Evaluating Accessibility of Los Angeles Metropolitan Area Using Data-Driven Time-Dependent Reachability Analysis

July 2023

A Research Report from the National Center for Sustainable Transportation

Cyrus Shahabi, University of Southern California

Seon Ho Kim, University of Southern California

National Center for Sustainable Transportation

METRANS Transportation Consortium
USC | CSULB

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. NCST-USC-RR-23-25	**2. Government Accession No.** N/A	**3. Recipient's Catalog No.** N/A

4. Title and Subtitle Evaluating Accessibility of Los Angeles Metropolitan Area Using Data-Driven Time-Dependent Reachability Analysis	**5. Report Date** July 2023
	6. Performing Organization Code N/A
7. Author(s) Seon Ho Kim, Ph.D., https://orcid.org/0000-0002-8410-0839 Cyrus Shahabi, Ph.D., https://orcid.org/0000-0001-9118-0681	**8. Performing Organization Report No.** N/A
9. Performing Organization Name and Address University of Southern California METRANS Transportation Consortium University Park Campus, VKC 367 MC:0626 Los Angeles, California 90089-0626	**10. Work Unit No.** N/A
	11. Contract or Grant No. USDOT Grant 69A3551747114
12. Sponsoring Agency Name and Address U.S. Department of Transportation Office of the Assistant Secretary for Research and Technology 1200 New Jersey Avenue, SE, Washington, DC 20590	**13. Type of Report and Period Covered** Final Research Report (August 2021 – August 2022)
	14. Sponsoring Agency Code USDOT OST-R

15. Supplementary Notes

DOI: https://doi.org/10.7922/G2DB8049
Dataset DOI: https://doi.org/10.5061/dryad.4j0zpc8gf

16. Abstract

This project is to investigate how accessibility of city blocks is quantified through the transport systems and real traffic flow data from the Los Angeles Metropolitan Area. The authors investigate the reachability problem and provide a solution with a functional system that is capable of visualizing the reachability map (isochrone). Unlike other studies, this approach is data-driven and does not depend on mathematical graph-theory to compute the isochrone which requires intensive computation. Instead, it focuses on directly processing the large amount of traffic flow data that the Integrated Media Systems Center at USC has collected from the Regional Integration of Intelligent Transportation Systems (RIITS) for more than 10 years under the Center's existing Archived Traffic Data Management System (ADMS) project. The reachability map construction is based on vehicle trajectories so the researchers devised the Data-Driven Trajectory Generator (DDTG), a data-driven, model-free, and parameter-less algorithm for generating realistic vehicle trajectory datasets from ADMS data. Since real world traffic is incomplete with lots of temporal and spatial missing data, the researchers studied imputation and interpolation methods to complete the dataset. Their experiments with real-world trajectory and traffic data show that the datasets generated by DDTG follow distributions that are very close to the distributions of a real trajectory dataset. Furthermore, to demonstrate the results from the proposed research, a web application was developed in which users can select a location, travel time, and the time of year to see the evaluated accessibility info in the form of an isochrone map. The outcomes of this project—synthetic vehicle trajectory dataset and reachability map construction—will be helpful in evaluating accessibility of city blocks for transport systems over a large area, essential for policymakers for effective city planning as well as to improve the well-being of citizens.

17. Key Words Traffic index, synthetic trajectory, reachability	**18. Distribution Statement** No restrictions.

19. Security Classif. (of this report) Unclassified	**20. Security Classif. (of this page)** Unclassified	**21. No. of Pages** 52	**22. Price** N/A

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

About the National Center for Sustainable Transportation

The National Center for Sustainable Transportation is a consortium of leading universities committed to advancing an environmentally sustainable transportation system through cutting-edge research, direct policy engagement, and education of our future leaders. Consortium members include: University of California, Davis; University of California, Riverside; University of Southern California; California State University, Long Beach; Georgia Institute of Technology; and University of Vermont. More information can be found at: ncst.ucdavis.edu.

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

The U.S. Department of Transportation requires that all University Transportation Center reports be published publicly. To fulfill this requirement, the National Center for Sustainable Transportation publishes reports on the University of California open access publication repository, eScholarship. The authors may copyright any books, publications, or other copyrightable materials developed in the course of, or under, or as a result of the funding grant; however, the U.S. Department of Transportation reserves a royalty-free, nonexclusive and irrevocable license to reproduce, publish, or otherwise use and to authorize others to use the work for government purposes.

Acknowledgments

This study was funded, partially or entirely, by a grant from the National Center for Sustainable Transportation (NCST), supported by the U.S. Department of Transportation (USDOT) through the University Transportation Centers program. The authors would like to thank the NCST and the USDOT for their support of university-based research in transportation, and especially for the funding provided in support of this project.

Evaluating Accessibility of Los Angeles Metropolitan Area Using Data-Driven Time-Dependent Reachability Analysis

A National Center for Sustainable Transportation Research Report

July 2023

Cyrus Shahabi and Seon Ho Kim

Integrated Media Systems Center, University of Southern California

TABLE OF CONTENTS

List of Figures

Evaluating Accessibility of Los Angeles Metropolitan Area Using Data-Driven Time-Dependent Reachability Analysis

EXECUTIVE SUMMARY

Accessibility to people, goods, services and places forms the basis of economic development in a city. The better and more efficient this access, the greater the economic benefits. As the population and employment grow in a city and the level of agglomeration gets higher, the city tends to have higher GDP per capita and higher levels of productivity. The way in which cities facilitate accessibility through their urban forms and transport systems also impacts directly on other measures of human development and well-being. Reliable evaluation of accessibility of city blocks for transport systems over a large area is essential for policymakers to achieve effective city planning as well as to improve well-being of citizens.

One major way to evaluate the accessibility of public facilities such as hospitals is reachability analysis which determines whether certain city blocks can be reached from at least one of these facilities within a certain time span. An isochrone is generally defined as a curve drawn on a map connecting points at which moving objects (e.g., cars) arrive at the same time for a given starting point and time duration and it is one of the most efficient tools used in addressing the reachability analysis problem. Thus, their construction is an important task in many application domains. As an example, in urban planning, isochrones are essential when assessing the placement of public services like hospitals and fire departments.

The objective of this proposal is to investigate how accessibility of city blocks is quantified through the transport systems and real traffic flow data from the Los Angeles Metropolitan Area. We will formally define the isochrone and reverse isochrone problems, describe our approach to solving them and provide a fully functional system that is capable of visualizing reachability maps. Unlike other studies, our approach is purely data-driven and does not depend on mathematical graph-theory to compute the isochrone which requires an intensive computation. Instead, we focus on directly processing the large amount of traffic flow data that our research center, the Integrated Media Systems Center at USC, has acquired from the Los Angeles County. This research exploits real-world big traffic sensor data collected from the Regional Integration of Intelligent Transportation Systems (RIITS) for more than ten years under our existing Archived Traffic Data Management System (ADMS) project.

We use a data-driven grid-based approach for constructing time-dependent isochrone maps considering specific time of day, week, and month. Our reachability map construction is based on vehicle trajectories, however not many real trajectory data are publicly available in reality due to various reasons such as privacy. Thus, we devised the Data-Driven Trajectory Generator, dubbed DDTG, a data-driven, model-free, and parameter-less algorithm for generating realistic synthetic vehicle trajectory datasets from real-world ADMS traffic flow data. Since real world traffic flow data are incomplete with lots of temporal and spatial missing values, we studied imputation and interpolation methods to complete the dataset before using them for synthetic trajectory generation.

The main contributions of our work are:

- Study and evaluate 1) imputation methods to handle temporally missing values for different lengths of imputing gaps, and 2) interpolation methods to estimate the traffic conditions near locations where sensor data is not available.
- Propose an algorithm that aggregates traffic data over a region of interest and constructs a Traffic Index that summarizes the traffic flow (i.e., average speed) at a certain level of analysis zones.
- Propose DDTG, a model-free algorithm that generates synthetic yet realistic vehicle trajectory datasets, and a set of metrics to evaluate the quality of generated datasets at both holistic and individual levels.
- Generate lots of realistic synthetic vehicle trajectories in the City of Los Angeles using DDTG and conduct comprehensive experiments to evaluate the quality of the generated trajectory datasets.
- Develop a web application (https://imscwww.usc.edu/app/) in which users can select a location, travel time, and the time of year and see the evaluated accessibility info in the form of isochrone maps.

The outcomes of this project, synthetic vehicle trajectory dataset and reachability map construction will be helpful in evaluating accessibility of city blocks for transport systems over a large area so will become essential for policymakers to achieve effective city planning as well as to improve well-being of citizens.

1. Introduction

Accessibility to people, goods, services and places forms the basis of economic development and public health in a city. The better and more efficient this access, the greater the economic benefits and the better the public health. As the population and employment grow in a city and the level of agglomeration gets higher, the city tends to have higher GDP per capita and higher levels of productivity. The way in which a city facilitates accessibility through their urban forms and transport systems also impacts directly on other measures of human development and well-being. Reliable evaluation of accessibility of city blocks for transport systems over a large area is essential for policymakers to achieve effective city planning as well as to improve well-being of citizens.

One major way to evaluate the accessibility of public facilities (e.g., hospitals) is reachability analysis which determines whether certain city blocks can be reached from at least one of these facilities within a certain time span. An isochrone is generally defined as a curve drawn on a map connecting points to which moving objects (e.g., cars) leaving from a specific origin can arrive at the same time and it is one of the most efficient tools used in addressing the reachability analysis problem (Figure 1) [45]. Thus, their construction is an important step for many transportation applications. As an example, in urban planning, isochrones are essential when assessing the placement of public services like hospitals and fire departments. However, accessibility is a complex function of origin locations, destination locations, the underlying transportation network, the dynamic weights on the edges of the networks representing travel times during different times of the day, etc. Among many factors, the cost measurements are essential and usually take a variety of forms such as any combinations of Euclidean distance, network distance, travel time, monetary cost or fare, comfort or subjective ease of travel. Effective and efficient access is the goal of most transportation activities. To understand accessibility, a proper evaluation of accessibility is essential. However, it is challenging due to many factors affecting accessibility in a city: transportation demand and activities, mobility, transportation modes, land use, transport network connectivity, and even teleworking [1].

The objective of this project is to investigate how accessibility of city blocks is quantified through the transport systems and real traffic flow data from the Los Angeles Metropolitan Area. The accurate and efficient construction of reachability maps is a challenging task. Towards that end, first, we will analyze the reachability at different times of the day and for that we need to construct time-dependent isochrone maps. Graph theory techniques attempt to solve this by assigning dynamic time-varying weights on edges [2], however, these techniques do not scale well to large and complex graphs such as large Los Angeles road networks.

Unlike other conventional studies, our approach is data-driven and does not depend on mathematical graph-theory to compute the isochrone which is computationally expensive. Instead, we focus on directly processing the large amount of traffic flow data that our research center, the Integrated Media Systems Center at USC, has acquired from the Los Angeles County in the past decade. We devised a data-driven grid-based approach for constructing time-dependent isochrone maps considering specific time of day, week, and month with the past, present and predicted data. The outcome of our work is the construction of time-dependent isochrone map display web application. To demonstrate the results from the proposed research, we developed a web application in which users can select a location, travel time, and the time of year and see the evaluated accessibility information in the form of isochrone maps.

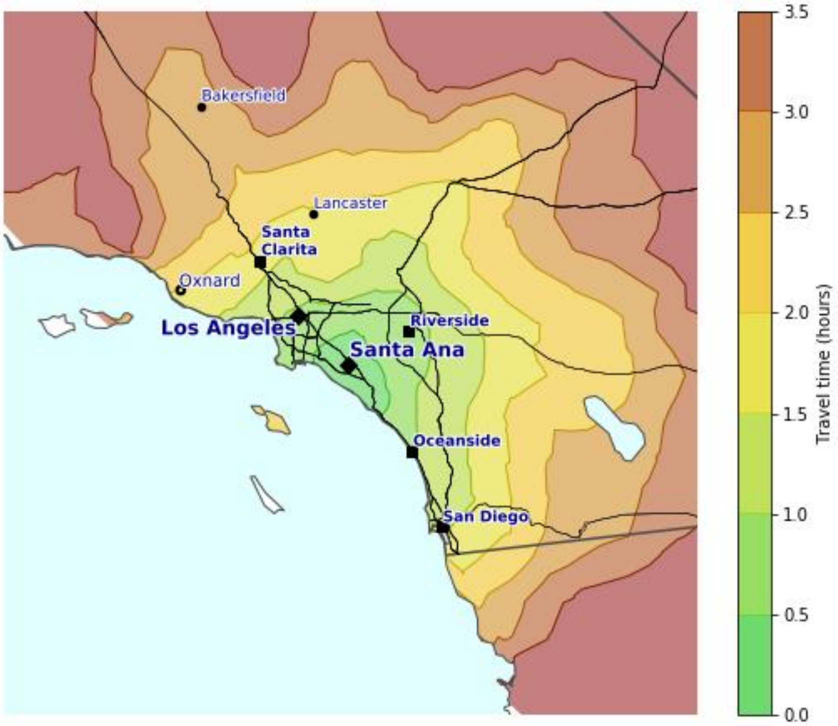


Figure 1. An example of isochrone map

1.1. Prior Work on Traffic Data Collection and Archiving

At USC's Integrated Media Systems Center (IMSC) with our partnership with Los Angeles Metropolitan Transportation Authority (LA Metro) and METRANS, we have developed a big transportation data warehouse: Archived Traffic Data Management System (ADMS) [3]. ADMS fuses and analyzes a very large-scale and high-resolution (both spatial and temporal) traffic sensor data from different transportation authorities in Southern California, including California Department of Transportation (Caltrans), Los Angeles Department of Transportation (LADOT), California Highway Patrol (CHP), Long Beach Transit (LBT). This dataset includes both inventory and real-time data with update rate as high as every 30 seconds for freeway and arterial traffic sensors (14,500 loop-detectors) covering 4,300 miles, 2,000 bus, and train automatic vehicle location (AVL), incidents such as accidents, traffic hazards and road closures reported (approximately 400 per day) by LAPD and CHP, and ramp meters. We have been continuously collecting and archiving datasets for the past 10 years. ADMS, with 11TB annual growth, is the largest traffic sensor data warehouse built so far in Southern California. Using this big traffic dataset, we have a unique opportunity to use data-driven approaches to understand the mobility and accessibility in the Los Angeles Metropolitan Area.

1.2. Constructing complete traffic dataset for the generation of synthetic trajectories

The data we have collected from Los Angeles Metropolitan Area are traffic flow data (speed and volume), not trajectories. Thus, first, we make heavy use of synthetic trajectories due to the lack of available high-resolution trajectory datasets for Los Angeles area. Given an origin, a destination, and historical traffic data of a road network, our algorithm employs a time-dependent routing algorithm to discover the shortest path from the origin to the destination. This time-dependent path is considered the generated trajectory of the vehicle. Thus, our first task is to develop an algorithm which takes the traffic conditions as input to generate realistic synthetic trajectories. However, we found that our historical traffic data are incomplete and have lots of missing values, temporally and spatially, which makes the generation of accurate synthetic trajectories hard. Thus, we needed to first work on the estimation of those missing values to make the dataset more complete.

Our experiments showed that datasets with many small-scale gaps (missing values) can be accurately imputed using statistical approaches such as linear and spline interpolation without requiring any expensive pre-processing steps. On the other hand, as missing data gaps become longer and more frequent, data-driven approaches such as historical averages provide much more accurate results. This implies that archived historical traffic flow data is highly important.

Through the project, it has been feasible to generate traffic flow data (i.e., average speed) for any given time and location in Los Angeles using both real sensed and/or estimated traffic flow data. A paper to estimate missing values was written and published. This includes algorithms to impute missing values spatially and temporally, and an algorithm to generate Traffic Index for the City of Los Angeles. Traffic Index is a spatial-temporal representation of the traffic conditions at a predefined region.

1.3. Constructing reachability map with synthetic trajectories

Our approach to generate isochrone map does not consider the complex underlying road network graph. Instead, our methodology only takes into consideration the trajectories and consists of retrieving all the reachable GPS measurements from them to construct isochrone maps. For efficiency, we used a grid-based index that can be utilized to efficiently filter and process only those trajectories that are guaranteed to meet the query criteria. Thus, unlike most conventional approaches, our approach achieves much lower response times than those of conventional graph approaches because it is not affected by the complexity of large road networks.

