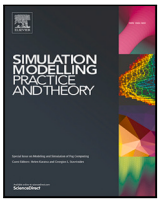
Simulation Modelling Practice and Theory 125 (2023) 102742

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

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



Realistic traffic model for urban environments based on induction loop data

José D. Padrón [a,*], Enrique Hernández-Orallo [a], Carlos T. Calafate [a], David Soler [b], Juan-Carlos Cano [a], Pietro Manzoni [a]

[a] Department of Computer Engineering (DISCA), Universitat Politecnica de Valencia, Valencia, Spain
[b] Institute of Multidisciplinary Mathematics (IMM), Universitat Politecnica de Valencia, Valencia, Spain

ARTICLE INFO

Keywords:
DFROUTER
OD traffic matrix
Induction loop detectors
Traffic modeling

ABSTRACT

As we gradually move towards smarter cities, having greater control of the traffic in the city becomes of utmost importance. In addition, to efficiently manage such traffic, it is critical to be able to predict the impact of different traffic policies, and potential changes to the city road systems structure. To this end, accurate traffic simulation models must be derived that can help in this task. This paper presents a tool that aims to improve the representativeness of traffic simulations by generating realistic Origin-Destination (OD) traffic matrices. In particular, we focus on cities whose source of traffic information are the induction loop detectors deployed through the different streets and avenues of the city. By comparing against the widely used DFROUTER tool, part of the SUMO open-source traffic simulation package, we show how we are able to improve the traffic model accuracy significantly. Specifically, we achieved more realistic route lengths and a better distribution of traffic sources and destinations.

1. Introduction

Smart cities are the present and future of our society. According to United Nations (UN), and its Sustainable Development Goals (SDG) [1], we can link the SDG number 11 (Sustainable Cities and Communities) [2] to the creation of Smart Cities. In particular, it states [2]: "*Rapid urbanization is resulting in a growing number of slum dwellers, inadequate and overburdened infrastructure and services (such as waste collection and water and sanitation systems, roads and transport), worsening air pollution and unplanned urban sprawl*". The International Telecommunication Union (ITU) [3] describes how these problems can be solved with the creation of applications for smart cities. For instance, they suggest the development of a *Digital Twin* that is able to monitor and manage the traffic in the city.

Likewise, the literature has recently provided different solutions for traffic management in smart cities [4–6]. Traffic management models can be applied in different ways, such as: reducing congestion using IoT in smart cities [7], harmonizing traffic flow in a city [8], reducing vehicle fuel consumption [9], and creating efficient traffic networks in urban environments [10]. These are good examples of how important traffic modeling is for a city. All solutions offered by traffic management models must be validated in simulators before being applied in real scenarios. Several aspects are addressed in this validation phase. For example, it is possible to test the variation in traffic volume when traffic restrictions are applied based on political or environmental decisions.

To ensure that a good model is created, having a reliable traffic dataset is fundamental. To create an effective traffic management model, Origin-Destination (OD) traffic matrices that are representative of the real traffic must be available. OD traffic matrices are

* Corresponding author.
E-mail address: jdpadper@disca.upv.es (J.D. Padrón).

https://doi.org/10.1016/j.simpat.2023.102742
Received 15 December 2022; Received in revised form 24 February 2023; Accepted 25 February 2023
Available online 28 February 2023

important because they describe the flow of vehicular traffic between different origin and destination points in a transportation network. In particular, it typically includes the number of vehicles that travel from each origin to each destination, providing a detailed picture of the traffic patterns in a specific area. Usually, it is difficult to obtain reliable OD traffic matrices if they are not obtained through GPS traces. In particular, most of the available GPS traces were obtained for taxis. Nevertheless, as our goal is to develop a generic traffic model, that is, for passenger vehicles, using GPS seemed a challenging task. This is due to the difficulty of accessing GPS data from personal vehicles or drivers' smartphones, which raises significant privacy concerns.

In general, the solution to the aforementioned problem lies in the use of induction loop detectors or cameras in the cities. Specifically, we will focus on the first one. Using induction loops, the number of vehicles that pass over them can be obtained. With this information, we can get an idea about which areas are more congested, and which are less. Yet, this is not enough to generate an OD traffic matrix. To do so, we need the aid of external tools to compute vehicle routes, such as DFROUTER [11], which, since its inception, has been oriented to highways.

The problem of using DFROUTER in the context of a city, is that, as highlighted by Zambrano et al. [12], the prediction of the traffic volume generated is not reliable, being 238% greater than the actual volume. Particularly, along with the traffic volume problem, we have detected that the length of the routes generated is not representative, being the average length obtained in this study [12] of only 1291 m. Considering that the studied area was the city of Valencia which has a size of 127.89 square kilometers (approx.: 12.6 km W-E; 10.1 km N-S), routes cannot be considered representative.

For this reason, this paper proposes a novel tool that is able to generate representative OD traffic matrices for a city based on induction loop data, and on the use of ABATIS [13]. This is an open-source route server which uses the Open Source Routing Machine (OSRM) and OpenStreetMap data as its input. It has various tools that can be combined to create routes for different vehicles, and our solution will utilize the Multi-Level Dijkstra approach for route selection. In particular, with our solution, we are able to improve DFROUTER's accuracy in three key aspects: traffic volume, route spatial distribution, and route length. Specifically, in terms of route length, we are able to generate more representative routes with an average length of 3937 m. This obtained result is definitely a more realistic and accurate value for further experimentation and traffic analysis.

The rest of this paper is organized as follows: the next section presents and discusses some related works on this topic. Then, in Section 3, we provide a clear view of the problem we address, highlighting which performance parameters we attempt to improve. Our proposed solution is presented in Section 4, which details the different elements that conform our workflow. Then, results are presented and discussed in Section 5. Finally, Section 6 presents the conclusions of our work, and discusses future work.

2. Related work

Traffic data can be obtained through different sources like GPS, cellular networks, inductive loops, video image processors, etc., as studied by Leduc [14], and by Jain et al. [15]. Within these sources, we can differentiate between Floating Car Data (FCD), and conventional "*in-situ*" technologies. FCD refers to approaches that obtain traffic data by locating each vehicle through two methods: GPS-based, and cellular phone based. "*In-situ*" refers to methods that collect traffic data through detectors located along the road. From these methods, we can highlight both induction loops and video image processors.

In recent years, thanks to the affordability of GPS systems, their integration in vehicles has increased exponentially. Along with this, the increase in the number of smartphones has meant that vehicles that are not equipped with this system can also be analyzed. Herrera et al. in [16] used this last source of information to create a system that retrieves the GPS data directly from smartphones. In particular, they built a four-layer system consisting on: (i) GPS-Enabled smartphones inside vehicles, (ii) a cellular network provider, (iii) a data collection infrastructure, and (iv) an information display system. The smartphones were able to send, every 3 s, their location (latitude, longitude, altitude), and combining such data with time, the car's speed was obtained. Thanks to this, the trajectory information was also obtained. Ge and Fukuda [17] go one step further and, using the aggregated data of the smartphone traces, create OD traffic matrices. In particular, their solution is based on the fitting of the GPS matrices by applying spatial interaction models. Then, they are able to obtain the OD traffic matrices. Yet, despite GPS data being the most straightforward approach to creating OD traffic matrices, they are not always easy to acquire. This is because most of the data are collected by different companies, and it is not easy to have access to them. Similarly, people often feel that their privacy is being violated, because, even though the data are anonymous, a system is keeping track of all their movements.

The acquisition of traffic data through the cellular network is one of the alternatives to GPS data collection. Thanks to newer technologies like LTE and 5G, the network infrastructure has increased, so it is easier to track vehicles. Cáceres et al. [18] evaluated six different models to detect the number of vehicles moving from one cell to another using anonymous phone call data. These models are based on the relationship that exists between call activity and traffic mobility. Among these, a physical model is selected. This is based on the assumption that a cell phone makes a moving call when either the user makes a call in each of the two cells forming the boundary within a short period of time, or the user has an active call and switches from one cell to the other. Results showed that a reasonable traffic flow was estimated. Similarly, Iqbal et al. [19] proposed a method to create OD traffic matrices based on cell phone Call Detail Records (CDR). In particular, their method consists of three steps; (i) collect the CDR data and generate a first OD matrix based on tower-to-tower transit, (ii) convert the first OD into a node-to-node transient OD matrix, and (iii) determine a scaling factor and obtain the OD traffic matrix. Nevertheless, these data are difficult to obtain because they depend on the different telecommunication companies. Moreover, the deployment of such a public infrastructure to record traffic conditions is not affordable for administrations.

Video image processors are one of the most extended "*in-situ*" methods to acquire traffic data. Thanks to the development of technology, in particular artificial intelligence techniques, nowadays it is easier to track traffic mobility through video cameras. Back

in 2009, Mallikarjuna et al. [20] developed a system that collects traffic data using video image processing. There are two principal methodologies: the data collection, and the traffic estimation. For the former one, they use cameras that are pointing to the center of the road in order to cover a considerable amount of road area. For the latter, they are based on the typical artificial intelligence steps: (i) learning, (ii) detection and classification, (iii) filtering and tracking, and (iv) feature extraction and classification. After that, they are able to obtain both macroscopic, and microscopic traffic data. Savrasovs and Pticina [21] proposed a methodology to generate OD traffic matrices. Their methodology is based on 5 steps. First, they collect the data from cameras and manually record the license plates of the vehicles. Second, they develop an algorithm that is able to detect the exit point of each entering vehicle and, consequently, generate the first OD matrix. Third, this matrix is manually validated, to obtain an OD probability matrix, based on which a second version of the OD matrix is obtained. Fourth, they calibrate this last matrix by applying different methods. Finally, they use regression analysis to the calibrated matrix, and if it achieves good performance indexes in terms of the R-square metric, the final OD matrix is obtained. To sum up, video image processing is a good method to register traffic flow. However, it is a very expensive method for the administration, because it requires installing multiple cameras all over the city. In some cases, this solution is good for a small city with few streets, but not for covering a large city.

