Zhou, Ze; Roncoli, Claudio

A scalable vehicle assignment and routing strategy for real-time on-demand ridesharing considering endogenous congestion

Published in:
Transportation Research Part C: Emerging Technologies

DOI:
10.1016/j.trc.2022.103658

Published: 01/06/2022

Document Version
Publisher's PDF, also known as Version of record

Published under the following license:
CC BY

Please cite the original version:
A scalable vehicle assignment and routing strategy for real-time on-demand ridesharing considering endogenous congestion

Ze Zhou, Claudio Roncoli
Department of Built Environment, Aalto University, Espoo 02150, Finland

ARTICLE INFO

Keywords:
Traffic control
Dynamic ridesharing
Network traffic
Vehicle assignment and routing

ABSTRACT

On-demand ridesharing has been recognised as an effective way to meet travel needs while significantly reducing the number of required vehicles. As the growth of on-demand mobility services and the advent of shared autonomous vehicles are envisioned to boost the presence of ridesharing vehicles, these may soon significantly affect traffic patterns in our cities. However, most previous studies investigating dynamic ridesharing systems overlook the effects on travel times due to the assignment of requests to vehicles and their routes. In order to assign the ridesharing vehicles while considering network traffic dynamics, we propose a strategy that incorporates time-dependent link travel time predictions into the request–vehicle assignment to avoid or mitigate traffic congestion. In particular, we formulate an efficient linear assignment problem that considers multiple path alternatives and accounts for the impact on travel times, which may be potentially caused by vehicles assigned to specific routes. Simulation experiments reveal that using an appropriate congestion avoidance ridesharing strategy can remarkably reduce passenger average travel and waiting times by alleviating endogenous congestion caused by ridesharing fleets.

1. Introduction

In densely populated areas, urbanisation, followed by the increasing need for mobility, has been constantly putting under serious stress the existing transportation systems, with consequent deteriorated performance in terms of efficiency and safety. Meanwhile, the situation is further aggravated by the growing number of private cars and the consequent low car occupancy rate. In fact, statistics show that only 1.7 passengers are, on average, on board of private cars in the US, with an even lower number of passengers for work-based trips (Santos et al., 2011). As a mobility supplement of public transit and private cars, on-demand ridesharing services are gaining popularity thanks to their flexibility and potentially more efficient operations and lower costs. In particular, by grouping trips that share similar itineraries and time schedules, ridesharing is potentially an effective and flexible way to alleviate congestion by serving the heavy demand with a reduced number of vehicles (Agatz et al., 2012). For instance, simulation experiments by Lokhandwala and Cai (2018) show that ridesharing combined with automated vehicles can decrease the traditional taxi fleet size by 59%, without significantly increasing passengers' waiting time. With these premises, such an approach opens up possibilities to devise more effective and sustainable future mobility systems.

Ridesharing is envisioned as an indispensable component of ridesourcing services1 because of its potential to reduce fleet size and improve efficiency. Emergent ridesourcing companies, such as Uber, Lyft, and Didi, lead to a ubiquitous phenomenon where

* Corresponding author.

E-mail addresses: ze.zhou@aalto.fi (Z. Zhou), claudio.roncoli@aalto.fi (C. Roncoli).

1 A ridesourcing service is defined as an internet- or smartphone app-based service that aims at connecting passengers with committed drivers. Drivers may operate their own vehicles, but are subject to some degree of coordination by the service provider.

https://doi.org/10.1016/j.trc.2022.103658
Received 6 August 2021; Received in revised form 19 March 2022; Accepted 19 March 2022
Available online 21 April 2022
0968-090X/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).
drivers throughout streets are providing on-demand services (Beojone and Geroliminis, 2021) and the popularity of such services is envisioned to both reshape and disrupt urban transportation (Li et al., 2021).

However, the implications of ridesharing and ridesourcing services on traffic congestion remain a hottest and controversial research topic (Ke et al., 2020). As ridesharing and ridesourcing are expected to become in the near future a mainstream choice in mobility, this may significantly affect traffic patterns in our cities, with the potential of becoming a source of traffic congestion itself (Correa et al., 2019). On the other hand, as they may mitigate congestion, by increasing car occupancy and operating vehicles in a more efficient manner, it is essential to investigate how ridesourcing companies dispatch tens of thousands of vehicles considering their effects on traffic dynamics. A recent study (Tirachini et al., 2020) investigated the impact of shared rides offered by a ridesourcing company on motorised traffic, concluding that shared riders in vehicles with larger capacity, such as vans, can reduce the vehicle kilometres travelled, which makes it a promising direction for future shared mobility.

Another potential future mobility scenario consists in combining ridesharing with shared autonomous vehicles, i.e., autonomous vehicles operating a short-term carsharing service (Fagnant and Kockelman, 2014). Compared to a traditional ride-hailing service, half of the SAV fleet with a ridesharing service may be enough to satisfy the same demand (Farhan and Chen, 2018). Previous studies (Martinez and Crist, 2015; Dia and Javanshour, 2017; Fagnant and Kockelman, 2018) also show that one SAV can replace between 1.18 and 11 traditional private cars, which suggests the potential of SAVs in reducing car ownership (Narayan et al., 2020). The convenience and low cost of SAVs do not only make them competitive with the taxi industry, but have also great potential to replace private vehicles in large scale. For instance, Nieuwenhuijsen et al. (2018) claim that, in a not so distant future, the majority of people’s daily trips would be served by SAVs rather than by traditional private vehicles. On the other hand, although a fleet of SAVs could serve the same demand with fewer vehicles, the total travel distance could increase due to vehicles travelling empty (Spieser et al., 2014). This side effect may be more evident at the perimeter of cities (Kumakoshi et al., 2021). Levin (2017) also concludes that SAVs may significantly contribute to the network congestion when tens of thousands of SAVs would have replaced the personal vehicles. Automated driving technology matches perfectly with ridesharing and carsharing services for two reasons. Firstly, automated vehicles have potential to provide more flexible and reliable services when dealing with frequently changed real-time travel schedule (Zhang et al., 2015). Secondly, automated driving enables a more efficient and precise relocation of vehicles, without incurring additional labour costs. In conclusion, the large-scale deployment of SAVs may catalyse a wide adoption of ridesharing.

In the aforementioned contexts, it is reasonable to expect that an increasing number of ridesharing vehicles may significantly affect the traffic patterns, with a risk of negative effects, such as congestion. Although existing research on emerging mobility on-demand, considering ridesharing services and SAVs, on congestion, they normally assume these services are provided in a static and passive way without actively optimising their operation (assignment and routing) to actively mitigate or avoid congestion. While only few works investigate how congestion-aware operation of those service can positively influence traffic.

In this work, we investigate real-time dynamic ridesharing assignment and routing problem, considering the endogenous effects of ridesharing on traffic patterns. More specifically, we assume that trip requests are served by a vehicle fleet of a given size, which is composed of multiple co-existing fleet operators. The requests are handled by a centralised management platform that is responsible for efficiently assigning requests and for maintaining a low congestion level in the network. Despite we do not directly specify the fleet composition, i.e., if they are human driven vehicles or AVs, these can be accounted by setting different traffic characteristics (e.g., road capacities, backward wave speeds, and vehicle maximum capacities) in our network model; see, e.g., Wei et al. (2017), Liu et al. (2020). Based on these premises, we design a management framework and propose an optimisation-based methodology for dynamic ridesharing services considering their impacts on traffic dynamics. The framework is based on the federated structure proposed by Simonetto et al. (2019), which is extended to incorporate the capability of congestion prediction. Besides, benefitting from the distribution of computation and the efficient computational time, a novel optimisation problem is formulated to simultaneously optimise the trip cost and reduce congestion at the path planning stage. To the best of our knowledge, the main contributions of this study are summarised as follows:

- We incorporate a DNL algorithm within a dynamic ridesharing management system to assess and predict the time-space varying travel time, which is affected by the ridesharing fleet operation.
- With the knowledge of predicted traffic dynamics, we present a novel time-dependent k-shortest path-based model to jointly optimise the ridesharing vehicle assignment and routes.
- We propose a decentralised management framework for the large-scale ridesharing system with dynamic travel time, which allows us to cope even with high traffic demand at a lower computational complexity, capable of operating in real-time.

An early version of this work is included in Zhou and Roncoli (2021), which is extended here in terms of formulation and numerical experiments. The remainder of this paper is structured as follows. Section 2 reviews the literature regarding large-scale dynamic ridesharing and vehicle assignment considering dynamic travel time. Section 3 explains the overall research problem and defines two types of predicted congestion. Section 4 elaborates on the proposed decentralised framework and its main components for the dynamic ridesharing systems. In Section 5, two sets of simulations are implemented to analyse the sensitivity of critical parameters and evaluate the performance of the proposed strategies. Finally, we summarise the main findings and provide some suggestions for future research in Section 6.

Note that, in this paper, we utilise the term ridesharing for a service where two or more travellers with similar routes share a ride by utilising some service platform, e.g., to reduce the trip fee; while we utilise the term carsharing for a short period car rental service.
2. Literature review

2.1. Large-scale dynamic ridesharing

Efficient operation of ridesharing fleets in large-scale networks poses a set of challenges, due to the large scale nature of the problem, as well as the inherent complex dynamics to be taken into account, which lead to large scale nonlinear problems to be solved, calling for efficient algorithms. For example, in large cities, we can envision scenarios where hundreds of thousands of on-demand vehicles need to be coordinated and matched within hundreds of zones, which requires efficient approaches in order to produce high-quality schedules (Agatz et al., 2012). We focus here on dynamic ridesharing, which, differently from static ridesharing allows that the schedule can be altered after assignment (Shen et al., 2016). From a mathematical viewpoint, dynamic ridesharing assignment is essentially a combination of the vehicle routing problem with the dynamic pickup and delivery problem with time windows (Mahmoudi and Zhou, 2016), which are both known as complex problems to be solved.

