For the increasing parameter of price p_c, we need to consider the fuel cost and the maintenance cost. Let c_{mtc} denote the maintenance cost for a bus per hour, and let d_{avg} denote the average distance per hour for each bus, then the maintenance cost per kilometer can be represented as c_{mtc}/d_{avg}. As a company, we need to make profits, therefore we need to multiply the total cost by $(1+p_{pr})$, where p_{pr} is the profit rate. Then we can split the bill among all passengers, and we assume that a bus has hL passengers on average, where h is the average bus occupancy rate. Finally, up to now we get the rational increasing parameter of a traditional bus, and for our bus platform, we need to multiply the result by a relative pricing ratio p_{bat}. Therefore, we get the final expression of the increasing parameter p_c as Eqn. (4) shown.

$$p_c = \frac{(1+p_{pr})p_{bat}}{hL}\left(c_b + \frac{c_{mtc}}{d_{avg}}\right) \tag{4}$$

For the base price parameter p_b, we also need to split the bills among all passengers. This time we do not need to multiply p_{bat} because we consider our buses in the fixed cost c_r. Therefore, we get the final expression of the base price parameter p_b as Eqn. (5) shown.

$$p_b = \frac{(1+p_{pr})c_r}{hL} \tag{5}$$

4 Algorithm Design

In this section, we will take ODRP Version 2 as an example and propose our algorithm designs, to make the algorithm simple and more feasible, we set n_l to n, that is, we can ignore any number of orders.

The ODRP problem can be divided into two parts: the order dispatching part and the route planning part. Once we make the order dispatching decisions, the route planning for each bus j is basically the distinguished TSP problem. We will introduce our algorithms for TSP problem first, and then propose the full algorithm for the ODRP problem.

4.1 Algorithms for TSP Problem

In this section, we will state the definition of TSP problem and introduce two algorithms to solve the problem, which use Dynamic Programming and Genetic Algorithm respectively, and then we can combined the algorithms together and form our final algorithm for TSP problem.

The formal statement of the TSP problem is: given an edge-weighted graph, find a Hamilton cycle with the minimum weights. In our ODRP problem, the vertices in the graph are the destinations, and the weight of edge connected vertex d_i and vertex d_j is $dist(d_i, d_j)$. Suppose we want to plan a route for bus j. Since the cost of bus j is only determined by the route length and some constant parameters, we want to design a cyclic route which can visit all the destinations in $D_j \in \mathbb{D}$ with the minimum route length.

We combine two algorithms together to solve the TSP problem corresponding to the small-data situation and the big-data situation respectively. We will set a threshold of the number of vertices to separate the situations. The detailed introductions of the algorithms are as follows.

Dynamic Programming Algorithm for TSP Problem Since we want to find a Hamilton cycle for bus j, we can start from any vertex, so we choose the departure station d_0 as the start vertex for convenience. For bus j, the destination set is D_j, which means we should visit all the destinations in D_j in the cyclic route.

Let $f(S, d_x)$ denote the minimum route length starting from d_0 and currently in destination $d_x \in D_j$ under the condition that the stations in $S \subseteq D_j \cup \{d_0\}$ is already visited. Obviously we require $d_0 \in S$ and $d_x \in S$. Therefore, we can enumerate the last destination we visited and choose the route which has the minimum length. Therefore, the recurrence is as follows.

$$f(S, d_x) = \begin{cases} 0, & d_x = d_0 \text{ and } S = \{d_0\} \\ \min\limits_{d_y \in S \backslash \{d_x\}} \{f(S \backslash \{d_x\}, d_y) + dist(d_y, d_x)\}, & d_0 \in S \text{ and } d_x \in S \\ +\infty, & \text{otherwise} \end{cases} \tag{6}$$

Then, the minimum length of the cyclic route of bus j is as follows.

$$\min_{d_x \in D_j} \{f(D_j \cup \{d_0\}, d_x) + dist(d_x, d_0)\}$$

Here is the pseudo-code of the algorithm.

Algorithm 1: TSP Problem - Dynamic Programming Solution

```
Input: The destination set $D_j$, the departure station $d_0$, and the distance between
       each pair of stations $dist[\cdot, \cdot]$.
Output: Minimum length of the cyclic route of bus $j$.
1  $f[\cdot, \cdot] \leftarrow +\infty$;
2  $f[\{d_0\}, d_0] = 0$;
3  foreach $S \in D_j \cup \{d_0\}$ do
4      foreach $d_x \in S$ do
5          foreach $d_y \in S \backslash \{d_x\}$ do
6              $f[S, d_x] \leftarrow \min(f[S, d_x], f[S \backslash \{d_x\}, d_y] + dist[d_y, d_x])$;
7  $ans \leftarrow +\infty$;
8  foreach $d_x \in D_j$ do
9      $ans \leftarrow \min(ans, f[D_j \cup \{d_0\}, d_x] + dist[d_x, d_0])$;
10 return $ans$;
```

The specific route can be easily generated by recording the transitions in the dynamic programming. We do not discuss it further here, and you can refer to the source code of the implementation for details.

Time Complexity: $O(2^k k^2)$, where the destination number k is the upper bound of $|D_j|$.

Genetic Algorithm for TSP Problem Since we want to find a Hamilton cycle for bus j, we can use a permutation π of $D_j \cup \{d_0\}$ to represent the route. Therefore, we can use the Improved Genetic Algorithm: GA-EO Algorithm [4] to solve the TSP problem by providing a relatively optimal cyclic route.

In the beginning, we randomly choose N permutations as the initial populations. Then we simulate the crossing over and the mutation of the genes, which is also the permutation. After about G generations, we will get the relatively optimal route. We also use the extremal optimization to prevent getting stuck in the local optimal solution. Therefore, we are likely to find the relatively optimal route.

Therefore, the main procedure of the GA-EO algorithm is as follows.

- **Step 1.** Randomly choose N permutations as the initial populations;
- **Step 2.** Select the best population and use it to update the final route;
- **Step 3.** Combine the best population with the rest of the populations, and perform random mutations to get the populations of next generation;
- **Step 4.** Randomly combine two populations together, and perform random mutations to get the populations of next generations.
- **Step 5.** Perform extremal optimization to prevent getting stuck in the local optimal solution;
- **Step 6.** Return to Step 2 until the number of generations reaches G;
- **Step 7.** Output the final route.

The algorithm can not grant to get the optimal route, but the final route is relatively optimal. Its time complexity is much less than the dynamic programming approach, so it can be regarded as a substitution approach when $|D_j|$ is large.

Time Complexity: $O(kGN^2)$, where the destination number k is the upper bound of $|D_j|$, N is a constant number of populations and G is a constant number of generations. We can choose G and N properly to make the time complexity acceptable.

Combined Algorithm for TSP Problem We combined two algorithms together and set a threshold to determine which algorithm should we use in different situations. For efficiency considerations, we set the parameters N and G in Genetic Algorithm to 100 and 200 respectively, and we set the threshold $D = 12$. If $|D_j| > D$, we will use the Genetic Algorithm to get a relatively optimal route; if $|D_j| \leq D$, we will use the Dynamic Programming approach to get the optimal route. Hence, the time complexity of every situation is quite acceptable.

4.2 Algorithms for ODRP Problem

In this section, we will propose several algorithms to solve the ODRP Problem. First we propose two optimal algorithms with different time complexity for different considerations, then we will introduce our final algorithm to the ODRP problem: GDP (Greedy-based Dynamic Programming Algorithm). Finally, we will give a direction for further improvements of GDP.

Optimal Algorithm 1 We can enumerate every possible situation of b_i, and there are at most $(m+1)^n$ situations (including ignored orders). For every situation, we can use our dynamic programming approach for TSP problem to compute the optimal route for every bus.

Time complexity: $O\left((m+1)^n \cdot 2^k k^2\right)$. Actually, this upper bound might be quite loose. First, k is just an upper bound of $|D_j|$, and the TSP problems here can hardly reach the upper bound. Second, a quite big part of the enumeration result is invalid, and we can eliminate it in the enumerating process to save some time. Therefore, we can use some simple optimizations to make the program run faster. But still, the optimal algorithm has a exponential time complexity and runs pretty slowly.