The challenge that we faced is the evaluation of the quality of isochrone map, which also depends on the quality (accuracy) of synthetic trajectories. However, there are few methods to evaluate the accuracy of synthetic vehicle trajectories in urban area, specifically for the City of Los Angeles. Thus, we needed to work on the evaluation of the quality of our synthetic trajectories.

We extracted real vehicle trajectories around Los Angeles from the Veraset trajectory dataset which is a large collection of real trajectories of moving objects including vehicles. We used the extracted real trajectories to evaluate the quality of our synthetic trajectories by quantifying the similarity of real and synthetic trajectories.

Subsequently, we devised an algorithm to generate isochrone map for any given time and location in Los Angeles using synthetic trajectories. We also developed a web application and tested the overall method for evaluating accessibility of Los Angeles Area. Reachability maps are presented in a variety of ways with each representation offering a different view of the result for the City of Los Angeles.

2. Traffic Data Imputation and Interpolation

2.1. Traffic Imputation

In reality, the consistency and completeness of time-series data generated by sensors is affected by many factors. Hardware malfunctions, infrastructure maintenance, and software updates are some of the examples that cause downtime, thus lead to gaps in time-series traffic data. Our analysis on historical traffic data shows that on average 15% of data is missing on a daily basis (Figure 2) due to various reasons such as sensor malfunctions, system maintenance, and data collection software problems. In this section, we explain how imputation methods can be employed to address the first challenge and discuss the advantages and disadvantages of several time-series imputation methods.

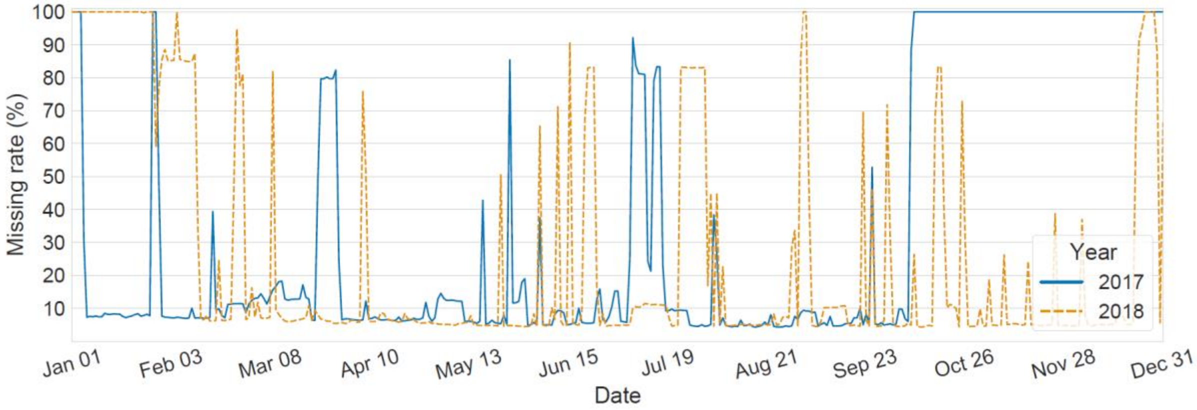


Figure 2. Daily rate of missing traffic data in the years or 2017 and 2018 in Los Angeles County.

2.1.1. Problem Definition

Let $T = (v_1, v_2, \dots, v_n)$ be the sequence of traffic speed observations over a period of n time steps. Now suppose that values v_5, v_6, v_7 are missing. The task of imputation aims to compute values $\tilde{v}_5$, $\tilde{v}_6$, $\tilde{v}_7$ that minimize the difference between the original and estimated values. The estimated values are generated by an estimator $\tilde{f}$ *(t; Θ),* where *t* is the time step to be imputed and *Θ* is an optional vector that carries additional information or context.

2.1.2. Imputation Methods

A natural way to fill in the missing values is by carrying forward the last available value; a method that is commonly known as forward-fill. Similarly, we can also carry backward the next available value. This method is commonly known as backward-fill. In an offline setting, both of these methods are expected to perform very similarly in terms of accuracy when the gap is not wide. However, in an online setting, backward-fill is infeasible as the next available value cannot be known in advance. Interpolation methods can also be used as imputation functions. Linear interpolation, for example, imputes a missing value v_i using an estimator function $\tilde{f}(x)$ that forms a straight line between the values exactly before and right after the gap.

Unlike previously discussed methods that are univariate, i.e., each time series must be independently imputed as a single variable, multivariate imputation attempts to recover all missing values at a given time step by exploiting the correlations between the underlying time series. In this setting, each time series is considered to be an independent variable and the estimator is a function that is trained to fit the available data before used to ”predict” the missing values.

Lastly, using historical traffic data, the expected behaviour (i.e., speed) of a sensor can be estimated and used to fill gaps in the data. More concretely, the average hourly speed of a sensor can be calculated from past observations. For example, if the speed was not observed at 3:45pm, the average speed of 3-4pm may be used to impute the missing observation. The day of the week may also be considered when calculating the average speed in order to better capture daily behaviours (e.g., weekday vs. weekend traffic).

2.1.3. Discussion

The missing data gaps can be divided into two categories; (i) small gaps where the duration of missing data is short enough that no significant change in the traffic dynamics might have occurred, and (ii) large gaps where the duration of missing data is long enough for the traffic dynamics to vary significantly. Figure 3 shows a real-world traffic speed time-series spanning two days, Sunday and Monday, in March in Downtown Los Angeles. We observe that the second gap is very small (only one time step) and the speed between the start and end of the gap remains similar. However, all other gaps span more time steps during which the speed varies significantly. For example, the third gap is almost 3 hours long and the speed changes from 15mph to 20mph.

Intuitively, small gaps such as the second gap in Figure 3 can be accurately imputed using simple approaches such as forward- and backward-fill. This is because, unlike other time-series, traffic flow data (e.g., speed measured by a sensor) do not vary dramatically from one time step to the next except during major incidents such as traffic accidents. However, the greedy nature of these methods fails to capture the gradual and progressive change in the traffic dynamics that occurs during large gaps and, hence, are not ideal candidates for long-term imputation which makes data-driven imputation methods more appropriate in such cases.

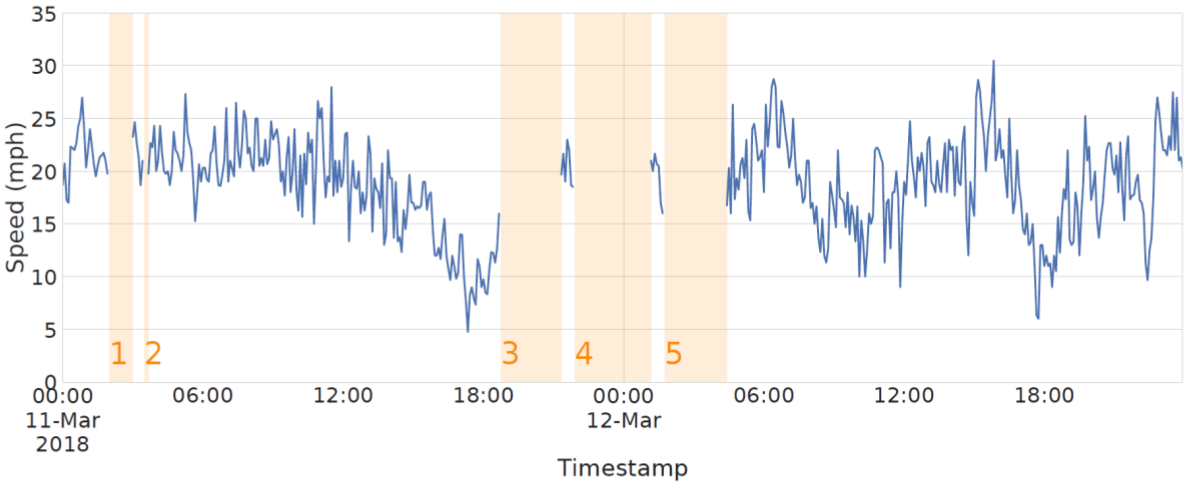


Figure 3. Real-world traffic speed measured by a sensor in Downtown LA during Mar. 2018 spanning two days (Sunday and Monday). Periods when data is missing (gaps) are highlighted.

2.2. Traffic Interpolation

Installing traffic sensors such as loop detectors all over a metropolitan city is prohibitively expensive, resulting in sparse traffic sensor data. However, it is still critical to obtain and harness a dense and accurate representation of the traffic situation for many multi-disciplinary studies. Therefore, we evaluate several interpolation methods for estimating the traffic at arbitrary locations where traffic is not directly sensed. We refer to these algorithms as Traffic Interpolation methods and in this section we describe how interpolation can be used to produce a more dense and complete view of the traffic situation at the scale of a metropolitan city.

2.2.1. Problem Definition

Let $S = (s_1, s_2, \ldots, s_k)$ be a set of k sensors that monitor traffic flow (e.g., vehicle speed) at fixed locations *s.loc*. Traffic interpolation is the problem of estimating the traffic data at a query point q by aggregating information generated by the set of sensors S and any additional context θ. Formally, we define a function $\tilde{g}$ such that $\tilde{g}(q, t; S, \theta)$ is equal to the traffic speed v that was observed at location q at time t, where t is in the past.

2.2.2. Interpolation Methods

Suppose we want to obtain an estimation of the traffic speed at an arbitrary location (road segment) q at time t. Below we explain how this can be achieved using several interpolation methods and discuss the advantages and disadvantages of each.

- k-Nearest Neighbors: The speed values at time t of the k-nearest sensors to q are averaged (arithmetic mean) and used as the speed value at location q.

- Inverse Distance Weighting: Similar to k-nearest neighbors with the key difference that a weighted average is performed instead. The weight of each sensor depends on its distance from q; the farther away the less the weight it is assigned.
- Kriging: Based on Gaussian processes, this method produces an unbiased interpolation function for estimating the values at unsampled locations. However, it is computationally intensive and hard to scale without implementing approximation methods that affect its accuracy. It is commonly used in a variety of domains, including pollution estimation, where the data are spatially related and estimates to fill in spatial gaps from actual measurements are required.

2.2.3. Discussion

The interpolation accuracy highly depends on the direction of travel at the interpolated location. It is common for the two directions of a road to be congested at different times of the day. Therefore, when interpolating the traffic at a location, sensors in the same direction have a larger impact on the calculation of the interpolated value. As we show in our experiments, when the direction of travel is considered, the interpolation accuracy increases.

2.3. Traffic Index

For urban planners and decision makers, it is critical to view and understand traffic at the Traffic Analysis Zone (TAZ) [4] or a certain region, such as census block, level. Large corporations such as Google and Apple collect high resolution traffic data from their users to support their navigation systems. However, federal and state transportation agencies do not have access to these traffic data and have to, instead, rely on traffic sensors such as loop detectors. These traffic sensors are deployed on highway and arterial roads of a metropolitan city and measure the traffic at a discrete resolution both spatially and temporally. While this resolution can be helpful when making decisions at a small scale, e.g., traffic lights change duration, it may not provide useful information at a macro level traffic flow understanding of a metropolitan city. Therefore, traffic data within a region of interest must be processed and summarized. We refer to the data structure that stores the traffic summaries as the Traffic Index.

The Traffic Index is a spatial-temporal representation of the traffic conditions at a predefined region. Spatially, the region is divided into a set of administrative districts, e.g., census blocks. Temporally, traffic values are aggregated to some predefined resolution, e.g., hourly. We use the notation $R = (r_1, r_2, \ldots, r_n)$ and $T = (t_1, t_2, \ldots, t_m)$ for regions and time steps, respectively. Our algorithm constructs the traffic index of an interested region from a real-world traffic dataset in three steps:

Imputation: All missing values of a traffic dataset D are imputed to obtain an estimated, yet complete traffic dataset D'.

Interpolation: A grid that covers R is constructed. Each cell of the grid has a square shape with size s. Then, the traffic at the center of each cell c is interpolated for all time steps.

Summarization: The traffic of each $r_i \in R$ for every $t_j \in T$ is summarized by aggregating the interpolated values of all the cells that intersect with it. Suppose that $S_i = \{c_1, c_2, \ldots, c_K\}$ is the set of K grid cells that intersect with region r_i. Then, its summarized traffic value at time t_j is calculated by $\frac{1}{K}\sum_{c_k \in S_i} \tilde{v}_k^j$, where $\tilde{v}_k^j$is the interpolated value of cell c_k at time step t_j .

2.4. Experiments

In this section we present the results of a comprehensive experimental study and discuss the trade-offs of each imputation and interpolation method.

2.4.1. Experimental Setup

All experiments were performed on an Ubuntu 18.04 server equipped with an Intel(R) Core(TM) i9-9980XE CPU at 3.00GHz and 128GB of RAM. The processor has 18 cores (36 threads) with private L1 (32KB for data and 32KB for instructions) and L2 (1MB) caches and a shared L3 (24.75MB) cache.

We use three real-world traffic speed datasets of the Downtown Los Angeles area:

- DTLA032018: 781 arterial sensors, March 2018.
- DTLA022020: 762 arterial sensors, February 2020.
- DTLA032020: 770 arterial sensors, March 2020.

In all datasets, the speed is recorded at 15-minute intervals and represents the average vehicle speed in miles per hour. In the original dataset, speed is reported every minute, however, for our experiments we use arithmetic average to estimate the speed of each interval (time step). We use Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) to evaluate the methods.

2.4.2. Imputation Experiments

In the first set of experiments, we study the performance of imputation methods. By nature, our real-world datasets contain missing values. Hence, in order to be able to test the accuracy of the imputation methods, we identify and select the five largest periods per dataset during which there are no missing data. Then, we randomly drop 10% of the values during those periods to simulate a Missing Completely at Random (MCR) situation and use the imputation methods described in Section III to recover them. We use the dropped values as the ground truth. Specifically, we compare the following imputation methods:

- Fill Forward (FFILL) & Backward (BFILL)
- Linear (LI) & Spline (SI) Interpolation
- Polynomial Interpolation (PI)
- Multivariate Imputation (MVI)
- Hourly average speed (HA H) calculated using the past one year history of each sensor (∼750GB of data)
- Hourly average speed per day of week (HA DH) calculated using the past one year history of each sensor (∼750GB of data)

- Hourly average speed per day of week (HA DH R) calculated using only the previous three months data of each sensor (~200GB of data)

Figure 5 summarizes the results. Out of all the methods, the multivariate imputation is consistently performing better than the other. An interesting observation is that using historical averages calculated by recent data instead of the entire history yields more accurate results. The intuition is that recent data better capture the ever-changing traffic dynamics of the period we want to impute.

Next, we simulate a Missing at Random (MR) scenario in the temporal dimension by dropping consecutive values on the DTLA032018 dataset. Figure 4 shows the MAE, RMSE, MAPE, and elapsed time when varying the missing period length from 1–20 time steps (15 minutes to 5 hours). The fastest methods are FFILL and BFILL, however, with significantly worse accuracy than the others. As expected, as the gaps become longer, it becomes more evident that using the data-driven methods, i.e., HA DH and HA DH R, yields much more accurate results. On the other hand, for very short gaps of up to 3–4 time steps, LI and SI yield accurate results without the need of calculating historical averages.

2.4.3. Interpolation Experiments

In the second set of experiments, we study the performance of interpolation methods. Since we cannot quantify the interpolation accuracy at arbitrary points, we instead split the sensors into two sets: train (90%) and test (10%). We use the sensors in the test set as the interpolation points and compute RMSE by comparing the interpolated to the actual values. Specifically, we compare the following interpolation methods: Nearest Neighbor (NN), Arithmetic Mean (AVG), Inverse Distance Weighting (IDW), and Kriging Method (KM).

We run experiments using two variants of each method. The first variant considers all sensors during the interpolation whereas the second variant only considers sensors that are installed on roads with the same direction as the sensor at which we are interpolating the traffic. Figure 5(a) shows the performance of the methods in terms of RMSE. Besides NN, which performs the worst, all other methods exhibit similar performance with IDW achieving slightly better accuracy than the others when direction is considered. The cause of NN's bad performance is that most of the times, the nearest sensor is the one installed on the opposite side of the road.

The intuition is that the traffic dynamics of the two sides of a street can be quite different. For example, during the morning rush hour, the streets that lead towards the downtown area are congested by people going to work whereas the streets that lead away from downtown incur moderate to low traffic (the opposite happens during the afternoon rush hour). This also explains why the interpolation shows a 3% improvement on average when direction of travel is taken into consideration. Next, we study how the accuracy of IDW is affected depending on the number of sensors that are used to interpolate the traffic. Figure 5(b) shows the results. On the x-axis we vary the number of sensors from two to twenty and on the y-axis the average distance from the query point to a sensor (left) and the RMSE (right). We observe that as the number of sensors increases the RMSE drops, i.e., the interpolation becomes more accurate.