Traffic data collected through the use of induction loops is a method widely used by administrations. With this solution, administrations can track traffic volume in almost all parts of the city. One way to collect traffic data is by placing induction loops throughout the road network. Then, these induction loops send the information to a server, which then saves this information into a database. In the same way, the traffic data collected is published through an API [22], and can be accessed by people and researchers. To create OD traffic matrices with the induction loops data, several solutions have been proposed. Among these, DFROUTER [11] stands out. In particular, DFROUTER follows four steps to achieve this goal. First, it imports the road network, including the location, and measurements of the induction loops. Second, it classifies the detectors in: source detectors, middle point detectors, and sink detectors. Third, it calculates the vehicle flow between consecutive detectors. Finally, it computes the different OD based on route usage probabilities and returns them.

In summary, induction loops are a good method to track traffic status in a city. Likewise, we have seen that literature has provided methods to collect data efficiently. However, despite the fact this method can calculate OD traffic matrices, these are not representative. Hence, in this paper, we will focus on this issue, and develop a tool to generate representative traffic characterization via OD matrices.

3. Problem statement

SUMO [23] is a widely-used open-source traffic simulator in the ITS field. It has been downloaded thousands of times and utilized in numerous research projects and papers. It also has various tools and modules for route searching, such as DFROUTER [11], JTRROUTER [24], OD2TRIPS [25], and DUAROUTER [26].

In this section, we will focus on DFROUTER as it can generate vehicle routes from induction loop count data. Flowrouter [27] is another tool that can generate data from the same input, but in preliminary tests, we have discarded it. This is because it was only able to generate 65 different routes according to our own experiments, which is a small number given that DFROUTER generated 38,003 (short) routes, as we will see later on. Therefore, we do not consider it as a representative number and have discarded it.

However, we also want to address some of the limitations of SUMO, specifically in regard to determining actual traffic injected into a network and analyzing the lengths of generated routes, particularly in urban environments. DFROUTER, in particular, was designed primarily for highways and not city streets or avenues, as its authors state in [28]: “The idea behind this router is that nowadays, most highways are well-equipped with induction loops, measuring each of the highways’ entering and leaving flows”. This means that its usefulness in generating an origin/destination matrix in a city is limited.

3.1. DFROUTER traffic injected issue

In [12], Zambrano et al. discussed a problem for DFROUTER, which is related to the number of vehicles that pass over the different induction loops in the output that is generated. In particular, the authors found that it has a large deviation compared to the real data. Furthermore, this deviation is also reflected in the number of vehicles injected. This means that DFROUTER injects a number of vehicles that is greater than expected, being that the overall difference obtained was +238% when compared to real data. Due to this issue, the OD matrix generated was not representative of the actual traffic conditions in the city.

For evaluating and analyzing our proposed new methodology, we have recreated their experiments to obtain the reliability of the DFROUTER data. Specifically, we have also used DFROUTER and the city induction loop real data, although such data was retrieved much more recently using web tools, and so correspond to current traffic patterns as opposed to the former ones. As a result, we have obtained via DFROUTER the traffic distribution throughout the city (OD matrix). Then, applying reverse engineering, we obtained the distribution of vehicles per induction loop. This allowed us to compare side by side the percentage error between the real data, and the DFROUTER provided data.

Analyzing the induction loop data distribution in more depth, we can look at the Empirical Cumulative Distribution Function (ECDF) presented in Fig. 1. This figure shows the number of vehicles that pass over each induction loop, sorted by size. In it, we can observe that there is a great difference between both functions, evidencing the DFROUTER is not generating realistic traffic routes. Specifically, for the real data function, we observe a parabolic function with a maximum value of around 20k vehicles. In contrast, the DFROUTER function has a maximum value of 13k vehicles, and it behaves like a straight line until 5k, a value beyond which it

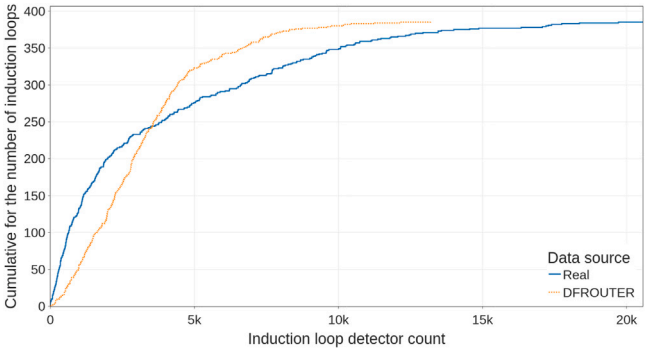


Fig. 1. ECDF of induction loop real and DFROUTER data.

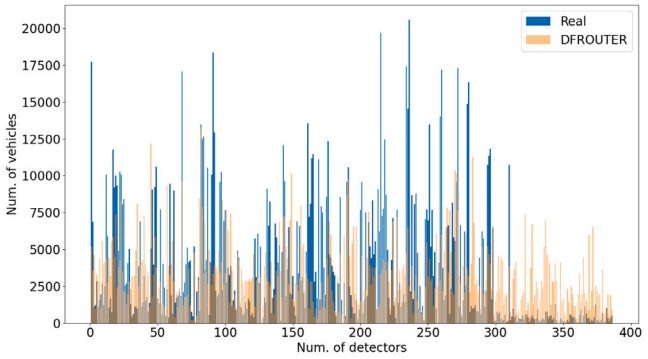


Fig. 2. Induction loop data difference (Real vs. DFROUTER).

Table 1
Induction loop data difference stats (Real vs. DFROUTER).

Num. of detectors	386
Avg.	+231%
Std. Dev.	+577%
Q1	−51%
Q2	+27%
Q3	+244%
Min.	−88%
Max.	+5417%

starts to make a parable. This leads us to think that the difference between the detector data for DFROUTER, and the actual traffic will be somewhat high.

Table 1 and Fig. 2 complement the data shown in Fig. 1. Firstly, Table 1 presents different statistical measures that provide a complete overview of the differences between the real and the DFROUTER induction loop data, In particular, the difference is calculated for each detector. We can see that the minimum difference between the data of induction loops is −88% in favor of real data, while the maximum goes up to 5417% more in DFROUTER. In terms of quartiles, we observe that the first half of the values (Q2) already experience an increase of +27%, but for the Q3 threshold, this number increases to +244%. Due to these numbers, the average percentage difference between real, and DFROUTER induction loop data is 231%. Secondly, Fig. 2 shows how the number of vehicles per detector variate. In particular, it can be observed the difference in the number of vehicles for each detector between the real traffic and the DFROUTER approach. As we can see, there exist big differences between both data. More specifically, at the end of the x-axis, from 300 to 400, we can see that the maximum value of 5417% is somewhere in this range.

All in all, considering both the analysis of Zambrano et al., and our own analysis, we can state that DFROUTER has an issue when distributing and injecting the traffic in a city when having induction loop data as input. In particular, we observe that the variation remains very similar: 238% in the former work, and 231% in our work. This led us to think that the error DFROUTER introduces is consistent, and repetitive. Specifically, in the next subsection, we focus on evaluating the length of routes.

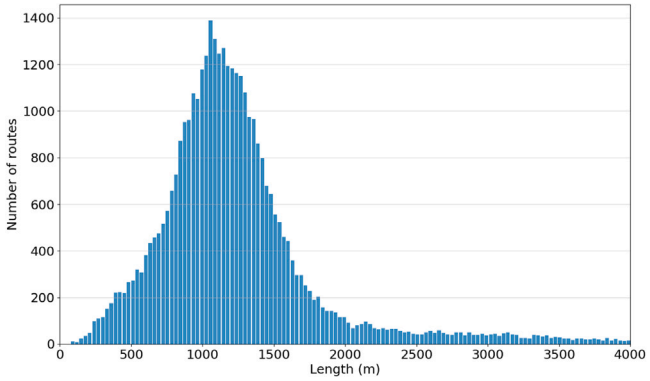


Fig. 3. Route length distribution for DFROUTER.

Table 2
Routes statistics from the DFROUTER route output file.

Num. of routes	38,003
Avg.	1291 m.
Std. Dev.	740 m.
Q1	917 m.
Q2	1154 m.
Q3	1425 m.
Min.	75 m.
Max.	10,049 m.

3.2. DFROUTER routes length issue

As previously stated, DFROUTER is more oriented toward generating routes for highways than for streets or avenues. Because there is a clear difference between a highway environment and a city environment, the resulting lengths of the routes are prone to have little representativeness in urban environments. In order to check the validity of this statement, we have generated the routes for the city of Valencia. To contextualize, in terms of population, Valencia is the 3rd city in Spain, and the 23rd in the European Union, with 800,215 inhabitants. Regarding size, the analyzed area has a size of 127.89 square kilometers (approx.: 12.6 km W-E; 10.1 km N-S). Likewise, the city has around 4000 streets and avenues with major roads such as the "*Calle San Vicente Mártir*" with a length of approximately 4 km, and minor roads such as the "*Calle Doctor Serrano*", with a length of approximately 180 m.

