MSc Artificial Intelligence
Master Thesis

Synthetic Highways: A Synthetic Dataset for SD Map Invalidation

by
Kevin Waller
11304421

June 8, 2022

48 ECTS
November 2021 - June 2022

Supervisors:
Dr Shaodi You
MSc Ysbrand Galama
MSc Arent Stienstra

Examiner:
Dr Shaodi You

Second reader:
MSc Qi Wang

Universiteit van Amsterdam

Contents

Abstract

In this paper, we introduce a synthetic dataset, called *Synthetic Highways* for Standard Definition (SD) Map Invalidation, in which that task is to figure out what roads in a map are changed/removed, based on trajectory data. Real-world datasets for this task are nearly impossible to obtain, due to the difficulty in knowing exactly what roads were actually inactive at specific times. Our dataset is scraped from the game *Cities: Skylines*, a city-building/simulation game in which users build and manage everything from road network, zoning, public services, taxes etc. Using a mod for the game, synthetic GPS data is exported from the game and full-control of the road network is managed by a custom game-mod that makes changes to the road network to obtain a reliable SD map invalidation dataset. Additionally, we introduce three methods for map invalidation on the dataset, two of which are based on the already well-established area of map-matching, and one purpose-built SD map invalidation method. Our results conclude that the purpose-built method outperforms the map-matching based techniques in almost any scenario or trajectory augmentation, motivating the need for research in this area separate from map-matching.

Chapter 1

Introduction

Maps and navigation are commonplace in today's society. Transportation and logistics companies, as well as commuters around the world, rely on up-to-date navigation apps on a day-to-day basis. Therefore, for map makers, it is quintessential to assure maps are updated to reflect the real-world. However, maps have been shown to become quickly outdated [16], due to the frequent changes in road networks.

This introduces the task of map change detection, which attempts to identify how a map may have changed based on some data. Map change detection can be performed on both Standard-Definition (SD) and High-Definition (HD) maps. SD maps are commonly used by the daily commuter, using a relatively simple map to find the most efficient route from A to B localizing using GPS sensors, whereas HD maps are used for highly-automated driving systems, combining sensor data with a highly detailed map to localize and navigate a vehicle accurately. The focal point of this paper is on SD map change detection, or more specifically, on the subtasks of map invalidation on SD maps. Map change detection focuses more generally on any kind of changes that may have occurred in the map, ranging from lane/road closure and addition, to a change in the speed limit, number of lanes, directionality, or anything else that is encoded in the SD map. Map invalidation is a more specific task, aiming to identify what portions of the map may have become invalid, ie, what roads may have been removed.

To perform map invalidation, some kind of data is needed to inform the map about the current/recent activity on the roads and what the state is. This can be done using a variety of data, but one common sensor used in GPS. As opposed to LIDAR, more commonly used for HD map change detection, GPS data is far more ubiquitous and inexpensive to acquire, and has become the data of choice for many SD map tasks, such as congestion analysis. However, datasets for map invalidation are hard to come by and difficult to build, as road changes are often times not communicated or logged such that a clean and reliable dataset can be constructed for such a task, especially for modern learning algorithms that require large amounts of data, motivating the need for a possible synthetic dataset.

In this paper, we introduce a synthetic GPS dataset for SD map invalidation, called *Synthetic Highways*. It is collected from the game *Cities: Skylines*, a city-simulation/building game, using a *game mod* or modification, which allows us to scrape data from the game and control the state of the map reliably. The dataset is structured to include the map and a set of trajectories over two "snapshots", or two differnet points in time, with changes that have happened in between. Although the dataset itself is rather small, this is due to it serving as a benchmark for evaluating different map invalidation methods. The mod is publically available for those who wish to scape additional data from the game. We hope for this dataset to serve as a useful benchmark for SD map invalidation using GPS probe traces.

Furthermore, we introduce three methods on the task of SD map invalidation, two based on *map-matching* [8] [14], as well as one purpose-built map invalidator which takes inspiration from

Kernel-Density-Estimation, a *map-inferencing* algorithm [3]. *Map-matching* as a task aims to match a single GPS trajectory to the edges/roads it may have actually traversed, and can be applied to the problem of map invalidation by analyzing what edges of the road network are not traversed or traversed infrequently. *Map-inferencing*, another mapping task which infers a map given only trajectories, may seem useful in performing map invalidation, however we motivate why it falls short on the task, mainly due to the fact that in map invalidation, most of the map remains the same, whereas map-inferencing algorithms consider only trajectories.

We design several experiments to test the robustness of our SD map invalidation methods to different kinds of trajectory augmentations, and conclude that the purpose-built map invalidator outperforms the map-matching based approaches in almost every scenario, motivating the need for the *Synthetic Highways* dataset and research into specific map invalidation algorithms.

Chapter 2

Related work

2.1 SD vs HD maps

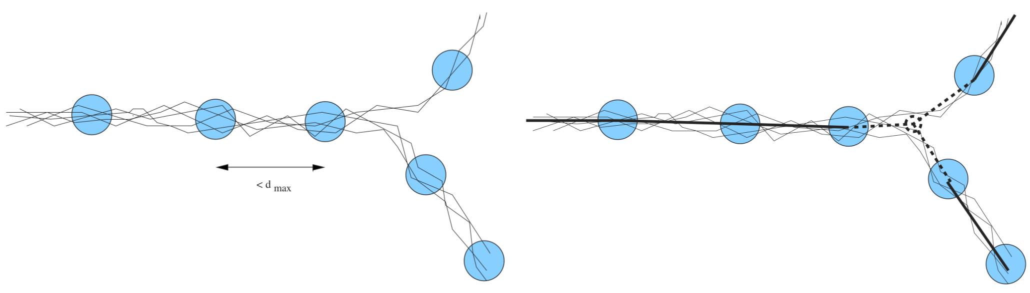
Before we proceed, it is essential to differentiate between *Standard Definition (SD)* and *High-Definition (HD)* maps. SD maps are what most people associate with maps for navigation, and are also the most common in everyday use. They contain the road network and road geometry, such that human drivers can use them to guide and navigate them to and from destinations, rather than the driver having to rely on memory or physical maps. HD maps, also referred to as Highly Automated Driving (HAD) maps, are highly accurate and detailed maps for the purpose of (highly) autonomous driving. Rather than having autonomous vehicles drive around with little to no prior knowledge of the road network ahead and rely solely on sensor data for its surroundings, HD maps offer a detailed look into the road ahead. It includes everything from road geometry to road barriers, traffic signs, traffic lights, lane markings, etc. There is a considerable amount of research being done in HD map change detection in recent years, due to the increased interest for autonomous driving capabilities [11] [9] [15] [21]. Although HD maps are becoming increasingly popular for navigation companies, such as TomTom and HERE Technologies, to build, SD maps are still essential to the everyday road-user. SD map invalidation has remained relatively unexplored in the research community. Therefore, this paper focuses on SD map invalidation through GPS trajectories.

2.2 SD Map Inferencing using GPS Probe Traces

A task similar to that of map invalidation is map inferencing through GPS probe data. Map inference algorithms do not directly focus on map invalidation, however, they can be part of a larger system used for map invalidation. Given a set of trajectories, the task is to infer the road network as a map or graph. There have been many approaches to solving map inferencing through GPS probe data, however, other approaches make use of satellite or aerial photography for map inferencing rather than GPS probe data. Due to the availability and frequency of GPS probe traces, this paper focuses on map inferencing through GPS probe data. Following will be an overview of the problem formulation of map inferencing, as well as the three main approaches to GPS map inferencing.

2.2.1 Problem Formulation

A map is represented as a graph G, containing vertices (or nodes) and edges, $G = (V, E)$. Every vertex has a set of coordinates, $v = (lat, lon)$, and two vertices can be connected to each other by means of edges $e = (u, v)$. Together, this creates a graph. Trajectories, denoted as T, also referred to as GPS probe traces, are sequences of timed points representing a path taken



(a) K-means Clustering (b) K-means Edge Inferencing

Figure 2.1: Figures showing how K-means map inferencing works. In figure 2.1a, the clusters are located based on the timed points' coordinates and headings. In figure 2.1b, the edges are inferred by looking at how many trajectories pass through subsequent cluster centroids [5]

by a vehicle over time. A trajectory is defined by a sequence of *timed points*, a point which is defined by its coordinates and timestamp, $p = (lat, lon, t)$, where t is the UNIX timestamp. Thus, a single trajectory, also known as a *timed path* is a sequence of successive *timed points*. Trajectories and maps can also include additional information, such as the number of lanes, road types and speed limits for maps, as well as headings and speed for trajectories. In *map inference*, the task is to construct a function f which, given the trajectories T, can infer the map G: $f(T) \rightarrow G$.

2.2.2 K-means Map Inferencing Methods

K-means based approaches are the most common and simple of the existing map inferencing approaches. Originally introduced by Edelkamp and Schrödl [5], *k-means* map inference methods work similarly to regular k-means algorithms, where clusters are computed in an iterative manner. k is a parameter referring to the number of clusters to be found, representing the road centerlines. Figure 2.1 shows the process of k-means map inferencing. In figure 2.1a, the clustering is performed based on the timed points' coordinates and heading. After several iterations of k-means clustering, this results in cluster centroids, representing the road centerline with coordinates as well as headings to indicate the road directionality. An additional constraint is added to the cluster centroids which ensures that consecutive cluster centroids are each separated by d_{max} meters. The road edges are inferred by connecting the centroids together based on a connectivity metric, shown in figure 2.1b. When trajectories pass through subsequent centroids, it can be inferred that an edge must be placed. Similarly, to infer intersections, if trajectories pass through a centroid but then split up to pass through multiple centroids, an intersection can be inferred, resulting in the inferred map.

