



Article

Demand Responsive Service-based Optimization on Flexible Routes and Departure Time of Community Shuttles

Jie Xiong ¹, Biao Chen ¹ , Xiangnan Li ², Zhengbing He ^{1,3}  and Yanyan Chen ^{1,*}

¹ Beijing Key Laboratory of Traffic Engineering, Beijing University of Technology, Beijing 100124, China; jxiong@bjut.edu.cn (J.X.); cb1125@foxmail.com (B.C.); he.zb@hotmail.com (Z.H.)

² China Airport Planning & Design Institute Co., Ltd, Beijing 100101, China; lixn@cacc.com.cn

³ Guangdong Provincial Key Laboratory of Intelligent Transportation System, School of Intelligent Systems Engineering, Sun Yat-sen University, Guangzhou 510275, China

* Correspondence: cdyan@bjut.edu.cn

Received: 19 December 2019; Accepted: 23 January 2020; Published: 25 January 2020



Abstract: This paper investigates the optimal routing design problem of a community shuttle system feeding to metro stations based on demand-responsive service. The solution aims to jointly optimize a set of customized routes and the departure time of each route to provide a flexible shuttle service. Considering a set of on-demand trip requests between bus stops and metro stations, a mixed-integer optimization model is formulated to minimize the total system cost, including the operation cost and passenger's in-vehicle cost, subject to the constraints on the route length, time window, detours, and vehicle capacity. To solve the problem, two metaheuristic algorithms, i.e. a tabu search (TS) and a variable neighborhood search (VNS), with different internal operators are specifically designed. A case study based on a realistic network is conducted to test the model and the solution, and comparisons of the performance of different algorithms are investigated.

Keywords: community shuttle; optimal network design; demand-responsive; metro station

1. Introduction

Community shuttle services provide flexible mobility to public transit passengers dispersing into a large community (especially in suburbs) to access nearby metro stations. However, due to the extremely uneven temporal distribution of the trip demands [1–3], shuttle operators have to significantly lower the service frequency or even suspend the service during the nonpeak hours. This situation usually occurs in suburban communities in some populous huge cities, such as Tiantongyuan and Huilongguan Communities in Beijing, China. Due to a large number of commuting trips in the community areas, the bus operator opens some special shuttle lines to serve them between their home locations and the metro stations during peak hours. However, the number of commuting trips declines sharply during non-peak hours and the operator has to downgrade the shuttle service with the consideration of the profits. To provide a high level of service for a large number of trips during peak hours, the bus operators usually focus on developing optimized shuttle systems with fixed routes and reasonable service frequencies [4–6]. While maintaining a stable level of service during nonpeak hours, demand-responsive transport (DRT) is recommended to be introduced into the community shuttle system. Compared with traditional fixed-route transit, DRT is an alternative mode that provides a more flexible service to the public transit passengers whose desired departure time is sparsely distributed over time. Therefore, investigating the demand-responsive community shuttle design problem is a critical task to enhance the service level of community shuttles during nonpeak hours by providing a type of precise service to passengers.

DRT systems are a class of transit services in which a fleet of vehicles dynamically changes routes and schedules in order to accommodate demand within a service area [7]. Organizing the one-off routing of vehicles to satisfy the predetermined trip requests is a basic step that determines the effectiveness of the subsequent scheduling and dynamic vehicle dispatching. This study thus focuses on a static routing design of community shuttles for given on-demand trip requests that include their pickup and delivery points, the number of passengers, and some limitations on the service time. Two classes of network-based combinatorial optimization problems in the literature, i.e., the pickup and delivery problem (PDP) and the dial-a-ride problem (DARP), are closely related to the proposed problem in this paper. For a comprehensive review of the models and algorithms of the two types of problems, interested readers can refer to [8,9]. Agatz et al. in 2012 surveyed the related operations research models for dynamic ride-share systems and pointed out the following broad areas for further research: (1) fast optimization approaches for real-life instance size, (2) incentive schemes to build critical mass, and (3) optimization approaches that allow choice [8]. Ho et al. in 2018 summarized the research on the DARP since 2007 and provided a taxonomy of the problem variants and the algorithms. Some diverse areas of applications for the DARP were also described in the research [9].

In the last few decades, numerous investigations regarding the PDP, which can be treated as a generalization of the vehicle routing problem (VRP), have been performed by many scholars. The PDP is usually concerned with the construction of optimal routes to satisfy transportation requests, each requiring both pickup and delivery under the constraints of vehicle capacity, time window and precedence constraints. The studies on the general PDP can be seen in [10–13]. For example, Ropke and Pisinger in 2006 [10] aimed to construct routes that visit all locations of the requests, such that corresponding pickups and deliveries of the requests are performed sequentially by the same route within a certain time window. Irnich in 2000 [12] introduced a type of multi-depot pickup and a delivery problem where all requests had to be picked up at or delivered to one central location which had the function of a hub or consolidation point.

In recent years, some variants of the PDP have been investigated with the consideration of a variety of particular cases. Naccache et al. in 2018 [14] investigated the multi-pickup and delivery problem with time windows in which a set of vehicles was used to collect and deliver a set of items defined within client requests. Unlike the general PDP, a request in their research was composed of several pickups of different items, followed by a single delivery at the client location. Haddad et al. in 2018 [15] considered the multi-vehicle one-to-one pickup and delivery problem with split loads where the loads can be split and fulfilled by multiple vehicles. The problem was linked with a variety of applications for bulk product transportation, bike-sharing systems, and inventory re-balancing. It is notable that most PDPs and the associated variants are concerned with freight transportation. That means that the metrics measuring the passenger service level do not need to be considered in these problems, whereas they are important factors in our study.

DARP is a variant of PDP in which the loads to be transported represent people. DARPs are always motivated by various real-life applications, such as offering services for elderly or disabled people [16,17], providing dedicated transportation to airports [18] or hospitals [19] and customized bus service design [20]. For example, Detti et al. in 2017 addressed a multi-depot dial-a-ride problem arising from a real-world healthcare application with the consideration of several constraints such as heterogeneous vehicles, vehicle–patient compatibility, quality of service requirements, patients' preferences, tariffs depending on the vehicles' waiting [19]. Tong et al. in 2017 developed a joint optimization model based on a space-time network to jointly optimize passenger-to-vehicle assignment and vehicle routing for customized bus service [20].

The objectives of DARPs usually include passengers' trip costs (e.g. the in-vehicle cost and some penalties incurred by unsatisfactory service) and the service provider's operating costs (e.g. the total vehicle travel distance and the number of required vehicles) (see [17–20]). Some realistic constraints with regard to these two aspects, such as the service time window and maximum route duration, are usually considered in DARPs. In dealing with the time window constraint, most studies treat it as a

hard constraint and determine the vehicle's departure time by calculating the margin time, which is defined as the maximum delay of the vehicle's arrival time. However, we may not obtain a sufficiently satisfactory solution that can be used in practice in such a case since some in-vehicle passengers may have to bear additional waiting time incurred by the waiting of the vehicle if it arrives earlier than the earliest departure time of a certain request.

Although PDPs and DARPs are known as NP-hard (non-deterministic polynomial-time hardness), efforts on exact approaches have been made by many studies [11], [14,15], [20–22]. For example, Ropke and Cordeau in 2009 introduced a branch-and-cut-and-price algorithm in which lower bounds were computed by solving through column generation of the linear programming relaxation of a set partitioning formulation [11]. Tong et al. in 2017 developed a solution algorithm based on the Lagrangian relaxation to decompose the primal problem into a generalized assignment problem and a time-dependent shortest path problem which can be solved by a dynamic programming method [20]. Braekers et al. in 2014 exactly solved the multiplot heterogeneous DARP by adapting the branch-and-cut algorithm for the standard dial-a-ride problem [21]. Although the optimal or approximate optimal solutions can be obtained by using these approaches, in theory, the “curse of dimensionality” always exists due to the difficulty of modeling all the constraints of real-world problems.

To solve large-sized problems, a lot of research attention is devoted to the development of heuristics and metaheuristics. The Tabu search technique has been widely applied to PDPs and DARPs (e.g., [13,19]). It used a tabu list to keep track of recent moves or visited solutions so that they can be forbidden for a number of iterations to avoid cycling. Another type of metaheuristics that was quite effectively used in solving the problems is the class of neighborhood-based search algorithms, mainly including the adaptive large neighborhood search (ALNS, e.g. References [10,15]) and variable neighborhood search (VNS, e.g. References [19,23]). The ALNS proposed in Reference [10] was composed of a number of competing subheuristics that were used with a frequency corresponding to their historic performance and was tested on more than 350 benchmark instances with up to 500 requests. The VNS proposed in Reference [23] used three classes of neighborhoods to generate new solutions. The main idea to make the ALNS and the VNS effective is to develop diversified neighborhood generating methods. The application of the diversified methods performed well in spreading out the directions of the evolution process in the huge solution space so as to avoid plunging into a local optimum.

This paper investigates the optimal routing design problem of a community shuttle system feeding to metro stations based on demand-responsive service. A combinatorial mixed-integer optimization model is formulated to jointly optimize a set of customized routes and the departure time of each route. A variety of realistic constraints, i.e., the time window, maximum detours, vehicle capacity, and route duration, are considered in the model. To solve the problem, two meta-heuristic algorithms, i.e., a tabu search (TS) and a variable neighborhood search (VNS), are designed to guide the solution evolving process. The major contributions of this paper are as follows:

1. An analytical method is embedded in the model to precisely determine the optimal vehicle departure time of each route by treating the related constraints as soft constraints. We believe that this method can provide a competitive solution that is suitable for practical use since it may further minimize the total passengers' in-vehicle time and the route duration at the expense of a certain violation of the time window constraint.
2. Different internal operators, i.e., request insert and sequence reorder, are designed to enable the TS and VNS to be conducted smoothly. Comparisons on the performance of the algorithms with different combinations of internal operators are presented.