To deal with this problem, a large-scale taxi sharing framework was developed by Santi et al. (2014), where an innovative shareability network method is proposed to handle the taxi trip sharing problem. However, such method has a limit of two rides per-trip, while trips are immutable once started. Alonso-Mora et al. (2017) extended the work by Santi et al. (2014) by scaling the problem size and introducing a 4-steps algorithm to cope with high-capacity ridesharing issues; this is achieved by generating a request–trip-vehicle graph considering various time and capacity constraints and then formulating an integer linear programming problem to calculate the optimal assignment plan. Their results are even more optimistic than those by Santi et al. (2014), showing that 3000 vehicles can cover 98% of the total taxi trips in Manhattan. Simonetto et al. (2019) developed further the method proposed by Alonso-Mora et al. (2017) by restricting that at most one passenger can be assigned to a vehicle at each optimisation epoch. This simple but powerful design not only reduce the dynamic ridesharing problem into a linear assignment problem, but also remarkably decrease the number of feasible passenger–vehicle assignments. Consequently, the simulation results by Simonetto et al. (2019) achieve a similar service quality as the ones by Alonso-Mora et al. (2017) with a considerably lower computational burden. Moreover, the computation can be partly distributed among different platforms, making it more applicable when dealing with multi-company competition.

2.2. Congestion-aware vehicle assignment

There exist a large body of scientific literature on taxi dispatching and shared automated taxis operation. For instance, Ma et al. (2017) explored reservation-based SAV assignment, while not considering ridesharing and congestion. Hyland and Mahmassani (2018) presented six strategies for real-time SAVs fleet operation problem with no shared rides. In order to simplify the travel time prediction among different origins and destinations, all the aforementioned studies, including those mentioned in Section 2.1, consider travel times that are either static or that are estimated utilising historical data. However, in reality, link travel times are dynamic and depend on the traffic conditions in each link (e.g., flows and densities). Besides, dynamic ridesharing service are susceptible to changes of travel times and schedules, whereas congestion only worsens the situation as it increases the unreliability of the system. Fielbaum and Alonso-Mora (2020) described two sources of unreliability in a ridesharing system: first, sudden changes because of new requests, and, second, the previously assigned requests need to be rescheduled due to operational rules. They found that more than a third of requests experienced some change using a real dataset from Manhattan. This will be even more evident without human drivers, since SAV-based systems may require more precise traffic information to improve their on-time arrival reliability (Liu et al., 2019). Thus, the methods and algorithms utilised for assigning vehicles to requests, as well as for routing, should be defined so that they account for the system-level effect on, e.g., travel times and overall traffic patterns to increase the reliability of ridesharing services. Some recent studies started to relax the unrealistic assumptions of constant travel times to incorporate more sophisticated travel time prediction methods.

Various traffic flow models may be employed to capture the traffic dynamics and update link travel times. A widely used static model is a link performance function called Bureau of Public Roads (BPR). Salazar et al. (2019) proposed a congestion-aware routing scheme with the purpose of actively rebalance and route vehicles towards system optimum that utilises a piecewise approximation of the BPR function to reduce the computational complexity. Nevertheless, the computational time is in the order of minutes when dealing with high demand in a large urban network. Solovey et al. (2019) reduced this congestion-aware on-demand vehicle routing problem to the traffic assignment problem and the classic Frank–Wolfe algorithm was extended solve the problem efficiently. However, both approaches leave aside the incorporation of ridesharing as well as vehicle assignment. Correa et al. (2019) employ BPR functions to evaluate the travel delay on road segments, proposing a method that can handle large scale networks due to the introduction of intermediate meeting point and pre-processing of shortest path. However, vehicle assignment and traffic prediction across different time steps are not considered in their work. In a first attempt to incorporate congestion effects in (SAVs) ridesharing, Liang et al. (2020) formulated an optimisation problem where congestion is modelled via BPR functions. Nevertheless, due to the high complexity of the resulting nonlinear optimisation problem, the method is computationally very expensive. Finally, note that all BPR-based methods are unable to depict more sophisticated congestion patterns, such as spillback, traffic waves, queues at intersection, etc. (Boyles et al., 2021).

Employing dynamic models, traffic can be modelled at a finer resolution, which enables more precise and realistic predictions. With this purpose, Levin et al. (2017) developed an event-based framework to simulate SAVs by a dynamic network loading (DNL) model. The results indicate that the empty repositioning trips by SAVs cause more congestion than conventional vehicles. However, only simple ridesharing rules are incorporated into their framework and no optimisation is implemented to minimise travel costs.
Furthermore, Levin (2017) investigated the SAVs routing problem by developing a linear program based on the link transmission model (Yorman et al., 2005). A similar work has been performed by Venkatraman and Levin (2021), where, instead of the link transmission model, the cell transmission model is employed to update the traffic states. Although the work by Venkatraman and Levin (2021) can be extended to include ridesharing, the authors claim that such extension would further complicate the model formulation, making it hard to be solved for large-scale problems. Moreover, Li et al. (2021a) developed a mixed integer linear programming problem to jointly optimise the SAVs fleet size, parking lot design, and daily system operation, where the link transmission model is incorporated to determine travellers’ departure time and vehicle routes. Nonetheless, ridesharing and active congestion-avoidance routing are excluded. Similarly, Seo and Asakura (2021) proposed a linear programming problem to optimise fleet size, road network design, and parking lot deployment of a SAV system, where traffic congestion is considered via a point-queue dynamic traffic assignment model. However, vehicle routing and static ridesharing are considered at macroscopic level, while individual-level routing and assignment for individual travellers and vehicles cannot be identified.

Two existing works are the closest to the problem addressed in this paper. Liu et al. (2020) developed a framework to integrate vehicle assignment and routing considering both endogenous road congestion and ridesharing. To address congestion, the approach proposed by Liu et al. (2020) is to restrict the number of vehicles on a road to remain below capacity; moreover, the method is not designed for real-time operations. Note that, differently, our proposed method involves assigning vehicles proportionally to the relative remaining capacity on a road while considering multiple path alternatives. Recently, Alisoltani et al. (2021) proposed a framework to incorporate the evolution of traffic dynamics when providing a ridesharing service, where a trip-based macroscopic fundamental diagram is employed as prediction model to estimate the actual speed at the beginning of each time step. The author proposed heuristic algorithms to reduce the computation time without significant loss of solution quality. Furthermore, the framework and algorithm are tested in a large-scale city in France. Results show that the higher the trip density, the better the performance in term of congestion mitigation with the ridesharing service. Our method differs from the one by Alisoltani et al. (2021) mainly in three aspects: (I) instead of using a spatially aggregated model (macroscopic fundamental diagram), we employ an LWR-based DNL method to predict the evolution of traffic, which enables a finer time-space granularity, i.e., we model at link level and for multiple time steps, while their model is at region level and looks at only one time step in the future; (II) instead of predefined fixed shortest paths, we consider time-dependent $k$-shortest paths; and (III) instead of only considering assignment to minimise total travel time of passengers and vehicles, we combine the vehicle assignment with route choices for the purpose of avoiding and mitigating congestion.

3. Preliminaries

Given a time-dependent directed graph $G = (N, E, C)$, where $N$ is a set of nodes; $E \subseteq N \times N$ is a set of edges; $C$ is a set of piecewise non-negative functions. For every edge $e \in E$, there is a function $c_e(t) \in C$, which specifies the travel time along edge $e$ if departing the upstream end $e^+$ at time $t$. Let $a(e^+)\) denote the arrival time at the link source $e^+$, given a path $p = (e_1, e_2, \ldots, e_k)$, the time-dependent path travel time $TD_p(t)$ is calculated recursively from the following equations:

$$a(e_1^+) = t$$

$$a(e_2^+) = t + c_{e_1}(t)$$

$$a(e_2^+) = a(e_1^+)$$

$$a(e_2^+) = a(e_1^+) + c_{e_2}(a(e_1^+))$$

$$\ldots$$

$$TD_p(t) = a(e_k^+) = a(e_1^+) + c_{e_2}(a(e_1^+)).$$

A request $r$ is defined as a tuple $r = \{t^e, t^p, n^e, n^p\}$, which contains request time $t^e$ (assumed equal to the required earliest pickup time), latest pickup time $t^p$, origin $n^e \in N$, and destination $n^p \in N$. The maximum waiting time is then introduced as $\Delta_w = t^p - t^e$.

We assume that all the requests are willing to share rides with others and are served by a fleet of ridesharing vehicles. A vehicle $v$ is defined as a tuple $v = \{l, p, R, S\}$, where $l$ represents the current vehicle location, $p$ is the currently designated vehicle path, $R$ denotes a collection of pre-dispatched requests, and $S = (n^1, n^2, \ldots, n^k, n^\prime_1, n^\prime_2, \ldots)$ consists of a series of sequential scheduled passengers’ pick-up and drop-off nodes. Here, we assume that all passengers’ requests are characterised by maximum acceptable waiting and detour times and we disregard any new assignment to the vehicle assigned to requests that is predicted to violate any of these time constraints, i.e., no new assignment is allowed since a passenger on board, or planned to board, is already late. It is noteworthy mentioning that new batches and assignments may generate additional traffic that alters the previous predictions, which may happen to cause additional delay for passengers already on board or newly assigned to a vehicle. In mathematical terms, a request $r$ can be served by vehicle $v$ only if there exists a new schedule $S^\prime_v$ that satisfies:

$$TD_{\{r, n^1, \ldots, n^\prime_n\}}(t) \leq t^p - t, \quad \forall r \in R_v \cup r$$

$$t + TD_{\{r, n^1, \ldots, n^\prime_n\}}(t) \leq t^p + TD_{\{r, n^1, \ldots\}}(t^p) + \Delta_w + \Omega_r, \quad \forall r \in R_v \cup r$$

where $TD_{\{n_1, n_2, \ldots, n_j\}}(t)$ represents the shortest time-dependent travel time when a vehicle sequentially visits nodes $n_1, n_2, \ldots, n_j$ departing at time $t$; and $\Omega_r$ represents the maximum detour time. Inequality (2a) ensures that each dispatched requests $r$ can be picked up before $t^p$, whereas constraint (2b) guarantees that new insertions do not violate each request latest arrival time. Note
that this is checked only at assignment phase, whereas, if such time windows are violated because of congestion (that was not predicted at the time of assignment), we assume that the pre-assigned passengers would simply continue their trip until they reach their destination.