2.2.3 Trace-merging Map Inferencing Methods

Trace-merging methods iteratively build a map by adding raw trace edges, and then prunes edges with low weights. For each edge in every GPS trajectory, an edge is added to the map if there is not already a similar existing edge in the map. The similarity is calculated based on location and heading. If a similar edge does exist, the existing edge is given a higher weight in the map. After all traces have been merged, the map is pruned using a threshold on the weight. A toy example of trace-merging is shown in figure 2.2. In figure 2.2a, all three raw traces are shown. Figure 2.2b merges trajectory (or trip) 1 in its entirety, due to there being no existing edges in the map. The second trajectory is partially merged, as shown in figure 2.2c, as there are some edges in the trajectory that closely match those in the map in 2.2b from trajectory 1. This means that those edges will get a higher weight, lowering the likelihood of

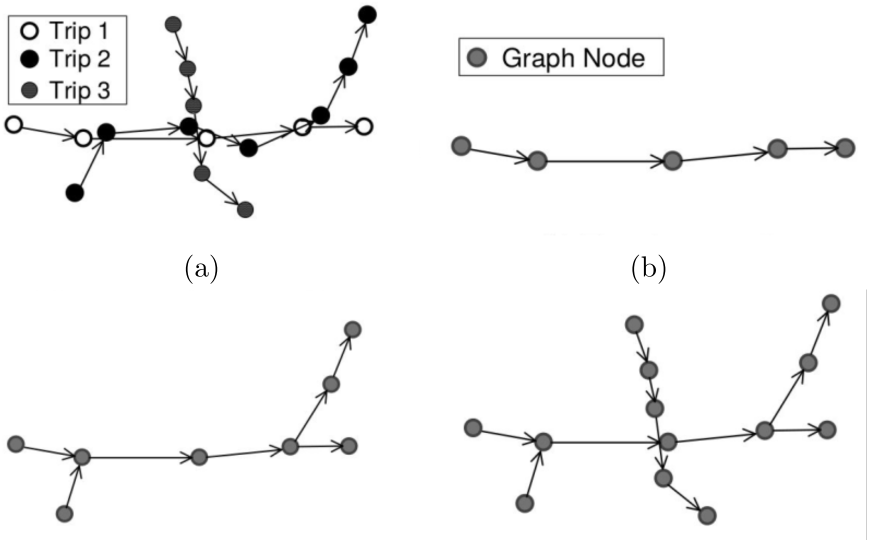


Figure 2.2: Figure showing the steps in trace-merging map inferencing methods. In 2.2a, all traces are shown. In 2.2b, the first trajectory (or trip) is added to the graph as there is no existing edges in the graph yet. In 2.2c, the second trajectory is added, but three of the edges from the trajectory are not added as there exist similar edges already in the graph, which increases the weights for those edges in the graph. Lastly, in 2.2d, the third trajectory is added in its entirety into the graph, as no similar edges exist in the graph. [2]

these edges being pruned later on. Lastly, figure 2.2d shows how the third trajectory is added in its entirety as well, as the current map does not contain similar enough edges. After this process, edges with low weights are pruned, resulting in the inferred map.

2.2.4 Kernel Density Estimation Map Inferencing Methods

Kernel Density Estimation, or *KDE* for short, consists of a three step process. Firstly, a kernel density estimate is constructed from the GPS probe traces by discritizing the map into a two-dimensional grid of cells. For each cell, we count the number of trajectories, whether it be edges or points, that fall into the grid cell, resulting in a two-dimensional histogram. Gaussian smoothing is applied to the histogram to remove potential gaps in the histogram. The histogram is thresholded, creating a binary image, separating the foreground, or the road network in this case, with the background. Using a contour extraction method, the road outline is inferred. Then, a Voronoi graph is constructed from the contours, giving us the road centerlines. An additional step is to infer the road directionality by analyzing the underlying trajectories headings. This then results in a final inferred SD map.

2.2.5 Other Map Inferencing Methods

[19], a paper from 2017, introduced a two-phase clustering algorithm named *Kharita*, which, similarly to k-means methods, reduces the number of points by finding cluster centroids, and then connects them using edges. This paper introduces both an online and offline variant of *Kharita*, where online implies that the inferred map can be updated when new GPS trajectories are recorded.

[10] introduced a map inference algorithm called *RoadRunner*, which builds a map in an iterative two-stage process from GPS probe data. It uses *tracing*, which adds new roads to the inferred map, and *merging*, which decides which existing road segments to merge together.

2.2.6 Comparison

[1] evaluated three classical methods for GPS map inference for each of the aforementioned approaches, namely a *k-means* based method by Edelkamp and Schrodl [5], a *trace-merging* method by Cao et al [2] and an adjusted version of a *Kernel Density Estimation* (KDE) based method by Davies et al [3]. The methods were evaluated in high and low-error sampled GPS data on the University of Illinois at Chicago (UIC) dataset, which contains GPS traces from university campus shuttle busses. The high-error sampled GPS data comes from areas with many tall buildings, whereas the low-error sampled GPS data comes from flatter areas of the map. In previous research, evaluation metrics were mostly computed by comparing the shortest path between a set of nodes in the inferred and ground truth maps. However, [1] found this to be an unreliable metric, and introduced the *TOPO* metric for comparing ground truth maps with inferred maps, a way of comparing the geometric and topological similarity of maps. From their results, the KDE-based map inference method from Davies et al produced the most high-quality maps. The trace-merging approach performed well on the low-error sampled GPS data, but poorly on high-error sampled GPS data, as the raw edges being merged also have high-error due to GPS noise. The k-means based approach produced better results on high-error sampled GPS data than the trace-merging approach, however, still adding many spurious roads due to the GPS noise.

[12] from 2012, evaluated the robustness of several classical map inferencing methods when used on sparse GPS data. The paper compares a KDE and a Trace Clustering algorithm (TC1) on sparse GPS data, and finds that sparse GPS data results in issues with existing map inference algorithms, motivating the need for more data for map inference.

2.3 Map Matching

Map-Matching deals with aligning GPS trajectories with the road network in a digital map. Rather than assuming no prior knowledge and inferring a map, as with *map inferencing*, *map-matching* has this prior knowledge in the form of a digital map, which is more similar to the problem of *map invalidation*. Due to sensor noise, this can be a fairly challenging task. We'll discuss three different classes of map-matching algorithms, namely local, global and statistical methods.

2.3.1 Local Map-Matching Methods

Local map-matching algorithms find a match based on individual GPS trajectory points, ignoring the rest of the trajectory. One such example of of [8], which leverages both the GPS coordinates and heading to find a matching road segment. Due to GPS noise and road intersections, matching a GPS to its nearest road segment is flawed. Figure 2.3 illustrates how this could fail. The road network is illustrated by the solid black lines, and the black dots show a GPS trajectory. By manually inspecting the trajectory, one could induce that the driver traversed road B^0, then taking a left of B^1. However, points P^4, P^5, P^6 are matched to incorrect road segments. [8] improved upon this by taking the GPS heading into account. Other local map-matching algorithms increase the context by not only looking at a single GPS point in a trajectory, but also by considering what past points have been matched with. Such methods are also referred to as *incremental* methods. However, these methods still remain local map-matching algorithms. Local map-matching methods perform fairly well on high-sample GPS trajectories, however, *arc-skipping* becomes a problem when working with low-sample GPS trajectories, where some road segments are skipped entirely when matching GPS points to the map.

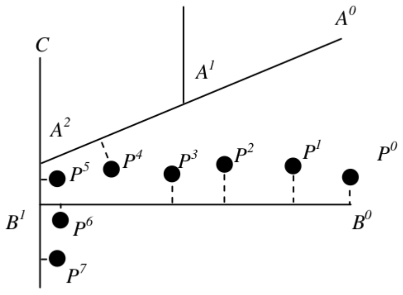


Figure 2.3: An example of when map-matching GPS points to the nearest road segment would fail [8]

2.3.2 Global Map-Matching Methods

Global map-matching methods aim to correlate the entire GPS trajectory to road segments. Local map-matching methods may fail in certain cases, even when taking heading into account. As seen in figure 2.4, the point p_b could be incorrectly matched to a road parallel to the road segments on which p_a and p_c lie. However, global methods which consider the entire trajectory could deduce that p_b likely traversed the same road as p_a and p_c. Many global map-matching methods use the *Fréchet* distance between the trajectory and the possible underlying paths on the road segments. The Fréchet distance is formally defined as follows:

$$d_{fréchet} = \underset{\alpha,\beta}{min} \underset{t\in[0,1]}{max} d\big(p_1(\alpha(t)), p_2(\beta(t))\big)$$

The Fréchet distance tackles a problem that can also be observed in figure 2.3. Besides the fact that P^4, P^5, P^6 are matched to the wrong segments, they also appear out of order. It would suggest that the vehicle went from segment B^0 at point P^3 to segment P^4 on segment A^2, before moving to P^5 on B', and then back to P^6 on segment B^0. The Fréchet distance can be described as having a person walking along the trajectory, holding the leash to a dog that can only walk on the road segment. The Fréchet distance is the minimum length of the leash necessary to for both the person and dog to walk along their respective curves, with the restriction that they cannot move backwards. α and β are reparametrizations of how the person and the dog in our analogy walk along their curves. These functions are restricted to be monotonically non-decreasing, meaning they cannot move backwards, but can stand still along their path. The *weak Fréchet* distance is the same, but removing the restriction that the reparametrizations must be monotonically non-decreasing. Fréchet distance methods then aim to find the path on the road segment with the minimum Fréchet distance.

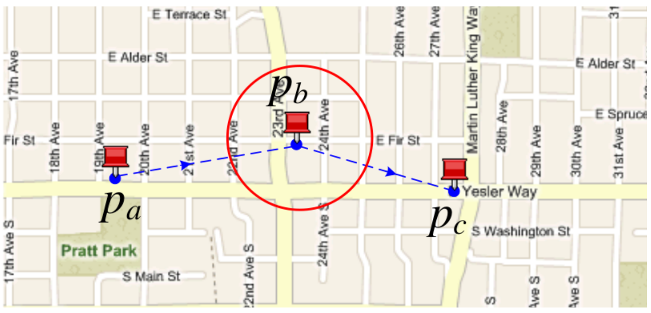


Figure 2.4: Example of when local map-matching methods would fail [13]

2.3.3 Statistical Map-Matching Methods

Statistical models can also be used for GPS map-matching. One such example is from Newson et al [14], which utilized a Hidden Markov Model (HMM) to find the most likely path given a sequence of observations. An HMM map-matching model treats possible points on nearby edges/roads as the hidden states for each real observation, which are the observed GPS points of the trajectory. Then, for every pair of observation and possible hidden states, an emission probability is calculated, which estimates the likelihood for a given GPS observations (z_t) to actually be on some road (r_i). The emission probability is defined as follows:

$$p(z_t|r_i) = \frac{1}{\sqrt{2\pi}\sigma_z} e^{-0.5\left(\frac{||z_t - x_{t,i}||_{greatcircle}}{\sigma_z}\right)} \tag{2.1}$$

This emission probability models the GPS error as Gaussian noise, where the $x_{t,i}$ is the point on road r_i closest to the observation z_t. The *great circle* distance measures the distance between two points, taking into consideration the curvature of the earth.

Additionally, the likelihood to transition from one hidden state to another is known as the transition probability.

$$p(d_t) = \frac{1}{\beta} e^{\frac{-d_t}{\beta}}$$

$$d_t = \Big| ||z_t - z_{t+1}||_{greatcircle} - ||x_{t,i^*} - x_{t+1,j^*}||_{route} \Big|$$

The core concept is that d_t measures the difference between the great circle distance of subsequent observations (z_t and z_{t+1}, and the shortest path distance in the available map from a pair of hidden states. The pair of projected points on the map with the most similar shortest path distance to the great circle distance is the most likely path that the trajectory took. After defining these two probabilities, the Viterbi algorithm is used to compute the best path through the HMM.

2.3.4 Other Map-Matching Methods

[13] is a global map-matching method which uses both spatial and temporal analysis functions to create a *ST function*, which measures the probability of moving from successive candidate points. The main idea is to sequentially, for each GPS point in the trajectory, the most suitable candidate point based on the ST function.

