Data-driven Flexible Buses Scheduling and Path Optimisation

Yongxuan Lai\textsuperscript{a,b,*}, Zheng Lv\textsuperscript{a}, Fan Yang\textsuperscript{b,c}, Tian Wang\textsuperscript{d}

\textsuperscript{a}School of Software, Xiamen University, Xiamen 360000, China.
\textsuperscript{b}Shenzhen Research Institute, Xiamen University, 19 Gaoxin Nansi Road, Nanshan District, Shenzhen 518000, China.
\textsuperscript{c}Department of Automation, Xiamen University, Xiamen 360000, China.
\textsuperscript{d}College of Computer Science and Technology, Huaqiao University, Xiamen 360000, China.

Abstract

This template helps you to create a properly formatted \LaTeX\ manuscript.

Keywords: elsarticle.cls, \LaTeX, Elsevier, template

2010 MSC: 00-01, 99-00

1. Introduction

Vehicle routing and scheduling is a critical part of the design and operations of any transport system. For traditional fixed-route transit systems where all transit vehicles stop at every station, schedules are easily developed for operational purposes. Other types of transit systems such as demand-responsive transit systems (DRT), however, are more complicated than traditional fixed-route transit systems, and if proper care is not given to the routing and scheduling of the service, very expensive and unreliable schedules may result with a high risk of failing to meet the required level of service expected by the passengers.

For Flex-Bus system, the routing and scheduling task becomes even more complex due to the existence of fixed-route and demand-responsive components in the operation of the service. The Flex-Bus system mainly contains two works.

\textsuperscript{*}This research is supported by the Natural Science Foundation of China (61672441), the Shenzhen Basic Research Program (JCYJ20170818141325209), the National Key Technology Support Program (2015BAH16F01), the State Scholarship Fund of China Scholarship Council (201706315020).

\textsuperscript{*}Corresponding author: Yongxuan Lai. E-mail: laiyx@xmu.edu.cn.
Firstly, vehicle sets the objective and some constraints, and then generate a flexible bus route based on the real-time request data of passengers. And the second one is to make a schedule for the operation of these vehicles which travel along flexible routes.

The problem of finding optimal route for receiving real-time request of passengers, also known as the Dial-A-Ride Problem (DARP), can be considered an expansion of the traveling salesman problem, which involves the calculation of an optimum journey for visiting a number of predetermined nodes on a network. [22, 23] are build model and take heuristic algorithm to solve DARP respectively, but they are focus more on bus travel time rather than revenue. We designed two heuristic algorithms in this paper which with more consideration of operation revenue. [6] proposed an path planning strategy that focuses on a limited potential search area for each vehicle by filtering out the requests that violate passenger service quality level, so that the global search is reduced to local search. But in this paper, the potential search area will change in real time according to the position of the bus, which leads to a lot of time is used to calculate the area of PSA. In this paper, we set a fixed search area between every two adjacent backbone stops to reduce the amount of calculation.

The traditional way to make a schedule for a bus is to first set the departure interval based on historical data, and then adjust the schedule at any time based on the actual arrival time of the bus arrive the destination and the experience of dispatchers. But this way may not get best performance on revenue when there are a large number of requests. In addition, when there are fewer requests, it may leave a lot of empty seats on the bus, which resulting in a waste of resources. So it’s necessary to find a way which make a schedule to reduce the gap between the demand and supply of bus seats. [21] built up a vehicle scheduling model based on minimum length of total transportation distance, but the author doesn’t consider the waste of bus seat resources.

To achieve vehicles could respond to real-time requests from passengers at any time, and change the bus route flexibly with the goal of maximising operating income and try to minimise energy consumption during bus travel. We
proposed two heuristic Origin-Destination insertion algorithms, named GTLP and GNPP, which generate vehicle route based on real-time request data while giving priority to time loss and the number of requests, respectively. Then we constructed a MILP model of scheduling with consideration of bus seat resource utilization. Experiments show that, compared with the traditional vehicle travel method, when the vehicle takes the GTLP and GNPP schemes, the total number of requests received and the per capita ride time have been significantly improved. The results of solving the MILP model also show that the new scheduling scheme proposed in this paper can significantly reduce the gap between the demand and supply of bus seats.

2. Related Work

Demand-responsive transit service is an alternative travel method to personal vehicles, carpool/vanpool and regular transit service. It is comprised of a number of customer requests that need to be served door-to-door or curb-to-curb by a set of vehicles [1, 2].

One important issue in demand-responsive transit service is to devise a real-time matching algorithm that determines the best vehicle (taxi, cab, bus) to satisfy incoming service requests. Ma et al. [3] proposed a taxi searching algorithm using a spatio-temporal index to quickly retrieve candidate taxis that are likely to satisfy a user request. The algorithm checks each candidate taxi and inserts the query’s trip into the schedule of the taxi that satisfies the query with minimum additional incurred travel distance. Based on [3], Ma et al. [4] reported a real-time taxi-sharing system based on the mobile-cloud architecture. Drivers and passengers exchange service and demands using an application installed on their smartphones, and the taxi that minimizes the increased travel distance of the ride request would be selected to pick up the new passenger. Chen et al. [9] proposed rules to build and prune the directed bus route graph. Based on the graph, they proposed a new heuristic algorithm, named bidirectional probability-based spreading (BPS) algorithm, to select the best bidirec-
tional bus route that can achieve the maximum number of passengers expected in two directions. Zhu et al. [5] proposed a heuristic precedence constrained origin-destination insertion algorithm for the public vehicle system to minimize vehicles’ total travel distance with service guarantee such as low detour ratio. Based on [5], the same authors proposed a path planning strategy that focuses on a limited potential search area for each vehicle by filtering out requests that violate passenger service quality level [6], and studied the joint transportation and charging scheduling for PV systems to balance the transportation and charging demands, ensuring the long-term operation [7]. To ensure real-time scheduling, Zhu et al. [8] proposed one efficient path planning strategy with balanced QoS by restricting search area for each PV, meanwhile they proposed potential search area, where the origins or/and destinations of possible requests whose QoS may be in constraints, while the other requests not in this PSA should be excluded. More recently, Cheng et al. [10] formulated the utility-aware ridesharing problem on road networks. It assigns time-constrained riders to capacity-constrained vehicles to maximize the entire utility value, which includes the vehicle-related utility, the riders-related utility, and the trajectory-related utility.