Let us consider a real-time vehicle dispatch centre where unknown ridesharing requests may arrive randomly. The service aims at dispatching vehicles to requests and generating vehicle routes dynamically to avoid or mitigate the effects of congestion. To deal with ridesharing requests arriving in real-time, an efficient matching strategy called request batching is adopted. Instead of simply assigning a request to its nearest available vehicle or the one with the shortest pick-up time, we gather and process a batch of incoming requests received during a given time interval (e.g., in the order of 10 s – 60 s), which is proved to be beneficial in term of computational efficiency without deteriorating performance (Ashlagi et al., 2018). Furthermore, a vehicle is entitled to serve only one request within each batch; however, note that this limitation does not prevent vehicles from serving multiple travellers from different batches. Conversely, it facilitates the optimisation of dynamic ridesharing by reducing the problem into a linear assignment problem (Simonetto et al., 2019). Note that, although the one-to-one matching mechanism may seem inefficient as it neglects the possibility of combining requests belonging to a single batch, the simulation results by Simonetto et al. (2019) demonstrated that such strategy can achieve a better computational efficiency (up to 4 time faster) than in the work by Alonso-Mora et al. (2017), while delivering a similar service quality. Eventually, we need to solve this decomposed problem at each decision epoch after request batching.

### 3.1. Problem decomposition

We are interested in assigning efficiently in real-time ridesharing requests to vehicles and to generate time-dependent paths considering network congestion, which is endogenously caused by the movement of these vehicles. As mentioned earlier, dynamic ridesharing alone is a NP-hard problem. Hence it is difficult to integrate it with, e.g., traffic prediction techniques such as link transmission model (Levin, 2017), or fleet operation optimisation methods (Li et al., 2020). Since formulating this as a unique optimisation problem would be heavy and computationally demanding, we decompose it into the following two subproblems:

(a) how to predict the time-dependent link travel time given currently known vehicles’ schedule;
(b) how to dynamically dispatch and route vehicles considering spatio-temporally varying traffic conditions.

These two questions are closely related to each other and, in fact, solving problem (a) is a necessary condition to solve problem (b); in addition, the assignment results and corresponding routes obtained from (b) are affecting the future predictions of (a). By solving these two problems sequentially and repeatedly, the system is capable of continuously determining which vehicle to serve which request along which road considering dynamic traffic in a rolling horizon way. Note that a similar decomposition has been also recently proposed by Alisoltani et al. (2021).

#### 3.1.1. Subproblem (a): Real-time traffic prediction

Research considering static traffic settings assign vehicles to requests while considering a constant travel time for each link of a network, which is typically the free-flow (uncongested) travel time. Despite Simonetto et al. (2019) suggests that congestion effects may be incorporated by considering some speed-related factor obtained from historical data, such approaches fail to describe and react to real-time traffic. In addition, the presence of non-recurrent congestion, which is typical in traffic networks, makes historical data-based approaches inaccurate.

To address these issues, we aim here at considering a traffic prediction problem that, given the known trips or vehicle paths (or a prediction of them) and departure times, delivers the predicted link travel time $c_t(t), \forall e \in E$. Specifically, the subproblem (a) can be solved by any dynamic network loading method, which is the process of modelling traffic evolution in a network, given the route and departure time of every vehicle (Boyles et al., 2021). A solution to this subproblem is presented in Section 4.2.

#### 3.1.2. Subproblem (b): Vehicle assignment considering two types congestion

Assuming a method for calculating predictions for traffic condition is available, we proceed to investigate how to efficiently assign and route vehicles to avoid or mitigate congestion. A concrete example is shown in Fig. 1 to illustrate how this system operates and the two types of congestion are defined in the following discussion. The three sub-figures in Fig. 1 represent current and predicted traffic conditions in three subsequent time steps. At time $t$, given all the predefined vehicle paths, shown as dashed lines, we assume an algorithm is employed to solve subproblem (a), so that the predicted traffic dynamics can be used as input for solving subproblem (b). Thus the resulting traffic conditions are known and shown as in Figs. 1(b) and 1(c) for time $t + 1$ and $t + n$, respectively, where the red bar represents a severely congested link at the corresponding time step. Meanwhile, there are two requests for a ridesharing service at decision epoch $t$: one request, coloured in red, involves a trip from origin–destination (2–9), i.e., from the centre of the network to the lower right corner; while the other request has a different origin–destination (4–3) and is coloured in blue; both are shown in Fig. 1(b). Given the knowledge of predicted travel times for each link at each future time step, the goal is to search for a solution to assign available vehicles to satisfy these two requests and to generate corresponding paths aiming at avoiding possible further congestion within the predicted horizon.

For the sake of simplicity, let us assume two vehicles are already scheduled to pick up the requests and the route for transporting the blue request is given. In this scenario, we may identify two types of congestion effects, which we denote as I) prior-predicted and II) batch-predicted, and will be explained hereinafter. Let us consider, for example, the red request: there exist three alternative paths $p_1 = \{2, 3, 6, 9\}$, $p_2 = \{2, 5, 6, 9\}$, and $p_3 = \{2, 5, 8, 9\}$ with similar travel time predicted at time $t$. In a static assignment
context, choosing any of these paths is essentially equivalent. However, in dynamic settings, choice $p_3$ should be discarded, since it is predicted that a severe congestion will happen at time $t + n$ (as can be seen in Fig. 1(c)). Note that this prediction is based on the vehicle assignment performed before time $t$, thus the term prior-predicted congestion (coloured in red). An efficient way to circumvent this type of congestion is by employing a time-dependent shortest path algorithm that exploits the knowledge of predicted travel times when routing vehicles.

Additionally, a new congestion may arise because of the batch (route) assignment that is performed at time $t$. For example, if we designate a vehicle to serve the blue request along the path $\{4, 5, 6, 3\}$ and the red request along path $p_2 = \{2, 5, 6, 9\}$, these two vehicles may eventually enter link $(5, 6)$ at the same period. If a similar occurrence happens for a large number of assigned trips, there is a possibility of over-utilising some of the links, whereas some others may remain underutilised. We name this type of congestion as batch-predicted (coloured in yellow). Therefore, path $p_2$ should also be discarded, since it may contribute to batch-predicted congestion in the near future. Eventually, the optimal path is $p_1$. Note that, in this case, we simplify the illustration by separating the process of vehicle assignment (decide which vehicle to pick up which request) and path selection. In reality, the assignment and path decision should be made simultaneously and not sequentially, otherwise it may lead to myopic schedule plans as both decisions are mutual-dependent and closely related.

From the above discussion, the importance of including time-dependent $k$-shortest paths is evident. In fact, selecting only the shortest path for serving a request may cause severe traffic jam (of type II), while these can be minimised by considering multiple alternative routes and determining an optimal combination of those routes at an early stage of path planning. This vehicle assignment and time-dependent multi-path choice problem are integrated and formulated as an integer linear program in Section 4.3.

### 4. Management framework

To solve the aforementioned dynamic ridesharing problem considering dynamic travel time, a new decentralised operation framework, illustrated in Fig. 2, is proposed, based on the work by Simonetto et al. (2019). The strengths of such architecture consists in the fact that the overall (complex) problem is decomposed in separated sub-problems, which are processed in separated modules, while communication and sharing of information is limited. Moreover, as the internal logic of each module is assumed as essentially independent, a certain level of privacy is maintained, allowing multiple, competitive, service providers to operate in such settings. Finally, the decentralised nature of such architecture allows a reduced computational complexity.

#### 4.1. Decentralised architecture overview

Here, we briefly explain the purpose of every module and their interactions, as well as describe how incoming requests are processed.

To start with, travel requests may enter the Ridesharing Platform at any time. We adopt a batch assignment strategy to group requests every time interval $\Delta T$, i.e., at time $t$, all requests arrived during the time slot $(t - \Delta T, t]$ are grouped and processed together. We normally assume $\Delta T$ in the range of 10 s – 30 s.

Batched requests are forwarded to a Vehicles Module, which has the role of producing a subset of vehicles that may potentially serve the travel requests. A distance-based searching method is commonly adopted to reduce the number of available vehicles considered for serving a request. This procedure facilitates the following steps by significantly reducing the problem size. Next, we
need to determine the candidate paths and costs for serving each new request with its corresponding vehicle subset. This involves two steps: (a) finding the optimal insertion plan and (b) generating $k$-shortest paths. At each step, vehicles may be already occupied by some pre-scheduled passengers and this steps requires solving a single vehicle Dial-a-Ride-Problem (DARP) (Marković et al., 2015; Ho et al., 2018) for each potential request candidate. We employ the same insertion heuristic method as Simonetto et al. (2019) to solve the DARP problem, while a new insertion is valid only when it satisfies inequality (2). As a result, the optimal insertion plan with minimum travel time and a series of sequential pick-up and drop-off nodes, is delivered to the Dynamic Routing block.

Nonetheless, only considering the shortest path may cause that severe batch-predicted congestion appear on certain links. Hence, multiple paths are calculated for each optimal insertion plan via the Dynamic Routing block. Given a schedule tuple $(p_1, p_2, \ldots, p_i, d_1, d_2, \ldots, d_i)$ and a link travel time function set $C = \{c_e(t), \forall e \in E\}$, this block aims at calculating the time-dependent $k$-shortest paths (TDKSP) as well as the corresponding travel times. Note that, although computing TDKSP online is time-consuming, especially in a larger network with tens of thousands of nodes and links, parallel computing can be utilised in the above two steps to reduce computation complexity significantly. Eventually, the candidate paths can be further filtered based on whether the time difference among them is within a given tolerance range.

