



HAL
open science

Instance generation tool for on-demand transportation problems

Michell Queiroz, Flavien Lucas, Kenneth Sörensen

► **To cite this version:**

Michell Queiroz, Flavien Lucas, Kenneth Sörensen. Instance generation tool for on-demand transportation problems. *European Journal of Operational Research*, 2024, 10.1016/j.ejor.2024.03.006 . hal-04504887

HAL Id: hal-04504887

<https://hal.science/hal-04504887>

Submitted on 9 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Instance generation tool for on-demand transportation problems

Michell Queiroz ^{*} Flavien Lucas [†] Kenneth Sörensen [‡]

August 26, 2022

Abstract

We present REQreate, a tool to generate instances for on-demand transportation problems. Such problems consist of optimizing the routes of vehicles according to passengers’ demand for transportation under space and time restrictions (requests). REQreate is flexible and can be configured to generate instances for a large number of problems in this problem class. In this paper, we demonstrate this by generating instances for the Dial-a-Ride Problem (DARP) and On-demand Bus Routing Problem (ODBRP). In most of the literature, researchers either test their algorithms with instances based on artificial networks or 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.

The aim of REQreate is to overcome these non-realistic and overfitting shortcomings. Rather than relying on either 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 catalogue 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 datasets 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 diverse benchmark sets, in addition to properties (size, dynamism, urgency and geographic dispersion) that could be used to comprehend what affects the performance of algorithms.

Keywords: transportation, instance generator, on-demand public transport, REQreate

1 Introduction

REQreate is a tool designed with the main objective to generate instances for on-demand transportation problems. Such problems consist of optimizing the routes of vehicles according to passengers’ demands under space and time restrictions. Recent years have seen 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). Many of these problems are NP-hard (Garey & Johnson, 1979), and the computational effort to optimally solve them often grows exponentially with the size of the instance input. Appropriately, optimization techniques have been proposed to be effective and feasible alternatives for solving these problems.

Instances are the core for evaluating the developed methods for optimization problems. Initially, the instances are essential because they can demonstrate that the method works. Subsequently, they can be

^{*}Department of Engineering Management, University of Antwerp Operations Research Group (ANT/OR), Prinsstraat 13, 2000, Antwerp, Belgium. (michell.queiroz@uantwerpen.be). Corresponding author.

[†]CERI Numeric Systems, IMT Nord Europe, Institut Mines-Télécom, Univ. Lille, Centre for Digital Systems, F-59000 Lille, France. (flavien.lucas@imt-nord-europe.fr)

[‡]Department of Engineering Management, University of Antwerp Operations Research Group (ANT/OR), Prinsstraat 13, 2000, Antwerp, Belgium. (kenneth.sorensen@uantwerpen.be)

used to test how sensitive the approach is to its parameters. Here it is important that the experiments demonstrate that the method is robust and presents good results for different types of instances. Additionally, instances are important to discover the limitations of the approach. A method should be tested both with easy and with 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. Lastly, instances are used to compare results with previous approaches, and in some sense, prove that one is better than the other. This is a particular challenge for on-demand public transportation problems which as been shown in a recent survey by [Vansteenwegen et al. \(2022\)](#), who conclude that standard benchmark sets for these problems generally do not exist. This lack of structure complicates any comparison between approaches. Given these previously mentioned goals, we present REQreate, an instance generation tool for on-demand transportation problems based on real-world networks from OpenStreetMaps. Besides using real-life networks, we will also provide a method to generate requests with different urban mobility properties that result in various scenarios, which the authors believe overcomes the disadvantage of artificially generating data.

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. For example, in the case of the Dial-a-Ride Problem (DARP) attributes may include: a) origin location; b) destination location; c) time window for departure; d) time window for arrival; and e) requirement for resources, such as wheelchair or stretcher. Furthermore, the relations between attributes might also constitute a part of the instance, e.g., the travel time/distance matrix between the set of origin and destination locations described in the DARP attributes.

The tool can assist researchers in testing their optimization algorithms by providing an easy and efficient way to create instances of varying types and sizes. The difficulty in obtaining real-world data is the main reason we decided for a randomly generated approach. There are several reasons to prefer artificially generated instances over real-life data. First, real-life data is difficult to obtain, as companies that implement a specific problem tackled by a researcher are usually not willing to share real cases. The main motives in maintaining such information confidential are to preserve user privacy and gain competitive leverage. One possibility to overcome this issue is to perform large and different surveys, although complexity and the likelihood of the outcome differs from reality are major drawbacks. Second, when an on-demand transportation problem does not yet exist in practice, obtaining real-world data might be impossible. Third, supply for on-demand transportation creates demand, and the real-life data of today might not be representative of tomorrow’s situation. Indeed, in [Gkiotsalitis & Stathopoulos \(2016\)](#), the authors take into account non-recurrent trips, e.g. leisure, which can account for more than half of trips in some cities, to improve the operations of a demand responsive public transportation system.

We developed the tool to be flexible and generic as possible. By providing a configuration file in JavaScript Object Notation (JSON) format, it allows to generate instances for a diversity of problems. Such problems include the Dial-a-Ride Problem (DARP), On-demand Bus Routing Problem (ODBRP), School Bus Routing Problem (SBRP), and many others. Considering that new problems emerge constantly, and 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.

In summary, REQreate was developed with the capability to generate benchmark instance sets that satisfy the following requirements: size, diversity, extensibility, and realism ([Vanhoucke & Maenhout, 2007](#)). In this paper, besides presenting the tool, we generate instances and analyze their properties. Two problems are used to demonstrate the potential of REQreate: the DARP and the ODBRP. We discuss how the parameters can be easily tuned to vary the size of instances from small to large, which allows computational studies to derive meaningful conclusions regarding the limitations of the approach, as it is common for larger instances to place a higher computational burden. Moreover, properties of the instances such as size, dynamism, urgency and geographic dispersion are described.

The diversity of a benchmark set can be evaluated with a measure called instance proximity, similar to an approach introduced in [Leefink & Hans \(2018\)](#). Instances are shown to be easily extensible by including or modifying the provided attributes and parameters in the configuration file. Furthermore, the transportation

research community is motivated by real-world problems, therefore instances should ideally resemble realistic scenarios. We demonstrate how the synthetic instances compare with real data from rideshare companies trips.

The remainder of this paper is organized as follows. Section 2 presents a literature review on the topic of instance generation. Sections 3 and 4 describe the processes to retrieve realistic networks and request generation, respectively. Section 5 outlines the simple method that can be included in the request generation to produce different urban mobility patterns. Section 6 introduces formal notations for the DARP and ODBRP used throughout the paper. Section 7 describes instance properties: size, dynamism, urgency and geographic dispersion. Section 8 introduces the concept of instance similarity. The analysis between real data and synthetic instances is performed in Section 9. Ultimately, final remarks are considered in Section 11.

2 Literature Review

Throughout the years, tools have been successfully implemented to generate instances for a diversity of problems. Rardin, Tovey, & Pilcher (1993) introduce an instance generator for the Traveling Salesman Problem (TSP). ProGen is a well-known instance generator for precedence and resource-constrained project scheduling problems. The project is described as a network where nodes represent jobs (tasks) and arcs the precedence relations. Jobs may use a set of available, often scarce, resources. ProGen is presented by Kolisch, Sprecher, & Drexel (1995), and it is based on concepts such as the topology of the network and resource availability to distinguish between easy and hard instances. Drexel, Nissen, Patterson, & Salewski (2000) introduce an extension of the aforementioned generator primarily aimed to incorporate labor time regulations, making it suitable to generate instances for problems that involve assigning individuals to a number of jobs. Other instance generators have been also proposed for project scheduling problems, such as DANGEN and RanGen (Agrawal, Elmaghraby, & Herroelen, 1996; Demeulemeester, Vanhoucke, & Herroelen, 2003). Both generators introduce measures of complexity for the instances. However, RanGen employs a wide range of parameters related to resources and network topology to overcome the shortcomings of previous generators.

Cirasella, Johnson, McGeoch, & Zhang (2001) introduce new random instance generators for the Asymmetric Traveling Salesman Problem (ATSP). In instances for the ATSP, the distances of moving back and forth between a pair of cities are not always the same. The authors model instances based on real-world applications such as Stacker Crane and Common Superstring Problems. Ultimately, they compare the performance of different algorithms and conclude that there is no dominance over all instance classes.

Pellegrini & Birattari (2005) present a procedure to generate instances for the Vehicle Routing Problem with Stochastic Demand (VRPSD). Like the Vehicle Routing Problem (VRP), the VRPSD consists of minimize costs while meeting the requirements for delivery of a set of customers. The distinctive characteristic is that before reaching the customer, only a probability distribution of the demand is known. Creation of the instances are supported by databases with information of population and distances between cities from European countries. Scenarios for concentration of customers are inspired by the location of retail stores in European cities, which are proportional to the number of citizens. Demand for each customer can be generated using uniform and Bernoulli probability distributions.

Differently from targeting on specific classes of optimization problems, instance generators that focus on providing instances that have particular structural features are referred to as landscape generators. The geometric features of the landscapes are influenced by given parameters. Accordingly, Gallagher & Yuan (2006) propose a landscape generator for continuous, bound-constrained optimization problems. The authors argue that useful conclusions on the performance of heuristic algorithms can be draw by conducting experiments on the landscape space. Other methods for obtaining desired landscapes can be found in the literature (Morrison & De Jong, 1999; Michalewicz, Deb, Schmidt, & Stidsen, 2000), including the generator proposed by Hernando, Mendiburu, & Lozano (2015), which allows to create instances with controlled properties for permutation-based COPs, such as a fixed number of local optima.