The aforementioned schemes assume a centralised server to execute algorithms for demand-responsive transit service, and the desired metadata are available as input for the algorithms. Different from those algorithms, the proposed scheme focuses on distributive processing of the algorithm, where matching and scheduling algorithms are executed on the fog nodes, and hence avoids a bottleneck of computing and storage. Moreover, the proposed scheme integrates the metadata gathering into the whole framework, where data is gathered and stored at distributed fog nodes in the edge of networks.

Demand-responsive transit service could be abstracted as a member of the general class of the Dial-a-Ride Problem [11, 12], which focuses on scenarios of planning schedules for vehicles, subject to the time constraints on pickup and delivery events. The proposed approach dispatches requests to distributed fog nodes that maintain lists of vehicles, which actually partitions a large dial-a-ride problem into multiple smaller ones that are easier to solve. Hu et al. [22] develop
a multi-objective model with considerations of service quality, eco-efficiency and speed level constraint to solve Dial-a-Ride Problem, [23] proposes a hybrid evolutionary heuristic for the dial-a-ride problem with time windows (DARPTW).

Zhang et al. [13] proposed a series of evaluation indicators for bus routes and provided a clear guide for the evaluation at the single route level.

3. Model Description

In flex route transit service, vehicles travel among stations while responding to demands. A vehicle, e.g. $c$, follows the following steps to provide the public transit service:

1. Routes are flexible and may alter with time. When vehicle $c$ arrives station $u$, the flex route system generates route $u \rightarrow v$ based on the historical transportation flows, where $v$ is another station.
2. During travelling along route $u \rightarrow v$, $c$ receives on-demand requests within its service area. A request might be accepted or rejected by the vehicle.
3. If the vehicle accepts a request, it travels to the place and picks up the rider; else, it sends a reject message to the rider.
4. Vehicle $c$ arrives $v$ before the running time of the route. Then the flex route system generates a new route $v \rightarrow w$ at time $t'$, which begins a new cycle from step 1 to 4 and $w$ could be any station.

In this section we introduce some concepts and definitions in this model.

3.1. Station and Stops

Vehicles park, pick up and drop off riders at stations. The set of stations are denoted by $S$. Vehicles travel among these stations to provide the transportation service.

Vehicles also pick up and drop off demand responsive riders at stops, which are temporary locations between stations. We denote the set of all possible stops as $E$. 
Both stations and stops could be predefined, or extracted from historical trajectories. In this research we adopt a clustering approach to identify stations and stops.

3.2. Flex Route

Symbol \( r(u, v, t) \) denotes a route from station \( u \) to station \( v \) starting at the scheduled departure time \( t \). When a vehicle is assigned to route \( r(u, v, t) \), it travels along path from \( u \) to \( v \). We denote the path from \( u \) to \( v \) through the shortest path by \( u \rightarrow v \), and its travelling time is denoted by \( tt(r) \). Yet a vehicle along the route would respond to riders’ requests, so it would go to pickup the riders. The actual travelling path is denoted by \( u \hookrightarrow v \), and its actual running time is denoted by \( at(r) \). Also, a route has a scheduled running time \( rt(r) \), which means the vehicle along route \( r \) should arrive \( v \) before the scheduled time \( rt(r) \). The following formula holds:

\[
\text{tt}(r) \leq \text{ar}(r) \leq \text{rt}(r)
\]  

(1)

Here we assume the times \( tt(r), ar(r), rt(r) \) take a predefined time slot as the unit. A time slot is denoted by \( U \) and it could be 5 or 10 minutes. The actual running time would increase as new requests are inserted into the route. So a request would be rejected when, if it is accepted, the actual running time is larger than the scheduled time.

For consistency we also define a wait route \( r(u, u, t) \) that has the same departure and destination station. Vehicle assigned to this route would stay at \( u \), and wait for the time slot \( t \) before being assigned to another route. The running time \( rt(r) \) for this route is a unit time slot \( U \).

3.3. Slack Time

Slack time is denoted by \( st(r) \) and defined as follows:

\[
st(r) = rt(r) - tt(r)
\]  

(2)
It is the extra time to serve on-demand requests within the service area of the route. Also, the slack ratio is denoted by $\alpha$:

$$\alpha = \frac{st(r)}{tt(r)}$$

(3)

In this study we assume $\alpha > 0$ is a predefined parameter for all the routes in the flex transit system. The running time could be calculated by the following formula:

$$rt(r) = \lceil tt(r) \ast (1 + \alpha) \rceil$$

(4)

where $\lceil x \rceil$ denotes the ceiling of the $x$ in unit time slot.

3.4. Request

A request is denoted by $req(t, o, d, w)$, where $t$ is the time when the request is submitted, $o$ is the pickup location, $d$ is the drop off location, and $w$ is the constraint time window for the pickup. A request might either be accepted or rejected by the vehicle.

3.5. Service Area

Service area is usually represented by an extended rectangle area along path $u \rightarrow v$. The width of service area defines how far away from the standard route...
a vehicle may deviate to pick up or drop off passengers. Fig. 3 illustrates a flex route $u \rightarrow v$ and its service area.

3.6. Operation Policies

The vehicles are not required to follow a specific route and could have a different route from time to time. The only constraint in the service is that all flex routes are required to start and end at stations, and depart and arrive within its their scheduled running time.