Firstly, the paper defines the *spatial analysis function* F_s as follows:

$$F_s(c_{t-1}^t \rightarrow c_i^s) = N(c_{i-1}^t \rightarrow c_i^s), 2 \leq i \leq n$$

where c_t^t is the t-th candidate point corresponding to the i-th GPS point p_i of some trajectory. $N(c_i^t)$ is a normal distribution with parameters $\mu = 0, \sigma = 20m$, defined as the observation probability. V is the *transmission probability*, describing the probability of transitioning from the previously selected candidate point to the current candidate points. Combining these functions, we get the spatial analysis function above.

Secondly, the *temporal analysis function* computes the cosine similarity between the induced vehicle speed and the speed limit associated to the road segments, resulting in the function F_t.

The *ST function* is then defined by simply multiplying the two:

$$F(c_{i-1}^t \rightarrow c_i^s) = F_s(c_{i-1}^t \rightarrow c_i^s) \times F_t(c_{i-1}^t \rightarrow c_i^s), 2 \leq i \leq n$$

The *ST function* is utilized by finding the path sequence with the highest ST-function value.

2.4 Synthetic Datasets

Datasets can serve as a useful benchmark to allow further research to develop. Datasets such as *ImageNet*, *MS-COCO*, *KITTI* and *ShapeNet* allow researchers to reliably compare performance of different algorithms. Synthetic datasets, such as Virtual KITTI, based on the original KITTI dataset, provide synthetic sensor data from virtual cars in a video game environment [6]. Because the data is synthesized, this allow for much more control when constructing the dataset. Even though closing the gap between real and simulated data can be challenging, Virtual KITTI provides a dataset for evaluation of existing models. CARLA is an open-source driving simulator for autonomous driving research, built on-top of Unreal Engine 4 [4]. It addressed the shortcomings of other car driving simulators and enabled a plethora of research in autonomous driving. These examples illustrate the important of a dataset to allow further research to progress, highlighting the potential a reliable GPS probe trace dataset could have on the task of map invalidation.

A paper from 2020, *GeoSkylines*, evaluated the realism of *Cities: Skylines* traffic API, by recreating two real-world cities in the game, and comparing the traffic congestion between the real and fake cities [17]. The traffic congested road segments in the game were compared to real traffic data from the Environmental Systems Research Institute (ESRI), and discovered a qualitative similarity. This showcases the possibility of using *Cities: Skylines* to generate synthesized GPS probe data, with the added benefit of having full control of the game's state to make changes for map invalidation.

The current state of GPS probe trace datasets are limited to a few, one of the more popular datasets being the University of Illinois at Chicago (UIC) shuttle bus dataset.

Chapter 3

Method

In this chapter, we will introduce the notation to be used and describe the dataset structure. Afterwards, in section 3.3, the three methods to be compared are explained in more detail, including the purpose-built map invalidation method. Lastly, in section 3.5 the experimental setup is described to test the robustness of the different methods on a variety of trajectory augmentations.

3.1 Map Invalidation Problem Definition

Now we are ready to define our task of *map invalidation*. Simply put, map invalidation aims to identify portions of an existing map which may have become invalid due to changes in the road network, by analyzing recent trajectory data. Figure 3.1 shows the expected input for map invalidation algorithms, and what they aim to output. The input consists of the old map, as well as recent trajectories, where the trajectories are used to inform the algorithm on which roads are infrequently traversed and may have become invalid. The output should be a map, where every edge is assigned weights expressing the confidence of an edge being removed.

In the rest of the paper, more formal notation will be used to describe maps, trajectories and timed-points, as well as the structure of our dataset, which are listed in table 3.1. For map invalidation, two snapshots are made of the map at two different points in time, with changes having occurred in the road network between these two snapshots. A snapshot is defined as $S = (G, T)$, which contains the map as a graph (G) and a set of trajectories (T). The first and second snapshots are denoted as S and S' respectively. Consequently, each snapshot consists of a map as a graph (G, G') as well as a set of trajectories (T, T').

The map, described as a graph, contains vertices and edges. Every vertex consists of a location, or coordinates, commonly latitude and longitude: $v_i = (lat, lon)$. The edges describe which nodes are connected: $e_i = (v_j, v_k)$. Trajectories $(t_i \in T)$ are defined as a sequence of chronologically timed points $t^i = (p^i_1, ...p^i_{|t^i|})$, where $|t^i|$ is the length of the trajectory t^i. Every timed point contains a coordinates describing its location, timestamp and commonly a heading as well: $p^i_j = (lat, lon, timestamp, heading)$. Timed points can also include speed and elevation, amongst other variables.

Now that the formal notation regarding maps and trajectories have been defined, we will formally define map invalidation. As visualized in figure 3.1, the input to a map invalidation algorithm is the old map, G, and the more recent trajectories from the second snapshot T', with the output being a set of confidence scores for every edge. Given a map invalidation method f, the problem formulation is as follows: $f(G, T') \rightarrow scores_f$. $scores_f$ contains the scores for every edge in the graph G: $scores_f(e_i) = s_{e_i}$. From these scores, a thresholding function can be used to definitively classify each edge or road as removed or remaining, outputting the final predicted map G'.

Table 3.1: Notation for Map Invalidation

Notation	**Description**
$S = (G, T)$	A single snapshot of a map as a graph (G) and the trajectories (T).
G, G'	The road network before (G) and after changes have occurred (G'), where the road network is described as a graph with vertices and edges $G = (V, E)$
v_i	The i^{th} vertex of a map, containing latitude and longitude: $v_i = (lat, lon)$
$e_i = (v_j, v_k)$	An edge of the map, described by the two vertices it connects.
T, T'	The trajectories from before (T) and after change have occurred (T'), where each trajectory is a sequence of chronologically timed points.
t^i	The i^{th} trajectory of a set of trajectories, with a length $\|t^i\|$. Each trajectory contains a sequence of chronologically timed points: $t^i = (p^i_1, ...p^i_{\|t^i\|})$
p^i_j	The j^{th} timed point of the i^{th} trajectory. A single timed point contains at least latitude, longitude, and commonly a heading and timestamp: $p^i_j = (lat, lon, timestamp, heading)$

Although we define the problem of map invalidation formally above, the input to a map invalidation algorithm does not have to be restricted to the (G, T'). One problem in map invalidation with GPS probe traces is the *coverage problem*, where some roads are not traversed by any trajectories in either T or T', but do still remain. An alternative description of map invalidation can be used, which includes the trajectories from the first snapshot: $f(G, T, T') \rightarrow scores_f$. By adding a third class, *unknown*, to *removed* and *remaining*, this can be turned into a three-class classification problem that attempts to better solve the *coverage problem*.

3.2 Dataset: Synthetic Highways

Now that the problem of *map invalidation* has been formalized, we can introduce the dataset: *Synthetic Highways*. *Synthetic Highways* is a synthetic GPS probe trace dataset for SD map invalidation. The data is collected from the city building game Cities: Skylines, a video game where the player aims to build and manage a functional city, managing everything from zoning, public services, taxes, and infrastructure. The cities in game are populated by *cims*, the digital citizens in game which each have a profile defining their occupation, age, and their home address. The cims make commutes on a daily basis in vehicles, and the synthetic GPS data is collected from vehicle locations in the game. *Cities: Skylines* has a large *modding* community: people who voluntarily add in new features into the game in the form of game *modifications*, or *mods* for short. This is possible due to the strong modding support of the game, providing a specialized *modding API* for modders to more easily build mods for the game. Using the modding API, a custom mod has been built under the same name as the dataset *Synthetic Highways*, which is capable of collecting snapshots from maps in the game, as well as automatically making changes to the road network. All of the code for the mod is publicly available on Github. An example of what the snapshots look like, as described in 3.1, for a tile of the dataset is shown in figure 3.2.

To make the dataset as realistic as possible, the raw data extracted from the game goes through several processing phases. Firstly, the game coordinates are Euclidean, whereas most real-world GPS data use some spherical coordinates system using latitudes and longitudes, such as *WGS84*. Therefore, the game coordinates are projected to latitude and longitude, specifically, *WGS84*, using a reference point in Amsterdam, the Netherlands: (52.356758, 4.894004). This does introduce an error, as these 2D game coordinates do not live on a spherical coordinate system, and so projecting them to latitude and longitude does not translate perfectly.

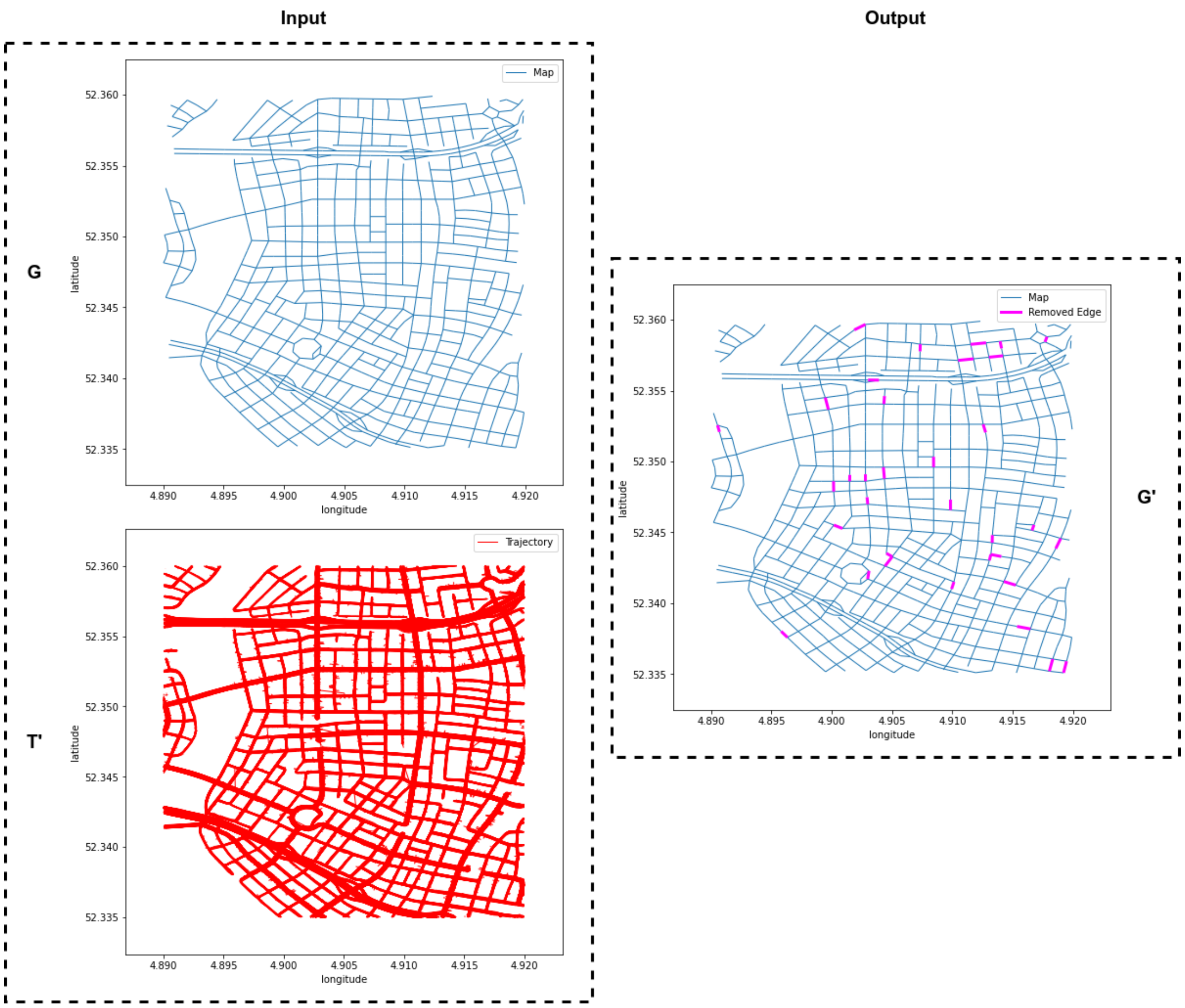


Figure 3.1: Diagram explaining the problem of map invalidation, $f(G, T') \to G'$, in terms of input and output. The input G is the old map, and T' are the more recent trajectories after some changes have occurred which are not reflected in G. Some algorithm takes in these two as input, and the output is supposed to be the predicted map G' with inferred changes.

