User-centered road network traffic analysis with MobilityDB

Mahmoud Sakr[1,2] | Esteban Zimányi[1] | Alejandro Vaisman[3] | Mohamed Bakli[1,4]

[1]École polytechnique de Bruxelles, Université Libre de Bruxelles (ULB), Bruxelles, Belgium

[2]FCIS, Ain Shams University, Cairo, Egypt

[3]Departamento de Informática, Instituto Tecnológico de Buenos Aires, Buenos Aires, Argentina

[4]FCIS, Assiut University, Assiut, Egypt

Correspondence
Mahmoud Sakr, École polytechnique de Bruxelles, Université Libre de Bruxelles (ULB), Bruxelles, Belgium.
Email: mahmoud.sakr@ulb.be

Funding information
Argentinian Scientific Agency; Innoviris

Abstract

Performance indicators of road networks are a long-lasting topic of research. Existing schemes assess network properties such as the average speed on road segments and the queuing time at intersections. The increasing availability of user trajectories, collected mainly using mobile phones with a variety of applications, creates opportunities for developing user-centered performance indicators. Performing such an analysis on big trajectory data sets remains a challenge for the existing data management systems, because they lack support for spatiotemporal trajectory data. This article presents an end-to-end solution, based on MobilityDB, a novel moving object database system that extends PostgreSQL with spatiotemporal data types and functions. A new class of indicators is proposed, focused on the users' experience. The indicators address the network design, the traffic flow, and the driving comfort of the motorists. Furthermore, these indicators are expressed as analytical MobilityDB queries over a big set of real vehicle trajectories.

1 | INTRODUCTION

Measuring the performance of road networks is an essential task in traffic management. Administrations use performance indicators to quantitatively describe and assess road networks. They are also used for road construction planning and expenditure, and to enhance transparency, accountability, and reporting. It is hence a core component of the daily work of road authorities. Devising performance indicators is not a trivial task. On the contrary, it is a process that strongly depends on the availability of the base data that are needed to compute an indicator, the cost of collecting it, and the timeliness of the base data. Furthermore, the indicator must accurately capture the management's requirements. Following the requirements of engineering literature (Dick et al., 2017), the key performance indicators (KPIs)

that measure these requirements can be classified into *functional* (e.g., the quality of traffic flow, travel time, delay at intersection) and *non-functional* (e.g., safety, environmental quality, cost-effectiveness).

As mentioned in Hohmann and Geistefeldt (2016), there is a lack of uniform definitions and comparable classification schemes for describing the traffic flow quality (this can been also observed in AECOM (2015) reviewed in Section 2). Indicator schemes depend on the measuring entity and the data availability. For example, the Belgian Road Safety Institute publishes 25 indicators clearly focused on road safety, the average speed in different regions, the average travel speeds, and a speed index for selected roads represented as the ratio between the average speeds and the free flow speeds (VIAS, 2017). Transport Infrastructure Ireland, as of 2016 (TII, 2016), lists five classes of indicators, namely: road network, economic, road condition, safety, and accessibility and environment. For instance, the road network indicators include the total length of roads by type, the number of vehicles per road and per day, the flow at rush hours compared to the free flow, trip duration, trip distance, traffic growth rate, and the network management facilities. Another well-known indicator is the *congestion index*, which represents the increase in travel time compared to the free flow situation, published by the TomTom traffic index (TomTom, 2004) in real time for 390 cities around the world.

In spite of the examples above, existing road network performance indicators do not express the users' driving experience. They are mainly focused on the technical features of the network construction and operations. The notion of user-centered performance is missing in this domain. The indicators mentioned above account for road attributes, such as average speed and utilization. This article proposes a collection of user-centered indicators that quantify attributes of the user trips. For instance, the traffic congestion and queuing at intersections can be analyzed from two perspectives. The *network perspective*, focused on assessing, for every queuing instance, the queue duration and the resulting total delay time of vehicles. On the other hand, the *user perspective* is focused on the user trips and the trip time increase due to the multiple queuing. In other words, the existing road indicators group the observation data by road while user-centered indicators group data by user trip. With this perspective, novel indicators can be devised. These indicators can be used to improve the users' experience based on factors that actually affect them and are not considered in currently used indicators. Furthermore, this article shows how these indicators can be computed using moving object databases that account for the actual user trajectories. This is accomplished using the moving object database MobilityDB, which is an extension of the popular open-source spatial database PostGIS.

As mentioned, the choice of indicators is limited by the data availability. This is probably the reason why user-centered indicators are not currently used. Typically, road administrations install sensors on the roads to count the number of passing vehicles, their speeds, the driving lane, and so on. For instance, the Belgian Federal Public Service Mobility and Transport collects traffic data for Belgian motorways using single inductive loop detectors and cameras at about 1200 places, located before and after almost every group of on- and off-ramps (Vanhove & De Ceuster, 2003). Such sensors treat vehicles anonymously. Thus, trip-based indicators cannot be computed. Nowadays, the many geo-enabled mobile Apps and in-car GPSs produce massive streams of spatiotemporal trajectories making this class of indicators possible.

To produce the kinds of indicators introduced above, data processing tools that can efficiently handle enormous amounts of mobility data are needed. In this article, we use MobilityDB (Zimányi et al., 2019, 2020), a novel database built upon PostGIS (the spatial extension of PostgreSQL), that extends the type system of PostgreSQL and PostGIS with abstract data types (ADTs) for representing Moving Object (MO) data. *Moving objects* (Güting & Schneider, 2005) are objects (e.g., cars, trucks, pedestrians) whose spatial features change continuously in time. Moving object data generally come in the form of long sequences of spatiotemporal points. To facilitate the analysis, these sequences are split into smaller portions of movement, called *trajectories*. A *continuous trajectory* represents the movement track of an object by means of a sequence of spatiotemporal points occurring within a certain interval, together with interpolation functions that allow computing the (approximate) position of the object at any time instant. A *discrete trajectory* contains only a sequence of spatiotemporal points but no interpolation function is defined. *Moving object databases* (MOD) are databases that allow storing and querying the positions of MOs at any point in time, that is, they are able to represent continuous trajectories. To represent MOs, the definition

of appropriate data types is needed (Bakli et al., 2018). The notion of *temporal type* refers to a collection of data types that capture the evolution over time of base types and spatial types. For instance, temporal integers may be used to represent the evolution in the number of employees in a department. Analogously, a temporal point may represent the evolution in time of the position of a vehicle, reported by a GPS device, which would yield a temporal geometry of type *point*. Over these kinds of data types, MO databases can be implemented. As mentioned, MobilityDB is the most recently developed MOD. The features of MobilityDB, as well as its ability to run in a distributed environment (Bakli et al., 2020), make it appropriate to compute complex indicators like the ones we present in this article.

This article presents a collection of user-centered indicators and their computation using MobilityDB (https://github.com/MobilityDB/MobilityDB). Each indicator is explained, along with the user perspective it reflects. The indicators, expressed in SQL extended with the MobilityDB temporal types, are also presented and discussed. Finally, we include visualization examples for the different indicators that are introduced. A data analytics approach is used, where data are first cleaned, indicators are computed and validated, and finally visualized. The performance of the indicators is studied using a data set containing 114 GB of vehicle GPS tracks. These data were provided by bey2ollak (https://desktop.bey2ollak.com), an important provider of traffic news in Egypt. The company runs a mobile App with over 1.3 million users. Users agree to share their driving tracks with bey2ollak, which extracts and broadcasts a traffic index for major roadways as well as other traffic events.

The contributions of this article can thus be summarized as follows:

- A new class of network performance indicators, that take into account the user's perspective is proposed.
- The use of a moving object database (MobilityDB) as a backend for storing and querying user trajectories and computing the proposed indicators is studied and discussed. All indicators proposed are expressed in SQL extended with MobilityDB data types and functions.
- A use case based on a large real-world data set. In this case study, an end-to-end data-driven analysis, starting from the raw user trajectories and ending with the computation of the indicators, is carried out and discussed. This big data set requires some special techniques to be developed, like distributed map matching. We show that the distributed technique and framework we used allow us to obtain an improvement of orders of magnitude over the default Barefoot (https://github.com/bmwcarit/barefoot) map matching library.

The remainder of the article is organized as follows. Section 2 discusses the related work, while Section 3 presents MobilityDB in some detail to make the article self-contained. Section 4 explains the data preparation tasks needed to compute the indicators proposed in this article and also reports some metrics of the data set used for this case study. Section 5 presents the indicators together with the SQL/MobilityDB expressions that compute them, and Section 6 presents and discusses the values of these indicators computed over the use case data set. Section 7 concludes the article.

The authors would like to remark that the article is not aimed at getting conclusions from the indicator values that are computed in the experiments. This is because reference values are missing. Generally, indicators are used to compare two different road networks or the same network at two different time instants, which is not done in this article. The intention of the article is to propose methods and tools to perform such analysis.

