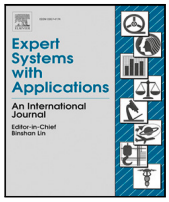
Expert Systems With Applications 238 (2024) 121949

Contents lists available at ScienceDirect

Expert Systems With Applications

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



A visual-based toolkit to support mobility data analytics

Sergio Di Martino [a], Enrico Landolfi [b], Nicola Mazzocca [a], Franca Rocco di Torrepadula [a], Luigi Libero Lucio Starace [a,*]

[a] Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Naples, Italy
[b] NetCom Engineering S.p.A., Via Nuova Poggioreale, 80143, Naples, Italy

ARTICLE INFO

Keywords:
Knowledge discovery from data
Intelligent transportation systems
Data analytics
Intelligent systems development

ABSTRACT

The Knowledge Discovery from Data (KDD) process is widely used across various domains to get valuable insights from data. Many platforms, like KNIME or RapidMiner, offer effective tools for KDD analysts, allowing them to perform data analytics tasks in a visual fashion, without writing code. In recent years, the increasing availability of mobility data has led to a surge in KDD-based initiatives from both industry and academia in the Intelligent Transportation Systems (ITS) domain. Still, KDD platforms lack comprehensive support for some typical mobility data manipulation tasks. As a result, mobility data analysis still requires a significant coding phase, with reduced productivity and hindered replicability of results.

To address this gap, this paper presents a novel solution aimed at supporting ITS data analysts in defining KDD processes more efficiently. More in detail, we extended the KNIME platform by introducing a collection of new components explicitly tailored to facilitate some peculiar KDD tasks from mobility data. These components encompass critical functionalities such as map coverage analysis, trajectory partitioning and map-matching.

To showcase the effectiveness of the proposed solution, we used it to replicate a study published in the ITS data analytics domain. Thanks to our proposal, such replication can be accomplished in a few minutes and with just a few clicks, without any manual coding, resulting in a pipeline that is easier to understand, distribute and re-execute, also for domain experts with no programming experience.

Our solution is open-source and freely downloadable from the Knime Hub. In this way, we aim to foster data-driven research and practice in the ITS field, by providing researchers and practitioners with more effective analytics tools to handle mobility data.

1. Introduction

In the last years, the wide availability of real-world mobility datasets has triggered a paradigm shift in the Intelligent Transportation Systems (ITS) domain, with data-driven approaches gaining more and more relevance, and being often used to complement or even replace traditional model-driven solutions (Veres & Moussa, 2019; Zhang et al., 2011). Typically, these data-driven ITS solutions leverage Data Analysis and Artificial Intelligence (AI) techniques for discovering patterns and trends from data, enabling a number of use cases, such as better characterizations of transportation demand or of mobility dynamics predictions (Bock, Di Martino, & Origlia, 2020; Lee, Eo, Jung, Yoon, & Rhee, 2021), better surveillance of urban scenarios (Lee et al., 2006), public transport crowding estimation (Jenelius, 2019; Zhang et al., 2017), and so on.

The implementation of such data-driven ITS solutions requires the definition of appropriate data analysis/processing pipelines, often conceptually based on the process of *Knowledge Discovery from Data* (KDD) (Fayyad, Piatetsky-Shapiro, & Smyth, 1996). KDD consists of a pipeline of five key phases, namely *Data Selection*, *Preprocessing*, *Transformation*, *Data Mining* and *Interpretation/Evaluation*. Each of these phases includes one or more tasks, intended as distinct functional blocks. To set up a KDD pipeline, data analysts need to implement said functional blocks and properly orchestrate them towards the analysis goals.

In scenarios that do not need to deal with spatio-temporal mobility data, several general-purpose data analytics platforms, e.g.: KNIME[1] or RapidMiner,[2] have been proposed and are largely used. Thanks to their *visual* programming paradigm (see Fig. 4), these platforms allow

* Correspondence to: Università degli Studi di Napoli Federico II, Via Claudio, 80125, Naples, Italy.
E-mail addresses: sergio.dimartino@unina.it (S. Di Martino), e.landolfi@netcomgroup.eu (E. Landolfi), nicola.mazzocca@unina.it (N. Mazzocca), franca.roccoditorrepadula@unina.it (F. Rocco di Torrepadula), luigiliberolucio.starace@unina.it (L.L.L. Starace).

[1] https://www.knime.com/
[2] https://rapidminer.com/

https://doi.org/10.1016/j.eswa.2023.121949
Received 24 November 2022; Received in revised form 29 September 2023; Accepted 29 September 2023
Available online 9 October 2023

data analysts to set up KDD pipelines without writing a single line of code, but rather by simply connecting the appropriate functional blocks that implement the tasks needed for the specific problem at hand. Indeed, these platforms feature a number of built-in functional blocks implementing common KDD tasks, such as reading data from files or databases, filtering data based on conditions, visualizing data, applying AI techniques, and so on. This has the key advantage of empowering domain experts and/or decision makers to set up their own data analytics solution at a high abstraction level, focusing more on the specific problem at hand and less on implementation details. Unfortunately, in our industrial and academic experience, we found that these general-purpose visual platforms present some limitations in supporting analyses of mobility data, as they lack functional blocks to carry out some peculiar tasks required by the spatio-temporal nature of this kind of data.

As a first step to overcome these challenges and allow ITS researchers/ practitioners to fully exploit the power of general-purpose analytics tools, in this work, we introduce a visual-based toolkit we developed, intended as an extension of the well-known KNIME Analytics Platform. Our toolkit enriches KNIME with a set of novel spatio-temporal data processing components designed to carry out common mobility data tasks such as *map matching*, *map coverage analysis*, *trajectory partitioning* and *restoration*. These components we developed are designed to be potentially further extended, for example with ad-hoc heuristics, and tailored to different needs. The extension is freely available on the official *KNIME Hub*[3] repository, and the source code can be found on GitHub (Di Martino, Principe, & Starace, 2021). We believe that the free availability of the extension can streamline the process of implementing KDD pipelines for conducting investigations based on mobility data within the ITS community.

Moreover, since KNIME pipelines can be easily exported and re-executed on any platform on which the KNIME platform can be installed, without the need to deal with dependencies and environment settings, our extension could also foster ITS research by facilitating the distribution of replicable research results. Lastly, we show how the proposed extension can empower ITS researchers by replicating, with just a few clicks and no need to write any code at all, a recently published investigation (Cruz, Couto, Costa, Fladenmuller, & de Amorim, 2020) in the ITS field about involving Bus Service Coverage Analysis in Rio De Janeiro (Brazil). A replication package with detailed instructions is available at Di Martino, Landolfi, Mazzocca, Rocco di Torrepadula, and Starace (2022).

Summarizing, this work makes the following contributions:

- We discuss and formalize, from a Software Engineering perspective, the development process of KDD pipelines. We thoroughly examine the peculiar challenges that ITS professionals encounter during the development process, aiming to provide a structured framework for effectively overcoming these obstacles.
- We present and describe in detail the visual-based toolkit we propose to support the definition of replicable KDD pipelines in the ITS domain, covering the key steps of data-driven ITS analytics.
- We demonstrate how the proposed toolkit can empower ITS researchers, showcasing how it enables the replication of a published investigation in the field, with just a few clicks and no need to write any code at all.
- We make our solution, as well as all means necessary to further extend it, freely available to any interested researcher and practitioner, to foster collaboration and promote further advancements in the field.

[3] https://hub.knime.com/, currently in the KNIME Community Extensions (Experimental) branch

The paper is structured as follows. In Section 2, we give an overview of the standard KDD process, the peculiar challenges arising from its application in the ITS domain, and some existing related solutions. Then, in Section 3, we start by providing a detailed overview of the KNIME tool and subsequently present the visual toolkit we devised, focusing on each of its components. Further details on the components we developed, including a formalization of their inputs and outputs and a detailed description of the available configuration options, are reported in Appendix A. In Section 4, we show how the proposed solution can be easily integrated within the KNIME ecosystem and used to design effective KDD pipelines on mobility data. We do so by partially replicating a recently published investigation in the ITS field (Cruz et al., 2020), performing a Bus Service Coverage Analysis leveraging a public dataset. Final remarks and future research directions conclude the paper, in Section 5.

2. Knowledge discovery from mobility data: Preliminaries, related works and challenges

In the last years, many works in the literature have focused on extracting knowledge from mobility data, typically with the goal of gaining new insights on spatio-temporal phenomena of interest, also enabling more efficient, safer, and profitable ITS solutions (Di Martino & Starace, 2022a; Khan, Rahman, Apon, & Chowdhury, 2017; Zheng, Wu, Chen, Qu, & Ni, 2016; Zhu, Yu, Wang, Ning, & Tang, 2018). For example, Nguyen et al. (2018) analyzed a dataset collected from buses in Los Angeles, to assess the performance of the public transit system. In Nuzzolo, Comi, Papa, and Polimeni (2018), data collected from taxis operating in Rome, Italy, were analyzed to extract knowledge on existing mobility demand patterns. Also Asprone, Di Martino, Festa, and Starace (2021) and Di Martino and Starace (2022b) analyzed taxi trajectories, with the goal of evaluating the practicality of leveraging a fleet of taxis to crowd-sense knowledge in urban environments, as an enabler of novel use cases for Smart Cities. It is worth noting that mobility data are not necessarily collected from vehicles. As an example, Di Lorenzo et al. (2015) propose a solution to visually analyze mobility patterns extracted from personal smartphone positioning data, with the goal of optimizing public transport.