We assume riders could get on and off vehicles at the departure and destination stations. They could also issue demand-responsive requests so that they could get on and off vehicles at some predetermined locations. We call these locations the dynamic stops, which are extracted from the trajectories logs. Request $req(t, o, d, w)$ would be transformed to $req(t, o', d', w)$, where $o', d'$ are the nearest dynamic stops to locations $o, d$. The flex rout system would calculate whether a request is compatible with a request. A request $req(t, o, d, w)$ is compatible with a route $r$ if it meets the following conditions:

$$t + w(l_c, o') \in r.w;$$  \hspace{1cm} (5)

$$\text{cost(path}(r)_{o',d'}) \leq rt(t)$$ \hspace{1cm} (6)

where $l_c$ is the current location of vehicle, $w(l_c, o')$ is the time cost of travelling from $l_c$ to $o'$, $\text{path}(r)_{o',d'}$ is the path after inserting $o'$ and $d'$ on route $r$. [5] means the vehicle should travel to $o'$ to picks up the rider on its constraint time window $w$, [6] means the total travelling time of the path after inserting $o', d'$ should be within the running time of route $rt(r)$.

The flex route system would accept a request if the request is compatible to the route. Then the system would send an “accept” message to the rider and guide the rider to walk to $o'$ for the pickup. And the vehicle would drop off the rider at $d'$, where the rider could walk to his/her destination. If the request is not compatible to the route, the system would send a “reject” message to the rider immediately.
4. Modeling

Our models use rectilinear movement because it is a good approximation of a realistic road network (Quadrifoglio et al., 2008b). The service vehicle travels at a velocity of $V_b$ and each stop features a dwell time of $T_s$. In this section, the analytical models and simulations are developed to evaluate the designed transit service.

4.1. System Performance Measures

We aim to study the feasibility of a public vehicular system that provides another public transportation method other than the bus or responsive taxies. So in this study we use the energy efficiency ($ee$), rider delivery ratio ($dr$), and the average walking distance ($wd$) as three main metric for the performance measures.

The energy efficiency is defined as follows:

$$ ee = \frac{\sum_{r \in D} l(r.o, r.d)}{\sum_{c \in F} tl(c) \ast \pi(c)} $$

where $l(r.o, r.d)$ is the distance of shortest path from the origin $r.o$ to the destination $r.d$, $tl(c)$ is the total traveling distance of vehicle $c$, and $\pi(c)$ is the average fuel consumption of $c$. The rider delivery ratio ($dr$) is calculated as:

$$ dr = \frac{|D|}{|R|} $$

where $|D|$ is the number of successful deliveries. The average walking distance $wd$ is calculated as:

$$ wd = \frac{\sum_{r \in D} l(r.o, r.\tilde{o}) + l(r.d, r.\tilde{d})}{|D|} $$

where $r.\tilde{o}$, $r.\tilde{d}$ are the real pickup and drop off locations for request $r$.

Other factors such as the average waiting time and detour ratio are also important indicators of the QoS (quality of service) of passengers. We would discuss them at the experimental analysis.
4.2. Flexible Route Scheduling

Given a flex route from station $u$ to $v$, the operation time domain is split into $n$ slots. For example, if there are 16 hours of operation time for the route, and each slot is two minutes, then there are $n = 480 = 16 \times 30$ time slots. $x_{ij}^u$ denotes the number of buses that are scheduled to depart from $u$ at interval $i$ and arrives $v$ at interval $j$. Similarly, $x_{ij}^v$ denotes the number of buses that are scheduled to depart from $v$ at interval $i$ and arrives $u$ at interval $j$. Then the scheduling is modelled as a Mixed-Integer Linear Programming (MILP) problem based on the historical demand of flows:
minimize : \[ \sum_{i}^{n} \sum_{j}^{n} |F_{ij}^u - K \times x_{ij}^u| + |F_{ij}^v - K \times x_{ij}^v| \] (10) subject to: \[ s_{i+1}^u = s_i^u - x_{ij}^u + \sum_{z=1}^{n} x_{z+1}^v, \quad i = 1..n - 1 \] (11) \[ s_{i+1}^v = s_i^v - x_{ij}^v + \sum_{z=1}^{n} x_{z+1}^u, \quad i = 1..n - 1 \] (12) \[ s_i^u + s_i^v \leq N \] (13) \[ 0 \leq s_i^u \leq cap^u, \quad i = 1..n \] (14) \[ 0 \leq s_i^v \leq cap^v, \quad i = 1..n \] (15) \[ x_{ij}^u, x_{ij}^v \geq 0, \quad i = 1..n, \quad j = 1..n \] (16) The objective (10) is to minimise the gap between the demand and supply of bus seats. \( F_{ij}^u \) is the number of demand flow that departs from station \( u \) at interval \( i \) and arrives at station \( v \) at interval \( j \); \( F_{ij}^v \) is the number of demand flow that departs from station \( v \) at interval \( i \) and arrives at station \( u \) at interval \( j \). \( F_{ij}^u \) and \( F_{ij}^v \) are given as constant variables for the model and we will discuss their calculation in the next section. \( K \) is the supply of seats in a single bus, which could be empirically calculated by \( cap \times \kappa \), where \( cap \) denotes capacity of the bus and \( \kappa \geq 1 \) denotes the empirical factor as there are get-ons and get-offs during the trip.

\( s_i^u \) denotes the number of buses at station \( u \) at interval \( i \). Constraints (11) and (12) imply the change of buses at \( u \) and \( v \) at interval \( i \) by subtracting the departed buses and adding the arrived buses. Constraint (13) ensures at the very beginning buses at station \( u \) and \( v \) are within the range of \( N \), which is the number of vehicles of the fleet. Constraints (14) and (15) imply the number of buses at station \( u \) and \( v \) should be greater than zero and smaller than the capacity of the stations, i.e. \( cap^u \) and \( cap^v \). Constraint (16) defines the value of the decision variables \( x_{ij} \) and \( y_{ij} \), which should be zero or positive integers.

The flow of trips is a key factor when scheduling the routes, which are calculated based on the OD dataset. As mentioned previously, the OD dataset
contains origin and destination GPS points, and the points are indexed by a set of grids on the map. We further split the OD pairs by time slots, so the set of OD pairs are stored and indexed according to spatial and temporal dimensions.

