European Journal of Operational Research 317 (2024) 696–717

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/eor

Instance generation tool for on-demand transportation problems

Michell Queiroz [a,*], Flavien Lucas [b], Kenneth Sörensen [a]

[a] Department of Engineering Management, University of Antwerp Operations Research Group (ANT/OR), Prinsstraat 13, 2000, Antwerp, Belgium
[b] CERI Numeric Systems, IMT Nord Europe, Institut Mines-Télécom, Univ. Lille, Centre for Digital Systems, F-59000 Lille, France

ARTICLE INFO

Keywords:
Transportation
Instance generator
On-demand public transport
REQreate

ABSTRACT

We present REQreate, a tool to generate instances for on-demand transportation problems. Such problems consist of optimizing vehicle routes according to passengers' demand for transportation under space and time restrictions (called requests). REQreate is flexible and can be configured to generate instances for a variety of problems types in this problem class. In this paper, we exemplify this with the Dial-a-Ride Problem (DARP) and On-demand Bus Routing Problem (ODBRP). In most of the literature, researchers either test their solution algorithms with instances based on artificial networks or they perform real-life case studies on instances derived from a specific city or region. Furthermore, locations of requests for on-demand transportation problems are mostly randomly chosen according to a uniform distribution, rather than being derived from actual data.

The aim of REQreate is to overcome any shortcomings from synthetic or specific instances. Rather than relying on artificial or limited data, we retrieve real-world street networks from OpenStreetMaps (OSM). To the best of our knowledge, this is the first tool to make use of real-life networks to generate instances for an extensive catalog of existing and upcoming on-demand transportation problems. Additionally, we present a simple method that can be embedded in the instance generation process to produce distinct urban mobility patterns. We perform an analysis with real-life data sets reported by rideshare companies and compare them with properties of synthetic instances generated with REQreate. Another contribution of this work is the introduction of the concept of instance similarity that serves as support to create a set of diverse instances, in addition to properties (size, dynamism, urgency, and geographic dispersion) that can be used to comprehend which characteristics of the problem instances have an impact on the performance quality (or efficiency) of a solution algorithm.

1. Introduction

In recent years there has been a surge of interest in advanced public transportation systems, and the related planning problems are increasingly being studied in the scientific literature. Vansteenwegen et al. (2022) refer to systems that do not operate on fixed routes and timetables as Demand-Responsive Public Bus Systems (DR-PBS). On-demand, dial-a-ride, and flexible transportation services are some other common names. The authors perform a comprehensive survey on planning problems for DR-PBS, which includes 151 papers published since 2000. Characteristics of on-demand transportation problems may vary according to the context, but they mainly consist of optimizing the routes of vehicles according to passengers' demand under space and time restrictions (called requests). This means that passengers can specify locations for departure and arrival within a predefined transportation network. Depending on the problem, departure or arrival locations may be fixed. Users can also specify time windows, which are time slots indicating the period in which their requests must be served. The service can allow deviations of predefined bus routes and timetables according to demand, or design routes and timetables entirely based on incoming requests. The objective function often involves conflicting goals, such as maximizing customer satisfaction and minimizing operating costs.

On-demand transportation problems are generally NP-hard (Garey & Johnson, 1979), and the computational effort to find optimal solutions often grows exponentially with the size of the instance input. Consequently, heuristic procedures are proposed in the literature to find feasible or near-optimal solutions for these problems. However, some of these studies do not test their framework on real scenarios or networks. See for instance Hörl (2017), Lee, Meskar, Nickkar, and Sahebi (2019), Melis and Sörensen (2022), Wang, Correia, and Lin (2019) and Winter, Cats, Correia, and van Arem (2016). Others test their solution approach on only one case study. See for instance Czioska

* Corresponding author.
E-mail addresses: michell.queiroz@uantwerpen.be (M. Queiroz), flavien.lucas@imt-nord-europe.fr (F. Lucas), kenneth.sorensen@uantwerpen.be (K. Sörensen).

https://doi.org/10.1016/j.ejor.2024.03.006
Received 13 May 2022; Accepted 4 March 2024
Available online 11 March 2024

et al. (2019), Koh, Ng, Pan, and Mak (2018) and Qiu, Li, and Haghani (2015). Moreover, Fielbaum and Alonso-Mora (2020) reflect that some of their conclusions may be too specific to their case study. Hyland and Mahmassani (2020) recognize that experiments are only run on one spatial–temporal demand distribution and leave it as open research to apply the same research methodology to different types of traveler demand and determine their impact. Similarly, Lu, Yu, Yang, Pan, and Zou (2016) and Qiu et al. (2015) present future research venues on how different numbers of locations and distributions of requests can impact solutions. These shortcomings have been a motivation for our work to develop a tool that can generate instances for a wide range of (realistic) settings. Therefore, we present REQreate, an instance generation tool for on-demand transportation problems.

Numerical instances are core to evaluate any developed method to solve an optimization problem. First, the instances are essential because they can demonstrate how effective the method is to find good solutions. Second, they can be used to test how sensitive the approach is to specific parameter values. The experiments must demonstrate that the method is robust and presents good results for different types of instances. Third, instances are essential to discover the limitations of a solution method. A method should be tested with easy and difficult instances, and the relationship between the size of the instances and the computation time required to find a good (or even feasible) solution should be established. Fourth, the instances are used to compare the performance of a new method to previously developed methods from the literature. This is particularly challenging for on-demand public transportation problems, as shown in a recent survey by Vansteenwegen et al. (2022). The authors conclude that standard benchmark sets generally do not exist. This lack of structure complicates any comparison between different solution approaches.

REQreate is developed with the capability to generate instances that satisfy the following requirements: size variation, diversity, extensibility, and realism (Vanhoucke & Maenhout, 2007). REQreate allows the generation of instances of varying sizes, which provides researchers with the opportunity to explore and evaluate on-demand transportation solutions across a wide range of operational scales. It is a simple yet important capability, as performing computational experiments with different problem scales offers valuable insights into algorithm performance and system behavior. Furthermore, algorithms should be tested on a diverse benchmark set such that results are not biased. The instance generation process ensures diversification by having different demand patterns, which means that time windows and geographic locations have different distributions. The diversity of a set of instances can be numerically evaluated with a measure called instance similarity, an approach introduced by Leeftink and Hans (2018).

On-demand transportation problems have complex temporal and spatial characteristics. The numerical instances to test the performance of any solution approach should reflect that. In this paper, we also present properties specifically defined for on-demand transportation problems, namely size, dynamism, urgency, and geographic dispersion. *Size* refers to the number of requests. *Dynamism* is the frequency that new events are revealed to the system. *Urgency* is the amount of time available to perform actions regarding a new dynamic request. *Geographic dispersion* quantifies the distribution of locations across the transportation network. Generating instances by varying these properties promotes instance diversity to a next level, as they effectively characterize instances on different aspects. In particular, these properties represent the different challenges of optimizing transportation systems in real-time, which will be valuable in evaluating the performance of any solution algorithm.

REQreate is designed with extensibility in mind. Problem descriptions can change, and the tool allows for easy integration of additional features into the instance generation process. Furthermore, considering that new on-demand transportation problems emerge constantly, and the absence of test instances is an issue, the tool has the potential to generate instances that could be applied to those novel problems by providing the necessary parameters. Problems to which the tool already supports the generation of instances include the Dial-a-Ride Problem (DARP), On-demand Bus Routing Problem (ODBRP), School Bus Routing Problem (SBRP), and other problems within the category of on-demand transportation.

Furthermore, the transportation research community is motivated by real-world problems. Therefore, instances should ideally resemble realistic scenarios as closely as possible. Demand plays a pivotal role when studying on-demand transportation problems. Instances should possess certain properties that allow researchers to accurately model real-world scenarios and evaluate optimization algorithms. First, demand is subject to fluctuation over time, such as time of the day, day of the week, and season of the year. Festivals, concerts, or major sporting events can also significantly impact demand patterns, leading to spikes in requests. Demand is also not uniformly distributed across an urban area, as certain locations such as the city center or popular landmarks tend to attract more requests than others. The stochastic nature of demand should not be overlooked. While demand may follow certain probability distributions, it can still deviate significantly from expected patterns. REQreate can be easily tuned to generate instances featuring transportation requests that showcase different demand patterns. This will produce a variety of scenarios, thus allowing researchers to assess the robustness of their algorithms.

To contribute to the representation of more realistic scenarios in the instance generation process, we present a method that takes into account the density of Points of Interest (POIs) to produce origins and destinations of requests. POIs are locations that attract attention from either residents or tourists, such as shops, offices, parks, and restaurants. We use the method to generate instances with various mobility patterns and validate how the synthetic instances compare with real data from rideshare companies' trips. Our approach provides the flexibility of conducting a study case in areas without readily available on-demand transportation data. Acquiring real-world data is in general challenging, and supply for on-demand transportation creates demand Ronald, Thompson, and Winter (2015). Therefore, REQreate supports the development of on-demand services for previously unexplored regions. In case real data exists, our tool can complement the verification of the strengths and weaknesses of optimization techniques by testing them with different demand patterns. This randomness in request generation also introduces a level of unpredictability that mirrors the stochastic nature of demand in real life. Accordingly, our tool offers a practical solution that enables researchers to conduct extensive experiments without being restricted by data limitations. Furthermore, instances can be easily modified to account for expected disruptions of demand from a newly implemented on-demand transportation system and the change of habits of the population.

In order to further enhance realism, we believe instances for on-demand transportation problems should be based on real-life networks, as they capture the complexities of urban mobility, which may not be replicated by artificial networks. Realistic travel distances may vary significantly from Euclidean distances due to road layouts, and one-way streets. Despite these differences, euclidean distances remain the most frequently used measure in artificial network benchmarks. See for instance Azadeh, Atasoy, Ben-Akiva, Bierlaire, and Maknoon (2022), Braekers and Kovacs (2016), Cordeau (2006) and Cordeau and Laporte (2003). REQreate uses real-life networks from OpenStreetMaps, which accurately reflects the road infrastructure, intersections, and connectivity of urban areas.

To the best of our knowledge, REQreate is the first tool that generates instances for an extensive catalog of existing and upcoming on-demand transportation problems. We use two problem types to demonstrate the potential use of REQreate: the DARP and ODBRP. REQreate can address the issue of the absence of common benchmark instances, which complicates the comparison between solution methods of on-demand transportation problems. Furthermore, we believe REQreate will promote exploratory studies on on-demand transportation

services, helping to answer many pending research questions, such as how far in advance requests should be placed or the influence of the demand level on performance (Vansteenwegen et al., 2022). REQreate is available on GitHub.[1]

The remainder of this paper is organized as follows. Section 2 presents a literature review. Section 3 introduces formal notations for the DARP and ODBRP that is used throughout the paper. Section 4 describes the process of retrieving realistic networks. Section 5 describes instance properties: size, dynamism, urgency, and geographic dispersion. Section 6 introduces the concept of instance similarity. Section 7 outlines the simple method that can be included in the request generation to produce different urban mobility patterns. Furthermore, an analysis between real data and synthetic instances is also performed in Section 7. Ultimately, final remarks are considered in Section 9.

2. Literature review

Throughout the years, tools have been successfully implemented to generate instances for a diversity of optimization problems. Early contributions in the transportation literature include instance generators for the Traveling Salesman Problem (TSP) (Rardin, Tovey, & Pilcher, 1993) and Asymmetric TSP (Cirasella, Johnson, McGeoch, & Zhang, 2001). Furthermore, Pellegrini and Birattari (2005) present a procedure to generate instances for the Vehicle Routing Problem with Stochastic Demand (VRPSD). Later, Liu, Singh, and Ray (2014) express the need to evaluate the performance of algorithms on classes of instances that resemble realistic scenarios. The authors present an instance for the Capacitated Arc Routing Problem (CARP). Their generator is able to control the density, connectedness, degree, and distance distribution of the underlying road network.

Ullrich, Weise, Awasthi, and Lässig (2018) attempt to overcome the limitations of numerous instance generators described in the literature, which are only suitable to specific problems. The authors propose a versatile tool capable of producing random instances for various discrete optimization problems. They illustrate their tool's flexibility by creating instances for the Traveling Salesman Problem (TSP), Maximum Satisfiability Problem (Max-SAT), and a new load allocation problem based on the Resource-Constrained Project Scheduling Problem (RCPSP) with time windows. The tool is designed such that the generated instances can be easily reproduced by sharing configuration files. Ultimately, the authors aspire that this standardized procedure supports the creation of new instances that are harder and/or larger. REQreate exhibits similar versatility as it is easily configurable to generate instances for several on-demand transportation problems, such as the DARP, ODBRP, and SBRP. According to Vansteenwegen et al. (2022), the community would benefit from an instance generator that can be customized to produce realistic instances for different on-demand transportation problems. Such generator should encompass different transportation demand patterns and distinct city layouts. Therefore, in this work, we focus more on realism instead of a purely generic strategy as seen in the work by Ullrich et al. (2018). Other approaches in the literature also have an emphasis on realism. For instance, a methodology to extend routing instances from the literature to render more realistic scenarios with time-dependant travel times for routing problems is proposed in the work by Maggioni, Perboli, and Tadei (2014).

Realism in instances of on-demand transportation problems is closely related to the representation of transportation networks and the modeling of request demand. In the context of real-life transportation networks, OpenStreetMaps has been frequently used as the data source. See for instance Atasoy, Ikeda, Song, and Ben-Akiva (2015), Drakoulis et al. (2018), Navidi, Ronald, and Winter (2018), Pandey, Monteil, Gambella, and Simonetto (2019), Santos and Xavier (2015) and Simonetto, Monteil, and Gambella (2019). Regarding demand, it can be generated artificially or inferred from real data. The former involves generating demand according to clustering methods (Cordeau & Laporte, 2003) or or a uniform distribution of demand (Braekers & Kovacs, 2016). As for real-life data sources, most common examples are smart card data (Gkiotsalitis, Wu, & Cats, 2019; Guo, Guan, & Zhang, 2018; Jäger, Brickwedde, & Lienkamp, 2018; Liu, Sun, Chen, & Ma, 2019; Pei, Lin, Liu, & Ma, 2019; Winter, Cats, Correia, & van Arem, 2018), taxi trips (Alonso-Mora, Samaranayake, Wallar, Frazzoli, & Rus, 2017; Bischoff, Maciejewski, & Nagel, 2017; Ma, Zheng, & Wolfson, 2013; Pandey et al., 2019; Simonetto et al., 2019), bus route surveys (Navidi et al., 2018), trip records from existing DR-PBS (Ronald et al., 2015; Vallée, Oulamara, & Cherif-Khettaf, 2017), and the area's population density (Atasoy et al., 2015). However, even with realistic data available, missing or additional information may be artificially generated. The use of both real and artificial data in the context of generating demand for on-demand transportation problems can be highly beneficial. Real data reflects real-world complexities, while artificial data can help to study hypothetical scenarios, thus allowing researchers to explore the performance of algorithms under different conditions. An overview of instances used for DR-PBS can be seen in the work by Vansteenwegen et al. (2022).

Artificially generated demand can emulate realistic patterns, as researchers have identified regularities in human mobility. The distribution of distances between consecutively visited locations, described as displacement, has been well approximated by truncated Levy flight models (Barbosa et al., 2018). These models involve random walk processes with step lengths following a power law distribution. Despite their success, Levy flight models do not account for geographical heterogeneity, leading to limitations. To address this, authors consider population density (Liu, Kang, Gao, Xiao, & Tian, 2012) and Points of Interest (POIs) (Noulas, Scellato, Lambiotte, Pontil, & Mascolo, 2012) to capture the displacements in an urban environment with higher accuracy. Using this as a foundation, we propose a method that generates transportation requests attempting to approximate real-world scenarios by selecting locations according to the density of Points of Interest (POIs). We also consider the distances between locations, where longer distances between two locations lead to a lower probability of them being chosen as the origin and destination of a request. We validate the observed urban mobility patterns with a real-life data set from a rideshare company.