All these cited works had to deal, to different extents depending on the specific applications, with some common tasks and challenges of KDD applied to mobility data. In the following, we first describe the KDD approach. Then, based on our industrial and academic experience in this field, we introduce some peculiar tasks required by KDD applied to mobility/FCD data, together with related works and open challenges.

2.1. The process of knowledge discovery from data

Extracting new knowledge from relevant amounts of data is an old and key multidisciplinary problem, which has been extensively tackled in the domain of Data Science (Van Der Aalst, 2016). More in detail, the process of Knowledge Discovery from Data (KDD) was first formalized in Fayyad et al. (1996) as the application "*of methods and techniques for making sense of data*". In that work, the authors identified five key steps (see Fig. 1) composing the KDD process:

1. *Data Selection*, consisting in selecting a data set, or a subset thereof, on which to apply the knowledge discovery process.
2. *Preprocessing*, dealing with cleansing and preprocessing the selected data, for example by identifying suitable strategies to handle noisy or missing data, to normalize measurements, to fit domain requirements, and so on.
3. *Transformation*, involving data reduction/projection to transform it into a more useful representation, for feeding the subsequent Data Mining algorithms, depending on the goal of the task.
4. *Data Mining*, which is concerned with discovering correlations or patterns in the investigated data, typically leveraging Machine Learning techniques.

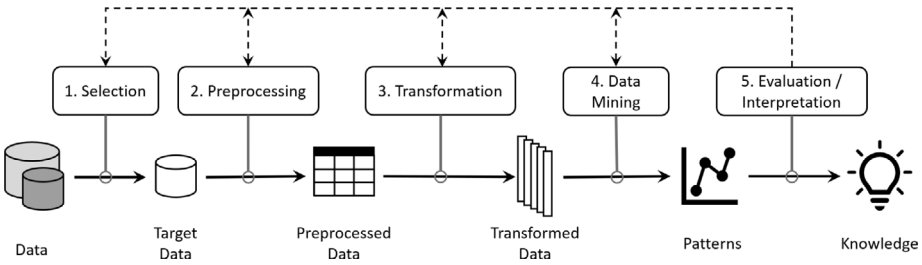


Fig. 1. An overview of the KDD process, based on Fayyad et al. (1996).

5. *Interpretation/Evaluation*, is the stage where correlations and patterns identified in the previous step are presented to a decision-maker for interpretation, often with the support of data visualization techniques.

Notice that the process of KDD is not necessarily linear. Indeed, in most cases, it can involve significant iterations, with potential loops between any two steps (Fayyad et al., 1996), as highlighted by the dashed lines in Fig. 1.

2.2. Peculiarities of knowledge discovery from mobility data

Applying a KDD process to mobility data often necessitates specific preprocessing steps due to the unique nature and characteristics of collected spatio-temporal information. In the following, based on our industrial and academic experience in this field, we briefly introduce some peculiar tasks required by KDD applied to mobility data. Firstly, mobility datasets are typically composed of massive sets of spatio-temporal structured data, typically including a GPS position, a timestamp, and possibly additional information of interest (e.g.: speed, direction, recorded temperature, etc.). Such data is often, but not necessarily, collected from vehicles, in which case it is given the name of Floating Car Data (FCD) (Zhang et al., 2011; Zhu et al., 2018). In the following, for the sake of simplicity, we refer to all spatio-temporal mobility data consisting of timestamped GPS positions enriched with additional information as FCD, but all the considerations also stand for similar data collected from other entities (e.g.: pedestrians with smartphones).

Typically, these Floating Car Data (FCD) datasets contain a stream of data collected from numerous vehicles over extended time periods. Examples of such datasets include those presented by Bracciale et al. (2014) or by Piorkowski, Sarafijanovic-Djukic, and Grossglauser (2009). In these cases, to enable analysis steps in the KDD data mining phase, it is necessary to split the data stream according to some filtering criteria (e.g. by vehicle, by trip, etc.). Thus, **Trajectory Partitioning** is usually the first task to be performed in the preprocessing phase, aimed at splitting such data stream into a set of independent trajectories, each representing a vehicle route from a given origin to a destination. To better understand these concepts, let us consider the example in Fig. 2. Fig. 2(a) depicts raw FCD from a real-world dataset of taxi trajectories (Piorkowski et al., 2009) collected in the City of San Francisco, USA. The dark dots correspond to the GPS positions of 10 taxis over a three-week timespan. In this scenario, the goal of Trajectory Partitioning is to divide the raw positioning data stream into several subsets, each containing only data for a specific trip. A possible output of Trajectory Partitioning on these data is reported in Fig. 2(b), where points are colored based on their belonging to the same trip. However, as highlighted in Fig. 2(b), simply connecting subsequent GPS positions is typically not sufficient to accurately reconstruct the actual trajectory of a vehicle, which is usually the basic information for the KDD process. Thus, a subsequent task, typically performed in the preprocessing phase, and crucial when there is a need to contextualize raw mobility data on real-world maps, is **Map Matching**. The aim is to align possibly inaccurate positioning data to an underlying logical representation of

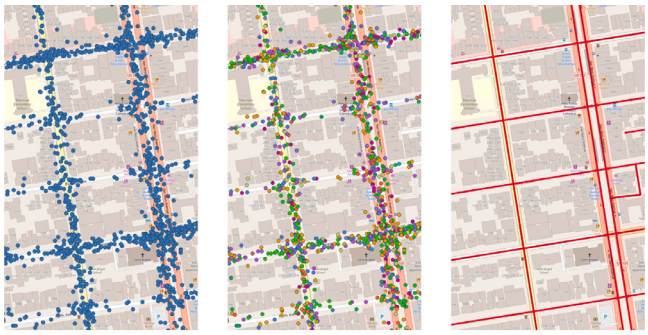


Fig. 2. Example of challenges for KDD on FCD. From left to right: (a) Raw GPS positions collected from taxis; (b) Positions after partitioning; (c) Map-matched trajectories.

the road network (Kubicka, Cela, Mounier, & Niculescu, 2018). The effort to perform map-matching heavily depends on the quality of the spatio-temporal data: the lower the sampling rates and the position accuracy, the harder the task is. To give an insight into the variability of the data quality of publicly available vehicular trajectory datasets, e.g. Bracciale et al. (2014), Piorkowski et al. (2009) and Yuan et al. (2010), the sampling rates in FCD datasets can vary from a few seconds (e.g.; 7 s on average in the Rome, Italy taxi dataset (Bracciale et al., 2014)), to a few minutes (e.g.: 177 s on average in the widely-used Beijing taxi dataset (Yuan et al., 2010)). In Fig. 2(c) we report the outcome of a Map Matching process on the trajectories identified after Trajectory Partitioning and depicted in 2 (b).

In cases where there are considerable positioning errors and/or insufficient sampling rates on dense road networks, like urban areas, reconstructing the original trajectory of a vehicle can become quite challenging. In these cases, further **Trajectory Restoration/Interpolation** tasks may be required, to reconstruct plausible trajectories from the sparsely recorded positions (Li et al., 2020), according to properly defined heuristics.

After reconstructing these trajectories, the subsequent processing steps of the KDD mostly depend on the analysis goals. For example, many applications, such as the assessment of public transit performance or the feasibility of vehicular crowd-sensing solutions, require to evaluate the spatio-temporal road-network coverage of a fleet of vehicles (Asprone et al., 2021; Nguyen et al., 2018). To this end, specific coverage metrics need to be computed at the desired granularity level, such as the raw number of visits or the median timegap between subsequent visits of one of the considered vehicles (Masutani, 2015).

2.3. Implementing KDD pipelines

From a software development viewpoint, KDD pipelines can be designed and implemented as modular software systems, in which each step/task of the process is carried out by a specialized module, or functional block, with a clear separation of concerns among these

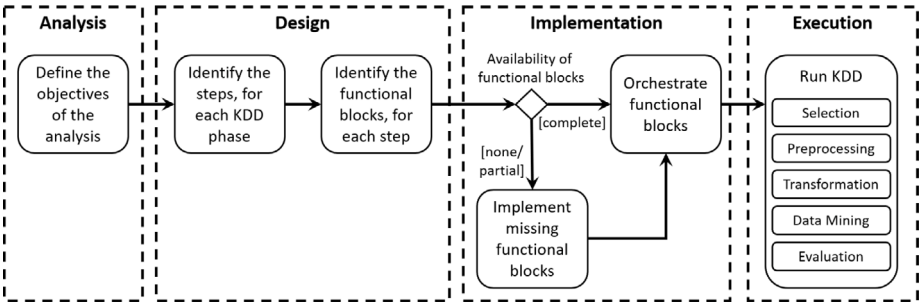


Fig. 3. Development lifecycle of a KDD pipeline.

blocks (Mikut & Reischl, 2011). Thanks to this modular design, it is possible to increase understandability, since functional blocks are a first conceptual abstraction of the pipeline, and make it easier to focus on what needs to be done rather than on how to actually do it. Moreover, a modular approach also fosters reusability, as a functional block carrying out a specific task can be easily reused in other KDD pipelines, in which the same task is needed.

The development lifecycle of a KDD pipeline as a modular system can be summarized as depicted in Fig. 3. The first phases are *Analysis*, in which the objectives of the pipeline are defined, and *Design*, in which high-level steps (*e.g.* the KDD phases) for the pipeline are defined. Subsequently, each high-level step is further detailed by defining one or more specific tasks (or functional blocks) that are necessary to carry out the step. Once the Design phase is completed, the *Implementation* phase begins, during which the functional blocks are developed (if not already available) and then properly orchestrated to compose the intended pipeline. As a result of this phase, the KDD pipeline is complete and ready to be executed. Indeed, the subsequent phase is the *Execution*, during which KDD steps are performed.