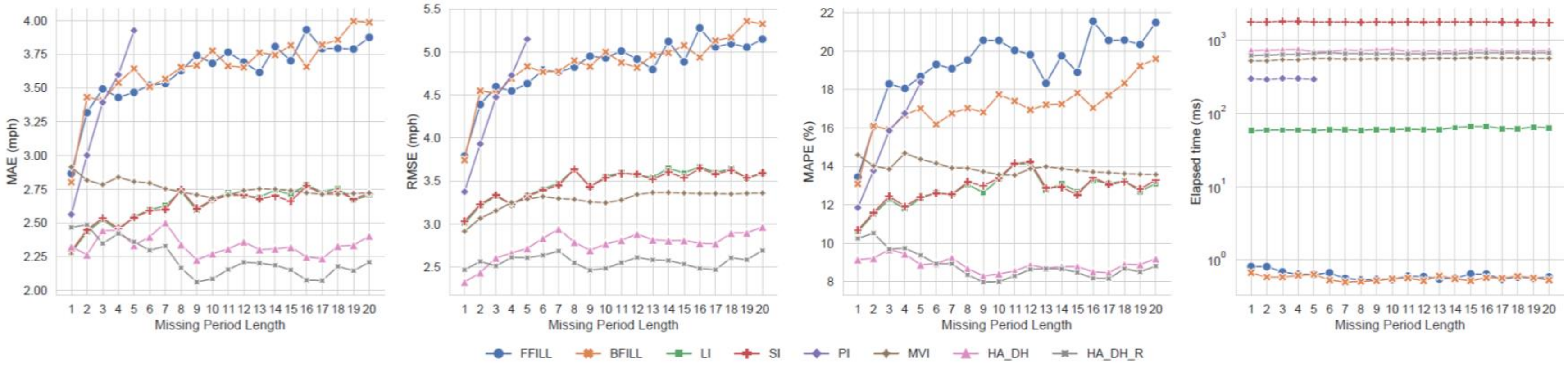
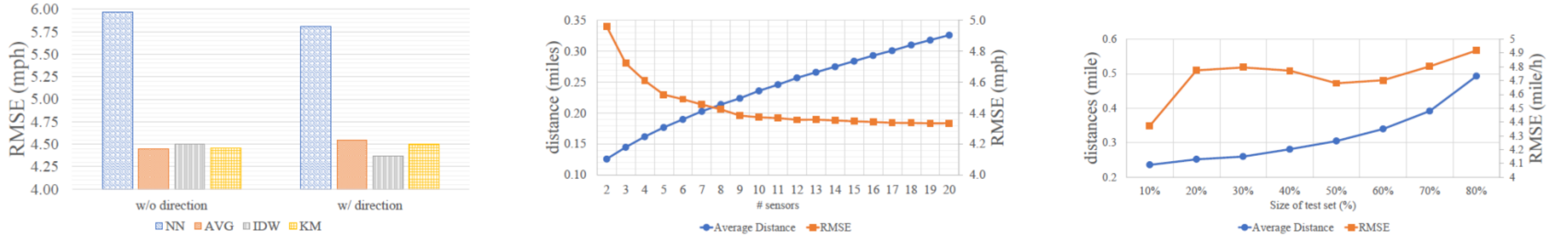


Figure 4. Imputation accuracy performance for different missing period lengths varying from 1 to 20 time steps (15 minutes to 5 hours).



(a) Accuracy of the interpolation methods with and without direction of travel.

(b) Performance of IDW when varying the number of neighbors used to interpolate traffic.

(c) Performance of IDW when varying the percentage of sensors used as part of the test set.

Figure 5. Traffic interpolation performance in terms of RMSE.

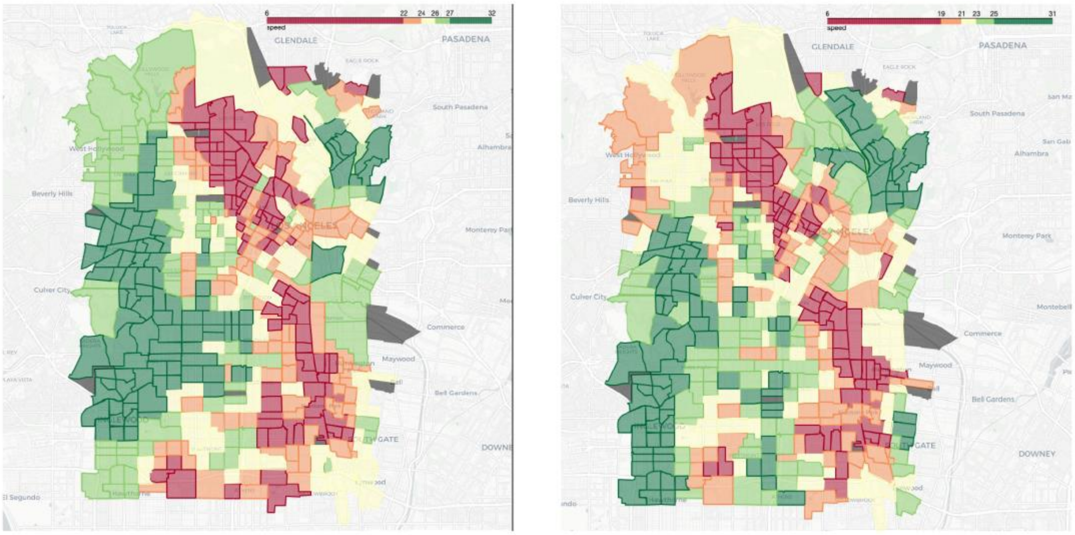
At the same time, we observe that the average distance between the query point and sensors increases which is expected. Furthermore, we observe that there is no noticeable improvement in accuracy when using more than 12 sensors.

Lastly, we study how the accuracy of IDW is affected when we vary the size of the training set. Specifically, we randomly split the dataset into train and test sets at different ratios varying from 10% to 80% for the test size. Figure 5(c) depicts the results. In the x-axis we vary the size of the test set while on the y-axis we show the average distance between sensors and the query point (left) and the RMSE (right). The accuracy is highest when the training set is largest.

2.4.4. Case Study: Los Angeles Traffic Index

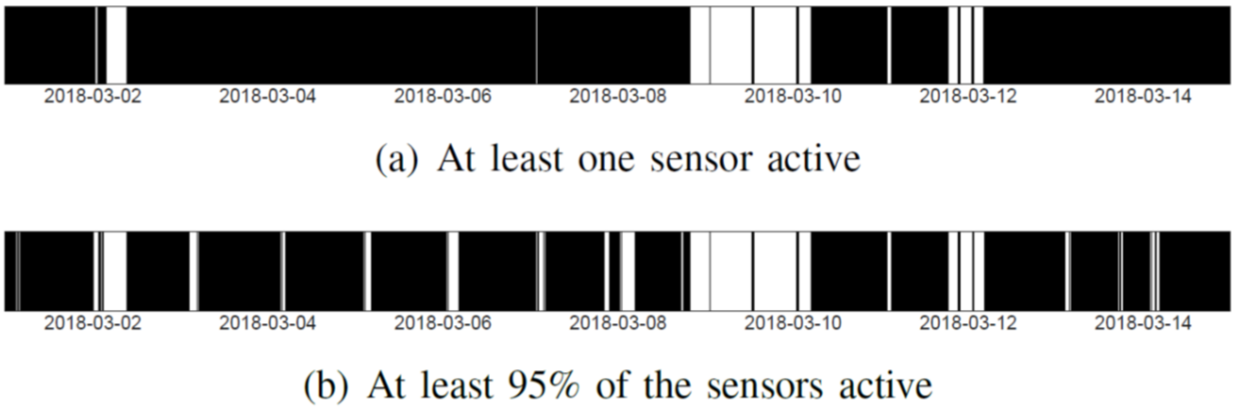
As we have already argued, the Traffic Index is critical for a variety of multi-disciplinary studies and is useful to many researchers. Our unique dataset emphasizes the real-world challenges we mention in this work and advocates the need for accurate temporal imputation and spatial interpolation methods in order to improve the performance of downstream tasks. We limit our case study to a subset of our traffic data, specifically Downtown Los Angeles in March 2018.

First, we perform an analysis to evaluate the quality of the dataset. As we previously discussed, real-world data tend to contain a variety of discontinuity, i.e., missing data occur often and at random periods. This fact can be clearly seen in Figure 7, where we plot the periods of activity (black stripes) and inactivity (white stripes) for the first two weeks of March 2018. Specifically, at the top (Figure 7(a)) time steps with at least one sensor reporting traffic are considered active whereas, at the bottom (Figure 7(b)) time steps with at least 95% of the sensors reporting traffic are considered active. A total of 201 time steps have no active sensors, i.e., 7% of the time no traffic flow is reported.



(a) Monday at 8:30am-8:45am (b) Thursday at 4pm-4:15pm

Figure 6. Traffic index aggregated at the level of TAZ for Downtown Los Angeles in March 2018.



(a) At least one sensor active

(b) At least 95% of the sensors active

Figure 7. Periods of missing data (white stripes) in a real-world traffic dataset at Downtown Los Angeles in the first two weeks of March 2018.

In order to construct the Traffic Index, we first impute the missing values using multi-variate imputation for short gaps up to four time steps (1 hour) and the historical average for larger gaps (e.g., between March 8th and March 10th). Then, we construct our interpolation grid with a cell size of 0.004 coordinate degrees (approximately 400 meters). Our grid contains a total of 250 interpolation points covering Downtown Los Angeles and its surrounding areas. The traffic at every grid cell is interpolated with IDW. Lastly, we summarize the traffic at the level of Traffic Analysis Zones.

The algorithm that constructs the Traffic Index incurs approximately 60 seconds of CPU time per time step, i.e., it takes approximately 24 milliseconds per cell for our 50x50 grid. In general, the time complexity of the algorithm is O(|T | × |G| × p), where |T | is the number of time steps, |G| the number of grid cells, and p is the complexity of the interpolation function that is used. In the case of IDW and using an R-tree to index the sensors, *p = O(n· logN)*, where *n* is the number of nearest neighbors to use and *N* the total number of sensors.

Figure 6 shows two instances of our Traffic Index at different time steps. We color code the zones based on how congested they are varying from heavy traffic (dark red) to no traffic (dark green). On the left (Figure 6(a)) shows the traffic index at a Monday morning whereas on the middle (Figure 6(b)) the traffic index on a Thursday afternoon. Both instances show the traffic during rush hours.

3. Generating Synthetic Vehicle Trajectories

3.1. Overview

Big Trajectory Data provides limitless opportunities for understanding human mobility, human interactions [5], traffic congestion patterns [6], improving routing and planning [7], and more. At the same time, it also introduces challenges in protecting the privacy of individuals [8] [9] making them hard and costly to obtain. On one hand, even with their increasing availability, most publicly available trajectory datasets can only cover a portion of the population and are limited to crowded regions and active hours of the day, hence leading to biased observations.

On the other hand, proprietary trajectory datasets are more comprehensive but their data holders such as Google, Apple, and other private entities (e.g., private truck drivers) are not willing to share.

In order to address these challenges, synthetic vehicle trajectory generation approaches have been proposed. The task of synthetic trajectory generation is useful when either the input set is small and we need more realistic trajectories for the downstream task (scale-up) or when the privacy of the input set must be preserved (privacy preservation). One can also use this methodology to generate trajectories for one geographical area (e.g., city, neighborhood) from the trajectories belonging to a different geographical area (diversification).

We divide the synthetic trajectory generators into two main categories: (i) *Computational*, and (ii) *Data-Driven*. The taxonomy of these generators is shown in Figure 8. Computational generators typically process information about the environment to generate vehicle trajectories. We further classify these into *Simulators* and *Moving-object Generators*. Approaches in the latter class generate network based moving objects using demand-supply models [10]. They used to be state-of-the-art back when they were originally proposed but are now outdated. Approaches that fall under the former class simulate the actions of agents (e.g., vehicles) in a closed environment [11]. Every action updates the state (i.e., the location) of the agent and their history represents a synthetic trajectory. In order to generate realistic vehicle trajectories, computational generators require a fully calibrated environment. However, acquiring and fine-tuning all their parameters is a challenging task [12].

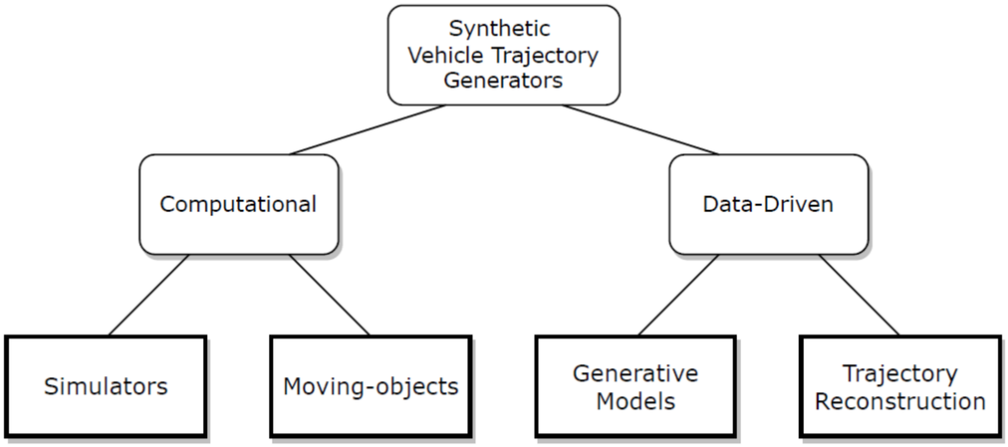


Figure 8. Taxonomy of synthetic vehicle trajectory generators.

Data-driven generators leverage real-world experiences (data) to synthesize trajectories. The current state-of-the-art in this category is the Generative Adversarial Networks (GAN). These deep learning models train on existing vehicle trajectory data and learn a generator model that synthesizes realistic vehicle trajectories given some noise as input [13]. Although GANs do not require the complex calibration of the computational generators, they need large amounts of data in order to be trained and, even with their increasing availability, most publicly available trajectory datasets can only cover a portion of the population and are limited to crowded regions and active hours of the day. As a result of this data skew, training leads to biased generative models. Another issue is that existing methods use a limited number of metrics that can only evaluate the holistic statistics of the generated dataset, ignoring how realistic each individual trajectory is on its own.

In this study, we propose a data-driven, model-free, and parameter-less method of generating synthetic yet realistic trajectory datasets using only two inputs: (i) traffic data for the region of interest, and (ii) a target Origin-Destination matrix. Unlike the computational methods, our *Data-Driven Trajectory Generator*, dubbed *DDTG*, does not require any kind of calibration before it starts generating trajectories and unlike GANs it does not require an existing dataset to train on. In fact, the input to *DDTG* is aggregated data that is publicly available and free of privacy concerns [14] [15] (aggregate data can also be released under stronger privacy guarantees [16] [17]). Additionally, the datasets that are generated by *DDTG* can be used as seed data for training generative models in lieu of real trajectory datasets which makes *DDTG* orthogonal and complementary to GANs. In fact, as we show in our experiments, when real-world and *DDTG*-generated trajectories are combined as training data, the trained model generates higher quality synthetic trajectories. *DDTG* disaggregates traffic and OD matrix data into individual trajectories, such that the collection of the generated trajectories results in the same aggregate data. The main challenge is that these aggregate data are time-dependent and hence the number and shape of individual trajectories differ depending on the time of the day. The main intuition behind *DDTG* is that individuals tend to select the fastest (or at least one of the fastest) path to their destination [10]. Thus, *DDTG*, in contrast to other methods, is only suitable for generating urban vehicle trajectories but, as we show in our experiments, it is capable of generating synthetic datasets that retain the underlying distributions of a real vehicle trajectory dataset more accurately.

3.2. Data Representation

We define a vehicle trajectory $s = (s^{(1)}, s^{(2)}, \dots ,\ s^{(M)}\ )$ as a time-ordered sequence of spatio-temporal points. Each point $s(m)$ consists of a pair of spatial coordinates (i.e., latitude, longitude) and its timestamp. A synthetic trajectory generator outputs a set $S = \{s_1, s_2, \dots, s_n\}$ of n trajectories that preserve the mobility characteristics of the real world.