We use a table, denoted by \( \Gamma(z) \), for the storage, and each tuple is in the form \(< t, g_o, g_d, num >\), where \( z \) is the grid, \( t \) is the time slot, \( g_o, g_d \) denotes the grid of the origin and destination respectively, and \( num \) is the number of the corresponding OD pairs. Note that for a grid \( z \), \( g_o \) equals \( z \) for all the tuples it stores. Table ?? is an example illustrating the OD table in a grid.

Supposed the route under consideration is \( r(u, v, t) \), which is from station \( u \) to \( v \) and departs at time \( t \). Then the trip flow \( F_t \) is then calculated as follows:

1. Split the service area into subareas. \( K \) denotes the number of subareas and is calculated as:

\[
K = \left\lfloor \frac{tt(r) \ast \alpha}{U} \right\rfloor
\]

(17)

where \( tt(r) \) is the travelling time of \( r \) and \( \alpha(r) \) is the slack ratio, \( U \) is the length of time slot unit. So the service area is equally split into \( K \) rectangle subareas \( \{A_1, A_2, \ldots, A_K\} \), which is illustrated in Fig. 3. (2) For each subarea \( A_i \), get the set of its covered grids \( G_i \). A grid \( g \) is covered by \( A_i \) if it the following condition holds:

\[
\frac{ar(A_i \cap g)}{ar(g)} \geq \theta
\]

(18)
where \( ar(X) \) denotes the area of polygon \( X \), \( \theta \in [0, 1] \) is the a predefined threshold.

(3) For each grid \( z \) within subarea \( A_i \), calculate its trip flow that begins at time \( t \). The flow of a grid is defined as \( fg(z, t) \):

\[
fg(z, t) = \sum_{x \in \Gamma(z)} x.num, \quad x.t \in slot(t, i), \quad x.gd \in G
\]

\[
slot(t, i) = [t + (i - 1) \times U, t + i \times U], \quad i = 1, 2, ..., K
\]

(19)

where \( x \) is a tuple in table \( \Gamma(z) \), \( G = G_1 \cup G_2 ... \cup G_K \) is the set of all the grids in the service area of \( r' \), \( slot(t, i) \) is the \( i^{th} \) time slot that begins at \( t \) and has an interval \( U \). Then the weight of flow in subarea \( A_i \) that begins at time \( t \) is defined as \( fw(A_i, t) \):

\[
fw(A_i, t) = \sum_{z \in G_i} gw(z, t)
\]

(20)

(4) Calculate the demand flow from \( u \) to \( v \) at time \( t \), which is defined as follows:

\[
f_t = \sum_{i=1}^{K} fw(A_i, t)
\]

(21)

And \( f_t \) is mapped to \( F_{ij}^u \) as follows:

\[
F_{ij}^u = \begin{cases} 
  f_t, & i = to_slot(t) \text{ and } j = to_slot(t + rt(r)) \\
  0, & i \neq to_slot(t) \text{ or } j \neq to_slot(t + rt(r))
\end{cases}
\]

(22)

where \( to_slot(t) \) is a function that maps time \( t \) to the index of time slot, \( rt(r) \) is the running time of bus that travels along route \( r(u,v,t) \). Similarly, flow \( F_{ij}^v \) is calculated based on route \( r(v,u,t) \) that departs \( v \) for \( u \) at time \( t \).

4.3 Optimal Path of Flex Route

When a route, e.g. \( r(u,v,t) \) is scheduled, information about its departure time and destination would be notified to potential riders. A rider might either go to the static stops to get on the bus, or just send a request \( req(t, o, d, w) \)
trying to be picked up, where the request might be accepted or rejected by the flex vehicle system.

In this section, we first introduce the identification of stops, and then defines the problem of path finding for the flex routes.

![Figure 4: Illustration of a route graph and the actual path of flex route. The path consists of backbone stops, which are selected based on historical OD data, and the ad-hoc stops, which are determined in real time.](image)

4.3.1. Stops Identification

We adopt a data-driven approach in this study. The pattern of OD (origin-destination) pairs are mined to identify dynamic stops. With a large number of origins and destinations of the travel demands, we could cluster these points collectively to represent potentially meaningful places. These places are the potential locations for the stops.

The shared nearest neighbors (SNN) is adopted as the basis of distance measure between two GPS points. Given two points A and B, the distance is defined as:

\[
dist(x, y) = 1 - \frac{w(N_k(x) \cap N_k(y))}{w(N_k(x) \cup N_k(y))}
\]

where \(N_k(x)\) is the set of \(k\) nearest neighbours of \(x\), \(w(Q)\) is the total weight of points in set \(Q\). This distance meets several requirements for the spatial
### Table 1: Notations in the OPFR problem.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x^+_i$</td>
<td>number of requests picked up at $i$</td>
</tr>
<tr>
<td>$x^-_i$</td>
<td>number of requests dropped off at $i$</td>
</tr>
<tr>
<td>$f^+_i$</td>
<td>number of picked up requests at $i$ (i is not included)</td>
</tr>
<tr>
<td>$f^-_i$</td>
<td>number of requests dropped off at $i$ (i is not included)</td>
</tr>
<tr>
<td>$a_{i,j}$</td>
<td>is 1 if $(i,j)$ is on the path; else 0</td>
</tr>
<tr>
<td>$b_{i,j}$</td>
<td>is 1 if both $i$ and $j$ are on the path and $i$ precedes (not necessarily immediately) $j$; else 0</td>
</tr>
<tr>
<td>$e_0$</td>
<td>number of on-board riders at $u$</td>
</tr>
<tr>
<td>$c_p$</td>
<td>capacity of vehicle</td>
</tr>
<tr>
<td>$num^+(i)$</td>
<td>number of desired pickups at $i$</td>
</tr>
<tr>
<td>$num^-(i)$</td>
<td>number of desired drop offs at $i$</td>
</tr>
<tr>
<td>$t(i,j)$</td>
<td>travelling time from $i$ to $j$</td>
</tr>
<tr>
<td>$V$</td>
<td>set of stops within service area plus $u$ and $v$</td>
</tr>
</tbody>
</table>

Clustering of origin/destination points: 1) a cluster would meet a minimum size constraint $k$, and each cluster is spatially contiguous; 2) it preserves the data resolution by constructing as many clusters as possible; 3) it identifies clusters of different point densities and different shapes. So the summary statistics (e.g. net flow ratio) for each cluster is meaningful and usually stable.