As highlighted in Section 1, general-purpose analytics platforms, based on a visual programming paradigm, can provide significant support to data analysts in the Design and Implementation phases. Indeed, the visual platforms force the analysts to design their pipelines in terms of interconnected functional blocks and typically offer out-of-the-box a comprehensive library of built-in components that can be used to carry out common tasks in KDD. In this way, the Implementation phase can be simplified too: if all identified blocks are already available as built-in components, then it is possible to proceed directly to their orchestration. Conversely, if one or more of the required functional blocks are not already available, they must be first implemented and then connected.

In this study, we leverage KNIME, an open-source analytics platform featuring a visual workbench that facilitates the composition and interactive execution of KDD pipelines. The user interface of the tool is displayed in Fig. 4. The main component of the graphical interface is the *Workflow Editor*, whose purpose is to allow users to visually build a KDD pipeline. The functional blocks that form these pipelines are depicted as *Nodes*, which can be chosen among the ones available from the *Node Repository* frame, and *drag-and-dropped* into the *Workflow Editor*. The KNIME platform encompasses an extensive array of built-in nodes, which can be employed to carry out various standard KDD tasks. These tasks include reading and writing files, data transformation, training and evaluating models using diverse machine/deep learning techniques, generating interactive visualization reports, and more. In the workbench, each node is visualized as a colored box featuring external interfaces, referred to as *input* and *output* ports, which are positioned on the left and right sides of the node, respectively, and depicted as black triangles. Moreover, the behavior of a node can be typically customized by accessing its graphical *Node Configuration Dialog*, in which relevant parameters (e.g.: the path of the file to read in a reader node, or the color to use in a plot in a visualization node) can be defined. To construct a pipeline, users graphically connect the output of one node to one or more inputs of the next node(s), effectively representing the successive tasks of the pipeline. To provide an example of how this type of tool works, in Fig. 4 we show a pipeline aimed at visualizing, by means of a scatter plot, some data loaded from a file. In detail, data is loaded from a Comma Separated Value (CSV) file by means of the *CSV Reader* node. Then, a *Row Filter* node is used to filter data, retaining only those matching the given criteria, specified in its configuration dialog. Finally, the *Color Manager* and the *Scatter Plot* nodes are used, respectively, to assign a color to each of the labels in the input data, and to visualize it in an interactive scatter plot (see *Interactive Visualization*). This pipeline can be executed by pressing the *play* icon on the top bar of the window.

Alongside its substantial collection of built-in nodes, KNIME is designed to be highly extensible, allowing the seamless integration of new, custom nodes to fulfill novel and/or specialized tasks. This has led to the creation of a massive ecosystem of third-party developed nodes, that can be downloaded, mostly from the Knime Hub,[4] and seamlessly combined with each other, as well as with built-in nodes.

2.4. Challenges of implementing KDD pipelines on mobility data

To the best of our knowledge, functional blocks implementing spatio-temporal data analysis/processing tasks like those described in Section 2.2 are typically not available in visual-based general-purpose platforms, like KNIME. Hence, scholars/practitioners willing to define KDD pipelines involving mobility data need to manually implement at least these mobility-specific tasks and functional blocks, or resort to not using visual platforms at all, giving up on the many benefits they provide. As a consequence, most of the works based on mobility data analytics presented in the literature dealt with these mobility-specific challenges using very specialized data processing pipelines, tailored to the specific tasks and datasets at hand. Consequently, even when available, the source code of these pipelines can prove to be quite difficult to re-execute or adapt to different datasets/scenarios, mostly due to the heterogeneity of the underlying tools. Therefore, researchers/practitioners willing to apply proposals from the literature to their own scenarios are often forced to re-implement analytic pipelines from scratch, which is time-consuming, error-prone and requires highly specialized skills.

In our industrial and academic experience, we found that scientists and practitioners working on mobility data are in practice either forced to buy highly specialized and costly commercial analytics platforms, such as DB4IoT,[5] Urban SDK[6] or IoKi,[7] or to create, often completely from scratch, complex and hard-to-reproduce scripts to orchestrate individual software fragments implementing the key KDD steps above described. Both these alternatives severely limit productivity and the replicability of the experiments within the ITS community.

[4] https://hub.knime.com/
[5] http://db4iot.com/
[6] https://www.urbansdk.com/platform/overview
[7] https://ioki.com/en/mobility-analytics/

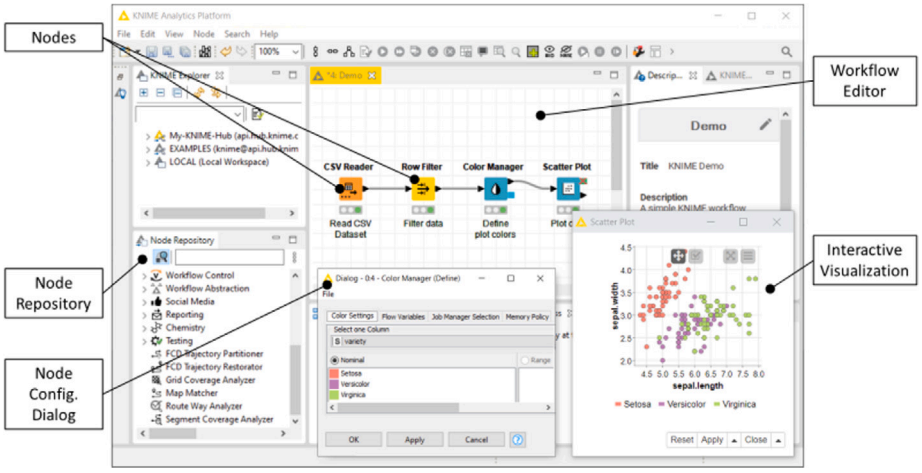


Fig. 4. The KNIME user interface.

Some attempts to address these problems have been proposed in the literature. For example, towards more general-purpose mobility data analytics approaches, Ruan, Li, Bao, He, and Zheng (2018) presented *CloudTP*, a cloud-based big data architecture (leveraging Apache Spark) designed to pre-process vehicular trajectories. CloudTP is a production-level solution to efficiently analyze vehicular trajectories in corporate IT environments, but its querying capabilities are quite basic and almost hard-coded, and its customizability, in general, is limited, as recognized by Fourati and Friedrich (2018), hindering its effectiveness in a research & development environment.

More recently, Fourati and Friedrich (2018) proposed *FANSI-tool*, an integrated, web-based solution to simplify FCD analytics, with querying and pre-processing capabilities. This solution, however, being a web-based application, might be difficult to integrate within larger workflows, requiring advanced programming skills to be extended and customized.

In summary, to date, there is a lack of effective solutions supporting KDD on spatio-temporal data, even within general-purpose visual analytics platforms. This motivated us to develop and distribute to the community a set of six software modules, intended as a KNIME extension, to support some common mobility-related tasks. In the following Section, we describe in detail each of the new components we developed.

3. The proposed solution

To provide an open and unified platform on which KDD processes involving mobility data can be easily implemented, we developed a modular solution to analyze heterogeneous spatio-temporal datasets, intended as an open-source plugin of the well-known KNIME Analytics Platform. In the following, we present the extension we developed, involving six new nodes explicitly meant for mobility data processing (depicted in Fig. 5). In the following, we broadly present each node, while technical details are reported in Appendix A.

3.1. The trajectory partitioner node

The first node we developed is meant to partition a massive FCD dataset composed of streams of positioning data collected from one or more vehicles, according to some pre-defined strategies. It produces two different outputs, namely the set of identified distinct trajectories and an augmentation of the input dataset, extended with trajectory data. Similarly to many other nodes we developed, the Trajectory Partitioner is designed to be easily extendable. Indeed, the partitioning strategy we devised can be easily replaced, by implementing a new, custom algorithm, compliant with a specific Java interface.

3.2. The Map Matcher node

The second node we developed is meant to perform the map matching of a sequence of positioning points, namely a trajectory, encoded as a WKT linestring, to an Open Street Map (OSM) road network (Haklay & Weber, 2008). This is a typical prerequisite for many location-based applications and use cases.

In our implementation, map-matching is realized through a well-known external routing solution, *i.e.* the *Open Source Routing Machine* (OSRM) (Luxen & Vetter, 2011), widely employed in several scientific studies (e.g.:Kaurav, Rout, & Vemireddy, 2021; Singh, Wu, Xiang, & Krishnaswamy, 2015). Also in this case, the node is easily extendable, as interested users can incorporate their own matching algorithms and/or road network representations.

3.3. The route calculator node

In many analytical/research scenarios, raw trajectories collected from vehicles might need to be properly pre-processed to make them suitable for analysis. For instance, when dealing with substantial positioning errors and/or extremely low sampling rates, more advanced trajectory restoration techniques (Li et al., 2020) can be employed to handle errors and fill gaps in the original raw trajectories. In other scenarios, an analyst might be interested in considering different routing alternatives for a given trajectory between an origin and a destination, for example, to evaluate the impact of different routing algorithms on mobility/ITS phenomena, as done in studies such as Asprone et al. (2021). To provide support in these contexts, we devised the *Route Calculator* node, processes a set of input trajectories and produces in output a set of new different trajectories, computed according to a customizable strategy.