The remainder of the paper is organized as follows. The customized routing design problem of demand-responsive community shuttles is described in Section 2, and a mixed-integer optimization model is formulated accordingly. In Section 3, an analytical method is introduced to give a precise vehicle departure time for a generated solution. In Section 4, the complete solution frameworks

by using the TS and the VNS with different internal operators are proposed to solve the problem. In Section 5, a computational experiment based on a realistic network is presented, along with the computational results of different meta-heuristics and different combinations of internal operators, and then comparisons of the performance and CPU time follow. Finally, conclusions are made in Section 6.

2. Model Formulation

2.1. Problem Description

The study area is shown in Figure 1, where there exist a depot and several shuttle stops and metro stations. For a given number of routes to be optimized and a set of predetermined trip demands between the shuttle stops and metro stations, the primary task is to design a set of customized routes to satisfy these trip demands with the consideration of the interests of both passengers and operators. Each numbered trip request as shown in the top-left subfigure of Figure 1 consists of five items, i.e., the starting stop/station, the terminal station/stop, the earliest departure time (the lower limit of the time window), the latest departure time (the upper limit of the time window), and the number of passengers. All the requests are recorded in the matrix Demand, which is shown on the bottom left of Figure 1. The customized routes, which are unique indexed, are presented in the top-right subfigure. They start/terminate at the depot and pass through a series of shuttle stops/metro stations in sequence to perform certain requests. All these routes make up a solution, which is represented as the matrix S on the bottom right of Figure 1. The rows in S from top to bottom represent the sequence of stops of all the routes, the indexes of the requests to be served at the stops, identification of loading or unloading at the stops (1-loading, 2-unloading), and the indexes of the current routes. It is clear to see that for each trip request, the activity of the pickup or the delivery is represented by a unique column of S . Note that the vehicle departure time of the routes is also a set of decision variables of the problem. As we will describe in Section 3, this problem can be solved by an analytical method.

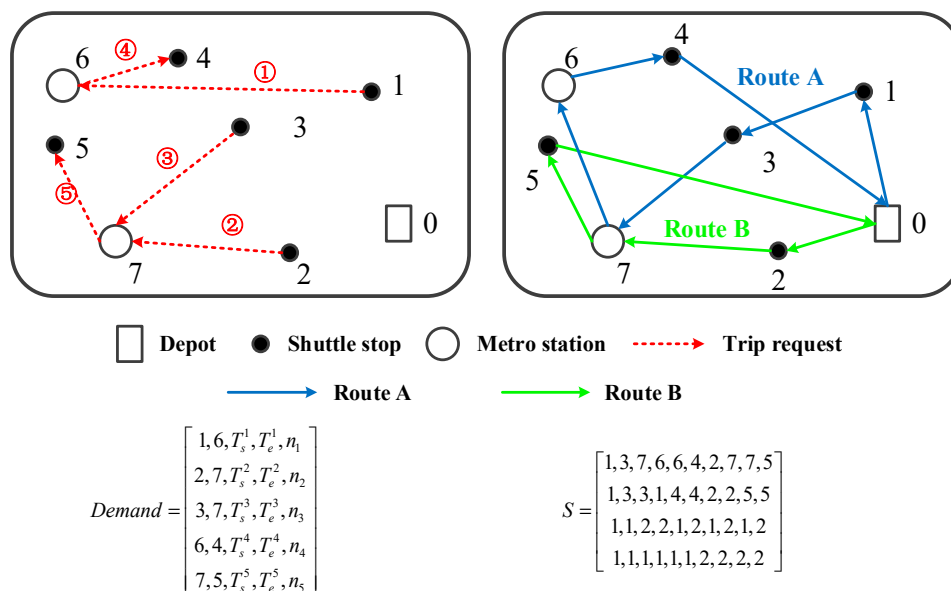


Figure 1. An example of the study area, trip requests and vehicle routes.

Some basic assumptions are made as follows:

1. The vehicle travel time between every two stops/stations and the dwell time at the stops/stations are assumed to be constants.
2. The requests are performed by a set of routes equipped with homogeneous vehicles.
3. The requested time window refers to the time window of the pickup service in this study, and the delivery time window is considered as the maximum allowable in-vehicle time.

2.2. Objective Function

The objective function of this study mainly consists of the passenger in-vehicle cost and the operator cost. It can be written as follows.

$$C_T = C_S(S) + C_I(S, D_0) \quad (1)$$

where C_T is the total cost, which includes both the operator cost and the passengers' in-vehicle cost. C_S is the operator cost, which is related to the set of designed routes (S) and can be further expressed as (2), and C_I is the passenger in-vehicle cost, which is related to both S and the set of vehicle departure time (denoted by D_0) and is further expressed as (3).

$$C_S = \mu_1 \sum_r \sum_i t_{i,i+1}^r \quad (2)$$

$$C_I = \mu_2 \sum_r \sum_k \sum_i (\delta_1(r, k, i) \cdot A_i^r - \delta_2(r, k, i) \cdot B_i^r) \cdot n_k \quad (3)$$

where $t_{i,i+1}^r$ is the vehicle travel time between every two adjacent stops on route r , A_i^r is the arrival time at stop i on route r , B_i^r is the service start time at stop i on route r , n_k is the number of passengers of request k . $\delta_1(r, k, i)$ and $\delta_2(r, k, i)$ are binary variables that equal 1 if stop i on route r is the starting and terminal stop of request k and equal 0 otherwise, μ_1 is the operator cost per unit of distance, and μ_2 is the passenger cost per unit of in-vehicle time.

For $\forall i$ and r , there exist some relationships among the vehicle arrival time (A_i^r), the service start time (B_i^r), and the vehicle departure time (denoted by D_i^r). The vehicle arrival time at a stop can be expressed as the departure time at the last stop plus the travel time between the two stops. The vehicle departure time at a stop can be further expressed as the service start time plus a dwell time at the stop (which is assumed as a constant and denoted by t_s). The service start time values the vehicle arrival time if the vehicle arrives no earlier than the lower limit of the service time window, which is denoted by $T_s^{r,i}(r, i$ -the index indicating the route and the stop), otherwise, it will wait until $T_s^{r,i}$.

2.3. Constraints

The constraints of the problem are formulated as follows.

$$T_s^{r,i} \leq B_i^r \leq T_e^{r,i}, \forall r, 1 \leq i \leq |r| \quad (4)$$

$$Ld_i^r \leq P, \quad \forall r, 1 \leq i |r| \quad (5)$$

$$D_{|r|}^r + t_{|r|,0}^r - D_0^r \leq T_{max} \quad (6)$$

$$\sum_r \sum_k \sum_i \delta_1(r, k, i) = 1, \quad \forall r, k, i \quad (7)$$

$$\sum_r \sum_k \sum_i \delta_2(r, k, i) = 1, \quad \forall r, k, i \quad (8)$$

$$\sum_i \delta_1(r, k, i) + \sum_i \delta_2(r, k, i) = 0 \text{ or } 2, \quad \forall r, k \quad (9)$$

For a pickup service, Constraint (4) is the time window constraint formulated by collecting the submitted information of each request, where $T_s^{r,i}$ and $T_e^{r,i}$ are the lower and upper limits of a service time window, $|r|$ is the number of stops (except the depot) on route r . It is worth noting that for a delivery service, Constraint (4) becomes the maximum in-vehicle delay constraint of each request. To combine the two constraints together, we define Constraint (4) as service time window constraint, where the service includes both the pickup service and the delivery service. For a pickup service,

the time window is given by the request, while for a delivery service, $T_s^{r,i}$ and $T_e^{r,i}$ are automatically generated by adding t_{min}^k and λt_{min}^k based on them where t_{min}^k denotes the least in-vehicle time of request k and λ is a coefficient greater than 1.

Constraint (5) is the vehicle capacity constraint which restricts the loading on any segment (Ld_i^r) not exceed the vehicle capacity (P). Constraint (6) is the route duration constraint, where D_0^r , $D_{|r|}^r$, $t_{|r|,0}^r$ denote the vehicle departure time at the depot, departure time at the last stop and the travel time between the last stop and the depot, respectively. Constraints (7)–(8) ensure that each request is performed by a vehicle once. Constraint (9) restricts that the start and the terminal stop of each request are visited by the same vehicle.

In this study, Constraints (7)–(9) are satisfied by the designed metaheuristics and the internal operators that will be described in Sections 4 and 5. Constraints (4)–(6) are treated as soft constraints and three types of penalties are formulated, accordingly.

$$C_{P1} = \sum_r \sum_k \sum_i \delta_1(r, k, i) n_k g_1(B_i^r) \quad (10)$$

$$C_{P2} = \sum_r \sum_i g_2(Ld_i^r) \cdot d_{i,i+1}^r \quad (11)$$

$$C_{P3} = \sum_r g_3(D_{|r|}^r + t_{|r|,0}^r - D_0^r) \quad (12)$$

where C_{P1} , C_{P2} , C_{P3} are the penalties incurred by the violations of the service time window constraint, the vehicle capacity constraint, and the route duration constraint. $g_1(x)$, $g_2(x)$, $g_3(x)$ are the corresponding penalty functions with respect to the service start time, the segment loading and the route duration. Combined with the optimal departure time determination method that will be described in Section 3, we believe that a more competitive solution can be obtained by the treatment.