However, due to the relatively small scale of maps in the game, this error becomes negligible. Additionally, some trajectories exported using the mod are actually multiple trajectories that are concatenated, and must be split up. This is done by splitting the trajectories using a threshold on the displacement between each subsequent timed-points. Figure 3.5 shows the training patch before and after splitting. As can be seen, many trajectories are stringed together before cleaning in figure 3.5a, resulting in artifacts with many of the trajectories cutting straight through the map. After splitting up the trajectories as described above, the trajectories look much cleaner in figure 3.5b.

The dataset comes with four maps with a set of snapshots for each. These maps are visualized in 3.3. However, due the runtime of some of the methods, as well as difficulties in getting maps without any coverage issues, the dataset will formally be defined over several tiles defined by bounding boxes of the maps. These are shown in figure 3.4. The statistics on these patches are listed in table 3.2.

Although the dataset is fairly limited in size, the provided mod can be used to scrape more data if necessary. There are freely available community built maps for *Cities: Skylines* for which data can be collected.

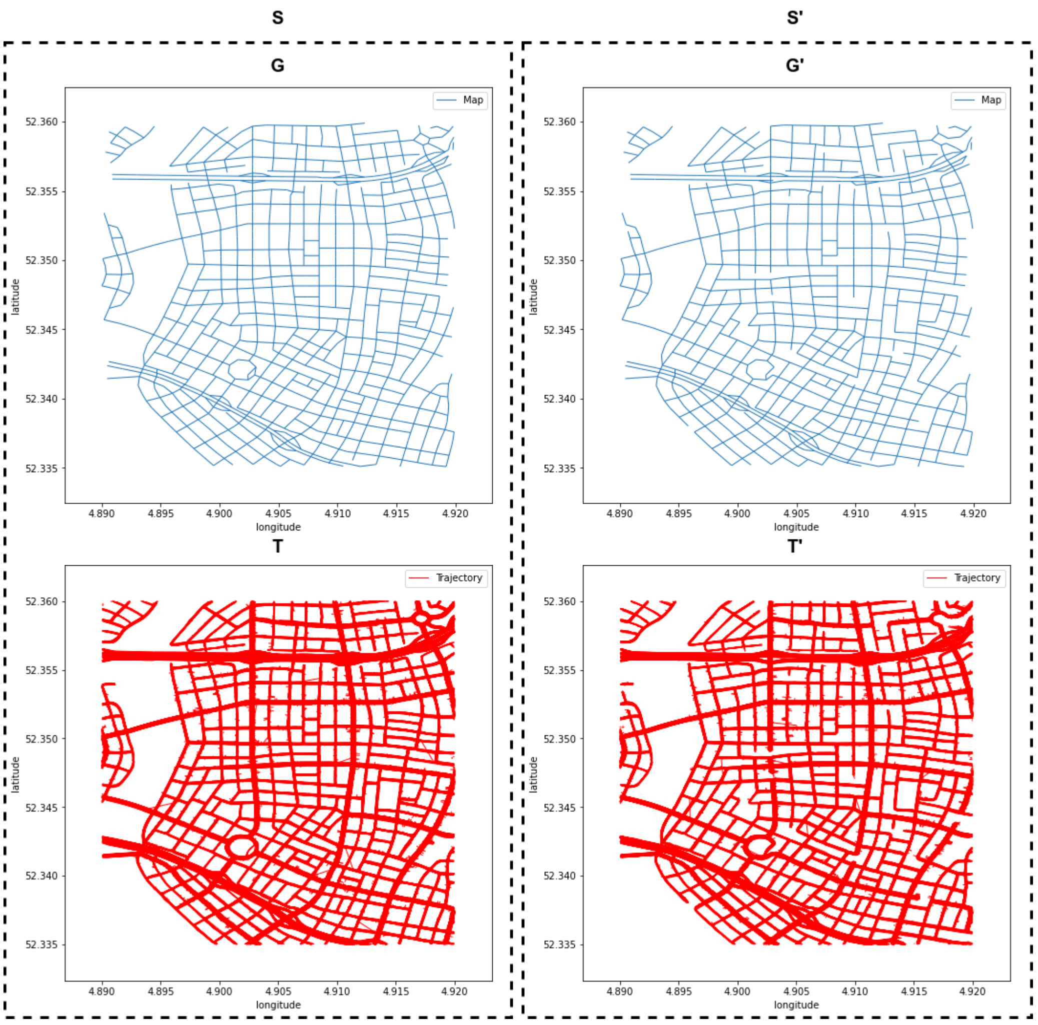


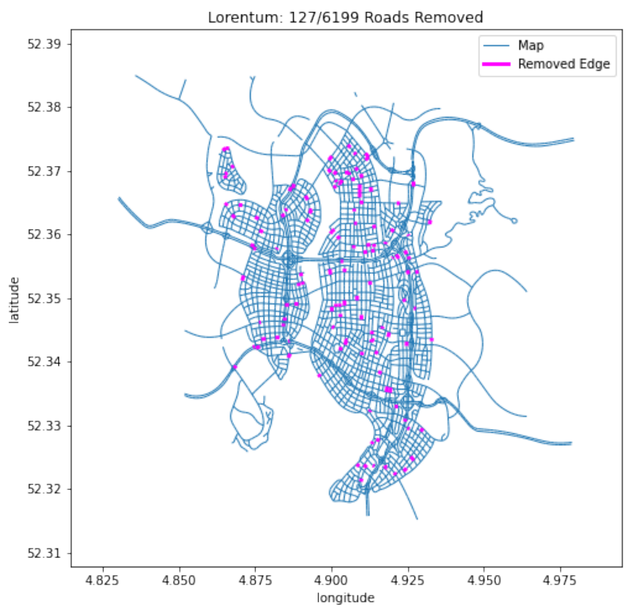
Figure 3.2: An example of the two synthetic snapshots $S = (G, T)$ and $S' = (G', T')$ from the *Synthetic Highways* dataset.

3.2.1 Noise

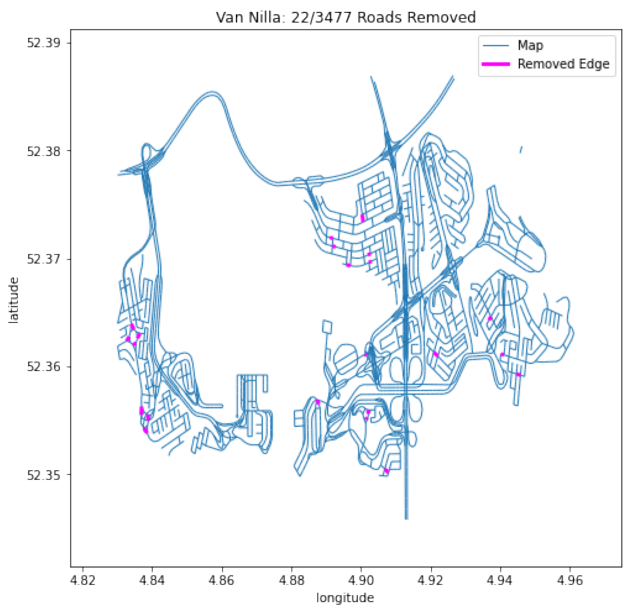
The synthetic GPS probe data, without preprocessing, contains no noise. This is unrealistic as GPS signal noise is prominent in real GPS probe data. In the data preprocessing steps, one important aspect is to add noise to the synthetic probe data, allowing in-depth analysis of robustness of models to increasing noise. A simple noise model adds random deviations at each point in a trajectory, independent on noise from the previous points. However, GPS signal noise is often correlated and not as sporadic, due to the presence of *Intertial Measurement Units* or *IMUs* which makes use of accelerometers and gyroscopes to infer a vehicle's position between GPS timed points based on its inertia. The *Ornstein-Uhlenbeck* process, a stochastic process simulating a random walk over time, has been found to model GPS noise fairly well [18]. The noise added to the trajectories in this dataset is based on the *Ornstein-Uhlenbeck* process.

Table 3.2: Dataset Statistics

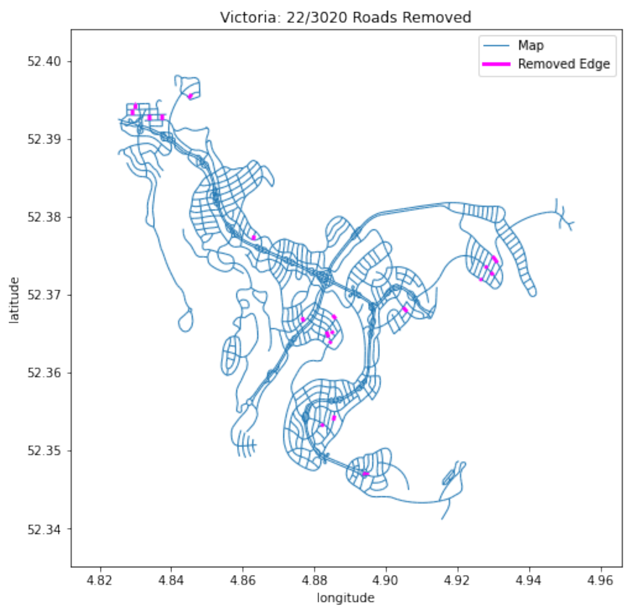
Statistic	**Training Patch**	**Test Patch**
Total number of roads (edges, distance)	1355, $\sim$ 91.8km	1689, $\sim$ 114.5km
Total removed roads (edges, distance)	34, $\sim$ 2.3km	59, $\sim$ 4.2km
Percentage of roads removed	$\sim$ 2.52%	$\sim$ 3.67%
Number of trajectories ($\|T'\|$)	$37,327$	$2,573$