De Corte & Sörensen (2014) draw motivation from the absence of high-quality benchmark networks for Water Distribution Network Design (WDND), and propose HydroGen, a tool to generate artificial Water

Distribution Networks (WDNs) varying in size and characteristics. The tool provided by the authors allow creating networks with the following distinguished characteristics, which are featured in realistic scenarios: tree-like versus looped network structures; densely populated versus rural areas; domestic versus industrial demand nodes, among others. The artificially generated WDNs with HydroGen are compared with real ones, and according to graph-theoretical indices, they are showed to have a high resemblance.

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 [Maggioni, Perboli, & Tadei \(2014\)](#). The authors incorporate real data produced by traffic sensors networks from the city of Turin in Italy, and consider the multi-path TSP with stochastic travel costs to test their technique.

[M. Liu, Singh, & Ray \(2014\)](#) present an instance generator and a memetic algorithm for the Capacitated Arc Routing Problem (CARP). The authors express the need to evaluate the performance of algorithms on classes of instances that resemble realistic scenarios, such as inspection of electric power lines, garbage collection and winter gritting. Their generator controls density, connectedness, degree and distance distribution of the underlying road network. Instances can be further customized by tuning the demand distribution depending on the application. As an example, for the garbage collection problem, the waste amount to be collected can be set as a function of the arc distance combined with population density.

[Macedo & Tchemisova \(2017\)](#) provides an additional application of instance generators. The authors apply the generator to construct non-regular instances for Semidefinite Programming (SDP). “SDP refers to convex optimization problems where a linear function is minimized subject to constraints in the form of linear matrix inequalities”, as defined by [Macedo & Tchemisova \(2017\)](#). The authors conduct numerical experiments with the most popular SDP solvers at the time, and discuss the poor efficacy when applied to non-regular instances.

[Leeftink & Hans \(2018\)](#) 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 their expected duration differs less than 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.

[Ullrich, Weise, Awasthi, & Lässig \(2018\)](#) attempt to overcome 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 evidence 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 so 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.

To the best of our knowledge, this is the first tool that generate instances for an extensive catalogue of existing and upcoming on-demand transportation problems. REQreate is also pioneer amongst instance generators to make use of real-life networks. These networks can be obtained from OpenStreetMaps (OSM), an open-source collaborative mapping project. OpenStreetMaps is consistently updated and has global coverage, thus is widely used in the transportation literature to perform real-life case studies. For example, [Dingil, Schweizer, Rupi, & Stasiskiene \(2018\)](#) perform an analysis and comparison of transport indicators to conduct an evaluation of different transport strategies used in 151 urban areas. They use open source data from OSM, and present results specially on the correlation of infrastructure and congestion levels. In [Navidi, Ronald, & Winter \(2018\)](#), the authors also use OSM to extract a real-world network used in the simulation study. Results lead to the conclusion of superiority of the demand responsive transit system, such as having the benefit of reducing passengers’ perceived travel time. Likewise, [Drakoulis et al. \(2018\)](#) import the network from “Trikala, Greece” using OSM. The authors study the implementation of an on-demand public bus transportation service on the city.

The instance generation approach described in this paper is the product of attributes, parameters, expressions, and constraints. A similar approach is presented in Ullrich, Weise, Awasthi, & Lässig (2018). However, the authors focus more on a generic strategy instead of realism. The tool described in this paper also differs from previous approaches by offering a diverse set of combinations for parameters and attributes that approximately control the output of instance properties. REQreate is available on GitHub¹.

3 Retrieving realistic networks

In the literature on on-demand public transport problems, researchers frequently test their algorithms with instances based on artificial networks or 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 to provide an easy way to create benchmark instances with realistic networks, we make use of the OSMnx (Boeing, 2017) package to retrieve and analyze real-world street networks from OpenStreetMaps (OSM). By simply providing a string, i.e., the name of the area, OSMnx downloads information on the boundaries of the area and creates a graph. Further computations done by REQreate, e.g., distance/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.

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 Figure 1, where the driving (Figure 1(a)) and walking (Figure 1(b)) networks of a neighborhood (“South Lawndale”) in “Chicago, Illinois” are obtained.

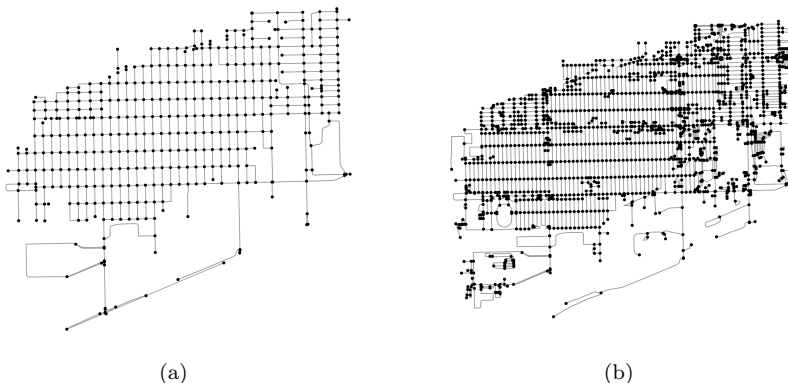


Figure 1: Figures 1(a) and 1(b) depict the driving and walking networks of “South Lawndale, Chicago, Illinois”, respectively. Note how they are significantly different, as the walking network has more nodes and arcs when compared to the driving one.

¹<https://github.com/michellqueiroz-ua/instance-generator>

Bus stations that are within the boundaries of the given area are also obtained using a built in function of OSMnx. This is done because several problems can make use of 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 Figure 2. Pairing locations between networks is possible with the conversion of geographic coordinates to the nearest node on each respective network. For example, this finds useful when a passenger is assigned to be picked up by a bus, therefore, there is a requirement to determine where approximately the bus should stop (driving network) and the place the passenger needs to wait (walking network).

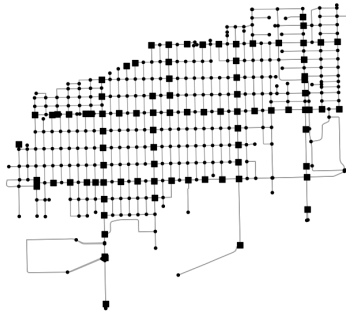


Figure 2: Location of bus stations on the driving network of “South Lawndale, Chicago, Illinois” represented as black squares.

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, frequency or estimated travel time. Therefore, the collection of data presented in Kujala, Weckström, Darst, Mladenović, & 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 drive network, which requires a speed value. We assume the driving speed to be fixed, which means that the travel times are not dependent on hour of 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} be the distance and maximum speed 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 “speed factor”, is a value in the interval $]0,1]$ determined by the user, and is used to calculate a realistic speed for each arc, which is usually far from the maximum. The definition of α is demonstrated in Section 4. It is also possible to replace ms_{uv} by an equal speed value to all arcs.

Travel times are important in order to measure, for example, assuming a door-to-door service, the cost of driving a passenger from their origin to the destination. Another case is to assess the feasibility of assigning passengers to nearby bus stations. Therefore, we build the travel time matrix using Dijkstra’s algorithm built in function of the NetworkX package by computing the shortest paths between the nodes of the network using travel times as the weight. Finally, OSMnx is also used to download information of school’s locations. This data is mostly relevant to generate instances for the SBRP.

4 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, Weise, Awasthi, & Lässig (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. In this section, we outline the values that should be provided in order to generate requests.

4.1 JSON syntax introduction

JSON files consists 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*.

Listing 1: Configuration file structure

```
1 {
2   "network": "Chicago, Illinois",
3   "seed": 100,
4   ...
5   "places": [...],
6   "parameters": [
7     {
8       "name": "simple_parameter",
9       "type": "integer",
10      "value": 10
11    },
12    {...}
13  ],
14  "attributes": [...]
```

4.2 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 3. 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 3, can be specified with *max_speed_factor*.

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.

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

```

1 { "network": "Chicago, Illinois",
2   "seed": 100,
3   "problem": "DARP",
4   "fixed_lines": true,
5   "max_speed_factor": 0.5,
6   "replicas": 10,
7   "requests": 500,
8   "instance_filename": ["network", "problem", "requests"],
9   ...
10 }
```

4.3 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. Tables 1 depicts the potential sub-items for *places*. The first column represent 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* (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.

Table 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”

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 raises an error otherwise. The polygon of “zone_representation” is established by fixing *length_lon* and *length_lat* to 1,000 meters. Lastly, “zone_center” is an area stipulated as a circumference with *radius* of 1,500 meters, and *centroid* set to “true”.

Listing 3: Examples for item *places*

```

1 {
2   ...
3   "places": [
4     {
5       "name": "location_example",
6       "type": "location",
7       "lon": -1.6457340723441525,
8       "lat": 48.100199454954804
9     },
10    {
11      "name": "zone_representation",
12      "type": "zone",
13      "lon": -1.6902891077472344,
14      "lat": 48.09282962664396,
15      "length_lon": 1000,
16      "length_lat": 1000,
17      "length_unit": "m"
18    },
19    {
20      "name": "zone_center",
21      "type": "zone",
22      "centroid": true,
23      "radius": 1500,
24      "length_unit": "m"
25    }
26  ],
27  ...