By adding the three penalties into the original objective function, the new objective can be written as follows.

$$C_T = C_S(S) + C_I(S, D_0) + C_{P1}(S, D_0) + C_{P2}(S) + C_{P3}(S, D_0) \quad (13)$$

3. Vehicle Departure Time Determination

For a route serving m trip requests, let $I = \{i1, i2, \dots, im\}$ be the set of pickup service nodes and $T_s^I = \{T_s^{i1}, T_s^{i2}, \dots, T_s^{im}\}$ and $T_e^I = \{T_e^{i1}, T_e^{i2}, \dots, T_e^{im}\}$ be the sets of lower limits and upper limits of the corresponding time windows, respectively. For each $T_s^{iu} \in T_s^I$, the speculated vehicle departure time at the depot (denoted by $D_0^{iu,s}$) is a critical departure time that makes the slack time at iu be 0. This means that a synchronization delay at iu caused by the delay at the depot occurs only when the vehicle departs later than $D_0^{iu,s}$. Similarly, for $\forall T_e^{iu} \in T_e^I$, the speculated departure time at the depot is denoted by $D_0^{iu,e}$. $D_0^{iu,e}$ can be treated as another critical departure time that makes the time window penalty at iu be 0, and the penalty is greater than 0 if the vehicle departs later than this time.

Let D_0^{Is} and D_0^{Ie} store the vehicle departure time that is speculated using all the elements in T_s^I and T_e^I , respectively. Two propositions are presented as follows:

Proposition 1. For $\forall iu \in I$, there always exists $D_0^{iu-1,s} \leq D_0^{iu,s} \leq D_0^{iu,e}$.

Proposition 2. For $\forall iu \in I$ and $\forall v > u$, if $D_0 \leq D_0^{iu,s}$, B_{iu} , D_{iu} , A_{iv} , B_{iv} , and D_{iv} do not change with D_0 , otherwise, A_{iu} , B_{iu} , and D_{iu} are positively correlated with D_0 , and there exists $\Delta A_{iu} = \Delta B_{iu} = \Delta D_{iu} = \Delta D_0$.

Before analyzing the impact of D_0 on each cost and penalty, we first divide C_{P1} in (10) into two parts: the time window penalty due to the pickup service delay (denoted by C_{P11}) and the one due to the delivery service delay (denoted by C_{P12}). The division is motivated by the fact that D_0 has different impacts on the two types of penalties. As we analyzed above, when D_0 gradually increases within the

time horizon, C_{p11} first remains stable and then increases with a growing slope since increasingly more values in T_e^l are exceeded by D_0 . In contrast, C_{p12} decreases in this process and finally becomes stable since it is determined by the total passengers' in-vehicle time. In addition, C_I and C_{p3} have a similar trend with C_{p12} as D_0 increases. A quantified analysis is illustrated as follows.

3.1. Impact of D_0 on C_I and C_{p12}

Assume that the request k is performed by route r , where its pickup and delivery nodes are denoted by $ik1$ and $ik2$, respectively. The in-vehicle cost (denoted by $C_I^{k,r}$) and the time window penalty of delivery service (denoted by $C_{p12}^{k,r}$) of the request k are expressed as follows.

$$C_I^{k,r} = \mu_2 n_k \cdot (A_{ik2} - B_{ik1}) \quad (14)$$

$$C_{p12}^{k,r} = n_k \cdot g_1(A_{ik2} - B_{ik1}) \quad (15)$$

Let in be the last pickup node between $ik1$ and $ik2$ if there exists one. Let $D_0^{in,s}$ be the speculated departure time at the depot using the lower limit of the time window of in . From propositions 1–2, $D_0^{in,s}$ is the maximum departure time that prevents A_{ik2} from increasing with increasing D_0^r . Similarly, $D_0^{ik1,s}$, which is the speculated departure time at the depot using the lower limit of the time window of $ik1$, is the maximum departure time that prevents B_{ik1} from increasing with increasing D_0^r . That is, when D_0^r increases within $[D_0^{ik1,s}, D_0^{in,s}]$, B_{ik1} increases synchronously while A_{ik2} remains unchanged. Therefore, $(A_{ik2} - B_{ik1})$ linearly decreases with increasing D_0^r , and we have $\Delta D_0^r = -\Delta(A_{ik2} - B_{ik1})$ from propositions 2. Based on the analysis, the first derivatives of $C_I^{k,r}$ and $C_{p12}^{k,r}$ with respect to D_0^r are obtained by Equations (16) and (17), respectively.

$$\frac{\partial C_I^{k,r}}{\partial D_0^r} = \begin{cases} -\mu_2 n_k, & D_0^r \in [D_0^{ik1,s}, D_0^{in,s}] \\ 0, & \text{otherwise} \end{cases} \quad (16)$$

$$\frac{\partial C_{p12}^{k,r}}{\partial D_0^r} = \begin{cases} -\mu_2 \frac{\partial g_1(A_{ik2} - B_{ik1})}{\partial (A_{ik2} - B_{ik1})}, & D_0^r \in [D_0^{ik1,s}, D_0^{in,s}] \\ 0, & \text{otherwise} \end{cases} \quad (17)$$

The impact of D_0^r on the sum of $C_I^{k,r}$ and $C_{p12}^{k,r}$ of all the requests on route r can be obtained by Equation (18).

$$\frac{\partial (C_I^r + C_{p12}^r)}{\partial D_0^r} = -\sum_k \left(\frac{\partial C_I^{k,r}}{\partial D_0^r} + \frac{\partial C_{p12}^{k,r}}{\partial D_0^r} \right) \quad (18)$$

3.2. Impact of D_0 on C_{p11}

For the request k performed by route r , the time window penalty of pickup service (denoted by $C_{p11}^{k,r}$) is expressed by Equation (19).

$$C_{p11}^{k,r} = n_k \cdot g_1(B_{ik1}) \quad (19)$$

Let $D_0^{ik1,e}$ be the speculated departure time at the depot using the upper limit of the time window of $ik1$. We can conclude that B_{ik1} increases with increasing D_0^r and is always greater than T_e^{ik1} if $D_0^r > D_0^{ik1,e}$. That is, $D_0^{ik1,e}$ is the maximum departure time that prevents $C_{p11}^{k,r}$ from increasing with increasing D_0^r . Combined with proposition 2, it is clear that $\Delta B_{ik1} = \Delta D_0^r$ and $\partial B_{ik1} / \partial D_0^r = 1$. Therefore, the first derivative of $C_{p11}^{k,r}$ with respect to D_0^r is obtained by Equation (20).

$$\frac{\partial C_{p11}^{k,r}}{\partial D_0^r} = \begin{cases} n_k \frac{\partial g_1(B_{ik1})}{\partial B_{ik1}}, & D_0^r \in (D_0^{ik1,e}, +\infty) \\ 0, & \text{otherwise} \end{cases} \quad (20)$$

For all the requests performed by route r , the impact of D_0^r on the sum of $C_{P11}^{k,r}$ can be expressed as Equation (21).

$$\frac{\partial C_{P11}^r}{\partial D_0^r} = \sum_k \frac{\partial C_{P11}^{k,r}}{\partial D_0^r} \quad (21)$$

3.3. Impact of D_0 on C_{p3}

For a route r that performs m requests, let D_{2m}^r be the departure time at the last service node, and let $t_{2m,0}^r$ be the travel time from the last service node to the depot, then, the route duration of r can be written as $(D_{2m}^r + t_{2m,0}^r - D_0^r)$. The route duration penalty of r is expressed as follows.

$$C_{p3}^r = g_3(D_{2m}^r + t_{2m,0}^r - D_0^r) \quad (22)$$

Similar to the impact on $C_{P12}^{k,r}$, the impact on C_{p3}^r is determined by a critical departure time $D_0^{im,s}$, which represents the speculated departure time at the depot using the lower limit of the time window of the last pickup service node on r . Therefore, the first derivative of C_{p3}^r with respect to D_0^r is expressed as follows.

$$\frac{\partial C_{p3}^r}{\partial D_0^r} = \begin{cases} -\frac{\partial g_3(D_{2m}^r + t_{2m,0}^r - D_0^r)}{\partial (D_{2m}^r + t_{2m,0}^r - D_0^r)}, & D_0^r \in [0, D_0^{im,s}] \\ 0, & \text{otherwise} \end{cases} \quad (23)$$

The derivative of the total cost on route r with respect to D_0^r (denoted by $\partial C_T^r / \partial D_0^r$) is determined by accumulating the three derivatives above (Equations (18), (21), (23)). Then, we seek all the extreme points of D_0^r that make $\partial C_T^r / \partial D_0^r = 0$ to formulate the set of candidate departure times. The optimal departure time is then determined by choosing the departure time with the minimum total cost (including penalties).

4. Solution Framework

4.1. Initial Route Generation

As stated above, the constraints of the time window, vehicle capacity, and route duration are modeled as soft constraints. In the solution generation stage, we need only to ensure that the pickup and delivery nodes of each request are sequentially performed once by the same route.