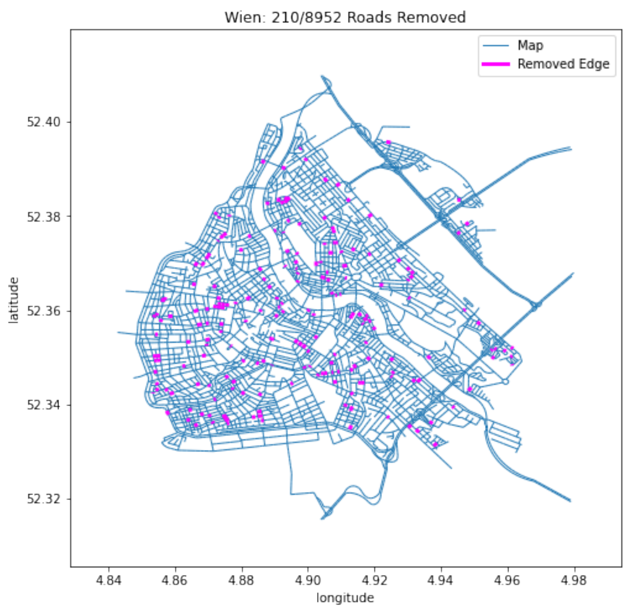
(a)



(b)

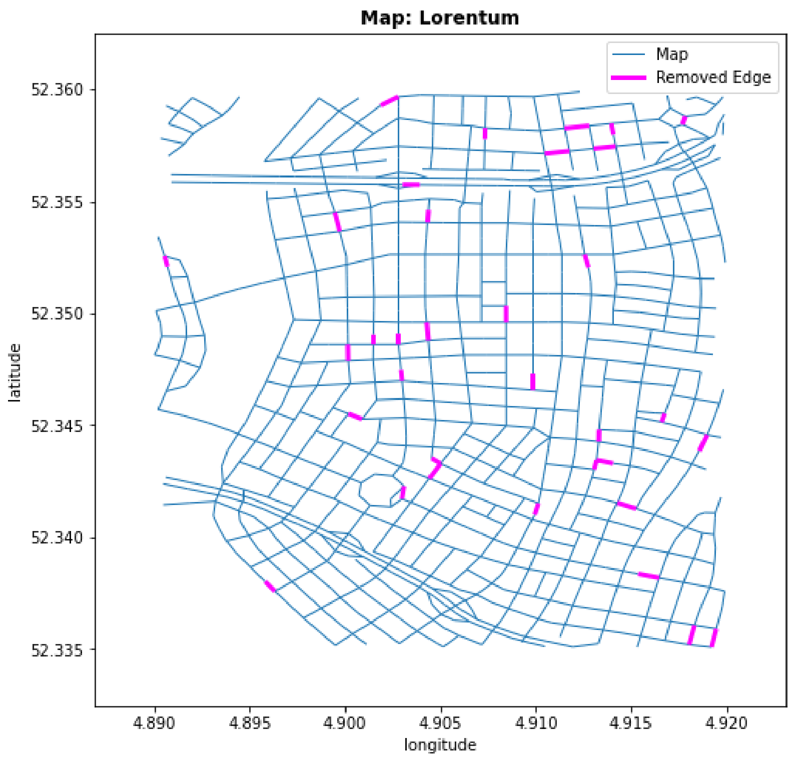


(c)

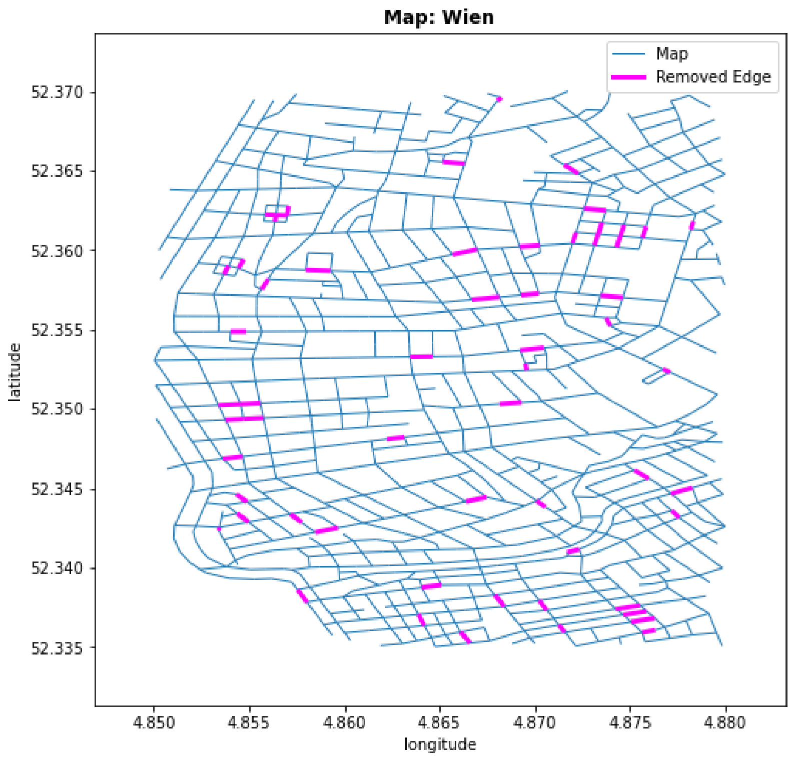


(d)

Figure 3.3: The four maps that come with the dataset. Although the formal dataset is defined in terms of tiles within these maps, other tiles can be used as cross validation tiles.

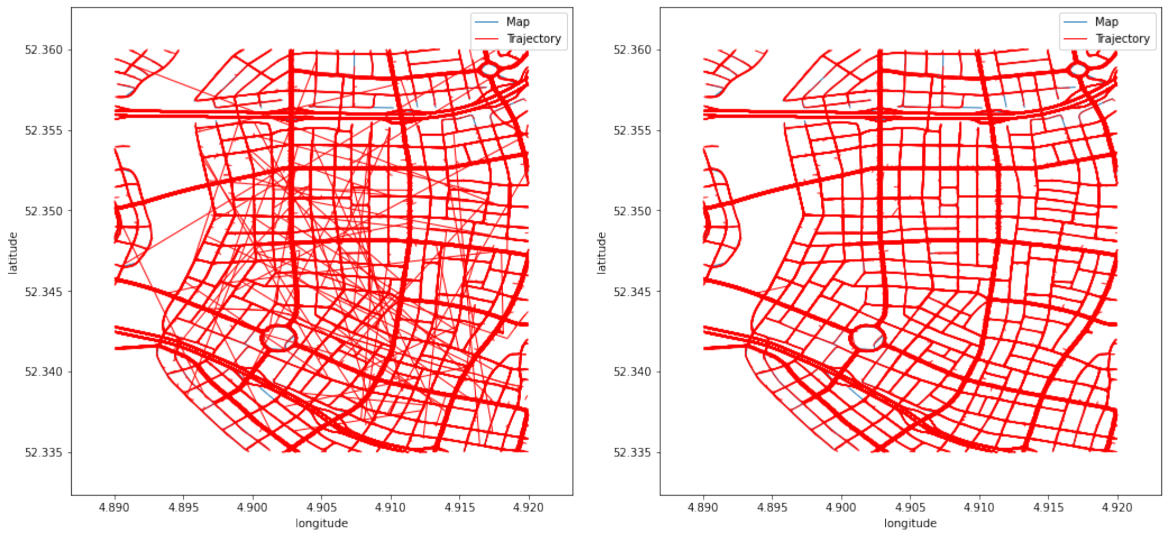


(a) The training patch of the dataset.



(b) The test patch of the dataset.

Figure 3.4



(a) The training patch before cleaning and splitting up the trajectories.

(b) The training patch after cleaning and splitting up the trajectories.

Figure 3.5: Figures showing the difference between the raw trajectories from the game and after some trajectory cleaning and splitting is performed. Due to how the game stores vehicles, the longer the mod runs and collects trajectories, the more trajectories actual separate trajectories will be merged into a single trajectory, causing these artifacts to appear. Splitting up the trajectories based on displacement results in much cleaner trajectories.

3.2.2 Road removal

One of the motivations to create a synthetic dataset for map invalidation is the ability to know exactly what changes have occurred in the map. However, we would like these changes to be made automatically, rather than manually for every map. The mod automatically selects suitable roads to be removed in between the snapshots. It is important to note that not just any subset of roads can be selected for removal, as this could possibly break the road network into two disjoint subgraphs. Therefore, the mod selects roads for removal according to a reachability criteria: only remove a set of roads such that, if all of the selected roads are removed, all nodes in the map remain reachable from the same nodes as before the selected roads are removed. This assures that the routing API of vehicles in the game does not break and send vehicles straight through grass to their destination. This criteria is written in *clingo*, an answer set programming (ASP) language, in which such criteria can easily be encoded [7]. The code is available on the Github page, but the ASP program encoding these criteria is also available in the appendix in listings 1.

3.3 Models

To test our dataset, we built a total of three map invalidation methods, excluding one random baseline method. All methods work by assigning a weight for every edge in the original map $e \in G$ based on the trajectories from the second snapshot T', where a higher weight indicates a higher likelihood of road removal, whereas a lower weight indicates a likelihood of the road remaining. The random method serves as a lower-bound baseline. The rule-based and Hidden Markov Model (HMM) methods are essentially map-matching algorithms, where the matching results of the trajectories are used to change the weight of the edges in the map. The histogram method is our custom method built for map invalidation.

3.3.1 Random Method

The first model that is introduced as a lower-bound baseline is a random method. For each edge in the existing map, G, the method prescribes a uniform random score of either 0 or 1, where 0 indicates that the road remains, while a 1 indicates that the road is removed. Note that using a uniform random distribution will results in a class balanced classification by the random method, classifying half of the edges as removed and the other half as remaining, unlike the class imbalance of the map patches.

3.3.2 Map-Matching Based Methods

Two of the map invalidators are based on local and statistical map-matching methods. These methods are similar and only differ in how they perform map-matching. Algorithm 1 describes the general framework for map-matching based map invalidators, where the $MapMatch$ function can be exchanged for any generic map-matcher. The general framework is very simple. For every trajectory, the map-matcher matches it to a path, being a sequence of edges of the map. For every edge in the path, we decrement the score by 1. The reason we decrement the score rather then incrementing, is due to the definition of *Positive* and Negative as defined in table 3.3. A positive edge is an edge that is removed, so a higher score means that the road is removed. By decrementing the edges scores, a lower score will correspond to more likely for the road to be remaining, whereas the higher the score is, or the closer it is to 0, the more likely it is to have been removed.