Optimal Algorithm 2 We can enumerate the stations of every bus first. There are totally 2^{mk} situations and we can use an $m \times k$ 0/1-matrix to represent them (0 means that the bus does not go to the station; and 1 means that the bus goes to the station). Once we determine the stations of every bus, we can use our dynamic programming approach for TSP problem to compute the optimal route for every bus. As for the order dispatching problem, we can use build a linear integer programming model and use some special tools, such as CPLEX[1], to solve it. Here we state the programming model as follows.

– **Variables.** The number of passengers heading for every station in every bus, totally mk integer variables ranging from 0 to L. Specially, we assume that if the bus is not heading for the station, then the corresponding variable can be arbitrary and is not necessary to be 0. In the following constraint, before using the variables, we just need to multiply the variables by the corresponding item in the enumeration matrix.
– **Constraint 1.** The number of passengers in a bus should be less than L;
– **Constraint 2.** The number of passengers heading for the same destination should not exceed the total number of passengers heading for that destination in all orders.
– **Maximization.** Since the cost is already determined by the result of solving TSP problem, we just need to maximize the total income.

Time complexity: $O\left(2^{mk}(2^k k^2 + LIP(mk))\right)$, where $LIP(n)$ means the time complexity of solving a linear integer programming with n variables. Also, the problem has a exponential time complexity and is quite slow.

Comparisons between two Optimal Algorithms We have introduced two optimal algorithm in the previous context. If you read carefully, you will notice that they have different time complexities and they are suitable for different situations. Optimal Algorithm 1 is suitable for those data which has a rather small n and k, and Optimal Algorithm 2 is suitable for those data which has a rather small m and k. The upper bound in Optimal Algorithm 1 is quite loose, but in Optimal Algorithm 2, some upper bounds are

[1] CPLEX: IBM ILOG® CPLEX® Optimization Studio, which can solve various programmings.

quite tight. Furthermore, the Optimal Algorithm 2 needs some external supports such as linear integer programming solver, and it's hard to predict the time cost of the solver. Therefore, we prefer Optimal Algorithm 1, and we will use it in numerical experiments. We will analyze them further in Section 4.3.

GDP: Greedy-based Dynamic Programming Algorithm This algorithm is based on the following observation: the passengers in a bus are likely to have close destinations. Therefore, we can divide the coordinates system into several parts, and send several buses to each part to fulfill the orders in it. Using this method, we build a polar coordinate system with the pole of the departure station, and divide the coordinate system into several parts according to the polar angle. Therefore, every bus will take the passengers whose destination stations are in a continuous polar angle range. Then, we can use the dynamic programming for order dispatching. Here are the specific details.

First of all, we transform all the coordinates in the polar coordinate system with the pole of the departure station, and then we sort order by the polar angles of their destinations. Our main idea is to use dynamic programming to dispatch all orders, and initially there is no limit to the total number of buses. Let $g(i)$ denote the maximum profit of satisfying first i orders, and we define $TSP(D)$ as the shortest path length calculated by Algorithm 1 with destinations set D. Then we have

$$g(i) = \min_{i-L<j\leq i}\left(g(j-1) - c_r - c_b \cdot TSP(d_j, d_{j+1}, \cdots, d_i) + \sum_{k=j}^{i} p_k\right)$$

with initial value $g(0) = 0$. After dispatching all orders, we may get a partition with more than m buses, then we calculate each buses profit and choose the largest m buses. Furthermore, we can drop all the buses which have negative profits to maximize our profit. Here is the pseudo-code of the algorithm.

Algorithm 2: GDP Algorithm - a solution to ODRP Problem

```
Input: The orders with all destinations $d_1, \cdots, d_n$, the limitation of number of buses
       $m$, the capacity limitation of bus $L$, the bus fare price of each order $p_i$, the
       fixed cost $c_r$ and the fuel cost per kilometer $c_b$
Output: The maximum profit with at most $m$ buses
1  $g[\cdot] \leftarrow +\infty$;
2  $id[\cdot] \leftarrow -1$;
3  $profit[\cdot,\cdot] \leftarrow 0$;
4  $g[0] \leftarrow 0$;
5  for $i = 1$ to $n$ do
6      $income \leftarrow 0$;
7      for $j = i$ downto $\max(i - L + 1, 1)$ do
8          $income \leftarrow income + p_j$;
9          $profit[j,i] \leftarrow income - c_r - c_b \cdot TSP(\{d_j, d_{j+1}, \cdots, d_i\}, D_0, dist)$;
10         if $g[j-1] + profit[j,i] > g[i]$ then
11             $g[i] \leftarrow g[j-1] + profit[j,i]$;
12             $id[i] \leftarrow j - 1$;
13 $i \leftarrow n$;
14 $bus\_profit \leftarrow []$;
15 while $i \neq 0$ do
16     append($bus\_profit$, $profit[id[num] + 1, i]$);
17     $i \leftarrow id[num]$;
18 if $len(bus\_profit) > m$ then
19     select_m_largest($bus\_profit$, $m$);    // select $m$ largest item in $bus\_profit$
20 drop_all_negative($bus\_profit$);    // drop all buses with negative profits
21 return $sum(bus\_profit)$;
```

Time Complexity: $O(nL \cdot TSP_c(L))$, where $TSP_c(t)$ is the time complexity of combined algorithm for TSP problem of t cities. For large L, we can assume that $TSP_c(L) = O(GLN^2)$, where N is a constant number of populations and G is a constant number of generations. Therefore, we can regard the time complexity of GDP algorithm as $O(nGL^2N^2)$ for big data.

SGDP: Simple Greedy-based Dynamic Programming Algorithm This algorithm is very similar to GDP algorithm, except that we do not use the polar coordinate system to sort the passengers in polar angle and we use the destination number to sort the passenger. The rest of the algorithm is exactly the same as the GDP algorithm. This algorithm usually makes less profit than the GDP algorithm, but it can be used as an assessment baseline in the numerical experiments.

IGDP: Improved Greedy-based Dynamic Programming Algorithm In GDP, we only divide the area by the polar angle, but polar radius matters, too. Consider two sets of orders in the same polar angle range but with the different poly radius, the better solution is to dispatch several buses for each order set. Based on this idea, we can first divide the destinations into two parts: short-distance part and long-distance part, and we use radius of 12 km as a boundary. And then, we apply GDP to each part to maximize the profit. This idea can efficiently reduce the total fuel cost and make more profits.

4.3 Algorithm Analyses

In this section, we will analyze the algorithms introduced above, and try to find out the physical limits of the optimal algorithms. Table 4 shows all the algorithms introduced above and their time complexities.

Algorithm	**Time Complexity**	***(Optional)* Explanations**
Optimal Algorithm 1	$O\left((m+1)^n \cdot k^2 \cdot 2^k\right)$	The upper bound is quite loose
Optimal Algorithm 2	$O\left(2^{mk} \cdot (k^2 \cdot 2^k + LIP(mk))\right)$	-
GDP-series Algorithm (GDP / SGDP / IGDP)	$O(nL \cdot TSP_c(L))$	$O(nGL^2N^2)$ for big data

Table 4. The time complexity of the algorithms

Assume that modern computer can execute 5×10^8 to 10^9 operations per second.

- For Optimal Algorithm 1, since the upper bound k is quite loose, we mainly focus on m and n. When n is small, we can regard k in the time complexity of TSP as n. If n is small, then m can not be very large, therefore, we can regard m as a small constant. Hence, the time complexity mainly depends on n, after some testing we find out that the physical limit of the Optimal Algorithm 1 is approximately $n = 15$.
- For Optimal Algorithm 2, the upper bound k is quite loose, too. When the data is small, we can regard k as n. If n is small, then m cannot be very large, therefore, we can regard m as a small constant. Since m is on the exponent, so it will have a great effect on the time complexity. What's more, the time complexity also depends on the efficiency of solving linear integer programming, which is sometimes unpredictable. Hence, we can calculate that the physical limit of Optimal Algorithm is approximately $n = 10$.
- For GDP, SGDP and IGDP algorithms, the time complexity is $O(nGL^2N^2)$ for big data, where G is set to be 200 and N is set to be 100. Based on our assumption, when $n \leq 5000$, GDP-series algorithm can output the order dispatching plan and the route within one minute, which is quite acceptable.

Therefore, the physical limit of the optimal algorithms is approximately $n = 15$, and for most of the time, GDP-series algorithms can output the result in an acceptable time.

5 Numerical Experiments

In this section, we will test the efficiency and effectiveness of GDP-series algorithm, which is introduced in Section 4, based on the real-world trace data from DIDI GAIA Open Dataset[1]. We will mainly use Python as the programming language in the following experiments. We filter the order first and only consider about the orders during nighttime (23:00 to 6:00 on the next day) according to the application of our platform.