To generate an initial solution, we first sort all the requests in ascending order according to $(T_s^k + T_e^k)/2$ and assign the first R requests to each route directly, where R is the number of routes. Then, we assign the following requests to a certain route in sequence according to a set of probabilities. Assuming that request k' is the current last request on route r , the probability of adding request k ($k > R$) to route r is related to three items: the spatial distance between the delivery stop of k' and the pickup stop of k (denoted by $d_{k,r}^s$), the temporal distance between k and k' (denoted by $d_{k,k'}^t$ and defined by Equation (25)), the current number of nodes on r (denoted by l_r). The probability is obtained by Equation (24).

$$P_{k,r} = \frac{1 / (\tau_1 d_{k,r}^s + \tau_2 |d_{k,r}^t| + \tau_3 l_r)}{\sum_{1 \leq r \leq R} 1 / (\tau_1 d_{k,r}^s + \tau_2 |d_{k,r}^t| + \tau_3 l_r)} \quad (24)$$

$$d_{k,k'}^t = \frac{T_s^k + T_e^k}{2} - \left(\frac{T_s^{k'} + T_e^{k'}}{2} + t_{\min}^{k'} \right) \quad (25)$$

where $P_{k,r}$ is the probability of adding request k to route r , τ_1 , τ_2 , and τ_3 are coefficients.

4.2. Request Insert Operator

The request insert is a basic operator that is repeatedly called by the following sequence reorder operator and meta-heuristics to ensure that the solution is feasible. To insert an unperformed request into a current route, the operator selects two positions on the route to insert the pickup node and service node with the aim of saving the cost increment as much as possible. To this end, two insert operators, which are denoted by *DI1* and *DI2*, are developed. *DI1* determines the positions exactly by enumerating all possible insertion cases, while *DI2* first determines the pickup position by a certain method and then enumerates all possible delivery positions. The steps of the two operators are given in Algorithm 1 and Algorithm 2.

Algorithm 1. Request Insert-1 (*DI1*).

- Step 1. For an unperformed request, the insert positions of the pickup and delivery nodes on route r are denoted by $ip1, ip2$. For $\forall ip1 \in [0, l_r], \forall ip2 \in (ip1, l_r + 1]$, calculate the total cost C_T^r after each attempt of the insert and store them in the set SC_T^r .
- Step 2. Choose the pair of $ip1, ip2$ that corresponds to the minimum value of SC_T^r as the insert positions.
-

Algorithm 2. Request Insert-2 (*DI2*).

- Step 1. Determine $ip1$ (the pickup position of an unperformed request k) by the following sub-steps:
- Step 1-1. For $\forall j \in [1, l_r]$, calculate the spatial distance between node j and the pickup stop of k and the temporal distance between j and k ($d_{k,j}^t$). Note that if j is a pickup node,
- $$d_{k,j}^t = |(T_s^k + T_e^k)/2 - (T_s^j + T_e^j)/2|,$$
- if j is a delivery node, the temporal distance is calculated by (33).
- Step 1-2. Set $j^* \leftarrow$ the j that corresponds to the minimum of the weighted sum values of spatial distance and temporal distance.
- Step 1-3. If $d_{k,j^*}^t \geq 0$, set $ip1 \leftarrow j^*$. Otherwise, set $ip1 \leftarrow j^* - 1$.
- Step 2. For $\forall ip2 \in (ip1, l_r + 1]$, calculate the total cost C_T^r after each attempt of the insert and store them in the set SC_T^r .
- Step 3. Choose $ip2$ that corresponds to the minimum value of SC_T^r as the delivery node insert position.
-

From Algorithms 1 and 2, it is clear that *DI1* obtains the best pair of insert positions exactly by $(l_r + 1)(l_r + 2)/2$ times of attempts. By using *DI2*, this number can be reduced to $(l_r + 2)/2$, however, the obtained pair of positions may not as good as the pair obtained by *DI1*.

4.3. Sequence Recorder Operator

When all the requests are assigned to a certain route, the sequence reorder operator is applied to optimize the performance sequence on each route. Two sequence reorder operators, which are denoted by *L1* and *L2*, are developed here. The steps are given in Algorithms 3 and 4.

Algorithm 3. Sequence Reorder-1 (*L1*).

- Step 1. Input route r . Let D^r be the set of requests performed by route r and sort these requests ascendingly by the earliest departure time.
- Step 2. For a request $k \in D^r$, remove it from r and re-insert it by using *DI1* or *DI2*. This process continues until all the requests in D^r have been chosen to do the operation in sequence.
-

Algorithm 4. Sequence Reorder-2 (L2).

-
- Step 1. Input D' . Initialize r as an empty set.
 Step 2. Insert each request in D' to r by using DI1 or DI2 in sequence.
-

Comparing Algorithm 3 and Algorithm 4, the input of L1 includes both the route (i.e., the current performance sequence of requested service) and the requests, while L2 needs to input only the set of requests. Assuming the number of requests in D' to be $l_r/2$, it is clear that the request insert operator is performed $l_r/2$ times by both L1 and L2. However, the detailed runs of L1 and L2 are still different since the lengths of routes to be inserted are different each time in L1 and L2. Table 1 gives the runs of calculations by different combinations of DI1, DI2, L1, and L2.

Table 1. Runs of calculations by different combinations of DI1, DI2, L1, and L2.

	DI1	DI2
L1	$l_r^2(l_r-1)/4$	$l_r^2/4$
L2	$\sum_{p=1}^{l_r} p(p-1)/2$	$l_r(l_r+2)/8$

4.4. Tabu Search

The tabu search (TS) algorithm starts from the initial solution and generates a new nontabu solution in each iteration. The new solution is generated based on the current one by a developed neighborhood search mechanism. To avoid searching cyclically, a tabu list is established to record some attributes of recently visited solutions. A newly generated solution that possesses the attributes is declared forbidden. In this study, we use the tabu scheme to prevent a request to be reinserted into the route that it was just removed from recently. After a number of iterations, the forbiddance will be released to maintain the stability of the size of the neighborhood solution space.

The neighborhood solution is obtained by transferring a certain request from its original route to another route obeying the tabu list. The request to be transferred each time is selected by a probability, which is expressed as follows.

$$ps_k = \frac{C_T - C_T^{k-}/n_k}{\sum (C_T - C_T^{k-}/n_k)} \quad (26)$$

where C_{Tk} is the total cost after removing request k , n_k is the number of passengers for request k .

Similarly, the route that the request is transferred to is also selected according to a certain probability, which is written as Equation (27). It shows that the probability is determined only by the current route length.

$$p_r = \frac{1/l_r}{\sum 1/l_r} \quad (27)$$

Let *Demand* be the set of requests, S_0 , C_{T0} , and D_{00} are the initial solution, the corresponding total cost and set of departure times, respectively. S^* , C_T^* and D_0^* are the obtained best solution, the minimum total cost and the corresponding set of departure time. θ is the number of iterations that a forbiddance is released. *Tabu* is the tabu list. P_Tabu is the matrix that records the number of iterations each forbiddance lasts. *MAXGEN* is the maximum number of iterations, and π is the number of iterations between two adjacent performances of sequence reordering. The detailed steps of the TS are described in Algorithm 5.

Algorithm 5. Tabu Search (TS).

- Step 1. Input $Demand$, S_0 , C_{T0} , D_{00} , θ , $MAXGEN$. Initialize $S^* \leftarrow S_0$, $D_0^* \leftarrow D_{00}$, $C_T^* \leftarrow C_{T0}$, $gen \leftarrow 1$. For each pair of request k and route r , if k is performed by r in S_0 , initialize $Tabu(k, r) \leftarrow 0$ to represent that k cannot be reinserted into r in the following θ iterations. Otherwise, $Tabu(k, r) \leftarrow 1$. Initialize each element in P_Tabu as zero.
- Step 2. Update $Tabu$ according to P_Tabu : for each pair of request k and route r , if $P_Tabu(i, r) = gen$, update $Tabu(k, r) \leftarrow 1$.
- Step 3. Obtain a neighborhood solution S_1 by the following sub-steps.
- Step 3-1. For all the requests in $Demand$, calculate the probabilities to be selected by (34). Then select a request, which is denoted by k' , according to the set of probabilities.
- Step 3-2. For all the routes, calculate the probabilities to be selected from (35). Then select a route, which is denoted by r' , according to the set of probabilities.
- Step 3-3. Remove from its original route k' and reinsert it to r' by using $DI1$ or $DI2$ to obtain S_1 .
- Step 3-4. Calculate the departure time of each route and the total cost according to S_1 , which are denoted by D_{01} and C_{T1} , respectively.
- Step 4. Update $Tabu(k', r') \leftarrow 0$, $P_Tabu(k', r') \leftarrow gen + \theta$.
- Step 5. If gen is an integer multiple of π , $L1$ or $L2$ is employed to update S_1 by reordering the service sequence of each route, then D_{01} and C_{T1} are updated accordingly, go to Step 6. Otherwise, go to Step 6 directly.
- Step 6. If $C_{T1} < C_T^*$, do the following updates: $S^* \leftarrow S_1$, $D_0^* \leftarrow D_{01}$, $C_T^* \leftarrow C_{T1}$, $S_0 \leftarrow S_1$, $gen \leftarrow gen + 1$. Otherwise, only update $S_0 \leftarrow S_1$, $gen \leftarrow gen + 1$.
- Step 7. If $gen > MAXGEN$, the algorithm ends, $L1$ or $L2$ is employed to update S^* , and D_0^* and C_T^* are updated accordingly, output S^* , D_0^* and C_T^* . Otherwise, return to Step 2.