Filtered paths and costs are then forwarded to the Optimisation Module. This module determines which vehicle should serve which request such that the sum of travel cost and predicted congestion are minimised. Once the optimisation problem is solved, the optimal solution is implemented and vehicles receive their assigned requests. These steps are iteratively performed to satisfy real-time ridesharing requests that keep entering into the system.

### 4.2. Traffic prediction module

To account for the fact that link travel time is endogenously affected by the assigned trips and routes, we include a Traffic Prediction Module, which aims at predicting the time–space traffic evolution based on the current traffic condition as well as the routes and departure rates that are designated at each step. In this work, we propose to utilise a DNL model to predict the impact of traffic on each link of the network. In fact, dynamic models outperform static ones in terms of accuracy and precision, with better travel time predictions especially in urban networks. However, note that the proposed modular framework allows to employ also other traffic flow models, either more aggregated (such as BPR functions) or disaggregated (such as microscopic model-based).

In particular, we consider here a differential algebraic equation formulation for DNL to calculate the time-dependent travel time on each link (Han et al., 2019). The general principle behind is the classic Lighthill, Whitham, and Richards theory that depicts the within-link spatial–temporal evolution of traffic dynamics, as described in the following partial differential equation:

$$\frac{\partial k(t,x)}{\partial t} + \frac{\partial Q(k(t,x))}{\partial k} \frac{\partial k(t,x)}{\partial x} = 0, \quad x \in [a,b], \quad t \in [t_0,t_f].$$

(3)

In this paper, we combine a generalised Dijkstra algorithm with Yen’s algorithm to solve the TDKSP problem. The overall time complexity is $O(KN(M + N \log N))$, where $K$ is the number of shortest paths, $M$ is the amount of edges, and $N$ is the number of nodes. However, any other advanced algorithm applying to the TDKSP problem can be employed.
where $Q(k(t, x))$ is the fundamental relationship between flow and density $k(t, x)$, the spatial interval $[a, b]$ denotes a link, $x$ is the position within the link, and $[t_0, t_f]$ is the time interval. The function $Q(k(t, x))$ satisfies $f(0) = f(k^{jam}) = 0$ where $k^{jam}$ represents the jam density. Besides, let $C$ denotes the maximum flow of the link $[a, b]$. Then there exists a unique value $k^{cr}$ at which $Q$ reaches its maximum $Q(k^{cr}) = C$. Here we adopt the widely used triangular fundamental diagram $Q(k)$ to depict the flow-density relationship $q(t, x) = Q(k(t, x))$. Based on the LWR theory, a system of differential algebraic equations is formulated to precisely describe the traffic dynamics across time and space.

Nevertheless, comparing to conventional DNL implementations, where every trip route is predefined and immutable, ridesharing vehicle routes needs to change repeatedly to accommodate to new requests. Consequently, it is necessary to keep track of the path travelled for initialisation and reload the traffic frequently whenever the previous path is altered. The complete procedure for computing link travel times function set $C = \{c_{i,j}(t), \forall e \in E\}$ are illustrated in Fig. 3. Firstly, we convert the vehicle paths to a departure rate matrix, employing an algorithm (Algorithm 1) presented in Appendix B. Each entry of the resulting matrix indicates the rate (veh/s) of vehicles entering link $l$ during time interval $\Delta t$. Furthermore, the DNL model generates link travel times, i.e., edge delay functions $c_{i,j}(t)$. These outputs are forwarded to Vehicles and Optimisation modules and will be utilised to handle the aforementioned two types of congestion. At each time step, new (actual) vehicle position and schedules are retrieved and the predicted rates are updated.

Note that a detailed discussion on DNL models is beyond the scope of this paper and more details on our implementation are provided in Appendices A and C. Interesting readers may refer also to Han et al. (2019) for more details.

### 4.3. Extended linear assignment problem

This section presents a novel integer linear assignment problem to determine the request–vehicle assignment and the corresponding path. This is solved for batched requests at each step $t$. As previously mentioned, for each request–vehicle pair, we assume that a set of TDKSP $\Phi_{i,j} = \{R_{i,j,1}, R_{i,j,2}, \ldots, R_{i,j,k}\}$ and related travel time set $\Psi_{i,j} = \{D_{i,j,1}, D_{i,j,2}, \ldots, D_{i,j,k}\}$, where $k$ denotes the number of shortest paths, are generated by the Dynamic Routing Module. A time tolerance $\epsilon$ is introduced to control the time difference between the shortest and longest path costs, so that $\max(\Psi_{i,j}) - \min(\Psi_{i,j}) \leq \epsilon$. Note that any two different paths belonging the $k$-shortest paths collection may share some arcs, but not necessarily. We introduce integer decision variables $x_{i,j,a}$ that indicate if vehicle $v$ is assigned to serve request $r$ along the path alternative $a$. Moreover, $\overline{\rho}_{i,j,v,a}$ denotes the total travel time for vehicle $v$ to serve request $r$ along the path alternative $a$. Note that vehicle $v$ may already have passengers on board. Let $(t_r, p_1, p_2, \ldots, p_s, d_1, d_2, \ldots, d_s)$ denote the vehicle schedule for serving all assigned (i.e., including the current assignment $(p_t, d_t)$) requests, where $t_r$ is the current vehicle position, while $p_i$ and $d_i$ represent the pick-up and drop-off location of assigned request $r_i$, respectively. Therefore, $\overline{\rho}_{i,j,v,a}$ represents the total travel time for serving all the requests until the last customer has been dropped off. Hence we calculate the total travel and waiting cost for all batched requests as

$$J_1 = \sum_{i \in V^c} \sum_r \sum_a X_{i,r,a} \overline{\rho}_{i,r,a},$$

where $V^c$ is a subset of the fleet $V$, denoting the available vehicles that can potentially serve the request $r$, and $s$ is the maximum number of vehicles considered. By limiting the number of available vehicles, one may considerably reduce the number of decision variables as well as the problem size dramatically, which turns useful especially when handling tens of thousands of vehicles.

We introduce binary parameter $a_{i,j,v,a}$, which can be derived from $\Phi$, and indicates whether vehicle $v$ assigned to request $r$ along alternative path $a$ is planned to traverse link $l$ during $(h, h + 1] \Delta h$. The predicted time span $H$ indicates the horizon considered for
congestion look-ahead prediction, which is further discretised into $h$ sub-timeframes, where $\Delta h$ denotes the step length. This division has to guarantee that each sub-timeframe is greater than the minimum link free-flow time, i.e., $\Delta h = H / h \geq \min(f_l^0)$. Consequently, we calculate the total number of vehicles assigned to each link $l$ at step $h$ as

$$A^h_l = \sum_{v \in V^s} \sum_r \sum_a x_{v,r,a}^h, \quad \forall l \in L, h \in H. \quad (5)$$

We introduce $\rho^h_l$ as the relative remaining capacity for a link $l$ at time $h$ with respect to the other links in the network, defined as

$$\rho^h_l = \frac{C_l - q^h_l}{\sum(C_l - q^h_l)} \quad \forall l \in L, h \in H, \quad (6)$$

where $C_l$ is defined as the number of vehicles corresponding to the unique critical density $k_l$ of link $l$. Note that the number of vehicles on each link $n^q_l$ is the output of DNL model, which has its own discretised step length $\Delta d$ and step index $d$. Hence we can derive:

$$q^h_l = \frac{\lfloor \frac{i \cdot \Delta d}{\Delta t} \rfloor + \frac{\Delta h}{\Delta t}}{\lfloor \frac{i \cdot \Delta d}{\Delta t} \rfloor} n^q_l, \quad (7)$$

where $i$ is the current request batch time and $\Delta t$ is the batch time length, while $\lfloor \frac{i \cdot \Delta d}{\Delta t} \rfloor$ denotes the maximum integer not greater than $\frac{i \cdot \Delta d}{\Delta t}$. The assignment problem is solved every $\Delta t$ second, which corresponds to the requests batching period. Within the optimisation problem, we consider a prediction horizon $H$, which is the horizon during which we calculate (predict) the congestion evolution caused by the ridesharing vehicles. That is, given the vehicle paths known at time $t$, we predict how traffic, i.e., link travel times, evolves during time $(t, t + H)$. The definitions and notations of different time steps are summarised in Table 1.

We then introduce the following cost function term with the goal of prioritising vehicle assignment to links with higher remaining capacity available:

$$J_2 = \sum_{h} \sum_l (1 - \rho^h_l) A^h_l. \quad (8)$$

Note that the bigger the remaining space $\rho^h_l$, the smaller the value of $(1 - \rho^h_l)$, thus, by minimising the summation of all $(1 - \rho^h_l) A^h_l$, we basically attempt to assign as many vehicles as possible to links with a larger remaining space.

To summarise, the multiple path ridesharing and vehicle assignment problem can be formulated as the following multi-objective integer linear programming (ILP) problem

$$\min \{ J_1, J_2 \} \quad (9)$$

subject to

$$\sum_r \sum_a x_{v,r,a} \leq 1, \quad \forall v \in V^s \quad (10)$$

$$\sum_{v \in V^s} \sum_r x_{v,r,a} \leq 1, \quad \forall r \quad (11)$$

$$\sum_{v \in V^s} \sum_r \sum_a x_{v,r,a} = \min\{ p, m \} \quad (12)$$

$$x_{v,r,a} \in \{0, 1\}. \quad (13)$$

Two objectives are considered in (9): $J_1$ aims at minimising the total travel time; while $J_2$ aims at prioritising routes that include links with larger remaining capacity. Constraint (10) guarantees that one vehicle serves at most one request via one route at a time; similarly, (11) guarantees that a passenger can be picked up at most by one vehicle along a path. Eq. (12) ensures that the number of successfully matched vehicle–request pairs is equivalent to the smaller value between the number of requests $p$ and the available fleet size $m$. Finally, (13) indicates that $x_{v,r,a}$ is a binary decision variable. Nonetheless, integer variables $x_{v,r,a}$ can be relaxed to continuous ones, since our extended formulation features the structure of a classic linear assignment problem (Kennington and Helgason, 1980; Papadimitriou and Steiglitz, 1998; Simonetto et al., 2019). This means that the solution of the ILP can be obtained by solving the corresponding linear programming (LP) relaxation. As a result, the above problem can be efficiently solved for large scale vehicles and requests.