[1] DIDI GAIA Open Dataset: https://outreach.didichuxing.com/research/opendata/en/

5.1 Destination Selection

In this section, we will use the methods in Section 3.2 to select k destinations based on Chengdu data and Haikou data respectively. We will make comparisons between two different destination selection methods. We analyze the hotel data based on the hotel data set of Meituan Hotel Platform[1].

Without considering other factors, it is obvious that if the number of orders for a particular destination exceeds the carrying capacity of a single bus, then we must have a reason to choose that destination, so that when the number of orders in a region exceeds this range. For convenience, we directly set α_o to the maximum possible value of L, which is 50.

We also set the linear coefficients β_o and β_h in the formula of the final weight to 0.4 and 0.6 respectively according to experiences and experiments.

Destination Selection in Chengdu First, if we only set k hot destinations based on the order data, then the selected destinations are shown in Fig. 1.

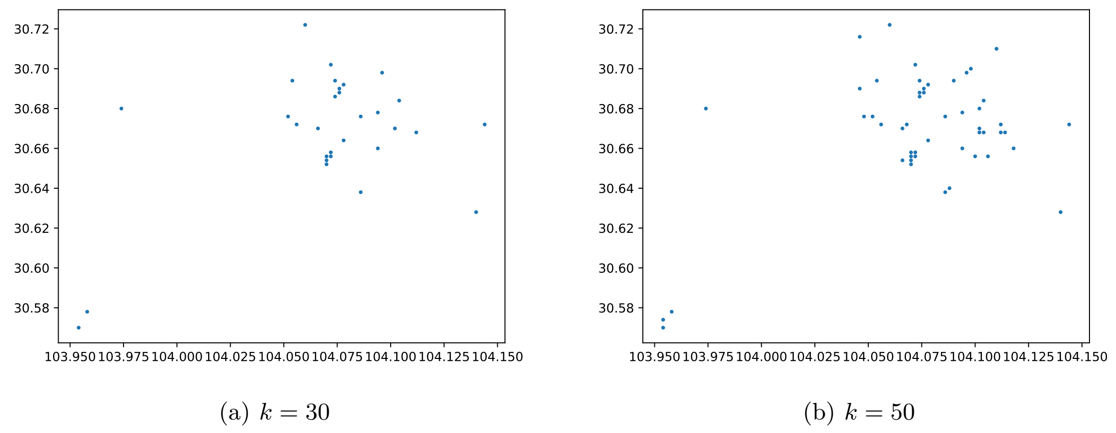


Fig. 1. Destination selection in Chengdu only based on order data

We can find that the destinations selected with this method in Fig. 1 are too dense. It's because some important problems are ignored, which we have explained in Section 3.2. Therefore, we take the hotel data into considerations and propose our destination selection model in Section 3.2. The selected destinations using this method are shown in Fig. 2. Notice here we set W_0 to 0.4.

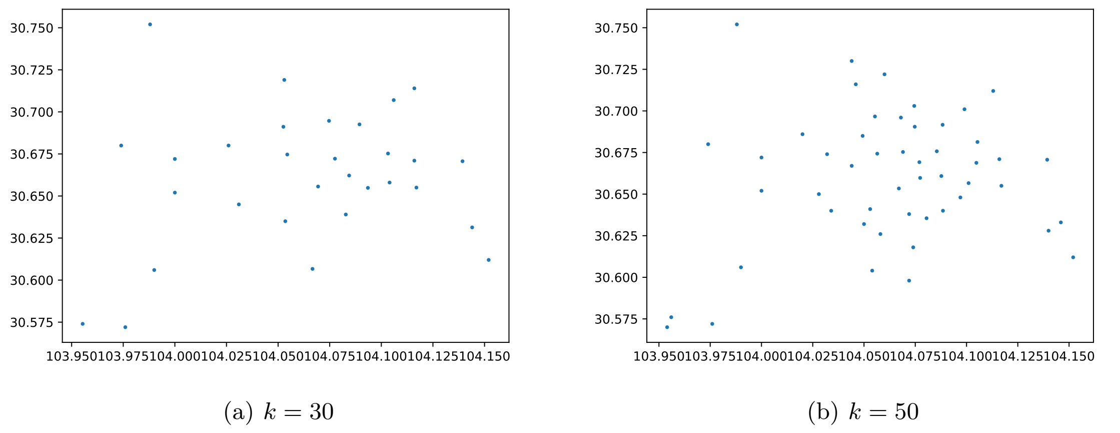
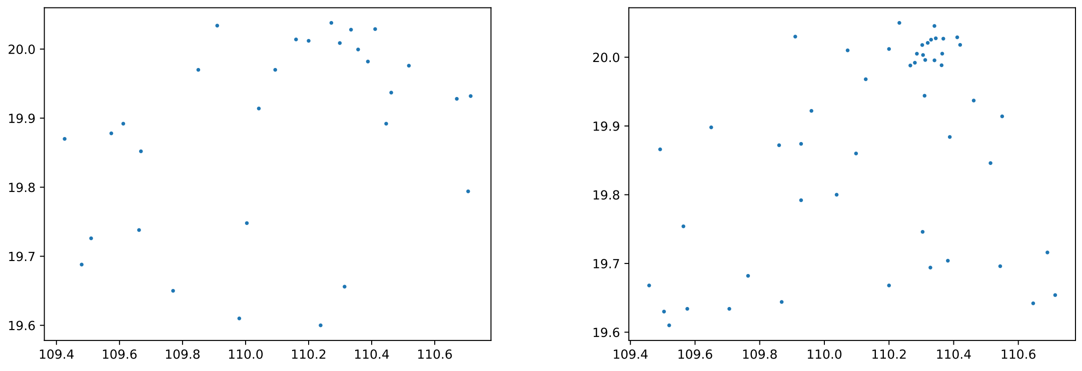


Fig. 2. Destination selection in Chengdu based on both order data and hotel data

The density of destinations in Fig. 2 is acceptable, and our destinations can cover most of the area in Chengdu city. Therefore, our destination selection model is more effective and more rational.

[1] Meituan Hotel Platform: `https://hotel.meituan.com/`

Destination Selection in Haikou Similar with the selection process in Chengdu, we use our destination selection model to select the destination stations, except that we set W_0 to 0.6 here because we have more taxi order data in Haikou. The results of destination selection are shown in Fig. 3.



(a) $k = 30$ (b) $k = 50$

Fig. 3. Destination selection in Haikou based on both order data and hotel data

Although the destinations are still dense in some area, but globally the density is acceptable. Besides, according to the departure selection experiments in Section 5.2, the densest area is selected as our "virtual airport", so the destinations after re-selection will become more acceptable. Therefore, our destination selection model is effective and rational.

5.2 Departure Station Selection

In this section, we analyze the order data and choose the most popular departure location as the virtual airport, and we set the departure location as a metanode with radius $r = 10$ km. The heatmap shown in this section are all generated using the API provided by Baidu Map Open Platform[1].

Departure Station Selection in Chengdu The heatmap of the departures of orders in Chengdu is shown in Fig. 4. We will set the center of the departure station to the area which has the most number of orders departed from. According to the heatmap we can find out that if we set the radius of the departure metanode to be 10 km, then it will cover most of the orders, which is quite rational. Therefore, according to the data, we let the metanode with radius $r = 10$ km center at $(30.64°N, 104.08°E)$.