The route length distribution is presented in Fig. 3. As we can see, most of the routes are concentrated in the range from 700 m to 1400 m. In addition, we can observe that there is a large number of routes with a length of less than 500 m, and even as low as 100 m in some cases. These short lengths are not representative of a city such as Valencia, with very limited parking availability, meaning that most people will make such trips on foot.

To make a more complete analysis of the results generated by DFROUTER we can observe the routes statistics from DFROUTER on Table 2. In particular, the minimum length of a route is 75 m, which is very small and is not close to real situations. On the contrary, the maximum length is 10,049 m., which is a large route, and representative of someone who lives on the outskirts of the city. Additionally, for median and average length, we have 1154 and 1291 m, respectively. These lengths, as we said before, are too short considering the actual citizen experience. In fact, 75% of the routes can be considered not representative as they are below 1425 m.

Finally, to summarize, considering that 75% of the routes are below 1425 m, we can state that the length of most of the routes clearly does not represent the traffic in this city considering its size. To sum up, it has been demonstrated that DFROUTER has an issue with the length of the routes when generating them in the context of a city.

4. Proposed solution

In traffic simulators, there is a constant problem when trying to generate representative traffic for a city based on data from induction loops. As detailed in the previous section, the problem is generated by DFROUTER. Specifically, DFROUTER faces two main problems when generating traffic within a city; (i) the amount of traffic injected is excessive, and (ii) the length of the routes remains too short. In this section, we detail how to solve these issues by proposing a novel traffic route generator based on using the ABATIS [13] routing engine. Since DFROUTER generates the routes, along with the traffic distribution in time and space, we divide our algorithm in 4 parts for the sake of clarity. These are: (i) data collection and curation, (ii) routing core, (iii) data expansion, and (iv) time distribution. In addition, we will explain how data is prepared in order to apply our algorithm effectively. The complete workflow of the proposed solution can be observed in Fig. 4.

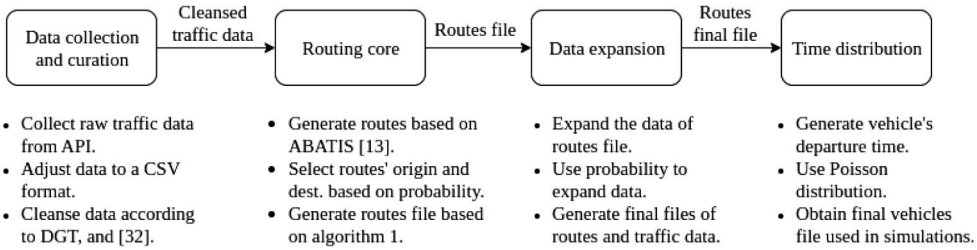


Fig. 4. Workflow of the proposed solution.

Table 3
CSV columns structure.

ATA	n_vehicles	way_id	Node
A1	17 750	429492850 429492853	4286326639 4278564760 ...
A10	6866	770904161 60432369	25767499 1596479816 ...
A102	1101	13945413 13945412	131943282 249507229 ...
...	...	...	...

4.1. Data collection and curation

We start by gathering data from the Valencia City Council through a public API [22]. Then, based on the study of Zambrano et al. [29], and Calafate et al. [13], we focus on data obtained from the 386 induction loops in the city between 8 and 9 am on an ordinary Monday. To clarify, all sensors are of the same type and were deployed in different sections of the main streets and avenues of the city. These are embedded under the surface of the roadway, one per lane. At a given location, all lanes are covered for the sake of completeness. In our case, the data used is from May 2021, not November 2012, as in that prior work. Afterwards, we converted the data from a JSON file to a CSV file in order to better deal with such data.

Secondly, we add to the CSV the ID of the street, and the nodes related to it, based on the information provided by OpenStreetMap [30]. Then, since data is split into 15-min slots, we combine all the data for the same induction loop to be more efficient. Having done that, the CSV columns will resemble the structure presented in Table 3. The “ATA” column corresponds to the ID of the induction loop as provided by the City Hall, and it adopts a string format. The “n_vehicles” column is the aggregated number of vehicles that pass over the induction loop between 8 and 9 am. Then, the “way_id” and the “node” columns correspond to the associated ID of the way(s) for the induction loop and the associated nodes of the way.

Thirdly, we cleansed the data based on two main factors: the number of passenger cars, and the number of vehicles cruising for parking.

Concerning the first one, we have obtained the number of passenger cars for Valencia from the Spanish Department of Traffic (*Dirección General de Tráfico (DGT)*) on their 2019 annual table statistics [31]. According to this, the number of passenger cars is 72% of the total. So, because our goal is to analyze the routes of passenger cars in particular, we reduce the number of vehicles corresponding to each induction loop by 28%.

Regarding the second one, the number of passenger cars cruising for parking is 15%. This value is estimated according to Hampshire et al. [32], considering that in Stuttgart (207.4 km^2, and 634,830 inhabitants), a city similar to Valencia (134.6 km^2, and 791,413 inhabitants), that is the percentage of vehicles cruising for parking. Since we only want vehicles that do a specific route instead of a cruise for parking, we now subtract a 15% from the 72% of the previous step. To sum up, we have reduced the data of the vehicles that pass through the induction loop by 38.8% so as to achieve more representative route data in terms of traffic that actually requires management and/or improvement.

Finally, with the use of NETCONVERT [33], we add the edges (ID of the road given by SUMO) and their associated nodes, and the nodes with their associated coordinates in a database. Likewise, we include in the database the cleansed data of the CSV, in order to have all the parameters in the same file.

At the conclusion of this process, we have the necessary data to operate the “Routing core” part.

4.2. Routing core

Having obtained the required data by following the procedures detailed above, we now proceed to describe the core part of our solution. As stated above, the engine of our routing algorithm will be ABATIS [13]. This is an open-source route server based on the Open Source Routing Machine (OSRM) [34]. As input data it uses the files extracted from OpenStreetMap. It has different tools that, when combined, are capable of generating different vehicle routes. In particular, for the route selection of our algorithm, we will use the Multi-Level Dijkstra (MLD) approach. Afterwards, along with ABATIS, our implementation consists of the combination of various steps. These can be observed in Algorithm 1, and with them, it will be able to generate more representative routes.

As can be seen in Algorithm 1, the inputs consist on: (i) the edges of SUMO and their relation to nodes; (ii) the nodes and their relation to coordinates; and (iii) the traffic data provided by the induction loops. Furthermore, the outputs are two CSV files: one

Algorithm 1: Proposed routing algorithm.

```
   Input: Edges, Nodes, Traffic data
   Output: CSV with routes, CSV with updated traffic data
1  d ← Num. of detectors, β ← reduction coefficient;
2  n ← Num. of vehicles for a detector, ∑_{i=1}^{d} n_i ← Total num. of vehicles;
3  δ_{o,des,mid} ← tolerated error for origin, destination and intermediate point;
4  endCondition ← (∑_{i=1}^{d} n_i) × (1 − β), currentVehicles ← Current ∑_{i=1}^{d} n_i;
5  ABATISRt, SUMORt, NodesRt ← {};
6  while currentVehicles ≥ endCondition do
7      originPoint = getMostProbablePoint(Traffic data);
8      Filter possible destination points → min. distance 2 km;
9      destinationPoint = getMostProbablePoint(Filtered data);
10     ABATISRoute = generateRoute(originPoint, destinationPoint);
11     SUMORoute = ABATIStoSUMOFormat(ABATISRoute);
12     NodesRoute = ABATIStoNodesFormat(ABATISRoute);
13     ATAList ← {};
14     if n_o ≤ δ_o and n_des ≤ δ_des then
15         Add origin and destination ATA to ATAList;
16         for node in NodesRoute do
17             ATA = getATAFromNode(node);
18             if n_mid ≤ δ_mid then
19                 Add the ATA to ATAList if is not present yet;
20                 if node == lastNode then
21                     Update currentVehicles ← Decrease n by 1 for each detector present in the ATAList;
22                     AddRouteToCorresponding Set(ABATISRoute, SUMORoute, NodesRoute);
23                 end
24             else
25                 break;
26             end
27         end
28     end
29 end
30 CSV with routes ← generateCSV(ABATISRt, SUMORt, NodesRt) ;
31 Return (CSV with routes, CSV with updated traffic data);
```

with the routes created, and the other one with the updated traffic data. The idea of the algorithm is to iteratively obtain the routes, selecting the origin and destination based on the number of vehicles that pass over the induction loops.

We now proceed to explain in detail the three key parts of our algorithm. Firstly, the loop condition present on line 5. This condition is fundamental because the algorithm iterates up to a certain value, which affects the accuracy of the results. This is because the process of obtaining routes is costly in terms of execution time because by placing constraints on the allowed error, the difficulty of obtaining a route gradually increases. Therefore, we have chosen to obtain the routes for a percentage of the vehicles through the use of a reduction factor β. These routes will be expanded in the next module "Data expansion". If we now analyze the values of the condition, these are: the total number of vehicles that pass over the induction loops, and a reduction factor β. This reduction factor will be $0 \leq \beta \leq 1$. As an example, if we want to iterate over 25% of the data, β needs to be 0.25. It is important to mention that the $\sum_{i=1}^{d} n_i$ on the left on the condition decreases according to line 20, while the $\sum_{i=1}^{d} n_i$ on the right is a copy, and remains unchanged.