One conventional approach to combine the two objectives is to simply employ their weighted sum, where each objective is multiplied by a pre-defined weight. However, as objective $J_1$ and $J_2$ have distinctive units and their values may have strong variations at different decision epochs, it is difficult to combine them by setting a fixed weight. Therefore, we propose to first solve the problem with a single objective that is entitled with a higher priority and then to solve the problem with a lower-priority objective, restricting the value corresponding to the first objective to be not too worse than the previous optimal value, via introducing an additional constraint. Here, we emphasise on solving the problem with higher priority assigned to objective $J_2$, 

---

Z. Zhou and C. Roncoli
Transportation Research Part C 139 (2022) 103658

---
but different priority settings are also tested in our simulation experiments. In particular, if we label the second objective function $J_2$ with a higher priority, then the multi-objective problem reduces to the following problem:

$$\min \sum_{i \in V_i^d} \sum_r \sum_a x_{i,r,a} \bar{p}_{i,r,a}$$

subject to (10)–(13) and

$$\sum_i \sum_h (1 - \rho_i^h) A_i^h \leq \epsilon_{opt},$$

where $\epsilon_{opt}$ is the optimal value of the single objective function ($J_2$), calculated by minimising (8) subject to (10)–(13). This multi-objective (ILP) problem can be solved directly by existing optimisation solvers, such as Gurobi (Gurobi Optimization LLC, 2020), which directly provides tools to optimise multi-goal problems in a hierarchical approach. The principle behind is solving these problem in a order such that the objective values of higher-priority objectives would not be degraded.

### 4.4. Problem extensions

A number of extensions of the formulated strategy may be envisaged, to deal with special situations when needed, under the requirement of preserving the structure of the cost criterion and the linearity of the constraints.

#### Long-distance trips or long-waiting requests

Minimising $J_1$ may lead to rejecting trips characterised by a higher cost, such as long-distance trips, in case there are not enough vehicles ($m < n$) available within a batch. Moreover, this can happen repeatedly within consecutive batches, which may cause further delays for some requests. This problem can be handled in a heuristic way, by adding the following constraint

$$\sum_{i \in V_i^d} \sum_a x_{i,r,a} = 1,$$

where $V_i^d$ denotes a subset of vehicles that can potential serve the request $\bar{r}$ without consideration of constraint (2a). Requests $\bar{r}$ can be chosen, for example, by introducing a criterion accounting if a long-trip request has not been accepted after several rounds of assignment or if a request is approaching its maximum waiting time.

An alternative approach for dealing with the same issue could be to introduce a “discount” weighting factor, $\lambda_{i,r,a}$, in cost $J_1$, to prioritise the assignment of long-waiting trips; the resulting cost formulation is

$$J'_1 = \sum_{i \in V_i^d} \sum_r \sum_a x_{i,r,a} \bar{p}_{i,r,a} \lambda_{i,r,a}, \quad \lambda_{i,r,a} \in (0, 1),$$

where, normally, $\lambda_{i,r,a} = 1$, while a value $\lambda_{i,r,a} < 1$ is used for specific trips; methods for iterative update $\lambda_{i,r,a}$ can also be envisioned.

#### Alternative cost function formulation accounting for link speed

The objective $J_2$ aims at achieving an more uniform utilisation of the available space for all links. However, one may argue that such formulation does not take directly into account the actual speed on the links, which may possibly lead to unfavourable outcomes, such as assigning vehicles to very slow streets just to balance capacity. Although the path travel time threshold $\epsilon$, introduced to ensure that the selected $k$-shortest path candidates have similar travel time, this could be explicitly taken into account by considering a speed-related weight in the formulation of objective $J_2$. In particular, we introduce weight $\frac{1}{v_i^h}$, where $v_i^h$ is the speed of link $l$ at time $h$ and define the flowing two alternative cost functions:

$$J'_2 = \sum_i \sum_h \frac{1}{v_i^h} A_i^h$$

$$J''_2 = \sum_i \sum_h (1 - \rho_i^h) \frac{1}{v_i^h} A_i^h.$$

In both functions $J'_2$ and $J''_2$, the weighing term is aimed at penalising assignments of vehicles to links characterised by low speed, either as unique criterion in (18) or in combination with a weight penalising the remaining capacity available in (19). These different cost functions have been tested in Section 5.1.1.

#### Aggregating requests from the same batch

As stated previously, the proposed method restricts that one vehicle is entitled to serve only one request within each batch. However, there may be cases when there are multiple requests with the same OD arriving simultaneously, including, e.g., people travelling together. Although one request is implicitly related to a single traveller, this can be easily extended to multiple travellers if they share the exactly same trip details (and constraints). This can be implemented by adding to requests an attribute for passenger numbers $n_r$, then, such request can be treated, without loss of generality, as a single travel request that is available only for vehicles with required capacity.
Incorporating exogenous traffic

Despite the proposed framework only considers congestion produced by ridesharing vehicles, a prediction of exogenous traffic can be incorporated. This requires knowledge of the departure rate matrix related to other traffic modes, which can then be fed into the DNL algorithm. If needed, dynamic traffic assignment can also be considered to more accurately model routing of human driven vehicles.

4.5. Model scalability

This section discusses the computational complexity of the proposed framework and its scalability properties, which are important for handling real-time large-scale scenarios. As our framework inherits some components from (Simonetto et al., 2019), we share various features in computational complexity of different modules and algorithms, e.g., the context mapping (for finding nearest available vehicle subset) and insertion heuristic. On the other hand, the proposed framework may have an increased computational burden mainly because of (a) including a traffic prediction module, (b) calculating online the TDKSP solution, and (c) solving the linear assignment problem.

The computational complexity for the traffic prediction model depends on the specific model used. In this article, we adopt a DNL model designed for large-scale networks (Han et al., 2019), which has proven efficient for large-scale networks. The computation time can be further reduced if a link-based DNL method is adopted instead of the typical path-based implementation (see, e.g., Patwary et al. (2021)). However, if performance is to be further improved, a simpler, static, model, e.g., based on BPR function, can be employed, which is characterised by a computation complexity of $O(n)$, where $n$ is the number of links.

As we discussed earlier, the time complexity for solving the TDKSP depends on the algorithm used. There exist various highly efficient algorithms (Ding et al., 2008; Zhao et al., 2008; Dehne et al., 2012; Yu et al., 2020) for finding both time-dependent shortest paths over large networks in polynomial time. Nonetheless, for thousands of vehicles and frequent queries, computing this in a centralised manner would cause a high computational workload. Thus, parallel computing is an efficient way to calculate real-time assignment costs. When operators receive a batch of requests, they can map each request to various available vehicles and distribute the computation of the optimal insertion plan, solving the TDKSP, and calculate corresponding costs independently. The resulting paths and costs are then forwarded to the central dispatcher, employing them for further calculations. With parallelised calculations, the computational complexity is essentially reduced to a constant, irrespectively of the number of vehicles.

Finally, setting up and solving the modified linear assignment problem may contribute to the computational load. When constructing the optimisation problem, calculating $A^h_k$ for each path alternative in a centralised manner could be time consuming since calculating parameter $a^h_{v,r,a,l}$ requires checking every vehicles path and the predicted entering time within each link that belongs to the path. However, $a^h_{v,r,a,l}$ can be computed directly while computing the TDKSP solution, thus a decentralised calculation of the TDKSP further decreases the computation burden. Additionally, the bi-objective assignment problem (9)–(13) is in general a NP-complete problem (Ehrgott, 2005), which is hard to solve. However, the total unimodularity property of the constraint matrix can be fully exploited if two objectives are integrated using weighted sum method (Seo and Asakura, 2021). Consequently, efficient algorithms (e.g., Bertsekas (1981)) can be exploited to solve such problem with very large sparse matrices in a time in the order of seconds (Simonetto et al., 2019).

5. Numerical simulation

In this section, a set of numerical experiments are designed to demonstrate the efficiency and robustness of the proposed framework and model. A first set of simulations is implemented on a toy network, considering various demand scenarios; in light of lower computational costs and simplified network, we show the effect of various model parameters on the system performance. A second group of experiments is conducted on a realistic urban network with peak-hour demand patterns, where the performances of different vehicle operation strategies are compared. All the experiments are coded in Python 3.7 and run on an i7-8750H 2.20 GHz computer with 8 GB RAM. For the DNL algorithm, we adopt the Matlab code package provided by (Han et al., 2019), which is released as open-source software and proved to be applicable for large-scale networks. The Python Matlab engine is thus used to bridge the code gap. Gurobi optimiser is employed to solve the optimisation problem.

5.1. Simulation experiments on the toy network

5.1.1. Network description and baseline scenario

As shown in Fig. 4, we consider here a grid network composed by 16 nodes and 48 homogeneous uni-directional links. The link attributes include capacity, length, free-flow time, which are set to 0.3 veh/s, 2000 m, and 180 s, respectively. Similarly as in the work by Han et al. (2019), the speed of backward shockwaves is assumed to be a third of the free flow speed. Therefore, the critical and jam density are 0.027 veh/m and 0.108 veh/m, respectively, which can be straightforwardly calculated and employed to identify a triangular fundamental diagram. On top of network parameters, we vary the remaining scenario-related parameters to investigate how they affect system performance and computation time.