3.4. The segment coverage analyzer node

In a number of analytical situations, it is essential to calculate the frequency with which each road segment in the considered map is traversed by the designated vehicles. For example, there is a wide literature related to this kind of analysis, within the Vehicular Crowd-Sensing (VCS) domain (e.g. Asprone et al., 2021; Masutani, 2015; Xu et al., 2019). To support this type of analysis at a road segment granularity level, we implemented the "*Segment Coverage Analyzer*" node. This node processes a sequence of map-matched trajectories, potentially computed by the Map Matcher node, and calculates two metrics: (I) the number of times each segment of the underlying road network was traversed, and (II) the average time gap between two subsequent traversals.

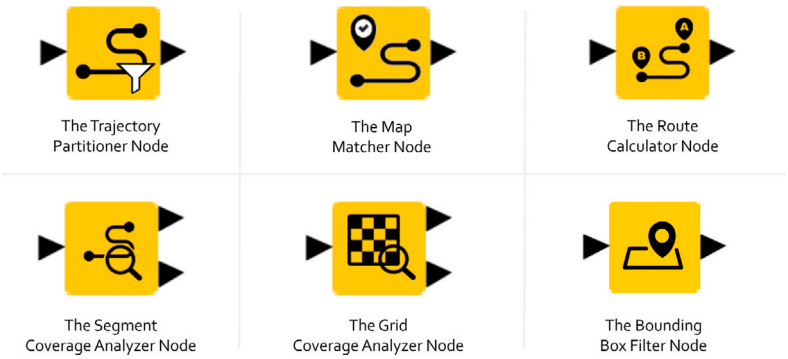


Fig. 5. The developed nodes.

3.5. The grid coverage analyzer node

The *Grid Coverage Analyzer* is designed to calculate spatio-temporal coverage metrics at a larger scale, encompassing entire urban areas rather than individual road segments. This kind of analysis is also common in the VCS domain, for tasks such as air quality or weather monitoring, that do not require a fine-grained, segment-level coverage analysis. The node enables users to graphically specify a grid over a given geographical area, using a bounding box and selecting the desired number of rows and columns for partitioning. Subsequently, it computes spatio-temporal coverage metrics for each grid cell at various temporal granularity levels.

3.6. The bounding-box filter node

In many analytical scenarios, researchers and practitioners dealing with spatial data may require their analyses to be limited to specific subsets of data, guided by spatial restrictions. This could be done, for example, to focus on a given region of interest. The *Bounding-Box Filter* takes as input a dataset containing at least one spatial feature, encoded in the standard WKT format, and enables users to perform data filtering depending on whether a certain spatial characteristic is fully contained within a customizable bounding box.

3.7. Comparison with other visual-based solutions

In this section, we compare, in terms of functionality, the proposed KNIME extension and a number of alternative, visual-based data analytics tools that are currently available on the market, including both open-source and commercial solutions.

Among those that are free-to-use and open-source, Weka is one of the most popular among researchers, data scientists, and practitioners. Weka (Holmes, Donkin, & Witten, 1994), short for *Waikato Environment for Knowledge Analysis*, is a comprehensive suite of machine learning algorithms and data preprocessing tools. Weka has been actively maintained since the 1990s (Holmes et al., 1994) and features a graphical workbench that allows users to visually define workflows (*knowledge flows*, in the Weka terminology), similarly to KNIME. Despite being mostly oriented towards training and evaluating machine learning models, some efforts have been directed towards extending the Weka platform for mobility data analytics. For example, Bogorny, Avancini, de Paula, Kuplich, and Alvares (2011) have developed an extension to include support for the annotation of vehicular trajectories with semantic data, while Sharma, Alam, and Rani (2012) have developed an extension to support clustering based on spatial features.

As for commercial solutions, RapidMiner (Hofmann & Klinkenberg, 2016) and FME (Feature Manipulation Engine)[8] are among the most widely adopted. Unlike open-source solutions, these tools are licensed products that require costly payments for their full feature set and support, and this can limit their accessibility, adaptability, and affordability for certain user groups.

RapidMiner, which features an educational program allowing free use for academics working in certain institutions, offers an integrated environment for general-purpose data mining, machine learning, and analytics. With its visual interface, users can intuitively design and execute complex data workflows, leveraging a vast library of built-in operators and algorithms. Thanks to third-party extensions, it is possible to include some limited spatial analytics capabilities in RapidMiner. For example, the GeoProcessing extension[9] implements dedicated components to manipulate Geometry objects (e.g.: compute intersections, unions, obtain coordinates from geometries, etc.). FME, on the other hand, focuses on the seamless exchange, transformation, and integration of data across diverse formats and systems, especially when dealing with spatial data. FME is mostly tailored towards automating data workflows, supporting connections with numerous and heterogeneous data sources, and simplifying complex data normalization and transformation tasks. Nonetheless, it offers limited support for more advanced manipulation and computation tasks.

To the best of our knowledge, all these above-mentioned visual data analytics platforms, even when considering also third-party extensions, lack the specialized functionalities that our solution brings to the table, being specifically tailored to address the challenges posed by spatio-temporal mobility data in the domain of ITS. Indeed, while there exist some overlaps in functionality (e.g.: the features offered by our *Bounding-Box Filter* node are also available in most of the other platforms through third-party extensions), no other visual solution allows users to easily integrate the key steps of map-matching, custom routing, and spatio-temporal coverage computation in their visual data analytics workflows.

4. Using the proposed toolkit in a real-world data-driven ITS analysis scenario

In this section, we show how the proposed extension can be used to realize data-driven analyses for ITS from massive mobility datasets. In particular, we replicate part of the work presented by Cruz Caminha, de Souza Couto, Maciel Kosmalski Costa, Fladenmuller, and Dias de Amorim (2018), on using public buses to crowd-sense information in an urban area. Acquiring contextual data is a key factor in building a smart city (Zanella, Bui, Castellani, Vangelista, & Zorzi, 2014). Thus, different works studied the suitability of sensor networks based on public vehicles (e.g. Cruz et al., 2020; Wang et al., 2021). However, analyzing the level of road network coverage achievable by these

[8] https://fme.safe.com

[9] https://marketplace.rapidminer.com/UpdateServer/faces/product_details.xhtml?productId=rmx_geoprocessing

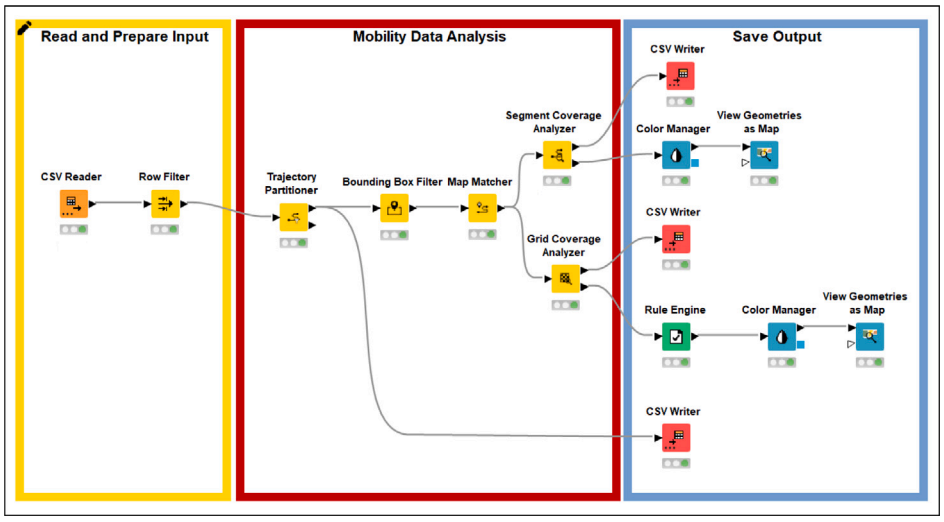


Fig. 6. The spatio-temporal coverage analysis workflow we developed, leveraging the proposed custom components in the Mobility Data Analysis phase.

vehicles is crucial to understanding whether using them for sensing is enough to perform given use cases (Asprone et al., 2021). To this aim, the first part of the paper by Cruz Caminha et al. (2018) analyzed the spatial coverage of a bus-based mobile wireless sensor network, using a large-scale, real-world mobility dataset collected from buses in the city of Rio de Janeiro, Brazil, over about two months.[10] The dataset contains basically millions of records, intended as FCD data plus vehicle_id and line of the bus.

As depicted in Fig. 3, to set up a KDD pipeline, we first need to define the objective of the analysis, namely analyzing the level of road network coverage achievable by the public buses of Rio de Janeiro. Then, we need to identify the required processing tasks and the corresponding functional blocks, *i.e.* the *nodes* of the KNIME tool. The KNIME workflow we realized to obtain information about bus coverage is shown in Fig. 6 and described in detail in the following. First of all, a built-in KNIME component (**CSV Reader**) is used to read the file in CSV format that contains the dataset. As done in Cruz Caminha et al. (2018), a filter is used (through the **Row Filter** KNIME node), retrieving only data collected on January 25th, 2019.

Then, our **Trajectory Partitioner** node is employed to split the trips of the buses, according to the implemented heuristic where a stream is divided whether there is a time gap higher than a threshold between two subsequent points. In the scenario we investigated, we set a threshold of 15 min. Furthermore, we discarded routes with less than 5 points, as done in many similar works (e.g.: Bock et al., 2020). At the end of this step, the node extracts 17,785 different trajectories, intended as linestrings obtained by connecting each couple of subsequent (noisy) GPS points belonging to the same trip. We save this output through a **CSV Writer** node. The **Bounding-Box Filter** node is used to discard trajectories falling outside the urban area of Rio de Janeiro. Then, to align raw and noisy GPS positions with the underlying representation of the road network provided by OpenStreetMap, we employ our **Map Matcher**. This concludes the pre-processing phase of our pipeline. Now it is possible to start the analytics tasks.