4.5. Variable Neighborhood Search

A variable neighborhood search algorithm that includes two types of neighborhood search methods is developed to diversify the neighborhood solution space. In the first method, two routes are selected to perform the cross-exchange/transfer of certain requests. An example of the method is presented in Figure 2, in which each cycle represents a request. The number of requests selected on each route to perform the cross-exchange/transfer is denoted by cn_1 .

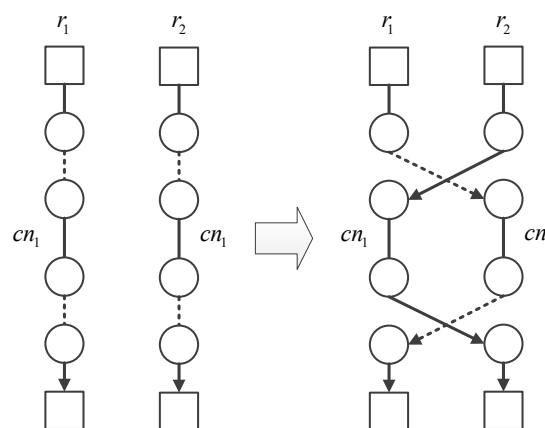


Figure 2. A cross-exchange/transfer of requests between two routes.

It is worth noting that to make the operation be carried out smoothly, at least one of the two routes needs to perform more than cn_1 requests. If the number of requests on both of the routes exceeds cn_1 and the difference between the two individual counts is no more than a preset threshold denoted by Δl , a cross-exchange of cn_1 requests is performed between the two routes. Otherwise, we select only cn_1 requests from the route with more requests and transfer them to the other one. The requests are still

selected according to the set of probabilities defined by (26). Then, $DI1$ or $DI2$ are performed to shift the requests to the other route.

The second neighborhood search is proposed to make a cyclic transfer of requests among multiple routes, which is also proposed in Reference [24] to solve the vehicle routing problem. An example of the method is presented in Figure 3, in which cm_2 and cn_2 respectively denote the number of routes and requests to participate in the transfer. Before the request transfer process, we first sort the routes in descending order according to the route length (i.e., the number of requests). To make the method be conducted smoothly, the number of requests on route r_1 should exceed cn_2 . Moreover, if the difference between the numbers of requests performed by the first and the last routes exceeds the threshold Δl , the process terminates by completing the requested transfer from the second to last route to the last one. Otherwise, it performs one additional transfer, i.e., the transfer from the last route to the first one. Compared with the request transfer scheme presented in Figure 2, this transfer method shows the ability to provide larger neighborhood solutions due to the participation of more than two routes.

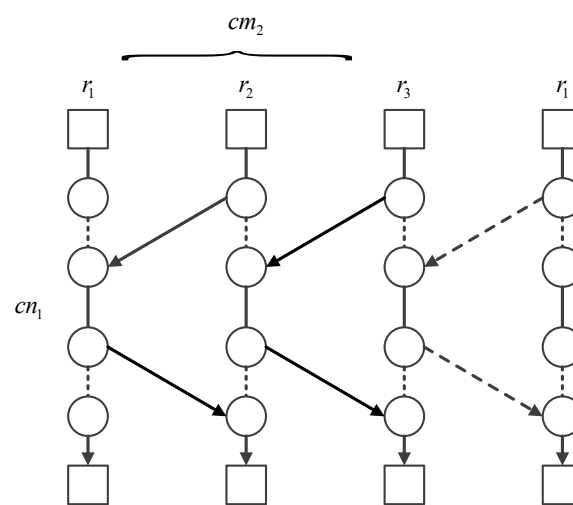


Figure 3. A cyclic transfer of requests among multiple routes.

To further diversity the neighborhood solution space, we replace the original parameters cn_1 , cm_2 , and cn_2 with three new parameters: cn_{1max} , cm_{2max} , and cn_{2max} , which denote the preset maximum values of the original parameters. In each performance of the neighborhood search methods, cn_1 , cm_2 , and cn_2 are randomly generated according to cn_{1max} , cm_{2max} , and cn_{2max} . In this study, we attempt to set cn_{1max} from 1 to 3, set cm_{2max} from 3 to 4, and set cn_{2max} from 1 to 3. Therefore, at most 9 attempts of neighborhood search are performed in each iteration. The parameters are sequentially listed as follows: $cn_{1max} \leftarrow 1 \Leftrightarrow cn_{1max} \leftarrow 2 \Leftrightarrow cn_{1max} \leftarrow 3 \Leftrightarrow cm_{2max} \leftarrow 3, cn_{2max} \leftarrow 1 \Leftrightarrow cm_{2max} \leftarrow 3, cn_{2max} \leftarrow 2 \Leftrightarrow cm_{2max} \leftarrow 3, cn_{2max} \leftarrow 3 \Leftrightarrow cm_{2max} \leftarrow 4, cn_{2max} \leftarrow 1 \Leftrightarrow cm_{2max} \leftarrow 4, cn_{2max} \leftarrow 2 \Leftrightarrow cm_{2max} \leftarrow 4, cn_{2max} \leftarrow 3$. The former 3 attempts to use the cross-exchange/transfer method, while the latter 6 attempts are based on the cyclic-transfer method. If the solution obtained by an attempt to search is better than the current one, the process breaks immediately. Otherwise, we choose the best solution from the 9 obtained solutions and check whether it replaces the current solution by the acceptance criterion of the simulated annealing. The detailed steps are given in Algorithm 6. For the meanings of some repeated notations, one can refer to Algorithm 5, we do not repeat them here.

Algorithm 6. Variable Neighborhood Search (VNS).

-
- Step 1. Input *Demand*, S_0 , C_{T0} , D_{00} , Δl , π , as well as the initial temperature (denoted by T_0), the ending temperature (denoted by T_{end}), the cooling rate (denoted by q). Let NS be the set of solutions obtained in the neighborhood search process in each iteration, N_m record the times of neighborhood search performance in each iteration. Initialize $S^* \leftarrow S_0$, $D_0^* \leftarrow D_{00}$, $C_T^* \leftarrow C_{T0}$, $T \leftarrow T_0$, $NS \leftarrow \emptyset$, $N_m \leftarrow 0$, $gen \leftarrow 1$.
- Step 2. For cn_{1max} valuing from 1 to 3, do the following sub-steps to perform the cross-exchange/transfer neighborhood search.
- Step 2-1. Randomly choose two routes r_1, r_2 and generate cn_1 . If the numbers of requests on both of the routes do not exceed cn_1 , repeat the choosing process until at least one route satisfies the length condition.
- Step 2-2. Perform the cross-exchange/transfer of some certain requests to obtain a new solution denoted by S_1 . These requests are selected according to a set of probabilities defined by (34) and are inserted into the other route by using *DI1* or *DI2*.
- Step 2-3. Calculate D_{01} and C_{T1} accordingly. If $C_{T1} < C_{T0}$, break and go to Step 3. Otherwise, update $NS \leftarrow NS + S_1$, $N_m \leftarrow N_m + 1$, go back to Step 2.
- Step 3. If $N_m < 3$, which implies that S_1 is a better solution than S_0 , if gen is an integer multiple of π , *L1* or *L2* is employed to update S_1 by reordering the service sequence of each route, and D_{01} and C_{T1} are updated accordingly, then go to Steps 4–5. Otherwise, go to Steps 4–5 directly. If $N_m = 3$, go to Steps 6–7.
- Step 4. If $C_{T1} < C_T^*$, update: $S^* \leftarrow S_1$, $D_0^* \leftarrow D_{01}$, $C_T^* \leftarrow C_{T1}$, $S_0 \leftarrow S_1$, $D_{00} \leftarrow D_{01}$, $C_{T0} \leftarrow C_{T1}$. Otherwise, update $S_0 \leftarrow S_1$, $D_{00} \leftarrow D_{01}$, $C_{T0} \leftarrow C_{T1}$.
- Step 5. If $T < T_{end}$, the algorithm ends, *L1* or *L2* is employed to update S^* , D_0^* , C_T^* , and output them. Otherwise, update $T \leftarrow T \cdot q$, $NS \leftarrow \emptyset$, $N_m \leftarrow 0$, $gen \leftarrow gen + 1$.
- Step 6. For cn_{2max} valuing from 1 to 3, do the following sub-steps to perform the cyclic transfer neighborhood search.
- Step 6-1. Randomly generate cm_2, cn_2 , and choose cm_2 routes. Let $\{r_1, \dots, r_{cm_2}\}$ be the set of chosen routes sorting in descending order according to route length. If the requests on r_1 is less than cn_2 , re-formulate $\{r_1, \dots, r_{cm_2}\}$ until r_1 satisfies the length condition.
- Step 6-2. Perform the cyclic transfer of some certain requests to obtain a new solution denoted by S_1 . Calculate the corresponding D_{01} and C_{T1} .
- Step 6-3. If $D_{T1} < D_{T0}$, break and go to Step 7. Otherwise, update $NS \leftarrow NS + S_1$, $N_m \leftarrow N_m + 1$, go back to Step 6.
- Step 7. If $N_m < 9$, which implies that S_1 is a better solution than S_0 , then, if gen is an integer multiple of π , *L1* or *L2* is employed to update S_1 , and D_{01} and C_{T1} are updated accordingly, then go to Steps 4–5. Otherwise, go to Steps 4–5 directly. If $N_m = 9$, go to Steps 8–9.
- Step 8. Select the best solution from NS , which is denoted by S'_1 , perform *L1* or *L2* to update S'_1 and the corresponding D'_{01} , C'_{T1} .
- Step 9. (i) If $C'_{T1} < C_T^*$, update: $S^* \leftarrow S_1$, $D_0^* \leftarrow D_{01}$, $C_T^* \leftarrow C_{T1}$, $S_0 \leftarrow S_1$, $D_{00} \leftarrow D_{01}$, $C_{T0} \leftarrow C_{T1}$.
(ii) If $C_T^* \leq C'_{T1} < C_{T1}$, update: $S_0 \leftarrow S_1$, $D_{00} \leftarrow D_{01}$, $C_{T0} \leftarrow C_{T1}$.
(iii) If $C'_{T1} \geq C_{T1}$, check whether there exists $\exp((C_{T1} - C'_{T1})/T) > p_{rand}$ (p_{rand} is a random number between 0 and 1). If so, update $S_0 \leftarrow S_1$, $D_{00} \leftarrow D_{01}$, $C_{T0} \leftarrow C_{T1}$.
- Return to Step 5.
-

5. Case Study

The network of the study area is the Huilongguan Community, which is large in size (about 4.8 km \times 2.2 km) and located in Beijing, China. As shown in Figure 4, there are 52 bus stops (Nodes 1–52), 6 metro stations (Nodes 53–58) and a depot (Node 0) in the area. By connecting the pairs of the nodes that no other stops exist on the shortest path between them, we formulate the corresponding topology network which is presented in Figure 5.