2 | RELATED WORK

In this section, we first discuss the related work on performance indicators for the road sector. We then review the main proposals for preprocessing moving object data for obtaining trajectories for analysis. Finally, we remark that since the indicators in the present article are computed using MobilityDB, this database is presented in detail in Section 3.

2.1 | Road performance indicators

There is an extensive literature about the use of car data to compute *road performance indicators*. In Krogh et al. (2012), GPS trajectories are used to estimate the free flow speed over road segments, to compute flow indicators at intersections, and to assess the coordination between the different traffic signals that affect traffic flow on a road. A microscopic analysis is performed for two selected roads using a set of GPS trajectories. This allows exploring interesting correlations between the vehicle's position on a road, the speed, and the traversal duration. It further develops a fine-grained specification of the free flow speed and queuing. A network scale approach is presented in Meng et al. (2017), where loop detectors are used in combination with taxi GPS trajectories to derive accurate traffic volume information. The authors argue that using only GPS trajectories might not give accurate indications, as the source vehicles remain a small and possibly biased sample of the whole traffic. In Bechtel et al. (2018), user tracks collected from cell phones and GPS are used to evaluate a congestion metric at thousands of roadway bridges in New Jersey, USA. None of these works include the user's perspective to compute the proposed indicators.

Traditionally, road administrations have developed their own schemes of performance measures. A study was commissioned by the EU Commission directorate general Mobility and Transport, aimed at establishing common KPIs for the road transport and ITS sectors (AECOM, 2015). The study reviews 228 indicators in use in EU countries. This big number shows that there is little consistency between the indicators used by the different countries and administrations. The study listed nine main reasons for this, including the different types of data and their different availability levels. The surveyed data sources generally lack GPS tracks. The study concludes with a set of recommended KPIs. Both, the surveyed indicators and those recommended by the study, lack the user's perspective. In 2013, the World Bank performed a study about the traffic congestion in Cairo, the capital of Egypt (Nakat, 2013). Manually classified traffic counts were collected at 24 locations and survey cars were used to collect floating car data. The data collection was done during the peak periods between 7:00 and 11:00 and between 15:00 and 19:00. Two indicators were computed in this study: traffic volume and average speeds.

The notion of *customer perspective* is introduced in Hohmann and Geistefeldt (2016) as a method of rating the traffic flow. Test persons were asked to attend test rides, and to continuously give their subjective rating of traffic. In these tests, the average travel speed was selected as the basic parameter used to define different traffic states. The authors argue that this decision was taken to facilitate user's understanding of the road traffic experience. In this experiment, 15,000 individual evaluations for 13 freeway segments and 37 urban road segments were taken into account and a subjective evaluation of the traffic flow quality was given by individuals. Finally, the authors come up with a categorical classification scheme just based on the average travel speed sensation provided by the users. Although an interesting first step to account for the users' perspective, the work in the present article goes far beyond this effort, providing a wider range of indicators that are tested over very large data sets making use of moving object technology.

In conclusion, some works use GPS trajectories for assessing the road network performance, although they are mainly focused on assessing network attributes. A scheme of indicators that assesses the network from the user perspective is still missing.

2.2 | Data preprocessing and preparation

Using GPS trajectories for computing performance indicators is not straightforward. The data come with many inaccuracies, such as random, missing, and repeated signals. However, data cleaning is not always detailed in research articles. For instance, considering the works mentioned above, Krogh et al. (2012) only mention that the work selected trajectories that follow the chosen paths, without any further explanation; also Meng et al. (2017) just report that map matching was used to infer the average speeds. We argue here that data cleaning is essential to trajectory analysis and that it deserves a detailed treatment. As we explain later in this article, two-thirds of the data set used

(a) input trajectories (b) cleaned trajectories (c) map matched trajectories

FIGURE 1 Trajectory preprocessing effect for a portion of the Cairo use case.

here are cleaned out, due to many reasons. The effect of cleaning in our use case is illustrated in Figure 1 for a small area of Cairo, showing to what extend the data set is reduced after the cleaning process takes place.

In Parent et al. (2013), a generic three-step trajectory preprocessing methodology is proposed. Trajectories are first cleaned, then map matched, and finally compressed. For cleaning, an implementation using regression-based smoothing and outlier removal is presented in Yan et al. (2013). Yet, the general practice in trajectory data cleaning is to manually investigate the data set at hand, with the goal of spotting errors and cleaning them (Fu et al., 2016). Existing works limit the cleaning to GPS signal errors. In the experiments presented in this article, many errors that are not treated in the existing literature are found in the data set. They are addressed in detail in Section 4.

Map matching is a well-known technique to the GIS community. It refers to the transformation of absolute GPS coordinates into a sequence of road segments. It constrains the raw GPS observations to the spatial coverage of the roads, to the speed limits, and to the road directions. GPS points that violate these constraints are skipped according to the tolerance of the map matching algorithm. As a result, the GPS observations are either aligned to the road network or removed. There are two flavors of map matching, online and offline (Wei et al., 2013). The former incrementally matches incoming GPS observations using only the data and the previous history. The latter works for trajectories and might look ahead for the following observations. Big data sets currently available require new map matching algorithms and techniques that can efficiently cope with the new requirements imposed by such large data sets. Recent work by Peixoto et al. (2019) introduce a framework for parallel map matching using Spark technology. Distributed algorithms for big data map matching are also proposed in Almeida et al. (2016) and Francia et al. (2019). Our approach in the present article extends the Barefoot library to allow efficient parallel execution. We explain this technique in Section 4.2, where we show that we obtained an improvement of orders of magnitude over the default Barefoot map matching library.

3 | MOBILITYDB

Given that the indicators are computed using the MobilityDB MO database, this section presents a brief overview of this database to make the article self-contained. Further details can be found in the system's documentation (Zimányi, 2020).

MobilityDB defines *temporal types* for handling objects whose values change over time, for example, stock prices, temperature, and of course moving objects. For this task, MobilityDB provides the following temporal types: `tbool`, `tint`, `tfoat`, `ttext`, `tgeompoint`, and `tgeogpoint`. These temporal types are based on the corresponding base types provided by PostgreSQL, and on the `geometry` and `geography` base types provided by PostGIS. Temporal types are initially built from a discrete set of values, and represent the evolution of the value of an object during a sequence of time instants. Since MO databases represent a continuous function, values between discrete time instants are interpolated using either a stepwise or a linear function.

All temporal types are based on four *time types*: the `timestamptz` type provided by PostgreSQL, and three new types, namely `period`, `timestampset`, and `periodset`. The `period` type is a more efficient implementation of the `tstzrange`

type provided by PostgreSQL. The former has a fixed length and disallows empty periods while the `tstzrange` type has a variable length and allows any value. The `timestampset` type is a collection of one or more `timestamptz` values, and the `periodset` is a non-empty collection of ordered and non-overlapping `period` values. Two *range types* are also introduced, namely `intrange` and `floatrange`. These ranges are pairs of upper and lower bounds that can interact with out-of-the-box operators like < @ (contained in).

The type system defines only the building blocks of MobilityDB's functionality. There is also a *collection of functions* that can be classified as follows:

(a) Functions and Operators for time types and range types: Perform different operations on time types and ranges. These are generally polymorphic functions that receive a data type and perform different calculations. For example:

```
1  startTimestamp({timestampset, periodset}): timestamptz
2  endTimestamp({timestampset, periodset}): timestamptz
3  timespan({timestampset, period, periodset}): interval
```

(b) Functions and Operators for Temporal Types: Perform different operations on Temporal types. These operations apply the traditional operations at each instant and yield a temporal value as result. For example:

```
1  cumulativeLength(tpoint): tfloatseq
2  speed(tpoint): tfloats
3  nearestApproachDistance({geo, tpoint}, {geo, tpoint}): float
```

A spatiotemporal object is generally built from a series of discrete timestamped location points. However, since trajectories are continuous, to represent and query the state of the object at any time *MobilityDB* interpolates the points using a linear function, generating a continuous approximation. This interpolation allows estimating the state of an object at any instant of a given interval. MobilityDB provides two different spatiotemporal object types, namely `TemporalGeometryPoint` and `TemporalGeographyPoint`, which correspond to PostGIS' data types. The difference between the two is the reference system: `geography` points use a geodesic reference system and offset accuracy for complexity, while `geometry` points use a Cartesian reference system and allow calculation of speed and other distance-related metrics.

Once the object's evolution is stored in the system, MobilityDB provides multiple functions to access and manage its values. Some of these functions such as `speed` and `direction` return temporal values, allowing the user to take full advantage of the database's type system. We illustrate the use of MobilityDB through some simple example queries next.