#### 4.3.2. OPFR Problem

Given a route $r(u,v,t)$, and a set of stops $Q$ within the service area $W$, we define finding the optimal path of this route as the OPFR problem (Optimal Path of Flex Route).

First we model the stops in the service area as vertices in a directed graph $G_r(V,E)$, which is also called the *route graph*. $V$ is defined as $\{u,v\} \cup Q$, an edge $(i,j)$ is added to $E$ if the distance $l(i,j)$ is less than a threshold $\tau_3$, where $i,j \in V$. Then finding a path in the service area could be model as a Mixed-
Integer Linear Programming (MILP) problem:

\[
\text{maximize : } \sum_{i \in V} x_i^+ \quad (24)
\]

\[
x_i^+ \in [0, \text{num}^+(i)] \quad (25)
\]

\[
x_i^- \in [0, \text{num}^-(i)] \quad (26)
\]

\[
f_i^+ + x_i^+ = f_j^+ \quad (27)
\]

\[
f_i^- + x_i^- = f_j^- \quad (28)
\]

\[
\sum_{(u,j) \in E} a_{i,j} = 1, \ j \in V - v \quad (29)
\]

\[
\sum_{(i,v) \in E} a_{i,j} = 1, \ i \in V - u \quad (30)
\]

\[
\sum_{(i,j) \in E} a_{i,j} \leq 1, \ i \in V - v \quad (31)
\]

\[
\sum_{(i,j) \in E} a_{i,j} \leq 1, \ j \in V - u \quad (32)
\]

\[
b_{i,j} \geq a_{i,j}, \ i,j \in V \quad (33)
\]

\[
b_{i,j} + b_{j,i} \leq 1, \ i,j \in V \quad (34)
\]

\[
b_{i,j} + b_{j,i'} + b_{j',j} \leq 2, \ i,j \in V \quad (35)
\]

\[
e_0 + f_i^+ - f_i^+ + x_i^+ - x_i^- \leq c_p, \ i \in V \quad (36)
\]

\[
\sum_{(i,j) \in E} a_{i,j} * t(i,j) < rt(r) \quad (37)
\]

\[
a_{i,j}, b_{i,j} \in \{0, 1\} \quad (38)
\]

\[
f_i^+, f_i^- \geq 0 \quad (39)
\]

where Table 1 denotes the meanings of the symbols. The goal (24) is to maximise the number of served requests. Constraint (25) and (26) ensure that the vehicle selectively picks up or drops off riders that belong to that stop. Constraint (27) and (28) imply the total number of picked or dropped riders when vehicle traverses the edge \((i,j)\). Constraint (29) and (30) means the path should start at \(u\) and end at \(v\). Constraint (31) means any location from \(V - u\) has one successor, and (32) means any location from \(V - v\) has one precursor. Constraint
implies the relationship between $a_{i,j}$ and $b_{i,j}$, which could be inferred from their definition. Constraint (34) implies that $(i,j)$ and $(j,i)$ could not both on the path, constraint (35) implies no circles on the path. Constraint (36) ensures the number of riders is smaller than the vehicle capacity at any stop. Constraint (37) ensures that the cost of travelling time is less than the running time. Finally, (38) and (39) constraint define the nature of the decision variables.

4.4. Origin-Destination Insertion based on Backbone Stops

The PFRP problem is a dial-a-ride problem and NP-Complete. When realtime requests are received by the flex route system, the set of $R$ changes accordingly, and the running time varies with the time. So in real situations the problem has larger complexity with dynamic finite capacity and with more constraints (e.g., time). In this section we present a heuristic algorithm that integrates the historical OD patterns to construct a path for the flex route.

4.4.1. Backbone Stops

The service area of a flex route consists of $K$ rectangle subareas. For each subarea $A_i$ of route $r(u,v,t)$, we define a backbone stop $bs_i$:

$$bs_i = \arg\max_s \{\beta * N_0(s) + (1 - \beta) * N_1(s) : s \in S_i\}$$

$$N_0(s) = num^+(s) + num^-(s), \quad N_1(s) = gw(grid(s),t)$$

where $\beta \in [0,1]$ is a balance factor, $S_i$ is the set of stops within the subarea $A_i$. $N_0(s)$ is the number of currently received pickup and drop off requests at stop $s$, $N_1(s)$ is the flow weight of $grid(s)$, which is denoted by $gw(grid(s),t)$ and defined at (19). Here $grid(s)$ is the grid that $s$ belongs to.

The set of backbone stops together with the stations $u,v$ are denoted by $BS = \{bs_0, bs_1, bs_2, ..., bs_K, bs_{K+1}\}$, where $bs_0 = u, bs_{K+1} = v$. Stops in $BS$ are arranged in topological order of the directed acyclic graph, and the shortest paths $\{bs_i \rightarrow bs_{i+1}\}$ between $bs_i$ and $bs_{i+1}$, $i = 0, ..., K$ could be calculated. So the initial path for the flex route is generated by iteratively connecting the backbone stops and their temporary stops between them. As illustrated in Fig.
(a), the shortest path between \(a\) and \(e\) is \((a - d - e)\), so \(d\) is also added to the path of the route. We denote the path of route \(r\) by \(\text{path}(r)\) and denote the set of all stops in \(\text{path}(r)\) by \(S(\text{path}, r)\). As there is only one backbone stop at each subarea within the service area, we assume that initial \(\text{path}(r)\) would always satisfy the running time constraint.

### 4.4.2. Path Insertion

The initial path is then extended, i.e. new pickup and drop off stops are inserted into the path, as new requests are coming in. Give a request \(\text{req}(o, d, w)\), suppose \(o'\) and \(d'\) is the nearest stop to \(o\) and \(d\), and \(d'\) is behind \(o'\) in topological order, there are three cases when inserting stops to the path:

1. When both \(o'\) and \(d'\) are in set \(S(\text{path}, r)\), the request is immediately accepted. This is because the insertion of request does not add extra cost to the path. If \(o'\) or \(d'\) is not a backbone stop, it becomes a backbone stop, and the stop is moved to \(BS\).