Leveraging the **Segment Coverage Analyzer** node, it is possible to calculate visit frequency achieved by the considered vehicles on the map, at a detailed level of individual road segments. Additionally, since the output represents road segments in a standard WKT format, it can be readily visualized, leveraging existing KNIME nodes such as **View Geometries as Map**, or saved in a CSV file for rendering in other GIS tools. Fig. 7 illustrates an example of such visualization, where road segments are highlighted on the map and color-coded according to how many times each of them was visited by a bus during the considered period. Segments in deep blue indicate less frequent visits, while segments in red represent the most heavily traversed ones.

Furthermore, we also conduct a broader map coverage analysis leveraging our **Grid Coverage Analyzer** component. We set up the node to cover an area of approximately 40 km^2 over the City of Rio de Janeiro, dividing it into a 10 by 10 grid of blocks, each encompassing an area of about 0.4 km^2 (see Fig. 8).

Moreover, as this analysis operates at a wider scale of entire city blocks, we have configured the component to register a new visit of a grid cell when 10 min or more have elapsed since the last sensing. The rationale behind this choice is that, in many scenarios requiring a coarse-grained analysis, such as for example real-time air quality monitoring, many temporally close vehicle passages in the same area are not necessarily useful, since they are likely to contain very similar information. We displayed the coverage information on an interactive map, depicted in Fig. 8, in which each block is colored according to a ramp of colors, proportional to the number of visits during the day.

Advantages of the proposed solution. When leveraging the proposed extension, we emphasize that the processing pipelines described above can be implemented within the KNIME Workbench with just a few clicks, without writing a single line of code. To highlight the potential benefits, in terms of productivity, of the proposed solution for ITS practitioners, we remark that one of the authors of the present work has previously faced a comparable ITS analytics scenario in the study outlined in Bock et al. (2020). That study aimed to assess the feasibility of utilizing high-mileage vehicles, such as taxis, to monitor the availability of on-street parking spaces in San Francisco. In that study, creating a KDD pipeline from scratch to compute the spatio-temporal coverage achieved by a fleet of taxis, similar to the ITS analysis scenario proposed above, demanded approximately two months of coding by two experienced research fellows.

It is worth pointing out that the proposed solution delivers a number of additional benefits that extend beyond the improvements in productivity. First, it empowers ITS domain experts with the capability to set up complex, data-driven ITS pipelines, without requiring advanced programming skills or costly commercial solutions.

Moreover, the proposed solution also helps mitigate another pain point in data-driven ITS research, i.e., the replicability and extendability of experimental artifacts. Indeed, in our industrial and academic experience, we found that the data processing pipelines implemented from scratch in many ITS studies, using programming languages such as Python, Java or R, are often hard to reproduce due to dependencies

[10] The dataset is freely available at https://www.kaggle.com/igorbalteiro/gps-data-from-rio-de-janeiro-buses

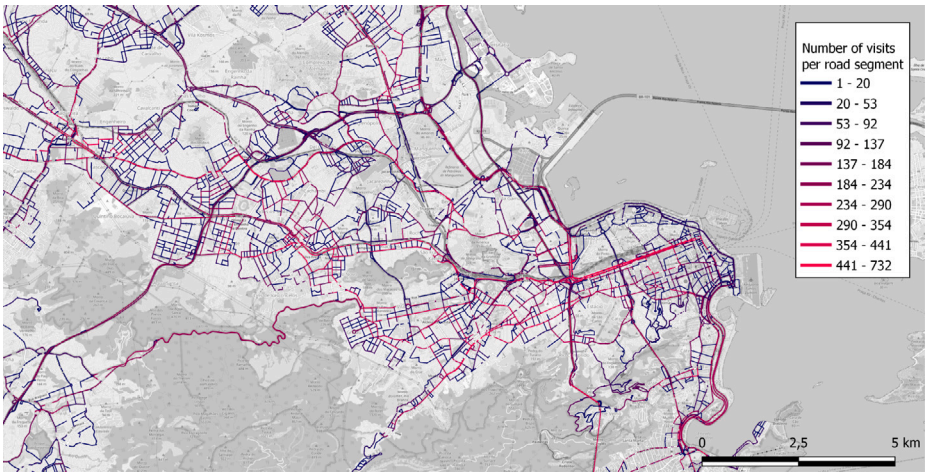


Fig. 7. Graphical visualization of road segment coverage achieved by all the buses in Rio de Janeiro in one day, as computed by the described KDD pipeline.

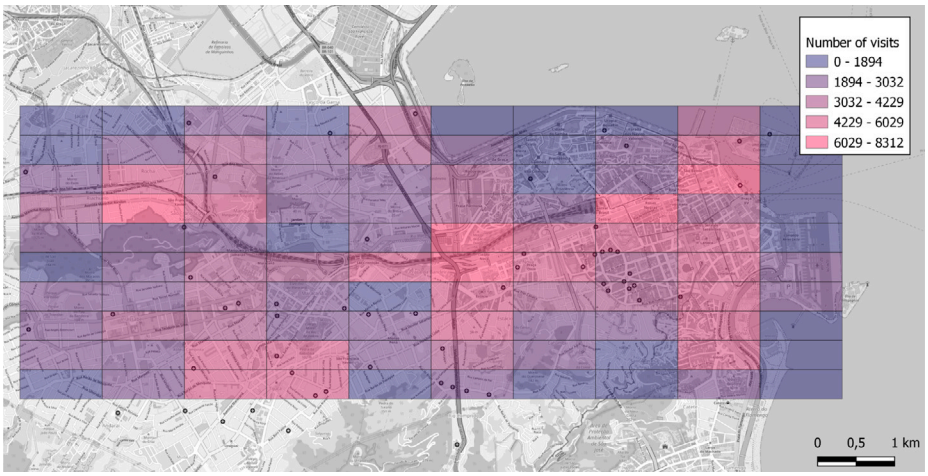


Fig. 8. Area coverage, intended as a 10 × 10 grid over the urban area, achieved by all the buses in Rio de Janeiro in one day, as computed by the described KDD pipeline.

on particular libraries, operating systems, and other execution environment settings. Moreover, the source code is generally difficult to comprehend and poorly documented, hindering the re-usability and extendability of the artifacts. On the other hand, pipelines defined using the KNIME platform and the proposed toolkit, thanks to the visual-based approach and the clear separation of concerns between the different processing nodes, are straightforward to comprehend, can be easily fine-tuned using a GUI, and can be distributed and re-executed on any machine capable of running KNIME. This significantly simplifies the distribution of research artifacts and improves replicability, which is essential to foster data-driven research in the ITS domain.

Lastly, our solution is designed to be straightforwardly used in combination with the vast ecosystem of KNIME components, which includes thousands of built-in and third-party nodes for managing different data sources, training and validating machine learning models, visualizing results, and more. This integration further empowers ITS practitioners to implement comprehensive workflows to extract knowledge from mobility data within a single platform, maximizing efficiency and convenience in their data-driven analyses.

5. Conclusions

Continuous advancements in mobile sensor networks have resulted in the availability of vast quantities of mobility data, which has significantly boosted data-driven proposals within the ITS community. However, extracting knowledge from these mobility datasets requires ITS practitioners to develop, often from scratch, intricate and difficult-to-reproduce scripts. This limits productivity and hinders the ability to replicate results and further extend research artifacts, to tailor them to different contexts.

This is also due to the fact that current general-purpose visual analytical platforms, such as KNIME, RapidMiner, or Weka, do not feature some necessary primitives to handle certain specialized steps of mobility dataset analysis, such as map-matching, road network coverage analysis, or trajectory partitioning.

To overcome such limitations and empower ITS researchers and practitioners to perform KDD tasks more effectively, we introduced an extension of the well-known KNIME Analytics Platform, introducing advanced preprocessing, transformation and analysis capabilities on mobility data. Specifically, we have introduced a set of new components that provide support for various tasks that are common in mobility data analysis. These tasks include map-matching, trajectory processing, spatio-temporal coverage analysis, and filtering based on geographical data.

We assessed the practicality and effectiveness of our proposal by using it to replicate a study published in 2018 in the ITS domain Cruz Caminha et al. (2018). More in detail, using the KNIME Analytics Platform in combination with our extension, we analyzed the spatio-temporal coverage achieved by public transit vehicles in Rio de Janeiro, leveraging a massive real-world mobility dataset. Notably, this replication was accomplished with remarkable ease, in a full visual fashion, without any manual coding or specialized scripts. Furthermore, the resulting

pipeline is easy to understand and fine-tune, using the Configuration GUIs associated with each component, and can also be readily distributed and re-executed on any system capable of running the KNIME platform.

To promote collaboration and facilitate the adoption of our solution, which we envision could bring significant improvements to productivity and experiment replicability in the ITS community, we made both the tool and its source code openly accessible on a dedicated GitHub repository (Di Martino et al., 2021).

Future works could be directed towards further expanding our proposal with new functionalities. For example, we are currently working on a novel component designed to infer trajectories of public transit vehicles starting from widely available General Transit Feed Specification (GTFS) files, commonly used in ITS analyses. Additionally, we are also currently exploring the feasibility of incorporating dedicated mobility simulators, like SUMO, into novel components, which could prove useful in investigating what-if mobility scenarios.