Secondly, the block between lines 6 and 11 includes two main functions: one that gets the most probable point, and one that retrieves the route from ABATIS. Regarding the first one, it works in a way where an ATA (ID of the induction loop) is selected based on the number of vehicles associated with it. Then, a random node of the ATA is selected, in order to obtain a specific latitude and longitude. This spatial distribution-based mechanism is applied to select origin and destination points, which results in a more accurate and comprehensive estimation. The only difference between them is that the destination point must be at least 2 km away in a straight line from the origin point. With regard to the second function, it sends an HTTP request to the ABATIS server, and receives a JSON with the route to be followed as a sequence of coordinates. This route covers the streets without induction loops because ABATIS adopts the full map to create paths, and not only the streets with induction loop detectors. Since routes are long and traverse many points of the city, roads without induction loops are mainly covered. At the end, this route is converted into the SUMO format and a sequence of nodes. This is done in order to be able to run both future simulations and the next part of this algorithm.

Finally, with regard to the rest of the lines, notice that the conditions present on lines 14 and 18, and the update step on line 21, are critical. With respect to the condition on line 14, the tolerated error $\delta_{o,des}$ is the ratio of vehicles exceeded (compared to the reference value), which we do not want to surpass to avoid high error values. Specifically, for the origin and destination points, the tolerated error is 0, and so the counter of that detector must not be less than 0. On the contrary, the δ_{mid} in condition on line 18, must not exceed a certain percentage of the initial value of the detector. This percentage will vary depending on the chosen β value. This is because, when a higher β is chosen, the execution time will be longer, and so a very restrictive δ_{mid} will be counterproductive. Regarding line 21, this is where we update the data in order to be able to execute the algorithm correctly. In particular, we subtract 1 from the counter of each induction loop present in the route. This will lead to the variation of $currentVehicles$, and, consequently, to the variation of the selected origin and destination points. This is because, as we have already mentioned, both points are selected based on probability, and the change in the number of vehicles will modify this probability.

4.3. Data expansion

In the previous subsection, we discussed that the total amount of vehicles is not generated by our solution due to an excessive execution time. This is the reason why we need to implement another algorithm that expands the generated data so that a reduced dataset can mimic the results of the full dataset. This process is accomplished by Algorithm 2.

Algorithm 2: Data expansion algorithm.

Input: Real traffic data, Generated traffic data, CSV routes file
Output: CSV with all the routes, CSV with the total traffic data
1 $\boldsymbol{\sigma^2_{real,gen.}} \leftarrow$ Variance of real traffic and generated traffic;
2 $\boldsymbol{\mu_{real,gen.}} \leftarrow$ Mean of real traffic and generated traffic;
3 $\boldsymbol{n} \leftarrow$ Detector count;
4 $\boldsymbol{p_{route}} \leftarrow$ Probability to choose a route;
5 $RoutesCopy \leftarrow$ copy of the data in the CSV routes file;
6 **while** $\frac{\mu_{gen.}}{\mu_{real}} < 1.1$ **do**
7 **if** ***not*** $\frac{\sigma^2_{gen.}}{\sigma^2_{real}} > 1$ **then**
8 Choose a route based on $\boldsymbol{p_{route}}$;
9 Add the route to $RoutesCopy$;
10 Update generated traffic data $\leftarrow +1$ to all $\boldsymbol{n}$ present in the route;
11 Recalculate $\boldsymbol{\mu_{gen.}}$ and $\boldsymbol{\sigma^2_{gen.}}$;
12 **else**
13 Generate CSV $\leftarrow$ $RoutesCopy$;
14 Generate CSV with the total generated traffic data;
15 **break**;
16 **end**
17 **end**
18 Return (CSV with all the routes, CSV with the total traffic data);

As can be seen, the algorithm is mainly based on the variance (σ^2) and mean (μ) values, for both real traffic data and the generated traffic data in Algorithm 1. The algorithm begins with an iteration that satisfies the condition $\frac{\mu_{gen.}}{\mu_{real}} < 1.1$. This is in order to ensure that the ratio between both means is acceptable (just a 10% higher) to generate representative data. Then, the rate between both σ^2_{real} and $\sigma^2_{gen.}$ must be less than 1, in order to generate a more accurate result. Afterwards, if the condition is satisfied, it will choose a route from the *Generated routes file* with probability p_{route}. Then, it updates by adding 1 to the values corresponding to those detectors present in the route. Finally, since detector data are changed, a recalculation of $\sigma^2_{gen.}$ and $\mu_{gen.}$ is needed.

On the contrary, if the variance condition is not satisfied, it means that the iteration process needs to stop. Before this, the algorithm will generate the output CSV including all the routes generated, and the traffic data generated.

4.4. Time distribution

In order to have the same outputs as DFROUTER we need to create a vehicles file. Specifically, what differentiates this file from the route file generated in Algorithm 2 is that our routes (from now on referred to as vehicles) have no depart time. These vehicles are generated by counting the number of routes and the number of repetitions of each of them. The number of repetitions will be the number of vehicles that travel through the route. Therefore, we generate all the routes, and then we count the number of repetitions of each route to assign the number of vehicles that travel through that route. Having made this clear, in order to satisfy the DFROUTER requirement, we need to add to all vehicles a specific departure time. This is done by applying a Poisson Distribution for the vehicles that take the same route. The complete behavior of this function can be observed in Algorithm 3.

As stated above, the key functionality of the algorithm is the use of the Poisson Distribution Function. We have chosen Poisson, and not Gaussian or other alternatives, since Poisson processes are widely accepted as being adequate for generating the number of vehicles passing through a given route segment during a specific time interval [35–37]. Regarding the algorithm structure, we

Algorithm 3: Time distribution algorithm.

Input: CSV with all the routes
Output: CSV with vehicles traffic information
$T \leftarrow$ simulation period;
$\lambda_{route} \leftarrow$ mean number of vehicles for a certain route during T;
$Routes \leftarrow$ New set with routes ID and num. of vehicles associated to it;
for $route, vehicles_{route}$ ***in*** $Routes$ **do**
  $A \leftarrow$ Empty set of size T;
  $\frac{vehicles_{route}}{T} \leftarrow$ Calculate λ_{route};
  for k=0:T **do**
    Draw a sample from $P(x = k) = \frac{e^{-\lambda_{route}} \lambda_{route}^{k}}{k!}$ and add it to A;
  end
  for $element$ ***in*** A **do**
    if $element$!= 0 **then**
      Add vehicle to the CSV vehicles file;
    end
  end
end
Generate CSV with vehicles traffic information;
Optimize for simulation the CSV with vehicles traffic information;
Return (CSV with vehicles traffic information);

can observe that it iterates over the routes and vehicles per route. Then, in order to be able to draw samples from the Poisson distribution, an empty set of size T is created for each route. At the same time, the λ_{route} is calculated. Afterwards, we draw a sample from the execution of the Poisson Distribution Function, and add it to A at position k. This position refers to the simulation time at which the vehicle will depart. Then, we add each vehicle in a particular route to the CSV file of vehicles' traffic information, and once the iteration over all the routes is completed, we generate the CSV. To conclude, and to avoid future simulation warm-up issues, we optimize it and return the result.

Algorithm 4: Optimization of vehicle departure time.

Input: CSV with vehicles traffic information
Output: CSV with vehicles traffic information optimized
$\sigma_{veh} \leftarrow$ Standard deviation of vehicles per second;
$\mu_{veh} \leftarrow$ Mean of vehicles per second;
$\epsilon \leftarrow$ Value that changes depending on the method used;
$\mu_{duration} \leftarrow$ Mean of vehicles trip duration;
$\Gamma(\mu_{veh}, \epsilon) \leftarrow$ Optimization function;
$Vehicles \leftarrow$ Set of vehicles traffic information from the CSV;
$vehicleCounter$, $departureTime \rightarrow 0$;
for $vehicle$ ***in*** $Vehicles$ ***where*** $vehicle_{departTime} \geq \mu_{duration}$ **do**
  if $vehicle_{departTime} \neq departureTime$ **then**
    $vehicle_{departTime} = vehicle_{departTime}$ - $\mu_{duration}$;
    $vehicleCounter = 1$;
  else
    if ***not*** $vehicleCounter \geq \Gamma(\mu_{veh}, \epsilon)$ **then**
      $vehicle_{departTime} = vehicle_{departTime}$ - $\mu_{duration}$;
      $vehicleCounter = vehicleCounter + 1$;
    end
  end
end
Return (CSV with vehicles traffic information optimized);

Elaborating on the optimization algorithm (see Algorithm 4), this consists of an iteration over the CSV file containing vehicles' traffic information. The iteration starts over the last vehicle with a departure time equal to or greater than the $\mu_{duration}$. Afterwards, the departure time of a certain number of vehicles is updated. This number is calculated by an optimization function $\Gamma(\mu_{veh}, \epsilon)$ which is established before running the iteration. This function is described as:

$$\Gamma(\mu_{veh}, \epsilon) = \mu_{veh} + \epsilon \tag{1}$$

where ϵ can be equal to 0, σ_{veh}, $-\sigma_{veh}$, or a random integer in the range $\{-\sigma_{veh}, \sigma_{veh}\}$. Finally, it returns the optimized CSV with the vehicles' traffic information.