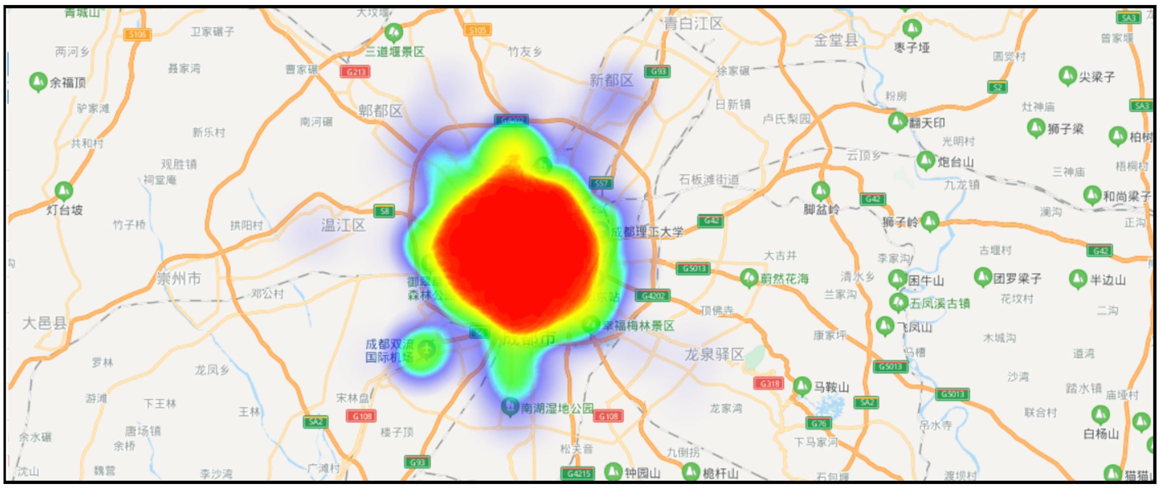


Fig. 4. The heatmap of the departures of orders in Chengdu

[1] Baidu Map Open Platform: `http://lbsyun.baidu.com/`

Departure Station Selection in Haikou The heatmap of the departures of orders in Haikou is shown in Fig. 5. We will set the center of the departure station to the area which has the most number of orders departed from. According to the heatmap we can find out that if we set the radius of the departure metanode to be 10 km, then it will cover most of the orders, which is quite rational. Therefore, according to the data, we let the metanode with radius $r = 10$ km center at $(20.02°N, 110.32°E)$.

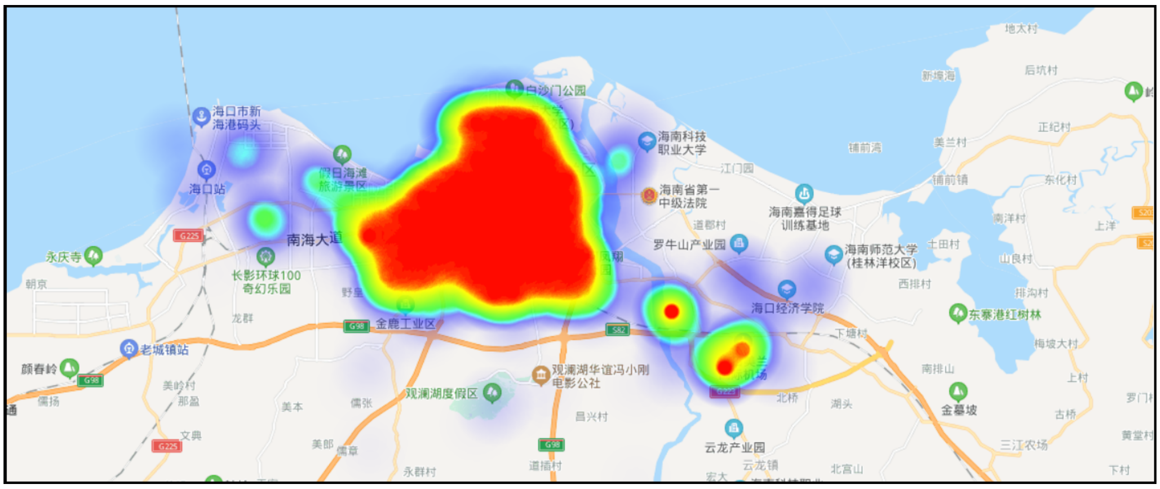


Fig. 5. The heatmap of the departures of orders in Haikou

5.3 Pricing Strategy

In our experiment, we set the parameters of our pricing strategy formulation in Section 3.6 as follows:

1. c_r**:** According to a research [3], the fix cost per bus running whole day is about 1000 yuan. Since every bus runs about 3 to 4 hours on average in our strategy, we set the fixed cost c_r to 150 yuan.
2. c_b**:** According to survey statistics[1], the fuel consumption of buses with 20 to 50 seats per 100 kilometers is about 30 L to 35 L. The average price of 0# diesel is approximately 5.2 yuan[2], so we may set the fuel cost per kilometers c_b to 1.8 yuan/km.
3. p_{pr}**:** According to a research[3], the average profit rate in public transport industry ranges from 9% to 14%, so we can set p_{pr} in our pricing strategy to 10%.
4. c_{mtc}**:** According to a research [3], we set the maintenance cost per bus per hour to 84.14 yuan/h,
5. d_{avg}**:** By observation, the average distance of a bus in each turn is about 25 km. Let's assume that each bus could finish task and return to departure station in 50 minutes, then the d_{avg} can be calculated as Eqn. (7) shown.

$$d_{avg} = \frac{25 \text{ km}}{50 \text{ minutes}} = 30 \text{ km/h} \tag{7}$$

6. h**:** We can assume that the average occupancy rate is 90%.

Then we can calculate p_b and p_c using Eqn. (4) and (5) in Section 3.6 for different L cases:

The value of L	The increasing parameter p_c	The base price p_b
40	0.28	4.58
30	0.38	6.11
20	0.56	9.16

Table 5. p_b and p_c in different L cases

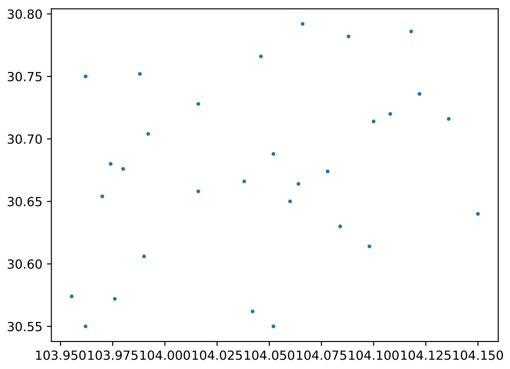
[1] Bus fuel consumption data: refer to https://www.zhihu.com/question/23030854
[2] Fuel price data: refer to http://youjia.chemcp.com/.
[3] Average profit rate: refer to https://wenku.baidu.com/view/139074a8f8c75fbfc77db2d7.html

5.4 Simulation

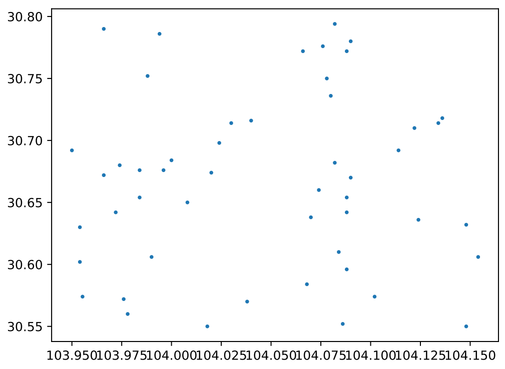
In this section, we use real world data to evaluate the performance our customized bus-booking platform. We conduct experiments on Chengdu data and Haikou data respectively as follows.

We take every half hour as the interval to observe the data comparison, and draw several diagrams to visualize the results. We also use Baidu Map Open Platform to get the real distance between every two stations in road network.

Simulation on Chengdu Data First, we re-select the destinations according to the departure station selected in Section 5.2. The result of re-selection is shown in Fig. 6. We choose $k = 30$ as an example in the following experiments.



(a) $k = 30$



(b) $k = 50$

Fig. 6. Destination re-selection in Chengdu

In the beginning, we assume that there are always enough buses at the airport, that is, m is large enough. After the following analyses, we will discuss the effect of different m. So at first, we set the number of buses m according to buses' capacity L to assure that there are always enough buses.

Now we will take $L = 20$ as an example to compare the profits of the proposed GDP-series algorithms. The profits of GDP, SGDP and IGDP algorithms are shown in Fig. 7.