Vehicle trajectories are by definition constrained to a road network. Because raw vehicle trajectories are very often noisy due to errors in localization sensors (e.g., GPS), before a trajectory dataset *S* is analyzed it is typically pre-processed so that every $s\ \in S$ is map matched to a road network *G* and converted into a more interpretable trajectory $\tilde{s}$. The transformed trajectory becomes a time-ordered sequence $\tilde{s} = \{\tilde{s}^{(1)}, \tilde{s}^{(2)}, \dots, \tilde{s}^{(P)}\}$ where every $\tilde{s}^{(p)}$ consists of the graph edge (i.e., road segment) and the timestamp at which the vehicle arrives at the edge. In essence, the raw trajectory is converted to the corresponding path that the vehicle took to reach its destination.

Unlike the computational methods that require calibration, and unlike GANs that require large amounts of existing trajectory data, DDTG only needs aggregated traffic data and an OD matrix for the region of interest along with the road network. Fortunately, this kind of aggregated data is free of any privacy concerns and very often publicly available as we provide examples below.

Road Network: Encoded as a graph $G = (V,E)$ with every node $v \in V$ representing an intersection and every edge $e \in E$ a road segment. Detailed public datasets are made available for most cities in the world by OpenStreetMap [18].

Traffic Data: Traffic sensors (e.g., loop detectors) are deployed on the road network and measure the average speed of passing vehicles at fixed time intervals. A traffic data point is a tuple $\rho = (l, t, v)$, where l are the spatial coordinates (i.e., latitude and longitude) of the sensor, t the timestamp, and v the speed [1] [14].

Origin-Destination Matrix: An $M \in R^{CxC}$ matrix that corresponds to a grid with $|C|$ cells overlayed over the region of interest. For each pair of cells (c_i, c_j) the OD matrix measures the probability that a trip originates within cell c_i and terminates within c_j. Such data are made available by some providers [19] but can also be estimated either from traffic [20] or trajectory data.

3.3. Algorithm

DDTG operates in two phases. In the first phase, n trip definitions are sampled from a provided OD matrix. A trip definition is a tuple (o, d, t), where o and d are the origin and destination coordinates, respectively, and t is the departure time. Subsequently, in the second phase, the trip definitions are processed and corresponding synthetic trajectories are generated. Every generated trajectory is essentially a path constrained to a road network G and enriched with the travel time on each road segment by using historical traffic data C. We elaborate on each phase in turn.

3.3.1. Phase I

DDTG starts by sampling n trip definitions from an OD matrix. The pseudocode is shown in Algorithm 1. The distribution specified by M is used to sample a pair of cells (c_i, c_j) (line 4). Subsequently, the exact origin and destination coordinates are randomly sampled within the origin and destination cells (lines 5-6), respectively. Next, a departure time is sampled. This departure time must adhere to the real world distribution, i.e., a noon departure time is more probable than a 2AM departure time. We capture the distribution as a histogram with bin size φ minutes and we sample departure times from it.

The first phase of *DDTG* plays a critical role in generating datasets of high holistic quality. Specifically, because the origins, destinations, and departure times are sampled from real-world distributions, the more trajectories that *DDTG* generates, the closer it imitates these distributions. In the previous section, we show how a bad choice of M (e.g., uniform) affects the holistic quality of the generated dataset.

Algorithm 1 Trip Sampler

```
Require: OD origin-destination matrix, $\mathcal{T}$ time domain
procedure GENERATETRIPDEFINITIONS($n$)
    $S \longleftarrow \{\}$
    for $1 ... n$ do
        $(c_i, c_j) \longleftarrow Sample\ (OD)$
        $O \longleftarrow PickLocation(c_i)$
        $D \longleftarrow PickLocation(c_j)$
        $T \longleftarrow Sample(\mathcal{T})$
        $S \longleftarrow S \cup \{(O, D, T)\}$
    end for
    return $S$
end procedure
```

3.3.2. Phase II

After all the trip definitions are generated, the algorithm proceeds to generate corresponding synthetic trajectories. The pseudocode of the second phase is depicted in Algorithm 2. Because *DDTG* directly operates on the road network, the first step is to match the origin and destination locations to their respective closest nodes *u* and *v* in *G* (lines 2-3). One challenge is what path to select as a trajectory. Intuitively, individuals exhibit a strong preference for taking a fast (typically the fastest but not always) path to their destination [10] instead of any other kind of path (e.g., shortest, random walk). However, always using the fastest as a trajectory leads to unrealistic datasets where all objects use the same path. To introduce the diversity of choices and personal preferences, a number of feasible paths from *u* to *v* is calculated (line 4). Specifically, the algorithm searches for the k time-dependent fastest paths [21] that have limited overlaps. This is achieved by using a variant of the *k*-alternative shortest paths algorithm [22] in which the cost function of the route becomes the travel time instead of the travel distance. Finally, one of the *k* paths is chosen as the actual trajectory (line 5).

Algorithm 2 Trajectory Generator

```
Require: Road Network $G$, Traffic Data $C$, $k$
  1: procedure GENERATETRAJECTORY($O, D, T$)
  2:     $s \longleftarrow Lookup(G, O)$                ▷ $Source\ node$
  3:     $t \longleftarrow Lookup(G, D)$                ▷ $Target\ node$
  4:     $P \longleftarrow FindFastestAlt\ (G, C, s, t, T, k)$        ▷
  k -Alternative fastest paths from s to t departing at T
  5:     $p \longleftarrow ChooseMostLikely(P)$
  6:     return $p$
7: end procedure
```

The second phase of *DDTG* generates trajectories of high individual quality. Using real-world traffic data, the travel times of the generated trajectories are accurately estimated. This is also confirmed by our experiments (Figure 9).

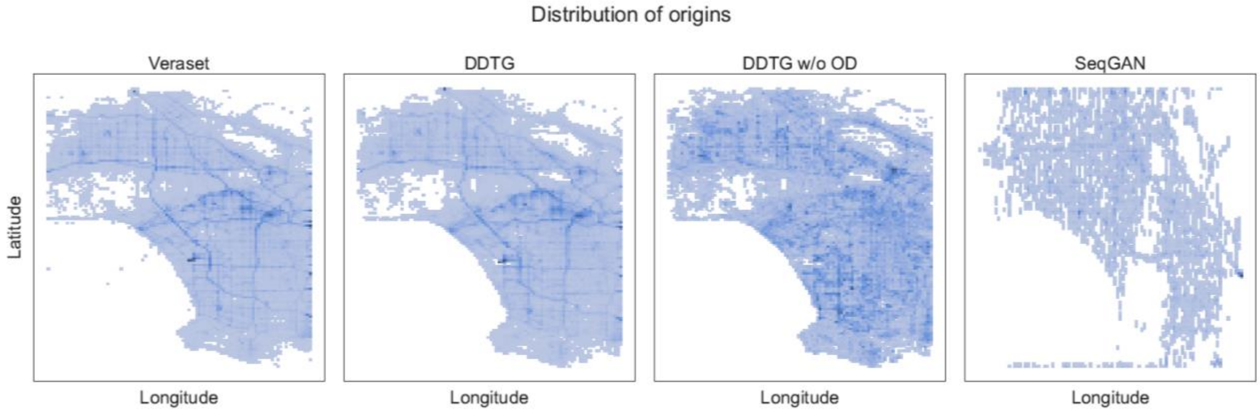


Figure 9. Distribution of origins in the real dataset (left) and the synthetic datasets generated by DDTG (center-left), DDTG without OD matrix information (center-right), and SeqGAN (right). DDTG imitates the real distribution with a higher accuracy.

Trajectories in the real-world exhibit a variety of mobility behaviors. For example, delivery trucks very often begin and end their trips at the same location (e.g., post office, warehouse), and their path is a round trip with many quick stops. One challenge with Algorithm 1 is that it assumes that individuals always travel with a purpose, i.e., going directly from one location to another with no stops or detours. Although the focus of this algorithm is on generating this specific type of urban vehicle trajectories, a generalized variant of Algorithm 1 can be used to generate delivery truck-like trajectories. In this generalized variant, trajectories are viewed as a sequence of smaller sub-trajectories $s_1, s_2, \dots, s_w$ in which the destination of each s_i, where $1 < i < w$, is the origin of s_{i+1}. Trip definitions are also redefined to a tuple $(\vec{L}, t, w)$ such that $\vec{L}$ is a sequence of l locations with $\overrightarrow{L_1}$ being the origin and $\overrightarrow{L_l}$ the final destination, *t* the trip departure time, and *w* a sequence of $l - 1$ stay durations at each intermittent location along the trip. Algorithm 3 presents the modified trajectory generator. The algorithm iterates over the trip locations (line 3) and generates one trajectory per each consecutive pair of locations (lines 4-6). After every generated trajectory, the departure time for the next trajectory is computed by adding the travel time and stay duration to the current trajectory's departure time (line 8). Using this variant, it becomes straightforward to model delivery truck trajectories; $\overrightarrow{L_1}$ and $\overrightarrow{L_l}$ will typically be the warehouse and $\overrightarrow{L_2}, \dots, \overrightarrow{L_{l-1}}$ will be the delivery locations.

Algorithm 3 Trajectory Via Stops Generator

```
procedure GENERATEVIATRAJECTORY($\vec{L}, T, d$)
    $\tilde{T} \leftarrow T$
    for $i = 2 \ldots |\vec{L}|$ do
        $O \leftarrow \vec{L}_{i-1}$
        $D \leftarrow \vec{L}_{i}$
        $p \leftarrow GenerateTrajectory(O, D, \tilde{T})$
        yield $p$
        $\tilde{T} \leftarrow \tilde{T} + p.travel\_time + d_{i-1}$ ▷ $Departure\ time\ for\ next\ trajectory$
    end for
end procedure
```

3.4. Quality of Synthetic Trajectories

Evaluating the quality of synthetic datasets and how realistic they are is a critical task. On one hand, the generated dataset may be realistic at a high level (e.g., preserving the origin destination matrix) while the individual trajectories are not reasonable (e.g., unrealistic travel time). On the other hand, the individual trajectories may be realistic while the holistic dataset is not. Therefore, a synthetic dataset must be evaluated at two levels: the holistic level and the individual level. In this section, we describe a set of metrics that can be used for each evaluation level.

3.4.1. Holistic Quality

At the dataset level, a synthetic trajectory dataset must retain the collective mobility characteristics of the metropolitan city. Such characteristics include, but are not limited to, the distribution of origins and destinations, the distribution of origin-destination pairs, the distribution of departure times, and the congestion density. Calculating these in a continuous space is computationally intractable, hence, the distributions are estimated by discretizing the space (and time) into an *RxC* grid with cell size *m*. The finer the grid, the closer the estimation is to the real distribution but the more computationally expensive.

We summarize these metrics below.

- **Distribution of Origins** (*p(o)*): The spatial distribution of the locations at which the trajectories begin. This is typically the first location of a trajectory.
- **Distribution of Destinations** (*p(d)*): The spatial distribution of the locations at which the trajectories end. This is typically the last location of a trajectory and implicitly represents the popularity of locations.
- **Origin-Destination (OD) Matrix** (*p(o, d)*): The distribution of the OD pairs *< O,D >*, where *O* is the origin of the trip and *D* is the destination. This captures the probability of visiting a region (i.e., grid cell) from another.

- **Transition Matrix** (*p(*c_i | c_j*)*): The probability of transitioning from cell c_i to cell c_j. Trajectories can be translated from a sequence of locations to a sequence of grid cells $\tilde{C} = \{\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_p\}$. Then, the pairs of consecutive cells in a trajectory can be used to estimate the transition matrix.
- **Distribution of Departures** (*p(τ)*): The temporal distribution of the timestamps at which the trajectories begin. This is typically the timestamp of the first location of a trajectory.

These distributions are estimated for both the real and generated datasets and compared using Jensen-Shannon distance, a well-known distance metric for probability distributions. The Jensen-Shannon distance d_{JS} is defined as follows:

$$d_{JS}(P,Q) = \sqrt{JS(P||Q)} = \sqrt{\frac{KL(P||M)+KL(M||Q)}{2}} \tag{1}$$

where $JS(P||Q)$ is the Jensen-Shannon divergence, $KL(P||Q)$ is the Kullback-Leibler divergence (Equation 2), and $M = \frac{P+Q}{2}$.

$$KL(P||Q) = \sum_i p_i \log \frac{p_i}{q_i} \tag{2}$$

The value of d_{JS} ranges from 0 to 1, where d_{JS} = 0 when the two distributions are identical and d_{JS} = 1 when they are completely different. The Jensen-Shannon divergence *JS*, and as a result the distance d_{JS}, is symmetrical, i.e., *JS(P||Q) = JS(Q||P)*.

When the number of trajectories in the two datasets is equal, the respective histograms may be compared (instead of the distributions) using Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE).

3.4.2. Individual Quality

At the individual level, the trajectories themselves must be examined for their quality. For example, the travel time of a synthetic trajectory must be within an acceptable range of the actual travel time. Additionally, the path from the origin to the destination must be reasonable, i.e., not straying far away from the destination or driving in loops. We summarize these metrics below.

- **Travel time**: The generated trajectory must be consistent with the traffic dynamics. Not only the overall travel time but also the travel time of each segment of the trajectory must be accurate. This can be verified using traffic data.
- **Deviation**: The generated trajectory should be mostly en route to the destination, i.e., it should not be taking detours or heading in a direction that is opposite to the destination. One way to evaluate this is by using Markov Models to check whether the transitions from one location to the next make sense. Another way when a reference dataset exists is to retrieve real trajectories that are similar (i.e., identical origin and destination, similar departure time) and calculate the average distance. A low value indicates that the generated trajectory does not differ much from a typical trajectory. A

large value indicates that the generated trajectory follows a path that is not common (or popular).

3.5. Experiments

In this section, we quantitatively and qualitatively evaluate synthetic trajectories generated by *DDTG* with experiments on real-world data.

3.5.1. Environment Setup

All experiments were performed on an Ubuntu 18.04 server equipped with an Intel(R) Core(TM) i9-9980XE CPU at 3.00GHz and 128GB of RAM. The processor has 18 cores (36 threads) with private L1 (32KB for data and 32KB for instructions) and L2 (1MB) caches and a shared L3 (24.75MB) cache. The server is also equipped with a GeForce RTX 2080 Ti GPU card.

Datasets:

We limit our study to the metropolitan region of the City of Los Angeles.

(1) We make use of a real-world trajectory dataset from Veraset as the ground truth for evaluation. The dataset contains tens of millions of trajectories out of which approximately 4 million trajectories are within our region of interest. However, these are raw trajectories that exhibit multiple behaviors. Every device that participates in this dataset records continuous traces, hence, we pre-process each trace to extract vehicle trajectories, i.e., when the device was actively traveling. The first step of the pre-processing process is to split the trajectories when two consecutive GPS readings are more than 5 minutes or 1500 meters apart. Next, we run a stay point detection algorithm [23] on each sub-trajectory to identify GPS readings that belong to a stay point and during which the device was not traveling. The resulting set contains 1.5 million trajectories spanning the first two weeks of December 2019.

(2) We use a real-world historical traffic dataset exported from ADMS [1]. The dataset contains approximately 16,000 sensors covering more than 4,300 miles in the region of interest. The sensors have a refresh rate of up to 60 seconds but we aggregate the traffic speed data into 15-minute intervals. Furthermore, for every road segment, i.e., edge in the road network graph, we calculate its time-dependent travel time by querying the traffic data for the speed. If traffic speed is not sensed at a particular road segment, we obtain an estimation of the traffic at that segment using interpolation methods [24].

(3) The origin-destination matrix *p(o, d)* for the region of interest is calculated from the trajectory dataset. We use a grid with a cell size of 500 meters. The cell that contains the first GPS reading of a trajectory is considered the origin. Similarly, the cell that contains the last GPS reading of the trajectory is considered the destination. The OD matrix measures the probability that a trajectory originates from a cell $o = c_i$ having cell $d = c_j$ as the destination.

Compared Approaches: We compare the real-world trajectory dataset from Veraset with 3 synthetic datasets.

- *SeqGAN*: A synthetic dataset of 1.5 million trajectories generated by a sequential GAN model [25]. The real trajectory dataset is used as training data for the model.
- *DDTG w/o OD*: A synthetic dataset of 1.5 million trajectories generated by DDTG without origin-destination distribution information, i.e., all origin-destination pairs have equal probability.
- *DDTG*: A synthetic dataset of 1.5 million trajectories generated by DDTG using an origin-destination matrix as input that was extracted from the Veraset dataset.