```

4.4 Parameters

Item *parameters* is also an array of objects. Table 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”.

Table 2: Summary for *parameters*

sub-items	possible values
name	string
type	“string”, “integer”, “real”, “array_primitives”, “array_locations”, “array_zones”
value	<i>type</i> dependent
time_unit	”s”, ”min”, ”h”
length_unit	”m”, ”km”, ”mi”
speed_unit	”mps”, ”kmh”, ”miph”
size	integer
locs	”schools”, ”random”

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. We

remark 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 value 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*. 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 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. Furthermore, “pair_zones” lists 2 zones: “zone_representation” and “zone_center”, both defined in Listing 3.

Listing 4: Example for item *parameters*

```

1 {
2   ...
3   "parameters":[
4     {
5       "name": "min_early_departure",
6       "type": "integer",
7       "value": 5,
8       "time_unit": "h"
9     },
10    {
11      "name": "many_locations",
12      "type": "array_location",
13      "value": ["location_example"],
14      "size": 3,
15      "locs": "random"
16    },
17    {
18      "name": "pair_zones",
19      "type": "array_zones",
20      "value": ["zone_representation", "zone_center"],
21      "size": 2
22    }
23  ],
24  ...

```

4.5 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 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 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 an 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 documentation². The declared values must be in

²<https://docs.scipy.org/doc/scipy/index.html>

conformity with the 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.

Table 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 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

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 a method later explained in Section 5 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 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 an 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 30600 seconds (8:30), standard deviation of 3600 seconds (1 hour), 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.

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

```

1 {
2   ...
3   "attributes": [
4     {
5       "name": "earliest_departure",
6       "type": "integer",
7       "time_unit": "s",
8       "pdf": {
9         "type": "normal",
10        "loc": 30600,
11        "scale": 3600
12      },
13      "constraints": [ "earliest_departure >= min_early_departure" ]
14    },
15    {
16      "name": "latest_arrival",
17      "type": "integer",
18      "time_unit": "s",
19      "expression": "earliest_departure + 1800"
20    },
21    {
22      "name": "origin",
23      "type": "location"
24    },
25    {
26      "name": "destination",
27      "type": "location",
28      "subset_zones": "pair_zones",
29      "weights": [1, 3]
30    }
31  ],
32  "travel_time_matrix": ["origin", "destination"]

```

4.6 Invoking method

The set of instances will be generated after invoking a method with the configuration file name as an argument (see Listing 6). The generator creates one request at a time. Analogous to the technique described in Ullrich et al. (2018), a directed graph is built with attributes representing nodes, and arcs defined by expressions or constraints. Then, the attributes (nodes) are topologically sorted in conformity with their dependencies (expressions and constraints). Initially, in accordance with the topological ordering, a value for an attribute is generated. Furthermore, the feasibility is checked based on the disclosed constraints and if any of them is violated, the procedure restarts by discarding the current attribute or the entire request. This process is repeated until all attributes for a request are valid. The instance is then completed upon the given number in *requests* is accomplished. We remark that is beyond the scope of the tool to certify if the set of constraints creates dependencies that can not be met, i.e., every possible combination of attribute values are infeasible.

Thus, after a large number of unsuccessful iterations, the generator will halt, raising an error informing it was unable to meet the requirements.

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

```
import input_json

input_json('example_config.json')
_
```

5 Urban mobility patterns

In this section we describe the method to generate various mobility patterns, which can be defined by the different probabilities of a place serving as origin and/or destination of a trip. Previous literature has revealed significant regularity in human mobility patterns, which directly affects process driven by it, such as urban planning, traffic engineering and even epidemic modeling (Song, Qu, Blumm, & Barabási, 2010). The distribution of distances between positions of two consecutive visited locations (often referred as displacement) have been well approximated by the Levy flight or truncated Levy flight models (Barbosa et al., 2018).

Conventionally, (truncated) Levy flight models are random walks 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 symbolizes cutoffs at small and large values of d . Given that short trips are more common than long trips, 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. In order to overcome these limitations, the conventional Levy flight model can be extended. For example, Y. Liu, Kang, Gao, Xiao, & Tian (2012) represent geographical heterogeneity as a function of population density, which means that high populated areas are expected to attract greater number of trips.

Furthermore, points of interest (POIs) such as airports, parks, shops, 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, Scellato, Lambiotte, Pontil, & Mascolo (2012) to propose the rank-distance model, which was shown to perform well and captured with high accuracy the displacements in an urban environment.

We integrated ideas from the previous 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 reported by rideshare companies. First, a zone u to contain either the origin/destination (order is uniformly randomly chosen) location is selected with a probability (P_u) that is directly proportional to the number of POIs in this zone. We thus have:

$$P_u \propto \text{number_POIs}(u) \tag{1}$$

Function $\text{number_POIs}(u)$ returns the number of POIs in zone u . Then, distances are randomly chosen according to the probability distribution function that best fits the data under consideration. To identify

the distribution we use the FITTER package (Cokelaer, n.d.). Finally, coordinates for a second location are generated guaranteeing that their distance from the first location approximate the randomly selected distance value, and at this stage, the direction is randomly chosen according to a uniform distribution. In summary, locations are randomly generated considering: a) proportionality to the presence of POIs in the area, which means that regions with greater concentration of POIs will attract more requests; and b) inverse proportionality to the traveled distance, i.e., the higher the distance between two locations lower the probability that they will represent the origin and destination of a request.

Listing 7 depicts an example of how to integrate the previously mentioned procedure. 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 latter describes the parameters of the probabilistic density function to which the distances will be randomly chosen. Results of the comparison between real data and synthetic instances are shown later in Section 9.

Listing 7: Demonstration of how the method can be embedded in the instance generation process

```

1 { ...
2   "attributes": [
3     ...
4     {
5       "name": "origin",
6       "type": "location"
7     },
8     {
9       "name": "destination",
10      "type": "location"
11    },
12    ...
13  ],
14  "method_pois": [
15    {
16      "locations": ["origin", "destination"],
17      "pdf": {
18        "type": "normal",
19        "loc": 6500,
20        "scale": 5000
21      }
22    }
23  ],
24  ...
25 }
```

6 Examples

In this section we describe two problems that the tool can generate instances for. The on-demand transportation problems under consideration in Subsections 6.1 and 6.2 are the DARP and the ODBRP, respectively. We briefly define each problem, and formally present the notations for the attributes that will be used throughout this paper.

6.1 DARP

The DARP consists of planning vehicle routes to serve users that indicate origin (pick-up/departure point) and destination (drop-off/arrival point) through requests. The DARP arises in various contexts and has many variations, but one of the most common is the door-to-door transportation service of elderly or disabled people. Different applications yield distinct constraints or objectives, although cost minimization is the most common goal. Variants of the DARP are either static or dynamic. In the static DARP, all request are known beforehand, while in the dynamic case, requests are announced to the system during the day and the vehicle routes must be adjusted in real-time. For complete surveys on research development of DARP variants we refer to Cordeau & Laporte (2007) and Ho et al. (2018).

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 $r \in R$ contains a couple (o_r, d_r) , which express, respectively, its origin and destination location. The set of origins and destinations of the users are denoted as $O = \bigcup_{r \in R} o_r$ and $D = \bigcup_{r \in R} d_r$, respectively. The moment each request r is announced to the system is denoted time stamp ts_r . Additionally, every request has two time windows, i.e., the earliest and latest times for pick-up and drop-off. The earliest and latest departure times are denoted e_r^u and l_r^u , respectively. Meanwhile, the earliest and latest arrival times are denoted e_r^o and l_r^o , respectively. Some requests may include requirements for vehicles that support wheelchairs. The fleet of vehicles are located in a set W of designated locations named depots. The travel time matrix consists of the travel times between all pair 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. An example configuration file for the DARP is showed in Appendix A.

6.2 ODBRP

The ODBRP involves the routing and scheduling of on-demand buses. For this problem, as opposed to being assigned to their origin or destination locations, passengers are picked up and dropped off at designated bus stations. Objectives include to effectively serve all the requests and minimize total user ride time. Melis & Sørensen (2020) have recently tackled this problem. The ODBRP can also be operated in the static and dynamic modes.

Instances for the on-demand bus routing problem consist of a set of requests R and a travel time matrix. Each request $r \in R$ figures one passenger and contains a set of potential stops that r can be assigned for pick-up S_r^u and a set of potential stops that r can be assigned for drop-off S_r^o . The maximum walking time, representing the passenger’s willingness to walk, is denoted by u_r . The stations in S_r^u and S_r^o are at most u_r from the origin (o_r) and destination (d_r) locations, respectively. Each request also has a time window consisting of an earliest departure time (e_r^u) and a latest arrival time (l_r^o). Note that the decision to define a latest departure time (l_r^u) is left to the system, therefore the journey is considered feasible as long as it is completed before l_r^o . L is defined as the complete set of bus station locations. The planning period is expressed as: $T_e = [ts_{min}, ts_{max}]$, and was previously described in Subsection 6.1. The travel time matrix consists of the estimated travel times between all the bus stations in L . An example configuration file for the ODBRP is showed in Appendix B.

7 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 6. For each instance, we measure the size, value of dynamism, urgency, and geographic dispersion. First, we formally describe these characteristics, and discuss their behavior and impact in realistic scenarios. The terminology used in the following definitions is aligned with the notations of attributes presented in Section 6. However, they can be extended to evaluate the properties of other problems that display similar traits. Our main goal is to assist researchers in the investigation of instance characteristics that affect the performance of algorithms, primarily the ones that influence negatively, which eventually supports the development of more robust methods capable of dealing with real life situations.