In addition to the deficiency concerning realistic instances, the literature on algorithms for on-demand public transportation systems is lacking in papers that prioritize offering valuable insights into the characteristics of the modeled systems (Vansteenwegen et al., 2022). Taking this into consideration, we compile four properties aiming at characterizing instances and identifying settings in which algorithms have a better performance or worse performance. These properties are size, dynamism, urgency, and geographic dispersion. Size is quantified by the number of requests and is often used to test the limitations of an approach, as larger instances impose greater challenges (Cordeau & Laporte, 2007; Melis & Sörensen, 2020). Several authors have proposed different definitions of dynamism and urgency and assessed their implications on solution quality. See for instance Borndörfer, Grötschel, Klostermeier, and Küttner (1999), Karami, Vancroonenburg, and Vanden Berghe (2020), Kilby, Prosser, and Shaw (1998), Larsen, Madsen, and Solomon (2002), Lund, Madsen, and Rygaard (1996), Pillac, Gendreau, Guéret, and Medaglia (2013) and Van Lon, Branke, and Holvoet (2018). However, the definitions used in this paper are based on the work by Lon et al. (2016), who are the first authors to propose that urgency and dynamism should be treated separately. The authors define dynamism as the frequency at which new events are introduced to the system, while urgency express the available time to carry out actions in response to a new dynamic request. The authors show that urgency and dynamism have a different influence on solution quality, as dynamism is negatively correlated to the operating costs whereas urgency is positively correlated. The interaction between dynamism and urgency also

[1] https://github.com/michellqueiroz-ua/instance-generator

reveals that scenarios with high urgency and low dynamism are particularly challenging. Geographic dispersion affects route lengths and can be used to analyze performance measures. Reyes, Erera, Savelsbergh, Sahasrabudhe, and O'Neil (2018) characterize geographic dispersion as the distribution of locations within the transportation network, with geographic dispersion increasing as the locations become more spread out. The authors study the Meal Delivery Routing Problem (MDRP) and analyze the impact of geographic dispersion on a performance measure that accounts for the difference between the time the order is announced and its delivery time. REQreate allows instances to be customized to display controlled values of these four properties.

To offer valuable insights, it is also crucial to evaluate algorithm performance across a diverse range of scenarios. This is because similar instances do not contribute with meaningful additional knowledge. Therefore, in an effort to address a gap in the literature on evaluating instance diversity for on-demand transportation problems, we introduce the concept of instance similarity. The approach is based on the concept introduced by Leeftink and Hans (2018). The authors develop a novel instance generation procedure for the Surgery Scheduling Problem and point out the lack of widely used benchmark instance sets in healthcare scheduling. Aiming to maximize the diversity of the instances, the authors measure the similarity between two instances over a concept called instance proximity. The approach is deterministic and compares instances that are generated based on similar characteristics. Each pair of surgeries between two instances is compared, and they are considered proximate if the difference between their expected duration is below a given threshold. Ultimately, the authors generate instances based on real-life and theoretical data. The benchmark set is diversified by selecting a subset of instances in which the maximum proximity between them is minimal.

3. Definitions

In this section, we present basic definitions of on-demand transportation problems for which REQreate is capable of generating instances. REQreate is deliberately designed with a focus on extensibility. Therefore, it is important to highlight that the tool accommodates the incorporation of additional attributes into the instance generation procedure. This flexibility enables the creation of instances for problems types that share similar characteristics to those described in this section.

An instance can be stored as one or multiple files. Usually, instances contain a request database table stored in a Comma Separated Values (CSV) file, where each row represents a passenger's request for transportation and the columns are attributes. An attribute is a piece of information that determines a specific property for each passenger's request. Passengers must submit requests to the operator to use an *on-demand* system.

Passengers are picked up and dropped off at specified locations within a predefined network. Networks exist in the real world and their foundations come from graph theory. Graphs are mathematical abstractions of networks and represent their elements, namely nodes or vertices, and eventual connections between those elements, often referred to as edges or arcs (West et al., 2001). We are interested in spatial networks, in which vertices depict geographic locations and edges are road segments between them (O'Sullivan, 2014). Pick-up and drop-off actions can be performed on a predefined set of potential stops (*stop-based*) or any location within the given network (*door-to-door*). *Many-to-many* systems pick-up and drop-off passengers between any two locations on the network. Otherwise, if the trips are performed from a shared origin or to a common destination, it is referred to as *many-to-one* or *feeder lines*.

Requests also include time windows. The earliest and latest times a passenger can be picked up are referred to *earliest departure* and *latest departure*, respectively. Furthermore, *earliest arrival* and *latest arrival* are the earliest and latest times a passenger can be dropped-off. On-demand transportation problems may include all or any combination of the previously mentioned time windows. The moment a request is announced to the system is referred to as *time stamp*. The difference between the time stamp of a request and the latest departure time is called *reaction time*. The reaction time can also be seen as the available time to perform actions regarding the assignment, optimization, and start service of a request (the actual process of picking up a passenger). The period in which the service is executed is presented as *planning horizon*. It consists of the whole day or some specific peak or off-peak hours.

A system with predefined bus routes and timetables that allows deviations according to demand is considered *semi-flexible*. Alternatively, if the routes and timetables are entirely based on data received from requests, then the system is *fully flexible*. Objective functions may consist of minimizing the passenger travel time (*passengers' perspective*), operational costs (*operator's perspective*), or both at the same time in a *multi-objective* approach.

On-demand transportation problems can have *static* and *dynamic* variants. All requests are known beforehand in static cases. In the dynamic variants, requests are announced to the system during the day and the vehicle routes must be adjusted in real-time.

We now describe two problems for which the tool can generate instances. The on-demand transportation problems under consideration in Sections 3.1 and 3.2 are the DARP and the ODBRP, respectively. We briefly define each problem and formally present the notations for the attributes used throughout this paper. REQreate takes as input a configuration file in JSON format in which the attributes are specified. Appendix A introduces how to write those configuration files.

3.1. DARP

In this paper, we consider the dynamic DARP. Instances for this problem consist of a set of requests R and a travel time matrix. Each request $r_i \in R$ represents one or more passengers and contains a couple (o_{r_i}, d_{r_i}), which expresses the origin and destination location, respectively for request r_i. The set of origins and destinations of the users are denoted as $O = \bigcup_{r_i \in R} o_{r_i}$ and $D = \bigcup_{r_i \in R} d_{r_i}$, respectively. Each request r_i has a time stamp ts_{r_i}. Additionally, every request includes two time windows, i.e., the earliest and latest times for pick-up and drop-off. The earliest and latest departure times for request r_i are denoted $e^u_{r_i}$ and $l^u_{r_i}$, respectively. Meanwhile, the earliest and latest arrival times for request r_i are denoted $e^o_{r_i}$ and $l^o_{r_i}$, respectively. Some requests may include requirements for vehicles that support wheelchairs. The fleet of vehicles is located in a set W of designated locations named depots. The travel time matrix consists of the travel times between all pairs of locations among the following sets: depot(s) (W), origins (O), and destinations (D). Let L be the complete set of locations, i.e., $L = W \cup O \cup D$. The planning period is expressed as $T_e = [ts_{min}, ts_{max}]$, where ts_{min} and ts_{max} denote, respectively, the earliest time a request can be picked up and latest time it can be dropped off. T_e also indicates the period dynamic requests may be announced to the system. Note that how far in advance vehicles should leave the depot can be explored during experimental analysis.

3.2. ODBRP

Instances for the On-Demand Bus Routing Problem consist of a set of requests R and a travel time matrix. Each request $r_i \in R$ represents one or more passengers and contains their initial origin and destination locations, as well as a set of potential stops $S^u_{r_i}$ that request r_i can be assigned for pick-up, and a set of potential stops $S^o_{r_i}$ that request r_i can be assigned for drop-off. During the service planning, the system must decide on each passenger's pick-up and drop-off location. This procedure is referred to as *the bus stop assignment*. The maximum walking time, representing the passenger's willingness to walk, is denoted by

u_{r_i}. The stations in $S^u_{r_i}$ and $S^o_{r_i}$ have a distance of at most u_{r_i} from the origin (o_{r_i}) and destination (d_{r_i}) locations, respectively. Furthermore, each request has a time window consisting of an earliest departure time ($e^u_{r_i}$) and a latest arrival time ($l^o_{r_i}$). Note that the decision to define a latest departure time ($l^u_{r_i}$) is left to the system. Therefore, the journey is feasible as long as it is completed before $l^o_{r_i}$. L is defined as the complete set of bus station locations. The planning period is expressed as $T_e = [ts_{min}, ts_{max}]$, and is previously described in Section 3.1. The travel time matrix consists of the estimated travel times between all the bus stations in L.

4. Retrieving realistic transportation networks

In this section, we highlight the range of information that REQreate can extract and utilize, supporting the creation of realistic instances. This encompasses various types of transportation networks, including driving and walking networks, as well as data on bus stops and fixed lines. Furthermore, we elucidate the process by which travel times are computed, employing parameters such as speed limits to ensure better accuracy and realism.

In the literature on on-demand public transport problems, researchers frequently test their algorithms with instances based on synthetic networks or they perform real-life case studies on a specific city or region. For the first case, a shortcoming is that the performance of these algorithms is evaluated in networks with non-realistic patterns. For the second case, it is unlikely to guarantee the robustness of the proposed method, as the results are probably overfitted to the specific case study. Aiming to overcome these obstacles and provide an easy way to create instances with realistic networks, we use the OSMnx package (Boeing, 2017) to retrieve and analyze real-world street networks from OpenStreetMaps (OSM). By simply providing a string (i.e., the area's name), OSMnx downloads information on the area's boundaries and creates a graph. Further computations done by REQreate (e.g., distance or travel time matrix between nodes) are done using information obtained with those graphs. On the retrieved network, requests for transportation are subsequently generated.

The street networks built with OSMnx are primal, i.e., nodes are intersections, and arcs represent street segments. They are also non-planar, as generally, street networks cannot be represented only in two dimensions because of structures such as bridges, tunnels, and overpasses. The arcs have weights associated with them, which represent the distance between two nodes. Another important characteristic of the networks is that they are multidigraphs with self-loops, as they are directed and have more than one arc between the same two nodes. An arc that connects a single node to itself is called a self-loop. Multidigraphs are particularly useful for capturing temporal dynamics within a network. Interactions occurring at different times can be represented, as multidigraphs allow multiple arcs between the same pair of nodes. Routes can be optimized considering variations in travel times throughout the day.

It is possible to download different network types, representing different travel options between nodes. Some examples are drivable public streets ($drive$), streets and paths that pedestrians can use ($walk$), streets and paths that cyclists can use ($bike$), and others. We work with both drive and walk networks because, in reality, they have different aspects and distances between nodes. An example is shown in Fig. 1, where the driving network (Fig. 1(a)) and walking network (Fig. 1(b)) of a neighborhood ("South Lawndale") in "Chicago, Illinois" are illustrated. Note how they are significantly different, as the walking network has more nodes and arcs when compared to the driving network.

Bus stations within the boundaries of the given area are also obtained using a built-in function of OSMnx. Several transport problems can use such information, such as the ODBRP, to assign passengers to pick-up and drop-off stations. After retrieving the bus stations, we perform a verification process to delete repeated entries and stations that are isolated in the network, i.e., unreachable stations. Bus stations on the driving network of "South Lawndale, Chicago, Illinois" are represented in Fig. 2. Pairing locations between networks is possible by converting geographic coordinates to the nearest node on each respective network. This can result in useful information when a passenger is assigned to be picked up by a bus. Therefore, there is a requirement to determine where the bus should stop (driving network) and where the passenger needs to wait (walking network).

Certain problems may take advantage of the existing fixed route network, which is the case in the Integrated Dial-a-Ride Problem (IDARP) (Häll, Andersson, Lundgren, & Värbrand, 2009). However, OSM often does not include fixed line routes, frequencies, or estimated travel times. Therefore, the collection of data presented by Kujala, Weckström, Darst, Mladenović, and Saramäki (2018) assists in obtaining such information. The authors published a curated collection of public transport (PT) network data sets for 25 cities, which ultimately provides a testbed for developing tools for PT network studies and PT routing algorithms, and supports an analysis of how PT is organized across the globe.

The next step is to compute vehicle travel times on the driving network, which requires a speed value. We assume the driving speed to be fixed, which means that the travel times are not dependent on the hour of the day. The NetworkX package (Hagberg, Swart, & S. Chult, 2008) allows to access and manage information associated with each node and arc of the network. Let d_{uv} and ms_{uv} represent the distance and maximum speed, respectively, between nodes u and v. The travel times are then computed and stored, for each arc in the network, according to the formula $tt_{uv} = \alpha \cdot \frac{d_{uv}}{ms_{uv}}$. Parameter α is called "the speed factor" and is a value in the interval (0,1] which is determined by the user. It is used to calculate a realistic speed for each arc, which is usually far from the maximum. The definition of α is demonstrated in Appendix A. It is also possible to replace ms_{uv} by an equal speed value to all arcs.

Travel times are important to measure, for example, to calculate the cost of driving a passenger from their origin to their destination in a door-to-door service. Another example is to assess the feasibility of assigning passengers to nearby bus stations. Therefore, we build the travel time matrix using Dijkstra's algorithm that is a built-in function of the NetworkX package. It computes the shortest paths between the nodes of the network using travel times as the weight.

5. Instance properties

In this section, we define several properties of instances for a diversity of on-demand transportation problems, including the ones described in Section 3. The properties are size, dynamism, urgency, and geographic dispersion. Size relates to the number of requests. The frequency in which new events are revealed to the system is measured by dynamism. Urgency represents the available time to handle new dynamic requests, and the distribution of locations across the transportation network is quantified by geographic dispersion. By introducing these properties into the instance generation process, our goal is to enhance instance diversity, as these properties characterize different aspects of the instances. Moreover, they also play a role in representing the challenges of optimizing transportation systems in real time. They serve to identify scenarios where an algorithm's performance is more effective or less effective. We formally describe these properties and discuss their impact in realistic scenarios by examining their usage in the existing literature. The terminology in the following definitions is aligned with the notations of attributes presented in Section 3. However, they can be extended to evaluate the properties of other problems that display similar characteristics. We report values of the properties discussed in this section for instances from literature in Appendix F.

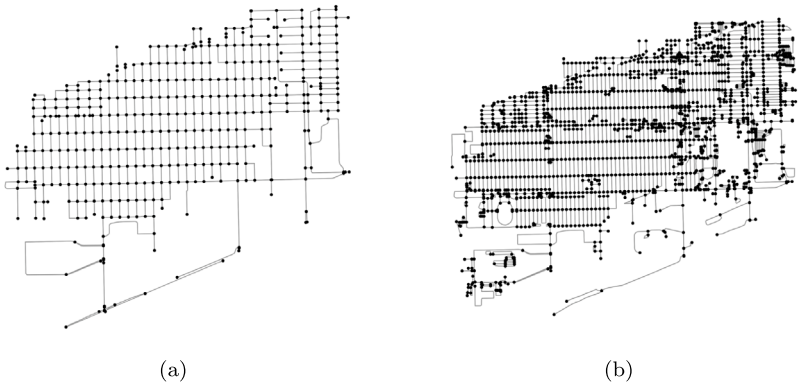


Fig. 1. The driving network 1(a) and walking network 1(b) of "South Lawndale, Chicago, Illinois".

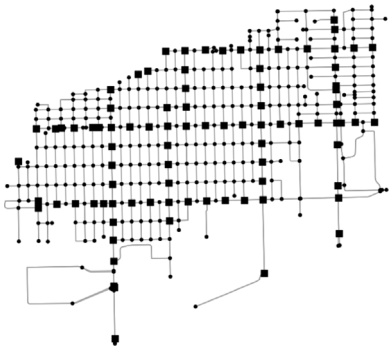


Fig. 2. Location of bus stations in the driving network of "South Lawndale, Chicago, Illinois" represented as black squares..

5.1. Size

The size of an instance is a measure that depends on the problem under consideration. For example, the number of cities determines the size of an instance for the TSP. In previous works that address DARP, ODBRP, and similar problems, the instance size is usually expressed as the number of requests (Cordeau & Laporte, 2007; Melis & Sörensen, 2020). The size of an instance is an essential aspect since larger instances frequently impose a greater challenge during experiments. Especially for exact solution techniques, the number of variables increases and the time required to obtain an optimal or feasible solution grows substantially. When using heuristics, the computational time also rises, and the solution quality often degrades as instances grow in size. In case a paper introduces an exact solution algorithm, one should test the limits of the approach by reporting instances that cannot be solved to optimality within a particular timeframe.

5.2. Dynamism

Dynamism captures the periodicity aspect of new information revealed during the planning period. In accordance with the definition provided by Lon et al. (2016), we treat the degree of dynamism as the continuity of *change*. Every time an event occurs that introduces new information to the problem, it can be viewed as a *change*. We consider any event that triggers re-optimization, such as transportation requests, congestion, vehicle breakdowns, and delays. Therefore, dynamism is measured as the frequency that new events are revealed to the system. In a highly dynamic scenario changes occur continuously. On the contrary, if changes occur occasionally, the instance can be considered less dynamic. Different scenarios with various levels of dynamism are represented in Fig. 3. The scenario in Fig. 3(a) changes continuously in evenly timed intervals, therefore, it represents a very dynamic scenario. The dynamism levels decreases in Figs. 3(b) to 3(e). In Fig. 3(f) all events are announced at the same time which results in a case with no dynamism.