3.5.2. Quality Analysis

We evaluate the quality of synthetic datasets at two levels using the metrics described in the previous section.

3.5.2.1. Holistic Quality

We calculate the holistic quality distributions for both the real dataset and the synthetic dataset and then use K-L divergence to evaluate how similar they are. Figure 9 shows the distribution of origins in the four datasets as a function of the spatial coordinates (i.e., longitude on the x-axis, latitude on the y-axis). We immediately observe that the distribution for *DDTG w/o OD* information is almost uniform. This is expected because the algorithm samples origins and destinations uniformly at random as it does not have any external information. *SeqGAN* is able to capture some of the hotspots but does not come close to the real distribution. The distribution of *DDTG*'s origins is very close to Veraset's with some slightly different hot spots. The *JS* distances between Veraset's origins and synthetic origins are 0.1629, 0.3466, and 0.9299 for *DDTG, DDTG w/o OD*, and *SeqGAN*, respectively. Note that the distance between the real and *DDTG* distributions is very close to 0. This means that the two distributions are very similar. In contrast, the distance of the *DDTG w/o OD* is two times larger meaning that it is significantly different from the ground truth.

Next, we compare the destination distributions of the datasets. These distributions are shown in Figure 10 as functions of the spatial coordinates (i.e., longitude on x-axis, latitude on y-axis). Similar to the origin distribution, *DDTG w/o OD* is almost uniform as expected, and while *SeqGAN* captures some of the hotspots it only generates a very sparse distribution. Figure 10 shows that *Veraset* and *DDTG* destination distributions only have slight differences. In *Veraset*, we can see a few hot spots close to the region's eastern border, whereas, in DDTG, not as much. This is mostly attributed to the cleaning method that was employed for the real dataset, i.e., trajectories that had a destination outside the region of interest were cut short and their destination was assumed to be at the boundary. The JS distances are 0.1326, 0.3292, and 0.9086 for *DDTG*, *DDTG w/o OD*, and *SeqGAN*, respectively. Even though there are minor differences in the destination hot spots of *Veraset* and *DDTG*, the majority of the remaining grid cells appear to incur similar probabilities of being the destination.

An interesting observation is that in the real (*Veraset*) and *SeqGAN* datasets, some origins and destinations appear to exist in the middle of the ocean (bottom left sub-region), whereas, in both synthetic datasets no such cases exist. In the case of the real dataset, this is due to the noise that GPS sensors naturally incur. In the case of *SeqGAN*, this happens because the model has no information about the road network.

Additionally, we note that *SeqGAN* generates very sparse origin and destination distributions. In essence, the model learns the skewed distribution of the input training dataset, i.e., trips concentrated in certain regions or times of the data. As a result, the model is biased toward generating trips in those areas instead of the entire region.

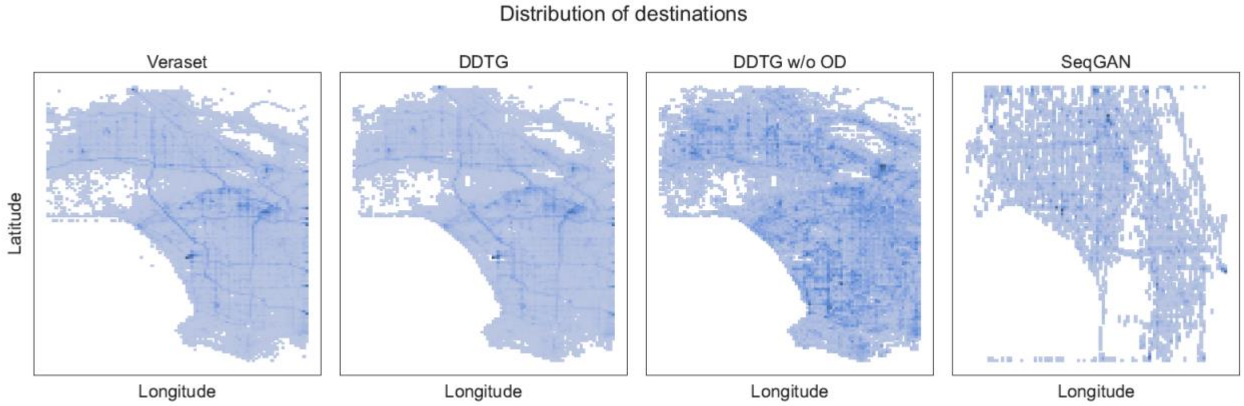


Figure 10. Distribution of destinations in the real dataset (left) and the synthetic datasets generated by DDTG (center-left), DDTG without OD matrix information (center-right), and SeqGAN (right). DDTG imitates the real distribution with a higher accuracy.

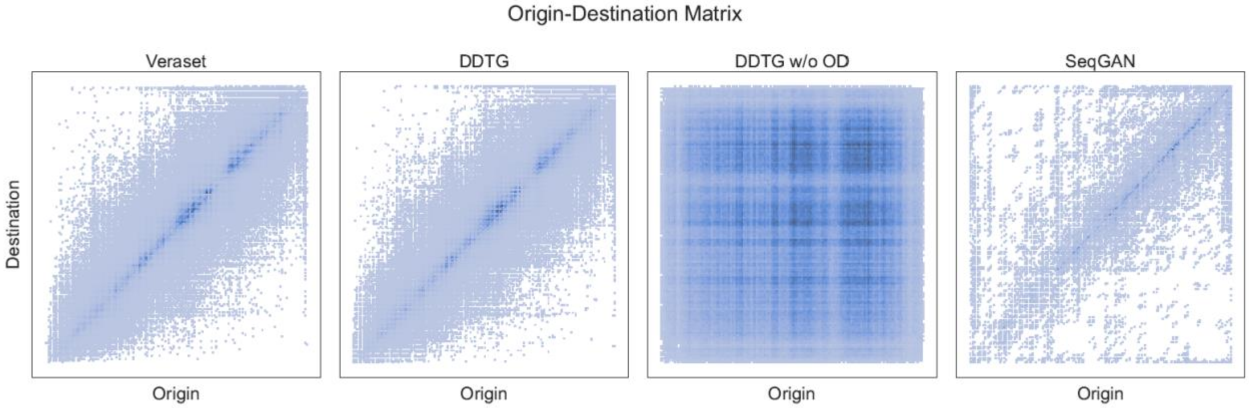


Figure 11. Origin-destination matrix calculated from the real dataset (left) and the synthetic datasets generated by DDTG (center-left), DDTG without OD matrix information (center-right), and SeqGAN (right). DDTG imitates the real distribution with a higher accuracy.

We also compare the distributions of origin-destination pairs. Figure 11 visualizes the OD Matrix of the four datasets. On the x-axis is the origin and on the y-axis is the destination. Again, as expected, the *DDTG w/o OD* information produces almost uniform results. *SeqGAN* achieves a visually similar distribution but with a great amount of noise. *DDTG*, on the other hand, imitates the true distribution almost perfectly. Some OD pairs that appear more prominently in the ground truth, are slightly smoothed out by *DDTG* due to the generator's randomness. However, the general distribution is preserved. The JS distance between Veraset's OD Matrix and *DDTG*'s is 0.3223 which, considering the vast amount of OD pairs, is close to zero. Table 1 summarizes the Jensen-Shannon distances for all datasets.

Table 1. Jensen-Shannon Distances

Distribution	**Jensen-Shannon distance d_{JS}**		
	DDTG	DDTG w/o OD	SeqGAN
$p(o)$	**0.1629**	0.3466	0.9299
$p(d)$	**0.1326**	0.3292	0.9086
$p(o,d)$	**0.3223**	0.8881	0.9356
$p(c_j \vert c_i)$	**0.2348**	0.3084	0.5928
$p(\tau)$	**0.0158**	0.2609	-

As we show in Table 1, *DDTG* generates trajectories that retain the transition matrix more accurately than the others. Specifically, the JS distance of *DDTG*'s and the real transition matrix is 0.2348, whereas *SeqGAN*'s is almost three times greater at 0.5928. This happens because *DDTG* is able to leverage the road network and generate high-resolution trajectories that retain the properties and constraints of the road network. On the other hand, trajectories generated by *SeqGAN* are more likely to contain noise, i.e., jump from one cell to another while skipping cells in between.

Lastly, we compare the distributions of departure times. Figure 12 plots the probability (y-axis) of each departure time (x-axis). Not surprisingly, the distribution for *DDTG w/o OD* is uniform. On the other hand, *DDTG*'s distribution is almost identical to that of the real dataset. These results are confirmed by the JS distance values (Table 1). We omit the departure times for *SeqGAN* because the model is highly discretized in the temporal domain.

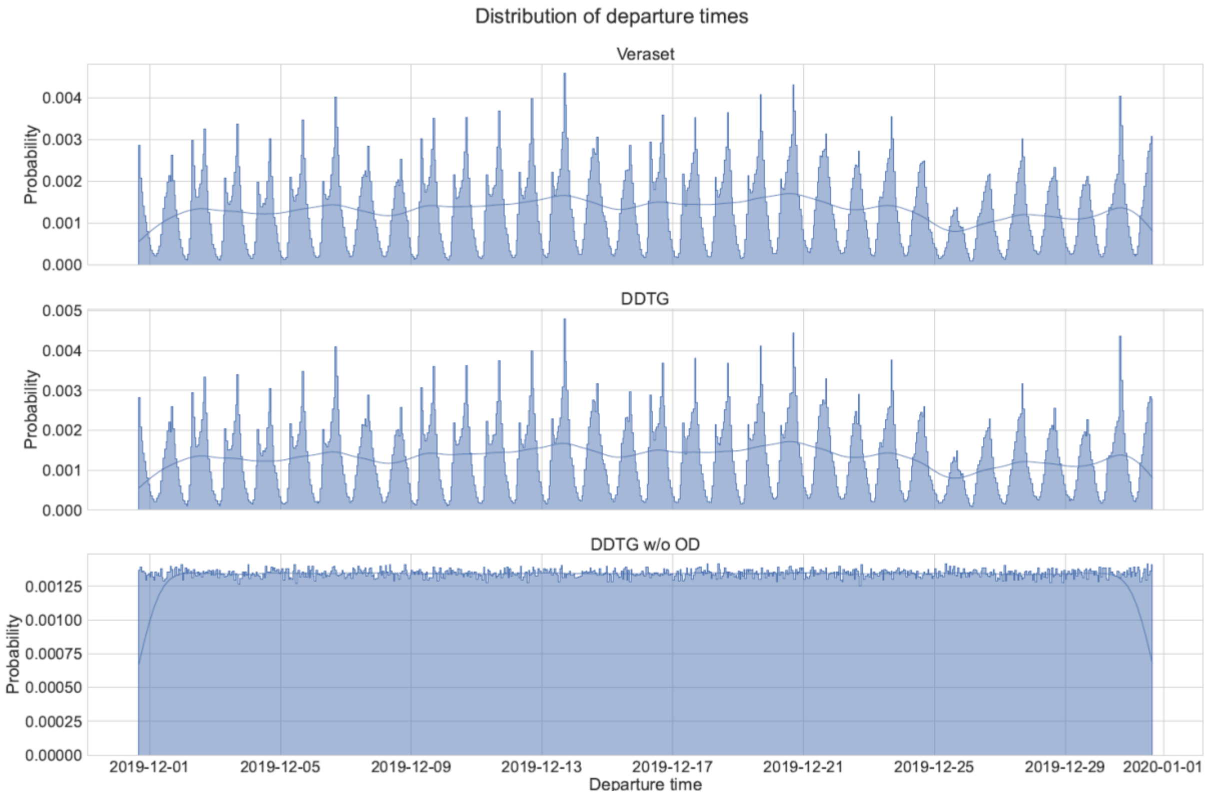


Figure 12. Departure time distributions calculated from the real dataset (top), the synthetic dataset generated by DDTG (middle), and the synthetic dataset generated by DDTG without OD matrix information (bottom).

Holistically, the dataset generated by DDTG retains the mobility characteristics of the real dataset.

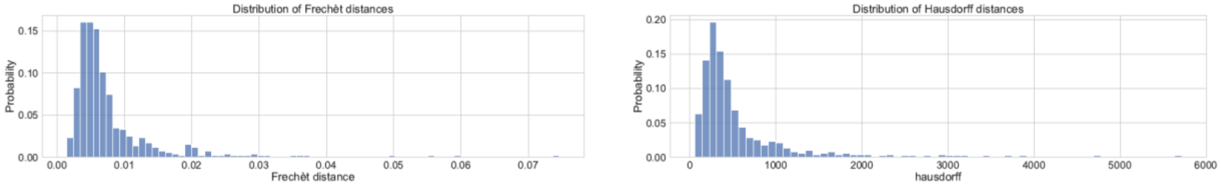


Figure 13. Distribution of Frechet (left) and Hausdorff (right) distances between generated and corresponding real trajectories.

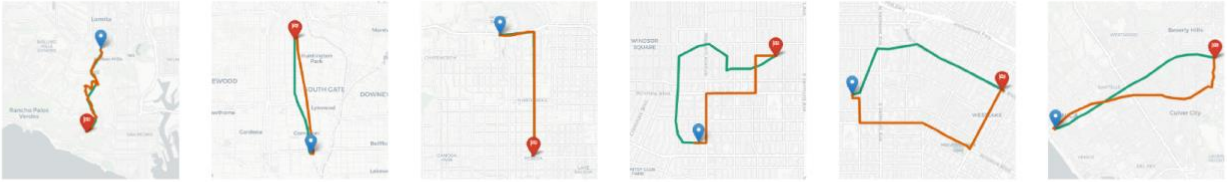


Figure 14. Examples of real trajectories (green lines) and corresponding synthetic trajectories (red lines) generated by DDTG.

3.5.2.2. Individual Quality

We verify how realistic each individual trajectory is by comparing how much it resembles common paths from origins to destinations. Specifically, we randomly sample 50,000 synthetic trajectories generated by *DDTG* and for each trajectory, we retrieve all similar real trajectories, i.e., trajectories with identical origin and destination cells and similar departure times. Every synthetic trajectory matched with 10-15 real trajectories. Next, the average distance between a synthetic trajectory and its corresponding real trajectories is calculated. For our experiments we use two distance metrics: the Frechet distance and the Hausdorff distance, two common trajectory dissimilarity metrics between curves [46]. Figure 13 presents the histograms of these distances. The lower the distance, the closer the synthetic trajectory is to the common real trajectories. For both distance metrics, the histogram is skewed closer to 0. This means that the generated trajectories are reasonably similar to the paths that individuals tend to follow. More critically, the generated trajectories are not exactly identical to the real paths. This shows that, while the individual trajectories make sense, they still protect the privacy of individuals.

Figure 14 presents 6 examples of real trajectories (green lines) and their corresponding synthetic trajectories (red lines). Origins are marked with blue pins and destinations with red pins. In the first three examples, the generated trajectory is very similar to the real trajectory. This happens because the real path taken by the individual was one of the fastest (if not the fastest) and *DDTG* itself uses that information to generate trajectories. In the last three examples, the generated paths are quite different. This can happen either because *DDTG* chose a different fastest path to the destination or because the real path itself was a detour and not one of the fastest paths.

4. Implementation of Web Application

For multi-disciplinary studies of traffic and socio- economic analysis, it is essential to obtain a high-level summarization of the traffic condition at different regions of an urban area. Traffic Index summarizes the traffic at the level of census blocks (or other zone types) and at separate times of the day. In this work, we proposed an algorithm that aggregates the traffic data of a large region and constructs a Traffic Index that can estimate the traffic at any given location in that region. This project started with the initial goal of building a platform to highlight these algorithm's results both for a census block and for a specific point on online map.

The number of components in the application soon increased with the introduction of the Reachability Map. A Reachability Map is defined as a polygon drawn on a map that defines an area of reachability within a defined time buffer. Their construction is an important task in many application domains. As an example, in urban planning, reachability maps are essential when assessing the placement of public services like hospitals and fire departments.

To get this Reachability Map many synthetic trajectories are generated based on time and location within a defined time buffer and finally, a polygon is constructed that circumscribes all of them.

The application has been developed using React for the frontend and Django for the backend connected via restful services. This report gives a scrupulous walkthrough of all its features along with areas of improvement.