CRediT authorship contribution statement

Sergio Di Martino: Conceptualization, Methodology, Supervision, Writing – review & editing. **Enrico Landolfi:** Conceptualization, Writing – original draft. **Nicola Mazzocca:** Conceptualization, Supervision, Writing – review & editing. **Franca Rocco di Torrepadula:** Software, Investigation, Writing – original draft, Visualization, Resources, Data curation. **Luigi Libero Lucio Starace:** Conceptualization, Software, Investigation, Writing – original draft, Writing – review & editing, Visualization, Validation, Resources, Data Curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data and code availability

All the custom KNIME nodes we developed, along with their source code and detailed documentation and installation instructions, are freely available at the repository (Di Martino et al., 2021). The dataset we employed and the complete KNIME workflow we described in Section 4 are available as well at (Di Martino et al., 2022) for replication purposes.

Acknowledgments

This work was partially funded by the PNRR MUR project PE0000-013-FAIR.

Appendix A. Detailed description of the implemented components

In this Appendix, further details about the implementation and the configuration options of the developed components are provided.

A.1. The trajectory partitioner node

The Trajectory Partitioner Node aims at partitioning an extensive FCD dataset, comprising streams of positioning data gathered from one or multiple vehicles, into different trajectories, according to some heuristics. This node, along with its configuration dialog, is represented in Fig. A.9.

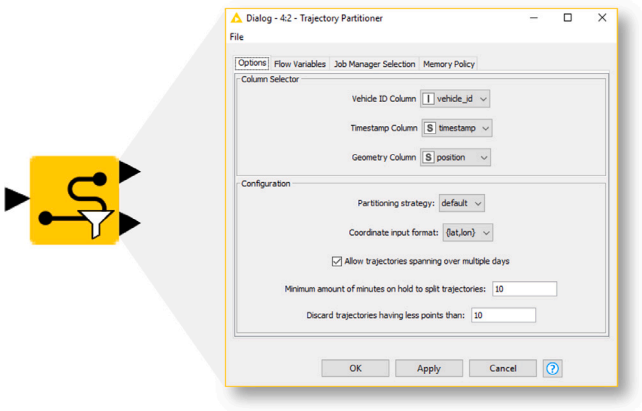


Fig. A.9. The Trajectory Partitioner node with its configuration dialog.

A.1.1. Inputs and outputs

The node takes as input a dataset representing the data stream of collected positions. The generic element of such dataset is a tuple of the form $(id, time, geom_{point}, attr_1, \ldots, attr_n)$, where id is the identifier of the vehicle, $time$ is the timestamp of the recording, and $geom_{point}$ is the GPS position in the *Well-Known Text* (WKT) format (*e.g.*: "`POINT(coord_x coord_y)`"), which is a ISO standard format to represent geometry objects defined by the Open Geospatial Consortium (Lott, 2015). Moreover, each record in the dataset might contain additional attributes ($attr_i$) representing supplementary recorded information (e.g.: speed, air quality level, etc.).

The node produces two different outputs: a set of trajectories and an augmentation of the input dataset with trajectory data. More in detail, the first output consists of records representing each distinct vehicular trajectory. In particular, each record is of the form $(id, time, geom_{line})$, where id is the vehicle id, $time$ is the timestamp representing the start time of the trajectory, and $geom_{line}$ is a geometry object representing the trajectory, encoded as a WKT *linestring* (e.g.: "`LINESTRING(coord_x1 coord_y1, . . . , coord_xn coord_yn)''`"). The second output is an augmentation of the input dataset with an additional trajectory ID attribute, associating each data point with the trajectory it belongs to.

A.1.2. Configuration options

As shown in Fig. A.9, the Trajectory Partitioner works according to a *partitioning strategy*, which can be customized through the dialog panel. By default, the node uses a heuristic we implemented (`default` partitioning strategy), based on the one presented in Bock et al. (2020), and working as follows. The FCD input stream is firstly grouped by vehicle ID and then sorted chronologically by timestamp. For each vehicle, the heuristic splits the stream in separate trajectories every time the vehicle remains stationary for more time than a customizable threshold. For example, a time gap of one hour between two subsequent data points might be due to the end of a trip/trajectory and the beginning of a new one. In that case, the previous trajectory is closed, and a new one is created, with the current FCD point being its first position. As depicted in Fig. A.9, the threshold is customizable through the node configuration dialog, using the *Minimum amount of minutes on hold to split trajectories* field. Furthermore, it is possible to specify: the *Coordinate input format* (lon, lat or lat, lon); whether a trajectory can span over multiple days (through the *Allow trajectories spanning over multiple days* checkbox); the minimum number of observations that a route must have to not be discarded (on the *Discard trajectories having less points than* field).

As most of the nodes we developed, the Trajectory Partitioner can be further extended by an interested user, by implementing a specific Java API we defined. In this way, it is possible to define customized

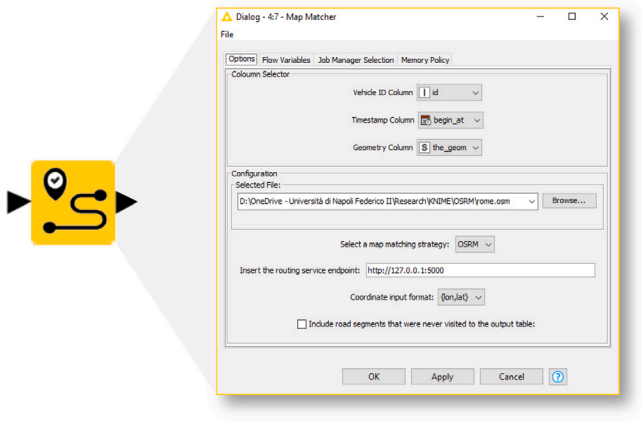


Fig. A.10. The Map Matcher node and its configuration dialog.

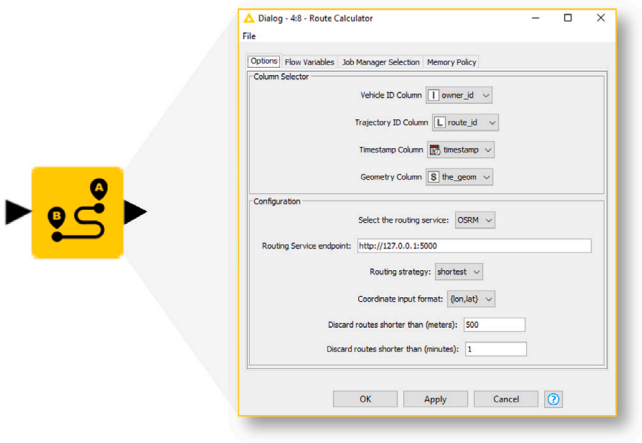


Fig. A.11. The Route Calculator node and its configuration dialog.

partitioning heuristics, also leveraging additional information in the considered dataset, such as, for example, the current occupancy status of a taxi.

A.2. The Map Matcher node

The Map Matcher Node, depicted in Fig. A.10, performs the map matching of a sequence of positioning points, namely a trajectory, to an Open Street Map (OSM) road network.

A.2.1. Inputs and outputs

This node, whose icon and configuration dialog are shown in Fig. A.10, takes as input a set of trajectories to match, represented as linestrings in the WKT format, potentially from a *Trajectory Partitioner* node. More in detail, input data consists of tuples of the form $(id, time, geom_{line}, attr_1, \ldots, attr_n)$, where id is the vehicle ID, $time$ is the starting time of the trajectory, $geom_{line}$ is the WKT linestring representing the trajectory, and $attr_1, \ldots, attr_n$ are additional optional attributes that are not considered in the map-matching process. The node computes, for each of the given input trajectories, the set of corresponding OSM road segments that are traversed. The output of the node is a set of matched segments consisting of tuples of the form $(id, traj\_id, start, end, origin, dest, tags, geom_{line})$. In particular, id is the vehicle ID, $traj\_id$ is the ID of the trajectory that traversed the road segment, $start$ and end are, respectively, the start and end timestamp of the visit to the road segment, $origin$ and $dest$ are, respectively, the OSM identifiers of the node from which the visited road segment starts and ends, $tags$ are the OSM tags associated with the road segment (containing information that might be useful in subsequent analyses), and $geom_{line}$ is a WKT representation of the visited OSM road segment.

A.2.2. Configuration options

In our implementation, map-matching is performed by means of a well-known external routing solution, namely the *Open Source Routing Machine* (OSRM) (Luxen & Vetter, 2011), widely used in many scientific studies (e.g.:Kaurav et al., 2021; Singh et al., 2015). Nevertheless, since the map-matching problem is still a very active research field (Chao, Xu, Hua, & Zhou, 2020), we designed this node to be easily extendable, allowing the interested user to potentially supply his/her own matching algorithms and/or road network representations by simply implementing a Java interface we defined. Indeed, the Map Matching Strategy can be specified in the node configuration dialog (depicted in Fig. A.10), along with the route decoder server and the path of the map data. Additional metadata to populate the *tags* field is fetched from OpenStreetMap data of the considered area, that the user can supply in the standard `.osm` format.

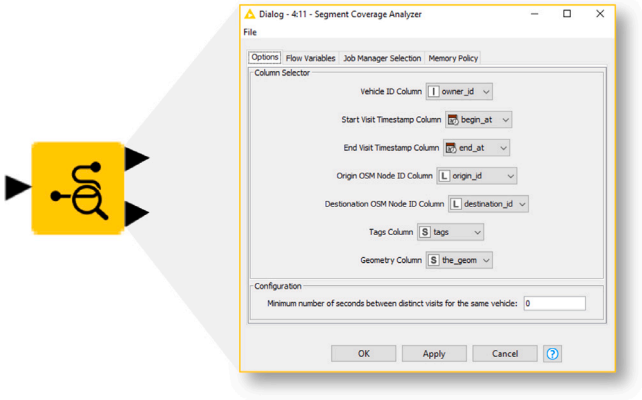


Fig. A.12. The Segment Coverage Analysis nodes and its configuration dialog.