The first example query deals with temporal integers. The first `SELECT` statement below constructs two `tint`s and then adds them. The resulting value is also a `tint`. Results are also shown, for clarity.

```
-- Temporal addition
SELECT tint '[1@2001-01-01, 1@2001-01-03)' + tint '[2@2001-01-02, 2@2001-01-05)';
--"[3@2001-01-02, 3@2001-01-03)"
```

Our Second example query computes the intersection between a temporal point and a given geometry, using the `SELECT` operation. The resulting value is a temporal Boolean value `tbool`.

```
--Temporal intersection
SELECT tintersects(tgeompoint '[Point(0 1)@2001-01-01,
        Point(3 1)@2001-01-04)',
        geometry 'Polygon((1 0,1 2,2 2,2 0,1 0))');
--"{[f@2001-01-01, t@2001-01-02, t@2001-01-03],
-- (f@2001-01-03, f@2001-01-04]}"
```

Finally, the following script shows how values are stored in a table. The `SELECT` statement returns the value of MOs (e.g., cars) at a specific timestamp, resulting in a point. The table `Trips` contains the trajectories of two MOs.

```
CREATE TABLE Trips(CarId integer, TripId integer, Trip tgeompoint);
INSERT INTO Trips VALUES
(10, 1, tgeompoint '{[Point(0 0)@2012-01-01 08:00:00,
Point(2 0)@2012-01-01 08:10:00, Point(2 1)@2012-01-01 08:15:00)}'),
(20, 1, tgeompoint '{[Point(0 0)@2012-01-01 08:05:00,
Point(1 1)@2012-01-01 08:10:00, Point(3 3)@2012-01-01 08:20:00)}');
-- Value at a given timestamp
SELECT CarId, ST_AsText(valueAtTimestamp(Trip, timestamptz '2012-01-01 08:10:00'))
FROM Trips;
-- 10;"POINT(2 0)"
-- 20;"POINT(1 1)"
```

4 | TRAJECTORY DATA PREPARATION

This section describes the end-to-end data analytics pipeline that we propose for computing indicators from a big data set of user trajectories. This pipeline goes from data acquisition to the computation of the indicators. Figure 2 summarizes the phases and tools that we used in our case study. Data acquisition, loading, and cleaning are presented in Section 4.1. These processes prepare data for the map matching process described in Section 4.2. Finally, from the map-matched trajectories, continuous trajectories are produced and loaded into MobilityDB as moving objects `trips`. This is studied in Section 4.3. The computation of the indicators using these trajectories is studied in Section 5.

4.1 | Data loading and cleaning

As mentioned in Section 1, the work reported here uses a data set consisting of 114 GB of vehicle GPS tracks shared by a traffic news provider in Egypt. The data come as multiple csv files containing 832 million GPS recordings in the following format:

`<id, latitude, longitude, speed, tripId, accuracy, battery, bearing, created, transId, timeonserver>`.

The first step of the data preparation task consists in *loading* the data into the database system, PostgreSQL in our case. This is a challenging task because of the data volume and errors such as missing values, illegal characters, and inconsistent formatting of dates and strings. The PGLoader tool (Fontaine, 2014) developed for PostgreSQL is used for this task. PGLoader parallelizes reads and writes over multiple threads, uses shared queues between them, and performs them in patches. All of these allow a very fast loading process.

The original data contain two timestamps: `created` and `timeonserver`, corresponding to valid time and transaction time, respectively, in temporal database terminology (Tansel et al., 1993). Recall that valid time refers to the time when an event occurs in the real world, while transaction time refers to the time when a data element is recorded in the database. In this work, valid time is used. Furthermore, the GPS logging system, from which the data are obtained, contains the complete trajectory in a single transaction. This means that all the GPS observations of a trajectory may have the same transaction time. The data also contain multiple keys: `id` is a unique sequence over the whole file; and both (`tripId`, and `transId`) form a composite key, that helps identifying different trips. This composite key is kept, along with the valid time and the coordinates. The remaining attributes are removed to reduce the data size since they are not needed for the analysis. Specifically, the speed attribute is removed and more accurately calculated during the map matching process, as explained later in this section. The attributes kept for the

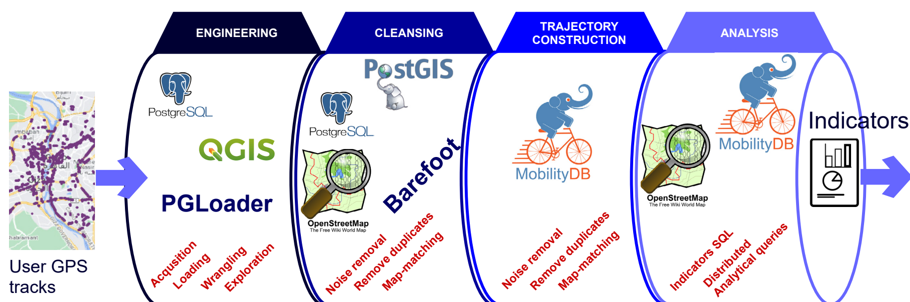

FIGURE 2 Methods and tools used in the data preparation pipeline.

analysis are, thus: `gps(tripId, transId, longitude, latitude, created)`. The `accuracy` attribute is not directly used, but it is reflected in the map matching process results (see Table 1).

After loading the data, a *cleaning* process is carried out, due to usual errors in the data set. The whole data cleaning process is implemented as SQL queries. The steps of the process are as follows:

1. Remove partially duplicated trajectories. Around 19,000 trajectories are partially repeated in the data. That is, for a part of the trip, the same combination of `tripId`, `transId`, `longitude`, `latitude` is repeated with different `created` timestamps. Although it could be possible to distinguish between the original and the duplicated trajectories (e.g., using the `created` attribute for splitting), this is a hard and error-prone task; therefore, we decided to remove these trajectories.
2. Remove duplicated trajectories. Around 183 K trajectories are repeated in the data, with different keys. That is, two different key pairs `(tripId, transId)` have exactly the same sequence of `longitude`, `latitude`, with different time shifts. Note that even if two vehicles drive together on the same route, it is extremely improbable that their GPS sensors will log the observations at the same sequence of coordinates. This redundancy in the data suggests that either the GPS logger or the App server repeatedly store the same data with different keys. To clean this, only the trajectory with the smallest initial timestamp is considered as the original and thus kept in the data set, while the remaining copies are removed.
3. Remove duplicated trajectories that have different keys. Similar to the previous error, some trajectories are repeated (yet *partially*), with different keys. From every such group, the trajectory with the longest sequence of GPS observations is kept. Note that this is different than the first case above, because here the keys of the repeated sub-sequences are different.
4. Removing stationary and short trips. Stationary and short trips are filtered out to avoid the bias in results that they may create. Stationary trips reduce the speed average in relevant calculations. Short trips, on the other hand, do not reflect real trips. It could be the case, for instance, where the application users initiate the logging of a trip by accident, while not driving. Stationary trips typically appear as a sequence of GPS points that jump around a parking location. They are identified by their small bounding box, and the high fluctuation of speed. To identify short trips, an histogram of the number of observations per trip is built. The trips with less than 50 GPS observations are deleted (about 162, 000 trips). Finally, the trips that are spatially out of the analysis area, which covers the extent of Cairo and Alexandria, are also deleted.

TABLE 1 Statistics for the distance between the original points and the map-matched points

Mean	**12.1431**	**Median**	**8.3570**
First quartile	3.5610	Standard deviation	10.6561
Third quartile	18.9519	Variance	113.5514

On the whole, the cleaning process took the data size from 832 M down to 377 M GPS points, and from 968 M to 579 K trips.

4.2 | Map matching

A trip trajectory in GPS file format is a sequence of absolute coordinate values. These values typically have inaccuracies due to weak and lost GPS signals. These inaccuracies can greatly affect calculations of trip attributes such as speed, heading, and travel distance, among others. Map matching aims at fixing these kinds of errors, aligning the raw observations to the road network. It maps the input GPS observations into coordinates located on the road network. It additionally enriches the coordinates with the identifiers of the road segment, and the relative position from its start. Figure 3 illustrates a raw trajectory and its map-matched counterpart. A good map matching algorithm can fix reasonable GPS shifting errors, and exclude outlier points. Additionally, data are enriched with the map context.

Barefoot (GmbH, 2015) is the tool used for map matching in this work. We used a base map downloaded from Open Street Maps (OSM). Barefoot is a Java library that integrates with PostGIS and provides both offline and online map matching. It implements a Hidden Markov Model (Newson & Krumm, 2009). A point in the raw data is matched to a point on the map based on two probability values. First, the *emission* probability quantifies the opportunity that a given position on the map is observed with the GPS coordinate in hand. Thus, a GPS observation can be associated with many map positions with different emission probabilities, called the *candidate vector*. The *transition* probability quantifies the opportunity of reaching a map point from the candidate vector, given a map point from the candidate vector of the previous observation in the trajectory. Points that cannot be map matched (i.e., noise) are skipped. In this data set, 83 million such noisy GPS points were found, representing almost 70% of the data after cleaning. Certain situations lead to splitting the input trajectory into multiple disconnected map-matched trajectories, for instance, when many consecutive GPS observations cannot be matched, when there is a long temporal gap between observations, and when the spatial distance between observations is too long.