Lund et al. (1996) present an earlier definition where dynamism is the proportion between dynamic and static requests. Since the relative timing distribution of arrivals is not accounted for in the measure, scenarios where no previous information is known before the planning period cannot be distinguished. In Larsen et al. (2002), a request is reportedly more dynamic when there is a tight interval between the time stamp when the request is made and the latest possible time to begin servicing the request. However, the authors also incorporate the urgency feature in their definition (see also Section 5.3), which makes it difficult to draw conclusions regarding the individual impact of these two concepts on the solution quality. More recently, Lon et al. (2016) provide separate measures for dynamism and urgency that addresses these limitations, which will be presented in the following paragraphs.

To present the dynamism measure, we start by introducing notations similar to Lon et al. (2016). Consider $R_d = \{r_1, r_2, \ldots, r_{|R_d|}\}$ to be the set of events introduced after the start of the planning period, and sorted by non-decreasing order when they occur, i.e., $ts_{r_j} \geq ts_{r_i},\ \forall j > i$. Let $\Delta = \{\delta_1, \delta_2, \ldots, \delta_{|R_d|-1}\} = \{ts_{r_j} - ts_{r_i} |\ j = i+1\ \forall r_i, r_j \in R_d\}$ be the set of interarrival times. Cases with 100% dynamism are characterized by events being presented in evenly timed intervals (see Fig. 3(a)). Then the perfect interarrival time can be computed as $\theta = \frac{ts_{max} - ts_{min}}{|R_d|}$. For each interarrival time $\delta_k \in \Delta$ it is now possible to compute its deviation from the 100% dynamism case, which is defined as Eq. (1)

$$\sigma_k = \begin{cases} \theta - \delta_k & \text{if } (k = 1) \ \wedge \ (\delta_k < \theta) \\ \theta - \delta_k + \frac{\theta - \delta_k}{\theta} \cdot \sigma_{k-1} & \text{if } (k > 1) \ \wedge \ (\delta_k < \theta) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The deviation of an entire scenario is then given by the summation $\lambda = \sum_{k=1}^{|R_d|-1} \sigma_k$. Consider bursts as announcements that occur with interarrival times shorter than δ. These bursts are penalized in σ_k by adding a proportion of the deviation from the previous interarrival time (i.e, by adding $\frac{\theta - \delta_k}{\theta} . \sigma_{k-1}$). To put it briefly, the penalty term is used to identify and measure the size of bursts and account for their contribution. Moreover, consider $\eta = \sum_{k=1}^{|R_d|-1} \overline{\sigma}_k$, where

$$\overline{\sigma}_k = \theta + \begin{cases} \frac{\theta - \delta_k}{\theta} \cdot \sigma_{k-1} & \text{if } (k > 1) \ \wedge \ (\delta_k < \theta) \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The factor η theoretically captures the maximum deviation for a scenario (0% dynamism) and normalizes the deviation from the 100% case. Therefore, the dynamism of an instance is measured by

$$\rho = 1 - \frac{deviation}{max \quad deviation} = 1 - \frac{\lambda}{\eta} \tag{3}$$

These definitions from Lon et al. (2016) are used to compute the dynamism for each scenario shown in Fig. 3. Each scenario consists

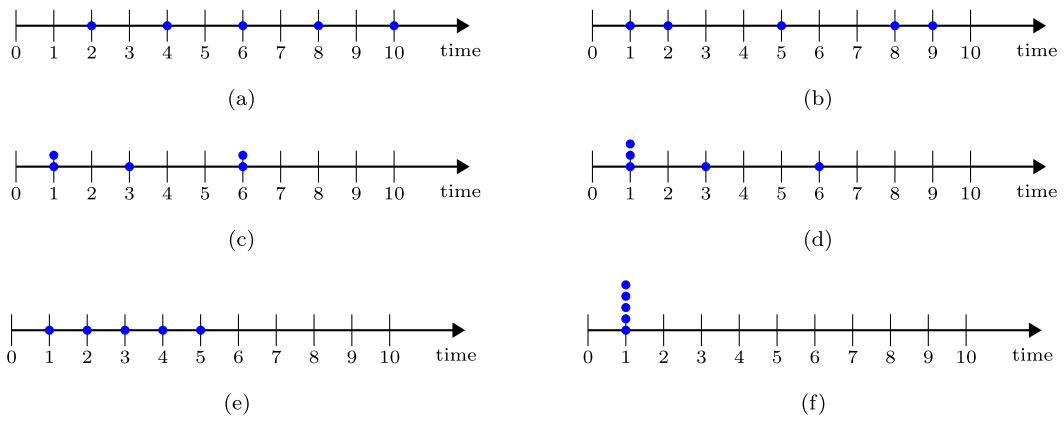


Fig. 3. Depiction of different levels of dynamism. Colored dots represent the time one event was announced to the system. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Table 1
Sets and values necessary to determine dynamism for scenarios shown in Fig. 3.

Scenario	Δ	σ	λ	$\overline{\sigma}$	η	ρ
Fig. 3(a)	{2.00, 2.00, 2.00, 2.00}	{0.00, 0.00, 0.00, 0.00}	0.00	{2.00, 2.00, 2.00, 2.00}	8.00	1.00
Fig. 3(b)	{1.00, 3.00, 3.00, 1.00}	{1.00, 0.00, 0.00, 1.00}	2.00	{2.00, 2.00, 2.00, 2.00}	8.00	0.75
Fig. 3(c)	{0.00, 2.00, 3.00, 0.00}	{2.00, 0.00, 0.00, 2.00}	4.00	{2.00, 2.00, 2.00, 4.00}	10.00	0.60
Fig. 3(d)	{0.00, 0.00, 2.00, 3.00}	{2.00, 4.00, 0.00, 0.00}	6.00	{2.00, 4.00, 2.00, 2.00}	10.00	0.40
Fig. 3(e)	{1.00, 1.00, 1.00, 1.00}	{1.00, 1.50, 1.75, 1.87}	6.12	{2.00, 2.50, 2.75, 2.87}	10.12	0.39
Fig. 3(f)	{0.00, 0.00, 0.00, 0.00}	{2.00, 4.00, 6.00, 8.00}	20.00	{2.00, 4.00, 6.00, 8.00}	20.00	0.00

of five events with $ts_{max} = 10$ and $ts_{min} = 0$. The intermediate results to calculate ρ are presented in Table 1. The perfect interarrival time is given by $\theta = 10/5 = 2$. Note that Fig. 3a represents the 100% dynamism case, since all interarrival times are equal to θ. Furthermore, dynamism levels decrease in Figs. 3(b) to 3(e), as they exhibit events that are revealed close to one another ($\delta_k < \theta$), which is penalized on the above equations. Ultimately, the five events depicted in Fig. 3(f) are announced simultaneously, so the dynamism is zero in such circumstances.

Note that it is impossible to distinguish between the dynamism of the scenarios in Figs. 3(c) and 3(d) by just looking at the interarrival times of requests. Both scenarios have two interarrival times of 0.00, one of 2.00, and one of 3.00. This example demonstrates the importance of the penalty factor in Eq. (1). Besides computing the deviation from the perfect interarrival time (i.e., $\theta - \delta_k$), the penalty factor makes a distinction between the two scenarios by also taking into consideration the prior interarrival time (i.e., $\frac{\theta-\delta_k}{\theta} \cdot \sigma_{k-1}$). The scenario in Fig. 3(e) is less dynamic than the other scenarios since all interarrival times are shorter than two time units.

Dubois (2022) studies a dynamic version of the Vehicle Routing Problem (VRP) and considers different dynamic events that trigger re-optimization during the operations (such as dynamic requests and service delays). Numerical results demonstrate that their proposed heuristic results in better quality solutions when there is high dynamism, but it becomes unreliable otherwise (i.e., no feasible solutions are found in one hour of computational time). The Same-Day Pickup and Delivery Problem (SD-PDP) is considered by Engelen (2018). Customers request goods that must be transported between pickup and delivery locations. The authors perform simulations and two settings of time windows are compared. In the first setting, pick-up and delivery locations have the same earliest service start time and a delivery deadline is established. In the second setting, independent time windows for both locations are modeled. A higher number of visits to locations is observed when dynamism is low and the scenario is subject to deadlines. Moreover, dynamism is shown to have a minor influence on independent time windows scenarios.

5.3. Urgency

Urgency is a feature that indicates the length of the available time to perform actions regarding a new dynamic request. Lon et al. (2016), who first proposed the properties of dynamism and urgency, perform computational experiments to evaluate the impact of urgency on route quality for the dynamic Pickup and Delivery Problem with Time Windows (PDPTW). In the PDPTW, a fleet of vehicles is used to transport items from pickup to delivery locations according to customer requests. The objective is to minimize route costs, which are inversely proportional to route quality. In the authors' definition, the urgency f_{r_i} of a single request r_i is computed as the interval between the time stamp and the latest pick-up time of r_i, i.e., $f_{r_i} = l^u_{r_i} - ts_{r_i}$. They also refer to this interval as the reaction time. Let $\chi = f_{r1}, f_{r2}, \ldots, f_{r|R_d|}$ be the set of urgency values of an instance. Consider f_{r_i} and f_{r_j} as the urgency of requests r_i and r_j, respectively. Supposing $f_{r_i} < f_{r_j}$, r_i is said to be more urgent than r_j. Accordingly, r_j is considered to be less urgent.

The mean ($\overline{\chi}$) and standard deviation (χ_s) of χ denote the urgency of the corresponding instance (Lon et al., 2016). They are computed as follows: $\overline{\chi} = \frac{\sum_{r_i \in R_d} f_{r_i}}{|R_d|}$ and $\chi_s = \sqrt{\frac{\sum_{r_i \in R_d} (f_{r_i} - \overline{\chi})^2}{|R_d|}}$. The standard deviation is part of the urgency measure because the mean is not a very distinctive feature by itself since scenarios with the same mean and different standard deviations may contain different numbers of higher and lower urgency values. Note that urgency is expressed in time units. The concept of urgency is depicted in Fig. 4, which represents a scenario with the following set of urgency values: $\chi = \{3, 1\}$. As a result, the urgency of this instance is: $\overline{\chi} = \frac{3+1}{2} = 2$ and $\chi_s = \sqrt{\frac{1+1}{2}} = 1$.

The instances generated by Lon et al. (2016) are grouped by sets that share similar characteristics, apart from values of dynamism and urgency. Based on numerical results, the authors conclude that these features have different influences on operating costs, as dynamism has a reasonably small effect on operating costs and the effect is negatively correlated, whereas urgency is positively correlated to the operating costs. Thus, the hypothesis that the route quality decreases when the urgency of a scenario increases is strongly supported. The interaction between dynamism and urgency shows that very urgent scenarios with

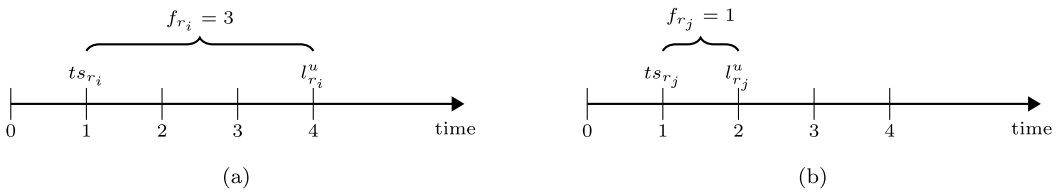


Fig. 4. Illustration of different urgency scenarios. Figs. 4(a) and 4(b) exhibit the reaction times for requests r_i and r_j, respectively. Note that $f_{rj} < f_{ri}$. Therefore, r_j is more urgent than r_i.

low dynamism are the hardest to solve. Possibly because of big bursts of urgent requests, which may be bigger than the available fleet.

Karami et al. (2020) assess the performance of a heuristic for the PDPTW and show how it is impacted by frequent request arrivals. Instances with low dynamism and high urgency levels are the most challenging to solve for their proposed algorithm. The authors determine the length of re-optimization steps of the algorithm. Experiments show that dividing the time horizon according to the urgency length of the most urgent request achieves the highest quality solution. Van Lon et al. (2018) compare the performance of a tabu search and genetic programming algorithms for a dynamic Pickup and Delivery Problem (PDP). The genetic programming algorithm is shown to outperform the tabu search, especially in more urgent and larger scenarios, due to its shorter computational cost.

5.4. Geographic dispersion

Geographic dispersion is a criterion that expresses the spread of important locations across the network. It can be expected that a large geographic dispersion, and thus a considerable distance between important locations in the network, will give rise to longer routes. The definition presented in this paper is based on the approach described by Reyes et al. (2018). The authors study the Meal Delivery Routing Problem (MDRP) which consists of building routes for couriers to deliver meals requested by customers. Couriers pick up orders as soon as they are available from the restaurant and deliver them to the designated customer's location. The objective function may include multiple metrics, such as courier compensation, the difference between the announcement of an order and its delivery (click-to-door), and the interval between the release time of an order from the restaurant to the arrival time at the drop-off location (ready-to-door).

Geographic dispersion is originally defined by Reyes et al. (2018, as the sum of two terms: 1) average travel time from restaurants to delivery locations; and 2) the average travel time between every pair of restaurants. Since the DARP and ODBRP differ from the MDRP, we modify this definition to establish it as more general and suitable for on-demand transportation problems. The first term can be translated as the travel times between origins (restaurants) and destinations (delivery locations). Meanwhile, the second term incorporates detours between locations, as couriers need to travel between restaurants to pick up bundled orders. First, we sum the estimated direct travel times for each request and calculate their average as $\mu = \frac{\sum_{r_i \in R} tt_{o_{r_i} d_{r_i}}}{|R|}$, where $tt_{o_{r_i} d_{r_i}}$ present the estimated direct travel time between origin and destination for the DARP. At the same time, $tt_{o_{r_i} d_{r_i}}$ denotes the estimated average travel time between stations surrounding the origin and destination for the ODBRP.

Second, we account for detours since requests can be served simultaneously by the vehicles. We incorporate the average travel time from the origin and destination to the nearest neighbors' locations of requests, namely ω. First, consider $L^o_{r_i}$ and $L^d_{r_i}$ to be subsets of locations that can potentially be served after the origin and destination of r_i, respectively. Consider a time threshold th_s indicating that the time windows are close enough to coincide. Specifically, $L^o_{r_i} = \{o_{r_j} \mid \|e^u_{r_i} - e^u_{r_j}\| < th_s \ \forall r_j \in R \setminus r_i\} \cup \{d_{r_j} \mid \|e^u_{r_i} - l^o_{r_j}\| < th_s \ \forall r_j \in R \setminus r_i\}$ and $L^d_{r_i} = \{o_{r_j} \mid \|l^o_{r_i} - e^u_{r_j}\| < th_s \ \forall r_j \in R \setminus r_i\} \cup \{d_{r_j} \mid \|l^o_{r_i} - l^o_{r_j}\| < th_s \ \forall r_j \in R \setminus r_i\}$.

Table 2
Sets and values necessary to determine geographic dispersion for each instance shown in Fig. 5.

Request	N^o_r	tn^o_r	N^d_r	tn^d_r
r_m	$\{\varnothing\}$	0.00	$\{o_{r_i}, o_{r_k}\}$	17.00
r_i	$\{o_{r_j}, o_{r_k}\}$	10.00	$\{d_{r_j}, d_{r_k}\}$	13.00
r_j	$\{o_{r_i}, o_{r_k}\}$	13.00	$\{d_{r_i}, d_{r_k}\}$	17.00
r_k	$\{o_{r_i}, d_{r_m}\}$	12.00	$\{d_{r_i}, d_{r_j}\}$	22.00

The at most n nearest locations in $L^o_{r_i}$ and $L^d_{r_i}$ are denoted by $N^o_{r_i}$ and $N^d_{r_i}$, respectively. This means $N^o_{r_i} \subset L^o_{r_i}$ and $N^d_{r_i} \subset L^d_{r_i}$. Let $tn^o_{r_i}$ and $tn^d_{r_i}$ correspond to the average travel times from the origin and destination of request r_i to locations in $N^o_{r_i}$ and $N^d_{r_i}$, such that

$$tn^o_{r_i} = \begin{cases} \frac{\sum_{v \in N^o_{r_i}} tt_{o_{r_i} v}}{|N^o_{r_i}|}, & \text{if } |N^o_{r_i}| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$tn^d_{r_i} = \begin{cases} \frac{\sum_{v \in N^d_{r_i}} tt_{d_{r_i} v}}{|N^d_{r_i}|}, & \text{if } |N^d_{r_i}| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

Accordingly, ω is used to approximate the duration of detours and can be computed as $\omega = \frac{\sum_{r_i \in R} tn^o_{r_i} + tn^d_{r_i}}{2 \cdot |R|}$. The definition of geographic dispersion (gd) is then calculated as:

$$gd = \mu + \omega \tag{6}$$

We exemplify the geographic dispersion of an instance with four requests using Fig. 5. Origin and destination nodes of a request r_n are denoted as o_{r_n} and d_{r_n}, respectively. Dashed lines represent a connection between two distinct locations. Travel times (T) are depicted next to the dashed lines. Finally, $e^u_{r_n}$ and $l^o_{r_n}$ are the earliest departure and latest arrival times ,respectively of a request r_n. The instance consists of 4 requests $R = \{r_i, r_j, r_k, r_m\}$. Assume $th_s = 10$ and $n = 2$. These values are measured in fictitious units of time. First, according to Fig. 5(a), the average estimated direct travel time equals $\mu = \frac{102+84+85+89}{4} = 90$. The sets and values necessary to determine geographic dispersion are reported in Table 2.