2. When only one stop, either \(o'\) or \(d'\), is in \(BS\), the flex route system would check whether the stop is feasible to be added to the path. Without loss of generality, suppose the drop off stop \(d'\) is already in the path, yet the pickup stop \(o'\) is to be checked. Suppose the subarea that contains \(o'\) is \(A_i\), and the set of backbone stops in \(\text{path}(r)\) contained in \(A_i\) is \(BS(A_i) = \{bs_{j+1}, bs_{j+2}, \ldots, bs_{j+m}\}\), then the possible insertion positions are: \(I_0, I_1, I_2, \ldots, I_m\), where \(I_k = (bs_{j+k}, bs_{j+k+1})\), \(k = 0, 1, \ldots, m\), and \(bs_j\) is the last backbone stop at subarea \(A_{i-1}\). Fig. 9 illustrates an example of an insertion into the path, where \((a, e), (e, f), (f, g)\) are the possible insertion positions for stop \(c\).

For every possible insertion position, a new path is built to contain the new requested stop \(o'\). The new path after insertion at \(I_x\) is denoted by \(\text{path}(r)_x\), and the insertion position is selected by following formula:

\[
k_1 = \arg\min_x \{\text{cost}(\text{path}(r)_x) : \text{cost}(\text{path}(r)_x) < rt(r), \quad x = 0, \ldots, m\}
\]
Algorithm 1: Origin-Destination Insertion Based on Greedy Strategy with Time Loss Is Priority (GTLP)

**Input:**
- \( t \): current time;
- \( l_c \): current location of the vehicle;
- \( o' \): nearby candidate origin bus stop to \( req(o,d,w) \);
- \( d' \): nearby candidate destination bus stop corresponding to \( o' \);
- \( OS' \): set of adjacent candidate origin stops to with requests;
- \( DS' \): set of adjacent candidate destination stops corresponding to \( OS' \);
- \( r(u,v,t) \): flex route; \( path(r) \): current path of \( r \);
- \( BS \): backbone stops at \( path(r) \); \( SA \): stops included in the service area.

**Output:** \( path(r) \): path after handling \( req \)

1. **if** \( o', d' \) are in \( path(r) \), \( o' \in OS', d' \in DS' \) **then**
   2. \( BS = BS \cup \{o', d'\} \);
   3. return \( path(r) \);
4. **else if** either \( o' \) or \( d' \) \( \notin \) \( SA \) **then**
   5. reject \( req(o,d,w) \);
6. **else**
   7. \( A_i \leftarrow \) subarea that contains \( o' \);
   8. \( A_j \leftarrow \) subarea that contains \( d' \);
   9. \( I_i \leftarrow \) insert positions between the penultimate stop and the last stop of \( A_i \);
   10. \( I_j \leftarrow \) insert positions between the penultimate stop and the last stop of \( A_j \);
   11. insert \( o' \) into \( I_i \), insert \( d' \) into \( I_j \);
12. **if** \( cap(k) > CAP, k \in path(o',d') \) **then**
   13. reject \( req(o,d,w) \);
14. **else**
   15. calculate cost and feasibility of inserting \( o' \) and \( d' \);
   16. \( STOP = \{o'|cost(path(r)_{o',d'}) < rt(r), o' \in OS', d' \in DS'\} \);
if $STOP = \phi$ then

$lc \leftarrow sp(lc, bs_i)_1$, the number of stop in $sp(lc, bs_i)$ is $m(0,1,...,m-1)$;

return $path(r)$;

else

$(o', d') \leftarrow \arg\min_{(o', d')} \{cost(path(r), o', d') : cost(path(r), o', d') < rt(r)\};$

accept req($o, d, w$);

lc $\leftarrow o'$;

path($r$) = path($r$) − {lc $\rightarrow bs_i$};

path($r$) = path($r$) + {lc $\rightarrow o'$} + {o' $\rightarrow bs_i$};

path($r$) = path($r$) − $A_{penu}^j$ $\rightarrow bs_j$;

path($r$) = path($r$) + $A_{penu}^j$ $\rightarrow d'$ + {d' $\rightarrow bs_j$};

$A_{penu}^j$ $\leftarrow d'$;

update $cap(k)$, $k \in path(o', d')$;

$BS = BS \cup \{o', d'\}$;

return $path(r)$;
Algorithm 2: Origin-Destination Insertion Based on Greedy Strategy
with The Number of Passengers Is Priority (GNPP)

1 else
2 \( (o', d') \leftarrow \arg \max_{(o', d')} \{ \text{req}_{\text{num}}(o', d') : \text{cost}(\text{path}(r)_{o', d'}) < \text{rt}(r) \} \);
3 accept \( \text{req}(o, d, w) \);
4 \( lc \leftarrow o' \);
5 \( \text{path}(r) = \text{path}(r) - \{ lc \rightarrow bs_i \} \);
6 \( \text{path}(r) = \text{path}(r) + \{ lc \rightarrow o' \} + \{ o' \rightarrow bs_i \} \);
7 \( \text{path}(r) = \text{path}(r) - \{ A^l_{\text{penu}} \rightarrow bs_j \} \);
8 \( \text{path}(r) = \text{path}(r) + \{ A^l_{\text{penu}} \rightarrow d' \} + \{ d' \rightarrow bs_j \} \);
9 \( A^l_{\text{penu}} \leftarrow d' \);
10 update \( \text{cap}(k), k \in \text{path}(o', d') \);
11 \( BS = BS \cup \{ o', d' \} \);
12 return \( \text{path}(r) \);

where \( \text{cost}(\text{path}(r)_x) \) is the cost of travelling along path \( \text{path}(r)_x \), \( \text{rt}(r) \) is the running time of route \( r \). The path with insertion \( I_{k_1} \) has the least travelling time. The path is selected and should also be feasible, i.e. meets the running time constraint.