Summing up, throughout this section we have described the workflow of our solution. After the collection and cleansing of the induction loop data, the generation of the routes for the vehicles is performed for a portion of these data. This is the main part of our solution. Then, since this process is very time-consuming, the number of vehicle routes obtained is extended by using statistical procedures. Finally, once all the induction loop data has been processed, an optimized distribution of the departure times is made to minimize the warm-up period required and achieve a stable number of vehicles in the system afterwards. This process presents a great scalability potential to handle an increasing number of OD elements because it generates routes based on induction loop data, and then it assigns the number of vehicles that take the route.

5. Results and discussion

Having analyzed the details of our proposed solution, we will now detail how the experiments used to validate our proposal were defined, and which performance metrics have been used. Afterwards, we discuss the quality of our solution by analyzing the results for each experiment.

It is important to mention that all experiments have been done using as reference the data from the induction loops, and comparing the results using either the OD matrices produced by DFROUTER, or our OD matrices generated through the combination of Algorithms 1, 2, and 3. Likewise, the hardware used for these experiments consists of a PC with an Intel i7-12700 3.6 GHz CPU, 16 GB DDR4 2400 MHz RAM, and an M.2 SSD storage with a reading speed of 3300 MB/s, and a writing speed of 1200 MB/s.

5.1. Tuning our algorithm

Firstly, we need to select an optimal β that achieves an acceptable performance in terms of mean squared error (MSE), while avoiding excessive execution times when compared to other solutions.

In particular, for our proposed solution, we have tested different reduction factors to find the one offering the best trade-off: $\beta = \{0.01; 0.10; 0.15; 0.20; 0.25; 0.30; 0.35; 0.40\}$. In the same way, the intermediate tolerated error used (normalized value) is $\delta_{mid} = 1.1$, and so the value for each detector cannot exceed 10% of its reference value. Having this in mind, we will divide our experiments into two groups: the traffic distribution analysis along with the algorithm validation, and the characterization of generated routes in terms of quality and representativeness.

In order to show the different trade-offs between our proposed solution and DFROUTER, we calculate the MSE with respect to the real data with the following equation:

$$\mathrm{MSE}(y, \hat{y}) = \frac{1}{n_{\text{detectors}}} \sum_{i=0}^{n_{\text{detectors}}-1} (y_i - \hat{y}_i)^2 \tag{2}$$

where:

$n_{\text{detectors}}$ = Total num. of detectors

y_i = num. of vehicles that pass over a detector

$\hat{y}_i$ = num. of vehicles that pass over a detector for some β

We do it for our solution when adopting different β values, and also for DFROUTER. Likewise, we calculate the execution time for our solution (again varying the β value), and for DFROUTER. Finally, an analysis of the trade-off between MSE and execution time is presented to show how much the MSE drops when increasing the execution time.

Fig. 5 shows the behavior of the MSE when increasing β. As we can observe, there is a decreasing tendency when β is higher, but for $\beta = \{0.01; 0.10\}$ the MSE obtained is higher than that of DFROUTER. Because of this, these β values are ruled out. Conversely, since the MSE for $\beta = \{0.15; 0.20; 0.25; 0.30; 0.35; 0.40\}$ is lower than the obtained by DFROUTER, they are still considered when targeting the optimal solution.

Regarding the performance in terms of execution time when increasing β, Fig. 6 shows that such time increases in a parabolic way. In particular, we can observe that the slope from $\beta = 0.01$ to $\beta = 0.20$ is limited, while from $\beta = 0.10$ to $\beta = 0.40$ it becomes very steep, having exponential growth. Moreover, most values achieved for our solution are higher than those for DFROUTER. This is because of the complexity of Algorithm 1, and our restrictive δ_{mid} parameter.

In Fig. 7 we combine the results of the previous figures to observe the general performance of our solution compared to DFROUTER. It shows that there is a correlation between MSE, execution time, and β; so, when β becomes higher, the execution time is higher, but the MSE becomes lower. Thus, we need to select an optimal β that achieves an acceptable performance in terms of MSE, while avoiding excessive execution times when compared to other solutions. In this case, the β value we select for the experiments that follow is 0.25, because it is able to reduce the MSE by 25% compared to the DFROUTER result. In addition, the execution time is much lower than that for higher values of β, which achieve a lower MSE.

Having identified the β value with the best trade-off, we proceed to analyze the behavior of our solution in SUMO over a simulation period of 60 min. To this end, we compare the use of various methods. "Default" refers to the unoptimized vehicle

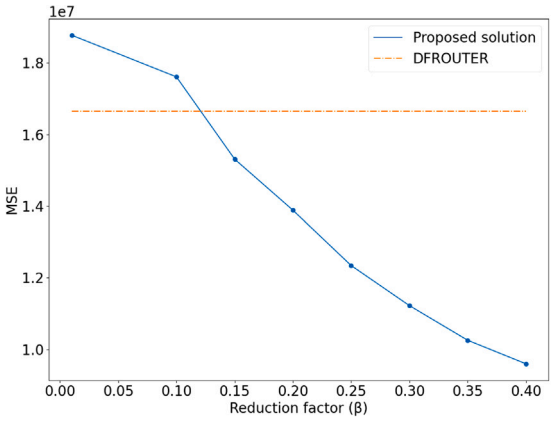


Fig. 5. MSE values when varying the value of β.

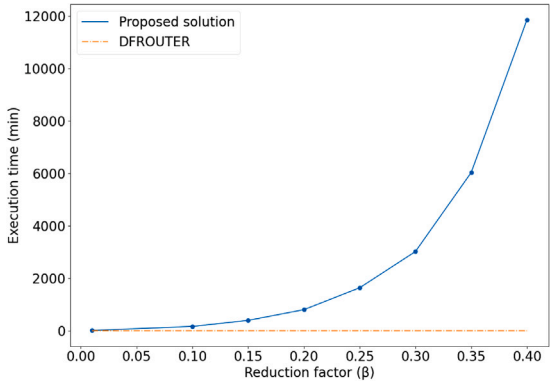


Fig. 6. Execution times when varying the value of β.

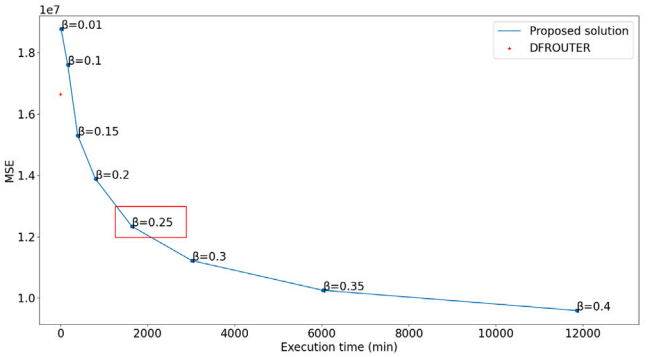


Fig. 7. Trade-off between MSE and execution time.

traffic information file generated in Algorithm 3. "$\epsilon = \{\sigma_{veh}, -\sigma_{veh}, 0$, and ϵ in $\{-\sigma_{veh}, +\sigma_{veh}\}\}$" refer to the optimized solution (see Algorithm 4).

As can be observed in Fig. 8, the unoptimized solution (Default), performs badly in terms of the number of vehicles during simulation. This is because it never reaches a steady state, and the number of vehicles keeps rising over time. Likewise, when compared to the other solutions, we notice that there are warm-up issues. While the other methods are achieving maximum values at about 9.5 min, "Default" reach it at the end of simulation time. Regarding optimized solutions, we observe that all methods reach the peak after 9.5 min. Furthermore, we observe that the steady-state has a period of 40 min for all optimized methods. In addition, $\epsilon = \sigma_{veh}$; $\epsilon = 0$; and $\epsilon = \epsilon$ in $\{-\sigma_{veh}, +\sigma_{veh}\}$ perform similarly, being the number of vehicles stable, while $\epsilon = -\sigma_{veh}$ rises

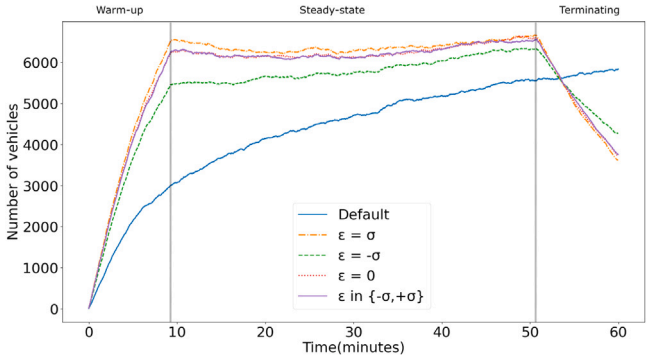


Fig. 8. Vehicles present during simulation time using different methods.