4.1. Features of Web

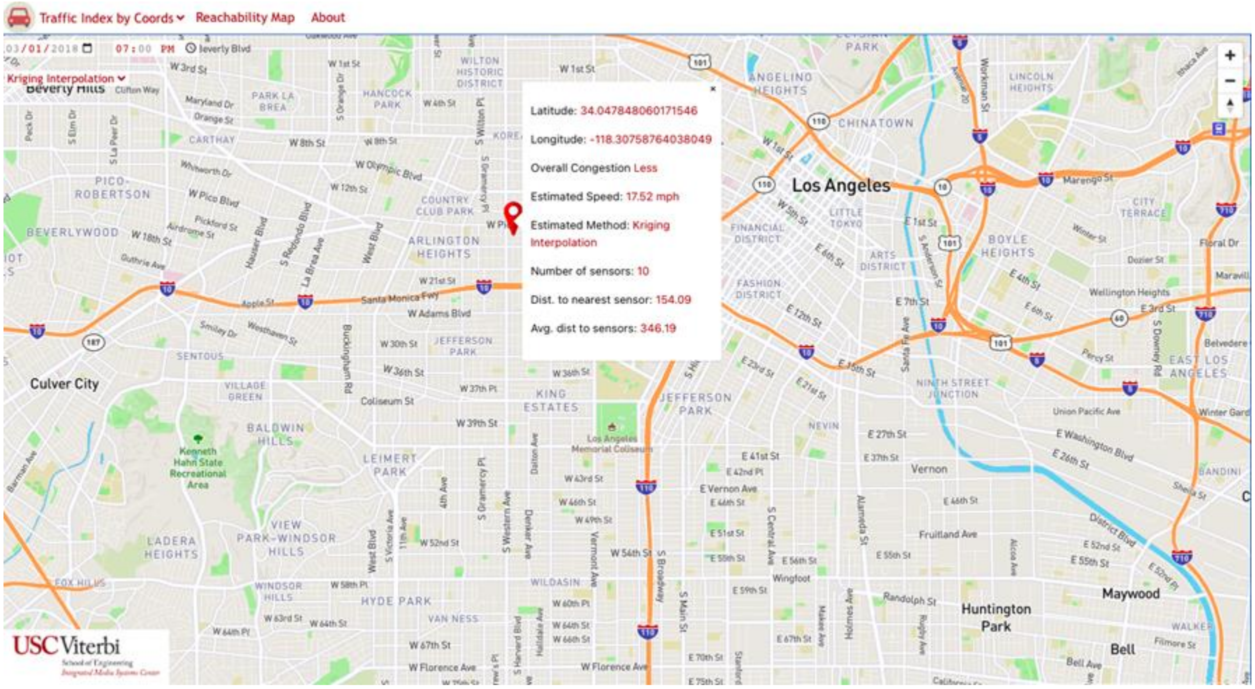
We built a web page to demonstrate our research results – https://imscwww.usc.edu/app/

This website allows the users to get an estimate of traffic congestion and reachability based on the day of the week, the time of the day and type of algorithm.

4.1.1. Traffic Index by Geo-Coordinates

As seen in the picture below, this component provides traffic estimation for a specific location based on the following attributes as input:

- Date and Time
- Latitude and Longitude
- Algorithm to be used for traffic estimation
    - Kriging Algorithm (Default)
    - IDW
    - Nearest Neighbour



Latitude: 34.047848060171546

Longitude: -118.30758764038049

Overall Congestion Less

Estimated Speed: 17.52 mph

Estimated Method: Kriging Interpolation

Number of sensors: 10

Dist. to nearest sensor: 154.09

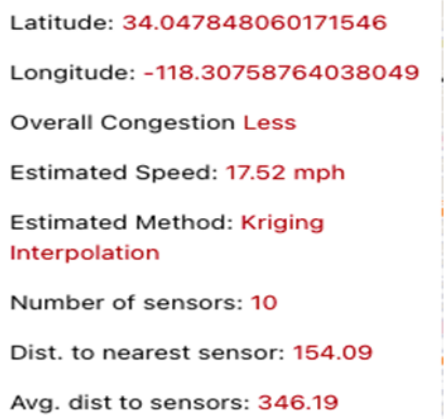
Avg. dist to sensors: 346.19

Figure 15. Example of traffic index input screen

When the user provides these attributes, he gets various layers of details about that location:

The estimated speed at that point, the number of sensors used for calculation, distance to the nearest sensor, and average distance to the sensors. Figure 16 shows how it looks on the web application.

The attribute Overall congestion is decided based on the speed at that point with speed less than 5mph describing” Standstill”, 5mph to 10mph as” Very High”, 10mph to 15mph as” High”, 15mph to 25mph as” Low” and above 25mph as no traffic.



Latitude: 34.047848060171546

Longitude: -118.30758764038049

Overall Congestion Less

Estimated Speed: 17.52 mph

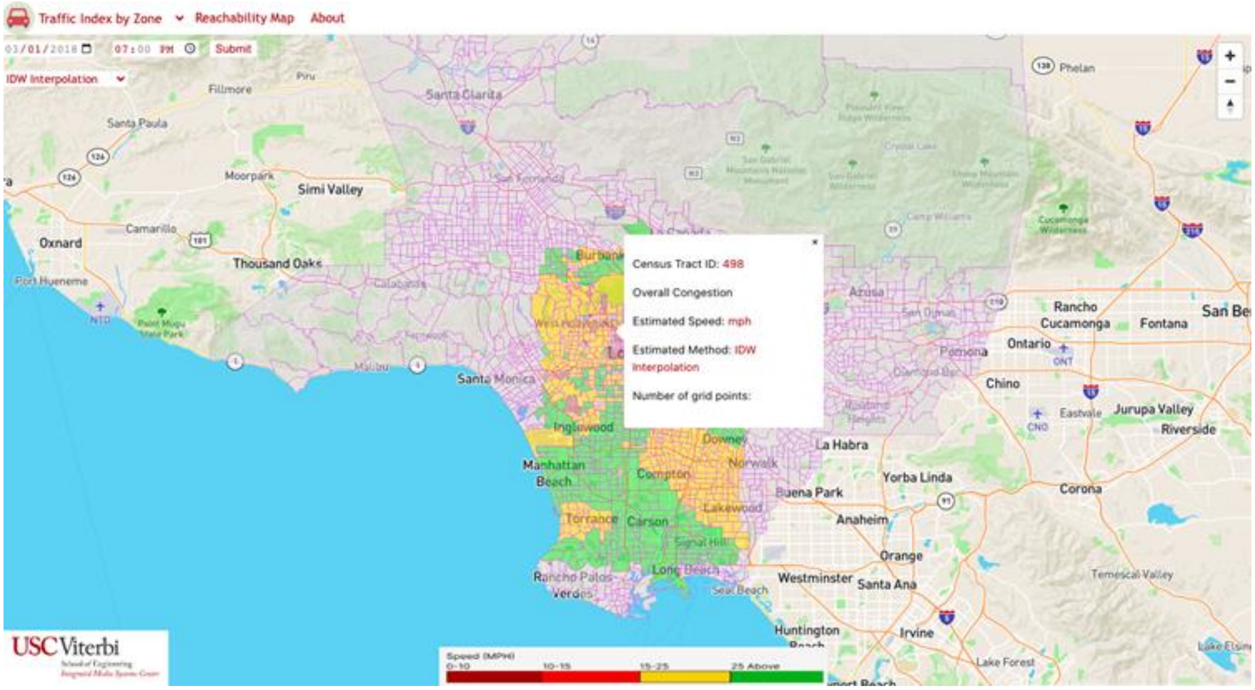
Estimated Method: Kriging Interpolation

Number of sensors: 10

Dist. to nearest sensor: 154.09

Avg. dist to sensors: 346.19

Figure 16. View of Attributes on Web Application



03/01/2018 07:00 PM Submit

IDW Interpolation

Census Tract ID: 498

Overall Congestion

Estimated Speed: mph

Estimated Method: IDW Interpolation

Number of grid points:

Figure 17. Example of traffic index output screen

4.1.2. Traffic Index by Zone

As seen in Figure 17, this component provides traffic estimation based on specific zones, i.e., census blocks in our study.

The speed for each block is calculated by averaging out the speeds for all grid points passing through that block. Grid points are obtained by the intersection of vertical and horizontal lines 500 meters (about 1640.42 ft) apart.

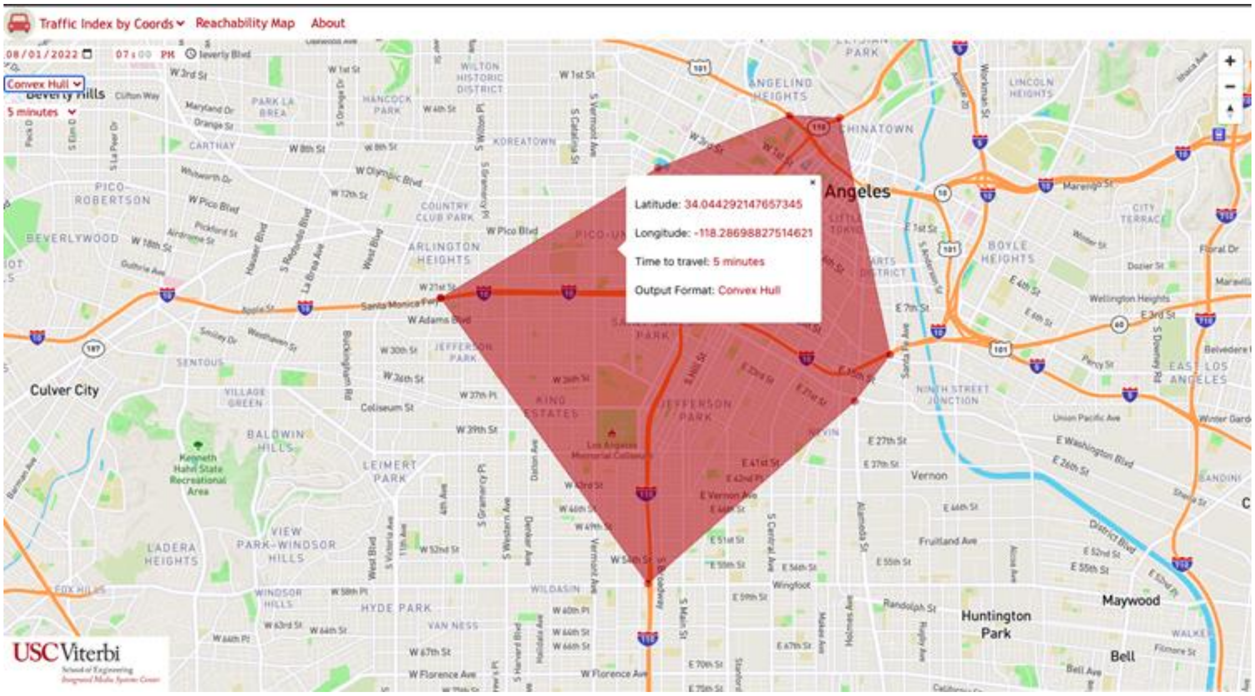
4.1.3. Reachability Map

To construct a Reachability Map, many synthetic trajectories are generated based on time and location within a defined time buffer and finally, a polygon is constructed that circumscribes all of them. Our approach processes isochrone queries by directly processing trajectories. Traffic conditions can vary highly at different times of a day and, therefore, so do isochrone maps. Our system integrates those traffic dynamics by generating time-dependent isochrone maps and can support four types of time-dependent queries. Formally, in graph theory, an isochrone is the minimal, possibly disconnected, sub-graph that covers all the vertices that are within a given time span (or weight budget) from the query source vertex. However, our approach does not consider the underlying road network graph. Instead, our methodology only takes into consideration the trajectories and consists of retrieving all the reachable segments or data points from them for the construction of isochrone maps.

A time-dependent single-source reachability query (SSRQ) is a tuple $Q = (s; t; d)$, where s is the query source point, t is the departure time, and $d > 0$ is the maximum acceptable time span. The query result consists of all location points that are reachable from the source point s within d minutes if one was to depart at time t. The query source can be any point of interest, we assume that the source is selected as a tuple $s = (id; loc)$ where id is a unique identifier for the center and $loc = (lat; lng)$ is its location in latitude and longitude degrees. The main idea is to find all those trajectories that pass by the given query point and retrieve the segment of each trajectory that falls within the given time limit. However, doing this in an efficient way is not an easy task. A straightforward approach is to build an R-tree index on top of the trajectories. Then, a radius search can retrieve all trajectories that pass by a given query point. A last pass over these trajectories will be required to filter out all those that do not pass by the given source at the given departure time. Therefore, this approach would not scale well for large and dense trajectory datasets simply because the filtering step is an expensive linear search. For the implementation we used a grid-based approach that we devised and presented in [26]. Figure 18 shows a generated polygon. The input attributes for the generation of reachability map are: Date and Time, Circumscribing algorithm, and Time buffer.

4.2. Implementation and Limitation

Using React, we split each part of the main page into a single React class - components- in its own file. These components are Traffic Index by Coordinates, Traffic Index by Zones, and Reachability Map all of which are imported into the main component class. Since React allows for easy management of state variables, we can easily send down data from the main component to the others.



08/01/2022

07:00 PM

Convex Hull

5 minutes

Latitude: 34.044292147657345

Longitude: -118.28698827514621

Time to travel: 5 minutes

Output Format: Convex Hull

Figure 18. Example of reachability map screen

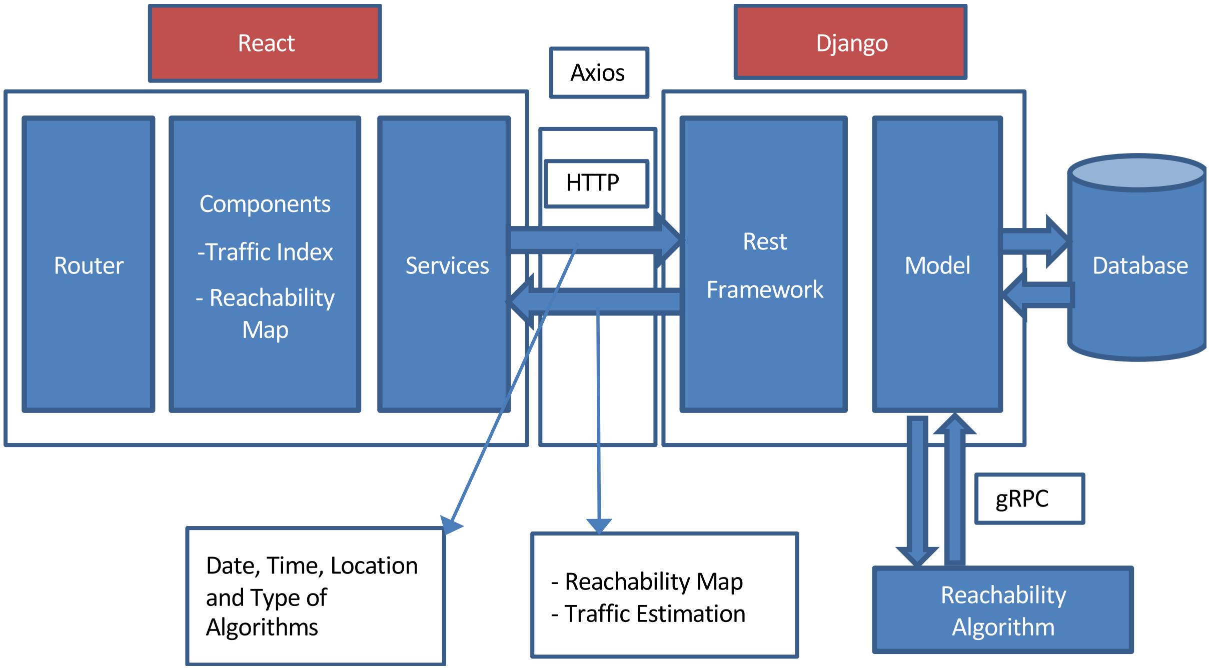


Figure 19. Implementation of Web Features

4.2.1. Traffic Index by Geo-Coordinates

This component of the application can be accessed using the base path of /app. When the user clicks on the map it fetches all the required attributes and generates a JSON which then hits the Django server with a POST request. Based on the location, date, time, and type of chosen algorithm traffic is estimated and the response is sent to the front end.