After map matching, the transition between two consecutive GPS points of a trip is matched to a sequence of road segments, that the vehicle has most probably traversed between the two points. This is represented in two relations `matchMaster` and `matchDetail` shown in Figure 4. A tuple in the `matchMaster` relation represents a pair of consecutive GPS observations of the same trip. The `id` attribute is a counter, added as a primary key. The `tripId` and `transId` attributes are kept, making it possible to join the `matchMaster` table with the input relation. Attributes `p1` and `p2` are two temporally consecutive GPS points from the same trip in the input. Attributes `t1` and `t2` are, respectively, the timestamps of the former. Attributes `mapPoint1` and `mapPoint2` are the coordinates that result from map matching `p1` and `p2`. Since `mapPoint1` and `mapPoint2` can be located on different road segments, the `matchDetail` relation represents the road segments that are traversed between them. One tuple in `matchMaster` joins with at least one tuple (probably more) in `matchDetail`. Starting from `mapPoint1` in ascending order of `matchDetail.id`, the relation `matchDetail` stores every traversed road segment until `mapPoint2`. For every road segment, it stores its `wayId` which can be used to join with the base OSM map. It also stores the segment length, for speeding up queries. The `startFraction` and `endFraction` attributes define the part of the road segment that has been traversed between `mapPoint1` and `mapPoint2`. The first and the last segments (i.e., the ones that, respectively, contain `mapPoint1` and `mapPoint2`) may be partially traversed. Every segment in-between them will be fully traversed, that is, it will have values `startFraction=0` and `endFraction=1`. Finally, the speed and the time spent by the vehicle on this segment are computed and denoted `durationMillisec`, assuming a constant driving speed between `mapPoint1` and `mapPoint2`. Again this helps speeding up queries. This speed is computed as the ratio of the network distance between `mapPoint1` and `mapPoint2` over the driving time `t2-t1`. Since `mapPoint1` and `mapPoint2` are coordinates corrected from the original GPS coordinates `p1` and `p2`, respectively, this speed value is more accurate than the one that was originally given in the input data.

FIGURE 3 Left: the original track, right: the map-matched track.

The tuples that belong to a single trajectory in the matchDetail relation represent, among others, the speed curve of the trajectory. It can be abstracted as a function speed(t)→real, that maps every time instant in the trajectory lifetime into a speed value. In the data set used in this work, as expected, the speed function contains noise and spikes. Map matching reduces such errors, although it does not completely eliminate them. To reduce the effect of these errors, a smoothing curve is applied to the speed function. For this, a k-nearest neighbor kernel average was used, with $k = 7$, that is, the three preceding and the three following observations are considered. This window size was chosen based on the sampling rate in the data. Shorter windows may not smooth the curve as required and larger values may smooth the curve in excess. Since about 80% of the trips have a sampling period between 1.5 and 20 s, the chosen window maps to a duration ranging between 10.5 and 140 s, a reasonable temporal range for smoothing the speed of a vehicle. Thus, every speed value in the `matchDetail` relation is updated with the average speed of its window. In the remainder of the article, mentions to the trajectory speed refer to this smoothed speed. The SQL query that implements the above is:

```
WITH matchDetailSmoothed AS (
    SELECT d.id, AVG(d.speed) OVER (PARTITION BY m.tripId, m.transId
        ORDER BY d.id ASC ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING) AS speedSmoothed
    FROM matchDetail d JOIN matchMaster m ON m.id = d.masterId)
UPDATE matchDetail a
SET speed= speedSmoothed
FROM matchDetailSmoothed b
WHERE a.id= b.id
```

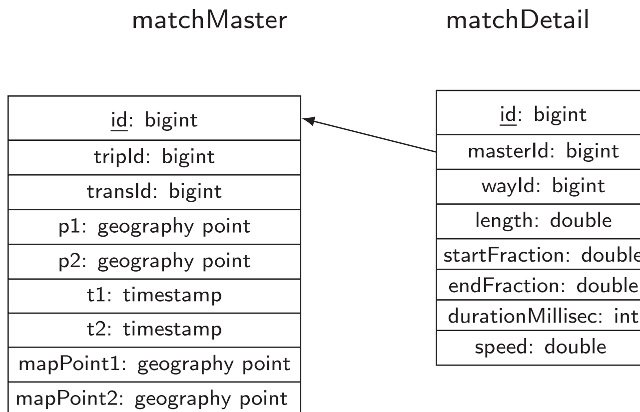

FIGURE 4 The analysis database schema.

4.3 | Distributed map matching

The map matching process is extremely demanding in terms of execution time, particularly for high data volumes as the one at hand. For this work, then, the *map matching process is distributed over a cluster of machines*, as shown in Figure 5. The cluster consists of one coordinator node and several worker nodes. All nodes have the same software stack, consisting of PostgreSQL, PostGIS, MobilityDB, PgRouting, and DbLink. The PgRouting extension is used to compute the shortest path between two spatial points. The DbLink extension is used to send and receive an SQL query from one node to another. This is used to transfer the input data table and the final results after doing the map matching. The *Barefoot algorithm has been modified* to make it work with different data tables in parallel. Barefoot itself does the spatial searching in parallel although this parallelization does not exploit all the existing cores of the machine. Therefore, only half of the machine cores are used for the searching algorithm, and the remaining cores are used to map match trajectories in parallel.

The coordinator partitions the input trajectories using the range partitioning method, producing a set of separate partitions. Each partition contains trips within a range of trip and transaction identifiers. The DbLink extension opens a connection with the worker nodes and transfers the partitions into separate tables on the worker nodes. The number of partitions in every worker is decided based on metadata collected from the workers. For instance, if the worker contains 12 cores, then six partitions can be transferred. The new update for the Barefoot is packed into an a.jar file and transferred using a shell command to the worker nodes. Once everything is ready, the coordinator sends a shell command to the worker nodes to start the map matching in parallel. When the map matching is done for one trajectory, the result is stored in the analysis database whose schema is depicted in Figure 4.

The distributed map matching architecture achieves significant speed up over the default single-node Barefoot. In this case, 579,120 trajectories containing 377 million spatiotemporal points were map matched in 3 days and 4 h. Every minute *127 trajectories* are map matched on a cluster of one coordinator and four workers. We remark that *the default Barefoot is only able to match four trajectories* on average every minute on a machine with12 cores. Therefore, it would have required more than 3 months to finish the map matching. The choice for using Barefoot was due to its accuracy and the possibility of parallelizing the map matching.

As an example of the results of this process, Table 1 shows the statistics concerning the distance (in meters) between the original points and the map-matched points.

4.4 | Trip generation

To load the trajectories into MobilityDB, the `tgeompoint` data type is used to represent a trajectory as a sequence of interpolated points. The `matchMaster` table describes the movement of the object as a set of units. Each unit describes the movement from a point `p1` to a point `p2` in a time interval `(t1,t2)`, which can be defined as a second. The end point `p2`, with timestamp `t2`, represents the starting movement of the next unit. Therefore, a conversion of each unit into an instant (i.e., point and time) is required. This is the base for constructing a `tgeompoint`. The following SQL query generates the trajectory instants:

```
CREATE TABLE tripInstants AS
SELECT tripId, transId, t1 AS t, ST_Transform(ST_SetSRID(p1, 4326), 22992) AS geom
FROM matchMaster
UNION ALL
SELECT m1.tripId, m1.transId, m1.t2 AS t, ST_Transform(ST_SetSRID(m1.p2, 4326), 22992) AS geom
FROM matchMaster m1
INNER JOIN (
        SELECT tripId, transId, MAX(t2) AS maxTend
        FROM matchMaster
        GROUP BY tripId, transId
) m2 ON m1.tripId = m2.tripId AND m1.transId = m2.transId AND m1.t2 = m2.maxTend;
```

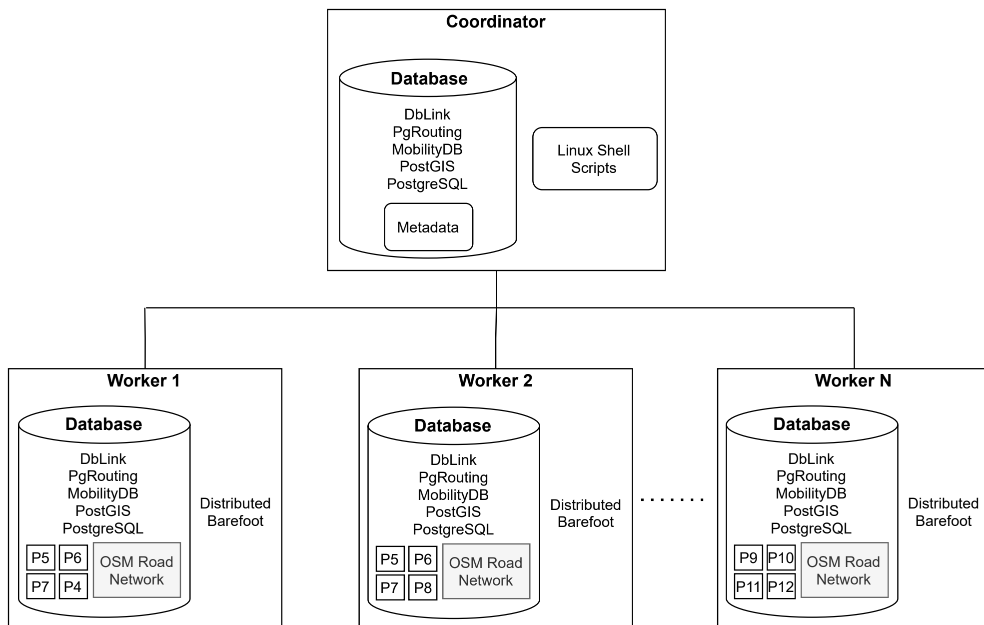

FIGURE 5 Distributed map matching architecture.