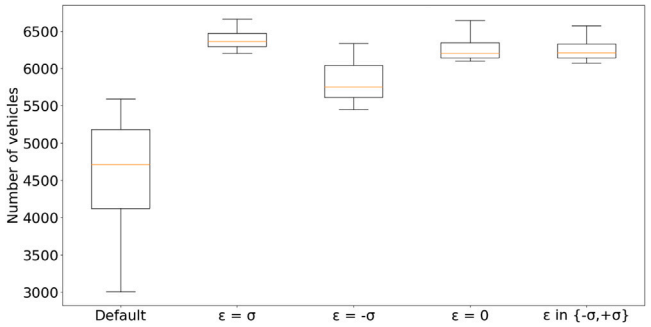


Fig. 9. Distribution of the number of vehicles during steady-state.

the number of vehicles over time. Finally, all optimized methods enter the terminating state at time $t = 50$ min, when the number of vehicles decreases substantially.

To determine which of the methods is the most stable one during the steady-state period, we analyzed the distribution of the number of vehicles during that period. Fig. 9 shows that when ϵ is σ_{veh}, 0, and ϵ in $\{-\sigma_{veh}, +\sigma_{veh}\}$, the number of vehicles are the more stable, while "Default" and $\epsilon = -\sigma_{veh}$ are not, being "Default" the least. In particular, we can see that $\epsilon = \sigma_{veh}$ has the smallest difference in the distribution. Therefore, we conclude that this is the best solution.

5.2. Traffic distribution analysis and algorithm validation

Having now selected the β value, we now proceed by observing in Fig. 10 the empirical cumulative distribution function (ECDF) that characterizes our solution, along with the distribution for DFROUTER, and the induction loop data used as reference. As mentioned in Section 3, DFROUTER has a maximum value of 13k vehicles, and it behaves like a straight line until 5k, a value beyond which it starts to make a parable. On the contrary, the function for our solution is very similar to the one for real data. Both behave with a similar parabolic function, and only differ in the maximum value (20k for real data, and 19k for our solution); also notice that there is a slight difference between the curves of the two functions for data between 150 and 250 on the Y-axis.

Data present in Table 4 and Fig. 11 complement the ECDF shown in Fig. 10 by providing further details so as to gain greater insight. As we did earlier in Section 3, we present different statistical metrics that provide a complete overview of the differences between the real data, DFROUTER, and our proposed solution.

On one hand, in Table 4 we can see that, on average, our solution introduces an average increase of 36% when compared to real data; yet, when compared to DFROUTER, we find that there is a reduction of 195%. Regarding minimum values, we observe that our solution has a minimum value of −100%. This occurs when our algorithm estimates that no vehicles will pass through a given induction loop, but instead, for the actual data, we have vehicles passing over it. On the contrary, we obtain a maximum difference compared to real data of 370%, in this case for the maximum value detected. Notice that such value becomes possible despite the fact we introduce limits to the error in Algorithm 1, since Algorithm 2 does not consider this particular type of error. Anyway, we greatly outperform DFROUTER by an 5047%, which is a very substantial improvement.

On the other hand, Fig. 11 shows the variation in the number of vehicles per detector. More concretely, it can be observed the difference in the number of vehicles for each detector between the real traffic, and our solution approach. As we can see, there is a common trend for both, and a notable difference only exists for some specific values. Likewise, by observing this figure and Fig. 2 we can determine that our solution has a better fit.

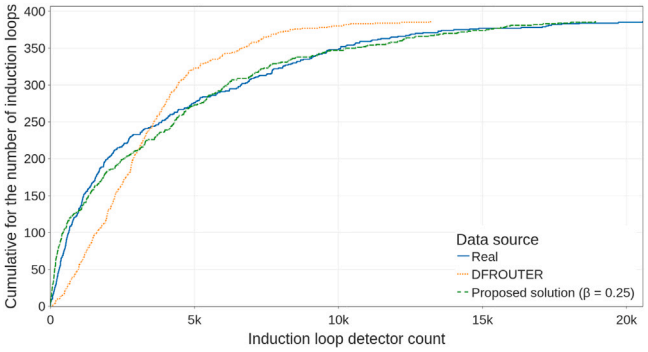


Fig. 10. ECDF of induction loop real, DFROUTER and our solution ($\beta = 0.25$) data.

Table 4
Induction loop data difference stats (Real vs. DFROUTER and Proposed solution).

Metrics	DFROUTER [%]	Proposed solution [%]	Δ [%]
Num. of detectors	386	386	–
Avg.	+231	+36	−195
Std. Dev.	+577	+132	−445
Q1	−51	−56	+5
Q2	+27	−16	−43
Q3	+244	+83	−161
Min.	−88	−100	+12
Max.	+5417	+370	−5047

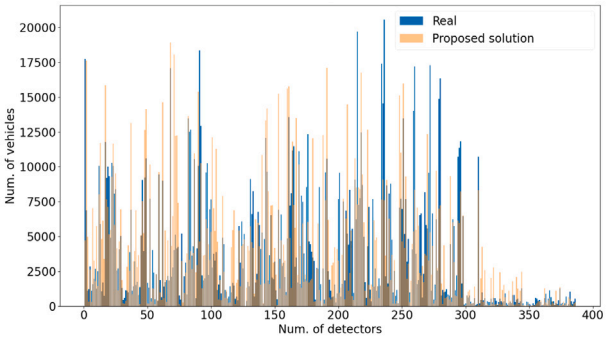


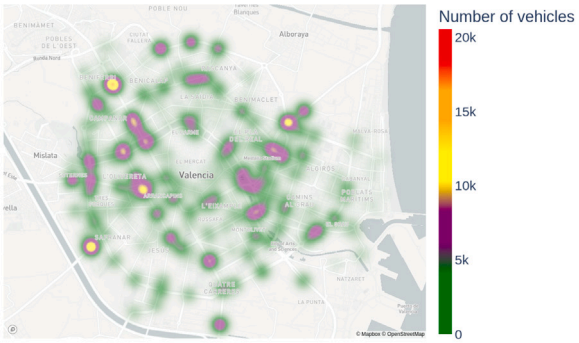
Fig. 11. Induction loop data difference (Real vs. Proposed solution).

As a last step, we now proceed to visualize the traffic distribution according to the induction loop data. To this end, we present three different heatmaps, as shown in Fig. 12. In particular, three subfigures are presented (see Figs. a, b, c), one for each dataset: real values (reference), DFROUTER, and our solution. We can observe that our solution is a better match to the real distribution than DFROUTER. This can be seen by noticing the concentration of vehicles that pass over the induction loop at the edges of the city for our solution (and the reference map), while for DFROUTER a vast majority of the data is concentrated in the city center. Likewise, as we discussed earlier, DFROUTER has a lower maximum value for detectors, the reason why its scale only grows up to 12k.

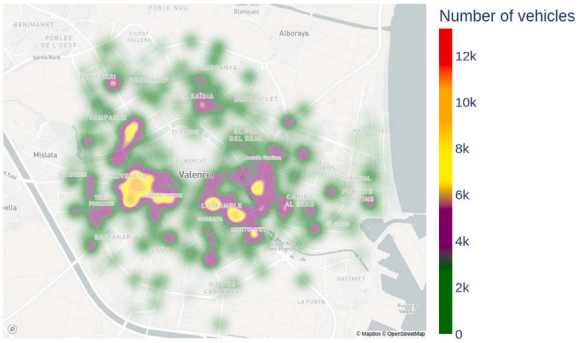
5.3. Routes characterization

As we discussed previously in Section 3, one of the problems associated with DFROUTER, in the context of a city, is the length of the generated routes. This is the reason why we analyze different statistics for the selected β versus the DFROUTER solution. Specifically, we analyze the mean value, the standard deviation, and the maximum and minimum values associated to the route length.

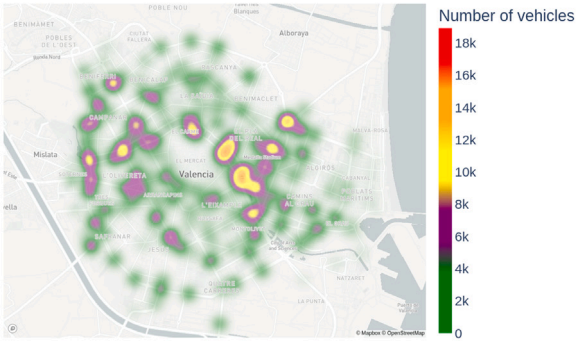
Fig. 13 shows a histogram of the length of the routes generated by our solution compared to the ones using DFROUTER. While most routes generated by DFROUTER concentrate in the range 500–1500 m, most of the routes for our proposed solution are larger, and fall within a wider range, between 2200 m and 5000 m. In addition, we can observe that DFROUTER routes start at a very small length, around 100 m, which is clearly unrealistic. On the contrary, the length of our routes starts above 2 km. In particular, we can



(a) Real data.



(b) DFROUTER.



(c) Proposed solution.

Fig. 12. Heatmap of the induction loop data distribution.

observe a cut in the histogram for our data. The reason behind this is the behavior of Algorithm 1. Since we establish a minimum distance of 2 km between geographical points, the minimum length of the route obtained is slightly higher than this value.