**Baseline scenario.** Table 1 provides all the parameters that were later varied for sensitivity analysis. The travel requests are randomly generated between two arbitrary nodes. We define a request arrival rate following the Poisson distribution for the first hour of simulation, which is further divided into three periods, characterised by request arrival intensity $\lambda$, namely the expected number of requests entering the system during a unit time, being 3.75, 7.5, and 3.75 requests per second, respectively. In total, there
are 9000 requests waiting to be served, while 1000 vehicles with four seats are present in the network, with their initial position randomly assigned to a node. When no further requests are assigned to a vehicle, it is assumed to stay at the last visited node until a new request is assigned (i.e., we do not consider any repositioning algorithm). For every request, the vehicle operators are responsible of identifying 15 suitable vehicles. However, if the number of available vehicles is less than 15 (even 0 vehicles available may occur during peak hours), the system continues operating with this number of vehicles and unmatched requests would wait for the next-round assignment. Moreover, we assume that each request has some tolerance in maximum waiting time and delay, both set as 300 s, which ensures vehicles can pick up the designated passengers within a distance of less than two links. For a fair comparison, we assume unserved requests would wait indefinitely until being re-dispatched. Nonetheless, the optimisation problem could be infeasible when no available vehicles can satisfy the request. In this case, we may force a subset of request being served if the original problem (9) – (14) is infeasible. The incoming requests are grouped every 20 s. Although the simulation step is 20 s, the link travel time (computed via the DNL method) is updated every 3 steps (60 s) to improve computational efficiency. We optimise the vehicle path with a 20 min prediction horizon, which is further divided into 8 sub-periods. Finally, we set a higher priority for the objective $J_2$ in the baseline scenario.

5.1.2. Sensitivity analysis of parameters variations

We measure the impacts of different choice of parameters on the performance of the system and document all simulation results with different parameters values in Table 2, where columns include the metrics considered for performance evaluation. The first index, all requests processed, records the time step when all the requests are assigned to vehicles and no more requests are waiting for service. Index all request completed indicates the time when all the requests have been completed (i.e., all travellers have reached their destinations). Note that, to prevent unreasonably long simulation time, which may happen when some links are extremely congested, we set a 2-hour threshold after which the simulation is terminated. The unit of the first two indexes is time step where one step represents 20 s. Index Average Static Travel Time (Avg STT) is the quotient between the sum of passengers’ static travel time, i.e., if all vehicles would travel with free-flow time, and the number of requests. The following two indexes are the Average Actual...
Table 2: Sensitivity analysis of different parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Scales</th>
<th>Service quality</th>
<th>Traffic condition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>All requests processed</td>
<td>All requests completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[time step]</td>
<td>[time step]</td>
</tr>
<tr>
<td>Baseline scenario</td>
<td></td>
<td>216</td>
<td>315</td>
</tr>
<tr>
<td>Capacity</td>
<td></td>
<td>1</td>
<td>(5178/9000)</td>
</tr>
<tr>
<td>Maximum waiting</td>
<td></td>
<td>200</td>
<td>218</td>
</tr>
<tr>
<td>and detour time</td>
<td></td>
<td>600</td>
<td>268</td>
</tr>
<tr>
<td>DNL time step</td>
<td></td>
<td>50</td>
<td>252</td>
</tr>
<tr>
<td>Fleet size</td>
<td></td>
<td>500</td>
<td>252</td>
</tr>
<tr>
<td>DNL update frequency</td>
<td></td>
<td>40</td>
<td>228</td>
</tr>
<tr>
<td>Maximum number of vehicles</td>
<td></td>
<td>5</td>
<td>274</td>
</tr>
<tr>
<td>Requests batch period</td>
<td></td>
<td>10</td>
<td>255</td>
</tr>
<tr>
<td>Optimisation prediction</td>
<td></td>
<td>10 min</td>
<td>213</td>
</tr>
<tr>
<td>Optimisation step</td>
<td></td>
<td>30 min</td>
<td>207</td>
</tr>
<tr>
<td>Optimisation priority</td>
<td></td>
<td>4</td>
<td>195</td>
</tr>
<tr>
<td>Alternative cost functions</td>
<td></td>
<td>5</td>
<td>202</td>
</tr>
<tr>
<td>In-Vehicle Time (Avg AIVT)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| and the Average Actual Waiting Time (Avg AWT), respectively. Note that we consider that passengers’ travel time is the sum of waiting time and in-vehicle time. Consequently, travel delay can be derived from the sum of real in-vehicle time and waiting time minus static travel time. The metrics above are mainly designed to assess the service quality.

Furthermore, to evaluate the network congestion degree, the Average Link Travel Delay (Avg LTD), \( \mu \), and Average Link Travel Time Standard Deviation (Avg-std LTT), \( \sigma \), are computed as follows:

\[
\mu = \frac{\sum_l (\bar{\mu}_l - f_0^l)}{L}; \\
\sigma = \frac{\sum_l \bar{\sigma}_l}{L};
\]

where

\[
\bar{\mu}_l = \frac{\sum_t f_t^l}{n_t}; \\
\bar{\sigma}_l = \sqrt{\frac{\sum_t (f_t^l - \bar{\mu}_l)^2}{n_t}};
\]

\( f_0^l \) is the free-flow time on link \( l \); \( f_t^l \) represents the actual link travel time during \([t-1, t]\) for link \( l \); \( n_t \) is the number of DNL time steps; and \( L \) denotes the total number of links. Variable \( \bar{\mu}_l \) denotes the time-mean travel time on link \( l \). Hence, we design the metric \( \mu \) to evaluate the average congestion degree across different links. For example, value \( \mu = 59.6 \) means that, on average, the travel time is 59.6 s longer than the free-flow travel time (which, in our experiments, is 180 s across all time steps and links), which results in an actual average link travel time of 239.6 s. Similarly, the travel time deviation \( \bar{\sigma}_l \) across the whole simulation horizon is utilised to assess the travel time fluctuation level on link \( l \). Average deviation \( \sigma \) among all links reflects the time–space mean fluctuation of...
travel delay, representing also a proxy for travel time reliability. In particular, we calculate all the link-related mean values within the first 70 min interval, which is chosen since most requests are served within one-hour.

The following scenarios are evaluated, by maintaining the baseline scenario settings while not mentioned otherwise.

Vehicle capacity We vary the vehicle capacity in the set 1, 4 (baseline scenario), 8. Setting capacity equal to 1 implies that actually no ridesharing is considered, but each vehicle serves at most one request at a time (i.e., operating in a similar way as a traditional taxi service). Results show that only 57.5% of requests are satisfied by the end of simulation (which is forced to terminate at step 360, i.e., after 2 h). Compared with shared rides, requests without ridesharing usually have shorter waiting and in-vehicle travel time, which is due to the fact that the vehicle travels to pick-up and drop-off passengers directly from origin to destination, without intermediate stops. Besides, average link delay and deviation are smaller, as fewer requests are satisfied. In summary, ridesharing can greatly improve service quality by serving many more requests without causing severe traffic jam. Furthermore, increasing vehicle capacity from 4 to 8 passengers does not have a strong impact, except causing a longer computation time. This is mainly due to the demand level in the tested scenario. In fact, we can clearly see that there is a large improvement when vehicle capacity is increased from 1 to 4 passengers, with a number of served requests increased from 5178 to 9000, while the network congestion indicators increased only slightly. However, considering a larger vehicle capacity does not generate any further advantages since larger capacity, despite may generate more shared rides, may also cause more detours and lead to longer waiting and travel time. Finally, it is reasonable to expect that with higher levels of demand, also increasing vehicle capacity would lead to better performance.

Maximum waiting and detour times We varied the maximum waiting and detour time from 200 s to 600 s. When the thresholds are low, the system is limited to assign nearby vehicles to serve requests. As a result, passengers experience shorter waiting and detour time as well as less congested network. On the other hand, when increasing the thresholds, longer trips become feasible, causing all indicators to grow. Therefore, choosing a relative small number but still allowing vehicles to travel a short distance could improve the system performance.

DNL time step DNL time step is varied in the set 30 (baseline scenario), 50, 150 s. All the value of metrics increase when enlarging the step length, since a smaller time step allows to generate more accurate predictions. However, a finer step requires more computational time and memory, especially for a large network. The trade-off between accurate prediction and memory use should be taken into account when selecting an appropriate time step.

Fleet size We test the performance of the proposed method with different fleet sizes. We observe that the link travel time increases with the growth of fleet size. On the other hand, from a passengers’ perspective, the average waiting time drops initially but increases afterwards because of network congestion. Meanwhile, the average in-vehicle travel time does not decrease as one may expect, due to the higher number of vehicles on road, which leads to stronger congestion, which, in turn, causes longer travel times.

Traffic updating periods The frequency of running Traffic Prediction Module is varied from every 40 s to 120 s. As the results show no obvious changes between different frequencies except for the waiting time index, we can conclude that the system is robust when the frequency is varied within two-minute horizon. Therefore, we can slightly increase the time gap when running the DNL without incurring losses to the system performance.

Maximum number of vehicles The maximum number of nearby available vehicles selected to satisfy each request is varied in the set 5, 10, 15, 20). As the number of available vehicles increases, as shown also in Fig. 5(a), link travel time declines slightly while passengers’ average waiting time drops significantly. However, when this value increases from 15 to 20, the differences between
two scenarios become negligible, implying that, beyond these values, keeping to increase the number of available vehicles does not lead to improvements in terms of service quality or traffic efficiency. However, this would require more computation time and resources to solve the optimisation problem.

**Requests batch period** The requests are grouped every 10, 20 (baseline case), and 40 s. As the batching step increases, as shown also in Fig. 5(b), the values representing network congestion indicators drop steeply at first and then flatten out. As a consequence, passengers manage to be transported more efficiently and thus experience a shorter travel time. Nonetheless, the average waiting time drops firstly but increasing significantly afterwards because larger batch period means longer waiting time. Although increasing the batch period may decrease the possibility of sharing, as sharing cannot happen within the same batch, the benefits derive from a “better” optimisation for each batch. Hence it is advisable to increase the batch periods properly, without violating the maximum waiting time constraint.