7.1 Size

The size of an instance is a measure that depend on to the problem under consideration, e.g., the number of cities determines the size of an instance for the TSP. In previous works addressing the DARP, ODBRP, and similar problems, the instance size is usually expressed as the number of requests (Cordeau & Laporte, 2007; Melis & Sørensen, 2020). The size of instances is an important aspect, since larger instances frequently impose

a greater challenge during experiments. Such behavior is especially evident with exact solvers: as the number of variables increase, the time required to obtain an optimal or even feasible solution grows substantially. Concerning heuristics, the computational time also rises and the solution quality often degrades as instances get larger. Instances of various sizes can be efficiently generated with REQreate, thus used during experiments to evaluate the influence of other variables such as number of available vehicles and their capacity. Testing the limits of a method regarding instance size is crucial, as these systems will be used in real world scenarios where demand can casually increase and the efficiency needs to be maintained.

7.2 Dynamism

Dynamism captures the periodicity aspect of new information revealed during the planning period. Regarding the problems contemplated in this paper, this new information comes in the form of transportation requests issued by users of the system. Therefore, dynamism is measured as the frequency that new requests are revealed to the system. In a highly dynamic scenario requests will arrive continuously. On the contrary, if there are long intervals without new information, the instance can be considered less dynamic. Different scenarios with various levels of dynamism are represented in Figure 3.

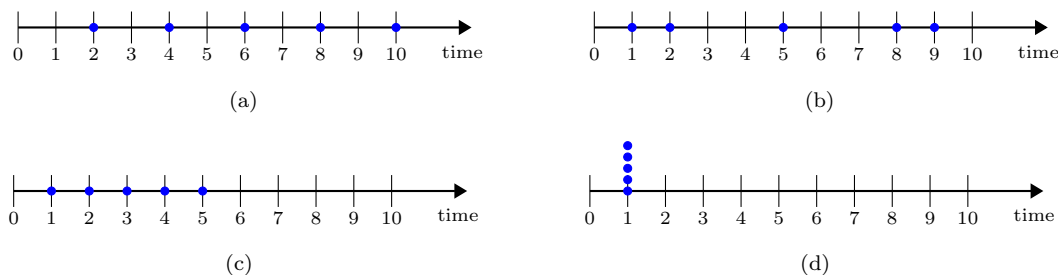


Figure 3: Depiction of different levels of dynamism. Colored dots represent the time one request was announced to the system. The scenario in Figure 3(a) changes continuously in evenly timed intervals, therefore it represents a very dynamic scenario. The dynamism levels decreases in Figure 3(b) and 3(c), as both scenarios have larger intervals without new announcements. In Figure 3(d) all information is known at the same time which results in a case with no dynamism.

According to observations made by Kilby, Prosser, & Shaw (1998) and Pillac, Gendreau, Guéret, & Medaglia (2013), direct implications of dynamism are the required number of restarts and the available time for optimization, i.e., constant announcements imply limited time to perform iterations intended to improve the solution. Borndörfer, Grötschel, Klostermeier, & Küttner (1999) comment on how planning operations for the DARP are impacted by dynamism. The authors remark that the schedule executed during operational hours is often different from the computed version for the static problem, because of several circumstances, such as new requests, cancellations, accidents, and many others. So, since static problems are usually unrealistic, dynamism and its consequences should be investigated.

Lund, Madsen, & Rygaard (1996) presented an earlier definition where dynamism is the proportion between dynamic and static requests. Since the relative timing distribution of arrivals are not accounted in the measure, scenarios where no previous information is known before the planning period can not be distinguished. In Larsen, Madsen, & Solomon (2002), a request is reportedly more dynamic when revealed to the system with a tight interval between the time stamp and the latest possible time to begin servicing the request. However, as discussed later, the authors actually incorporate the urgency feature in their definition, leading to difficulties in drawing separate conclusions on the correlation of dynamism and urgency related to solution quality. More recently, Van Lon et al. (2016) provided separate measures for dynamism and urgency that addresses these limitations, which will be presented in the following paragraphs.

In order to present the dynamism measure, we start by giving some supporting definitions. Consider $R_d = \{r_1, r_2, \dots, r_{|R_d|}\}$ to be the set of requests introduced after the start of planning period (dynamic requests),

and sorted by non decreasing order of time stamps, i.e., $ts_{r_j} \geq ts_{r_i}, \forall j > i$. Let $\Delta = \{\delta_1, \delta_2, \dots, \delta_{|R_d|-1}\} = \{ts_{r_j} - ts_{r_i} \mid j = i + 1 \forall r_i, r_j \in R_d\}$ be the set of interarrival times. Cases with 100% dynamism are characterized by requests being presented in evenly timed intervals (Figure 3(a)), thus possessing the perfect interarrival time (θ), computed as follows: $\theta = \frac{T_e}{|R_d|}$. For each interarrival $\delta_k \in \Delta$ is now possible to compute its deviation (σ_k) from the 100% dynamism case:

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

The deviation of an entire scenario (λ) is then given by the following summation: $\lambda = \sum_{\delta_k \in \Delta} \sigma_k$. Consider bursts as announcements that occur in short periods, consequently leading to interarrival times smaller than θ . Note that the term $\frac{\theta - \delta_k}{\theta} \cdot \sigma_{k-1}$ penalizes those bursts by adding a proportion of the deviation from the previous interarrival time. Moreover, consider $\eta = \sum_{\delta_k \in \Delta} \bar{\sigma}_k$, where:

$$\bar{\sigma}_k = \theta + \begin{cases} \frac{\theta - \delta_k}{\theta} \cdot \sigma_{k-1} & \text{if } (k > 1) \wedge (\delta_k < \theta) \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

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

$$\rho = 1 - \frac{\lambda}{\eta} \quad (4)$$

Following the provided definitions, we now can compute the dynamism for scenarios shown in Figure 3. Each of these scenarios exhibit 5 dynamic requests and a perfect interarrival time of 2, i.e., $\theta = \frac{10.00}{5.00} = 2.00$. The latter permits us to compute the necessary sets and values to determine the dynamism, which are reported in Table 5. In this table, the first column specifies the scenario (Figures 3(a)-3(d)). The second and third columns indicate the set of deviation values (σ) and its summation (λ), respectively. Meanwhile, fourth and fifth column report the normalization set values ($\bar{\sigma}$) and its summation (η), respectively. The last column in the table gives the dynamism value (ρ) for the corresponding scenario. Note that Figure 3(a) represents the 100% dynamism case, since all interarrival times are equal to θ . Furthermore, dynamism levels decrease in Figures 3(b) and 3(c), as they exhibit requests that are revealed close to one another ($\delta_k < \theta$). Ultimately, the 5 requests depicted in Figure 3(d) were announced at the same time, so in such circumstance, the dynamism is zero.

Table 5: Sets and values necessary to determine dynamism for scenarios shown in Figure 3.

Scenario	Δ	σ	λ	$\bar{\sigma}$	η	ρ
Figure 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
Figure 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
Figure 3(c)	{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
Figure 3(d)	{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

7.3 Urgency

The interval between the time stamp of a request and the latest pickup time is referred as reaction time. Urgency is a feature that indicates the length of this interval, i.e., the available time to perform actions regarding a new dynamic request. Let $\chi = \{f_{r_1}, f_{r_2}, \dots, f_{r_{|R_d|}}\}$ be the set of urgency values of an instance, where the urgency (f_{r_i}) of a single request r_i is computed as follows: $f_{r_i} = l_{r_i}^u - ts_{r_i}$. 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 ($\bar{\chi}$) and standard deviation (χ_s) of χ denote the urgency of the corresponding instance. They are computed as follows: $\bar{\chi} = \frac{\sum_{r_i \in R_d} f_{r_i}}{|X|}$ and $\chi_s = \sqrt{\frac{\sum_{r_i \in R_d} (f_{r_i} - \bar{\chi})^2}{|X|}}$. The standard deviation is part of the urgency measure because the mean alone is not a very distinctive feature, 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 Figure 4, which represents a scenario with the following set of urgency values: $\chi = \{3, 1\}$. As a result, the urgency of this instance is: $\bar{\chi} = \frac{3+1}{2} = 2$ and $\chi_s = \sqrt{\frac{1+1}{2}} = 1$.

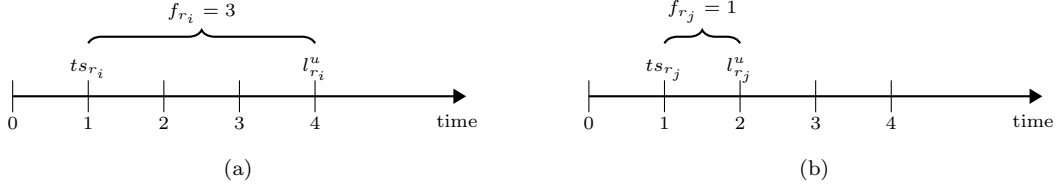


Figure 4: Depiction of different urgency scenarios. Figures 4(a) and 4(b) exhibits the reaction times for requests r_i and r_j , respectively. Note that $f_{r_j} < f_{r_i}$, therefore r_j is more urgent than r_i .