A.3. The route calculator node

The aim of the Route Calculator Node (represented in Fig. A.11) is to calculate different routing alternatives for a given trajectory between an origin and a destination.

A.3.1. Inputs and outputs

The Route Calculator node takes as input a set of vehicular trajectories, containing the GPS positions associated with each trajectory. More in detail, a generic input record is a tuple of the form $(id, traj\_id, time, geom_{point}, attr_1, \ldots, attr_n)$, where id is the vehicle ID, $traj\_id$ is the trajectory ID, $time$ is a timestamp, $geom_{point}$ is a GPS position, and $attr_1, \ldots, attr_n$ are, as usual, optional attributes that are not considered by the Route Calculator node. Let us note that such input has the same structure as the second output (augmented GPS points) of the Trajectory Partitioner node, which can indeed be seamlessly fed to the Route Calculator node.

The Route Calculator node processes each input trajectory according to a customizable strategy, producing, as a result, a set of new trajectories, each being represented as a tuple of the form $(id, start_{point}, end_{point}, time, distance, begin\_at, end\_at, geom_{line})$. In particular, id is the vehicle ID, $start_{point}$ and end_{point} are, respectively, the GPS points from which the trajectory starts and ends, $time$ and $distance$ are the duration in seconds and the distance in meters of the trajectory, and $begin\_at$ and end_at are, respectively, the departure and arrival timestamps of the trajectory. Lastly, $geom_{line}$ is a WKT representation of the new trajectory.

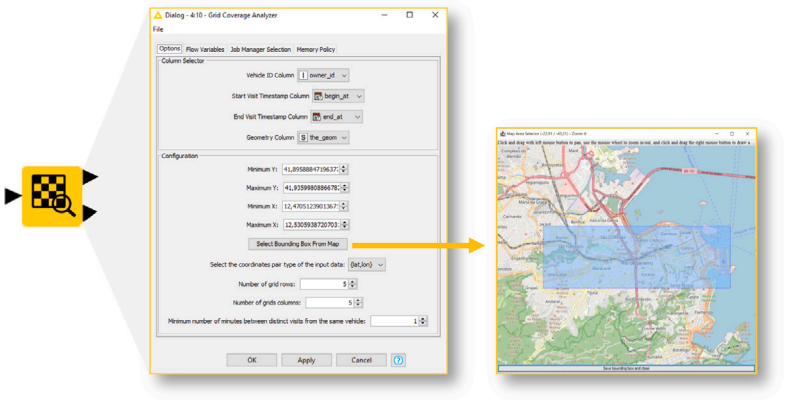


Fig. A.13. The Grid Coverage Analysis node and its configuration dialog.

A.3.2. Configuration options

As depicted in Fig. A.11, the current implementation of the Route Calculator node features a single routing strategy, which we called "*shortest path*", realized using the OSRM routing service. As the name suggests, this strategy creates new trajectories by computing the shortest path between the first and the last vehicular positions in the original trajectory, ignoring all the points in between. This strategy could help investigate what-if scenarios in which all vehicles followed the shortest possible path to their destination. Nonetheless, the node is designed to be highly extensible, allowing interested users to incorporate additional routing/restoration strategies to process the input trajectories. Furthermore, it is also possible to specify the *Coordinate input format*, and the minimum distance and duration of a trajectory (through the *Discard routes shorter than (meters/minutes)* fields).

A.4. The segment coverage analyzer node

The Segment Coverage Analyzer allows users to calculate spatio-temporal coverage metrics at the fine-grained level of individual road segments.

A.4.1. Inputs and outputs

This node, depicted in A.12, takes as input a set of map-matched trajectories, similar to the output produced by the Map Matching node (see Section 3.2), whose generic element is a tuple of the form $(id, traj\_id, start, end, origin, dest, tags, geom_{line})$. From this data, the Segment Coverage Analyzer produces two outputs: the first one reports road segment coverage metrics aggregated over the entire timespan of the input data, while the second one reports them on a finer-grained temporal scale of a single day and different selected time slots in each day (i.e.: 00.00 - 08.00; 08.00 - 14.00; 14.00 - 20.00; 20.00 - 24.00). In greater detail, the first output provides information for each road segment in the analyzed road network, including the number of times that segment is visited by one of the vehicles, along with the average and median time gaps between consecutive visits. Moreover, OpenStreetMap metadata for each road segment, which could be useful in subsequent analyses, are retained. The second output, on the other hand, reports the same information for each day and for each selected time slot.

A.4.2. Configuration options

As shown in Fig. A.12, the node configuration panel allows users to specify a number n of seconds, in order to ignore all visits that are less than n seconds away from the last one. The rationale behind this configuration option is to adapt to scenarios requiring different sensing frequencies. For example, in a scenario in which probe vehicles are used to monitor road pavement status, a subsequent visit on a given road segment being only 5 s apart from the previous one is likely not providing new information, since road pavement status is not expected to change so rapidly. This parameter allows analysts to discard visits that are likely to be not useful since they are too temporally close to a previous one.

A.5. The grid coverage analyzer node

Similar to the previously described Segment Coverage Analyzer node, the Grid Coverage Analyzer is designed to calculate spatio-temporal coverage metrics. However, it functions at a coarser-grained scale, encompassing entire urban areas instead of individual road segments.

A.5.1. Inputs and outputs

The Grid Coverage Analyzer, depicted in Fig. A.13, uses the same input as the Segment Coverage Analyzer (see Section 3.4), and the outputs are also similar to the ones provided by that node. Indeed, the Grid Coverage Analyzer node features two outputs. The first output reports, for each cell in the defined geographical grid and considering the entire timespan of the input data, the number of times that cell is visited by one of the vehicles, as well as the average and median timegaps between subsequent visits. The second output, on the other hand, reports the coverage metrics on a finer-grained temporal scale of a single day and different selected time slots in each day (i.e.: 00.00 - 08.00; 08.00 - 14.00; 14.00 - 20.00; 20.00 - 24.00). The availability of reports at different temporal granularity levels could prove to be useful in many Vehicular Crowd-Sensing scenarios, to understand the spatio-temporal distribution of the trajectories over a given area. For example, let us assume that a decision maker is willing to assess whether a given fleet of vehicles is suitable to crowd-sense air quality in a given area, corresponding to a certain grid cell (Devarakonda et al., 2013). If 10 vehicles were to visit that grid cell with an even distribution throughout the day, this would probably be adequate for monitoring the air quality in that area. If, on the other hand, the same visits were all concentrated between 8.00 AM and 8.15 AM, they would not provide enough information.

A.5.2. Configuration options

As shown in Fig. A.13, through the dialog panel of the Grid Coverage Analyzer it is possible to select a custom grid over a given geographical area, either by manually specifying the minimum and the maximum latitude/longitude of the grid or by straightforwardly drawing the bounding box on an interactive map. Moreover, it is possible to indicate the coordinates pair type, the number of columns/rows of the grid, and the minimum number of minutes passing between distinct recordings.

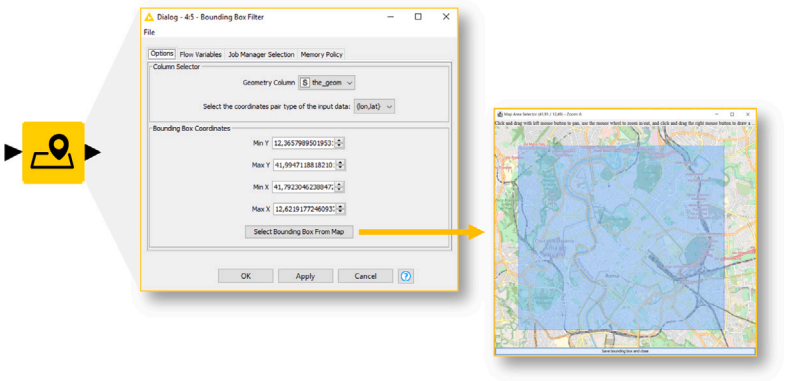


Fig. A.14. The Geometry Filter node and its configuration dialog.

A.6. The bounding-box filter node

The Bounding-Box Filter node, depicted in Fig. A.14, enables users to filter data by determining whether a given spatial feature is entirely contained within a customizable bounding box.

A.6.1. Inputs and outputs

The Bounding-Box Filter node takes as input any dataset containing at least one column representing spatial objects, encoded in the standard WKT format, and allows users to define their own desired bounding box. As a result, the node produces the subset of the input data such that the spatial objects in the selected geometry column are fully contained in the given bounding box.

A.6.2. Configuration options

Similarly to the Grid Coverage Analyzer node, users can select a Bounding Box by either specifying the minimum and maximum latitude/longitude or by straightforwardly drawing it on an interactive map, as shown in Fig. A.14, on the right. The node supports all standard WKT objects, including points, linestrings and polygons, making it directly applicable for refining the output of the other nodes we have developed.

References

Asprone, D., Di Martino, S., Festa, P., & Starace, L. L. L. (2021). Vehicular crowd-sensing: A parametric routing algorithm to increase spatio-temporal road network coverage. *International Journal of Geographical Information Science*, http://dx.doi.org/10.1080/13658816.2021.1893737.

Bock, F., Di Martino, S., & Origlia, A. (2020). Smart parking: Using a crowd of taxis to sense on-street parking space availability. *IEEE Transactions on Intelligent Transportation Systems*, *21*(02), 496–508. http://dx.doi.org/10.1109/TITS.2019.2899149.