As we can observe, Table 5 shows in detail the routes' characterization in terms of length for both approaches. In particular, the minimum and maximum values achieved by our proposal are 2139 m and 19,030 m, respectively. In contrast, as seen earlier in Section 3, minimum and maximum values for DFROUTER are 75 m and 10,049 m. In this regard, our solution outperforms DFROUTER by 2750% and 89%, respectively. In fact, it is very important that our minimum route value has such a length because that is one of the main disadvantages of DFROUTER. It does not seem realistic that a vehicle is going to travel a route of 75 m, or even 500 m, in a city like Valencia, where finding a parking stop is quite problematic, and where several areas have parking restrictions, requiring citizens to park far away so as to promote public & green transportation (bikes, electric scooters, etc.). Regarding quartiles, we observe that our solution improves DFROUTER by 215%, being half of our routes longer than 3636 m, while for DFROUTER this value is at 1154 m. Our solution achieves a route length of 3937 m on average, while DFROUTER stays at 1291 m; hence, our improvement is 205%.

With respect to the distribution of the origin of the routes, Fig. 14 shows the distribution of the vehicles for DFROUTER and our solution. In particular, this distribution is for two specific consecutive avenues (inside the red lines): *Gran Vía Marqués del Turia*, and *Gran Vía de Ramón y Cajal*. As it can be seen, our solution distributes more spatially the origin points of the routes than DFROUTER.

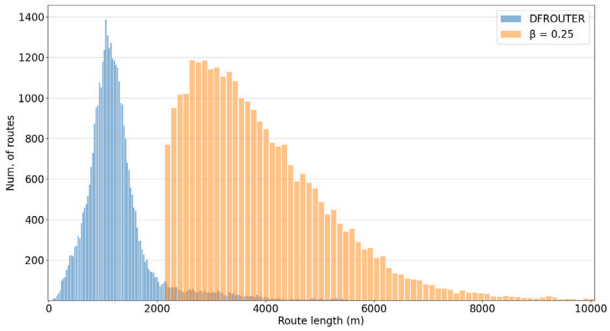
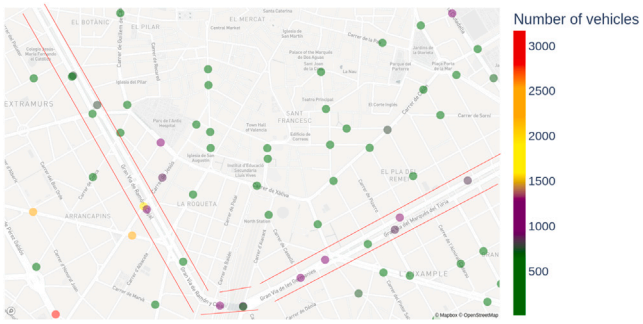


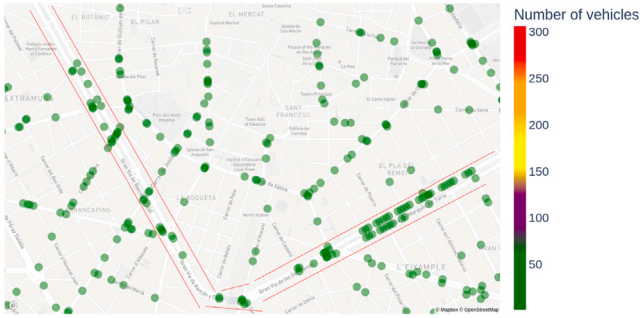
Fig. 13. Histogram for the route length when comparing DFROUTER to our solution.

Table 5
Route length statistics (DFROUTER vs. Proposed solution).

Metrics	DFROUTER	Proposed solution	Difference (%)
Avg.	1291 m	3937 m	+205
Std Dev.	739 m	1418 m	+92
Q1	917 m	2911 m	+217
Q2	1154 m	3636 m	+215
Q3	1425 m	4639 m	+225
Min.	75 m	2139 m	+2750
Max.	10,049 m	19,030 m	+89



(a) DFROUTER



(b) Proposed solution

Fig. 14. Route origin points distribution for two selected roads (*Gran Vía Marqués del Turia*, and *Gran Vía de Ramón y Cajal*).

Specifically, while DFROUTER only has 17 points along these avenues, our solution multiplies by more than 5 their results, having 88 points.

Fig. 15 shows a violin plot analyzing the number of vehicles in Fig. 14 area. Moreover, for our solution, the average number of vehicles at a point is two orders of magnitude lower than with the DFROUTER approach. Specifically, while our mean is less than 10 vehicles, their approach reaches almost 1000 vehicles. Also, our minimum and maximum values are 1 and 27 vehicles, while

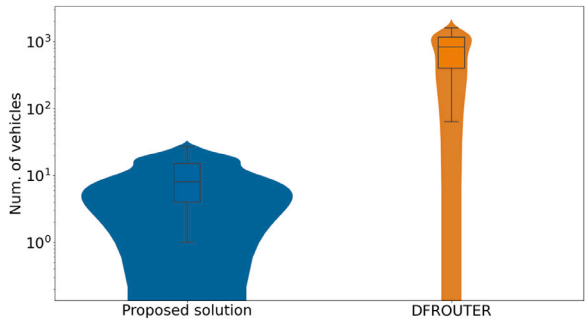


Fig. 15. Violin plot for distribution of number of vehicles (DFROUTER vs. Proposed solution).

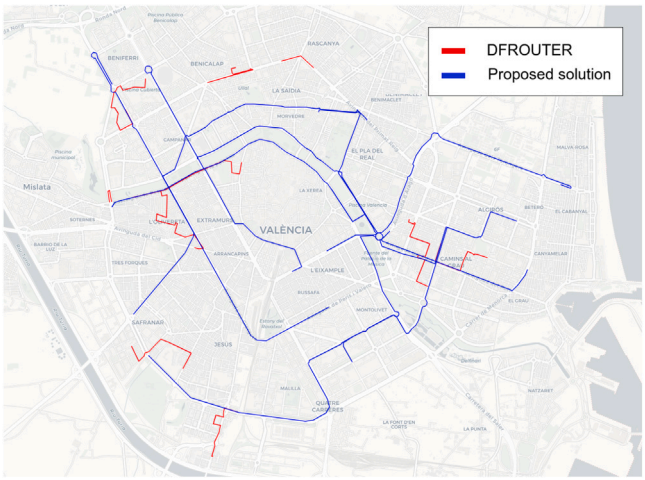


Fig. 16. Length of 10 sample routes (DFROUTER vs. Proposed solution).

for DFROUTER they are 64 and 1606 vehicles, respectively. These values are not representative, since the probability of so many cars departing from the same point is extremely low, not representing real-life situations.

To conclude, Fig. 16 displays 10 random sample routes of each approach. Through this geographical example, we can better understand the data previously presented in Fig. 13, and Table 5. As shown, DFROUTER generates routes with a small length. In this case, these routes are distributed through various parts of the city, but their length is too short for them to be representative. On the contrary, taking a closer look at the routes of our solution, we observe that their length is significantly longer. More importantly, all of our routes are representative, since they represent a significant distance being covered in the city, and it seemed more natural and usual for someone to actually use their car to take these routes.

6. Conclusions and future work

In this paper, we have seen that the process to generate representative OD traffic matrices is not a simple task. In particular, we have analyzed the generation of these matrices with the data from induction loops. For this purpose, we have assessed a widely adopted tool, DFROUTER. With it, we have demonstrated that the data obtained in terms of traffic volume and route length is not representative of the case of a city.

For this reason, we have proposed a novel approach to perform reverse engineering by generating OD traffic matrices based on induction loop data in the context of a city. Specifically, we have demonstrated that our solution improves DFROUTER in various aspects.

Firstly, we have achieved an improvement in terms of traffic volume. In particular, DFROUTER had 231% more traffic (on average) than the real measured values that were used as reference. With our solution, we were able to reduce that percentage by 195%, only introducing an average increase of 36% compared to the real data. Likewise, the MSE obtained for the selected reduction factor has improved the MSE of DFROUTER by 25%.

Secondly, a better spatial distribution of the origin points of the routes has been obtained. In particular, we have discussed an example of a representative street. In this case, we have obtained 418% more origin points than DFROUTER. So, with our solution, we are able to achieve a much better distribution for the actual departure areas.

Finally, we have obtained larger routes than those generated by DFROUTER. In particular, we have increased the average length of the routes by 205%. This means that, with our solution, we are able to obtain routes with an average length of 3937 m, while with DFROUTER it remains at 1291 m.

To sum up, we have demonstrated in this paper that our solution outperforms DFROUTER in three key aspects: traffic volume, route spatial distribution, and route length providing more realistic and accurate values for further experimentation and traffic analysis.

As future work, we plan to continue working on the improvement of the execution time of the main algorithm of our solution. Despite the results achieved are significantly better, the execution time in relation to DFROUTER remains a drawback of our solution.

Similarly, our next goal is to use the results obtained with our tool in future simulations. In particular, since we have obtained representative OD traffic matrices, we will use them to test different realistic scenarios to reduce pollution, and traffic congestion in the city.

CRediT authorship contribution statement

José D. Padrón: Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Enrique Hernández-Orallo:** Conceptualization, Formal analysis, Funding acquisition, Methodology, Supervision, Validation, Writing – review & editing. **Carlos T. Calafate:** Formal analysis, Funding acquisition, Methodology, Project administration, Supervision, Validation, Writing – review & editing. **David Soler:** Supervision, Validation, Writing – review & editing. **Juan-Carlos Cano:** Resources, Supervision, Validation, Writing – review & editing. **Pietro Manzoni:** Resources, Supervision, Validation, Writing – review & editing.