In this table, the first column specifies the request. The second and third columns indicate the n nearest locations to the request's origin and the average travel time between the origin to these locations, respectively. The fourth and fifth columns report the n nearest locations to the request's destination and the average travel time between the destination to these locations, respectively. Now, it has become possible to calculate ω as $\omega = 104/8 = 13$. Finally, the geometric dispersion for this instance is given by $gd = 90 + 13 = 103$.

Reyes et al. (2018) propose an algorithm for the MDRP and assess the impact of dynamism, urgency, and geographic dispersion on performance measures. The authors observe that higher dynamism increases route costs, as it is harder to combine orders in a single route. An increase of 20 min in geographic dispersion is translated into a 2.5 min increase in click-to-door time. Finally, with an additional minute of reaction time, click-to-door times could be reduced to around 100 s on average.

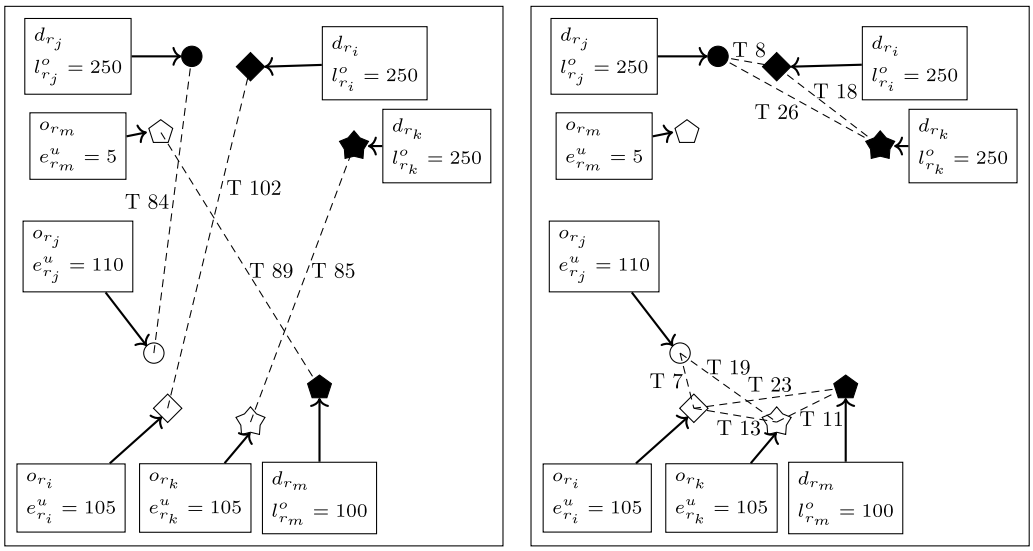


Fig. 5. Illustration of an instance with four requests to support computations for geographic dispersion. White shapes are origin nodes and black shapes are destination nodes.

6. Instance similarity

Algorithms should be tested on a diverse set of instances. The direct rationality is that similar instances are unlikely to contribute meaningful additional knowledge. Therefore, in this section, we present a concept of instance similarity based on the approach introduced by Leeftink and Hans (2018). REQreate enables to measure the similarity of instances, facilitating the eventual selection of a diverse set of instances. We report values of instance similarity for instances from literature in Appendix F.

Leeftink and Hans (2018) study surgery scheduling and propose a benchmark set. Instances for surgery scheduling are based on a case mix, which is a collection of surgery types. The frequency, duration distribution, and expected duration for each surgery type are given. The authors measure the similarity between instances that are generated based on the same case mix. They define that two surgeries are considered proximal if the difference in their expected duration is below a certain threshold. Surgeries can be proximal to more than one other surgery from the opposing instance. Therefore, they find the maximum matching of proximal surgeries. For an undirected graph, a set of edges that do not share common vertices is considered a matching. The maximum matching is a matching of maximum cardinality. So, given the maximum matching, the authors compute the similarity between two instances, which is the total workload of proximal surgeries divided by the total workload of all surgeries in both instances.

We consider that instances for on-demand transportation problems consist of a set of requests distributed over a network during a period. So, in order to assess the similarity of two instances I and J of the same size, we must compare the requests in I and J. To measure the proximity between any two distinct requests r_i and r_j, where $r_i \in I$ and $r_j \in J$, we compare the proximity among multiple elements of a request, i.e., locations, time stamps, and earliest departure times.

First, we compute and sum the travel times between the origins and destinations of r_i and r_j as $\phi_{r_i r_j} = tt_{o_{r_i} o_{r_j}} + tt_{d_{r_i} d_{r_j}}$. Requests r_i and r_j are ϕ-proximate if $\phi_{r_i r_j}$ is below a given threshold, i.e., $\phi_{r_i r_j} < th_{tt}$. On the assumption that two requests are ϕ-proximate, we also compare if they have a similar time stamp and earliest departure times. Let $\tau_{r_i r_j} = \|ts_{r_i} - ts_{r_j}\|$ and $\vartheta_{r_i r_j} = \|e^u_{r_i} - e^u_{r_j}\|$ be the difference of time stamps and earliest departure times between r_i and r_j, respectively. Similarly, to measuring proximity between location pairs, two requests are τ-proximate and ϑ-proximate if $\tau_{r_i r_j}$ and $\vartheta_{r_i r_j}$ are lower than specified thresholds, i.e., $\tau_{r_i r_j} < th_{ts}$ and $\vartheta_{r_i r_j} < th_e$. Combining the definitions given so far, the similarity level between two requests r_i and r_j (denoted by $\xi_{r_i r_j}$) becomes

$$\xi_{r_i r_j} = \begin{cases} 1.00 & \text{if } (\phi_{r_i r_j} < th_{tt}) \wedge (\tau_{r_i r_j} < th_{ts}) \wedge (\vartheta_{r_i r_j} < th_e) \\ 0.75 & \text{if } (\phi_{r_i r_j} < th_{tt}) \wedge ((\tau_{r_i r_j} < th_{ts}) \oplus (\vartheta_{r_i r_j} < th_e)) \\ 0.50 & \text{if } (\phi_{r_i r_j} < th_{tt}) \wedge (\tau_{r_i r_j} \geq th_{ts}) \wedge (\vartheta_{r_i r_j} \geq th_e) \\ 0.00 & \text{otherwise} \end{cases} \tag{7}$$

Note that the concept of similarity assesses the requests individually, considering their respective locations and time windows. Since there are no distinct values of dynamism, urgency, or geographic dispersion for each request, we have omitted these properties from the similarity measure. Moreover, these properties inherently improve instance diversity, as they contribute to variety in the generated instances. Dynamism and urgency directly impact time windows, while geographic dispersion influences locations.

We illustrate the concept of similarity with Fig. 6, where we depict different requests, and we analyze their level of similarity regarding r_i. The notations are the same as the ones introduced in Fig. 5. Consider the threshold levels to be $th_{tt} = 20$, $th_{ts} = 10$, and $th_e = 10$.

The values necessary to determine the similarity between the two sets of requests are reported in Table 3. In this table, the first column specifies for which pair of requests the similarity is computed. The marks ✓ and ✗ beside the numbers in the following three columns indicate if the values are below or above their corresponding thresholds respectively. The last column reports the level of similarity between the pair of requests. The first two rows indicate the requests shown in Fig. 6(a). Since all values for requests r_i and r_j are below the given thresholds, we get $\xi_{r_i r_j} = 1$. Meanwhile, the difference of earliest departure times (ϑ) is the only value that exceeds the threshold for requests r_i and r_k. Therefore, $\xi_{r_i r_k} = 0.75$. The final two rows indicate the requests represented in Fig. 6(b). Note that requests r_i and r_m are only ϕ−proximate, so $\xi_{r_i r_m} = 0.50$. The sum of the distances between the locations of requests r_i and r_q exceeds th_{tt}. For this reason $\xi_{r_i r_q} = 0$.

We introduce the following definition to help determine the similarity between two instances I and J: consider a bipartite graph $G_s = (V_s, E_s)$, where the set of vertices V_s represent the requests, and there is a non-negative weight w_e associated with each edge $e \in E_s$. The weights represent the similarity level between the two requests of opposite instances. This means that there is an edge $e = (r_i, r_j)$ and its associated weight is $w_e = \xi_{r_i r_j}$, $\forall r_i \in I$ and $r_j \in I$. Similarity is computed for every pair of requests between the two instances, and as

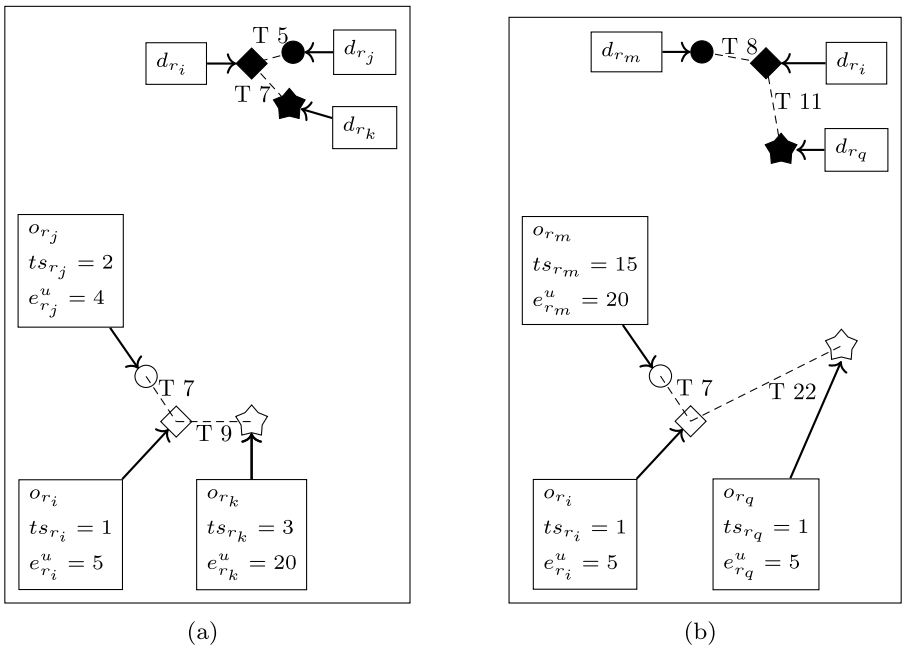


Fig. 6. Examples of requests with different levels of similarity. White shapes are origin nodes and black shapes are destination nodes.

Table 3
Computed values necessary to determine the similarity between requests shown in Fig. 6.

Request pair	ϕ	τ	ϑ	ξ
$r_i r_j$	12.00 ✓	1.00 ✓	1.00 ✓	1.00
$r_i r_k$	16.00 ✓	2.00 ✓	15.00 ✗	0.75
$r_i r_m$	15.00 ✓	14.00 ✗	15.00 ✗	0.50
$r_i r_q$	33.00 ✗	0.00 ✓	0.00 ✓	0.00

a result, the same request will most likely exhibit a different level of similarity from multiple requests. Therefore, to determine the similarity between I and J, we identify the maximum matching $M_s \subset E_s$ and measure the average weight of edges $\Omega_{IJ} = \frac{\sum_{m \in M_s} w_m}{|M_s|}$.

Essentially, this approach attempts to capture requests that are likely to impose similar restrictions during the execution of the algorithm, e.g., occupy vehicles that travel between the same areas at a similar period. Instances sharing the same configuration file (replicas) are expected to have a higher level of similarity when compared to the remainder. Nevertheless, our objective is also to ensure that those instances are still distinct enough to justify performing experiments to extract insightful results.

7. Urban mobility patterns

In this section, we describe the method to approximate various mobility patterns, which can be defined by the different probabilities of a location serving as the trip's origin or destination. Previous literature has revealed significant regularity in human mobility patterns, which directly affects processes driven by it, such as urban planning, traffic engineering, and even epidemic modeling (Song, Qu, Blumm, & Barabási, 2010). The underlying concept here is to generate requests that exhibit more realistic patterns of demand. We validate our method with a real-life data set from rideshare companies.

7.1. Proposed method

The distribution of the distances between positions of two locations consecutively visited (often referred to as displacement) has been well approximated by (truncated) Levy flight models (Barbosa et al., 2018). Conventionally, these models are random walk processes in which the step lengths are distributed according to a (truncated) power law tail probability distribution. In addition, travel directions are random and isotropic. More specifically, it has been demonstrated that the step size denoted by Δd (quantifying the distance between consecutive locations) follows a power law distribution $P(\Delta d) \sim \Delta d^{(-1+\beta)}$ or a truncated power law distribution $P(\Delta d) \sim (\Delta d + d_0)^{(-\beta)} exp(-\Delta d/\kappa)$, where $0 < \beta < 2$, and d_0 and κ symbolize cutoffs at small and large values of d. Short trips are more common than long trips, so the proven distributions of distance decays are coherent.

However, Levy flight models have some limitations, as they do not account for geographical heterogeneity, which expresses that the probability of a location serving as a possible stop in a journey fluctuates according to geographical space. To overcome these limitations, the conventional Levy flight models can be extended. For example, Liu et al. (2012) represent geographical heterogeneity as a function of population density, meaning that highly populated areas are expected to attract more trips.

Furthermore, points of interest (POIs) such as airports, parks, shops, and offices, among others, can increase the number of trips when compared to the estimation by population density. So, inspired by Stouffer's theory of *intervening opportunities* (Stouffer, 1940), which states that "the number of people going a given distance is directly proportional to the number of opportunities at that distance and inversely proportional to the number of intervening opportunities", led Noulas et al. (2012) to propose the rank-distance model, which captures the displacements in an urban environment with high accuracy.

We integrate ideas from the previously mentioned works in order to create a simple method to approximate urban mobility patterns and generate transportation requests that are more realistic. The purpose is to identify how closely the density of POIs combined with randomly chosen distances according to a probabilistic density function can resemble real trips. Given a list of tags (e.g., "restaurant", "bar", "office", "school", "hospital") a built-in function of OSMnx retrieves all POIs from a given area that matches at least one given tag.

First, a zone u needs to be identified, which will contain the origin or destination location. We have $P_u \propto number\_POIs(u)$. Function $number\_POIs(u)$ returns the number of POIs in zone u. The zone is selected with probability P_u that is directly proportional to the number of POIs in this zone, which means that regions with a greater

concentration of POIs will attract more requests. The coordinate for the first location of the request is randomly generated within this zone. Then, distances are randomly chosen according to a probability distribution function (PDF). In this context, a longer distance between two locations indicates a lower probability that they will represent the origin and destination of a request. In case a data set with real trips in the area is available, the PDF can be chosen based on the data under consideration. To identify the distribution that best fits the data, we use the FITTER package (Cokelaer, 2022). Finally, coordinates for a second location are generated, guaranteeing that their distance from the first location approximates the randomly selected distance value. At this stage, the direction is randomly chosen according to a uniform distribution. Comparison between real data and synthetic instances is shown in Section 7.2.

7.2. Comparison between real data and synthetic instances

In this section, we analyze the mobility patterns of real trips performed by rideshare companies. We use data sets that were made publicly available from "Chicago, Illinois" (Chicago Department of Business Affairs & Consumer Protection, 2021). The period selected is September 1 to 30, 2019. We divide these data sets into different periods and study statistical properties, such as spatial and distance distributions. We generate synthetic instances that attempt to approximate the observed patterns in the real data sets and discuss some results. The instances are generated according to attributes for the DARP. The tool takes a configuration file for the DARP as input. The template for this file is discussed in Appendix B. Furthermore, an example of how to integrate the method proposed in Section 7.1 is shown in Appendix C. The output by the tool is a set of instances, and the files are available to download[2]. However, in case the reader is interested in the ODBRP, we refer to the configuration file shown in Appendix D. Furthermore, we explain how to control the size, dynamism, urgency, and geographic dispersion of each instance on Appendix E.