Bogorny, V., Avancini, H., de Paula, B. C., Kuplich, C. R., & Alvares, L. O. (2011). Weka-STPM: A software architecture and prototype for semantic trajectory data mining and visualization. *Transactions in GIS*, *15*(2), 227–248.

Bracciale, L., Bonola, M., Loreti, P., Bianchi, G., Amici, R., & Rabuffi, A. (2014). CRAWDAD dataset roma/taxi (v. 2014–07–17). http://dx.doi.org/10.15783/C7QC7M, Downloaded from https://crawdad.org/roma/taxi/20140717.

Chao, P., Xu, Y., Hua, W., & Zhou, X. (2020). A survey on map-matching algorithms. In *Australasian database conference* (pp. 121–133). Springer.

Cruz, P., Couto, R. S., Costa, L. H. M., Fladenmuller, A., & de Amorim, M. D. (2020). Per-vehicle coverage in a bus-based general-purpose sensor network. *IEEE Wireless Communications Letters*, *9*(7), 1019–1022.

Cruz Caminha, P. H., de Souza Couto, R., Maciel Kosmalski Costa, L. H., Fladenmuller, A., & Dias de Amorim, M. (2018). On the coverage of bus-based mobile sensing. *Sensors*, *18*(6), 1976.

Devarakonda, S., Sevusu, P., Liu, H., Liu, R., Iftode, L., & Nath, B. (2013). Real-time air quality monitoring through mobile sensing in metropolitan areas. In *Proceedings of the 2nd ACM SIGKDD international workshop on urban computing* (p. 15). ACM.

Di Lorenzo, G., Sbodio, M., Calabrese, F., Berlingerio, M., Pinelli, F., & Nair, R. (2015). Allaboard: Visual exploration of cellphone mobility data to optimise public transport. *IEEE Transactions on Visualization and Computer Graphics*, *22*(2), 1036–1050.

Di Martino, S., Landolfi, E., Mazzocca, N., Rocco di Torrepadula, F., & Starace, L. L. L. (2022). Replication package: A visual-based toolkit to support data-driven intelligent transportation systems analysis. http://dx.doi.org/10.5281/zenodo.7120432, Dataset and materials for replicability on Zenodo.

Di Martino, S., Principe, S. M., & Starace, L. L. L. (2021). KNOT - GitHub repository. URL: https://github.com/luistar/knot.

Di Martino, S., & Starace, L. L. L. (2022a). Towards uniform urban map coverage in vehicular crowd-sensing: A decentralized incentivization solution. *IEEE Open Journal of Intelligent Transportation Systems*, *3*, 695–708. http://dx.doi.org/10.1109/OJITS.2022.3211540.

Di Martino, S., & Starace, L. L. L. (2022b). Vehicular crowd-sensing on complex urban road networks: A case study in the City of Porto. *Transportation Research Procedia*, *62*, 350–357.

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, *17*(3), 37.

Fourati, W., & Friedrich, B. (2018). FANSI-tool: An integrated software for floating data analytics. In *25th ITS world congress, Copenhagen, Denmark*.

Haklay, M., & Weber, P. (2008). Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, *7*(4), 12–18.

Hofmann, M., & Klinkenberg, R. (2016). *RapidMiner: data mining use cases and business analytics applications*. CRC Press.

Holmes, G., Donkin, A., & Witten, I. H. (1994). Weka: A machine learning workbench. In *Proceedings of ANZIIS'94-Australian New Zealand intelligent information systems conference* (pp. 357–361). IEEE.

Jenelius, E. (2019). Data-driven bus crowding prediction based on real-time passenger counts and vehicle locations. In *6th international conference on models and technologies for intelligent transportation systems*.

Kaurav, R. S., Rout, R. R., & Vemireddy, S. (2021). Blockchain for emergency vehicle routing in healthcare services: An integrated secure and trustworthy system. In *2021 international conference on communication systems networks* (pp. 623–628). http://dx.doi.org/10.1109/COMSNETS51098.2021.9352903.

Khan, S. M., Rahman, M., Apon, A., & Chowdhury, M. (2017). Chapter 1 - characteristics of intelligent transportation systems and its relationship with data analytics. In M. Chowdhury, A. Apon, & K. Dey (Eds.), *Data analytics for intelligent transportation systems* (pp. 1–29). Elsevier, http://dx.doi.org/10.1016/B978-0-12-809715-1.00001-8, URL: https://www.sciencedirect.com/science/article/pii/B9780128097151000018.

Kubicka, M., Cela, A., Mounier, H., & Niculescu, S.-I. (2018). Comparative study and application-oriented classification of vehicular map-matching methods. *IEEE Intelligent Transportation Systems Magazine*, *10*(2), 150–166.

Lee, K., Eo, M., Jung, E., Yoon, Y., & Rhee, W. (2021). Short-term traffic prediction with deep neural networks: A survey. *IEEE Access*, *9*, 54739–54756.

Lee, U., Zhou, B., Gerla, M., Magistretti, E., Bellavista, P., & Corradi, A. (2006). Mobeyes: Smart mobs for urban monitoring with a vehicular sensor network. *IEEE Wireless Communications*, *13*(5), 52–57.

Li, B., Cai, Z., Kang, M., Su, S., Zhang, S., Jiang, L., et al. (2020). A trajectory restoration algorithm for low-sampling-rate floating car data and complex urban road networks. *International Journal of Geographical Information Science*, 1–24.

Lott, R. (2015). *Geographic information-Well-known text representation of coordinate reference systems*. Open Geospatial Consortium.

Luxen, D., & Vetter, C. (2011). Real-time routing with OpenStreetMap data. In *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems* (pp. 513–516). New York, NY, USA: ACM, http://dx.doi.org/10.1145/2093973.2094062, URL: http://doi.acm.org/10.1145/2093973.2094062.

Masutani, O. (2015). A sensing coverage analysis of a route control method for vehicular crowd sensing. In *Pervasive computing and communication workshops (PerCom Workshops), 2015 IEEE international conference on* (pp. 396–401). IEEE.

Mikut, R., & Reischl, M. (2011). Data mining tools. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *1*(5), 431–443.

Nguyen, K., Yang, J., Lin, Y., Lin, J., Chiang, Y.-Y., Shahabi, C., et al. (2018). Los angeles metro bus data analysis using GPS trajectory and schedule data. *Technical Report*, California. Department of Transportation.

Nuzzolo, A., Comi, A., Papa, E., & Polimeni, A. (2018). Understanding taxi travel demand patterns through floating car data. In *The 4th conference on sustainable urban mobility* (pp. 445–452). Springer.

Piorkowski, M., Sarafijanovic-Djukic, N., & Grossglauser, M. (2009). CRAWDAD dataset epfl/mobility (v. 2009-02-24). http://dx.doi.org/10.15783/C7J010, Downloaded from http://crawdad.org/epfl/mobility/20090224.

Ruan, S., Li, R., Bao, J., He, T., & Zheng, Y. (2018). Cloudtp: A cloud-based flexible trajectory preprocessing framework. In *2018 IEEE 34th international conference on data engineering* (pp. 1601–1604). IEEE.

Sharma, R., Alam, M. A., & Rani, A. (2012). K-means clustering in spatial data mining using weka interface. In *International conference on advances in communication and computing technologies, vol. 26* (p. 30).

Singh, A. D., Wu, W., Xiang, S., & Krishnaswamy, S. (2015). Taxi trip time prediction using similar trips and road network data. In *2015 IEEE international conference on big data* (pp. 2892–2894). IEEE.

Van Der Aalst, W. (2016). Data science in action. In *Process mining* (pp. 3–23). Springer.

Veres, M., & Moussa, M. (2019). Deep learning for intelligent transportation systems: A survey of emerging trends. *IEEE Transactions on Intelligent Transportation Systems*, *21*(8), 3152–3168.

Wang, P., Huang, Z., Lai, J., Zheng, Z., Liu, Y., & Lin, T. (2021). Traffic speed estimation based on multi-source GPS data and mixture model. *IEEE Transactions on Intelligent Transportation Systems*.

Xu, S., Chen, X., Pi, X., Joe-Wong, C., Zhang, P., & Noh, H. Y. (2019). iLOCuS: Incentivizing vehicle mobility to optimize sensing distribution in crowd sensing. *IEEE Transactions on Mobile Computing*.

Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., et al. (2010). T-drive: driving directions based on taxi trajectories. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems* (pp. 99–108).

Zanella, A., Bui, N., Castellani, A., Vangelista, L., & Zorzi, M. (2014). Internet of things for smart cities. *IEEE Internet of Things Journal*, *1*(1), 22–32.

Zhang, J., Shen, D., Tu, L., Zhang, F., Xu, C., Wang, Y., et al. (2017). A real-time passenger flow estimation and prediction method for urban bus transit systems. *IEEE Transactions on Intelligent Transportation Systems*, *18*(11), 3168–3178.

Zhang, J., Wang, F.-Y., Wang, K., Lin, W.-H., Xu, X., & Chen, C. (2011). Data-driven intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, *12*(4), 1624–1639.

Zheng, Y., Wu, W., Chen, Y., Qu, H., & Ni, L. M. (2016). Visual analytics in urban computing: An overview. *IEEE Transactions on Big Data*, *2*(3), 276–296.

Zhu, L., Yu, F. R., Wang, Y., Ning, B., & Tang, T. (2018). Big data analytics in intelligent transportation systems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, *20*(1), 383–398.