Data availability

Data will be made available on request.

Acknowledgments

This work was partially supported by R&D project PID2021-122580NB-I00, funded by MCIN/AEI/10.13039/501100011033 and "ERDF A way of making Europe", and by Programa de Ayudas de Investigación *y* Desarrollo (PAID-01-21), funded by Universitat Politècnica de València.

References

[1] UN, Transforming our world: The 2030 Agenda for Sustainable Development, United Nations, 2016.
[2] UN, Cities - United Nations Sustainable Development Action 2015, United Nations, 2022, URL https://www.un.org/sustainabledevelopment/cities/, Last accessed 28th October 2022.
[3] ITU, Smart sustainable cities, 2021, URL https://www.itu.int/en/mediacentre/backgrounders/Pages/smart-sustainable-cities.aspx, Last accessed 28th October 2022.
[4] F. Al-Turjman, C. Altrjman, Enhanced medium access for traffic management in smart-cities' vehicular-cloud, IEEE Intell. Transp. Syst. Mag. 13 (4) (2021) 273–280, http://dx.doi.org/10.1109/MITS.2019.2962144.
[5] U.K. Lilhore, A.L. Imoize, C. Li, S. Simaiya, S.K. Pani, N. Goyal, A. Kumar, C. Lee, Design and implementation of an ML and IoT based adaptive traffic-management system for smart cities, Sensors 22 (8) (2022) 2908, http://dx.doi.org/10.3390/s22082908.
[6] Z. Ning, J. Huang, X. Wang, Vehicular fog computing: Enabling real-time traffic management for smart cities, IEEE Wirel. Commun. 26 (1) (2019) 87–93, http://dx.doi.org/10.1109/MWC.2019.1700441.
[7] M.R. Jabbarpour, A. Nabaei, H. Zarrabi, Intelligent guardrails: An IoT application for vehicle traffic congestion reduction in smart city, in: 2016 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), 2016, pp. 7–13, http://dx.doi.org/10.1109/iThings-GreenCom-CPSCom-SmartData.2016.29.
[8] Z.H. Khan, T.A. Gulliver, A macroscopic traffic model for traffic flow harmonization, Eur. Transp. Res. Rev. 10 (2) (2018) 1–12, http://dx.doi.org/10.1186/s12544-018-0291-y.
[9] A. Singh, M.S. Obaidat, S. Singh, A. Aggarwal, K. Kaur, B. Sadoun, M. Kumar, K.-F. Hsiao, A simulation model to reduce the fuel consumption through efficient road traffic modelling, Simul. Model. Pract. Theory 121 (2022) 102658, http://dx.doi.org/10.1016/j.simpat.2022.102658.
[10] E. Thonhofer, T. Palau, A. Kuhn, S. Jakubek, M. Kozek, Macroscopic traffic model for large scale urban traffic network design, Simul. Model. Pract. Theory (ISSN: 1569-190X) 80 (2018) 32–49, http://dx.doi.org/10.1016/j.simpat.2017.09.007.
[11] T.V. Nguyen, D. Krajzewicz, M. Fullerton, E. Nicolay, DFROUTER—Estimation of vehicle routes from cross-section measurements, in: Modeling Mobility with Open Data, Springer, 2015, pp. 3–23, http://dx.doi.org/10.1007/978-3-319-15024-6_1.
[12] J.L. Zambrano, C.T. Calafate, D. Soler, J.-C. Cano, P. Manzoni, Using real traffic data for ITS simulation: Procedure and validation, in: 2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld), 2016, pp. 161–170, http://dx.doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0045.
[13] C.T. Calafate, D. Soler, J.-C. Cano, P. Manzoni, Traffic management as a service: The traffic flow pattern classification problem, Math. Probl. Eng. 2015 (2015) http://dx.doi.org/10.1155/2015/716598.
[14] G. Leduc, Road traffic data: Collection methods and applications, Work. Pap. Energy Transp. Clim. Change 1 (55) (2008) 1–55.
[15] N.K. Jain, R. Saini, P. Mittal, A review on traffic monitoring system techniques, Soft Comput.: Theor. Appl. (2019) 569–577, http://dx.doi.org/10.1007/978-981-13-0589-4_53.
[16] J.C. Herrera, D.B. Work, R. Herring, X.J. Ban, Q. Jacobson, A.M. Bayen, Evaluation of traffic data obtained via GPS-enabled mobile phones: The mobile century field experiment, Transp. Res. C 18 (4) (2010) 568–583, http://dx.doi.org/10.1016/j.trc.2009.10.006.
[17] Q. Ge, D. Fukuda, Updating origin–destination matrices with aggregated data of GPS traces, Transp. Res. C 69 (2016) 291–312, http://dx.doi.org/10.1016/j.trc.2016.06.002.

[18] N. Caceres, L.M. Romero, F.G. Benitez, J.M. del Castillo, Traffic flow estimation models using cellular phone data, IEEE Trans. Intell. Transp. Syst. 13 (3) (2012) 1430–1441, http://dx.doi.org/10.1109/TITS.2012.2189006.
[19] M.S. Iqbal, C.F. Choudhury, P. Wang, M.C. González, Development of origin–destination matrices using mobile phone call data, Transp. Res. C 40 (2014) 63–74, http://dx.doi.org/10.1016/j.trc.2014.01.002.
[20] C. Mallikarjuna, A. Phanindra, K.R. Rao, Traffic data collection under mixed traffic conditions using video image processing, J. Transp. Eng. 135 (4) (2009) 174–182, http://dx.doi.org/10.1061/(ASCE)0733-947X(2009)135:4(174).
[21] M. Savrasovs, I. Pticina, Methodology of OD matrix estimation based on video recordings and traffic counts, Procedia Eng. 178 (2017) 289–297, http://dx.doi.org/10.1016/j.proeng.2017.01.116.
[22] C.H. of Valencia, Intensitat transit trams, 2022, URL https://valencia.opendatasoft.com/explore/dataset/intensitat-transit-trams-intensidad-trafico-tramos/api/, Last accessed 2nd December 2022.
[23] P.A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, E. Wieß ner, Microscopic traffic simulation using sumo, in: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE, 2018, pp. 2575–2582, http://dx.doi.org/10.1109/ITSC.2018.8569938.
[24] G.A.C. (DLR), JTRROUTER, 2022, URL https://sumo.dlr.de/docs/jtrrouter.html, Last accessed 20th July 2022.
[25] G.A.C. (DLR), OD2TRIPS, 2022, URL https://sumo.dlr.de/docs/od2trips.html, Last accessed 20th July 2022.
[26] G.A.C. (DLR), DUAROUTER, 2022, URL https://sumo.dlr.de/docs/duarouter.html, Last accessed 20th July 2022.
[27] M. Behrisch, J. Erdmann, Route estimation based on network flow maximization, in: E. Wie{\ss}ner, L. L\"ucken, R. Hilbrich, Y.-P. Fl\"otter\"od, J. Erdmann, L. Bieker-Walz, M. Behrisch (Eds.), SUMO 2018- Simulating Autonomous and Intermodal Transport Systems, in: EPiC Series in Engineering, 2, EasyChair, 2018, pp. 173–182, http://dx.doi.org/10.29007/rjj7, URL https://easychair.org/publications/paper/r6Q6.
[28] G.A.C. (DLR), Routes from observation points, 2022, URL https://sumo.dlr.de/docs/Demand/Routes_from_Observation_Points.html#dfrouter, Last accessed 5th October 2022.
[29] J.L. Zambrano-Martinez, C.T. Calafate, D. Soler, L.-G. Lemus-Zúñiga, J.-C. Cano, P. Manzoni, T. Gayraud, A centralized route-management solution for autonomous vehicles in urban areas, Electronics 8 (7) (2019) 722, http://dx.doi.org/10.3390/electronics8070722.
[30] OpenStreetMap contributors, Planet dump, 2022, retrieved from https://planet.osm.org, https://www.openstreetmap.org.
[31] DGT, Anuario estadístico general, DGT (2019) 81, URL https://www.dgt.es/export/sites/web-DGT/.galleries/downloads/dgt-en-cifras/publicaciones/Anuario_Estadistico_General/Anuario-General-2019-Accesible.pdf.
[32] R.C. Hampshire, D. Shoup, What share of traffic is cruising for parking? J. Transp. Econ. Policy (JTEP) 52 (3) (2018) 184–201.
[33] G.A.C. (DLR), Netconvert, 2022, URL https://sumo.dlr.de/docs/netconvert.html, Last accessed 10th October 2022.
[34] D. Luxen, C. Vetter, Real-time routing with OpenStreetMap data, in: Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS '11, ACM, New York, NY, USA, 2011, pp. 513–516, http://dx.doi.org/10.1145/2093973.2094062, URL http://doi.acm.org/10.1145/2093973.2094062.
[35] M. Zarei, A.M. Rahmani, Analysis of vehicular mobility in a dynamic free-flow highway, Veh. Commun. 7 (2017) 51–57.
[36] N.S. Nafi, M. Hasan, A.H. Abdallah, Traffic flow model for vehicular network, in: 2012 International Conference on Computer and Communication Engineering (ICCCE), IEEE, 2012, pp. 738–743.
[37] D.L. Gerlough, A. Schuhl, Use of Poisson Distribution in Highway Traffic, Eno Foundation for Highway Traffic Control Saugatuck, CT, 1955.