The data set contains more than 7 million trips from ridesharing companies during the selected period. Each record contains information on the pick-up and drop-off time and locations of the trips, distance traveled, fare paid, and many other fields. All records are anonymous. Furthermore, any time stamps are rounded to the nearest 15 min for privacy reasons. Consequently, any results generated with this data set will be an approximation of reality.

Considering that travel patterns change during the day and between days of the week, it is reasonable to conduct this examination on different days and time intervals. First, we split the data sets by days according to working days (business days) and non-working days (weekends and holidays). Second, for working days, we extract two time periods of higher demand: between 7am and 10am, and between 4pm and 8pm. During non-working days, the period considered is between 4pm and 8pm, since in the morning, the activity is almost negligible. The origin and destinations are randomly chosen according to the method described in Section 7, (i.e., following the number of POIs and the best fitted distribution of distances corresponding to each combination of days and time periods).

Fig. 7 illustrates the spatial distribution of origin and destination locations in the data set and the generated instances for each of the three time intervals. It illustrates similar distribution patterns for the intervals. This can be explained as the number of POIs is larger in the more visited areas (see the yellow area with some orange and red spots in Fig. 8). This is further evidence of the positive correlation between the spatial distribution of POIs and the potential of a location to be associated with individuals' trajectories. Despite being more evenly spread than the locations in the real-life data sets, the synthetic instances also display a slightly denser concentration in the region where most of the POIs are situated.

Distance distributions are also similar during working and non-working days. They are depicted in Fig. 9(a). Such statistical measures capture the extent of the population's interactivity within the urban environment. It is important to note that the magnitude and shape of the studied area play an important role. For example, smaller cities impose a more limited upper bound to trip lengths when compared to bigger ones. For the city of "Chicago, Illinois", the distances of 75% of the trips performed by rideshare companies are under 9.17 km for the morning period and below 8.37 km for the afternoon period during working days. Meanwhile, for the interval between 4pm and 8pm during non-working days, 75% of the trips had a length smaller than 8.84 km. A potential reason is that citizens prefer participating in activities within a short to middle-range distance from the areas they spend most of the time (e.g., home and workplace). After reaching its peak (around 2 km), the probability of distance decreases continuously, except when reaching 27 to 30 km. This might be due to the presence of the "O'Hare International Airport", as it is located relatively far from the city.

As previously indicated, the function and parameters that return the best fit for the distance distributions are used for the synthetic instances. The distributions used to generate the synthetic instances are illustrated in Fig. 9(b). The shape of the graphs is relatively similar. First, it is worth noting that the scales of the graphs are slightly different. For the synthetic instances, the occurrence of small to medium distances is somewhat lower when compared to the real trips. 75% of the distances from the generated requests on working days are below 10.22 km and 9.75 km for the morning and afternoon periods, respectively. Furthermore, 75% of the rides in the afternoon on non-working days report distances lower than 10.28 km. These values are slightly higher (about 10% to 15%) when compared with those of real trips from rideshare companies.

The average speed can be evaluated based on the traveled distances and the duration of the trips. Speed values are slightly higher during non-working days when compared to working days. On working days, the computed speed averages are 22.11 km/h and 21.30 km/h for the morning and afternoon intervals, respectively. The average speed during afternoons on non-working days is 24.79 km/h.

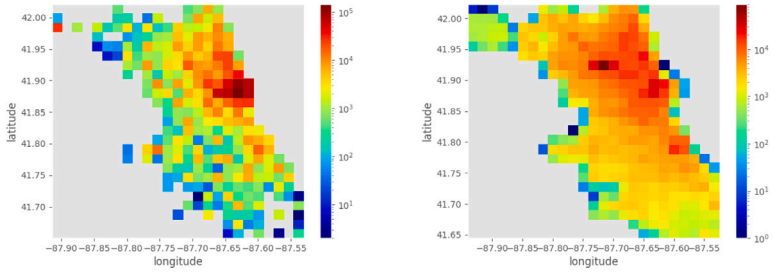
According to Fig. 8, one could argue that considering solely the concentration of POIs can possibly improve the spatial distributions of origin and destination locations in synthetic instances. However, the distances decay effect presented in Fig. 9 might be lost. For this reason, new methods that balance both aspects may be elaborated in further research.

Before concluding this analysis, we make three remarks. First, some results are influenced by the method of transport (e.g., car, bus). Since we utilize data from rideshare companies, the trips also have an economic constraint factor, as it is unlikely for citizens to favor this transportation modality as fares rise significantly with long distances. For example, an investigation of subway rides may return a somewhat different distance distribution, in which higher distances are expected to occur more frequently. Second, the observed spatial and distance distributions are similar for the studied day and time intervals. However, this might change based on the city. Finally, the outcome is sensitive to the available information concerning POIs locations. OSM users provide details and some of them might be missing.

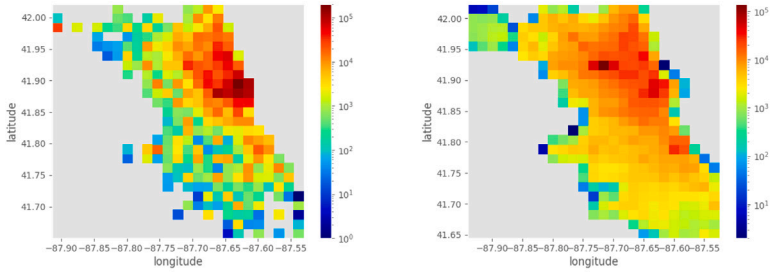
8. Applications of REQreate

In this section, to showcase the value of the tool, we explore how REQreate has been utilized in existing literature. Galarza Montenegro, Sörensen, and Vansteenwegen (2023) consider a dynamic Feeder Service with Mandatory Stops (DFSMS). This system involves the use of buses to efficiently shuttle passengers from areas with low demand to a central location, such as a transportation hub or city center. The buses

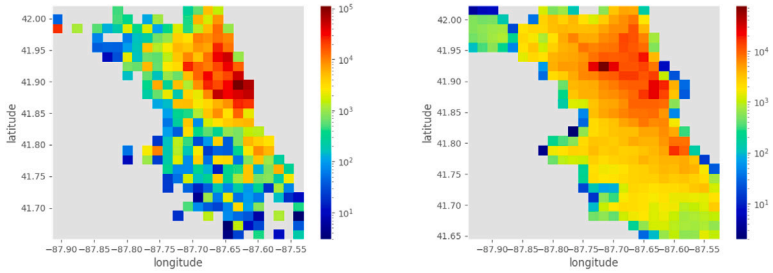
[2] http://bit.ly/3UAXwZ4



(a) Spatial distribution between 7am to 10am during working days.



(b) Spatial distribution between 4pm to 8pm during working days.



(c) Spatial distribution between 4pm to 8pm during non-working days.

Fig. 7. Spatial distribution of origin and destination locations. Figures on the left refer to trips reported by rideshare companies in "Chicago, Illinois". Meanwhile, figures on the right concern synthetic instances. The color bar accounts for the number of locations.. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

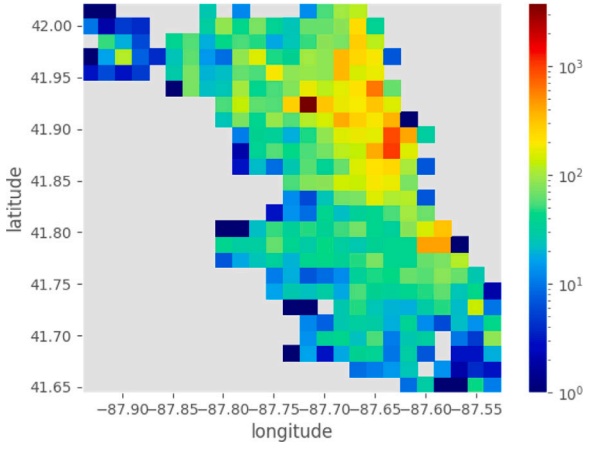


Fig. 8. Spatial distribution for Points of Interest (POIs) in the city of "Chicago, Illinois" according to reported locations in OpenStreetMaps (OSM). The color bar accounts for the number of POIs. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

are required to make stops at specific locations, and they may also make additional stops based on passenger requests. The objective function minimizes various factors, including passenger in-vehicle time, walking time, and the discrepancy between desired departure/arrival times and the actual times. The authors develop a metaheuristic, and to assess its effectiveness, they conduct a case study in Antwerp, using REQreate to generate instances. In their evaluation, they compare the performance of the DFSMS with the existing public transportation options.

The results of their study demonstrate that the DFSMS outperforms the current public transportation alternatives, both in scenarios with low and high passenger demand. In instances involving 100 passenger requests over a two-hour period, the DFSMS outperforms the existing transit options by 1.7% in terms of the objective function when utilizing only six buses. With 15 buses, this performance gap increases to 31.6%. It is worth noting that the current public transit options encompass more than 20 bus lines, indicating that they utilize more resources in comparison to the DFSMS.

Lian, Lucas, and Sörensen (2023) investigate the Electric On-Demand Bus Routing Problem (EODBRP), a complex problem that goes beyond just determining bus routes and schedules to fulfill passenger requests. In the EODBRP, there is also the critical consideration of when to charge the electric buses used in the system. The authors develop a

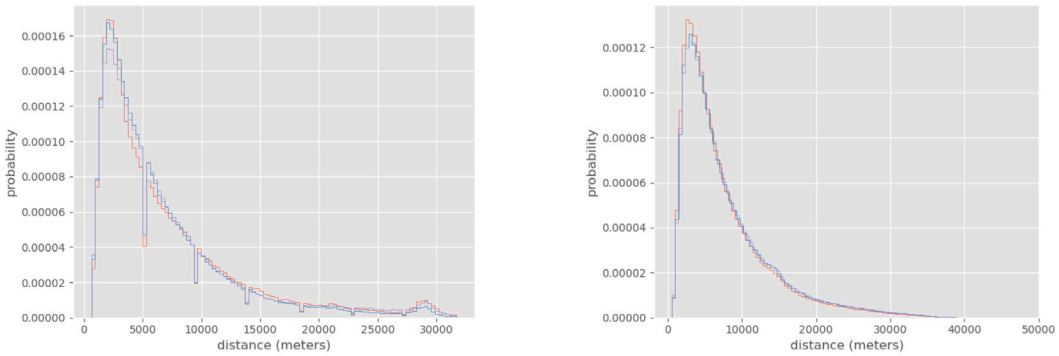


Fig. 9. Distance distribution referring to trips reported by rideshare companies in the city of "Chicago, Illinois" (on the left) and for the set of synthetic instances (on the right). The periods are depicted by different colors. Working days are represented by orange (7am to 10am) and blue (4pm to 8pm). Meanwhile, purple is the period of non-working days between 4pm to 8pm.. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Large Neighborhood Search and apply it to problem instances generated using REQreate. They observe that better solutions are found when buses undergo partial charging during their routes. Furthermore, the authors also analyze different temporal distributions, in which passengers are divided during peak and off-peak hours. Results display that at most one extra electric vehicle is needed when compared to conventional buses. This highlights the potential of electric vehicles to replace conventional ones, despite tighter scheduling constraints due to necessary recharging during service.

Lian, Lucas, and Sörensen (2024) study the ODBRP with a particular focus on prepositioning. Prepositioning involves strategically dispatching buses to stations that are anticipated to have future passenger requests. The authors propose a Variable Neighborhood Search and also test their approach with instances generated with REQreate. The instances used in their study feature dynamic requests characterized by stochastic behavior, meaning there is an associated probability for these requests to materialize. Prepositioning is shown to outperform the dynamic solution without prepositioning when the forecast accuracy of stochastic requests is above 40%.

9. Conclusion and future research

In this paper, we present REQreate, a tool aimed at generating instances that are significantly more realistic than previous approaches for on-demand public transportation problems. Instances of these problems mainly consist of demand from passengers for transportation. Hence, real-life networks are retrieved from OpenStreetMaps (OSM) with the support of the OSMnx tool (Boeing, 2017). We describe in what manner the attributes and parameters for instances can be given as input to REQreate via a JSON file. Given notations for the Dial-A-Ride Problem (DARP) and On-Demand Bus Routing Problem (ODBRP), we further present properties that can be considered when analyzing the performance of optimization algorithms: size, dynamism, urgency, and geographic dispersion. The concept of instance similarity is also proposed to introduce more diversity in the instance sets.

We use data sets provided by rideshare companies from the city of "Chicago, Illinois" and examine which statistical properties of human movement could be inferred. Furthermore, REQreate is utilized to generate synthetic instances, and we compare the properties of the two groups. Requests are randomly chosen according to a simple framework taking into account Points of Interest (POIs) and distances as major factors that impact human trajectories. The method has simple yet reasonable assumptions which can be useful in producing distinct urban patterns when data accessibility is an issue. The resemblance between the distribution of requests in rideshare companies' data sets and the distribution of PoIs indicates that REQreate is capable of generating instances replicating realistic patterns.

While the primary focus of this paper revolves around on-demand transportation problems tied to passenger ride requests, it is worth noting that REQreate can also be adapted to generate instances for scenarios involving the transportation of goods. However, achieving realism for these specific contexts may necessitate additional research to establish reasonable patterns. This is due to the fact that the transportation of goods may not align with PoIs, particularly since many deliveries are made to residential locations. We also acknowledge that another of REQreate's current limitations is the absence of realistic travel time data. Obtaining precise road travel time information poses a challenge, which has led to the tool relying on approximations of maximum road speeds.

Appropriate future research directions include new methods to mathematically describe a wider variety of human behavior regarding intra-urban traveling. Moreover, additional studies using data sets from sources other than rideshare companies, such as metro rides, can also be explored in future research. Another interesting direction is to understand how the proposed properties (size, dynamism, urgency, and geographic dispersion) influence the performance of a newly developed method. Studies can then be conducted to evaluate how these properties affect various performance metrics, such as deadheading times, shared times, and extra travel time. Additionally, they can help determine the adequate fleet size and vehicle capacity for specific scenarios. Such analyses contribute to the development of more robust algorithms and the implementation of systems that achieve a better balance between customer satisfaction and operational costs.

Acknowledgments

This project was funded by the University of Antwerp BOF, Belgium.

Appendix A. Generating requests

The process of creating instances for the targeted problems in this paper requires generating a set of requests with attributes randomly chosen according to probability density functions (pdf) or expressed as relations between each other. Attributes and parameters that define an instance are described in a configuration file given as input to REQreate. Similarly to Ullrich et al. (2018), we also choose the JSON format, considering it is simple, flexible, and easily readable by Python, the language in which our instance generator itself is developed. We

now outline the values that should be provided in order to generate requests.

A.1. JSON syntax introduction

JSON files consist of data represented by *name/value* pairs. The field *name* is written in double quotes. The field *value* is separated from *name* by a colon (:), supports primitives data types (strings, numeric, boolean), and more complex structures such as arrays and objects. Arrays are enclosed in square brackets ([ ]) and are ordered sets of values. Objects are enclosed in curly brackets ({ }) and are collections of *name/value* pairs. The full review of JSON syntax is out of the scope of this paper, therefore we recommend the user to study the examples provided in the tool's GitHub page to acquire familiarity with the format.

A simple structure of a standard configuration file in JSON format is shown in Listing 1. Default data pairs will be referred to as items. The names for supported items are: *network*, *seed*, *problem*, *fixed_lines*, *max_speed_factor*, *replicas*, *requests*, *places*, *parameters*, *attributes*, and *instance_filename*. Pairs enclosed in objects will be referred to as *sub-items*, e.g., in item *parameters*, an object is depicted with 3 sub-items: *name*, *type*, and *value*.

A.2. General items

Listing 2 illustrates the items *network*, *seed*, *problem*, *fixed_lines*, *max_speed_factor*, *replicas*, *requests*, and *instance_filename*. First, *network* is a string used to retrieve the network details as described in Section 4. The *seed* consists of a single integer value and can be used to force the random number generator to generate the same set of numbers at every run of the tool. This allows for reproducibility, meaning that is only necessary to share the configuration files to obtain the same set of instances, instead of the instance files themselves. The problem's acronym can be specified with a string in item *problem*. Information regarding fixed public transport lines can be requested by adjusting the value of binary item *fixed_lines* to "true". The value of α, mentioned in Section 4, can be specified with *max_speed_factor*.