The schema of the `trips` table is as follows:

$$\text{trips}: \ <\text{tripId}: \ \textbf{int}, \ \ \text{transId}: \ \textbf{int}, \ \ \text{trip}: \ \text{tgeompoint}>$$

With this schema, and after generating the instants the trip is simply created using the following query:

```
CREATE TABLE trips AS
SELECT tripId, transId, tgeompointseq(array_agg(tgeompointinst(geom, t) ORDER BY t)) AS trip
FROM trip_instants
GROUP BY tripId, transId
```

Using MobilityDB to store the trajectories is much simpler than using complex SQL queries using PostGIS. For example, the `speed` function allows calculating the speed of a moving object at every timestamp. The results are stored in a MobilityDB `temporalfloat` type. Therefore, the speed values can be analyzed to detect, for example, the changes in the gear type. The `matchMaster` relation contains 178 M tuples, and the `matchDetail` relation contains 286 M tuples. These relations represent the map-matched trajectories of 579 K trips. For MobilityDB, only the trips table is used, containing 579 K trajectories. The total travel distance of all trips is 2.1 M kilometers. The time span ranges from March 2016 through April 2017. Finally, indexes are created to speed up the queries.

5 | USER-CENTERED INDICATORS

In this section, we introduce and describe the proposed user-centered indicators, together with the MobilityDB queries that compute them, which are explained in detail. We finally show how they can be visualized using typical GIS tools.

Indicators are characterized in two classes as follows:

1. Road network indicators.
   - Deviation from the shortest-distance path.
   - Deviation from a straight-line path.
2. Road traffic indicators
   - Queuing time.
   - Travel time between landmarks.
   - Deviation from free flow.
   - Rate of gearing.

These indicators are explained next.

5.1 | Road network indicators

The two indicators in this class are as follows: (a) *Deviation from the shortest distance path*, which aims at measuring to what extent a driver on a road network follows a path other than the shortest one; (b) *Deviation from a straight-line path*, which measures for a road network, to what extent trips between origin and destination differ from a straight-line trajectory. The former is related to the road and traffic conditions, while the latter is also related to the network design.

5.1.1 | Deviation from the shortest distance path

In ideal road and traffic conditions, drivers would follow the path with shortest distance. However, because of poor road conditions or high traffic on the shortest distance path, drivers often follow alternative paths. This has economic and environmental consequences, like more fuel consumption and higher pollution. This indicator aims at measuring to what extent users deviate from the shortest path. The two quantities involved in the calculation are the trajectory length, and the shortest distance path. The deviation value is then calculated per trajectory as the ratio (trajectory length/shortest path length).

In MobilityDB, the total traveled distance of each trip can be computed using the `length` function on the `trip` column.

```
SELECT tripid, transId, length(trip) FROM trips
```

This query retrieves the distance in meters. The computation of the shortest-path distance is performed by the function `pgr_dijkstraCost`, provided by the *pgRouting* extension of PostGIS (pgRouting Community, 2013). The SQL query in MobilityDB is as follows:

```sql
-- Shortest distance between source and destination

CREATE TEMPORARY TABLE IF NOT EXISTS tripSourcesTargets AS
    SELECT tripId, transId, length(trip) AS actualDistance,
        (SELECT source
        FROM bfmap_ways
        WHERE ST_Intersects(startValue(trip), geom)
        LIMIT 1
        ) AS source,
        (SELECT target
        FROM bfmap_ways
        WHERE ST_Intersects(endValue(trip), geom)
        LIMIT 1
        ) AS target
    FROM Trips;

-- Main Query:

WITH pgr_dijkstraCost_call AS (
    SELECT * FROM pgr_dijkstraCost(
        'SELECT gid AS id, source, target, ST_Length(ST_Transform(geom, 22992)) AS cost
         FROM bfmap_ways',
        'SELECT DISTINCT source, target
        FROM TripSourcesTargets WHERE source IS NOT NULL AND target IS NOT NULL', true)
)
SELECT t1.tripId, t1.transId, t1.partId, t1.actualDistance, t2.agg_cost
FROM tripSourcesTargets t1, pgr_dijkstraCost_call t2
WHERE t1.source = t2.start_vid AND t1.target = t2.end_vid;
```

The first part calculates and stores in a temporary table the source and target line segments of each trip, which are used when calling the `pgr_dijkstraCost` function. In addition, the length of each trip is computed, and the actual distance between source and target is returned. To compute the source and the target, the `ST_Intersects` function is applied, between the start or end point of the trip and every line segment in the network. The second part shows a Common Table Expression (CTE) query that calculates the shortest distance between every source and target. Then, the actual length and shortest distance of each trip are returned.

We computed this indicator over the Cairo data set. The preliminary results showed that trips were, on average, 326% longer than their shortest path distance. Since this difference suggested some error, a closer inspection revealed that some trips end at locations near to their start, that is, they are round trips. Thus, they were removed from the data set, filtering source and target points less than 2 km apart. The remaining trips deviated from their shortest path by 160%. More statistics concerning this indicator are shown in Table 2.

5.1.2 | Deviation from a straight-line path

This indicator aims at assessing the convenience of the design of the road network to users. The shortest distance between two points in the free space is the length of the straight line that connects them, which would be the ideal path from a driver's point of view. While it is impossible to structure the street network like this, the closer to the straight-line distance is the network, the better and more convenient results the network structure to users.

To compute this indicator, we first need to compute the ideal straight-line trajectory. For this, we use The Euclidean distance between the start and end points of a trajectory. This will be compared against the actual user trajectory

over the network. A distance-preserving spatial projection needs to be applied first. For this data set, the EPSG projection 32,636—WGS 84/UTM zone 36 N is used, which is adjusted to the zone of the data set, namely, Cairo and Alexandria. The total length of the trajectory is also calculated, as in the previous indicator. For each individual trajectory, we then compute the indicator as the quotient (trajectory length/Euclidean distance[start, end]).

In MobilityDB, the computation is straightforward, since the trip distance and the straight-line distance between the source and target are obtained using MobilityDB functions. The MobilityDB query is given next.

```
SELECT tripId, transId, ST_Distance(startValue(trip), endValue(trip)) AS straightLineDistance,
    length(trip) AS actualDistance
FROM trips;
```

The query calculates the actual distance for each trip using the `length` function. For the straight-line distance, the `ST_Distance` function of PostGIS is used. This function takes the starting and ending points of the trip and returns the distance between them.

Similar to the previous indicator, round trips are excluded from the result. Over the Cairo data set, the deviation of the remaining trips from their straight-line distance is 240%. Other statistics about this indicator are shown in Table 3. It can be noticed that the number of trajectories in the table is higher than the one in Table 2, because for some trajectories it was not possible to compute the shortest path due to the clipping of the map.

5.2 | Road traffic indicators

The four indicators in this class are as follows: (a) *Queuing time*, which measures the time that a moving object in a trajectory spends at a traffic queue; (b) *Travel time between landmarks*, that measures the average travel time between predefined points of interest in a road network; (c) *Deviation from a free flow*, that measures the difference between the travel time of a moving object trajectory, with respect to the travel time under free flow traffic conditions; and (d) *Rate of gearing*, that measures the number of changes of gear per kilometer.

5.2.1 | Queuing time

The queuing time indicator quantifies the time lost per trip due to queuing on a road network. Queuing events occur mainly at intersections and at u-turns. They happen when the drivers have to reduce speed, wait for traffic lights, or wait for the preceding vehicles to exit the road. Queuing can also happen due to obstacles on the road, accidents, security checks, and wait at toll gates, among other reasons. A queuing event is characterized by low speeds and frequent events of accelerations and stops, that occur during long periods. Note that this indicator differs from well-known indicators explained in Sections 1 and 2, that quantify the total time lost due to queuing at a certain road facility. To compute the queuing time indicator, queuing events need to be identified within individual trajectories, along with their start and end times. For this, periods of at least 5 min of speed less than 15 km/h are detected. Short intervals of speedups during such periods are ignored or smoothed. The events that last for more than 30 min are discarded, since these are more likely parking events.