Van Lon et al. (2016) perform computational experiments to evaluate the impact of dynamism and 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 set of requests made by costumers. The objective is to minimize route costs, which is inversely proportional to route quality. The instances generated by the authors were grouped by sets that share similar characteristics, apart from values of dynamism and urgency. The investigated hypotheses led to conclude that these features have different influences on operating costs, as dynamism has a reasonably small effect on operating costs but is negatively correlated with it, while urgency is positively correlated.

7.4 Geographic dispersion

Geographic dispersion is a criterion that express the spread of important locations across the network. It can be expected that a large geographic dispersion, and thus a large 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 in Reyes, Erera, Savelsbergh, Sahasrabudhe, & O’Neil (2018), although we perform slight modifications to make it suitable for the DARP and ODBRP. First, we sum the estimated direct travel times for each request and calculate their average: $\mu = \frac{\sum_{r_i \in R} tt_{o_{r_i} d_{r_i}}}{|R|}$, where for the DARP $tt_{o_{r_i} d_{r_i}}$ represents the estimated direct travel time between origin and destination, while for the ODBRP it depicts an estimated average travel time between stations surrounding the origin and destination. 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 $L_{r_i}^o$ and $L_{r_i}^d$ to be subset of locations that can potentially be served after the origin and destination of r_i , respectively. Consider th_s to be a time threshold indicating that the time windows are close enough to possibly coincide. Specifically, $L_{r_i}^o = \{o_{r_j} \mid \|e_{r_i}^u - e_{r_j}^u\| < th_s \forall r_j \in R \setminus r_i\} \cup \{d_{r_j} \mid \|e_{r_i}^u - l_{r_j}^o\| < th_s \forall r_j \in R \setminus r_i\}$ and $L_{r_i}^d = \{o_{r_j} \mid \|l_{r_i}^o - e_{r_j}^u\| < th_s \forall r_j \in R \setminus r_i\} \cup \{d_{r_j} \mid \|l_{r_i}^o - l_{r_j}^o\| < th_s \forall r_j \in R \setminus r_i\}$. The at most n nearest locations in $L_{r_i}^o$ and $L_{r_i}^d$ are depicted as $N_{r_i}^o$ and $N_{r_i}^d$, respectively. Let $tn_{r_i}^o$ and $tn_{r_i}^d$ correspond to the average travel times from the origin and destination of r_i to locations in $N_{r_i}^o$ and $N_{r_i}^d$:

$$tn_{r_i}^o = \begin{cases} \frac{\sum_{v \in N_{r_i}^o} tt_{o_{r_i}v}}{|N_{r_i}^o|}, & \text{if } |N_{r_i}^o| > 0 \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$tn_{r_i}^d = \begin{cases} \frac{\sum_{v \in N_{r_i}^d} tt_{d_{r_i}v}}{|N_{r_i}^d|}, & \text{if } |N_{r_i}^d| > 0 \\ 0, & \text{otherwise} \end{cases} \quad (6)$$

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

$$gd = \mu + \omega \quad (7)$$

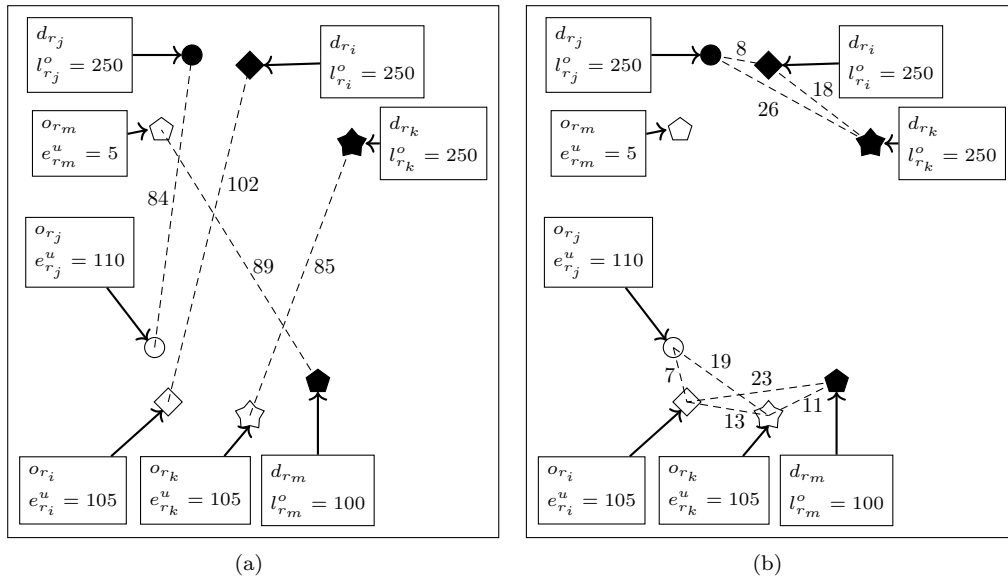


Figure 5: Illustration of an instance with 4 requests to support computations for geographic dispersion.

We exemplify geographic dispersion of an instance using Figure 5. The instance consists of 4 requests $R = \{r_i, r_j, r_k, r_m\}$. Assume $th_s = 10$ and $n = 2$. Travel times between locations are represented by the numbers adjacent to the dashed lines. These values are measured in fictitious units of time. First, according to Figure 5(a), the average of estimated direct travel times is measured as follows: $\mu = \frac{102+84+85+89}{4} = 90$. The sets and values necessary to determine geographic dispersion are reported in Table 6.

Table 6: Sets and values necessary to determine geographic dispersion for instance shown in Figure 5.

Request	N_r^o	tn_r^o	N_r^d	tn_r^d
r_m	$\{\emptyset\}$	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

In this table, the first column specifies the request. The second and third columns indicate the n nearest locations to the origin of the request and the average travel time between the origin to these locations,

respectively. The fourth and fifth columns report the n nearest locations to the destination of the request and the average travel time between the destination to these locations, respectively. Now, it has become possible to calculate ω , thus: $\omega = \frac{104.00}{8.00} = 13.00$. Finally, the geometric dispersion for this instance is: $gd = 90.00 + 13.00 = 103.00$.

Reyes, Erera, Savelsbergh, Sahasrabudhe, & O’Neil (2018) evaluated the effect of dynamism, urgency and geographic dispersion features on performance metrics towards solutions for the Meal Delivery Routing Problem (MDRP). The MDRP consists of building routes for couriers to deliver meals requested by costumers. Couriers pick up orders as soon as they are available from the restaurant and deliver them at the designated costumer’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 arrival time at the drop-off location (ready-to-door). Addressing the MDRP accompanies many challenges, as orders arrive constantly (very dynamic) and are expected to be delivered quickly (high urgency), which poses obstacles to optimize performance measures, specially if the number of couriers is small. The authors observe that higher dynamism increases the route costs, as it is less like to combine orders in a single route to reduce costs. Increases in the geographic dispersion measure are converted in higher click-to-door and ready-to-door mean times. In the meantime, stronger negative effect on these performance measures were observed with increased urgency.

As previously mentioned, we designed REQreate with the capacity of generating instances with different sizes, levels of urgency, dynamism and geographic dispersion. Consequently, similar studies to Van Lon et al. (2016) and Reyes et al. (2018) can be performed to explore the potential effects of these measures on solution quality for a comprehensive list of existing and future on-demand transportation problems. A major benefit is to design methods that overcome pitfalls leading to poor performance. For example, upon arrival of very urgent requests it might be interesting to change momentarily the target from building routes with minimal traveled distance to reduce delay times, in favor of meeting a reasonable balance between customer satisfaction and profit for service providers.

8 Instance similarity

To perform informative analyses algorithms should be tested on a diverse benchmark set. The direct rationality is that similar instances are unlikely to contribute with meaningful additional knowledge. Therefore, we present a concept of instance similarity, based on the approach introduced in Leefink & Hans (2018). In order to assess the similarity of two instances of the same size I and J , we take into account that instances for the problems described in this paper consist of a set of requests distributed over a network during a time period. First, we measure the proximity between locations of two distinct requests r_i and r_j , where $r_i \in I$ and $r_j \in J$. We compute and sum the travel times between the origins and destinations of r_i and r_j : $\phi = tt_{o_{r_i}o_{r_j}} + tt_{d_{r_i}d_{r_j}}$. Requests r_i and r_j are ϕ -proximate if ϕ is below a given threshold, i.e., $\phi < th_{tt}$.

On the assumption that two requests are ϕ -proximate, we also compare if they have a similar time stamp and earliest departure times. Let: a) $\tau = \|ts_{r_i} - ts_{r_j}\|$; and b) $\vartheta = \|e_{r_i}^u - e_{r_j}^u\|$, 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 τ and ϑ are lower than specified thresholds, i.e., $\tau < th_{ts}$ and $\vartheta < th_e$. Combining the definitions given so far, the similarity level between two requests r_i and r_j ($\xi_{r_i r_j}$) becomes:

$$\xi_{r_i r_j} = \begin{cases} 1.00 & \text{if } (\phi < th_{tt}) \wedge (\tau < th_{ts}) \wedge (\vartheta < th_e) \\ 0.75 & \text{if } (\phi < th_{tt}) \wedge ((\tau < th_{ts}) \oplus (\vartheta < th_e)) \\ 0.50 & \text{if } (\phi < th_{tt}) \wedge (\tau \geq th_{ts}) \wedge (\vartheta \geq th_e) \\ 0.00 & \text{otherwise} \end{cases} \quad (8)$$

We exemplify the concept of similarity with Figure 6. In each figure we depict different requests, and we analyze their level of similarity regarding r_i . Hypothetically, consider threshold levels to be $th_{tt} = 20$,

$th_{ts} = 10$, and $th_e = 10$. Identical to Figure 5, travel times between locations are represented by the numbers adjacent to the dashed lines. These values are again measured in fictitious units of time.