Three types of algorithms were implemented for traffic estimation:

- Kriging Algorithm (Default): Kriging, also known as Gaussian process regression, is a method of interpolation based on Gaussian process governed by prior covariances. Under suitable assumptions of the prior, kriging gives the best linear unbiased prediction (BLUP) at unsampled locations. For Traffic Estimation we built a hyperplane (Kriging) using 20 closest sensors. The close-by sensors are found by building a KD-Tree. The built hyperplane is then used to get the speed at the unsampled location.
- IDW Algorithm: Inverse distance weighting (IDW) is a type of deterministic method for multivariate interpolation with a known scattered set of points. The assigned values to unknown points are calculated with a weighted average of the values available at the known points. For traffic estimation 20 closest sensors were used inversely weighted on their distance from the position of interest.
- Nearest Neighbour: It is an average of 20 closest sensors found using a KD-Tree.

Below figure shows the accuracy of the three algorithms

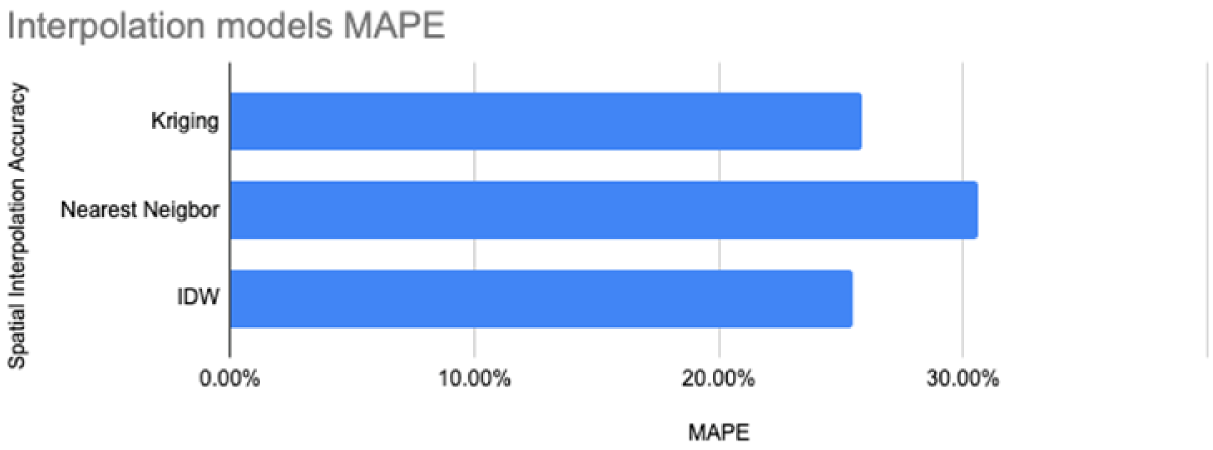


Figure 20. Spatial Interpolation Accuracy

The reason for using 20 closest sensors for traffic estimation is that model accuracy does not increase any further by increasing the number of sensors.

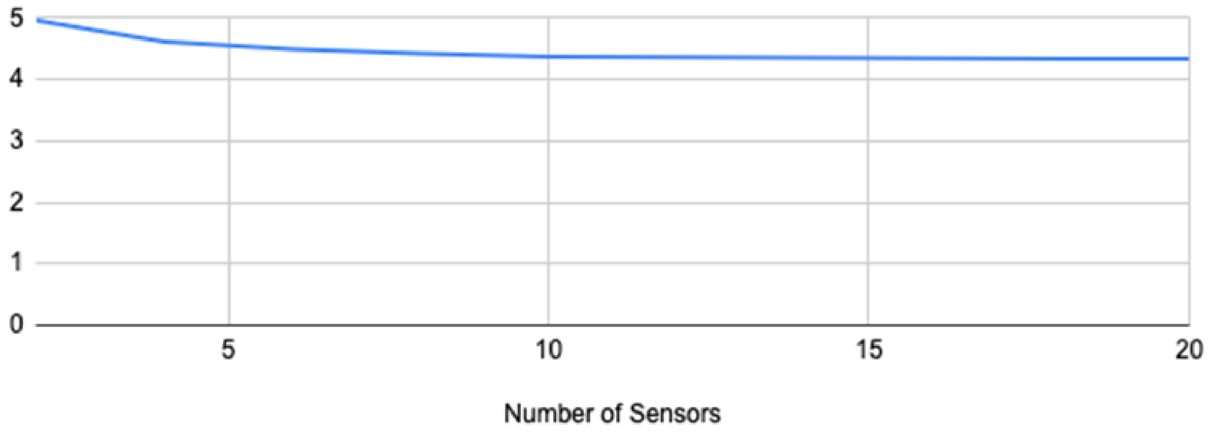


Figure 21. RMSE vs. Number of Sensors

For assorted reasons such as power failure etc. the data that we get from the sensors are usually missing and they need to be imputed before we can do traffic estimation based on the algorithms described above.

Imputing the data on every API call is a very repetitive and time-consuming process. To overcome this challenge, all the missing sensor data is inputted beforehand and stored in the database. The linear Imputation algorithm is used for filling up the missing values with an average MAPE of 6%.

4.2.2. Traffic Index by Zone

This component of the application can be accessed using the base path of /app/traffic-index-zone. When the user clicks on the submit button, it fetches all the required attributes and generates a JSON which then hits the Django server with a POST request. Based on the date, time, and type of chosen algorithm traffic is estimated and for all the census zone and the response is sent to the front end.

The speed for each block is calculated by averaging out the speeds for all grid points passing through that block. Grid points are obtained by the intersection of vertical and horizontal lines 500 meters (about 1640.42 ft) apart. The speed for each grid points is calculated in the same way as the Traffic Index by Coordinates. The latitude and longitude of all grid locations are hashed in the web application, to speed up the traffic estimation when a user clicks on the submit button. The time to estimate speed for all the blocks varies from algorithm to algorithm. Kriging takes the highest amount of time for estimation while IDW is the fastest. The model building time increases linearly for IDW with the increase in number of close-by sensors, while for Kriging the building time increases exponentially as can be seen in Figure 22 and Figure 23.

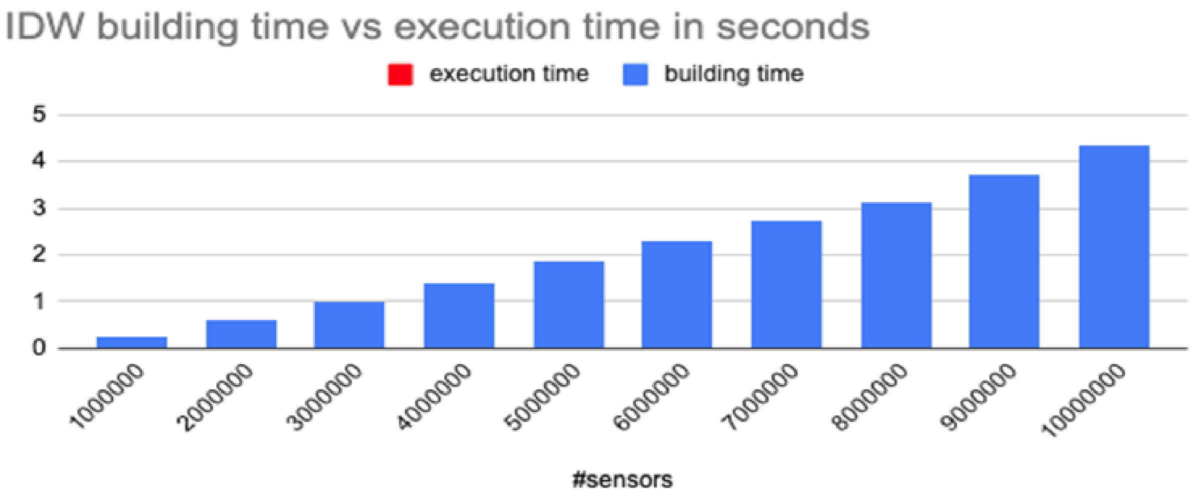


Figure 22. IDW building time vs execution time

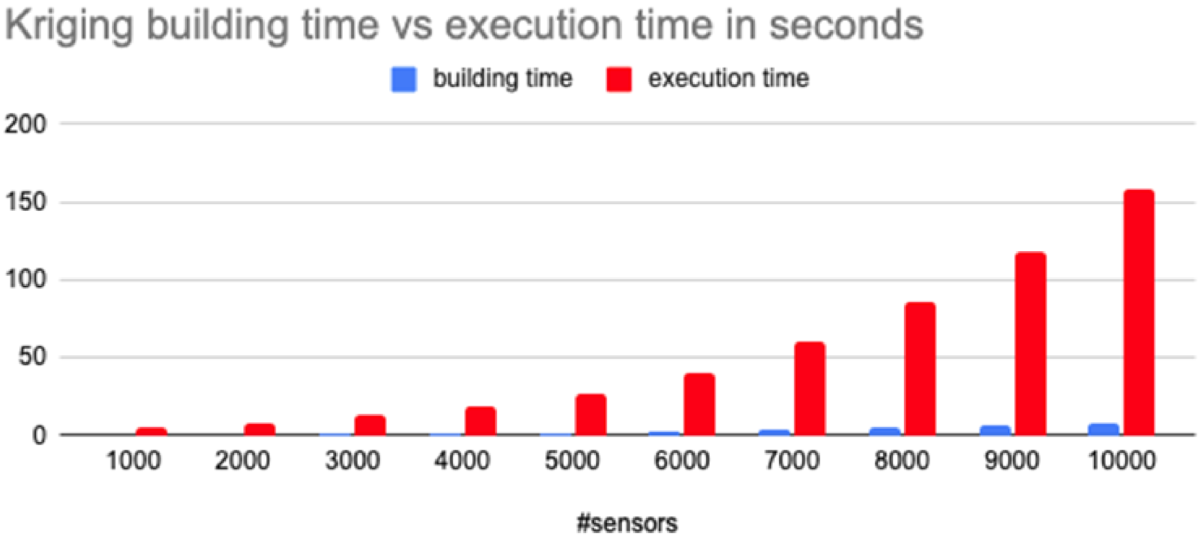
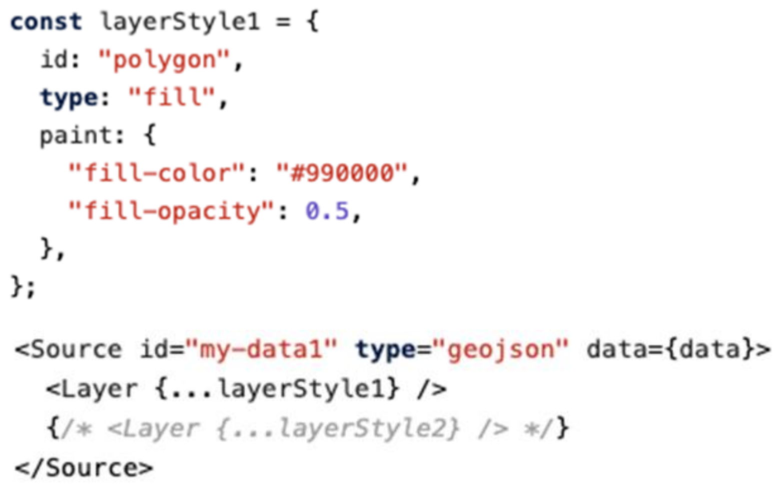


Figure 23. Kriging building time vs execution time

Because of this generating traffic index by zones takes close to 20 seconds for IDW while takes more than 120 seconds (about 2 minutes) for Kriging, as the model building and execution is happening on the fly.

4.2.3. Reachability MAP

This component of the application can be accessed using the base path of /app/reachability. When the user clicks on the map it fetches all the required attributes and generates a JSON file which then hits the Django server with a POST request. The Django server then encapsulates all the information and sends it to the Reachability Algorithm hosted on port 80 via the gRPC service. The Reachability Algorithm then sends back the required structure of the polygon in a JSON format via gRPC services. To display the polygon on the front end the GeoJson-polygon property in Mapbox is used.



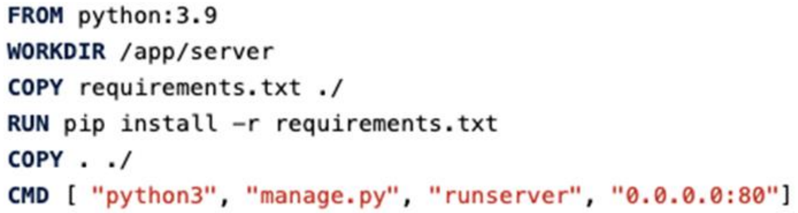
```
const layerStyle1 = {
  id: "polygon",
  type: "fill",
  paint: {
    "fill-color": "#990000",
    "fill-opacity": 0.5,
  },
};

<Source id="my-data1" type="geojson" data={data}>
  <Layer {...layerStyle1} />
  {/* <Layer {...layerStyle2} /> */}
</Source>
```

Figure 24. GeoJson-polygon property script

After building the individual components in Recat we converted all of them to static JS files, using NPM RUN BUILD. The advantage of this is that we can move all of them to the presentation layer (Template) of Django which helps us to avoid hosting the front-end and back-end separately. Now we just need to Dockerize the Django to run the application.

The Dockerize script is follows:



```
FROM python:3.9
WORKDIR /app/server
COPY requirements.txt ./
RUN pip install -r requirements.txt
COPY . ./
CMD [ "python3", "manage.py", "runserver", "0.0.0.0:80"]
```

Figure 25. Dockerize script

The above lines of code packages the application into a docker container by combining applications source code with the operating system (OS) libraries and dependencies required to run that code in any environment. As we can see the application would be running port 80, which is publicly accessible.

4.2.4. Possible Improvements

- Execution time for traffic estimation can be enhanced. This can be achieved by saving in the traffic index for all grid points in the database to avoid on-the-fly calculations.
- Connection to the Database currently is working only for a small subset of the data, which can be extended to all data.
- More detailed displays in both Traffic Index and Reachability map might enhance human perception of the results.

5. Related Work

5.1. Traffic Analytics

Several systems have been proposed in the past for managing and analyzing traffic data to derive insights. PeMS [14], short for Performance Evaluation Monitoring System, collects and archives highway traffic data with the main goal of detecting traffic patterns and estimating the travel time on highway links. ADMS [1], short for Archived Data Management System, is a big data system that can support spatiotemporal queries through an interactive web-based interface over a large transportation database. However, none of these systems focus on improving the quality of the collected data or summarizing traffic at the level of TAZ.

5.2. Temporal Imputation

Data imputation methods are commonly divided into three categories: prediction-based, interpolation-based, and statistical learning-based. Most recent studies are focused on imputing time series data using machine learning methods. In [27] the authors train a large Generative Adversarial Network (GAN) to impute the missing values in timeseries datasets. In [28] a recurrent neural network (RNN) is employed to impute time series datasets. In this work, the missing values are treated as variables in the RNN model and are iteratively filled during backpropagation. MVLM [29] leverages the fact that driver behavior tends to be correlated in both the spatial and temporal domains. Modified window-based nearest neighbor methods have also been proposed for imputing traffic data [30], [31]. However, none of these previous works offer a comprehensive comparison of the imputation performance on sensor-based traffic data.

5.3. Spatial Interpolation

Most of the work in this field proposes general interpolation methods. Specifically, such methods are applied to air quality data in order to interpolate pollution metrics at locations where data is not sensed [32]. Adaptations of interpolation methods have also been proposed for traffic data. In [33] the Kriging method is used on an Annual Average Daily Traffic (AADT) dataset to interpolate the volume at unknown locations. An improved distance metric is later proposed to replace the Euclidean distance that is widely used along with Kriging [34]. Previous work mostly focuses on highway traffic which is fundamentally different from arterial (street) traffic.

5.4. Computational Generators of Synthetic Trajectories

One of the first studies in spatio-temporal moving object generators (MOG) is [35] with the goal to generate artificial trajectories with realistic spatiotemporal properties. An extension was later proposed [36] to enable a greater degree of configuration by adding additional parameters to the generator. These early MOGs, however, generate trajectories in free space but vehicle trajectories are by definition constrained to a road network. To overcome this, network-constrained generators have been proposed. The most widely cited attempt of generating moving object constrained to a network in a realistic way is proposed in [10]. The framework selects origin and destination nodes and routes an agents along the network with limited interactions with other agents on the network. Perhaps the most prominent simulator is the Simulation of Urban MObility (SUMO) framework [37]. SUMO is a microscopic traffic simulator that generates and exports trajectories by simulating the environment from three inputs: road network, traffic infrastructure, and traffic demand. It also allows the user to specify a variety of parameters such as traffic infrastructure (e.g., traffic lights), number of lanes, and vehicle types.