To compute the queuing time using MobilityDB, the speed of the trip is calculated, and the waiting time between every two consecutive points is checked. The query reads:

```
SELECT tripID, transId, gettime(atRange(speed(trip), floatrange '[0, 15]')) AS queueStartEnd
FROM Trips
WHERE timespan(atRange(speed(trip), floatrange'[0, 15]')) BETWEEN interval '5 minute' AND interval '30 minute';
```

TABLE 2 Traveling distance deviation from the shortest-path distance

# Trajectories	525,009	Mean	1.604
First quartile	1.04	Median	1.31
Third quartile	1.79	Standard deviation	5.52
Variance	30.456	Standard error	0.0176

The query above retrieves the start and end times of every queuing event during each trip. The function `speed` returns the speed of the object at every timestamp as a `tfloat` type. The value of the temporal float can then be compared against a specific float range, using the `atRange` function. This function returns the portion of the trip that is within that range. That is, if the speed value is within the specified range (i.e., less than 15 km/h in our case), it means that the car might not be moving at this timestamp, and thus the waiting interval is computed using the timespan function in the `WHERE` clause. If there are multiple stops on different intervals, this function only computes the sum of the waiting intervals ignoring the gaps between them. If the waiting interval is between 5 and 30 min, then it can be said that the street is crowded. The `gettime` function in the `SELECT` clause returns a period set type that represents the queuing time period for each stop. For example, below we can see the output for a given trip.

```
Trip: [POINT(669846 828032)@2016-04-01 10:37:26, POINT(669804 828026)@2016-04-01 10:37:57, ...]
timespan: 00:20:38
gettime:{[2016-04-01 10:37:26, 2016-04-01 10:40:56), [2016-04-01 10:41:01, 2016-04-01 10:41:06)}
```

In this study, the value of the indicator is 61% as an average for all trips.

5.2.2 | Travel time between landmarks

This indicator aims at reflecting the users' experience with the traffic conditions by considering the travel times between certain landmarks (e.g., from a university campus to a train station). The quality of this indicator of course depends on an appropriate selection of landmarks. In the case study presented in this work, we extracted 26 landmarks from the Bey2ollak mobile application, which is the source of the data set as mentioned in Section 1, and contains the main landmarks in Cairo and Alexandria. Every neighborhood is represented as a spatial point, namely its center.

To compute the indicator, every trip is annotated with the closest landmarks to its source and to its destination. If no landmark is found within 5 km of either points or if the source and destination landmarks are the same, the trip is discarded. Then the travel time is averaged per different source and destination landmark pair. There are 650 such possible pairs and not all of them had enough traversing trajectories in the data set. A certain number of trajectories must exist between a pair of landmarks to consider the aggregated average as significant. Here, this threshold is set to 100. The number of landmark pairs that fulfilled this condition is 158. The SQL query in MobilityDB is given next.

TABLE 3 Traveling distance deviation from the straight-line distance

# Trajectories	579,120	Mean	2.45
First quartile	1.5	Median	1.954
Third quartile	2.792	Standard deviation	1.59
Variance	2.529	Standard error	0.004978

```
WITH landMarkTrips AS(
    SELECT tripId, transId, duration(trip) AS tripDuration,
        (SELECT name
        FROM EgyptLandMark
        WHERE ST_DWithin(startValue(t.trip), geom, 5000)
        ORDER BY startValue(t.trip) <-> geom ASC
        LIMIT 1) AS initialLandMark,
        (SELECT name
        FROM EgyptLandMark
        WHERE ST_DWITHIN(endValue(t.trip), geom, 5000)
        ORDER BY endValue(t.trip) <-> geom ASC
        LIMIT 1) AS finalLandMark
    FROM trips t
)
SELECT initialLandMark, finalLandMark, COUNT(*), AVG(tripDuration)
FROM landMarkTrips
WHERE initialLandMark IS NOT NULL AND finalLandMark IS NOT NULL AND initialLandMark <> finalLandMark
GROUP BY initialLandMark, finalLandMark
HAVING COUNT(*) >= 100;
```

In this query, the `landMarkTrips` CTE aggregates the average trip duration for all pairs of landmarks that have at least 100 traversing trips. The trip duration is calculated simply using the `duration` function that returns the total trip duration, including the waiting time intervals. For each trip, the initial landmark is obtained by performing a spatial K-nearest neighbor query, that returns the top most closest landmark to the starting point of the trip. An analogous process is performed for the end point of the trip, to return the final landmark.

5.2.3 | Deviation from free flow

The free flow speed for a street segment represents the speed of cars under good conditions, such as no congestion, good weather, no accidents, and no temporary obstacles. There are many ways of estimating the free flow speed on a given street (Fazio et al., 2014; Krogh et al., 2012). Some of them add 5–10 km/h to the regulatory speed limit as an estimation. Other proposals conduct field surveys, take actual measurements, and fit them to some statistical distributions. The estimation and method must adapt to different places. The method used here for estimating the free flow is briefly explained next. We first sort the observed speed values per road segment. Then, the highest 10% values are ignored and the next value is chosen to represent the free flow speed. The ignored 10% accounts for errors and outliers. The MobilityDB query is given next.

```
WITH segment_speed AS (
    SELECT gid, twavg(speed(atGeometry(trip, geom))) AS speed_km
    FROM trips T1, bfmap_ways T2
    WHERE Intersects(T1.trip, T2.geom)
        AND length(atGeometry(T1.trip, T2.geom)) > 5
), calc_freeflowspeed_km_h AS (
    SELECT gid, percentile_disc(0.1) WITHIN GROUP (ORDER BY speed_km DESC) AS freeflowspeed_km_h
    FROM segment_speed

    GROUP BY gid
)
SELECT tripId, transId, timespan(trip) AS tripTime, (length(trip) * 3600)/freeflowspeed_km_h
    AS durationFreeflowMillisec
FROM trips t1, bfmap_ways t2, calc_freeflowspeed_km_h t3
WHERE Intersects(t1.trip, t2.geom)
    AND t2.gid = t3.gid
    AND t3.freeflowspeed_km_h <> 0;
```

The `freeFlowSpeed` CTE computes, for every road segment its estimated free flow speed, which is then used in the main query to compute the travel time under the free flow condition. In our case, the global aggregate of all trips results in the fact that the travel time is on average 375% higher than in a free flow condition. This reflects a high level of congestion, which is typical of Cairo. More detailed statistics are reported in Table 4.

5.2.4 | Rate of gearing

The last indicator to be studied is the *Rate of gearing*, which measures the number of changes of gear per kilometer. A gearing event occurs when a driver has to change the gear. It denotes an obstacle, a congestion, or other kinds of distraction that interrupt the driving and require a driver response. A high number of such events can be tiring for the driver. This is captured by the indicator proposed here, that is defined as the number of gearing events per trip kilometer. A high rate of gearing indicates that the network has many distractions. As shown next, this indicator is computed by analyzing the speed curve of the vehicle. Note that, of course, this indicator is less relevant for automatic transmission. Since this information is not part of the GPS data used for the analysis, we assume that both manual and automatic vehicles have similar speed curves and, thus, the indicator value is not biased by the inclusion of automatic transmission vehicles in the analysis. We consider this assumption realistic, since all vehicles are equally affected by the traffic and road conditions.

The gearing events in a trajectory are counted by analyzing its speed curve. Some works aim at establishing a relationship between the gear choice and the vehicle speed (Eckert et al., 2014; Ericsson, 2000; Ngo, 2012). For instance, Ngo (2012) identify gear shifting profiles using the different driving styles that consider fuel economy, maximum power, and optimized performance. Those works reveal that there is a speed band for every gear shift. The MobilityDB query is given next.

```
WITH GearingEvents AS (
    SELECT 2 As GearType, floatrange'[0, 10]' AS speedRange UNION
    SELECT 3 As GearType, floatrange'[11, 30]' AS speedRange UNION
    SELECT 4 As GearType, floatrange'[31, 50]' AS speedRange UNION
    SELECT 5 As GearType, floatrange'[51, 80]' AS speedRange
)
SELECT t1.tripId, t1.transId, t2.GearType, t2.speedRange, timespan(atRange(speed(trip), t2.speedRange)) AS
        gearPeriod
FROM trips t1, gearingEvents t2
WHERE atRange(speed(t1.trip), t2.speedRange) IS NOT NULL
```

This query consists of a CTE that defines the gearing events and applies a join between them and every trip. The join operation is based on the intersection between any part of the trip speed, and any speed range of the gearing events. That is, if part of the trip speed is within the speed range, then the trip duration during that range is computed.

The indicator is computed by dividing the total number of the gearing events, by the total distance traveled by all vehicles. For the data set considered in the present work, the indicator's value is of 1.58 gears per km. Figure 6 shows the average number of gearing events per kilometer per day, for every gear band and per hour (except for periods between 0:00–7:00 and 19:00–24:00, which are considered as one period). For each day period, the gearing events whose timestamps fall in the period are counted and the total traveled distance of all vehicles is aggregated. The figure shows the result of dividing these two components, for every gear. The choice of the day periods is based on the analysis done in the Cairo Traffic Congestion Study, carried out by the World Bank in 2013 (Nakat, 2013). The figure shows that the rate is almost the same over the whole hour, with a mild increase during both the morning and the afternoon peaks. It also shows that the third gear is the most frequent one.