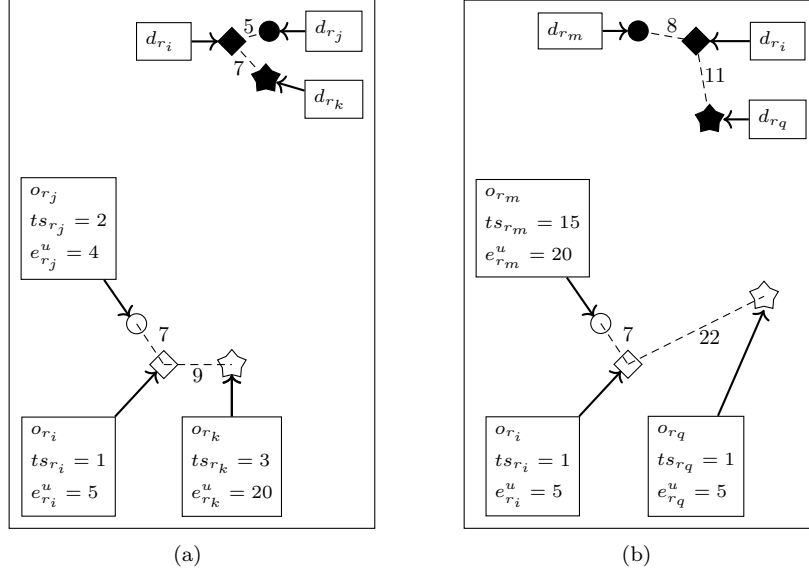


Figure 6: Examples of requests with different levels of similarity.

The values necessary to determine similarity between the pictured requests are reported in Table 7. In this table, the first column specifies which pair of requests the similarity is being computed for. The following three columns give the computed values for ϕ , τ and ϑ , respectively. In these columns, the marks \checkmark and \times besides the numbers indicate if the values are, respectively, below or above their corresponding thresholds. The last column report the level of similarity between the pair of requests. The first two lines indicate the requests shown in Figure 6(a). Since all values for requests r_i and r_j are below the given thresholds, then $\xi_{r_i r_j} = 1.00$. 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 lines indicate the requests represented in Figure 6(b). Note that requests r_i and r_m are only ϕ -proximate, so $\xi_{r_i r_m} = 0.50$. Meanwhile, the sum distance between locations of r_i and r_q exceeds th_{tt} , for this reason $\xi_{r_i r_q} = 0.00$.

Table 7: Computed values necessary to determine similarity between requests shown in Figure 6.

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

We introduce the following definition to help determine the similarity between two instances: 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. Similarity is computed for every pair of requests between the two instances, and as 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 maximal matching $M_s \subset E_s$ and measure the average weight of edges: $\Omega_{IJ} = \frac{\sum_{m \in M_s} w_m}{|M|}$.

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 of times. We acknowledge the fact that 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.

9 Comparison between real data and synthetic instances

In this section, we analyze the mobility patterns of real trips performed by rideshare companies. We use datasets that were made available publicly from “Chicago, Illinois” (“Chicago Department of Business Affairs & Consumer Protection”, n.d.). We divide these datasets in different time 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 datasets and discuss some results. The instances for this Section were generated according to attributes for the DARP.

The period selected in which all trips were performed is from September 1, 2019 to September 30, 2019. Ridesharing companies dataset from “Chicago, Illinois” documents more than 7 million trips during the given period of time. Each record contains information on the pick up and drop off time and locations of the trips, traveled distance, fare paid, along with many other fields. All records are anonymous. Besides, privacy is also preserved by rounding times to the nearest 15 minutes. Therefore, results regarding each of those fields will be an approximation of reality.

Considering that travel patterns change according to time and day of the week, it is reasonable to conduct this examination on different days and time intervals. First, we split the datasets 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: a) between 7:00 and 10:00; and b) between 16:00 and 20:00. During non-working days, the time period considered was between 16:00 and 20:00, since in the morning the activity is almost negligible. To generate the synthetic instances, the origin and destinations were randomly chosen according to the method described in Section 5, i.e., following the number of POIs and the best fitted distribution of distances corresponding to each combination of days and time periods.

Figure 7 illustrates the spatial distribution of origin and destination locations referring to trips reported by rideshare companies in the city of “Chicago, Illinois” and the set of synthetic generated instances. The three plotted periods for the set of real trips portray similar distribution patterns, as the same regions attracted a large amount of individuals during both working and non-working days (orange and red colored portions). Certainly, this behaviour can be explained as the number of POIs is higher in the aforementioned region (see the yellow area with some orange and red spots in Figure 8). This is further evidence of the positive correlation between spatial distribution of POIs and the potential of a location to be associated with individual’s trajectories. Despite being more evenly spread than the locations in the real life datasets, the synthetic instances also display a slightly denser concentration in the region where most of the POIs are situated.

Distance distributions were also similar during working and non-working days. They are depicted in Figure 9(a). Such statistical measure 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 plays an important role, as 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 were 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 16:00 and 20:00 during non-working days, 75% of the trips had a smaller length than 8.84 km. A potential reason is that citizens prefer taking part 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 about 27.00 to 30.00 km. This might be due to the presence of “O’Hare International Airport”, as it is located rather far from the city.

As previously indicated, the function and parameters that returns the best fit for the distance distributions

are used for the synthetic instances. The distributions used to generate the synthetic instances are illustrated in Figure 9(b). The shape of the graphs are relatively similar, however, for the synthetic instances, the occurrence of small to medium distances was somewhat lower when compared to the real journeys. 75% of the distances from generated requests were below 10.22 km and 9.75 km for periods 7:00 to 10:00 and 16:00 to 20:00, respectively, during working days. Also, 75% of rides for the interval between 16:00 to 20:00 during non-working days reported 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.

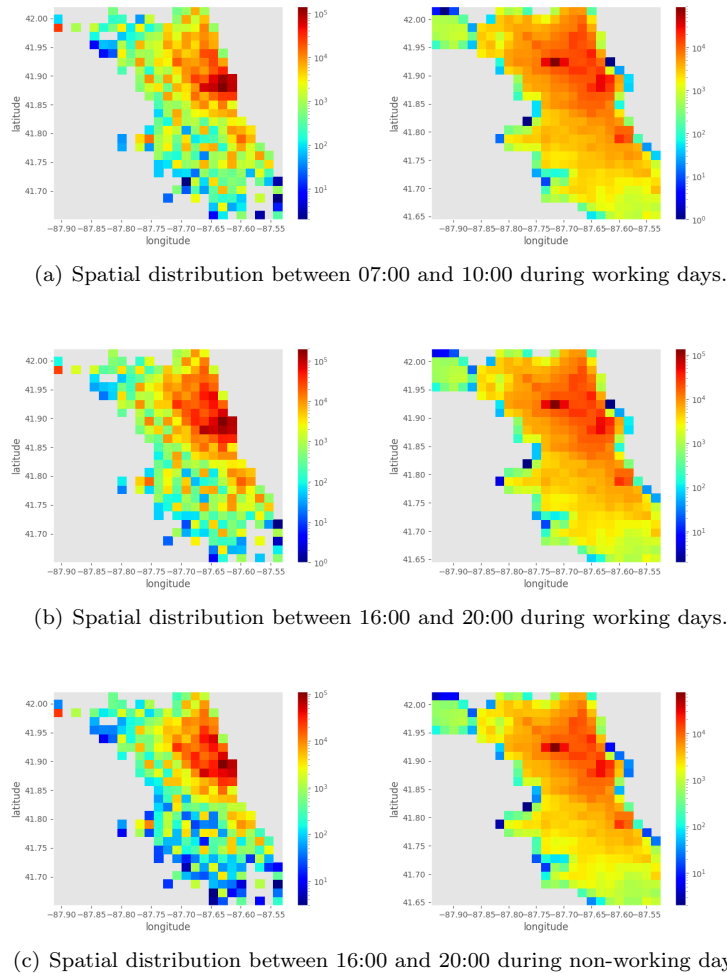


Figure 7: Spatial distribution of origin and destination locations. Figures on the left refer to trips reported by rideshare companies in “Chicago, Illinois”, meanwhile figures on the right concern synthetic generated instances. The color bar accounts for the number of locations.

Based on the traveled distances and the duration of the trips, the average speed can be evaluated. Speed values were slightly higher during non-working days when compared to working days. For the latter, the computed averages of speeds were 22.11 km/h and 21.30 km/h for the intervals between 07:00 to 10:00 and 16:00 to 20:00, respectively. From 16:00 to 20:00 during non-working days, the average speed was 24.79 km/h.

According to Figure 8, one could argue that considering solely the concentration of POIs can possibly improve the spatial distributions of origin and destinations displayed in synthetic instances, however, the

distances decay effect presented in Figure 9 might be lost. For this reason, new methods that balance both aspects may be elaborated in further research.