Listing 1: Configuration file structure

```
{
    "network": "Chicago, Illinois",
    "seed": 100,
    ...
    "places": [...],
    "parameters": [
        {
            "name": "simple_parameter",
            "type": "integer",
            "value": 10
        },
        {...}
    ],
    "attributes": [...]
}
```

Both *requests* and *replicas* are integer numbers. The former specifies the number of requests in the instance, while the latter indicates the number of generated request files. Each replica will share the same configuration file, and have e.g., the same number of requests, but the requests themselves will differ because of the randomization process. The pattern of the instance filename is provided as an array in item *instance_filename*. The name of any item that has a single primitive as a value can be provided. In Listing 2, the values of "network", "problem", and "requests" will constitute the instance filename separated by underscores. Additionally, as each instance must have a unique name, a number from 1 to *replicas* will automatically be printed at the end based on the sequence of generation. Consequently, "Chicago,Illinois_DARP_500_1" is the filename of the first instance for this particular case.

Table A.1
Summary for *places*.

Sub-items	Possible values
name	string
type	"location", "zone"
lon	longitude (geographic coordinate)
lat	latitude (geographic coordinate)
centroid	boolean
class	"school"
length_lon	real
length_lat	real
radius	real
length_unit	"m", "km", "mi"

Listing 2: Example values for items *network*, *seed*, *problem*, *fixed_lines*, *replicas*, *requests*, and *instance_filename*

```
{   "network": "Chicago, Illinois",
    "seed": 100,
    "problem": "DARP",
    "fixed_lines": true,
    "max_speed_factor": 0.5,
    "replicas": 10,
    "requests": 500,
    "instance_filename": ["network", "problem", "
        requests"],
    ...
}
```

A.3. Places

The item *places* is an array of objects. This item provides the option to define locations and zones. Specific locations could be used to specify the coordinates of a garage where vehicles are kept waiting to be dispatched (i.e., a depot). Zones are areas within the network and are suitable to characterize a city center, for example. Table A.1 depicts the potential sub-items for *places*. The first column represents the default name. The second column details the data types or possible values of the sub-item. Possible values are differentiated from data types by being written between double quotes. The *name* is the identifier of the place and could be represented as any sequence of characters between double quotes (string), with the exception of the existing default names. The two options for *type* are "location" and "zone". For a *location*, longitude (*lon*) and latitude (*lat*) must be specified (coordinates in the spherical coordinate system), or *centroid* (the center point of the particular network) must be set to "true". Certain locations must have a sub-item called *class*. The only option at the moment for *class* is "school", but this can be easily extended.

Listing 3: Examples for item *places*

```
{   ...
    "places": [
      {
          "name": "location_example",
          "type": "location",
          "lon": -1.6457340723441525,
          "lat": 48.100199454954804
      },
      {
          "name": "zone_representation",
          "type": "zone",
          "lon": -1.6902891077472344,
          "lat": 48.09282962664396,
          "length_lon": 1000,
          "length_lat": 1000,
          "length_unit": "m"
      },
```

```
        {
            "name": "zone_center",
            "type": "zone",
            "centroid": true,
            "radius": 1500,
            "length_unit": "m"
        },
    ],
    ...
}
```

Regarding *zone*, *lon/lat* or *centroid* represent the center point of the area. Apart from that, the polygon must also be determined by providing *length_lon* and *length_lat* to specify the side lengths of a rectangle/square shape, or *radius* to indicate the radius of a circumference. Finally, *length_unit* represents the units of length of values declared inside the object, and can be set to "m", "km" or "mi", which stands for meters, kilometers, and miles, respectively. Listing 3 illustrates item *places*. A place is categorized as a "location" and named "location_example", including "lon" and "lat" coordinates equal to −1.6457340723441525 and 48.100199454954804, respectively. Next, a zone has "zone_representation" as the identifier, and center coordinates set to −1.6902891077472344 ("lon") and 48.09282962664396 ("lat"). We remark that the tool always verifies if those given coordinates are within the boundaries of the network and it raises an error otherwise. The polygon of "zone_representation" is established by fixing *length_lon* and *length_lat* to 1,000 m. Lastly, "zone_center" is an area stipulated as a circumference with *radius* of 1,500 m, and *centroid* set to "true".

A.4. Parameters

Item *parameters* is also an array of objects. Table A.2 depicts the potential sub-items for *parameters*. The *name* is the identifier of the parameter. Possible options for *type* are "string", "integer", "real", "array_primitives", "array_locations" and "array_zones". Sub-item *value* is *type* dependent: (a) a sequence of characters for "string"; (b) an integer number for "integer"; (c) a real number for "real"; (d) an array with primitive values for "array_primitives"; (e) an array with the names of locations declared in *places* for "array_locations"; and (e) an array with the names of zones declared in *places* for "array_zones".

Listing 4: Example for item *parameters*

```
{   ...
    "parameters":[
        {
            "name": "min_early_departure",
            "type": "integer",
            "value": 5,
            "time_unit": "h"
        },
        {
            "name": "many_locations",
            "type": "array_location",
            "value": ["location_example"],
            "size": 3,
            "locs": "random"
        },
        {
            "name": "pair_zones",
            "type": "array_zones",
            "value": ["zone_representation", "
                zone_center"],
            "size": 2
        }
    ],
     ...
}
```

Units of measurement for time, length, and speed are expressed in *time_unit*, *length_unit*, and *speed_unit*, respectively. The abbreviations "s", "min", and "h" for *time_unit* refer to seconds, minutes, and hours, respectively. The possible values for *length_unit* were previously explained, whereas, in *speed_unit*, "mps" "kmh" and "miph" represent meters per second, kilometers per hour and miles per hour, respectively.

Table A.2
Summary for *parameters*.

Sub-items	Possible values
name	string
type	"string", "integer", "real", "array_primitives", "array_locations", "array_zones"
value	*type* dependent
time_unit	"s", "min", "h"
length_unit	"m", "km", "mi"
speed_unit	"mps", "kmh", "miph"
size	integer
locs	"schools", "random"

We remark on the importance of providing these units of measurement, so these values can be properly converted by the generator since all internal operations are done in meters, seconds, and meters per second. Assuming the type for a parameter to be *array_locations* or *array_zones*, an integer number must be specified in *size* to delimit its size. In the event of *size* being greater than the number of input names on the given array, random locations will be randomly chosen according to *locs*. This provides the flexibility of having fixed and unique locations/zones across different instances. Values accepted for *locs* are "random" and "schools". We emphasize that in both cases coordinates within the boundaries of the network are randomly chosen.

Examples for *parameters* are shown in Listing 4. Parameter "min_early_departure" has *type* established as an integer, its *value* equals to 5, and reported with "h" (hour) as the unit of time, and in the 24-hour clock format can be interpreted as 5:00. The 24-hour clock format will be the notation used throughout this paper, however in REQreate time is represented in seconds. Representation of a set of locations with *size* 3 can be seen in a parameter named "many_locations". The *value* for this parameter is an array containing the name "location_example" (defined in Listing 3), and its 2 remaining locations will be randomly chosen. Assume, for example, this parameter represents the locations of depots. Presupposing the user has one fixed location already determined (declared as "location_example"), they can generate different instances and use them to study how much of an impact these randomly generated locations of depots have on solution quality. Furthermore, "pair_zones" lists 2 zones: "zone_representation" and "zone_center", both defined in Listing 3. More details on how these parameters can be used are exemplified on Appendix A.5.

A.5. Attributes

The *attributes* of an instance are declared in an array of objects. They are usually represented by a range of possible values, or by a relation between other *attributes* and *parameters*. Table A.3 depicts the potential sub-items for *attributes*. The sub-items *name*, *type*, *time_unit*, *length_unit*, and *speed_unit* are equivalent to the ones previously presented for *places* and *parameters*. The sub-item *pdf* (probability density function) is an object to represent a range of values for the attribute, and its structure is detailed in Table A.4. The value of an attribute that carries a *pdf* in its declaration is randomly chosen according to the following types of probabilistic distribution functions: "cauchy", exponential ("expon"), "gamma", "gilbrat", lognormal ("lognorm"), "normal", "powerlaw", "uniform" and "wald". Other distributions can be made available in the future. The sub-items "loc" and "scale" for "uniform" indicate the closed interval (["loc","loc" + "scale"]) in which values will be randomly chosen according to a uniform distribution. Regarding "normal", the mean and standard deviation of the distribution are expressed by "loc" and "scale", respectively. Besides "loc" and "scale", some functions require an additional parameter "aux". For a full rundown on what each parameter means for each distribution, we refer to SciPy's documentation[3]. The declared values must be in conformity with the

[3] https://docs.scipy.org/doc/scipy/index.html

Table A.3
Summary for *attributes*.

Items	Possible values
name	string
type	"string", "integer", "real", "location", "array_primitives"
time_unit	"s", "min", "h"
length_unit	"m", "km", "mi"
speed_unit	"mps", "kmh", "miph"
pdf	object
expression	string
constraints	array of strings
subset_primitives	string
subset_locations	string
subset_zones	string
weights	array of numbers
output_csv	boolean

Table A.4
Summary for *pdf*.

Sub-items	Possible values
name	string
type	"cauchy", "expon", "gamma", "gilbrat", "lognorm", "normal", "powerlaw", "uniform", "wald"
loc	real
scale	real
aux	real

unit of measurement given for the attribute. Moreover, an attribute can be represented as a mathematical expression transmitted as a string. Restrictions are imposed with *constraints*, which is an array of formulas to be evaluated as true or false. The syntax for *expression* and *constraints* follow Python standards and can contain numbers, identifiers for other attributes or parameters, mathematical symbols, and parentheses.

Sub-item *subset_primitives* takes as value an identifier of a parameter previously declared as an array of primitives, and whenever set, the value of the attribute will be chosen considering the given array. Regarding attributes that have "location" as a type, coordinates will be randomly chosen taking into account the full network area. Exceptions are when the method explained in Section 7 is set, or when either *subset_locations* or *subset_zones* are declared, whose values consist of a string containing the *name* of a parameter stated as an array of locations or zones. Regarding *subset_locations* a location will be randomly chosen from the declared array, but in the case of *subset_zones*, one zone is first randomly chosen and then a coordinate within its boundaries is selected. The option to influence the possibility of randomly choosing a value from *subset_primitives*, a location from *subset_locations*, or a zone from *subset_zones*, is supported with *weights*, which contains an array of numbers indicating the probability of an element to be selected. Note that the numbers in *weights* must be sequentially declared in accordance with the values of the array it refers to. Item *output_csv* indicates if the attributed is printed in the instance CSV file ("true", which is the default) or not ("false"). The attributes that are not printed have merely the purpose to assist in the definition of other attributes or to impose more constraints.

Lastly, we provide the opportunity to express the interaction of locations through a travel time matrix, i.e., a structure formed by cells indicating the journey time between origin and destination pairs. For this purpose, *travel_time_matrix* must be reported containing an array with *names* of locations originally declared in *parameters* or *attributes*. The tool computes the trip duration for every location pair using Dijkstra's algorithm and populates the cells of a separate CSV file, whose first row and column are headers indicating the source and target, respectively. A graph is also created, in which nodes are locations, and arcs have a weight associated indicating the travel time. The graph is saved as GraphML format.

Examples for *attributes* and *travel_time_matrix* are depicted in Listing 5. First, "earliest_departure" has type "integer", and its values are randomly chosen according to a normal probability distribution with mean 30,600 s (8:30), a standard deviation of 3600 s (1 h), and constrained to be greater than or equal to parameter "min_early_departure" (declared in Listing 4). Meanwhile, attribute "latest_arrival" is calculated after the mathematical expression "earliest_departure + 1800". The coordinates for "origin" can be a part of anywhere in the network, whereas, for "destination", the coordinate will belong to either zone in "pair_zones" (see Listing 4), with "zone_center" being 3 times more likely to be chosen according to *weights*. The locations that will constitute *travel_time_matrix* are the values of attributes "origin" and "destination" of each request.

A.6. Invoking method

The set of instances will be generated after invoking a method with the configuration file name as an argument (see Listing 6). According to the attributes declared in the configuration file, the tool generates the data of one request at a time. Analogous to the technique described by Ullrich et al. (2018), a directed acyclic graph $G = (N, A)$ is built with attributes representing nodes N, and arcs A defined by expressions or constraints. Then, the attributes (nodes) are topologically sorted in conformity with their dependencies (expressions and constraints).

Listing 5: Examples for items *attributes* and *travel_time_matrix*

```
{   ...
   "attributes": [
      {
         "name": "earliest_departure",
         "type": "integer",
         "time_unit": "s",
         "pdf": {
                   "type": "normal",
                   "loc": 30600,
                   "scale": 3600
                },
         "constraints": [ "earliest_departure >=
             min_early_departure"]
      },
      {
         "name": "latest_arrival",
         "type": "integer",
         "time_unit": "s",
         "expression": "earliest_departure + 1800"
      },
      {
         "name": "origin",
         "type": "location"
      },
      {
         "name": "destination",
         "type": "location",
         "subset_zones": "pair_zones",
         "weights": [1, 3]
      }
   ],
   "travel_time_matrix": ["origin", "destination"]
}
```

Given an ordered list of nodes, if node u precedes v on the list for every arc $(u, v) \in A$, then such ordering is considered a topological ordering. For example, in Listing 5, attribute "latest_arrival" is calculated according to the expression "earliest_departure + 1800", which represents a dependency (edge) of "latest_arrival" with attribute "earliest_departure". This means that the value of attribute "earliest_departure" must be known before calculating the expression "earliest_departure + 1800". Therefore, attribute "earliest_departure" must precede attribute "latest_arrival" in a topological ordering. The ordering ["origin", "destination", "earliest_departure", "latest_arrival"] is an example of a topological ordering for attributes declared in Listing 5.

Listing 6: Demonstration of invoking method to generate instances declared in configuration file "example_config.json"

```
import input_json

input_json('example_config.json')
```

Initially, in accordance with the topological ordering, a value for an attribute is generated. Then, the feasibility is checked based on the disclosed constraints and if any of them is violated, the procedure restarts by discarding the current attribute. This process is repeated until all attributes for a request are valid. Considering Listing 5, the locations of "origin" and "destination" are randomly chosen within the boundaries of the network. Later, the tool generates a time for "earliest_departure" and inspect if it does not violate the constraint "earliest_departure ≥ min_early_departure". Finally, the value for "latest_arrival" can be computed according to the expression "earliest_arrival + 1800". These steps are repeated for each request. We remark that is beyond the scope of the tool to certify if the set of constraints creates dependencies that cannot be met, i.e., every possible combination of attribute values is infeasible. Thus, after a large number of unsuccessful iterations, the generator will halt, raising an error informing it was unable to meet the requirements.

Appendix B. Configuration file for the DARP

The configuration file example for the DARP is divided into Listings 7 and 8. We remark that these Listings are part of the same configuration file, but they are split for visualization purposes. Moreover, this is a simplified version of the configuration file, as items *seed*, *network*, among others were omitted. First, in Listing 7, the planning period is given as input in parameters "min_planning_period" (ts_{min}) and "max_planning_period" ts_{max}. The declared values will later be used to indicate the interval for request announcements is between 7:00 and 10:00. The vehicles are located in a single depot randomly generated ("depots"). The "origin" (o_p) and "destination" (d_p) attributes will be randomly chosen within the boundaries of the network. The integer "wheelchair_requirement" attribute will be uniformly chosen for each request inside the interval [0,1], where 1 indicates a requirement for a vehicle that supports a wheelchair, and 0 otherwise. The direct travel time between "origin" and "destination" is disclosed in attribute "direct_travel_time", established by an expression including a predefined function that computes an estimated travel time between two locations: "dtt(x,y)", where "x" and "y" are attributes or parameters declared with type "location". The earliest departure is named "earliest_departure" (e_p^u), the values are randomly chosen according to a normal distribution with mean 30,600 (8:30) and standard deviation 3600 (1 h), and the constraint state it must be greater than or equal to parameter "min_planning_period".