**Optimisation prediction horizon** Except for the baseline scenario, we tested the other two cases with optimisation predicted horizon $H$ set to 10 min and 30 min. When increasing the horizon length, as it is displayed also in Fig. 5(c), service-related metrics, such as passengers' average waiting and in-vehicle time, decrease, while traffic-related indicators, such as average link delay, increase slightly. One explanation may be that the long-term prediction is not robust due to the highly dynamic nature of such system. In fact, as the average travel time is around 10–20 min, only a few vehicles are predicted to travel across the network after 20 min. Therefore, the congestion-avoidance strategy loses its effectiveness once the network is almost empty. Nonetheless, the real network condition after 20 min may not be empty due to the continuous arrival of new travel requests, which need to be assigned to vehicles. Besides, both the constantly changing vehicle routes and link travel times greatly affect the accuracy of long-term prediction, while remaining still accurate in the short-term. One supporting evidence is that the results obtained while considering only objective $J_1$ or prioritising $J_1$ are similar to the 30-minute horizon case, which indicates that a longer prediction horizon weakens the effectiveness of objective $J_2$.

**Optimisation steps** The optimisation step $s$ is the number of predicted horizon discretisation. Fixing the optimisation horizon as 20 min, we vary $m$ in the set $\{4, 5, 8\}$; hence, for each scenario, the time step is 300 s, 240 s, 150 s, respectively. The simulation results are visualised in Fig. 5(d), revealing that a finer discretisation leads to better traffic conditions. Although the average in-vehicle time and waiting time increase, the actual travel delay remains approximately the same. This can be explained by the longer static travel time, indicating that the vehicles assigned to transport the passengers tend to have longer travel time even without considering congestion.

**Optimisation priority** The original optimisation problem has two objectives: I) minimise total travel time and II) prioritise the usage of less congested links. We tested the following four different optimisation strategies:

(a) $\{J_2\}$ only objective $J_2$ is considered;

(b) $\{J_1\}$ only objective $J_1$ is considered;

(c) $\{J_1, J_2\}$ hierarchical objective, where $J_2$ has a higher priority (the same as in baseline scenario);

(d) $\{J_1, J_2\}$ hierarchical objective, where $J_1$ has a higher priority.

By comparing strategies (a) and (b) with the baseline scenario, one can observe that considering a single objective, either $J_1$ or $J_2$, leads to worse outcomes: while only considering objective $J_1$, the system has better performance in terms of service quality but worse resulting traffic conditions; while all indicators are worse when only $J_2$ is considered. This further demonstrates that the overall less congested network does not guarantee a superior service quality and vice-versa. Therefore, it is strongly beneficial to include the proposed two objectives while implementing request–vehicle assignment. Moreover, prioritising the objective $J_1$ improves the service quality related metrics. However, the traffic condition under strategy (d) is worse than the baseline scenario, revealing that emphasising only on the travel cost minimisation can lead to even stronger congestion when serving identical demand.

**Alternative cost functions accounting for link speed** We test the alternative cost functions for $J_2$ introduced in Section 4.4, i.e., (18) and (19), to assess how different formulations of the objective function explicitly accounting for weights defined based on link speeds affects the performance of the proposed method. We can observe that employing the cost function $\{J_2, J_1\}$ (i.e., the baseline scenario) leads to better results in terms of all metrics. This is more evident for the metrics describing service quality (Avg STT, Avg AIVT, and Avg AWT), with losses in the order of 20 s – 100 s; whereas the congestion metrics (Avg LTT and Avg std LTT) seem to be less affected by the different cost functions. That may be due to the fact that the different objectives ($J_1$ and $J_2$, $J_2^\prime$, or $J_2^\prime\prime$) are intertwined. In fact, while only minimising $J_1$, we aim at serving passengers in the shortest possible time, which implies that service quality-related metrics are better when congestion is not too severe. In contrast, when minimising also $J_2$, $J_2^\prime$, or $J_2^\prime\prime$, we are sacrificing the performance of $J_1$ to prioritise the links that are less used ($J_2$) or have higher speed ($J_2^\prime\prime$), so as to the resulting congestion is eased, which, however, causes also a deterioration of service quality. Comparing cost functions $\{J_2, J_1\}$ and $\{J_2^\prime, J_1\}$, we observe that the service-related indicators increase while traffic condition is almost the same, demonstrating that $\{J_2^\prime, J_1\}$ is also capable of alleviating the congestion but with a higher sacrifice on service quality. Instead, $\{J_2, J_1\}$ can achieve the same effect with a smaller increase in average waiting time. Also, it is worth noting that cost function $\{J_2^\prime\prime, J_1\}$ leads to the worst performance (see Fig. 6) (see Table 3).

5.2. Simulation experiments on a realistic network

5.2.1. Sioux falls network

The Sioux Falls network, as shown in Fig. 6, is a real-world network composed of 24 nodes and 76 links. The attributes of the links are set according to the open data source provided by the Transportation Networks for Research Core Team (2021), which also
Zhou and Roncoli

provides demand matrices. The attributes of the sample links are given in Table 3. The original static travel demand is distributed over six consecutive 10-minute periods according to a dynamic departure time weight vector \([0.15, 0.18, 0.20, 0.21, 0.16, 0.10]\), as in the work by Pel and Nicholson (2013). Since we only consider ridesharing vehicles, only one third of the demand is used for simulation. The requests within each 10-minutes horizon are randomly distributed; the demand profile over a one-hour horizon is shown in Fig. 7. In total, there are 120000 trip requests that wait to be served, while 10000 vehicles with four seats are present in the network, with their initial position randomly assigned to a node. The maximum waiting time and delay are both set equal to 400 s. The requests are grouped every 10 s, while the link travel time is updated every 6 steps to improve computational efficiency.

**Table 3**

<table>
<thead>
<tr>
<th>LinkId</th>
<th>From_node</th>
<th>To_node</th>
<th>Capacity [veh/s]</th>
<th>Length [m]</th>
<th>FFT [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>7.19</td>
<td>9656</td>
<td>360</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>6.50</td>
<td>6437</td>
<td>240</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>7.19</td>
<td>9656</td>
<td>360</td>
</tr>
<tr>
<td>76</td>
<td>24</td>
<td>23</td>
<td>1.41</td>
<td>3219</td>
<td>120</td>
</tr>
</tbody>
</table>

**Fig. 6.** Simplified road network for simulation.

**Fig. 7.** Requests distribution.
Table 4: Scenario of different strategies.

<table>
<thead>
<tr>
<th>Modules</th>
<th>Scenarios</th>
<th>$P_S^a+MC^b$</th>
<th>$P_D^a+MC^c$</th>
<th>$PD^b+MC$</th>
<th>$CS^a+MC^d$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optimisation</td>
<td>MC</td>
<td>MC</td>
<td>MC</td>
<td>MP</td>
<td></td>
</tr>
<tr>
<td>Ridesharing Platform</td>
<td>Static</td>
<td>Current</td>
<td>TDSP</td>
<td>TDSP</td>
<td></td>
</tr>
<tr>
<td>Vehicle Routing</td>
<td>Static</td>
<td>Current</td>
<td>TDSP</td>
<td>TDSP</td>
<td></td>
</tr>
<tr>
<td>Traffic Prediction</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Vehicle updating</td>
<td>actual speed</td>
<td>actual speed</td>
<td>actual speed</td>
<td>actual speed</td>
<td></td>
</tr>
</tbody>
</table>

*Complete static,  bMinimise cost,  cPartial static,  dPartial dynamic,  eComplete dynamic,  fMulti-path.

Table 5: Performance of different strategies.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CS+MC</td>
<td>(118628/120000)</td>
<td>1440</td>
<td>901.6</td>
<td>1571.1</td>
<td>3136.4</td>
<td>126.4</td>
<td>89.0</td>
</tr>
<tr>
<td>PS+MC</td>
<td>671</td>
<td>1023</td>
<td>917.1</td>
<td>1018.6</td>
<td>1317.2</td>
<td>73.7</td>
<td>75.8</td>
</tr>
<tr>
<td>PD+MC</td>
<td>671</td>
<td>935</td>
<td>852.2</td>
<td>755.2</td>
<td>1387.2</td>
<td>62.7</td>
<td>54.6</td>
</tr>
<tr>
<td>CD+MP</td>
<td>510</td>
<td>748</td>
<td>900.7</td>
<td>798.0</td>
<td>805.5</td>
<td>32.6</td>
<td>43.7</td>
</tr>
</tbody>
</table>

5.2.2. Tested scenarios

The following four scenarios are devised to evaluate the proposed strategies, which are also summarised in Table 4.

- Scenario 1: complete static + minimising travel cost. When assigning requests to vehicles, the shortest path between two nodes is calculated based on static travel time. Besides, only the shortest path for each request–vehicle pair is considered, while the optimisation module utilises only objective $J_1$. However, during the simulation, we still calculate realistic link travel times, which are taken into account when updating vehicles' position and status.
- Scenario 2: partial static + minimising travel cost. In contrast to Scenario 1, spatio-temporal varying travel times, which is calculated via the DNL algorithm, are considered while assigning and dispatching vehicles. However, only the current travel time (assumed constant in future time steps) is utilised when determining the shortest path. Besides, we still consider only objective $J_1$.
- Scenario 3: partial dynamic + minimising travel cost. Based on predictions delivered by the DNL algorithm, we calculate a time-dependent shortest path algorithm to find the best route option. Only the shortest path is considered and objective $J_1$ is the sole objective.
- Scenario 4: complete dynamic + multi-path. In this scenario, not only predicted dynamic travel times are considered, but also the multi-path congestion avoidance strategy is implemented. In particular, we consider three possible path alternatives for each request–vehicle pair and solve the multi-objective problem (8)–(12), where objective $J_2$ is entitled with a higher priority.