Figure 4. The network, bus stops and metro stations of Huilongguan Community.

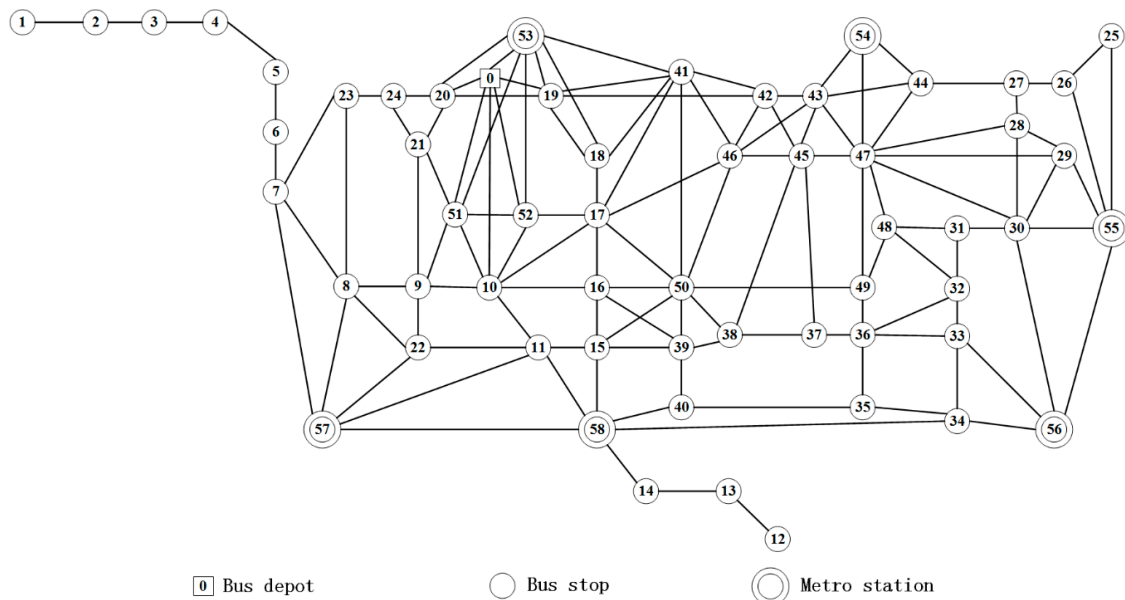


Figure 5. The topological structure of the network of Huilongguan Community.

To simulate the process of generating a trip request, we first randomly choose two nodes from Nodes 1–52 and 53–58 as the origin and the destination since each trip request in the study is defined between a bus stop and a metro station. To generate the reversed trip requests which start from a metro station and terminate at a bus stop, the origin and the destination are just needed to be exchanged according to a certain probability, which can be understood as the ratio of the bidirectional trip requests. Here we set the probability as 0.7. That is if the randomly generated number between 0 and 1 after generating a trip request is greater than 0.7, we exchange the origin and the destination. To randomly generate the time window and the number of passengers, some parameters are set as follows: the time horizon $LN = 300$ min, the range of the pickup time window obeys the normal distribution $N(10, 2^2)$, and the maximum number of passengers per request is set as $D_{Nmax} = 5$. We repeat the generation process to randomly generate 100 requests and formulate the matrix *Demand* by sorting them in ascending order according to $(T_s^i + T_e^i)/2$.

The other parameters are set as follows: $R = 5$, $t_s = 0.5$ min, $\mu_1 = 9$ RMB/min, $\mu_2 = 1$ RMB/min, $\lambda = 1.5$, $P = 11$, and $T_{max} = 180$ min. The penalty functions $g_1(x)$, $g_2(x)$, and $g_3(x)$ are defined by Equations (28)–(30).

$$g_1(x) = \begin{cases} 3.8 \cdot (x - T_e^{r,i}) & x > T_e^{r,i} \\ 0 & \text{otherwise} \end{cases} \quad (28)$$

$$g_2(x) = \begin{cases} 11 \cdot (x - P) & x > P \\ 0 & \text{otherwise} \end{cases} \quad (29)$$

$$g_3(x) = \begin{cases} 58 \cdot (x - T_{max}) & x > T_{max} \\ 0 & \text{otherwise} \end{cases} \quad (30)$$

In the TS algorithm, $\theta = 30$, $MAXGEN = 300$, and $\pi = 10$. In the VNS algorithm, $T_0 = 3000$, $T_{end} = 0.001$, $q = 0.96$, $\Delta l = 22$, and $\pi = 10$. The results presented below are obtained by using MATLAB 2015a on a personal computer with Intel Core i5-3230 CPU @2.6GHz.

Based on the preset parameters above, 4 sets of experiments (represented by DI1+L1, DI1+L2, DI2+L1, and DI2+L2) with different combinations of internal operators are designed to test the TS performance of each case. The convergence processes of the 4 cases are presented in Figure 6. Some related results are given in Table 2.

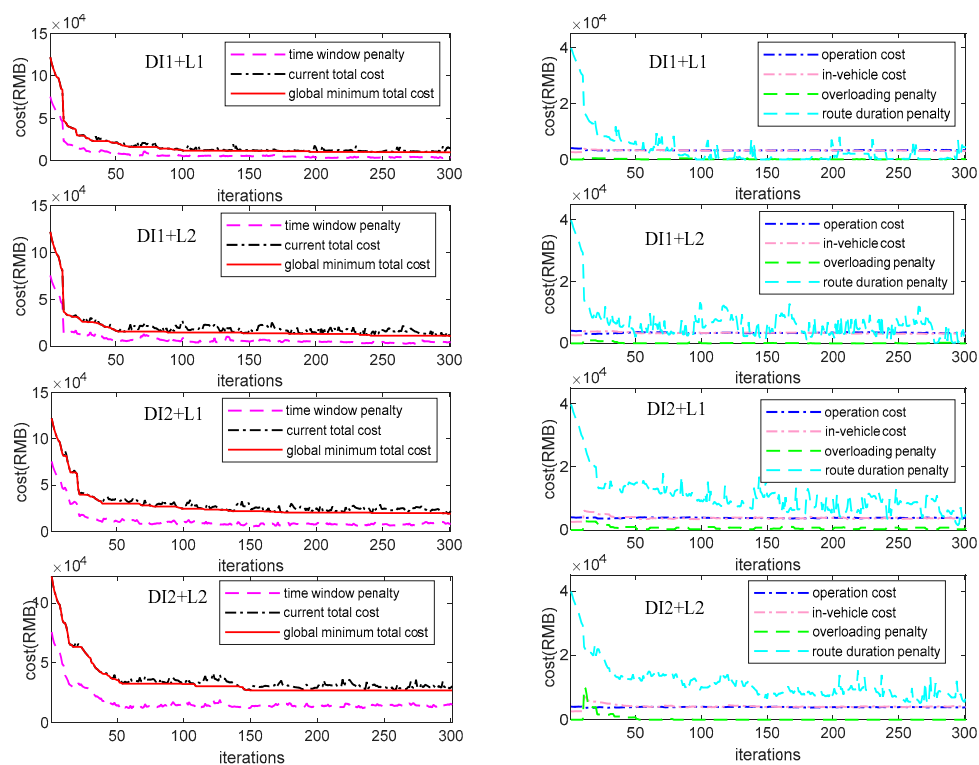


Figure 6. Evolving curves of each experiment by using a tabu search (TS).

Table 2. Detailed results and CPU time of each experiment by using TS.

	C_s	C_I	C_{P1}	C_{P2}	C_{P3}	C_T	CPUtime(s)
DI1+L1	3306.8	3055.3	3497.9	163.0	167.3	10190.4	1384.1
DI1+L2	3240.6	3213.0	4240.0	0	164.4	10858.0	532.2
DI2+L1	3791.1	3608.2	9251.2	261.0	2262.8	19174.6	117.2
DI2+L2	3901.7	3993.7	12409.3	0	6572.4	26877.0	72.6

Note: All the results are obtained by taking the average of the best values for 5 runs.