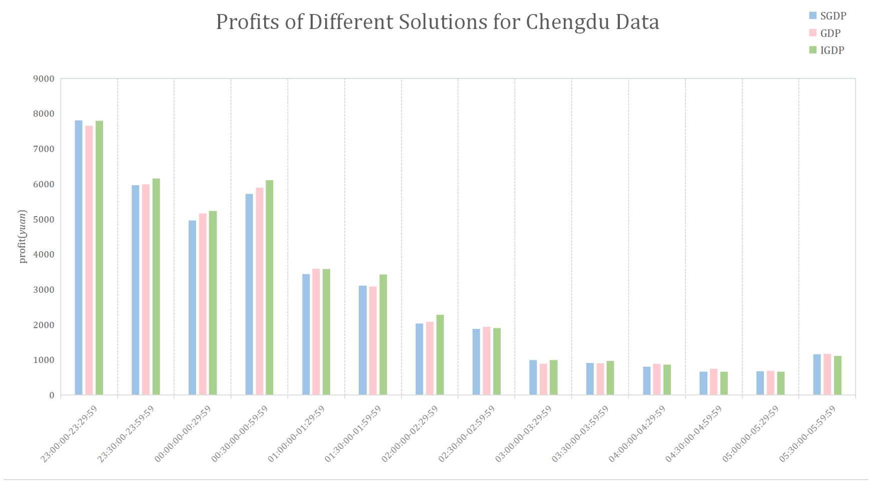


Fig. 7. The profits of GDP, SGDP and IGDP algorithms when $L = 20$

Notice that IGDP almost always reaches the relatively best performance as we expected, especially when there are many orders, so we conduct further researches on it. Now in order to evaluate the algorithm performance on real data changed over time from 23:00 to 6:00 on the next day, we observe the following three metrics: bus number, profit and order number. The visualization of these three metrics is in Fig. 8.

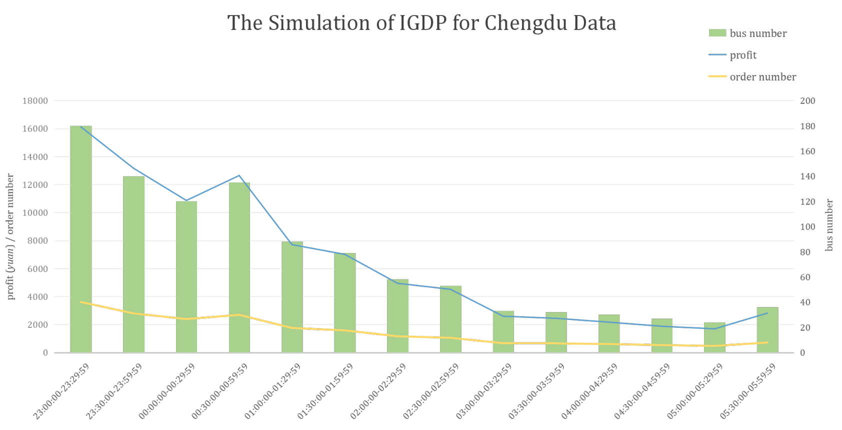


Fig. 8. The simulation of IGDP algorithm for Chengdu Data when $L = 20$

From Fig. 8 we know that bus number, profit and order number all decreased over time, which indicates that these three metrics are in positive correlations to some extent. Before 1:00, there are relatively more orders, while as the night wears on, less and less passengers needs the transportation services. As the sun rises again, at about 5:30 in the morning, both the number of orders and the profit goes up again.

In the real world, the number of passengers who arrive at the airport after midnight is far less than the number of passengers who arrive at the airport during daytime and in early night, therefore, both the number of orders and the profits should have the same trend, and they indeed do. Therefore, the phenomenon observed from Fig. 8 is quite reasonable.

Then, we analyze the influence of bus capacity L on the number of buses and total profit in Fig. 15.

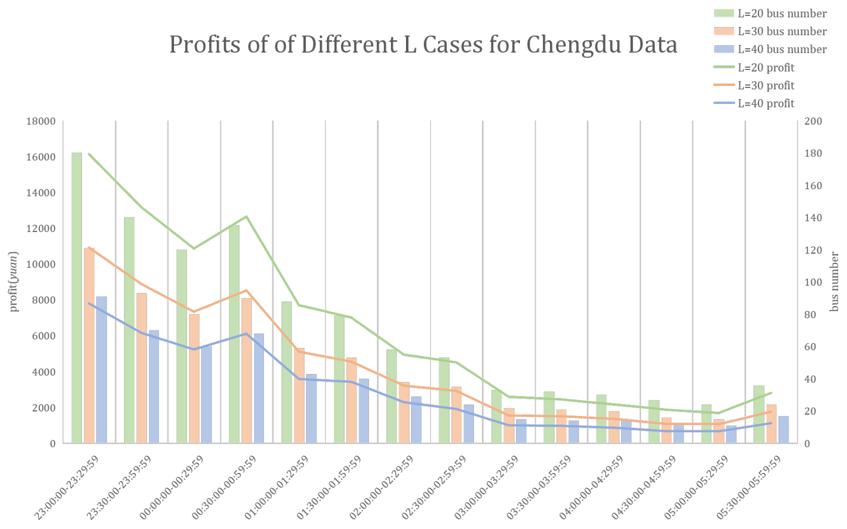


Fig. 9. The influence of L on needed bus number and profit

As we can see, the larger the bus capacity is, the less profit our system could make. On one hand, there are several reasons listed as follows which can make the profit less.

- According to our pricing strategy, as the bus capacity L grows, the parameters p_c and p_b will become more and more small, which means the average price is going down and our income becomes less.
- When L grows larger, the occupancy rate h might decrease instead, and it might drop under our average setting: 90%, which makes our profit less. Meanwhile, we notice that when bus numbers are relatively large, the difference on the number of passengers per bus is not that large, so it makes h smaller and will cause a bad influence on profit when we add the bus capacity.

On the other hand, the larger the bus capacity is, the less bus number is needed. When L is small, there might be a situation that a bus to certain destination could not carry all passengers that want to go there, so we must assign two or more buses to carry them. As the bus capacity is larger, we could solve this problem without adding an extra bus. So from this point of view, the larger the bus capacity is, the more profit we should make.

Therefore, the reasonable explanation is that as the bus capacity becomes larger, both of the reasons have effect on the profit, and the negative effect on the profit outreaches the positive effect on the profit. Hence, in our experiment, our profit becomes less as the bus capacity grows.

At last, we analyze the influence of bus capacity L on the average profit earned in each order, which is shown in Fig. 10.

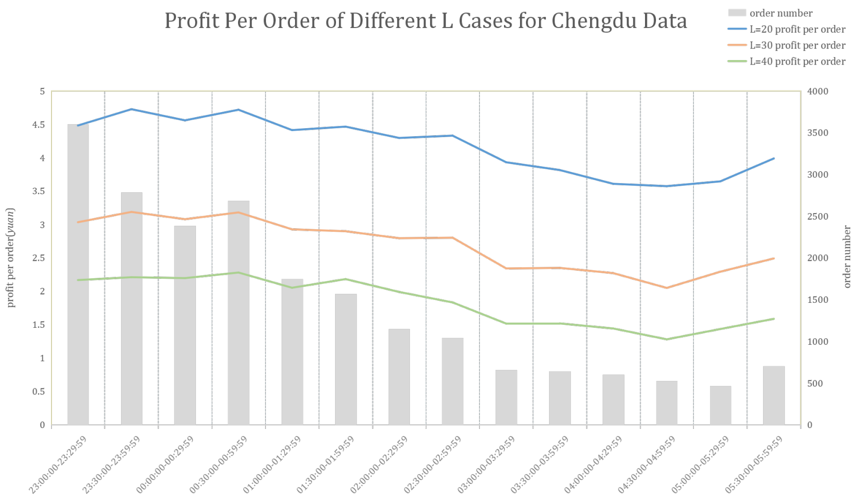


Fig. 10. The influence of L on profit per order

We notice that the profit per order in different L cases fluctuates around the respective horizontal values with the increase and decrease of order number, this is because when the orders become dense, the number of points destined by fewer orders will be reduced, which means the driving time/length of the bus with no load (or less passengers) will decrease and the corresponding low income routes are disappearing. Also, the price of lower bus capacity is relatively higher.

Now, we have completed our analyses when m is large enough, and it's time to see the influence of m on the profit. We assign different values ranging from 0 to 200 to m when $L = 20$ and we use IGDP algorithm to maximie the profit, and the result is shown in Fig. 11.