The remainder of the attributes are declared in Listing 8. The attribute "time_stamp" (ts_p) is specified by the subtraction of "earliest_departure" from an attributed named "lead_time" ("earliest_departure - lead_time"), where the latter is uniformly chosen between 0 and 600 s (0–10 min). Additionally, "time_stamp" must also respect the planning period time constraints. Similarly, "latest_departure" (l_p^u) is specified by an expression ("earliest_departure + time_window_size"), where values for "time_window_size" are randomly chosen within the interval [300,600] seconds (5–10 min) according to a uniform distribution. The "earliest_arrival" (e_p^o) is simply calculated from the expression "earliest_departure + direct_travel_time". Attribute "latest_arrival" (l_p^o) is calculated by adding "time_window_length" to "earliest_arrival" and is restricted to be lesser than or equal to parameter "max_early_departure". The number of users is characterized by the integer "number_users", and this attribute will be uniformly chosen for each request within the interval [1,3]. The travel time

Listing 7: Configuration file example for the DARP - *part 1*

```
{   ...
   "parameters":[
      {
         "name": "min_planning_period",
         "type": "integer",
         "value": 7,
         "time_unit": "h"
      },
      {
         "name": "max_planning_period",
         "type": "integer",
         "value": 10,
         "time_unit": "h"
      },
      {
         "name": "depots",
         "type": "array_locations",
         "size": 1,
         "locs": "random"
      }
   ],
   "attributes":[
      {
         "name": "origin",
         "type": "location"
      },
      {
         "name": "destination",
         "type": "location"
      },
      {
         "name": "wheelchair_requirement",
         "type": "integer",
         "pdf": {
                     "type": "uniform",
                     "loc": 0,
                     "scale": 1
                  }
      },
      {
         "name": "direct_travel_time",
         "type": "integer",
         "time_unit": "s",
         "expression": "dtt(origin,destination)",
         "output_csv": false
      },
      {
         "name": "earliest_departure",
         "type": "integer",
         "time_unit": "s",
         "pdf": {
                     "type": "normal",
                     "loc": 30600,
                     "scale": 3600
                  },
         "constraints": [ "earliest_departure >=
             min_planning_period"]
      }
      ...
   ]
    ...
}
```

matrix consists of estimated travel times between all pairs of locations in "depots" (W), "origin" (O), and "destination" (D). Finally, we emphasize that each instance can be tested with several combinations of fleet size and vehicle capacity.

Appendix C. Urban mobility patterns example

Listing 9 depicts an example of how to integrate the procedure described in Section 7. A supplementary item named "method_pois" is included containing two sub-items: "locations" and "pdf". The former contains a list with the two locations from *attributes* that will be either selected according to the higher density of POIs or based on a random distance value from the preceding location, meanwhile the

Listing 8: Configuration file example for the DARP - *part 2*

```
{   ...
   "attributes": [
      {
         "name": "lead_time",
         "type": "integer",
         "time_unit": "s",
         "pdf": {
                    "type": "uniform",
                    "loc": 0,
                    "scale": 600
                },
         "output_csv": false
      },
      {
         "name": "time_stamp",
         "type": "integer",
         "time_unit": "s",
         "expression": [ "earliest_departure -
             lead_time"]
         "constraints": [ "time_stamp >=
             min_planning_period", "time_stamp <=
             max_planning_period"]
      },
      {
         "name": "time_window_size",
         "type": "integer",
         "time_unit": "s",
         "pdf": {
                    "type": "uniform",
                    "loc": 300,
                    "scale": 300
                },
         "output_csv": false
      },
      {
         "name": "latest_departure",
         "type": "integer",
         "time_unit": "s",
         "expression": "earliest_departure +
             time_window_size"
      },
      {
         "name": "earliest_arrival",
         "type": "integer",
         "time_unit": "s",
         "expression": "earliest_departure +
             direct_travel_time"
      },
      {
         "name": "latest_arrival",
         "type": "integer",
         "time_unit": "s",
         "expression": "earliest_arrival +
             time_window_size",
         "constraints": [ "latest_arrival <=
             max_planning_period"]
      },
      {
         "name": "number_users",
         "type": "integer",
         "pdf": {
                    "type": "uniform",
                    "loc": 1,
                    "scale": 2
                }
      }
   ],
   "travel_time_matrix": ["depots", "origin", "
       destination"]
}
```

latter describes the parameters of the probabilistic density function to which the distances will be randomly chosen.

Appendix D. Configuration file for the ODBRP

The configuration file example for the ODBRP is divided into Listings 10 and 11. As previously remarked in Appendix B, these Listings

Listing 9: Demonstration of how the method can be embedded in the instance generation process

```
{   ...
   "attributes": [
        ...
      {
         "name": "origin",
         "type": "location"
      },
      {
         "name": "destination",
         "type": "location"
      },
      ...
   ],
   "method_pois": [
      {
         "locations": ["origin", "destination"],
         "pdf": {
                    "type": "normal",
                    "loc": 6500,
                    "scale": 5000
                }
      }
   ],
    ...
}
```

are part of the same configuration file. They were also simplified for visualization purposes. First, in Listing 10, a zone named "zone_center" is declared in item *places*, where its center point coincides with the network's centroid. The planning period is given as input in parameters "min_planning_period" (ts_{min}) and "max_planning_period" ts_{max}. The declared values indicate the interval for request announcements is between 6:00 and 9:00. An array of zones named "zones_dest" is declared containing only "zone_center". The "origin" (o_p) and "destination" (d_p) attributes will be randomly chosen within the boundaries of the network, however, "destination" is bounded to the specific zone declared in array "zones_dest". Suppose this configuration file is a hypothetical example where the target location of requests is the city center, usually employees commute to this area during morning peak hours. The direct travel time between "origin" and "destination" is disclosed in attribute "direct_travel_time" and was previously described in Appendix B. The "earliest_departure" (e_p^u) values are randomly chosen according to a uniform distribution within the interval [25200, 32400] (7:00 to 9:00). We remark that constraint "earliest_departure >= min_planning_period" is indeed redundant for this attribute, however, we display it in the configuration file for elucidation purposes.

The remainder of the attributes are declared in Listing 11. First, "max_walking" and "walk_speed" are default names for attributes that store, correspondingly, the maximum time a passenger is willing to walk to a bus station and their average walking speed. For this example, the values from "max_walking" and "walk_speed" are randomly chosen according to a uniform distributions between 300 to 600 s (5–10 min) and 4 to 5 km/h, respectively. The set of departure and arrival bus stations are declared in attributes "stops_orgn" and "stops_dest", respectively. The expressions that specify both attributes include a predefined function "stops(x)", which returns all bus stations within less than "max_walking" from location "x". The constraints in attributes "stops_orgn" and "stops_dest" are written with built-in functions of Python[4]. The first constraint is "len(y) > 0", which guarantees that the array "y" is not empty. The second constraint "not (set(y) & set(z))" guarantees that there is no intersection between the two arrays "y" and "z". The attributes "time_stamp" (ts_p) and

[4] We strongly recommend getting familiar with the built-in Python functions, as it will help the user grasp the capabilities of the generator.

"lead_time" are declared exactly as in Appendix B, except for the addition of sub-item *static_probability*. By default, *static_probability* can take the value of a real number within the interval [0,1], expressing the probability of a request being known before the planning period stars (the "time_stamp" is set to 0 which represents a static request), therefore allowing to schedule requests in advance. Attribute "latest_arrival" (l_p^o) is calculated by multiplying "direct_travel_time" to 1.5 and adding the result with "earliest_departure". Constraint "latest_arrivel <= max_planning_period" ensures that "latest_arrival" does not surpass the planning period time window. The attribute "number_users" is declared exactly as in Appendix B. The travel time matrix consists of estimated travel times between all pairs in "bus_stations", which is a default name for the bus stations within the boundaries of the network. Finally, we further emphasize that each instance can be tested with several combinations of fleet size and vehicle (bus) capacity.

Listing 10: Configuration file example for the ODBRP - *part 1*

```
{   ...
    "places": [
      {
          "name": "zone_center",
          "type": "zone",
          "centroid": true,
          "radius": 2000,
          "length_unit": "m"
      }
   ],
   "parameters":[
      {
          "name": "min_planning_period",
          "type": "integer",
          "value": 6,
          "time_unit": "h"
      },
      {
          "name": "max_planning_period",
          "type": "integer",
          "value": 9,
          "time_unit": "h"
      },
      {
          "name": "zone_dest",
          "type": "array_zones",
          "size": 1,
          "value": ["zone_center"]
      },
   ],
   "attributes":[
      {
          "name": "origin",
          "type": "location"
      },
      {
          "name": "destination",
          "type": "location",
          "subset_zones": "zone_dest",
      },
      {
          "name": "direct_travel_time",
          "type": "integer",
          "time_unit": "s",
          "expression": "dtt(origin,destination)",
          "output_csv": false
      },
      {
          "name": "earliest_departure",
          "type": "integer",
          "time_unit": "s",
          "pdf": {
                    "type": "uniform",
                    "loc": 25200,
                    "scale": 7200
                },
          "constraints": [ "earliest_departure >=
              min_planning_period"]
      },
      ...
   ]
    ...
}
```

Listing 11: Configuration file example for the ODBRP - *part 2*

```
{   ...
   "attributes": [
      {
          "name": "max_walking",
          "type": "integer",
          "time_unit": "s",
          "pdf": {
                    "type": "uniform",
                    "loc": 300,
                    "scale": 300
                },
          "output_csv": false
      },
      {
          "name": "walk_speed",
          "type": "real",
          "time_unit": "kmh",
          "pdf": {
                    "type": "uniform",
                    "loc": 4,
                    "scale": 1
                }
      },
      {
          "name": "stops_orgn",
          "type": "array_primitives",
          "expression": "stops(origin)",
          "constraints": ["len(stops_orgn) > 0"]
      },
      {
          "name": "stops_dest",
          "type": "array_primitives",
          "expression": "stops(destination)",
          "constraints": ["len(stops_dest) > 0", "not
               (set(stops_orgn) & set(stops_dest))"]
      }
      {
          "name": "lead_time",
          "type": "integer",
          "time_unit": "s",
          "pdf": {
                    "type": "uniform",
                    "loc": 0,
                    "scale": 600
                },
          "output_csv": false
      },
      {
          "name": "time_stamp",
          "type": "integer",
          "time_unit": "s",
          "expression": [ "earliest_departure -
              lead_time"]
          "constraints": [ "time_stamp >=
              min_planning_period", "time_stamp <=
              max_planning_period"],
          "static_probability": 0.5
      },
      {
          "name": "latest_arrival",
          "type": "integer",
          "time_unit": "s",
          "expression": "earliest_departure + (
              direct_travel_time * 1.5)",
          "constraints": [ "latest_arrival <=
              max_planning_period"]
      },
      {
          "name": "number_users",
          "type": "integer",
          "pdf": {
                    "type": "uniform",
                    "loc": 1,
                    "scale": 2
                }
      }

   ],
   "travel_time_matrix": ["bus_stations"]
}
```

Appendix E. Control instance properties

Size can be easily adjusted with item *requests*, as shown in Appendix A.2. Listing 12 depicts a configuration file for an instance with 500 requests. Meanwhile, dynamism can be regulated by adding a sub-item inside an attribute called "time_stamp", as seen in Listing 12. In this example, the instance would have 50% (0.50) dynamism. We note that for example, an exact 100% dynamism case can only be generated if the number of dynamic requests is a factor (exact divisor) to the size of the planning period, i.e., exactly one request arriving every $\frac{ts_{max}-ts_{min}}{|R_d|}$ seconds. Therefore one should be careful when setting a specific value for dynamism since according to the planning period and number of requests, only approximate levels of dynamism are achievable.

Listing 12: Example on how to control properties.

```
{   ...
    "attributes": [
        "requests": 500,
        {
            "name": "time_stamp",
            "type": "integer",
            "time_unit": "s",
            "pdf": {
                        "type": "uniform",
                        "loc": 30000,
                        "scale": 500
                    },
            "dynamism": 50
        },
        {
            "name": "reaction_time",
            "type": "integer",
            "time_unit": "s",
            "pdf": {
                        "type": "normal",
                        "loc": 600,
                        "scale": 60
                    }
        },
        {
            "name": "latest_departure",
            "type": "integer",
            "time_unit": "s",
            "expression": "time_stamp + reaction_time"
        },
        {
            "name": "direct_travel_time",
            "type": "integer",
            "time_unit": "s",
            "expression": "dtt(origin,destination)",
            "constraints": ["direct_travel_time >= 600"
                , "direct_travel_time <= 1200"]
        }
    ],
}
```

The attribute "reaction_time" in Listing 12 controls the levels of urgency. See how "latest_departure" is computed as the sum of "time_stamp" and "reaction_time", which causes the length of the intervals to perform actions to be equal to attribute "reaction_time". In this example, we use a normal distribution (sub-item *pdf*), and the urgency of this instance will have an approximate mean ($\overline{\chi}$) of 10 min (600 s) and standard deviation (χ_s) of 1 min (60 s). To obtain higher or lower levels of urgency one must appropriately change these values. Finally, geographic dispersion can be manipulated by declaring an interval of values in the attribute named "direct_travel_time" (see again Listing 12), which stores the direct travel time between two locations named "origin" and "destination". The expression includes a predefined function that computes the estimated travel time between the locations. In the given example, every request will have between 10 and 20 min of time distance between the origin and the destination. So, the interval can be adjusted in order to create instances with a greater or smaller value of geographic dispersion.

Appendix F. Common values in the literature

In this appendix, we measure the values of size, dynamism, urgency, geographic dispersion and instance similarity on proposed instances in the literature. Most available instances are proposed for static problems. However, dynamism and urgency are exclusively properties for dynamic problems, thus we convert these instances into dynamic ones using methods described in the literature. We discuss how these properties apply to the numerical instances currently used in the existing literature. To achieve this, we report the minimum, maximum, average, and standard deviation for each property. Researchers can then select the appropriate benchmark set or find research gaps according to their specific goals and the characteristics they wish to examine in dynamic optimization for on-demand transportation problems.

The first introduced converting method is by Berbeglia, Cordeau, and Laporte (2012). The authors use one parameter ϱ_1 to define the percentage of requests known in advance (static requests). They also specify an upper bound on how far in advance a request must be known in order to be able to serve it. The time a request is known is expressed by subtracting this upper bound from another parameter ϱ_2. Instances are converted from static to dynamic instances using parameters $\varrho_1 = 25\%$ and $\varrho_2 = rand(60, 240)$, where $rand(60, 240)$ is a randomly generated number in the interval $[60, 240]$ according to a uniform distribution. Second, Wong, Han, and Yuen (2014) consider that the interarrival times of requests are exponentially distributed with the mean equal to the size of the planning period divided by the number of dynamic requests. The authors also range the percentage of dynamic requests from 0% to 100% in steps of 10.

We converted the instances proposed by Braekers and Kovacs (2016), Cordeau (2006) and Cordeau and Laporte (2003) and report the values for size, dynamism, urgency, and geographic dispersion in Tables A.5 to A.7. Since the conversion methods rely on randomness, we run each method ten times for each instance and report the minimum (min), maximum (max), average (avg), and standard deviation (std) according to each benchmark set. The conversion methods are referred to as c1 for Berbeglia et al. (2012) and c2 for Wong et al. (2014). The instances provided by Schilde, Doerner, and Hartl (2014) are already dynamic. Therefore, no conversion is needed. However, since those instances are also stochastic (each request has an associated probability of happening), the reported values are also out of 10 runs.

Table A.5 shows that Schilde et al. (2014) have the set with the largest instances, with an average of almost 400 requests per instance. The smallest instance provided by Schilde et al. (2014) (135 requests) is larger than the average of all other three sets. Meanwhile, Cordeau (2006) has the smallest benchmark set, with an average of around 46 requests. The obtained dynamism ranges from smaller to higher values for both c1 and c2. The average and standard deviation point out that most instances stay in the range of 30% to 60% dynamism using these conversion methods. Instances by Schilde et al. (2014), which are already dynamic, have a low variety in dynamism values with a range from 37.52% to 54.50%.

Urgency values reported in Table A.6 show that c2 can produce instances with a wider range of urgency, ranging from 15 min of the mean to values as high as 700. The instances of Schilde et al. (2014) follow a pattern of having reaction times around 60 min. Finally, geographic dispersion values are reported in Table A.7. Note that geographic dispersion is not dependent on the instance being dynamic, so no conversion method is necessary. We use $th_s = 10$ minutes and $n = 5$. The highest geographic dispersion is seen in the instances of Schilde et al. (2014), which can be explained by the fact that they are based on a real-world scenario. The reported standard deviations in Cordeau (2006), Cordeau and Laporte (2003), and Schilde et al. (2014) show that trip sizes stay mostly consistently in the same range, as all their standard deviations are below 1 min. Instances of Braekers and Kovacs (2016) have a slightly higher deviation of 3.21 min.

Table A.5
Values of size and dynamism for benchmark instances proposed in the literature.

Reference	Convert	Size				Dynamism %			
		min	max	avg	std	min	max	avg	std
Cordeau and Laporte (2003)	c1	24	144	86.40	40.80	10.87	93.39	46.84	8.16
	c2					0.00	98.33	28.63	15.63
Cordeau (2006)	c1	16	96	46.16	22.84	2.29	100.00	54.63	12.93
	c2					0.00	97.11	31.43	15.83
Schilde et al. (2014)	n.a.	135	647	387.60	165.80	37.52	54.50	44.64	2.86
Braekers and Kovacs (2016)	c1	10	252	82.30	58.96	13.71	72.85	40.02	6.61
	c2					6.06	95.00	42.17	8.62

Table A.6
Values of urgency for benchmark instances proposed in the literature.