Algorithm 1 Map-Matching Based Map Invalidators

```
1: $scores \leftarrow \{\}$
2: for each $t \in T$ do
3:     $path \leftarrow MapMatch(t, G)$
4:     for each $e \in path$ do
5:         $scores(e) \leftarrow scores(e) - 1$
6:     end for
7: end for
```

Rule-based Map Matcher

Taking inspiration from local map-matching techniques from [8], a rule-based method was built that assigns each timed point in the trajectories to a single road or edge in the map G, for which it decrements the weight for that edge, indicating it is less likely to be removed. A timed point is matched to an edge as follows:

1. Each timed point, $p^i_{j,k}$, considers the n nearest edges $e_1 \cdots e_n$ using a *KD-Tree*.
2. The distance to the closest points on each of the n edges to the timed point is computed, and a distance score is calculated. For example, an edge defined by vertices $e_n = (v_{n1}, v_{n2})$, with the points $lat_1, lon_1 = v_{n1}$ and $lat_2, lon_2 = v_{n2}$, and a timed point with the coordinates p_{lat}, p_{lon}, the distance score is computed as follows:

$$score_d = \frac{\left|(lon_2 - lon_1)(lat_1 - p_{lat}) - (lon_1 - p_{lon})(lat_2 - lat_1)\right|}{\sqrt{(lon_2 - lon_1)^2 + (lat_2 - lat_1)^2}} \tag{3.1}$$

3. A similarity score based on heading of the timed point and the n edges is calculated. For an edge with heading e_h and a timed point with heading p_h, the heading score is calculated as follows:

$$score_h = (e_h - p_h) \,\%\, 360 \tag{3.2}$$

$$score_h = \begin{cases} score_h - 360, & score_h \geq 180 \\ score_h, & \text{otherwise} \end{cases} \tag{3.3}$$

4. The distance and heading similarity scores are combined into a weighted score of the two:

$$score = (C_d * score_d) + (C_h * score_h) \tag{3.4}$$

5. The edge with the lowest score is the selected edge for the timed point

HMM Map Matcher

Similarly to the rule-based map matcher, the Hidden Markov Model (HMM) map matcher also map matches trajectories to edges and decrements their weight. However, HMM models do so globally and statistically, ie. for the entire trajectory and calculating the probability of multiple possible paths the trajectory could have taken, as explained in 2.3.3. An existing HMM implementation is used [20] which is based on the original HMM map matching paper by [14].

3.3.3 Histogram Method

The final map invalidation method is our purpose-built map invalidator, called the histogram method. The histogram method takes inspiration from *map inferencing* techniques, specifically KDE based map inferencing approaches, as describe in 2.2.4, which first discretizes the space into a grid and uses trajectories to produce a 2D histogram. The complete algorithm is described in algorithm 2, and figure 3.6a visually explains the histogram method's pipeline. The histogram method starts by producing a 2D histogram from the trajectories, by filling in the histogram for every trajectory. This is performed using the *EdgeToHist* function, which sets cell values to -1 in the presence of a trajectory that passes through it. This step of the histogram algorithm uses *Bresenham's line algorithm* to fill in the cells between trajectory timed-points. One *EdgeToHist* has been run for every trajectory, this results in the histogram shown in the middle image in figure 3.6a. Finally, to compute the scores for every edge, the *ExtractScores* function is called, which accumulates the cell scores in the histogram between edge nodes. If there are no trajectories present between two nodes of an edge, this means it is likely not traversed, which results in a relatively high score for said edge. The last image shows a heatmap colored according to the resulting weights assigned by the Histogram method after running *ExtractScores*, where roads with a high score, or roads that have a high likelihood of being invalid, are red colored while roads that are likely to be remaining are colored yellow. Figure 3.6b shows the ground-truth of the map tile used in the pipeline example, highlighting the removed roads in magenta. Note how the removed roads correspond well with the roads that are highlighted red by the algorithm in the final heatmap of figure 3.6a.

3.4 Metrics

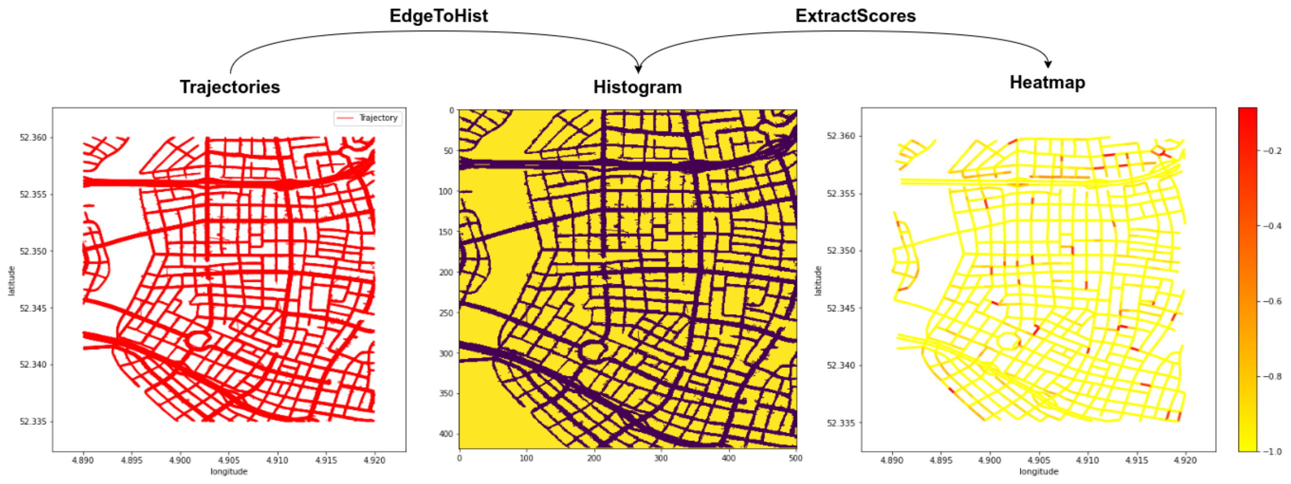
To evaluate the performance of the different map invalidation methods, we use precision-recall curves, area-under the precision-recall curve (PR-AUC) as well as the F1-Score of the method

Algorithm 2 Histogram Map Invalidator Algorithm

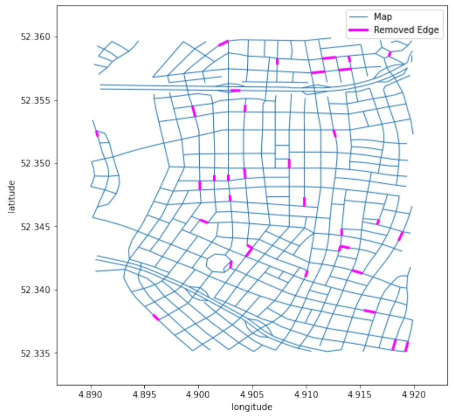
```
 1: $H \leftarrow InitializeHistogram(cellresolution)$
 2: for each $t \in T$ do
 3:     $i \leftarrow 0$
 4:     for each $p \in t$ do
 5:         if $i == 0$ then
 6:             $i \leftarrow i + 1$
 7:             skip
 8:         end if
 9:         $edge \leftarrow (t[i-1], p)$
10:         $H \leftarrow EdgeToHist(H, edge)$
11:     end for
12: end for
13: $scores \leftarrow ExtractScores(G, H)$

 1: function EdgeToHist(H, edge)
 2:     $p_1, p_2 \leftarrow edge$
 3:     $c_1 \leftarrow discretize(H, p_1)$
 4:     $c_2 \leftarrow discretize(H, p_2)$
 5:     $C \leftarrow bresenham(c_1, c_2)$
 6:     for each $c \in C$ do
 7:         $H(c) \leftarrow -1$
 8:     end for
 9:     return $H$
10: end function

 1: function ExtractScores(G, H)
 2:     $scores \leftarrow \{\}$
 3:     for each $edge \in G$ do
 4:         $p_1, p_2 \leftarrow edge$
 5:         $c_1 \leftarrow discretize(H, p_1)$
 6:         $c_2 \leftarrow discretize(H, p_2)$
 7:         $C \leftarrow bresenham(c_1, c_2)$
 8:         $score \leftarrow 0$
 9:         for each $c \in C$ do
10:             $score \leftarrow score + H(c)$
11:         end for
12:         $score \leftarrow \frac{score}{|C|}$
13:         $scores[edge] \leftarrow score$
14:     end for
15:     return $scores$
16: end function
```



(a) Histogram pipeline visually explained on a map tile. The first image shows the trajectories. The second image is after the *EdgeToHist* is run on every trajectory, resulting in a 2D histogram. Finally, the heatmap at the end visualizes the assigned weights to every edge in the map G from the assigned scores from the histogram after running the *ExtractScores* function.



(b) Ground-truth of the map tile used in the pipeline, with removed roads highlighted in magenta

Figure 3.6: Histogram pipeline example, showing the full pipeline of the algorithm in figure a, and the ground-truth/removed roads in figure b. The result shows that many of the edges highlighted in red as removed roads in the final heatmap of figure a correspond with the removed edges highlighted in magenta in figure b.

Table 3.3: Metrics for Map Invalidation

Notation	Description
TP - True Positives	Removed roads that are correctly classified as removed
TN - True Negatives	Remaining roads that are correctly classified as remaining
FP - False Positives	Removed roads that are misclassified as remaining
FN - False Negatives	Remaining roads that are misclassified as removed
Precision	$\frac{\lvert TP\rvert}{\lvert TP\rvert+\lvert FP\rvert}$
Recall	$\frac{\lvert TP\rvert}{\lvert TP\rvert+\lvert FN\rvert}$
F1-Score	$2\frac{precision \times recall}{precision+recall}$

after thresholding the scores per edge. The reason for using precision recall curves is due to the fact that the Synthetic Highways dataset can be seen as a class-imbalanced dataset. Every edge can be classified as either removed (1) or remaining (0), but there are far less removed roads than there are remaining roads. Therefore, using a regular measure of accuracy would result in methods always classifying a road as remaining to score well, while misclassifying all removed roads. By considering precision-recall curves, and its area under the curve, we can evaluate methods over different thresholds to see the precision-recall trade-off and to identify the best performing methods. Table 3.3 explains how the precision, recall and F1-scores are defined in the context of map invalidation. Positive, in this case, implies road removal, while negative means a road remains.

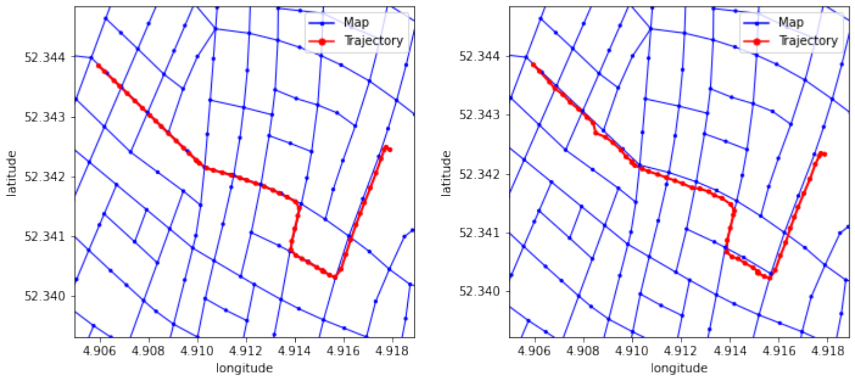
3.5 Experimental Setup

To compare the different methods, we evaluate the PR-AUC and F1-Scores of each method on three different kinds of trajectory augmentations. The raw data collected from the game contains no noise, and is purposely collected with a high sample frequency per vehicle, so that the trajectories can be augmented and downsampled. This allows for detailed analysis of the map invalidators to the different trajectory augmentations. The first is robustness to trajectory noise, where each timed point has added noise. The second experiment is designed to test robustness to trajectory sparsity, ie, how the number of trajectories can impact the methods. Lastly, the third experiment tests robustness to temporal sparsity, which changes the distance and number of timed-points in each trajectory by resampling them with a fixed interval. All of the experiments are run over three seeds, namely seed 42, 142 and 420, and the results are reported as an average over the three seeds. Following is a more in-depth explanation for every trajectory augmentation. The parameters for each experiment are shown in table 3.4.