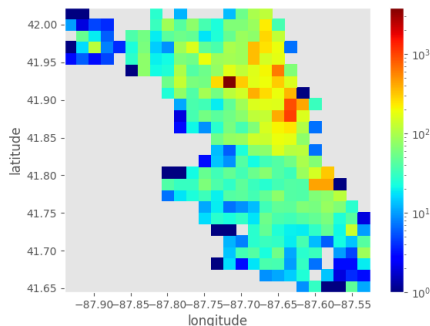


Figure 8: Spatial distribution for Points of Interest (POIs) in the city of “Chicago, Illinois” according to reported locations in OpenStreetMaps (OSM). The color bar accounts for the number of POIs.

Before concluding this analysis we make three remarks. First, some results are influenced by the method of transport (car, bus, etc.). Since we utilize data from rideshare companies, the trips have also an economic constraint factor, as it is unlikely for citizens to favor this transportation modality as fares rise significantly with long distances. One should bear in mind that an investigation of subway rides, for example, may return a rather different distance distribution, in which higher distances are expected to occur more frequently. Second, the observed spatial and distance distributions were 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 provided concerning POIs locations. Details are provided by OSM users and some of them might be missing.

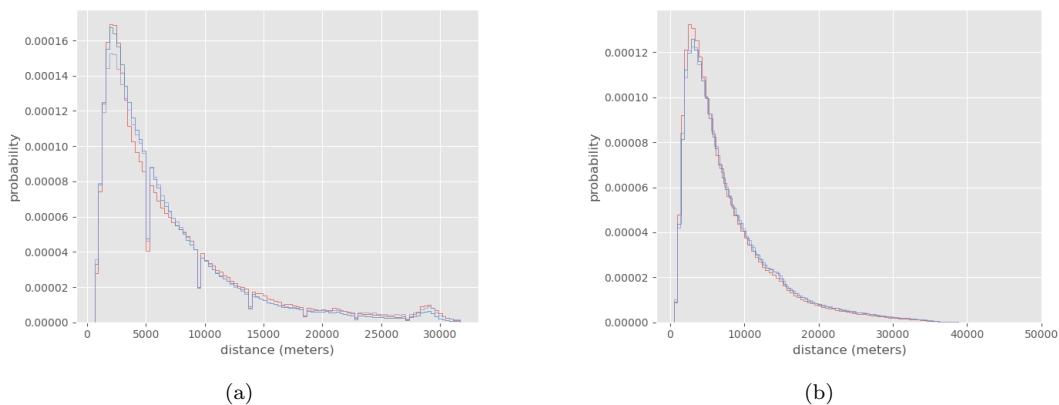


Figure 9: Distance distribution referring to trips reported by rideshare companies in the city of “Chicago, Illinois” (on the left) and for the set of synthetic instances (on the right).

10 Benchmark set

In this section we generate a benchmark set for the ODBRP. Instances are generated according to different sizes, levels of dynamism, urgency and geographic dispersion. The set was built to be diverse so it is possible to evaluate the influence of these properties on proposed methods for the ODBRP. First, we explain how we approximately control the output of each instance property. Then, we describe how instances are organized.

Size can be easily adjusted with item *requests*, as shown in Section 4.2. Listing 8 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 8. 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 8: Example on how to control properties.

```

1 { ...
2   "attributes": [
3     "requests": 500,
4     {
5       "name": "time_stamp",
6       "type": "integer",
7       "time_unit": "s",
8       "pdf": {
9         "type": "uniform",
10        "loc": 30000,
11        "scale": 500
12      },
13      "dynamism": 50
14    },
15    {
16      "name": "reaction_time",
17      "type": "integer",
18      "time_unit": "s",
19      "pdf": {
20        "type": "normal",
21        "loc": 600,
22        "scale": 60
23      }
24    },
25    {
26      "name": "latest_departure",
27      "type": "integer",
28      "time_unit": "s",
29      "expression": "time_stamp + reaction_time"
30    },
31    {
32      "name": "direct_travel_time",
33      "type": "integer",
34      "time_unit": "s",
35      "expression": "dtt(origin,destination)",
36      "constraints": ["direct_travel_time >= 600", "direct_travel_time <= 1200"]
37    }
38  ],
39 }
```

The attribute “reaction_time” in Listing 8 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 ($\bar{\chi}$) of 10 minutes (600 seconds) and standard deviation (χ_s) of 1 minute (60 seconds). 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 8), 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 minutes of time distance between origin and destination. So, the interval can be adjusted in order to create instances with a greater or smaller value of geographic dispersion.

We used the network from “Chicago, Illinois” to generate instances. Instances are divided into small (between 300 and 600 requests, in steps of 300), medium (between 900 and 1800 requests, in steps of 300), and large (between 2100 and 3000 requests, in steps of 300). The planning period is between 07:00 and 08:00 (1 hour). The dynamism levels vary between 0% and 100% in steps of 10. Meanwhile, urgency mean is set to

lie in $\{5, 10, 15, 30\}$ minutes. Meanwhile, the urgency standard deviation is set to lie in $\{0, 5, 10\}$ minutes. Three limited intervals for geographic dispersion were utilized: a) between $[180, 1000]$ (short distance trips); b) $]1000, 3000]$ (medium distance trips); and c) $]3000, 6000]$ (long distance trips). A fourth option with no interval limit is also considered.

Each instance group is identified by $N_p_s_b_e_d_m_t_g$, where N denotes name of city for which the network was retrieved; p is an acronym for the problem, in this case ODBRP; s gives the size of the instance; b and e denote the beginning and end of the planning period, respectively; d gives the dynamism level; m and t provide the urgency mean and standard deviation, respectively; and g denotes the geographic dispersion value. We carefully generated the instances so it is possible to fairly evaluate the different levels of dynamism, urgency and geographic dispersion. This means that some instances will portray the same requests locations, and varying only the properties values. The instance files are available on Github³.

11 Conclusion and future research

In this paper, we presented REQreate, a tool aimed at generating instances that are significantly more realistic than previous approaches for on-demand public transportation problems. Instances from these problems mainly consist of demand from passengers for transportation. Hence, real life networks are retrieved from OpenStreetMaps (OSM) with support of the OSMnx tool (Boeing, 2017). We described 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 presented properties that can be considered when analyzing the performance of optimization algorithms: size, dynamism, urgency, and geographic dispersion. The concept of instance similarity was also proposed to provide some level of diversity in the benchmark instance sets.

We used datasets provided by rideshare companies from the city of “Chicago, Illinois” and examined which statistical properties of human movement could be inferred. Furthermore, REQreate was utilized to generate synthetic instances and we performed a comparison between the properties of the two groups. Requests were 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 and yet reasonable assumptions which can be useful in producing distinct urban patterns when data accessibility is an issue. We also show that the tool can provide constructive suggestions for the planning of on-demand transportation systems. For example, under which spatial, temporal and distance distribution conditions the implementation is profitable. Finally, we also proposed a benchmark set consisting of 5280 instances for the ODBRP.

Appropriate future research directions include new methods to describe mathematically a broader variety of human behaviour regarding intra-urban traveling. Hypothetically, population density combined with POIs could strengthen the outcome of spatial distributions. Another interesting direction is to understand how the proposed properties (size, dynamism, urgency and geographic dispersion) influence the performance of a newly developed method.

Acknowledgments

This project was funded by the University of Antwerp BOF.

References

Agrawal, M., Elmaghraby, S. E., & Herroelen, W. S. (1996). Dagen: A generator of testsets for project activity nets. *European journal of operational research*, *90*(2), 376–382.

³https://github.com/michellqueiroz-ua/instance-generator/tree/master/examples/ODBRP_benchmark_configuration_files

- Barbosa, H., Barthelemy, M., Ghoshal, G., James, C. R., Lenormand, M., Louail, T., . . . Tomasini, M. (2018). Human mobility: Models and applications. *Physics Reports*, *734*, 1–74.
- 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.
- 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).
- Cokelaer, T. (n.d.). *Fitter package*. (Online reference, last access on April 4, 2022, <https://fitter.readthedocs.io/en/latest/>)
- Cordeau, J.-F., & Laporte, G. (2007). The dial-a-ride problem: models and algorithms. *Annals of operations research*, *153*(1), 29–46.
- De Corte, A., & Sörensen, K. (2014). Hydrogen: an artificial water distribution network generator. *Water resources management*, *28*(2), 333–350.
- Demeulemeester, E., Vanhoucke, M., & Herroelen, W. (2003). Rangen: A random network generator for activity-on-the-node networks. *Journal of scheduling*, *6*(1), 17–38.
- Dingil, A. E., Schweizer, J., Rupi, F., & Stasiskiene, Z. (2018). Transport indicator analysis and comparison of 151 urban areas, based on open source data. *European Transport Research Review*, *10*(2), 58.
- Drakoulis, R., Bellotti, F., Bakas, I., Berta, R., Paranthaman, P. K., Dange, G. R., . . . Amditis, A. (2018). A gamified flexible transportation service for on-demand public transport. *IEEE Transactions on Intelligent Transportation Systems*, *19*(3), 921–933.
- Drexl, A., Nissen, R., Patterson, J. H., & Salewski, F. (2000). Progen/ π x—an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. *European Journal of Operational Research*, *125*(1), 59–72.
- Gallagher, M., & Yuan, B. (2006). A general-purpose tunable landscape generator. *IEEE transactions on evolutionary computation*, *10*(5), 590–603.
- Garey, M. R., & Johnson, D. S. (1979). A guide to the theory of np-completeness. *Computers and intractability*, 641–650.
- Gkiotsalitis, K., & Stathopoulos, A. (2016). Demand-responsive public transportation re-scheduling for adjusting to the joint leisure activity demand. *International Journal of Transportation Science and Technology*, *5*(2), 68–82.
- 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.
- Hernando, L., Mendiburu, A., & Lozano, J. A. (2015). A tunable generator of instances of permutation-based combinatorial optimization problems. *IEEE Transactions on Evolutionary Computation*, *20*(2), 165–179.
- Ho, S. C., Szeto, W., Kuo, Y.-H., Leung, J. M., Petering, M., & Tou, T. W. (2018). A survey of dial-a-ride problems: Literature review and recent developments. *Transportation Research Part B: Methodological*, *111*, 395–421.