If there is a feasible path after insertion, the request would be accepted; otherwise, the request would be rejected. When \( o' \) is inserted at \( I_{k_1} = (bs_{j+k_1}, bs_{j+k_1+1}) \), stop \( o' \) is added to \( BS \) and the path is updated by following operations:

\[
\begin{align*}
\text{path}(r, o') &= \text{path}(r) - \{ bs_{j+k_1} \rightarrow bs_{j+k_1+1} \} \\
\text{path}(r, o') &= \text{path}(r) + \{ bs_{j+k_1} \rightarrow o' \} + \{ o' \rightarrow bs_{j+k_1+1} \}
\end{align*}
\]

where \( \{ a \rightarrow b \} \) is the shortest path from \( a \) to \( b \). In Fig. 9, \( (a, e) \) is the insertion position, new paths are built by adding shortest paths \( a \rightarrow c \) and \( c \rightarrow e \), and removing \( a \rightarrow e \).

(3) When both stop \( o' \) or stop \( d' \) are not in \( BS \), two insertion positions are
Algorithm 3: Traditional Vehicle Travel Mode (TVTM)

Input:
- $path(r)$: fixed route traveled by vehicle;
- $o'$: nearby origin bus stop to $req(o,d,w)$;
- $d'$: nearby destination bus stop corresponding to $o'$;
- $BS$: backbone stops at $path(r)$.

Output: $req_{num}$: the total number of requests received by the vehicle.

1. if $o', d'$ are in $path(r)$, and $d'$ behind $o'$ then
2.   if $cap(k) > CAP$, $k \in path(o', d')$ then
3.     reject $req(o,d,w)$;
4.   else
5.     accept $req(o,d,w)$;
6. else
7.   reject $req(o,d,w)$;
8. return $req_{num}$
identified and the feasibility of a new path after insertions is checked. The
insert procedure is similar to case (2). Suppose the subarea that contains o′
is $A_i$, and the set of backbone stops in $\text{path}(r)$ contained in $A_i$ is $\text{BS}(A_i) = \{bs_{j+1}, bs_{j+2}, ... bs_{j+m}\}$. The possible insert position of $o'$ is defined as $I_0$, $I_1$, $I_2$, ..., $I_m$, where $I_k = (bs_{j+k}, bs_{j+k+1})$, $k = 0, 1, ..., m$, and $bs_j$ is the last backbone stop at subarea $A_{i-1}$. Similarly, we define the subarea that contains $d'$ as $A_{i'}$, and the possible insert positions of $d'$ is defined as $I'_0$, $I'_1$, $I'_2$, ..., $I'_n$, where $I'_k = (bs_{j'+k}, bs_{j'+k+1})$, $k = 0, 1, ..., n$. $bs_{j'}$ is the last backbone stop at subarea $A_{i'-1}$. Then the insert positions for $o'$ and $d'$ are calculated by the following formula:

$$(k_1, k_2) = \arg\min_{(x,y)} \{\text{cost}(\text{path}(r)_{x,y}) : \text{cost}(\text{path}(r)_{x,y}) < rt(r), \}

\begin{align*}
&x = 0, ..., m, \quad y = 0, ..., n
\end{align*}

(43)$$

where $\text{path}(r)_{x,y}$ denotes the path of $r$ if inserting $o'$ at $I_x$ and $d'$ at $I_y$.

Only when both insertions are allowed, the request is accepted; otherwise, the request is rejected. The stops would be added to set $\text{BS}$ if the request is accepted.

4.4.3. Algorithm Description (Origin-Destination Insertion)

Algorithm 1-2 presents the pseudocode of the origin-destination insertion algorithm based on greedy strategy.

Algorithm 1 is a greedy algorithm with time loss is given priority(GTLP). In GTLP, we give priority to the insertion of O-D pairs with the least time loss. When a PV is travelling, if the subsequent adjacent request-origin point and its corresponding request-destination point are both included in the backbone set(line - line ), they will be directly inserted into the driving route(line - line ). Otherwise, the algorithm will judge whether these two points are within the scope of service area(line - line ), if one of them does not included in the service area(line - line ), the PV will refuse to accept the request(line - line ). Before inserting the origin-destination pair into the route, GTLP will calcu-
late whether the number of passengers on the PV from request-origin point to request-destination point will exceed the maximum PV capacity (line - line ), it will also refuse to accept the request if the insertion of the Origin-Destination pair does not satisfy the PV’s capacity constraint (line - line ). Finally the algorithm will calculate the time loss caused by the insertion of Origin-Destination pairs satisfying the capacity constraints in turn (line - line ), and select the Origin-Destination pair which with the minimum time loss to insert into the route (line - line ), meanwhile, the selected request-origin point and its corresponding request-destination point will be added to the backbone set (line - line ). If all adjacent stations both do not meet the insertion conditions, the shortest path from the current position to the next backbone will be selected as the driving direction (line - line ).

In algorithm 2, the number of passengers is given priority (GNPP). After all insertion conditions are met (line - line ), in order to receive the most requests at present without breaking the bus capacity constraint, PV will select the Origin-Destination pair with the largest number of passengers to insert into the route (line - line ), and if all adjacent stations both do not meet the insertion conditions, the shortest path from the current position to the next backbone also will be selected as the driving direction (line - line ).

4.4.4. Algorithm Description (Scheduling)

In this paper, we propose scheduling is modelled as a Mixed-Integer Linear Programming (MILP) problem in chapter 4.2 and use CPLEX to solve this MILP problem. CPLEX Optimizer provides flexible, high-performance mathematical programming solvers for linear programming, mixed integer programming, quadratic programming and quadratically constrained programming problems.
Figure 5: An example of an insertion into the path. $\langle a, e \rangle, \langle e, f \rangle, \langle f, g \rangle$ are the possible insertion positions for stop $c$. Yet $\langle a, e \rangle$ is the insertion position, new paths are built by adding shortest paths $a \rightarrow c$ and $c \rightarrow e$, and removing $a \rightarrow e$.