Most of these methods are highly configurable. However, to generate realistic trajectories, their parameters must be calibrated in order to match the target real-world environment; a task that is often time-consuming and required domain knowledge. Additionally, as the region of interest becomes large, e.g., at the scale of a metropolitan city, simulators require large amounts of computational power, a fact that led to the proposal of distributed architectures [38] [39]. *DDTG* does not require any calibration or time-consuming configuration before it can generate realistic vehicle trajectories and can easily scale to metropolitan-sized road networks. *DDTG*'s algorithm can also be parallelized to generate millions of trajectories in minutes.

5.5. Data-Driven Generators of Synthetic Trajectories

Computational generators, as mentioned above, require the calibration of complex parameters. To address this, nonparametric machine learning methods have been proposed to synthesize trajectories. In the earliest attempt [40], an LSTM architecture is proposed for predicting movements from GPS traces. Because very often the training dataset size is limited due to privacy and other constraints, the authors in [41] use a variational autoencoder to map the input to a hidden space where the characteristics can be preserved.

More recently, Generative Adversarial Networks (GAN) have been proposed as the state-of-the-art in generating synthetic trajectories. In [42], the authors discretize the input in both the spatial and temporal domains and train a nonparametric generative model. The discretized output is then converted back into location trajectories. [43] proposes a self attention-based sequential modeling network as the generator to encode the complex temporal transitions of human mobility. The authors also make use of a mobility regularity-aware loss in the discriminator to help it distinguish between real and synthetic trajectories. In [13] the authors formulate the task as an imitation learning problem in a partially observable Markov decision process. In [21] the GAN is trained with a new loss function, named TrajLoss, that accounts for the dissimilarity of generated trajectories to the input.

Generative models require large amounts of data in order to learn how to generate realistic trajectories. However, even in big trajectory datasets, the distribution tends to be skewed, i.e., limited to only certain regions of the metropolitan city and hours of the day, and hence the trained models are biased. Another shortcoming of these methods is that the optimization objective is limited to the holistic level quality of the generated dataset and the quality of individually generated trajectories is mostly ignored. *DDTG* only requires the traffic data and a target OD matrix for the region of interest, both of which are very often publicly available and free of privacy concerns. The synthetic datasets generated by *DDTG* follow distributions that are very close to those of real trajectory datasets.

6. Conclusions and Future Directions

We presented and compared methods for imputing data in time series traffic data and interpolating the traffic values at locations where data is not sensed. On top of these methods, we propose an algorithm to construct a metropolitan-scale Traffic Index that represents the traffic condition at each census block at different times of the day. The result can be used not only for visualization purposes, but also for informing downstream tasks such as socio-economic analysis and urban planning. Our experiments showed that datasets with many small scale gaps can be accurately imputed using statistical approaches such as linear and spline interpolation without requiring any expensive pre-processing steps. On the other hand, as missing data gaps become longer and more frequent, data-driven approaches such as historical averages become much more accurate. This work was published in the IEEE International Conference on Intelligent Transportation 2022 [24].

Real-world trajectory datasets are often unavailable due to privacy reasons or are limited in size and spatio-temporal diversity. The task of synthetic trajectory generation aims to create a realistic set of artificial trajectories that are representative of the true mobility characteristics and underlying distributions. Existing methods require either a lot of complex parameters to be calibrated (simulators) or existing trajectory datasets (GAN) in order to be able to generate synthetic datasets. In this paper, we proposed *DDTG*, a data-driven, model-free, and parameter-less algorithm for generating realistic vehicle trajectory datasets. Unlike existing approaches, *DDTG* only requires two main inputs: the traffic data, and the origin-destination matrix for the region of interest. These aggregated inputs are very often publicly available because they do not reveal any private information. As we showed in our experiments, the datasets generated by *DDTG* closely mimic the distributions of a real trajectory dataset. We also argue that our method is not meant to replace the existing state-of-the-art methods but rather complement them in generating synthetic datasets of higher quality. This work has been accepted and will be published in the IEEE International Conference on Big Data 2022 [44].

References

[1] T. Litman, Evaluating Accessibility for Transport Planning: Measuring People's Ability to Reach Desired Goods and Activities, Victoria Transport Policy Institute, June 2020, https://www.vtpi.org/access.pdf.

[2] [B. Ding, J. X. Yu, and L. Qin, "Finding time-dependent shortest paths over large graphs," in EDBT'08, ser. EDBT '08. New York, NY, USA: ACM, 2008, pp. 205–216.

[3] C. Anastasiou, J. Lin, C. He, Y.-Y. Chiang, and C. Shahabi, "Admsv2:A modern architecture for transportation data management and analysis," in Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Advances on Resilient and Intelligent Cities, 2019, pp. 25–28.

[4] L. M. Mart´ınez, J. M. Viegas, and E. A. Silva, "A traffic analysis zone definition: a new methodology and algorithm," Transportation, vol. 36, no. 5, pp. 581–599, 2009.

[5] Q. Huang and D. W. Wong, "Modeling and visualizing regular human mobility patterns with uncertainty: An example using twitter data," Annals of the Association of American Geographers, vol. 105, no. 6, pp. 1179–1197, 2015.

[6] J. Kim and H. S. Mahmassani, "Spatial and temporal characterization of travel patterns in a traffic network using vehicle trajectories," Transportation Research Procedia, vol. 9, pp. 164–184, 2015.

[7] Q. Huang and Y. Xiao, "Geographic situational awareness: mining tweets for disaster preparedness, emergency response, impact, and recovery," ISPRS International Journal of Geo-Information, vol. 4, no. 3, pp. 1549–1568, 2015.

[8] C. Keßler and G. McKenzie, "A geoprivacy manifesto," Transactions in GIS, vol. 22, no. 1, pp. 3–19, 2018.

[9] C.-Y. Chow and M. F. Mokbel, "Trajectory privacy in location-based services and data publication," ACM Sigkdd Explorations Newsletter, vol. 13, no. 1, pp. 19–29, 2011.

[10] T. Brinkhoff, "A framework for generating network-based moving objects," GeoInformatica, vol. 6, no. 2, pp. 153–180, 2002.

[11] J. Barcel´o, E. Codina, J. Casas, J. L. Ferrer, and D. Garc´ıa, "Microscopic traffic simulation: A tool for the design, analysis and evaluation of intelligent transport systems," Journal of intelligent and robotic systems, vol. 41, no. 2, pp. 173–203, 2005.

[12] Y. Hollander and R. Liu, "The principles of calibrating traffic microsimulationmodels," Transportation, vol. 35, no. 3, 2008.

[13] S. Choi, J. Kim, and H. Yeo, "Trajgail: Generating urban vehicle trajectories using generative adversarial imitation learning," Transportation Research Part C: Emerging Technologies, vol. 128, p. 103091, 2021.

[14] "Pems: Freeway performance measurement system," https://pems.eecs.berkeley.edu/, accessed: 2022-08-13.

[15] P. Krishnakumari, H. van Lint, T. Djukic, and O. Cats, "A data driven method for od matrix estimation," Transportation Research Procedia, vol. 38, pp. 139–159, 2019.

[16] S. Zeighami, R. Ahuja, G. Ghinita, and C. Shahabi, "A neural database for differentially private spatial range queries," Proceedings of the VLDB Endowment, vol. 15, no. 5, pp. 1066–1078, 2022.

[17] S. Shaham, G. Ghinita, and C. Shahabi, "Differentially private publication of origin-destination matrices with intermediate stops," Proceedings of the 25th International Conference on Extending Database Technology (EDBT), 2022. [Online]. Available: https://par.nsf.gov/biblio/10333510

[18] "Openstreetmap," https://openstreetmap.org, accessed: 2022-08-13.

[19] "Veraset website," https://www.veraset.com/about-veraset, accessed: 2022-08-13.

[20] T. Abrahamsson, "Estimation of origin-destination matrices using traffic counts: A literature survey," IIASA Interim Report, 1998.

[21] U. Demiryurek, F. Banaei-Kashani, C. Shahabi, and A. Ranganathan, "Online computation of fastest path in time-dependent spatial networks," in International Symposium on Spatial and Temporal Databases. Springer, 2011, pp. 92–111.

[22] T. Chondrogiannis, P. Bouros, J. Gamper, U. Leser, and D. B. Blumenthal, "Finding k-shortest paths with limited overlap," The VLDB Journal, vol. 29, no. 5, pp. 1023–1047, 2020.

[23] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma, "Mining user similarity based on location history," in Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems, 2008, pp. 1–10.

[24] C. Anastasiou, J. Zhao, S. H. Kim, and C. Shahabi, "Data-driven traffic index from sparse and incomplete data," IEEE Transactions on Intelligent Transportation Systems, 2022.

[25] L. Yu, W. Zhang, J. Wang, and Y. Yu, "Seqgan: Sequence generative adversarial nets with policy gradient," in Proceedings of the AAAI conference on artificial intelligence, vol. 31, no. 1, 2017.

[26] C. Anastasiou, C. Huangy, S. H. Kim, C. Shahabi. Time-Dependent Reachability Analysis: A Data-Driven Approach. IEEE International Conference on Mobile Data Management (MDM '19), June 2019.

[27] Y. Luo, X. Cai, Y. Zhang, J. Xu et al., "Multivariate time series imputation with generative adversarial networks," Advances in neural information processing systems, vol. 31, 2018.

[28] W. Cao, D. Wang, J. Li, H. Zhou, L. Li, and Y. Li, "Brits: Bidirectional recurrent imputation for time series," Advances in neural information processing systems, vol. 31, 2018.

[29] L. Li, J. Zhang, Y. Wang, and B. Ran, "Missing value imputation for traffic-related time series data based on a multi-view learning method," IEEE Transactions on Intelligent Transportation Systems, vol. 20, no. 8, pp. 2933–2943, 2019.

[30] B. Sun, L. Ma, W. Cheng, W. Wen, P. Goswami, and G. Bai, “An improved k-nearest neighbours method for traffic time series imputation,” in 2017 Chinese Automation Congress (CAC), 2017, pp. 7346–7351.

[31] S. Tak, S. Woo, and H. Yeo, “Data-driven imputation method for traffic data in sectional units of road links,” IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 6, pp. 1762–1771, 2016.

[32] D. W. Wong, L. Yuan, and S. A. Perlin, “Comparison of spatial interpolation methods for the estimation of air quality data,” Journal of Exposure Science & Environmental Epidemiology, vol. 14, no. 5, pp. 404–415, 2004.

[33] X. Wang and K. M. Kockelman, “Forecasting network data: Spatial interpolation of traffic counts from texas data,” Transportation Research Record, vol. 2105, no. 1, pp. 100–108, 2009.

[34] H. Zou, Y. Yue, Q. Li, and A. G. Yeh, “An improved distance metric for the interpolation of link-based traffic data using kriging: a case study of a large-scale urban road network,” International Journal of Geographical Information Science, vol. 26, no. 4, pp. 667–689, 2012.

[35] Y. Theodoridis, J. R. Silva, and M. A. Nascimento, “On the generation of spatiotemporal datasets,” in International Symposium on Spatial Databases. Springer, 1999, pp. 147–164.

[36] N. Xu, L. Trinh, S. Rambhatla, Z. Zeng, J. Chen, S. Assefa, and Y. Liu, “Simulating continuous-time human mobility trajectories.”

[37] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Fl¨otter¨od, R. Hilbrich, L. L¨ucken, J. Rummel, P. Wagner, and E. Wießner, “Microscopic traffic simulation using sumo,” in 2018 21st international conference on intelligent transportation systems (ITSC). IEEE, 2018, pp. 2575–2582.

[38] H. Xie, E. Tanin, K. Ramamohanarao, S. Karunasekera, L. Kulik, R. Zhang, and J. Qi, “Generating traffic data for any city using smarts simulator,” SIGSPATIAL Special, vol. 11, no. 1, pp. 22–28, 2019.

[39] J. Yu, Z. Fu, and M. Sarwat, “Dissecting geosparksim: a scalable microscopic road network traffic simulator in apache spark,” Distributed and Parallel Databases, vol. 38, no. 4, pp. 963–994, 2020.

[40] X. Song, H. Kanasugi, and R. Shibasaki, “Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level,” in Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016, pp. 2618–2624.

[41] D. Huang, X. Song, Z. Fan, R. Jiang, R. Shibasaki, Y. Zhang, H. Wang, and Y. Kato, “A variational autoencoder based generative model of urban human mobility,” in 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR). IEEE, 2019, pp. 425–430.

[42] K. Ouyang, R. Shokri, D. S. Rosenblum, and W. Yang, “A nonparametric generative model for human trajectories.” in IJCAI, vol. 18, 2018, pp. 3812–3817.

[43] J. Feng, Z. Yang, F. Xu, H. Yu, M. Wang, and Y. Li, “Learning to simulate human mobility,” in Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining, 2020, pp. 3426–3433.

[44] C. Anastasiou, S. H. Kim, C. Shahabi, “Generation of Synthetic Urban Vehicle Trajectories”, To appear in the IEEE International Conference on Big Data (BigData 2022), Dec. 2022.

[45] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko, “Computing isochrones in multi-modal, schedule-based transport networks”, In GIS’08, ser. GIS’08. New York, NY, USA: ACM, 2008, pp. 78:1–78:2.

[46] de Berg, Mark, "Analyzing Trajectories of Moving Objects", Computational Geometry, Two Selected Topics (PDF), pp. 11–75.

Data Summary

Products of Research

This project produced time-dependent synthetic trajectory dataset for Los Angeles based on the traffic flow data from 1) real sensors whenever available, and 2) estimation if sensors are not available. These synthetic trajectory data were used for our reachability research. We also have completed the steps to estimate missing values in generating synthetic trajectories. We took the step to represent all data in popular structured formats following the recommendation for data format and metadata standard in the NCST data management plan. The synthetic dataset of 1.5M trajectories were stored on the UC's data repository, Dryad and made it publicly accessible.

We generated two publications from this research:

- C. Anastasiou, J. Zhao, S. H. Kim, and C. Shahabi, "Data-driven traffic index from sparse and incomplete data," IEEE Transactions on Intelligent Transportation Systems, 2022.
- C. Anastasiou, S. H. Kim, C. Shahabi. Generation of Synthetic Urban Vehicle Trajectories. To appear in the IEEE International Conference on Big Data (BigData 2022), Dec. 2022.

Data Format and Content

Title of Dataset: DDTG Synthetic Vehicle Trajectory Dataset

This dataset contains a total of 1.5 million (1.5M) synthetic trajectories generated by our DDTG algorithm. The trajectories are limited to the Metropolitan City of Los Angeles (bounding box: `top=34.335035 left=-118.667665 bottom=33.699820 right=-118.180532`) and to the month of December 2019.

Description of the data and file structure: The dataset consists of two files. One file contains the actual trajectory data in GPS measurements form. The other file contains metadata about each trajectory. Both files are in Parquet (https://parquet.apache.org/) format.

File 1: trajectories.parquet
- uuid: Unique user id
- tuid: Unique trajectory id
- latitude: Latitude degree of location coordinate
- longitude: Longitude degree of location coordinate
- accuracy: GPS accuracy at time of reading (not used in this version)
- timestamp: Time of reading

File 2: trajectory_meta.parquet
- uuid: Unique user id
- tuid: Unique trajectory id
- origin_latitude: Latitude degree at the trip origin
- origin_longitude: Longitude degree at the trip origin

- destination_latitude: Latitude degree at the trip destination
- destination_longitude: Longitude degree at the trip destination
- departure_time: Time at trip's departure
- travel_distance: Total distance travel in this trip
- travel_time: Total travel time of this trip (estimated with real-world traffic data)
- displacement: The total displacement from origin to destination
- count: Number of GPS readings in this trajectory

Data Access and Sharing

Data is available on the UC's data repository, Dryad. Anyone can download the two files from the repository (https://doi.org/10.5061/dryad.4j0zpc8gf).

An example of Python code to access the data is:

```python
import pandas as pd

trajectories = pd.read_parquet('./trajectories.parquet')
meta = pd.read_parquet('./trajectory_meta.parquet')
```

Reuse and Redistribution

No restriction. Dataset should be cited as follows:

Anastasiou, Chrysovalantis (2022), Synthetic vehicle trajectory dataset for the metropolitan city of Los Angeles using DDTG, Dryad, Dataset, https://doi.org/10.5061/dryad.4j0zpc8gf