5.3 | Visualizing the results

In this section, we briefly show how these indicators (and any MobilityDB query) can be visualized using typical GIS software, to help the analyst's work easier. We also show how of MobilityDB functions can be used to produce ad hoc indicators that modify or extend the ones presented in the two sections above.

As an example, consider the analysis of the speed in a road network, which is part of many road performance indicators. Figure 7 shows the speed map using QGIS (https://qgis.org/). A gradient color is assigned to each edge, ranging from blue to red. This color represents the speed of the trips that traverse the edges of the network. Since the maximum speed of edges ranges between 20 and 120 Km/h, it is interesting to compare the speed of the trips at an edge, with respect to the maximum speed for that edge. The following query implements such comparison.

```
WITH edgeSpeed AS (
    SELECT P.gid, AVG(twavg(speed(atGeometry(t.trip, ST_Buffer(p.geom, 0.1))))*3.6) AS twavg
    FROM trips t, OSMRoadNetwork p
    WHERE Intersects(t.trip, p.geom)
    GROUP BY P.gid

    ORDER BY P.gid
)
SELECT e.gid, maxspeed_forward AS maxspeed, geom, avg, avg / maxspeed_forward AS perc, count
FROM OSMRoadNetwork e, edgeSpeed t
WHERE e.gid = t.gid;
```

In this query, a CTE query restricts the trip to the geometry of the edge and computes the time-weighted average of the speed. After that, the average for each edge is used to produce the speed map. Note that the `ST_Buffer` function is used to cope with the floating point precision required.

6 | PERFORMANCE EVALUATION

In this section, the implementation and performance of the proposed indicators in MobilityDB are evaluated on both a single- and a multi-node cluster. The multi-node cluster setup consists of a coordinator node and three other worker nodes, all of them running MS Citus, an open-source extension to PostgreSQL that turns the latter into a distributed database

TABLE 4 Trip time deviation from the free flow condition

# Trajectories	579,120	Mean	3.75
First quartile	2.05	Median	2.77
Third quartile	4.216	Standard deviation	2.87
Variance	8.25	Standard error	0.0076

(https://www.citusdata.com/). Citus distributes data and queries across nodes in a cluster of commodity machines, horizontally scaling PostgreSQL using sharding and replication. Its query engine parallelizes incoming SQL queries across these servers to enable real-time responses on large data sets. The latest version of Citus also allows columnar storage, which makes Citus appropriate for analytical queries. All nodes in the cluster have the same stack of software: PostgreSQL 13, PostGIS 2.5, PgRouting 3.1.1, MobilityDB 1.0 beta3, and MS Citus 10 running on Ubuntu 18.04. Each node is equipped with 6 cores per socket (2 sockets and 2 threads per core), Intel i7-7800X @ 3.50 GHz CPU, 1 TB SSD, and 24 GB of RAM. The `trips` table (i.e., the table containing the continuous trajectories) is distributed using the hash partitioning method provided by Citus into 30 partitions. These partitions are stored in the worker nodes, where each node contains 10 partitions. To support the local optimization of the queries, we build spatiotemporal indexes on the `trajectory` attribute in each partition. In MobilityDB, indexes are defined as extensions to the generalized search tree GiST and the space partitioning search tree SP-GiST. Both indexes are used to improve the performance of the spatiotemporal predicates such as `Intersects`. The use of Citus is aimed at devising an efficient query plan for distributing the workload over the cluster partitions.

Table 5 compares the execution times of the indicators over the cluster versus a single-machine implementation. It can be seen that, as expected, the distributed computation of the indicators outperforms the single-node computation since queries are executed on load balanced small partitions. Each query has been executed five times and the average execution time is reported. All queries are of the *broadcast join type*, which means that one table (e.g., `trips`) is distributed, while the other ones are replicated in each node (e.g., the table representing the road network). Replicating the lookup tables allows to distribute the join computations over workers so that the joins are evaluated locally in each partition. Therefore, all queries fall into the `pushdownable` class of MS Citus, which distributes the MobilityDB query into all partitions and aggregates the results on the coordinator.

Although it may appear obvious that a clustered architecture would deliver better performance that a single node one, the performance gain due to distribution varies depending on different factors such as the amount of data transfer over the network and the amount of work done on the coordinator node. For example, for the indicator in Section 5.1.1 (deviation from shortest path), finding the source and destination of each trip is done on the worker nodes, while loading the network into pgRouting and calculating the shortest path distance between every source and destination are done on the coordinator node. Computation at the coordinator node takes most of the response time. Because of this, this indicator does not benefit much from a distributed evaluation. For the other indicators, most of the query evaluation is distributed over the workers. Therefore, their performance on the cluster is up to one order of magnitude faster than on the single node. This is, for example, the case of the computation of the deviation from free flow indicator (Section 5.2.3). This query is very expensive because it must compute the join between the trip trajectories and the segments of the full road network. This query is optimized using the GiST index although the join cardinality remains huge. The second factor impacting this query is that it uses expensive functions such as `atGeometry`. In conclusion, distributed computation of the performance indicators (in this case, using the distributed version of MobilityDB) is crucial given the usual sized of the mobility data sets.

7 | CONCLUSIONS

This article introduces the idea of user-centered road network indicators, that is, indicators that account for the user experience and perspective, opposite to the most usual ones that normally just consider the road and traffic

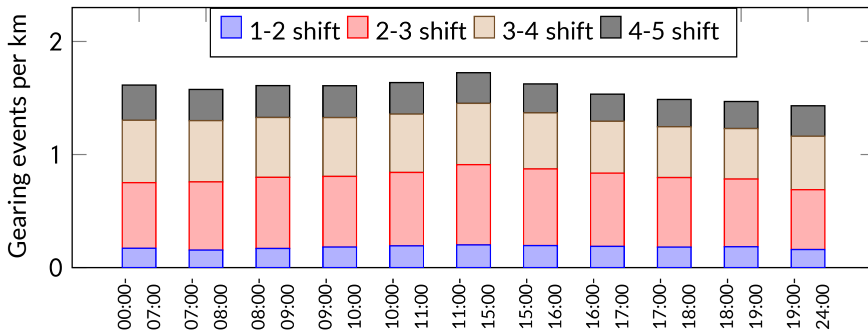

FIGURE 6 Count of each gear per traveled km per day period.

conditions, regardless of how the users feel about their driving experience. The work classifies the indicators into two broad classes: road network and traffic indicators, defines them precisely, and shows how that they can be efficiently implemented even for the big mobility data sets. The indicators account for three key aspects of user experience in a road network: driving comfort, flow convenience, and network convenience to motorists. Furthermore, the indicators are computed using MobilityDB, a moving object database that extends the well-known PostgreSQL database with spatiotemporal data types and functions.

The whole process of acquiring data, transforming, cleaning, and map matching the data to the road network, computing, and visualizing the results is also explained and discussed using a real-world large data set containing data of Cairo traffic in Egypt. In addition, the data set size allows us to show how the indicators are computed in a distributed environment. Running times in centralized and distributed settings are compared, and the effect of distribution is reported and discussed, showing how it dramatically reduces execution times. The SQL and MobilityDB expressions that compute all indicators are included and sample visualizations of the results are also shown.

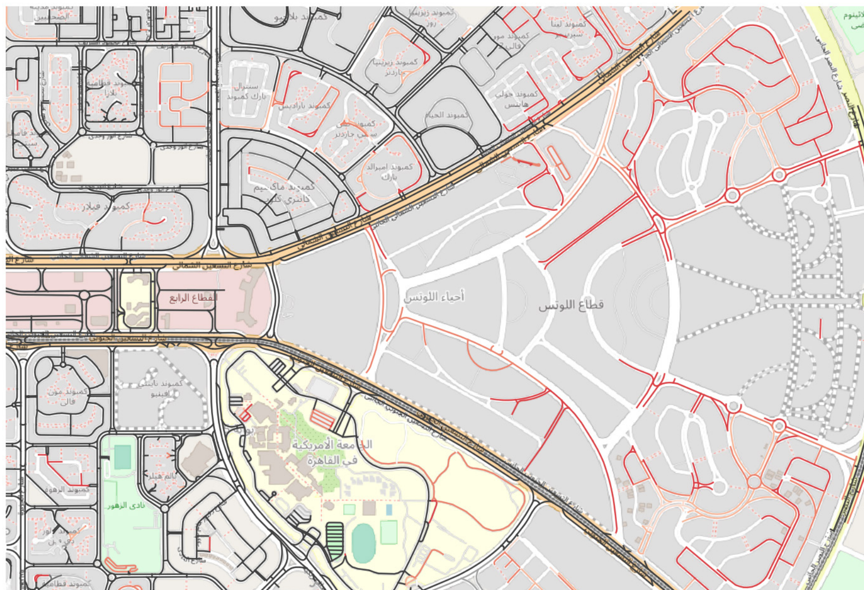

FIGURE 7 Visualization of the network edges according to the number of trips that traversed the edges.