In Figure 6, the results of each set are presented in two subfigures so that the optimization process of each cost or penalty is presented clearly. In each pair of subfigures of each set, the penalties of the service time window and the route duration take up most of the total cost within the about first 50 iterations. After that, the curves of these two items decrease sharply so that the infeasible solutions with serious violations of the constraints are ensured to be eliminated. Due to the parameters we set, the remaining three items, i.e. the operation cost, the in-vehicle cost, and the overloading penalty account for a small portion of the total cost and keep relatively stable in the whole optimization process. From the left subfigures, it is clear to see that the curves of the global minimum total costs of the 4 cases remain stable after approximately 50 iterations. However, minor fluctuations always exist on the curves of the current total costs even in late iterative processes, especially for the last three sets, i.e. the sets of DI1+L2, DI2+L1, DI2+L2. Therefore, it seems that we may obtain a satisfactory solution fast by using the TS, but the algorithm also shows an inadequate ability in providing continuous improvements in the late iteration process.

In Table 2, the case of DI1+L1 obtains the solution with the lowest objective value, however, it is also the most time-consuming case. Conversely, another extreme performance with the worst solution and the least CPU time occurs in the case of DI2+L2. The results of C_T do not differ much between the cases of DI1+L1 and DI1+L2. However, DI1+L2 saves more than half of the CPU time compared with DI1+L1. This indicates that the use of L2 may also lead to a good solution with much higher efficiency. A much worse solution may be obtained if we use DI2 instead of DI1, which can be confirmed by comparing the former two cases in Table 2 with the latter two ones.

To test the performance of the VNS, 4 sets of experiments with the same combinations of internal operators in the VNS framework are conducted. The evolving processes are presented in Figure 7 where each set is also presented by two subfigures, and the related results are given in Table 3.

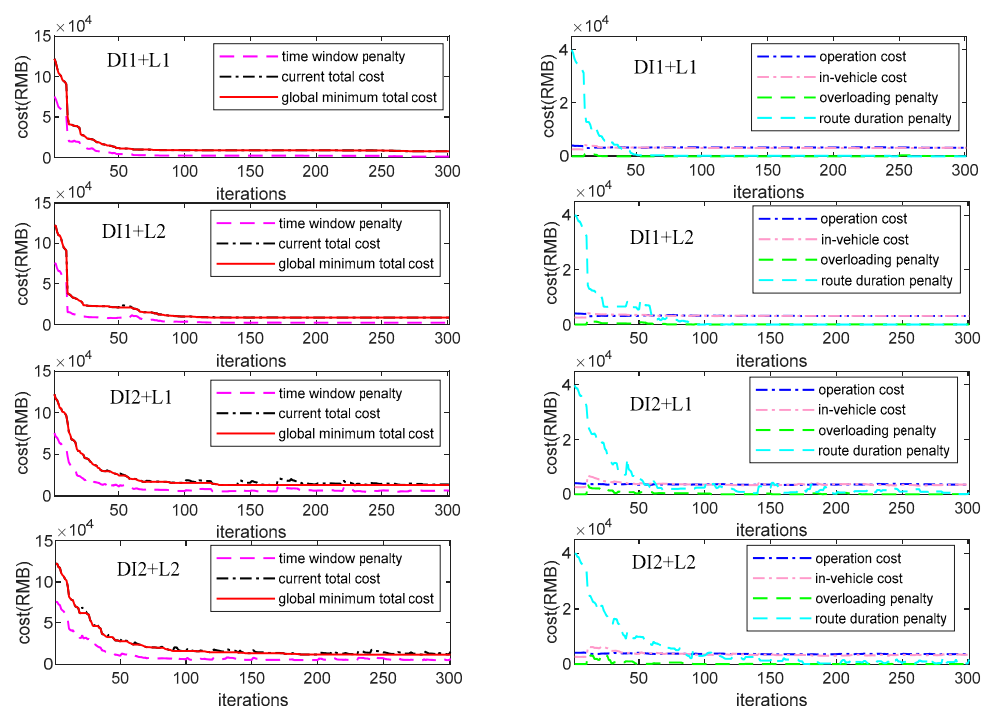


Figure 7. Evolving curves of each experiment by using variable neighborhood search (VNS).

Table 3. Detailed results and CPU time of each experiment by using VNS.

	C_s	C_I	C_{P1}	C_{P2}	C_{P3}	C_T	CPUtime (s)
DI1+L1	3211.7	3173.0	1522.7	104.0	0	8011.4	3683.7
DI1+L2	3147.5	3126.8	2282.2	68.6	0.5	8625.6	1589.3
DI2+L1	3589.9	3350.7	5235.1	0	743.9	12919.6	153.1
DI2+L2	3653.5	3115.9	4445.4	0	220.1	11435.0	95.8

Note: All the results are obtained by taking the average of the best values for 5 runs.

In each pair of subfigures of Figure 7, the penalties of the service time window and the route duration still take up most of the total cost and decrease sharply at the start of the evolving process. In contrast, the curves of the remaining three items keep stable with relatively small values in the whole process. Comparing the curves in Figure 7 with the ones in Figure 6, it is clear that the curves of almost all the items in Figure 7 are more stable with fewer fluctuations, although the global minimum total costs converge slightly slower (approximately within 70 iterations). In particular, in the case of DI1+L1 by using the VNS, the curves of the global minimum total cost and the current total cost are completely coincident. This confirms the effectiveness of the VNS since the current solution is constantly optimized as the iteration processes.

The results in Table 3 indicate similar conclusions about the performance of the four cases, as shown in Table 2. However, the objective values here are less than the ones in Table 2 for all four cases, especially for the latter two cases. By comparing the two tables, the percentages of the total cost reduction in Table 3 are found to be 21.4%, 20.6%, 32.6%, and 57.5%. On the other hand, the VNS costs more CPU time in general. The percentages of the CPU time increase by using VNS are 166.1%, 198.6%, 30.6%, and 32.0%. Therefore, we conclude that DI2 is well suited to the VNS framework since the latter two cases in Table 3 provide much better solutions than the corresponding ones in Table 2 without much CPU time consumption. Moreover, the VNS solutions are less influenced by different combinations of the internal operators than the other solutions since the objective values of the four cases in Table 3 do not differ significantly. However, the VNS seems to be more sensitive from the perspective of CPU time.

To evaluate the performances of the TS and the VNS more comprehensively, 15 sets of experiments (represented by 1a~5c) with different problem scales and time windows are conducted. The number of requests in cases 1~5 are set as 60, 80, 100, 120, and 140. The time windows in subcases a~c are randomly generated based on the normal distribution $N(5, 12)$, $N(10, 22)$, and $N(15, 32)$. Both the TS and the VNS with DI1+L2 are used to solve the 15 sets of problems 5 times. The minimum objective value (denoted by C_T^* and $C_T^{*'}),$ the average objective values (denoted by $\overline{C_T}$ and $\overline{C_T}'$), and the CPU time of the 5 runs by using the TS and VNS of each case are recorded in Table 4. In addition, the percentage of differences between C_T^* and $\overline{C_T}$ (denoted by ΔC_T^* and $\Delta \overline{C_T}$, respectively) solved by the TS and VNS in each case are also given in Table 4.

In Table 4, the CPU time by using each meta-heuristic has a remarkable growth from case 1 to case 5 as the problem scale increases. For each case from case 1 to case 5, the minimum objective values and the average values obtained by the two meta-heuristics both decreases from subcase a to subcase c in general since the intensity of the time window constraint becomes weaker as the time window width increases. Comparing the performances of the TS and VNS, the VNS obtains a better solution in each of the 15 experiments, which can be confirmed by the negative values of ΔC_T^* and $\Delta \overline{C_T}$ in each case. Moreover, the difference between the objective values obtained by the two meta-heuristics increases as the problem scale increases. For the problems with 60 requests (case 1a~1c), ΔC_T^* and $\Delta \overline{C_T}$ are approximately 10%. For the problems with 140 requests (case 5a~5c), the values increase to 30~40%. Therefore, a much better solution may be obtained by the VNS for a large-scale problem. However, the CPU time of the VNS also increases sharply as the problem scale increases. For example, it is found to increase by 4353.9 sec by comparing case 1a with case 5a, while the CPU time of the TS increases

by only 1715.8 sec. This suggests that making a trade-off between the solution quality and CPU time consumption is critical when solving a large-scale problem in practice.

Table 4. Comparison of the results obtained by TS and VNS in each case.

	TS			VNS			ΔC_T^* ^b	$\Delta \overline{C_T}^c$
	C_T^*	$\overline{C_T}$	CPU Time ^a	C_T^*	$\overline{C_T}$	CPU Time ^a		
1a	3957.7	4146.3	83.2	3783.6	3882.2	359.4	−4.4	−6.4
1b	3906.7	4118.2	83.3	3365.4	3390.7	334.2	−13.9	−17.7
1c	3824.2	3973.4	93.5	3667.6	3722.1	343.9	−4.1	−6.3
2a	5788.5	6467.5	218.3	5130.9	5367.4	714.3	−11.4	−17.0
2b	5084.9	5263.1	201.8	4600.1	5063.9	700.4	−9.5	−3.8
2c	4970.2	5160.3	226.9	4409.8	4539.7	750.7	−11.3	−12.0
3a	8866.3	9781.4	458.7	6464.7	7133.2	1366.8	−27.1	−27.1
3b	9250.1	10443.4	539.8	8625.6	9723.0	1506.7	−6.8	−6.9
3c	7023.3	7272.2	453.7	5689.1	6528.0	1409.1	−19.0	−10.2
4a	21273.7	23526.3	1143.0	11346.9	15439.8	2037.0	−46.7	−34.4
4b	14962.7	16593.3	1089.6	11739.2	12231.2	2207.8	−21.5	−26.3
4c	11571.4	13610.6	1089.6	9414.3	10383.4	1916.4	−18.6	−23.7
5a	37463.5	37780.3	1799.0	23258.1	24949.1	4713.3	−37.9	−34.0
5b	24515.9	24820.9	1698.1	14670.7	17515.4	4472.0	−40.2	−29.4
5c	23692.6	25059.6	1727.8	16750.6	17691.2	4536.3	−29.3	−29.4

^a unit of CPU time: sec. ^b $\Delta C_T^* = 100(C_T^* - C_T^*)/C_T^*$. ^c $\Delta \overline{C_T} = 100(\overline{C_T} - \overline{C_T})/\overline{C_T}$.