Evaluation will be performed on a specific bounding box of one of the maps in the dataset. This is due to the *coverage issue* as mentioned in section 3.1, which occurs when parts of the road network are never traversed by any vehicles in the game due. This is due to several factors, such as city design or lack of facilities in certain areas, eliminating the need for the cims to ever traverse certain roads. Although this is also a problem in the real-world, this paper focuses on the noise aspect of trajectories on map invalidation, whereas the *coverage issue* is a different problem to tackle all together. Figure **??** shows the patch to be evaluated from the map *Lorentum*, with the bounding box of (52.335, 52.36, 4.89, 4.92) with the format (min. latitude, max. latitude, min. longitude, max. longitude).

Table 3.4: The parameters used for the three different experiments. First column shows the degrees of noise for the robustness to noise experiment. The second column shows the number of trajectories sampled for the robustness to sparsity experiment, and the third column shows the resampling intervals used for the robustness to temporal sparsity experiment

Noise (degrees for σ, θ)	**Trajectory Sparsity (# of trajectories)**	**Temporal Sparsity (resample interval)**
0	7535	1
1.25E-5	15,070	4
3.75E-5	22,605	7
6.25E-5	30,140	10
8.75E-5	37,676	13
-	-	16



(a) (b)

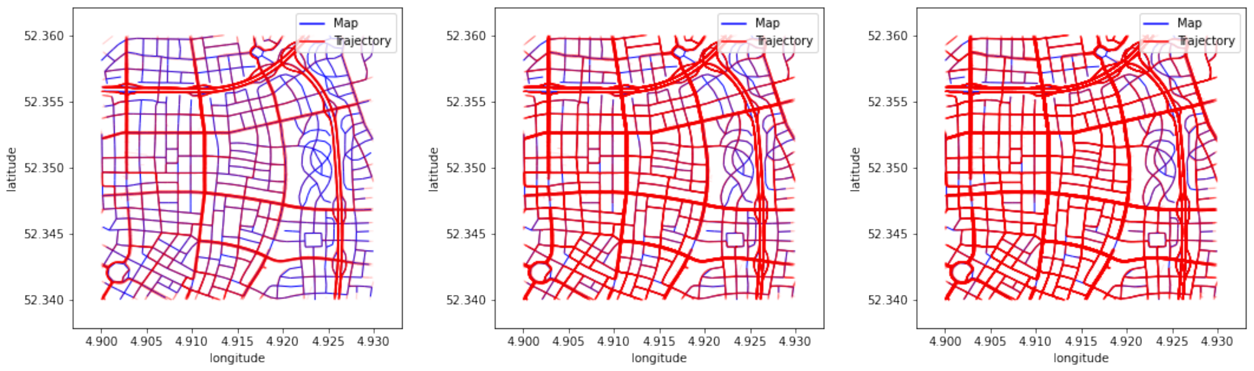
Figure 3.7: An example of a trajectory before 3.7a and after adding *Ornstein-Uhlenbeck* noise 3.7b

3.5.1 Robustness to Noise

The aim of this experiment is to evaluate how well the methods compare to increasing noise levels of the trajectories. The raw trajectories extracted from the game have no noise, as they are directly read from the game coordinates. Using the *Ornstein-Uhlenbeck* process as a noise model, the original trajectories are augmented such that the trajectories better reflect real trajectories. Figure 3.7 shows an example of a trajectory before and after noise has been added. The first column in table 3.4 shows the noise being added in degrees. These parameters define the σ and θ parameters of the *Ornstein-Uhlenbeck* noise model.

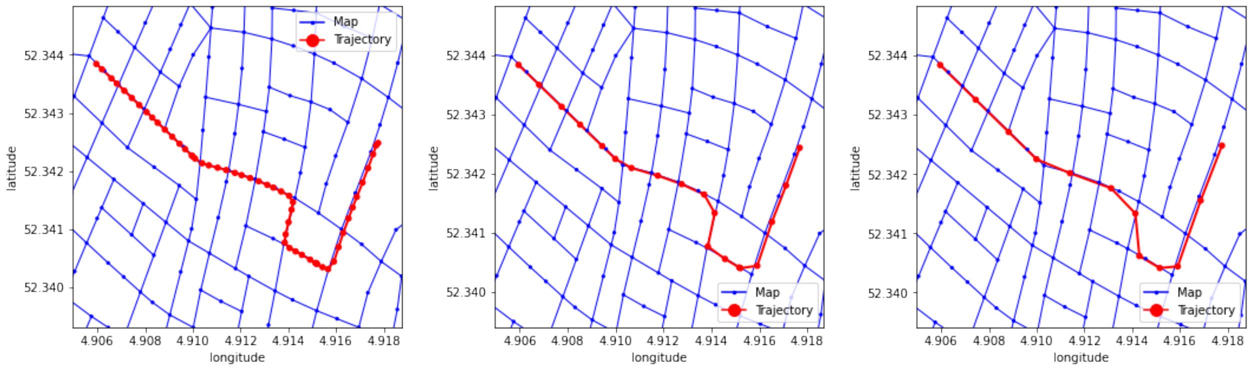
3.5.2 Robustness to Trajectory Sparsity

The dataset contains tens of thousands of trajectories per city. However, in real-life scenarios, trajectories may be more sparse. Thus, this experiment aims to evaluate the robustness of different map invalidation methods' to the number of trajectories available. This is done simply by sampling a subset of the total trajectories available. Figure 3.8 shows an example of a map patch with increasing number of sampled trajectories: $2,000$, $6,000$ and $10,000$ trajectories respectively. The hypothesis is that, with a higher sample, methods should generally perform better, as when trajectories are sparse, they may not traverse all the roads that are still in use, provided that the trajectories contain little noise. The number of trajectories that are sampled is shown in the second column of table 3.4.



(a) 2,000 Trajectories (b) 6,000 Trajectories (c) 10,000 Trajectories

Figure 3.8: Trajectories in a map patch showing an increasing trajectory density/decrease in trajectory sparsity.



(a) Original trajectory (b) Resampling every 3 points (c) Resampling every 5 points

Figure 3.9: An example of a single trajectory with increasing temporal sparsity

3.5.3 Robustness to Temporal Sparsity

Every trajectory in the dataset is sampled with a high frequency, meaning that subsequent timed-points in a trajectory are close to each other. A simple way to synthetically decrease the temporal frequency is to resample timed-points per trajectory in steps, ie, sampling every-n point in a trajectory $n > 1$. This results in trajectories with temporal sparsity, rendering the routes the vehicles took more ambiguous. An example of this can be seen in figure 3.9, where the original trajectory is shown in 3.9a, and the same trajectory is resampled to have increasing temporal sparsity in figures 3.9b and 3.9c. The third column of table 3.4 shows the different resampling intervals used for the experiments.

Chapter 4

Analysis

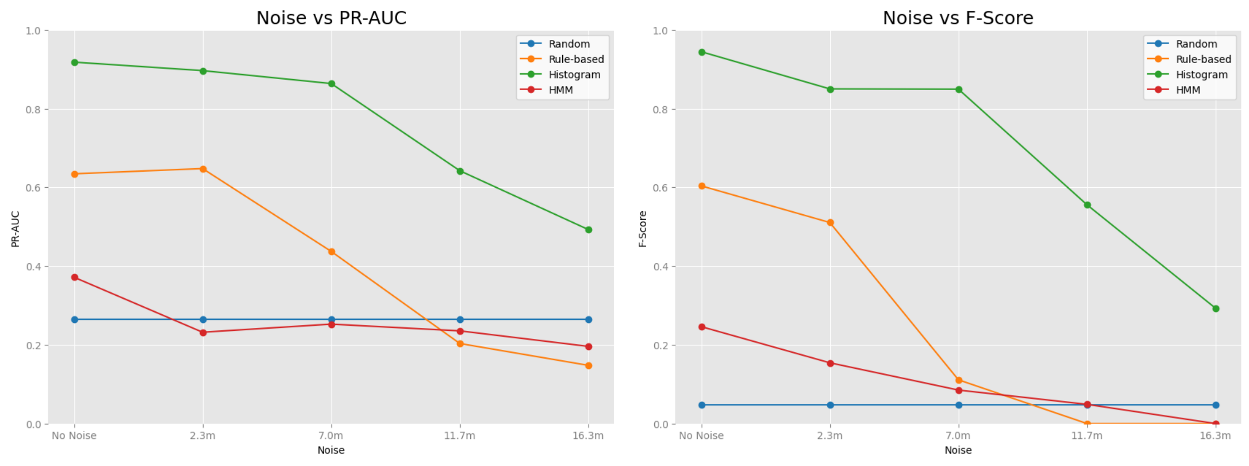
In this chapter, we will present the results of the main experiments on the different kinds of trajectory augmentations discussed in 3.5. Furthermore, additional ablation studies are presented on the two best performing map invalidators, as well as a more in-depth analysis where the map-matching based approaches make mistakes, giving insight into the shortcomings of map-matching based map invalidators.

4.1 Results

The results for the robustness to noise experiments are shown in figure 4.1, with the precision-recall areas under the curves (PR-AUC) shown in the first figure 4.1a and the F1-Score for the methods in figure 4.1b, illustrating how the methods respond to increasing noise levels. Individual precision-recall curves can be found in the appendix Figure 4.2 illustrates the same metrics, PR-AUC and F1-Scores, for the trajectory sparsity experiments. Lastly, figure 4.3 shows the robustness of the methods to increasing levels of temporal sparsity of the trajectories.