Reference	Convert	Urgency $\overline{\chi}$				Urgency χ_s			
		min	max	avg	std	min	max	avg	std
Cordeau and Laporte (2003)	c1	71.11	259.44	157.66	14.75	0.00	80.95	48.39	10.13
	c2	17.00	544.84	221.4	67.41	0.00	220.29	109.95	31.39
Cordeau (2006)	c1	63.67	260.31	153.72	19.32	0.00	87.43	45.67	13.72
	c2	15.00	693.28	227.92	97.23	0.00	305.25	124.91	55.60
Schilde et al. (2014)	n.a.	60.74	67.62	63.97	1.39	17.17	22.78	19.43	1.05
Braekers and Kovacs (2016)	c1	113.90	202.38	158.02	9.71	15.41	75.58	50.61	4.98
	c2	15.00	616.50	115.87	44.56	0.00	298.63	132.86	38.67

Table A.7
Values of geographic dispersion for benchmark instances proposed in the literature.

Reference	Geographic dispersion			
	min	max	avg	std
Cordeau and Laporte (2003)	7.27	10.32	8.7	0.92
Cordeau (2006)	11.61	15.66	14.15	0.93
Schilde et al. (2014)	17.42	21.52	18.88	0.96
Braekers and Kovacs (2016)	2.30	16.64	9.89	3.21

Table A.8
Values of similarity for benchmark instances proposed in the literature.

Reference	Similarity %			
	min	max	avg	std
Cordeau and Laporte (2003)	27.43	60.94	49.01	11.77
Cordeau (2006)	0.00	21.48	9.17	5.12
Schilde et al. (2014)	62.09	69.30	65.70	3.60
Braekers and Kovacs (2016)	0.00	48.50	11.36	7.87

Similarity values are reported in Table A.8. We consider threshold levels, in minutes, to be $th_{tt} = 2$, $th_{ts} = 1$, and $th_e = 1$. Cordeau (2006)'s benchmark is shown to be the most diverse, with reported values for similarity ranging between 0% and 21.48%. Instances with the highest similarity are found in the work by Schilde et al. (2014).

References

Alonso-Mora, J., Samaranayake, S., Wallar, A., Frazzoli, E., & Rus, D. (2017). On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of the National Academy of Sciences*, *114*(3), 462–467.

Atasoy, B., Ikeda, T., Song, X., & Ben-Akiva, M. E. (2015). The concept and impact analysis of a flexible mobility on demand system. *Transportation Research Part C (Emerging Technologies)*, *56*, 373–392.

Azadeh, S. S., Atasoy, B., Ben-Akiva, M. E., Bierlaire, M., & Maknoon, M. (2022). Choice-driven dial-a-ride problem for demand responsive mobility service. *Transportation Research, Part B (Methodological)*, *161*, 128–149.

Barbosa, H., Barthelemy, M., Ghoshal, G., James, C. R., Lenormand, M., Louail, T., et al. (2018). Human mobility: Models and applications. *Physics Reports*, *734*, 1–74.

Berbeglia, G., Cordeau, J.-F., & Laporte, G. (2012). A hybrid tabu search and constraint programming algorithm for the dynamic dial-a-ride problem. *INFORMS Journal on Computing*, *24*(3), 343–355.

Bischoff, J., Maciejewski, M., & Nagel, K. (2017). City-wide shared taxis: A simulation study in berlin. In *2017 IEEE 20th international conference on intelligent transportation systems* (pp. 275–280). IEEE.

Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, *65*, 126–139.

Borndörfer, R., Grötschel, M., Klostermeier, F., & Küttner, C. (1999). Telebus berlin: Vehicle scheduling in a dial-a-ride system. In *Computer-aided transit scheduling* (pp. 391–422). Springer.

Braekers, K., & Kovacs, A. A. (2016). A multi-period dial-a-ride problem with driver consistency. *Transportation Research, Part B (Methodological)*, *94*, 355–377.

Chicago Department of Business Affairs & Consumer Protection (2021). Transportation network providers - trips. Chicago Data Portal, Online reference, https://data.cityofchicago.org/. (Last Access on 21 July 2021).

Cirasella, J., Johnson, D. S., McGeoch, L. A., & Zhang, W. (2001). The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Workshop on algorithm engineering and experimentation* (pp. 32–59). Springer.

Cokelaer, T. (2022). FITTER package. Online reference, https://fitter.readthedocs.io/en/latest/. (Last Access on 04 April 2022).

Cordeau, J.-F. (2006). A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, *54*(3), 573–586.

Cordeau, J.-F., & Laporte, G. (2003). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research, Part B (Methodological)*, *37*(6), 579–594.

Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem: Models and algorithms. *Annals of Operations Research*, *153*(1), 29–46.

Czioska, P., Kutadinata, R., Trifunović, A., Winter, S., Sester, M., & Friedrich, B. (2019). Real-world meeting points for shared demand-responsive transportation systems. *Public Transport*, *11*(2), 341–377.

Drakoulis, R., Bellotti, F., Bakas, I., Berta, R., Paranthaman, P. K., Dange, G. R., et al. (2018). A gamified flexible transportation service for on-demand public transport. *IEEE Transactions on Intelligent Transportation Systems*, *19*(3), 921–933.

Dubois, F. (2022). *The vehicle routing problem for flash floods relief operations* (Ph.D. thesis), Université Paul Sabatier-Toulouse III.

Engelen, L. (2018). The same-day pickup and delivery problem.

Fielbaum, A., & Alonso-Mora, J. (2020). Unreliability in ridesharing systems: Measuring changes in users' times due to new requests. *Transportation Research Part C (Emerging Technologies)*, *121*, Article 102831.

Galarza Montenegro, B. D., Sörensen, K., & Vansteenwegen, P. (2023). The real-time dynamic online feeder service with a maximum headway at mandatory stops. *Transportmetrica A: Transport Science*, Article 2227738.

Garey, M. R., & Johnson, D. S. (1979). A guide to the theory of NP-completeness. *Computers and Intractability*, 641–650.

Gkiotsalitis, K., Wu, Z., & Cats, O. (2019). A cost-minimization model for bus fleet allocation featuring the tactical generation of short-turning and interlining options. *Transportation Research Part C (Emerging Technologies)*, *98*, 14–36.

Guo, R., Guan, W., & Zhang, W. (2018). Route design problem of customized buses: Mixed integer programming model and case study. *Journal of Transportation Engineering, Part A: Systems*, *144*(11), Article 04018069.

Hagberg, A., Swart, P., & S. Chult, D. (2008). *Exploring network structure, dynamics, and function using NetworkX: Tech. rep.*, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

Häll, C. H., Andersson, H., Lundgren, J. T., & Värbrand, P. (2009). The integrated dial-a-ride problem. *Public Transport*, *1*(1), 39–54.

Hörl, S. (2017). Agent-based simulation of autonomous taxi services with dynamic demand responses. *Procedia Computer Science*, *109*, 899–904.

Hyland, M., & Mahmassani, H. S. (2020). Operational benefits and challenges of shared-ride automated mobility-on-demand services. *Transportation Research Part A: Policy and Practice*, *134*, 251–270.

Jäger, B., Brickwedde, C., & Lienkamp, M. (2018). Multi-agent simulation of a demand-responsive transit system operated by autonomous vehicles. *Transportation Research Record*, *2672*(8), 764–774.

Karami, F., Vancroonenburg, W., & Vanden Berghe, G. (2020). A periodic optimization approach to dynamic pickup and delivery problems with time windows. *Journal of Scheduling*, *23*(6), 711–731.

Kilby, P., Prosser, P., & Shaw, P. (1998). *vol. 1*, *Dynamic VRPs: A study of scenarios*: *Technical report*, (no. 11), University of Strathclyde.

Koh, K., Ng, C., Pan, D., & Mak, K. S. (2018). Dynamic bus routing: A study on the viability of on-demand high-capacity ridesharing as an alternative to fixed-route buses in Singapore. In *2018 21st international conference on intelligent transportation systems* (pp. 34–40). IEEE.

Kujala, R., Weckström, C., Darst, R. K., Mladenović, M. N., & Saramäki, J. (2018). A collection of public transport network data sets for 25 cities. *Scientific Data*, *5*, Article 180089.

Larsen, A., Madsen, O., & Solomon, M. (2002). Partially dynamic vehicle routing—models and algorithms. *Journal of the Operational Research Society*, *53*(6), 637–646.

Lee, Y.-J., Meskar, M., Nickkar, A., & Sahebi, S. (2019). Development of an algorithm for optimal demand responsive relocatable feeder transit networks serving multiple trains and stations. *Urban Rail Transit*, *5*(3), 186–201.

Leeftink, G., & Hans, E. W. (2018). Case mix classification and a benchmark set for surgery scheduling. *Journal of Scheduling*, *21*(1), 17–33.

Lian, Y., Lucas, F., & Sörensen, K. (2023). The electric on-demand bus routing problem with partial charging and nonlinear function. *Transportation Research Part C (Emerging Technologies)*, *157*, Article 104368.

Lian, Y., Lucas, F., & Sörensen, K. (2024). Prepositioning can improve the performance of a dynamic stochastic on-demand public bus system. *European Journal of Operational Research*, *312*(1), 338–356.

Liu, Y., Kang, C., Gao, S., Xiao, Y., & Tian, Y. (2012). Understanding intra-urban trip patterns from taxi trajectory data. *Journal of Geographical Systems*, *14*(4), 463–483.

Liu, M., Singh, H. K., & Ray, T. (2014). Application specific instance generator and a memetic algorithm for capacitated arc routing problems. *Transportation Research Part C (Emerging Technologies)*, *43*, 249–266.

Liu, L., Sun, L., Chen, Y., & Ma, X. (2019). Optimizing fleet size and scheduling of feeder transit services considering the influence of bike-sharing systems. *Journal of Cleaner Production*, *236*, Article 117550.

Lu, X., Yu, J., Yang, X., Pan, S., & Zou, N. (2016). Flexible feeder transit route design to enhance service accessibility in urban area. *Journal of Advanced Transportation*, *50*(4), 507–521.

Lund, K., Madsen, O. B., & Rygaard, J. M. (1996). *Vehicle routing problems with varying degrees of dynamism*. IMM, Institute of Mathematical Modelling, Technical University of Denmark.

Ma, S., Zheng, Y., & Wolfson, O. (2013). T-share: A large-scale dynamic taxi ridesharing service. In *2013 IEEE 29th international conference on data engineering* (pp. 410–421). IEEE.

Maggioni, F., Perboli, G., & Tadei, R. (2014). The multi-path traveling salesman problem with stochastic travel costs: Building realistic instances for city logistics applications. *Transportation Research Procedia*, *3*, 528–536.

Melis, L., & Sörensen, K. (2020). *The on-demand bus routing problem: A large neighborhood search heuristic for a dial-a-ride problem with bus station assignment*: *Tech. rep.*, University of Antwerp, Faculty of Business and Economics.

Melis, L., & Sörensen, K. (2022). The real-time on-demand bus routing problem: The cost of dynamic requests. *Computers & Operations Research*, *147*, Article 105941.

Navidi, Z., Ronald, N., & Winter, S. (2018). Comparison between ad-hoc demand responsive and conventional transit: A simulation study. *Public Transport*, *10*(1), 147–167.

Noulas, A., Scellato, S., Lambiotte, R., Pontil, M., & Mascolo, C. (2012). A tale of many cities: Universal patterns in human urban mobility. *PLoS One*, *7*(5), Article e37027.

O'Sullivan, D. (2014). Spatial network analysis. In M. M. Fischer, & P. Nijkamp (Eds.), *Handbook of regional science* (pp. 1253–1273). Berlin, Heidelberg: Springer Berlin Heidelberg, ISBN: 978-3-642-23430-9, http://dx.doi.org/10.1007/978-3-642-23430-9_67.

Pandey, V., Monteil, J., Gambella, C., & Simonetto, A. (2019). On the needs for maas platforms to handle competition in ridesharing mobility. *Transportation Research Part C (Emerging Technologies)*, *108*, 269–288.

Pei, M., Lin, P., Liu, R., & Ma, Y. (2019). Flexible transit routing model considering passengers' willingness to pay. *IET Intelligent Transport Systems*, *13*(5), 841–850.

Pellegrini, P., & Birattari, M. (2005). *Instances generator for the vehicle routing problem with stochastic demand*: *Tech. rep.*, Citeseer.

Pillac, V., Gendreau, M., Guéret, C., & Medaglia, A. L. (2013). A review of dynamic vehicle routing problems. *European Journal of Operational Research*, *225*(1), 1–11.

Qiu, F., Li, W., & Haghani, A. (2015). An exploration of the demand limit for flex-route as feeder transit services: A case study in Salt Lake City. *Public Transport*, *7*(2), 259–276.

Rardin, R. L., Tovey, C. A., & Pilcher, M. G. (1993). Analysis of a random cut test instance generator for the tsp. In *Complexity in numerical optimization* (pp. 387–405). World Scientific.

Reyes, D., Erera, A., Savelsbergh, M., Sahasrabudhe, S., & O'Neil, R. (2018). The meal delivery routing problem. *Optimization Online*.

Ronald, N., Thompson, R. G., & Winter, S. (2015). Comparison of constrained and ad hoc demand-responsive transportation systems. *Transportation Research Record*, *2563*(1), 44–51.

Santos, D. O., & Xavier, E. C. (2015). Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive. *Expert Systems with Applications*, *42*(19), 6728–6737.

Schilde, M., Doerner, K. F., & Hartl, R. F. (2014). Integrating stochastic time-dependent travel speed in solution methods for the dynamic dial-a-ride problem. *European Journal of Operational Research*, *238*(1), 18–30.

Simonetto, A., Monteil, J., & Gambella, C. (2019). Real-time city-scale ridesharing via linear assignment problems. *Transportation Research Part C (Emerging Technologies)*, *101*, 208–232.

Song, C., Qu, Z., Blumm, N., & Barabási, A.-L. (2010). Limits of predictability in human mobility. *Science*, *327*(5968), 1018–1021.

Stouffer, S. A. (1940). Intervening opportunities: A theory relating mobility and distance. *American Sociological Review*, *5*(6), 845–867.

Ullrich, M., Weise, T., Awasthi, A., & Lässig, J. (2018). A generic problem instance generator for discrete optimization problems. In *Proceedings of the genetic and evolutionary computation conference companion* (pp. 1761–1768).

Vallée, S., Oulamara, A., & Cherif-Khettaf, W. R. (2017). Maximizing the number of served requests in an online shared transport system by solving a dynamic DARP. In *International conference on computational logistics* (pp. 64–78). Springer.

Van Lon, R. R., Branke, J., & Holvoet, T. (2018). Optimizing agents with genetic programming: An evaluation of hyper-heuristics in dynamic real-time logistics. *Genetic Programming and Evolvable Machines*, *19*(1), 93–120.

Van Lon, R. R., Ferrante, E., Turgut, A. E., Wenseleers, T., Berghe, G. V., & Holvoet, T. (2016). Measures of dynamism and urgency in logistics. *European Journal of Operational Research*, *253*(3), 614–624.

Vanhoucke, M., & Maenhout, B. (2007). Nsplib–a nurse scheduling problem library: A tool to evaluate (meta-) heuristic procedures. In *Operational research for health policy: Making better decisions, proceedings of the 31st annual meeting of the working group on operations research applied to health services* (pp. 151–165).

Vansteenwegen, P., Melis, L., Aktaş, D., Montenegro, B. D. G., Vieira, F. S., & Sörensen, K. (2022). A survey on demand-responsive public bus systems. *Transportation Research Part C (Emerging Technologies)*, *137*, Article 103573.

Wang, S., Correia, G. H. d., & Lin, H. X. (2019). Exploring the performance of different on-demand transit services provided by a fleet of shared automated vehicles: An agent-based model. *Journal of Advanced Transportation*, *2019*.

West, D. B., et al. (2001). *Introduction to graph theory*: *vol. 2*, Prentice hall Upper Saddle River.

Winter, K., Cats, O., Correia, G. H. d. A., & van Arem, B. (2016). Designing an automated demand-responsive transport system: Fleet size and performance analysis for a campus–Train station service. *Transportation Research Record*, *2542*(1), 75–83.

Winter, K., Cats, O., Correia, G., & van Arem, B. (2018). Performance analysis and fleet requirements of automated demand-responsive transport systems as an urban public transport service. *International Journal of Transportation Science and Technology*, *7*(2), 151–167.

Wong, K.-I., Han, A., & Yuen, C. (2014). On dynamic demand responsive transport services with degree of dynamism. *Transportmetrica A: Transport Science*, *10*(1), 55–73.