TABLE 5 Execution times for the performance indicators on single-node and clustered machines

Indicator	Single node	Multi node
(1) Network indicator: Deviation from straight-line path	00:45:17.63	00:20:23.66
(2) Network indicator: Deviation from shortest distance path	00:00:03.08	00:00:00.14
(3) Road indicator: Queuing time	00:13:08.69	00:02:06.14
(4) Road indicators: Travel time between landmarks	00:28:41.79	00:05:44.31
(5) Road indicators: Deviation from free flow	12:48:33.15	01:23:47.29
(6) Driving comfort indicator: Rate of gearing	00:19:39.03	00:03:09.58

The article also reviews the many road performance indicators and classifications currently available and shows that they do not consider the user's experience or do this just to a very limited extent. We believe that accounting for user's experience in a road performance analysis cannot be avoided in modern times. Furthermore, given the amount of data available nowadays, defining indicators and computing them efficiently as shown in this article, can provide the analysts important tools for their work and is more practical that carrying out costly and limited surveys (as proposed in some works discussed here) that also can take a long time to prepare and complete. We showed how current technology allows us to efficiently process large mobility data sets.

As future work, new indicators can be developed to address different situations and audiences. Furthermore, since the study presented in this work averages the indicators over all vehicle profiles (e.g., personal cars, taxi, etc.), over all trip purposes (e.g., work, leisure, etc.), and over all driver profiles, as future work it would be interesting to segment the trajectories and compute fine-grained indicators. In another direction, the work opens the possibility of defining indicators for smaller extents or even individual road segments, to assess the effect of road works on the driver's experience.

FUNDING INFORMATION

Argentinian Scientific Agency, Project: PICT 2017–1054; Innoviris, the Brussels Institute for Research and Innovation, Project: MobiWave 2019–2021.

CONFLICT OF INTEREST

No potential conflicts of interest are reported by the authors.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from Bey2ollak. Restrictions apply to the availability of these data, which were used under license for this study. Data are available from https://desktop.bey2ollak.com/ with the permission of Bey2ollak.

ORCID

Alejandro Vaisman https://orcid.org/0000-0002-3945-4187

REFERENCES

AECOM. (2015). *Key performance indicators for intelligent transport systems* (Study final report). https://ec.europa.eu/transport/sites/transport/files/themes/its/studies/doc/its-kpi-final_report_v7_4.pdf

Almeida, A. M. R., Lima, M. I. V., de Macêdo, J. A. F., & Machado, J. C. (2016). DMM: A distributed map-matching algorithm using the mapreduce paradigm. *19th IEEE International Conference on Intelligent Transportation Systems*, Rio de Janeiro (pp. 1706–1711). IEEE.

Bakli, M. S., Sakr, M. A., & Soliman, T. H. A. (2018). A spatiotemporal algebra in Hadoop for moving objects. *Geo-Spatial Information Science*, *21*(2), 102–114. https://doi.org/10.1080/10095020.2017.1413798

Bakli, M., Sakr, M., & Zimányi, E. (2020). Distributed mobility data management in MobilityDB. *21st IEEE International Conference on Mobile Data Management*, Versailles, France (pp. 238–239). IEEE.

Bechtel, A., Brennan, T., Gurski, K., & Ansley, J. (2018). Using anonymous probe-vehicle data for a performance indicator of bridge service. *Infrastructure Asset Management*, *5*, 1–42. https://doi.org/10.1680/jinam.17.00015

Dick, J., Hull, M. E. C., & Jackson, K. (2017). *Requirements engineering* (4th ed.). Springer.

Eckert, J., Santiciolli, F., dos Santos Costa, E., Corrêa, F., José Dionísio, H., & Giuseppe Dedini, F. (2014). Vehicle gear shifting c-simulation to optimize performance and fuel consumption in the Brazilian standard urban driving cycle. *XXII Simpósio Internacional de Engenharia Automotiva*, São Paulo (pp. 615–631). Brazilian Association of Automotive Engineering.

Ericsson, E. (2000). *Driving pattern in urban areas: Descriptive analysis and initial prediction model*. Traffic Planning, Bulletin 185. Lund Institute of Technology.

Fazio, J., Wiesner, B. N., & Deardoff, M. D. (2014). Estimation of free-flow speed. *KSCE Journal of Civil Engineering*, *18*, 646–650. https://doi.org/10.1007/s12205-014-0481-7

Fontaine, D. (2014). *Pgloader*. https://github.com/dimitri/pgloader

Francia, M., Gallinucci, E., & Vitali, F. (2019). Map-matching on big data: A distributed and efficient algorithm with a hidden markov model. *42nd International Convention on Information and Communication Technology, Electronics and Microelectronics*, Opatija, Croatia (pp. 1238–1243). IEEE.

Fu, Z., Tian, Z., Xu, Y., & Qiao, C. (2016). A two-step clustering approach to extract locations from individual gps trajectory data. *ISPRS International Journal of Geo-Information*, *5*, 166. https://doi.org/10.3390/ijgi5100166

GmbH, B. C. I. (2015). *Barefoot*. https://github.com/bmwcarit/barefoot

Güting, R. H., & Schneider, M. (2005). *Moving objects databases*. Morgan Kaufmann.

Hohmann, S., & Geistefeldt, J. (2016). Traffic flow quality from the user's perspective. *Transportation Research Procedia*, *15*, 721–731. https://doi.org/10.1016/j.trpro.2016.06.060

Krogh, B., Andersen, O., & Torp, K. (2012). Trajectories for novel and detailed traffic information. *3rd ACM SIGSPATIAL International Workshop on GeoStreaming*, Redondo Beach, CA, USA (pp. 32–39).

Meng, C., Yi, X., Su, L., Gao, J., & Zheng, Y. (2017). City-wide traffic volume inference with loop detector data and taxi trajectories. *25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Redondo Beach, CA, USA (pp. 1-10). ACM. https://doi.org/10.1145/3139958.3139984

Nakat, Z. (2013). *Cairo traffic congestion study* (Study final report). World Bank. http://hdl.handle.net/10986/18735

Newson, P., & Krumm, J. (2009). Hidden markov map matching through noise and sparseness. In *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Seattle, WA, USA (pp. 336–343). ACM.

Ngo, D. (2012). *Gear shift strategies for automotive transmissions*. PhD dissertation. Department of Mechanical Engineering, Technische Universiteit Eindhoven.

Parent, C., Spaccapietra, S., Renso, C., Andrienko, G., Andrienko, N., Bogorny, V., Damiani, M. L., Gkoulalas-Divanis, A., Macedo, J., Pelekis, N., Theodoridis, Y., & Yan, Z. (2013). Semantic trajectories modeling and analysis. *ACM Computer Surveys*, *45*, 42. https://doi.org/10.1145/2501654.2501656

Peixoto, D. A., Hung, N. Q. V., Zheng, B., & Zhou, X. (2019). A framework for parallel map-matching at scale using spark. *Distributed Parallel Databases*, *37*, 697–720. https://doi.org/10.1007/s10619-018-7254-0

pgRouting Community. (2013). *pgrouting project - open source routing library*. http://pgrouting.org/

Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., & Snodgrass, R. (1993). *Temporal databases: Theory, design, and implementation*. Benjamin-Cummings.

TII (Transportation Infrastructure Ireland). (2016). *National road network indicators*. http://www.til.ie/tii-library/strategic-planning/

TomTom. (2004). *Tomtom traffic index*. https://www.tomtom.com/en_gb/trafficindex/

Vanhove, F., & De Ceuster, G. (2003). *Traffic indices for the use of the belgian motorway network*. http://www.tmleuven.be/project/verkeersindices/200301_paper.pdf

VIAS Institute. (2017). *Belgian key inidcators road safety*. http://www.vias.be/en/research/notre-publications/

Wei, H., Wang, Y., Forman, G., & Zhu, Y. (2013). Map matching: Comparison of approaches using sparse and noisy data. *21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Orlando, FL, USA (pp. 444–447). ACM.

World Bank. (2013). *Cairo traffic congestion study* (Final report). Author.

Yan, Z., Chakraborty, D., Parent, C., Spaccapietra, S., & Aberer, K. (2013). Semantic trajectories: Mobility data computation and annotation. *ACM Transactions of Intelligent Systems and Technology*, *4*, 49–49. https://doi.org/10.1145/2483669.2483682

Zimányi, E. (2020). *Mobilitydb*. https://docs.mobilitydb.com/MobilityDB/master/mobilitydb.pdf

Zimányi, E., Sakr, M., & Lesuisse, A. (2020). MobilityDB: A mobility database based on PostgreSQL and PostGIS. *ACM Transactions on Database Systems*, *45*, 19:1–19:42. https://doi.org/10.1145/3406534

Zimányi, E., Sakr, M., Lesuisse, A., & Bakli, M. (2019). Mobilitydb: A mainstream moving object database system. In *16th International Symposium on Spatial and Temporal Databases*, Vienna, Austria (pp. 206–209). ACM.