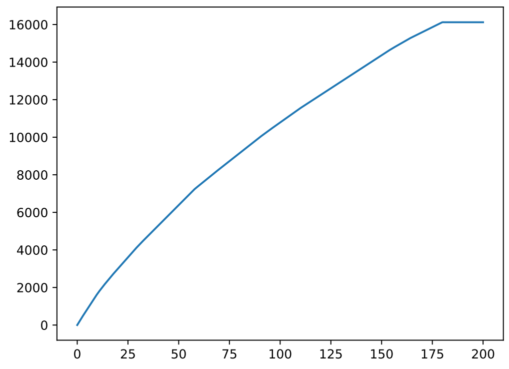
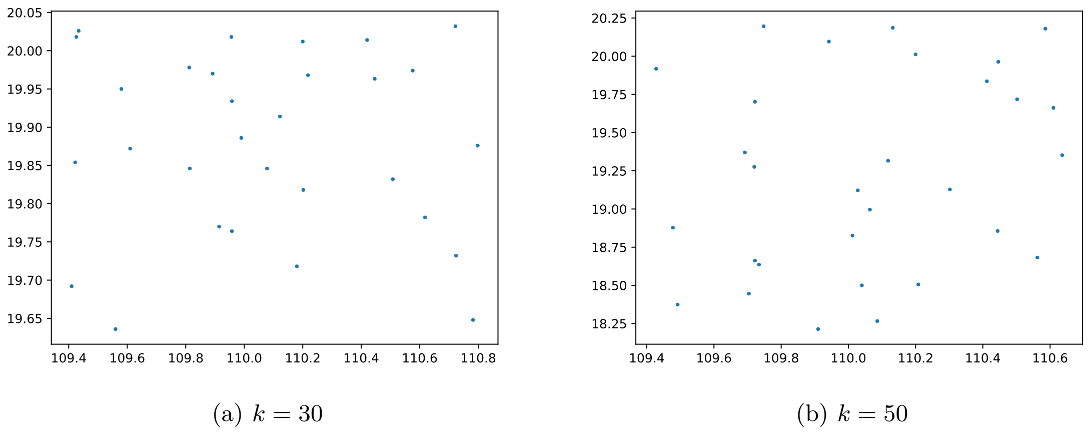


Fig. 11. The influence of different m on the profit when $L = 20$

From Fig. 11, we can find that when m grows larger, the slope of the curve becomes smaller. When m is large enough to reach the maximum profit, the curve becomes as flat as a horizontal line. This is because our algorithm select m buses with largest profit in the end to satisfy the requirement of m. It is greedy-based and may not be optimal, but the result is relatively optimal comparing with other algorithms in experiments. In Section 5.5, we will introduce a strategy to set the reasonable m and L.

Simulation on Haikou Data Similarly, we re-select the destinations according to the departure station selected in Section 5.2. The result of re-selection is shown in Fig. 12. We also choose $k = 30$ as an example in the following experiments.



(a) $k = 30$ (b) $k = 50$

Fig. 12. Destination re-selection in Haikou

Here we also suggest that there are always enough buses. We first compare the performance of GDP-series algorithms by analysing the profits they made. Now we set the bus capacity L to 20 in comparison, and the result is shown in Fig. 13.

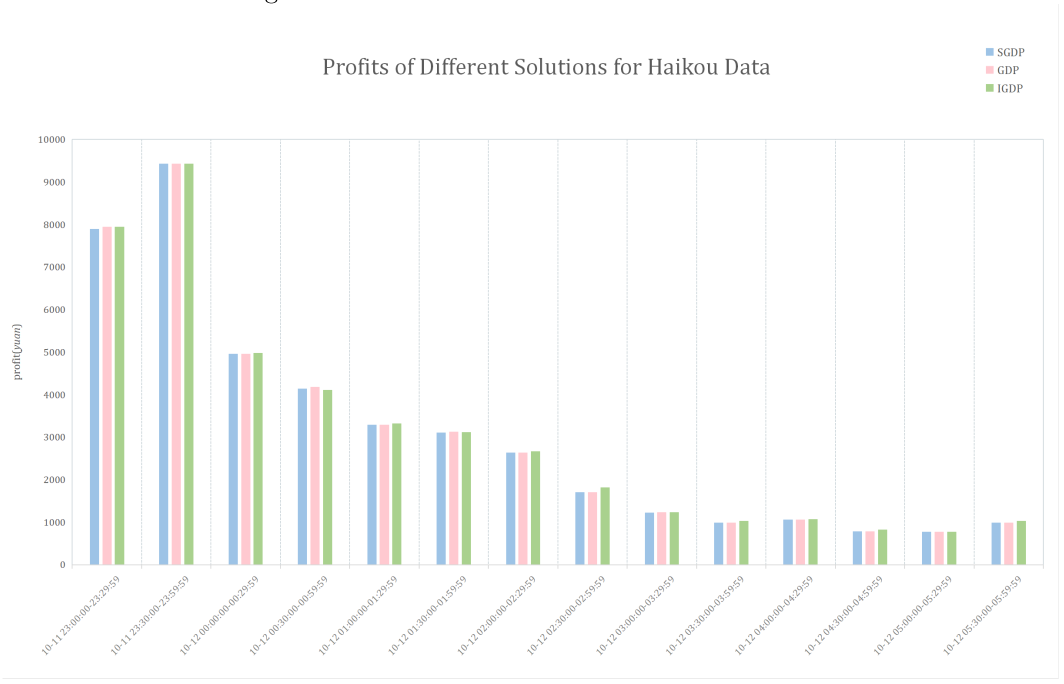


Fig. 13. The profits of GDP, SGDP and IGDP algorithms when $L = 20$

Here we only observe difference from 23:00 to 6:00 in the next day. Since the number of orders in Haikou is relatively less than in Chengdu data, the difference between three algorithms is not that obvious. However, we can still recognize that in most time period, IGDP is the best among them. Also, we calculate the total profit of three algorithms, which is shown in Table 6, and it also indicates that IGDP algorithm is relative optimal among the algorithms. Therefore, we conduct further research on it.

SGDP Algorithm	GDP Algorithm	IGDP Algorithm
$205,158.4108$ yuan	$205,775.9554$ yuan	$206,256.172$ yuan

Table 6. The total profits of GDP, SGDP and IGDP algorithms when $L = 20$

Then, we evaluate the performance of IGDP algorithm changed over time in totally 4 nights when $L = 20$. We also take the three metrics above into consideration, which are bus number, profit and order number. The result is shown in Fig. 14.

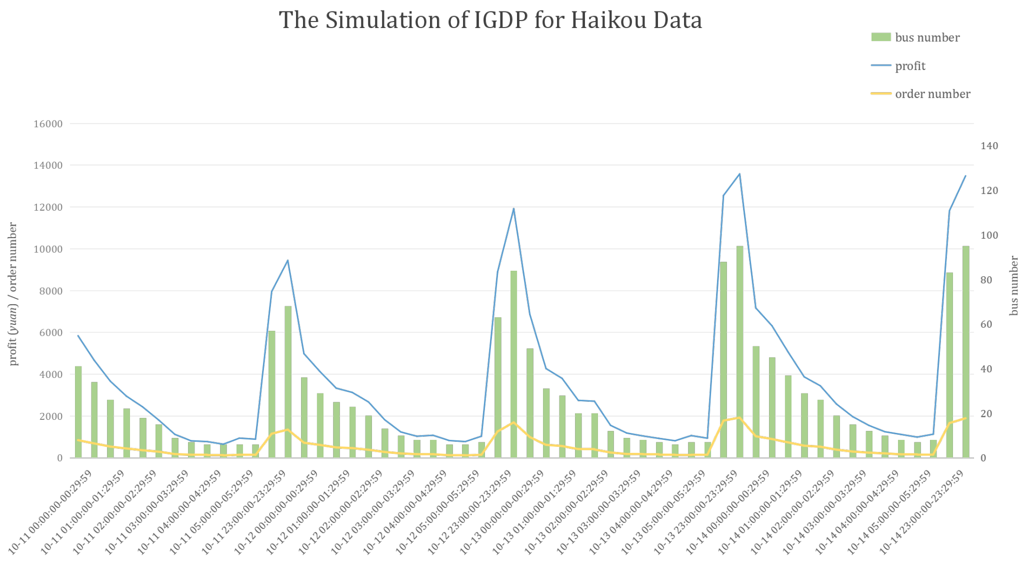


Fig. 14. The simulation of IGDP algorithm for Haikou Data when $L = 20$

From Fig. 14, we can give then same conclusion as the conclusion we derive from Chengdu data that these three metrics are in positive correlations to some extent. Meanwhile, bus number, profit and order number all keep decreasing as the night wears on. What's more, since October 13th in 2017 is Friday, metrics of the last two days are obviously higher than the metrics in other days. Use the same method as the method used in Chengdu data, we know that the phenomenon observed from Fig. 14 is also quite reasonable.

As for the influence of bus capacity on profits and needed bus numbers, we can also use the same method to analyze it, and the result is shown in Fig. 15.