5.2.3. Results and discussion

Table 5 shows a comparison of performance for the different optimisation and congestion-aware strategies. Despite scenario $CS+MC$ is unrealistic, we include it as a baseline scenario for comparison. Since we assume that the vehicle routing module is unable to perceive the changes of traffic dynamics, vehicles are assigned and routed completely based on static travel time. As a consequence, frequently used links are extremely congested, where the travel time increases up to 2 h, whereas other links remain virtually empty. This also explains the existence of 1372 unsatisfied requests at the end of the simulation (shown as a red cell in Table 5). Note that, although unrealistic, various existing studies consider these assumptions for the request–vehicle assignment. Scenario $PS+MC$ considers real-time travel delay but no prediction. At each decision epoch $t$, the shortest path is calculated based on current travel time. This scenario is essentially representative of state-of-the-art strategies (Liang et al., 2020; Alisoltani et al., 2021) when dispatching ridesharing vehicles in consideration of traffic congestion. Compared to scenario $CS+MC$, all the requests are successfully handled at time 671. Both passengers' mean in-vehicle time and waiting time are significantly reduced because of the smoother traffic flow. Nonetheless, such scenario is improved by introducing the time-dependent shortest path approach with a purpose of avoiding prior-predicted congestion. In fact, scenario $PD+MC$ outperform the myopic scenario $PS+MC$ in most metrics. Although the mean waiting time slightly increases, the average in-vehicle time decreases remarkably, due to the less congested network. Finally, the fully dynamic scenario $CD+MP$, which accounts for the two types of congestion outperforms all previous strategies in all metrics. In fact, because of the minimised travel delay across time and the network, the service quality is improved significantly.

Figs. 8 Top and Bottom show time-mean travel delay (i.e., time-mean travel time minus link free-flow time) on links where the travel delay is larger than 30 s. Obviously the scenario $CD+MP$, which is marked in yellow, produces the least congested network. Furthermore, the trends of number of vehicles on the most congested links are shown in Fig. 9. As a comparison, the number of
Fig. 8. Time-mean link delay.

Fig. 9. Number of vehicles on a selection of most congested links under four scenarios.

Fig. 10. Number of vehicles on less congested links under four scenarios.

vehicles on less congested links are selected and displayed in Fig. 10. The four colours denote aforementioned four scenarios. The horizontal line marks the critical number of vehicles derived from the link critical density. Nonetheless, some horizontal lines are not shown in the figures as the corresponding critical values exceed the maximum value of the $y$ axis.
By comparing the trend of the red line in Figs. 9 and 10, we notice that there is one obvious feature: in congested links, the red line maintains the number of vehicles at a minimum level, whereas the red lines in less congested links maintain relatively the highest level, still without exceeding the critical value. For instance, let us observe the most congested links (29, 48, 49): the average travel delay exceed 40 min in the scenario CS+MC. However, as we assume that vehicles are unable to perceive the congestion, in the process of assignment and route planning, the number of vehicles assigned to these links (blue line) keep increasing as presented in Figs. 9(b) and 9(c), which even causes gridlocks. Although scenarios PD+MC and PS+MC can alleviate congestion by reacting promptly to the real-time or even predicted traffic conditions, the overall effect is inferior to the scenario CD+MP. This phenomenon can be explained by the dynamic travel time prediction and multi-path assignment scheme. Specifically, scenario CD+MP maintains the number of vehicles below the critical value in those congested links. Moreover, our strategy diverts the overloaded traffic to other undersaturated links without exceeding the critical volume. These two figures clearly demonstrate the strength of our strategy in better utilising the road capacity. In fact, by avoiding two types of predicted congestion while assigning and routing vehicles, our strategy improves not only the ridesharing service quality but also the network traffic efficiency.

6. Conclusions

Despite the fact that previous studies overlooked the congestion effect caused by ridesharing vehicles or SAVs, the popularity and growth of ridesourcing services and the advent of SAVs is expected to boost the large-scale adoption of ridesharing. As a result, these on-demand vehicles have a great potential for disrupting the transportation system. In this paper, we proposed a computationally efficient strategy to solve the dynamic ridesharing vehicle assignment problem in the presence of dynamic travel time. In particular, we designed a framework that includes a traffic prediction module and we defined two types of congestion in the process of request–vehicle assignment and routing. Furthermore, we formulated an assignment problem that considers both multi-path alternatives and the congestion potentially caused by ridesharing assigned requests and routes. The proposed strategy is first tested in a small grid network to evaluate how different parameters affect the simulation results and system performance. Then, another set of simulations are implemented for a realistic network considering a high demand. Four different congestion-aware scenarios, considering different optimisation strategies, are examined. The results indicate that our strategy can significantly reduce congestion endogenously caused by ridesharing vehicles, outperforming strategies with assumptions similar as current state-of-the-art. Moreover, the decentralised framework shed the light on how to efficiently solve real-time city-scale dynamic ridesharing problem considering both endogenous and exogenous traffic congestion.

Nevertheless, several open questions remain unsolved. The current study assumes that vehicle stops once arrived at their destination if there are no passengers on board; however, how to effectively re-balance those empty vehicles is also crucial for congestion reduction (Levin and Boyles, 2015; Levin et al., 2019). Moreover, despite we mentioned that multiple ridesharing operators can offer the same service simultaneously, this study has not investigated how multi-company competition may be implemented and it would be interesting to explore the impact of competition on system performance in future research. Moreover, our simulations experiments assume that there are only ridesharing vehicles in the network, while other travel modes, such as private cars and buses, exist in a real transportation system, with implications on both travellers choices and congestion impact. Finally, incorporating pricing and travellers' preferences into the model would make these results more appealing for ridesharing companies.

CRediT authorship contribution statement

Ze Zhou: Conceptualisation, Methodology, Software, Validation, Formal analysis, Writing – original draft, Visualisation. Claudio Roncoli: Conceptualisation, Methodology, Writing – review & editing, Supervision, Funding acquisition.

Acknowledgement

This research is partly funded by Henry Ford Foundation Finland (Henry Fordin Säätiö) and the FINEST Twins Center of Excellence (H2020) under Grant 856602. The authors are very grateful to the three anonymous Reviewers and the Editors who provide valuable comments to help improving the quality of the paper.

Appendix A. Departure rate matrix

Before running the DNL algorithm in our experiments, we need to convert the real-time assignment results to a departure rate matrix, which is treated as an input for the DNL module. The desired departure rate is a matrix of dimension $L \times T$, where $L$ is the number of links and $T$ is the number of time steps. In the original DNL model, rows correspond to all the possible routes between all O-D pairs, meaning that all paths need to be enumerated. However, path of ridesharing vehicles are changing constantly for two reasons: first, vehicle path is altered whenever new pick-up and drop-off nodes are inserted; second, when the traffic is propagating through the network, the shortest path between two arbitrary nodes is changing accordingly. Since it is not realistic to enumerate all the possible time-dependent shortest paths for all the OD pairs in advance for a large-scale network, the model needs to be modified to adapt to the changes when integrated with a ridesharing system. We solve this issue by disaggregating the paths $p = (l_1, \ldots, l_k)$ into links $l \in L$ to flexibly generate time-dependent $k$-shortest paths. Hence, the rows in the departure rate matrix are fixed to
the number of links, which is proved to be valid as the original matrix contains those individual links and they were treated as single-link paths. Moreover, as vehicles are designated to serve requests along a route, which, however, may change at each step, the rate matrix needs to be updated accordingly to reload the network traffic. On top of route changing, we reduce the gap between the prediction and the real traffic states by updating the matrix according to actual vehicle path and position. Finally, the output of Traffic Prediction Module is the link travel time and cumulative number of vehicles entering and exiting each link across time horizon $T$.

Appendix B. Generating path departure rate

Algorithm 1. Generating path departure rate

Input: vehicles $V = \{l_i, p_{i_n}\}$, travel time $Y = L \times T$, current time $t$, time step $dt$

Output: departure rate $D = L \times T$

1: for veh $\in V$ do
2:   if $p_{i_n} = \emptyset$ then
3:     continue
4:   end if
5:   Initialisation: the column index $nt$ of $D$, path travel time $\Phi = t, nt_u = \Phi \parallel dt, nt \in T$
6:   for $l \in p_{i_n}$ do
7:     update $\Phi$ and $nt$: $\Phi \leftarrow \Phi + Y(l, nt_u), nt = \Phi \parallel dt$
8:     update $D$: $D(l, nt) \leftarrow D(l, nt) + 1$
9:   end for
10: end for
11: return $D$
12: After running DNL, clean the rate matrix $D$: $D(:, \sim (t + 1) : -1) \leftarrow 0$

As shown in Algorithm 1, the required inputs for generating departure rate matrix include vehicle fleet $V$ with their current location $l_i$ and path $p_{i_n}$, dynamic link travel time matrix $Y = (L, T)$, current time step $t$ and departure rate time step $dt$.

For every vehicle in the network, we first check if its path collection is empty or not. If true, we skip to the next round. Otherwise, we initialise the column index $nt$ of departure rate $D$ as $nt_u = \Phi \parallel dt$. The path travel time $\Phi$ equals to current time $t$ and $dt$ is the time step of departure rate. Note that this $dt$ can be differ from the request batching period $\Delta T$. Besides, $dt$ must satisfy the following inequality $dt \leq \frac{t}{f f t_{\text{min}}}$ where $ff t_{\text{min}}$ is the smallest link free flow time, to prevent vehicles from traversing a link within $dt$. Next, there is an inner loop to iterate through all the links $l_i \in p_{i_n}$. For each link, we calculate the vehicle arriving $nt_u$ and exiting $nt$ time according to the dynamic link travel time $Y$. We then update the $l_{in}$ row of departure rate by adding one to the every element that has a column index between $nt_u$ and $nt$. After walking through every link of every vehicle path, the result is a departure rate matrix that can be readily used for calculating the link travel delay.

Appendix C. Priorities for signalised junction model

As the DNL model incorporates the signalised junction model proposed by Han et al. (2014) and Han and Gayah (2015), signal priority parameters specify how traffic propagates at intersections. Instead of using a binary variable to capture the on-and-off nature of signal status, a more flexible and robust continuous signal control approach is implemented, that is, for any given node, each of its priority parameters specify how traffic propagates at intersections. Instead of using a binary variable to capture the on-and-off nature of signal status, a more flexible and robust continuous signal control approach is implemented, that is, for any given node, each of its priority parameters specify how traffic propagates at intersections. Instead of using a binary variable to capture the on-and-off nature of signal status, a more flexible and robust continuous signal control approach is implemented, that is, for any given node, each of its priority parameters specify how traffic propagates at intersections.

References