- Kilby, P., Prosser, P., & Shaw, P. (1998). Dynamic vrps: A study of scenarios. *University of Strathclyde Technical Report*, 1(11).
- Kolisch, R., Sprecher, A., & Drexl, A. (1995). Characterization and generation of a general class of resource-constrained project scheduling problems. *Management science*, 41(10), 1693–1703.
- 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, 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.
- Leeftink, G., & Hans, E. W. (2018). Case mix classification and a benchmark set for surgery scheduling. *Journal of scheduling*, 21(1), 17–33.
- 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, 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.
- 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.
- Macedo, E., & Tchemisova, T. (2017). A generator of nonregular semidefinite programming problems. In *Congress of apdio, the portuguese operational research society* (pp. 177–199).
- 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.
- Michalewicz, Z., Deb, K., Schmidt, M., & Stidsen, T. (2000). Test-case generator for nonlinear continuous parameter optimization techniques. *IEEE Transactions on Evolutionary Computation*, 4(3), 197–215.
- Morrison, R. W., & De Jong, K. A. (1999). A test problem generator for non-stationary environments. In *Proceedings of the 1999 congress on evolutionary computation-cec99 (cat. no. 99th8406)* (Vol. 3, pp. 2047–2053).
- 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), e37027.
- 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.
- “Chicago Department of Business Affairs & Consumer Protection”. (n.d.). *Transportation network providers - trips*. (Online reference, last access on July 21, 2021, <https://data.cityofchicago.org/>)

- 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*.
- 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).
- 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).
- 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.
- 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, 103573.

Appendix

A Configuration file for the DARP

The configuration file example for the DARP is divided in Listings 1 and 2. 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 1, 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 explicit 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: “ $dt(x,y)$ ”, 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 30600 (8:30) and standard deviation 3600 (1 hour), and the constraint state it must be greater than or equal to parameter “min_planning_period”.

The remainder attributes are declared in Listing 2. 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 seconds (0-10 minutes). 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 minutes) according to an 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 travel time 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.

B Configuration file for the ODBRP

The configuration file example for the ODBRP is divided in Listings 3 and 4. As previously remarked in Appendix A, these Listings are part of the same configuration file. They were also simplified for visualization purposes. First, in Listing 3, 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 to be 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 A. The “earliest_departure” (e_p^u) values are randomly chosen according to an uniform distribution within the interval [25200, 32400] (7:00 to 9:00). We remark that constraint “earliest_departure \geq min_planning_period” is indeed redundant for this attribute, however we display it in the configuration file for elucidation purposes.

The remainder attributes are declared in Listing 4. 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 an uniform distributions between 300 to 600 seconds (5-10 minutes) 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 Python⁴. 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 “lead_time” are declared exactly as in Appendix A, 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 know 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_arrival \leq max_planning_period” ensures that “latest_arrival” does not surpass the planning period time window. 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.

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

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

```

1 { ...
2   "parameters":[
3     {
4       "name": "min_planning_period",
5       "type": "integer",
6       "value": 7,
7       "time_unit": "h"
8     },
9     {
10      "name": "max_planning_period",
11      "type": "integer",
12      "value": 10,
13      "time_unit": "h"
14    },
15    {
16      "name": "depots",
17      "type": "array_locations",
18      "size": 1,
19      "locs": "random"
20    }
21  ],
22  "attributes":[
23    {
24      "name": "origin",
25      "type": "location"
26    },
27    {
28      "name": "destination",
29      "type": "location"
30    },
31    {
32      "name": "wheelchair_requirement",
33      "type": "integer",
34      "pdf": {
35        "type": "uniform",
36        "loc": 0,
37        "scale": 1
38      }
39    },
40    {
41      "name": "direct_travel_time",
42      "type": "integer",
43      "time_unit": "s",
44      "expression": "dtc(origin,destination)",
45      "output_csv": false
46    },
47    {
48      "name": "earliest_departure",
49      "type": "integer",
50      "time_unit": "s",
51      "pdf": {
52        "type": "normal",
53        "loc": 30600,
54        "scale": 3600
55      }
56      "constraints": [ "earliest_departure >= min_planning_period" ]
57    }
58  ]
59  ...
60  ...
61 }

```

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

```

1 { ...
2   "attributes": [
3     {
4       "name": "lead_time",
5       "type": "integer",
6       "time_unit": "s",
7       "pdf": {
8         "type": "uniform",
9         "loc": 0,
10        "scale": 600
11      },
12      "output_csv": false
13    },
14    {
15      "name": "time_stamp",
16      "type": "integer",
17      "time_unit": "s",
18      "expression": [ "earliest_departure - lead_time" ]
19      "constraints": [ "time_stamp >= min_planning_period", "time_stamp <= max_planning_period" ]
20    },
21    {
22      "name": "time_window_size",
23      "type": "integer",
24      "time_unit": "s",
25      "pdf": {
26        "type": "uniform",
27        "loc": 300,
28        "scale": 300
29      },
30      "output_csv": false
31    },
32    {
33      "name": "latest_departure",
34      "type": "integer",
35      "time_unit": "s",
36      "expression": "earliest_departure + time_window_size"
37    },
38    {
39      "name": "earliest_arrival",
40      "type": "integer",
41      "time_unit": "s",
42      "expression": "earliest_departure + direct_travel_time"
43    },
44    {
45      "name": "latest_arrival",
46      "type": "integer",
47      "time_unit": "s",
48      "expression": "earliest_arrival + time_window_size",
49      "constraints": [ "latest_arrival <= max_planning_period" ]
50    }
51  ],
52  "travel_time_matrix": ["depots", "origin", "destination"]
53 }

```

Listing 3: Configuration file example for the ODBRP - *part 1*

```

1 {
2   ...
3   "places": [
4     {
5       "name": "zone_center",
6       "type": "zone",
7       "centroid": true,
8       "radius": 2000,
9       "length_unit": "m"
10    }
11  ],
12  "parameters":[
13    {
14      "name": "min_planning_period",
15      "type": "integer",
16      "value": 6,
17      "time_unit": "h"
18    },
19    {
20      "name": "max_planning_period",
21      "type": "integer",
22      "value": 9,
23      "time_unit": "h"
24    },
25    {
26      "name": "zone_dest",
27      "type": "array_zones",
28      "size": 1,
29      "value": ["zone_center"]
30    }
31  ],
32  "attributes":[
33    {
34      "name": "origin",
35      "type": "location"
36    },
37    {
38      "name": "destination",
39      "type": "location",
40      "subset_zones": "zone_dest",
41    },
42    {
43      "name": "direct_travel_time",
44      "type": "integer",
45      "time_unit": "s",
46      "expression": "dtt(origin,destination)",
47      "output_csv": false
48    },
49    {
50      "name": "earliest_departure",
51      "type": "integer",
52      "time_unit": "s",
53      "pdf": {
54        "type": "uniform",
55        "loc": 25200,
56        "scale": 7200
57      },
58      "constraints": [ "earliest_departure >= min_planning_period" ]
59    },
60    ...
61  ]
62 }

```

Listing 4: Configuration file example for the ODBRP - *part 2*

```

1 { ...
2   "attributes": [
3     {
4       "name": "max_walking",
5       "type": "integer",
6       "time_unit": "s",
7       "pdf": {
8         "type": "uniform",
9         "loc": 300,
10        "scale": 300
11      },
12      "output_csv": false
13    },
14    {
15      "name": "walk_speed",
16      "type": "real",
17      "time_unit": "kmh",
18      "pdf": {
19        "type": "uniform",
20        "loc": 4,
21        "scale": 1
22      }
23    },
24    {
25      "name": "stops_orgn",
26      "type": "array_primitives",
27      "expression": "stops(origin)",
28      "constraints": ["len(stops_orgn) > 0"]
29    },
30    {
31      "name": "stops_dest",
32      "type": "array_primitives",
33      "expression": "stops(destination)",
34      "constraints": ["len(stops_dest) > 0", "not (set(stops_orgn) & set(stops_dest))"]
35    }
36  ],
37  {
38    "name": "lead_time",
39    "type": "integer",
40    "time_unit": "s",
41    "pdf": {
42      "type": "uniform",
43      "loc": 0,
44      "scale": 600
45    },
46    "output_csv": false
47  },
48  {
49    "name": "time_stamp",
50    "type": "integer",
51    "time_unit": "s",
52    "expression": ["earliest_departure - lead_time"]
53    "constraints": ["time_stamp >= min_planning_period", "time_stamp <= max_planning_period"],
54    "static_probability": 0.5
55  },
56  {
57    "name": "latest_arrival",
58    "type": "integer",
59    "time_unit": "s",
60    "expression": "earliest_departure + (direct_travel_time * 1.5)",
61    "constraints": ["latest_arrival <= max_planning_period"]
62  }
63 ],
64 "travel_time_matrix": ["bus_stations"]
65 }

```