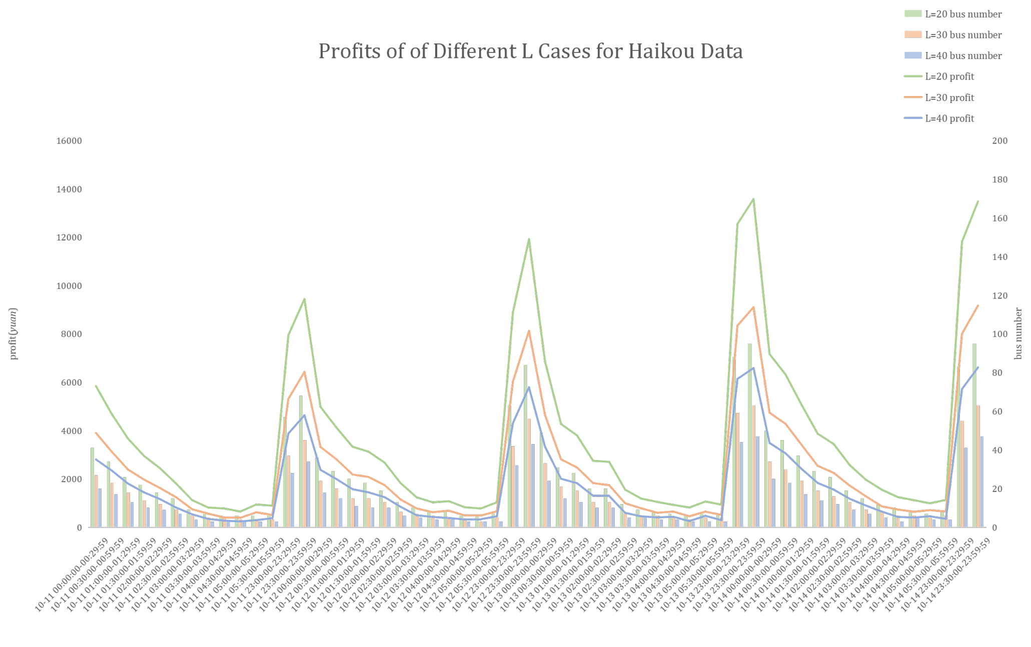


Fig. 15. The influence of L on needed bus number and profit

We notice that the larger the bus capacity L is, the less the bus numbers and profits are. The major reasons are the same as in simulation of Chengdu data. One important thing is that the difference on profits and bus numbers between $L = 20$ and $L = 30$ is larger than that between $L = 30$ and $L = 40$, which indicates that when in $L = 30$ case, the bus occupancy is already not so high.

In the last part, we compare the profit of each order in different bus capacity L cases in Fig. 16.

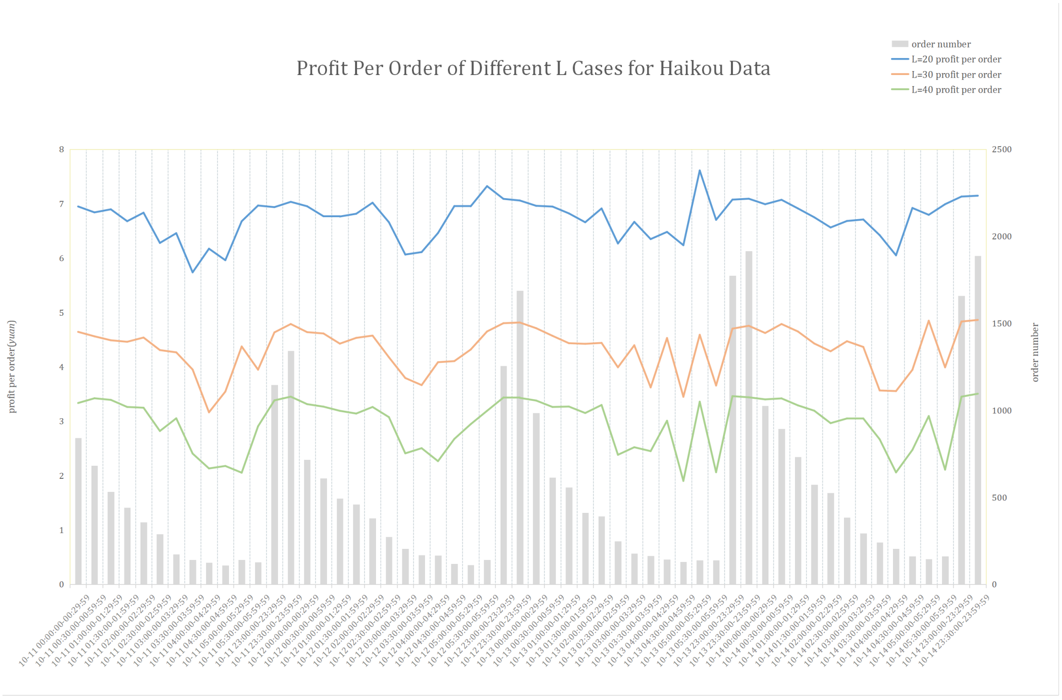


Fig. 16. The influence of different m on the profit when $L = 20$

Just like analyses on Chengdu Data, these three lines fluctuates around their own fixed values, but after being disturbed, the amplitude of up and down fluctuation is greater than Chengdu's at some time. An important factor is that there are so fewer orders sometimes that the distance between each pair of orders has very large damage to profit, which can be verified from the trend of the large order quantity from 23:00 to 2:00 on the next day. Their natural values (also known as balanced values) are determined by pricing strategies in different L cases, which is similar to the analyses of the impact of L on profits above, so we do not repeat the process here, and you can refer to simulation on Chengdu data for details.

As for the influence of m on the profit, it is quite similar with the analyses in Chengdu data, and Chengdu data is more representative, so we do not analyze it again here.

5.5 Strategies about setting m and L

In this section, we mainly introduce the strategies about setting m and L properly.

We learned through the sales department of some distinguished bus companies[1] that the prices of several kind of buses are as follows: the price of 19-seat bus is about $230,000$ yuan, 28-seat bus is about $280,000$ yuan and 40-seat bus is approximately $380,000$ yuan. An important observation is that the price per seat decreases when the seat number increases.

Through the above experimental simulation, we find that when the number of seats per bus decreases, our profit per order and overall profit increase, but it also means the increase in the number of buses that our company needed to buy when our platform was initially founded. From the price of each bus, we can know that this will also cause a substantial increase in start-up capital. Simultaneously, the more buses need more space to park and it will also result in rising management costs. Therefore, if we have enough start-up capital and want to do long-term business, we can directly set $L = 20$ and let m satisfy the maximum number of bus departures in our experiment. However, if the parking space is limited and don't have much money to buy buses, we'd like to just buy more 40-seat buses and let the number of

[1] Some distinguished bus companies: as required, we can not expose the specific company names here.

total buses as much as possible. So basically we should weigh pros and cons in different situations and choose the reasonable m and L.

5.6 Bus price compared with Taxi

In this section, we take Haikou as an example to compare to bus price with taxi.

According to our pricing strategy, the bus price function under different L cases and the taxi fare function we found on the official website of Haikou City Development and Reform Commission[1] are shown in Fig. 17. Obviously, no matter what L is equal to, the fare of the bus is lower than that of a taxi, especially after more than 3 km, the gap between the two fares is gradually getting farther away.

Low price attracts passengers, so they are more likely to choose our service than the taxi services. What's more, according to the simulation in Section 5.4, we can make a relatively high profit under such low prices, which shows the rationality of our model and pricing strategies.

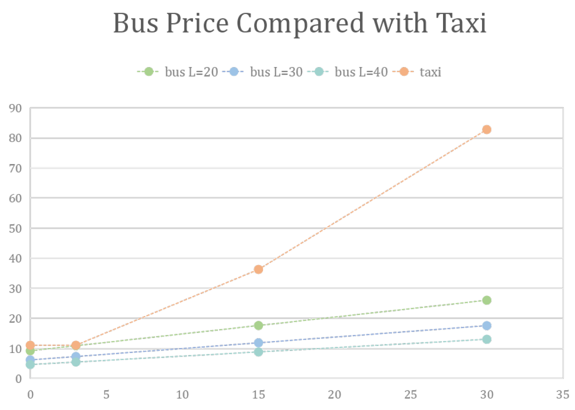


Fig. 17. Bus Price Compared with Taxi

5.7 Route Planning Visualization

In this section, we take Chengdu as an example to visualize the route planning program.