6. Conclusions

This paper optimizes the routes and the vehicle departure times of community shuttles based on demand-responsive service. By considering the constraints of the time window, vehicle capacity and route duration as soft constraints, an analytical method of determining the vehicle departure time is developed to obtain a competitive solution that can be effectively used in real-world operations. To solve the routing problem, a tabu search (TS) and a variable neighborhood search (VNS) with multiple internal operators are designed. Finally, a case study is presented to test the algorithms followed by analysis. The key conclusions are summarized as follows:

Both the TS and the VNS with different combinations of internal operators provide clear convergence trends within 100 iterations during the evolving process. The curves of the VNS converge slightly slower but are more stable than those of the TS in general. Of the two, the VNS usually obtains a better solution with more CPU time consumption. The gaps in the objective value and the CPU time by using the two metaheuristics widen as the problem scale increases. In the 15 sets of experiments with different numbers of requests and time window widths, the percentage differences of objective values vary from less than 10% to more than 30% as the number of requests increases from 60 to 140. However, the difference in the CPU time also increases sharply in this process.

Two request insert operators (i.e., DI1 and DI2), as well as two sequence reorder operators (i.e., L1 and L2), are developed. DI1 and L1 are used to improve the solutions through additional runs. Therefore, ‘DI1+ DI1’ provides the best solution with the most CPU time, while ‘DI2+ DI2’ is the opposite extreme case. In the experiments involving both the TS and the VNS, L2 is found to be a competitive sequence reorder operator that leads to good solutions with much higher efficiency than L1. In contrast, DI2 is more suitable than DI1 for the VNS framework. From the perspective of solution quality, of the two metaheuristics, the VNS is less influenced by different combinations of the internal operators. However, it is more sensitive from the view of CPU time.

As described above, the model includes an analytical method to determine the optimal vehicle departure time by treating the related constraints as soft constraints. Undoubtedly, the departure time solution is greatly affected by the penalty functions and it may not be effectively used into practice if the relevant parameters are not well calibrated. Besides, the model is developed with a given number of routes. However, the shuttle operators are usually more sensitive to the fleet size of the vehicles they can invest rather than the number of routes. Therefore, future research could be performed by

relaxing the preset number of routes and replacing this parameter with a fleet size constraint. This step would result in a jointly optimal design problem of routing and vehicle scheduling. By solving the problem effectively, some relationships among the trip requests, number of routes, departure time, and fleet size could be further explored.

Author Contributions: Conceptualization, J.X. and B.C.; methodology, J.X.; formal analysis, J.X.; investigation, B.C.; writing—original draft preparation, B.C. and X.L. writing—review and editing, J.X. and Z.H.; supervision, Y.C.; funding acquisition, J.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research is sponsored by National Key Research and Development Program of China (2018YFB1601300), National Natural Science Foundation of China (71601006), Science and Technology Development Foundation of Beijing Municipal Education Commission (KM201710005030).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. He, Z.; Zheng, L.; Chen, P.; Guan, W. Mapping to Cells: A Simple Method to Extract Traffic Dynamics from Probe Vehicle Data. *Comput. Civ. Infrastruct. Eng.* **2017**, *32*, 252–267. [\[CrossRef\]](#)
2. Jiang, S.; Guan, W.; He, Z.; Yang, L. Measuring Taxi Accessibility Using Grid-Based Method with Trajectory Data. *Sustainability* **2018**, *10*, 3187. [\[CrossRef\]](#)
3. Dong, H.; Wu, M.; Ding, X.; Chu, L.; Jia, L.; Qin, Y.; Zhou, X. Traffic zone division based on big data from mobile phone base stations. *Transp. Res. Part C: Emerg. Technol.* **2015**, *58*, 278–291. [\[CrossRef\]](#)
4. Xiong, J.; Guan, W.; Song, L.; Huang, A.; Shao, C. Optimal Routing Design of a Community Shuttle for Metro Stations. *J. Transp. Eng.* **2013**, *139*, 1211–1223. [\[CrossRef\]](#)
5. Xiong, J.; He, Z.; Guan, W.; Ran, B. Optimal timetable development for community shuttle network with metro stations. *Transp. Res. Part C: Emerg. Technol.* **2015**, *60*, 540–565. [\[CrossRef\]](#)
6. Xiong, J.; Chen, B.; Chen, Y.; Jiang, Y.; Lu, Y. Route Network Design of Community Shuttle for Metro Stations Through Genetic Algorithm Optimization. *IEEE Access* **2019**, *7*, 53812–53822. [\[CrossRef\]](#)
7. Amirgholy, M.; Gonzales, E.J. Demand responsive transit systems with time-dependent demand: User equilibrium, system optimum, and management strategy. *Transp. Res. Part B: Methodol.* **2016**, *92*, 234–252. [\[CrossRef\]](#)
8. Agatz, N.; Erera, A.; Savelsbergh, M.; Wang, X. Optimization for dynamic ride-sharing: A review. *Eur. J. Oper. Res.* **2012**, *223*, 295–303. [\[CrossRef\]](#)
9. Ho, S.C.; Szeto, W.; Kuo, Y.-H.; Leung, J.M.; Petering, M.; Tou, T.W. A survey of dial-a-ride problems: Literature review and recent developments. *Transp. Res. Part B: Methodol.* **2018**, *111*, 395–421. [\[CrossRef\]](#)
10. Røpke, S.; Pisinger, D. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transp. Sci.* **2006**, *40*, 455–472. [\[CrossRef\]](#)
11. Røpke, S.; Cordeau, J.-F. Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows. *Transp. Sci.* **2009**, *43*, 267–286. [\[CrossRef\]](#)
12. Irnich, S. A multi-depot pickup and delivery problem with a single hub and heterogeneous vehicles. *Eur. J. Oper. Res.* **2000**, *122*, 310–328. [\[CrossRef\]](#)
13. Landrieu, A.; Mati, Y.; Binder, Z. A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. *J. Intell. Manuf.* **2001**, *12*, 497–508. [\[CrossRef\]](#)
14. Naccache, S.; Côté, J.-F.; Coelho, L.C. The multi-pickup and delivery problem with time windows. *Eur. J. Oper. Res.* **2018**, *269*, 353–362. [\[CrossRef\]](#)
15. Haddad, M.N.; Martinelli, R.; Vidal, T.; Martins, S.; Ochi, L.S.; Souza, M.J.F.; Hartl, R. Large neighborhood-based metaheuristic and branch-and-price for the pickup and delivery problem with split loads. *Eur. J. Oper. Res.* **2018**, *270*, 1014–1027. [\[CrossRef\]](#)
16. Qu, Y.; Bard, J.F. The heterogeneous pickup and delivery problem with configurable vehicle capacity. *Transp. Res. Part C Emerg. Technol.* **2013**, *32*, 1–20. [\[CrossRef\]](#)
17. Masson, R.; Lehuédé, F.; Péton, O. The Dial-A-Ride Problem with Transfers. *Comput. Oper. Res.* **2014**, *41*, 12–23. [\[CrossRef\]](#)
18. Reinhardt, L.B.; Clausen, T.; Pisinger, D. Synchronized dial-a-ride transportation of disabled passengers at airports. *Eur. J. Oper. Res.* **2013**, *225*, 106–117. [\[CrossRef\]](#)

19. Detti, P.; Papalini, F.; De Lara, G.Z.M. A multi-depot dial-a-ride problem with heterogeneous vehicles and compatibility constraints in healthcare. *Omega* **2017**, *70*, 1–14. [[CrossRef](#)]
20. Tong, L.; Zhou, L.; Liu, J.; Zhou, X. Customized bus service design for jointly optimizing passenger-to-vehicle assignment and vehicle routing. *Transp. Res. Part C: Emerg. Technol* **2017**, *85*, 451–475. [[CrossRef](#)]
21. Braekers, K.; Caris, A.; Janssens, G.K. Exact and meta-heuristic approach for a general heterogeneous dial-a-ride problem with multiple depots. *Transp. Res. Part B Methodol.* **2014**, *67*, 166–186. [[CrossRef](#)]
22. Parragh, S.N.; Schmid, V. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Comput. Oper. Res.* **2013**, *40*, 490–497. [[CrossRef](#)] [[PubMed](#)]
23. Parragh, S.N.; Doerner, K.F.; Hartl, R.F. Variable neighborhood search for the dial-a-ride problem. *Comput. Oper. Res.* **2010**, *37*, 1129–1138. [[CrossRef](#)]
24. Ibaraki, T.; Imahori, S.; Kubo, M.; Masuda, T.; Uno, T.; Yagiura, M. Effective Local Search Algorithms for Routing and Scheduling Problems with General Time-Window Constraints. *Transp. Sci.* **2005**, *39*, 206–232. [[CrossRef](#)]



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).