5. Performance Evaluation

5.1. Data pre-processing

In this paper, we use the map of Xiamen as experimental environment to verify the performance of scheduling algorithms and Origin-Destination insertion algorithms respectively. We compare the results of the operation indicators in scheduling by two schemes that solving the MILP problem of scheduling (MILPS) and the fixed time interval scheduling (FTIS) with a largescale real-world taxi GPS data set, which is generated from taxis in Xiamen on July 1, 2014 from 7:00am to 9:00am. Meanwhile, we use the virtual request data to verify the performance on two heuristic Origin-Destination insertion algorithms and traditional fixed routes in vehicle travelling.

We first divide sub-service area between adjacent backbone stops in an elliptical shape according to a predetermined size, and then select the O-D pairs of the origin and destination points are both within sub-service area as experimental data from all O-D data in a given time slot.

5.2. Environmental Setup

The schemes are implemented in Java 1.8 and experiments are run on a desktop server with Intel Core i7 (4 cores), 16G DDR3 RAM.
5.3. Comparison Algorithms

5.3.1. Scheduling

To show the advantages of MILP model of scheduling (MILPS) and the performance of the CPLEX Optimizer, we also designed a scheme which more suitable for real-time scheduling named-Fixed Time Interval Scheduling (FTIS): give K which means the number of vehicle seats, N which means total number of vehicles, and the departure time period. The algorithm will assign the same time interval to these N vehicles on the up and down routes. The vehicle will follow TVTM for travelling, and finally calculate fsr (Flow Satisfaction Rate, we will mention it later) and objective function value in MILPS.

5.3.2. Origin-Destination insertion Algorithms

Besides two heuristic origin-destination insertion algorithms GTLP and GNPP, another algorithm is implemented for the comparison purposes:

Traditional Vehicle Travel Mode (TVTM): the vehicle travel follows a fixed route, which has more initial backbone stations. As shown in the pseudocode of algorithm 3, when a vehicle passes through these stations, it will only accept the request that destination points belong to the remaining non travelled sections (line - line ), and the number of passengers on the vehicle will not exceed the maximum passenger capacity of the vehicle (line - line ).

5.4. Evaluation of the experiment

5.5. Evaluation on Solution of MILP Question of Schedule

As illustrated in MILPS, we study how \( K \) and \( N \) influence the model’s objective when \( F_{ij}^u \) and \( F_{ij}^v \) are given as constant variables calculated from historical data for the model. We measure the gap between the demand and supply of bus seats as follows:

\[
fsr = \frac{\sum_{i=0}^{n-1} \sum_{j=1}^{n} K \ast (x_{ij}^u + x_{ij}^v) }{\sum_{i=0}^{n-1} \sum_{j=1}^{n} F_{ij}^u + F_{ij}^v }, \ j > i
\]  

The \( fsr \) (Flow Satisfaction Rate) shows the sum of requests received by all scheduled vehicles. To better understand the influence of \( K \) and \( N \) on flow
Figure 6: Impact of the number of PV’s seats $K$ and the number of vehicles $N$ on the $fsr$ value (left) and the objective function value (right) by taking FTIS (cold colored surface) and MILPS (rainbow surface).

Figure 7: O-D pairs insertion in RFI (left) and DFI (right) when detour ratio $\alpha$ set as 0.6 and PV capacity set as 50.

satisfaction rate, we conduct experiments under different parameter settings to study how they affect the $fsr$ result.

The results are shown in Fig.6. The left figure shows that, with the change of $K$ and $N$, the trend of $fsr$ value by taking FTIS and MILPS. The rainbow surface represents the result of $fsr$ by taking MILPS and the cold colored surface represents FTIS. We can see that the value of $fsr$ gradually increase with the increase of $K$ and $N$, meaning that, when the vehicles has more seats and the station can accommodate more vehicles, the closer the $fsr$ is to 1. Also known as more seats and vehicles provide more possibilities for receiving requests. It can be clearly seen from the 3-D graph that there has a large gap between two surfaces, actually compared with the FTIS, the MILPS makes the $fsr$ value increase by %.

The right figure shows that, with the change of $K$ and $N$, the trend of objective function value in the MILP model of scheduling by taking the schemes of FTIS and MILPS. Similarly, the rainbow surface represents MILPS and the cool surface represents FTIS. We can see that the value of objective function gradually decrease with the increase of $K$, which means when the bus has more
Figure 8: Impact of detour ratio alpha on (a) the number of O-D pairs, (b) the number of requests received, (c) travel time, (d) average travel time.

Figure 9: Impact of PV capacity on (a) the number of O-D pairs, (b) the number of requests received, (c) travel time, (d) average travel time.

seats, the smaller the gap between the demand and supply is. But we also can see, with the $N$ increase, the value of objective function shows a trend of decreasing first and then increasing slowly. It also can be clearly seen that there has a large gap between MILPS surface and FTIS surface, actually compared with the FTIS scheme, the MILPS scheme makes the objective function value decrease by $\%$.

5.6. Evaluation on GTLP

More detour time and more seats make a PV has more relaxed restrictions when picking up passengers. It can be seen that with the increase of passenger capacity parameter and detour ratio parameter, the number of inserted O-D pairs and the total number of requests received will also increase.

Fig.7 shows the location of the inserted O-D pair in the map. When an O-point is inserted in the former segment sub service area, it will also insert a D-point in the latter one, thus when a PV travels in the latter segment sub service area, the O-point to be inserted will face more stringent constraints of time and bus capacity. We can see that since two heuristic Origin-Destination
insertion algorithms adopts greedy strategy, the number of inserted O-D pairs shows a distribution law of more before and less after. Fig.8 shows the metrics with different combinations of detour ratio and PV’s passenger capacity. We can see that the total number of passengers received by using two heuristic Origin-Destination insertion algorithms is twice as much as that of TVTM and the average travel time of passengers decreased by 30 % to 50 %. When the detour ratio below 0.6, GTLP has a higher pick-up efficiency than GNPP, but the total travel time of PV in GTLP will be higher than in GNPP. Meanwhile, because GTLP has higher pick-up efficiency, the average ride time of passengers in GTLP is less than GNPP.

6. Conclusions

References