When there are lots of orders, in our design, one bus may only pass few destination stations since there are many passengers heading for the same destination stations. Therefore, the visualization effect is no so good. In order to show our combined TSP algorithm is effective, we assume that there is one bus passing all the destination stations, and we use combined TSP algorithm to perform route planning for it. The planning result is shown in Fig. 18, in which the green point is departure station and the red points are destination stations.

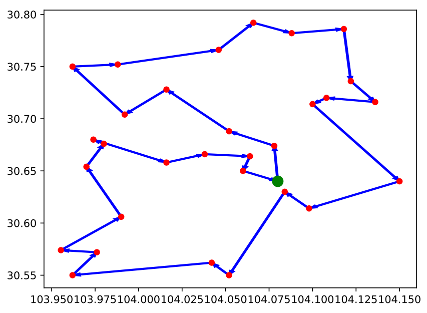


Fig. 18. Route Planning in Chengdu

From Fig. 18 we find that our route is relatively optimal. Therefore, our combined TSP algorithm is effective and rational in route planning.

[1] Taxi fare in Haikou: refer to http://jtj.haikou.gov.cn/ywdt/dtyw/201904/t20190408_1395622.html

6 Conclusion

In this paper, We have formally state ODRP, the order dispatch and route planning problem, to model our bus-booking platform. After introducing our destination and departure station selection model, we have proposed Greedy-based Dynamic Programming (GDP) series algorithms, an effective and rational approach of solving ODRP problems. As for pricing strategy, we make thorough considerations about multiple influence factors, including fuel consumption, maintenance cost and so on. After that, we use real-world data to instantiate our model and evaluate its performance in multiple aspects. From the performance simulations we learn that our algorithms are indeed rational and are able to make relatively maximal profits.

Acknowledgements

– We would like to give our sincere gratitude to Prof. Xiaofeng Gao, our teacher who, with extraordinary patience and consistent encouragement, gave us great help by explaining the problems clearly and providing useful inspirations during classes.
– Many thanks to T.A. Yiming Liu and Shuodian Yu for valuable early suggestions and explanations.
– Sincere thank to Shanghai Jiao Tong University for providing us with a platform where we learn and cooperate with each other.
– Also thanks to Baidu Map Open Platform, which provides us with many useful tools of visualizing and calculating the real distance in the city road network, and Meituan Hotel, which provides us the information of hotels in Chengdu and Haikou.

References

1. Hartigan, J.A., Wong, M.A.: Algorithm as 136: A k-means clustering algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**(1) (1979) 100–108
2. Jing-wei, G., Lan, M., Xiu-cheng, G., Gu, W.: Reasonable price ratio of taxi based on efficiency price [j]. Transport Standardization Z **1** (2009)
3. Deming, Y.: Research on the Analysis and Calculation Method of Urban Public Transport Operation Cost. PhD thesis, Xinan Jiaotong University
4. He, J., Xue-Dong, L.: Improved genetic algorithm: Ga-eo algorithm. Application research of Computers **29** (2012) 3307–3308

Appendix A

Detailed Definitions and Notations

All the definitions and notations are listed in Table 7.

Notations	Definitions
α_o	The upper bound of the number of orders determined by experiences in the experiment.
α_{cd}	non-negative coefficient of dissatisfaction cost caused by ignored order
α_{cl}	non-negative coefficient of passengers' displeasure cost caused by route length
α_p	the threshold of displeasure cost caused by route length
β_o	coefficient of weight W_i^o determined in the experiments
β_h	coefficient of weight W_i^h determined in the experiments
i	the passenger order number
j	the buses order number
b_j	the bus where the i-th person is
c_r	fix cost per bus
c_b	fuel cost per kilometers
$c_d(\cdot)$	the dissatisfaction cost function
$c_l(\cdot)$	the displeasure cost function caused by route length of passenger i
d_0	the departure station
d_i	the destination of passenger i
$dist(d_i, d_j)$	the distance between d_i and d_j
d	the as-the-crow-flies distance used for pricing
d'	the distance suitable for taxi's base price
m	the total number of buses
n	the total number of passengers
n_i	the number of ignored orders
p_i	the bus fare price for passenger i from the departure station to station d_i
p_b	basic price of pricing strategy
p_c	increasing parameter of bus fare price
p_{tb}	taxi's base price
p_{tc}	the increasing parameter of taxi's price
$p_t(\cdot)$	the taxi price function
B_j	the set of orders dispatched to bus b_j
C_r	the total fixed cost of all buses
C_b	the total fuel cost of all buses
D_j	the set of destinations of orders in B_j
$\mathbb{D}$	the set of all destinations
$\mathbb{D}_c$	the candidate destination set
L	the capacity limitation of each bus
P_r	the pricing ratio
h	the average occupancy rate of a bus
C_{run}	the cost of running
C_{mtc}	the cost of maintenance
C_{man}	the cost of management
W	the total income of our strategy
W_0	the lower bound of the final weight
W_i^o	the order weight of block i
W_i^h	the hotel weight of block i
W_i	the final weight of block i
$\pi(D_j)$	a visit order for destination set D_j

Table 7. Definitions and Notations

Appendix B

Code List

All codes are implemented in Python, here is the detailed information. Approximately 50 KB codes are written to fully implement the project in total. Refer to the github repository[1] for our project for details.

All the codes are listed in Table 8.

Code name and directory	**Descriptions**
code/analyze_m.py	To analyze the influence of m on profits.
code/csvreader.py	To parse csv files
code/datahelper.py	To parse json files and write json files
code/departure_selection.py	To select the departure station
code/dest_selection.py	To select and re-select the destination stations using our proposed model
code/get_dist.py	To get the distance between two stations using Baidu Map API
code/heapmap_gen.py	To generate the data for heatmap visualization
code/hotel_address_analyze.py	To get the longitudes and latitudes of hotels based on the addresses using Baidu Map API
code/hotel_scratch.py	To get the hotel data from Meituan Hotel using PhantomJS API
code/kmeans.py	To perform K-Means algorithm
code/odrp_gdp.py	To perform GDP algorithm introduced in Section 4.2
code/odrp_igdp.py	To perform IGDP algorithm introduced in Section 4.2
code/odrp_opt.py	To perform optimal algorithm introduced in Section 4.2
code/odrp_sgdp.py	To perform SGDP algorithm introduced in Section 4.2
code/order_assigner.py	To assign orders according to the given information and store the result in a json file
code/order_batch.py	An batch program used for simulation
code/order_filter.py	To filter the orders according to the given time period
code/reformat_csv.py	To reformat the data in Haikou as csv files
code/simple_dest_selection	To select the destination stations using the simple method (only consider about taxi orders)
code/tsp.py	An TSP solver interface
code/tsp_dp.py	To solve the TSP problem using DP algorithm introduced in 4.1
code/tsp_gene.py	To solve the TSP problem using GA-EO algorithm introduced in 4.1
code/tsp_visualize.py	To visualize the route planning program

Table 8. The code list

[1] The github repository for our project: https://github.com/Galaxies99/CS214-Project

Appendix C

File List

All the related files are listed in Table 9.

File name and directory	**Descriptions**
files/chengdu/departure.csv	The departure data collection in Chengdu
files/chengdu/destination.csv	The final destination selection in Chengdu when $k = 30$ in Section 5.4
files/chengdu/dist.json	The realistic distance between two stations in Chengdu using Baidu Map API
files/chengdu/initial_destination_k30.csv	The destination selection in Chengdu when $k = 30$ in Section 5.1
files/chengdu/initial_destination_k30.csv	The destination selection in Chengdu when $k = 50$ in Section 5.1
files/chengdu/order_filter_chengdu.csv	The order filter result in Chengdu according to departure station and time period
files/haikou/departure.csv	The departure data collection in Haikou
files/haikou/destination.csv	The final destination selection in Haikou when $k = 30$ in Section 5.4
files/haikou/dist.json	The realistic distance between two stations in Haikou using Baidu Map API
files/haikou/initial_destination_k30.csv	The destination selection in Haikou when $k = 30$ in Section 5.1
files/haikou/initial_destination_k30.csv	The destination selection in Haikou when $k = 50$ in Section 5.1
files/haikou/order_filter_haikou.csv	The order filter result in Haikou according to departure station and time period

Table 9. The file list