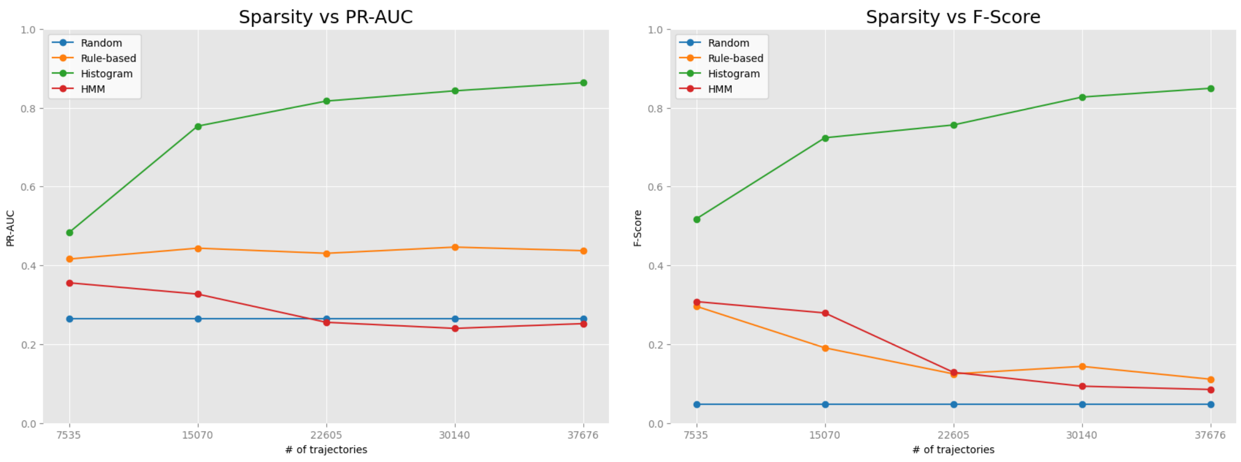
4.2 Evaluation

From the results of the robustness to noise experiments in figure 4.1, there is a clear trend that shows all methods, besides the random baseline method, decrease in performance on both the PR-AUC as well as their F-Scores as the noise is increased. However, the only method that



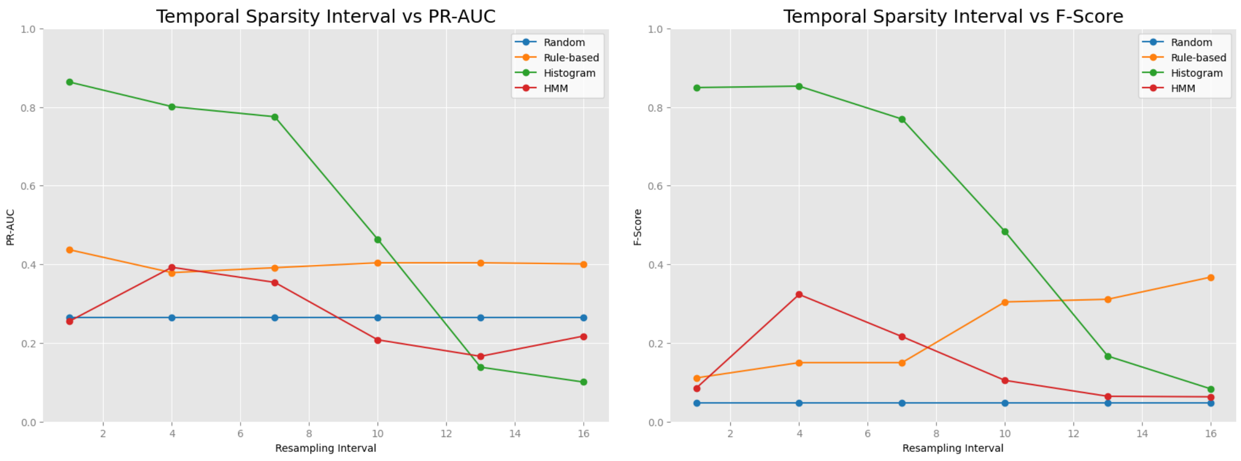
(a) (b)

Figure 4.1: The results of the noise experiments



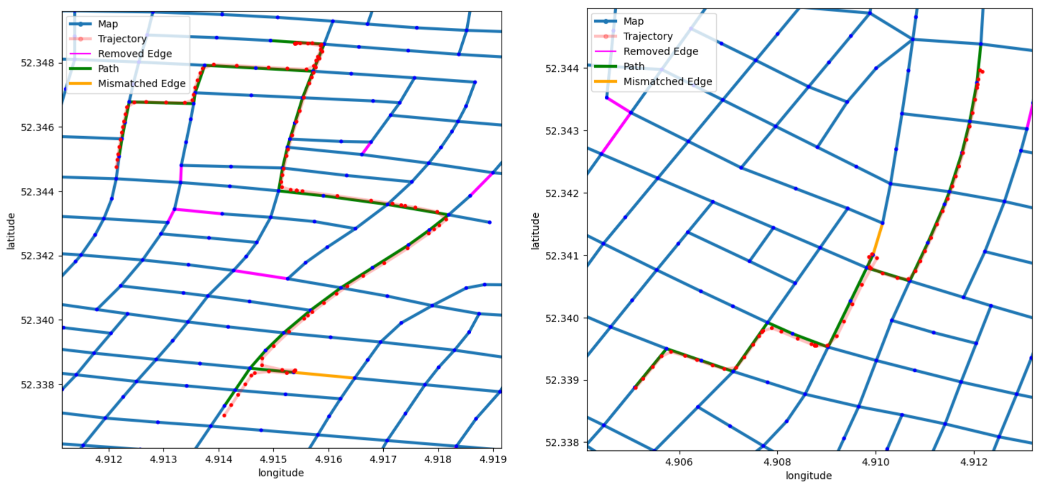
(a) (b)

Figure 4.2: The results of the sparsity experiments



(a) (b)

Figure 4.3: The results of the temporal sparsity experiments



(a) (b)

Figure 4.4: HMM mismatches

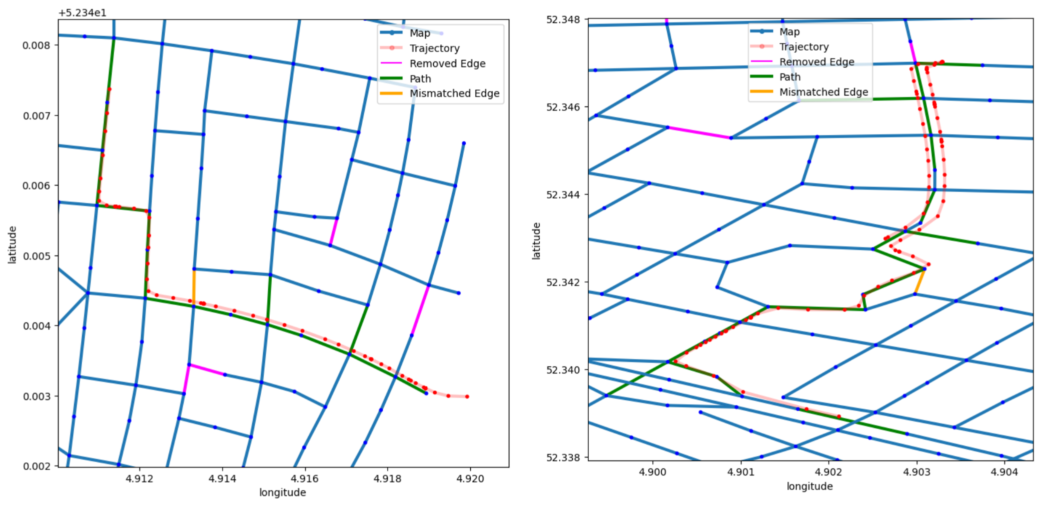
achieves a near perfect score with no-noise added to the trajectories is the histogram method. In both the PR-AUC and F-Scores, the histogram method outperforms all other methods at each noise level. Although the rule-based method is shown to outperform the HMM method in all noise levels in terms of PR-AUC in figure 4.1a, this does not seem to be the case with the F-Scores. A significant drop in performance can be seen in in figure 4.1b at the third noise configuration corresponding to $7.0m$ noise. Surprisingly, the HMM method performed the worst out of all three methods. This seems rather counter-intuitive, as the HMM method uses a more sophisticated map-matcher than the rule-based method.

The results from the robustness to trajectory sparsity experiments show a predictable trend for the histogram method, in figure 4.2. As the trajectory density increases, the histogram method generally achieves a higher PR-AUC as well as a higher F-Score.

The results from the temporal sparsity experiments in figure 4.3 shows that with high temporal density, the histogram method still outperforms the other map-matching based techniques. However, at very high resampling rates, such as 16, the rule-based method starts to outperform the histogram method.

Figures 4.4 and 4.5 show two examples for the HMM and rule-based methods of mismatched trajectories to removed roads, showing some insight as to the mistakes each methods make respectively. Each figure shows a single trajectory in red, and its matched path on the map in green. Removed roads are shown in magenta, however, any removed road that the trajectory is matched to is considered a mismatch, and this is plotted as an orange edge. Both HMM mismatches in figures 4.4a and 4.4b show mismatches in U-turns, when the vehicle moved towards a removed road and turned around before reaching it. U-turns are notoriously difficult to map-match correctly, and often times the trajectories from the dataset move towards a removed road but perform a U-turn, making these cases hard for the HMM matcher to match correctly. This, in turn, has a negative affect on the score for map invalidation as well.

The mismatches made by the rule-based method shows different kinds of mistakes being made, as shown in figures 4.5a and 4.4a. In the case of figure 4.5a, the trajectory is mismatched on adjacent roads quite frequently. Not only does it mismatch to a removed road, but also mismatches to two additional adjacent roads that are not removed, which are clearly not part of the original trajectory. This is also the case in the second example in 4.5b where a mismatch to a removed road can be seen on the roundabout, as well as mismatch to an adjacent road near the top.



(a) (b)

Figure 4.5: Rule-based mismatches

4.3 Ablation Studies

From the previous experiments, it is apparent that our histogram methods generally performs the best on the task of map invalidation compared to the methods based on map-matching. This is rather counter-intuitive, as the method is quite crude and simple, whereas the map-matching based methods score rather poorly. In this section, we perform some ablation studies on the histogram and rule-based methods, the top two performing methods on map invalidation, to identify what components and parameters contribute to their higher scores.

4.3.1 Histogram Method Ablation Studies

The histogram method is a simple method with two main steps. The first being the computation of the histogram itself, the second being the way the edge scores are extracted from the histogram. As seen in the function *EdgeToHist* in algorithm 2 in line 7, where the histogram is constructed, the method only looks for trajectory *occurrences*, meaning a cell in a histogram is set to -1 if there is a trajectory passing through the cell. However, another alternative is to *accumulate* the the number of trajectories passing through each cell, by decrementing the cell by -1 for every trajectory that passes through it. In the score extraction step, for each edge the score is simply the average cell value the edge passes through. This could inflate scores of infrequently travelled roads if they are connected to a busy intersection. The alternative here would be to only consider cells in the center of the edge, rather than everything in between the nodes. This creates four different configurations for which we can run the histogram method with, which are also listed in table 4.1. The results of this experiment are listed in figure 4.6a, where the F-Scores are compared for the different histogram configurations in a bar plot. It illustrates clearly how the performance is significantly degraded when the histogram is constructed by *accumulating* the trajectories per cell. The reason for this drastic decrease in scores is that many removed edges are adjacent to edges that are very frequently traversed, which means that when calculating the scores per edge, the histogram cells near the nodes of a removed edge will have very low values, and thus decreases the eventual edge score significantly. By simply using the *occurence* of a trajectory per cell, this problem is averted and increases the F-Scores. Furthermore, the difference in summing the *center* cells versus all the *intersecting* cells also results in a lower F-Score, although not as significantly. The best configuration is clearly the *Intersect+Accum* configuration.

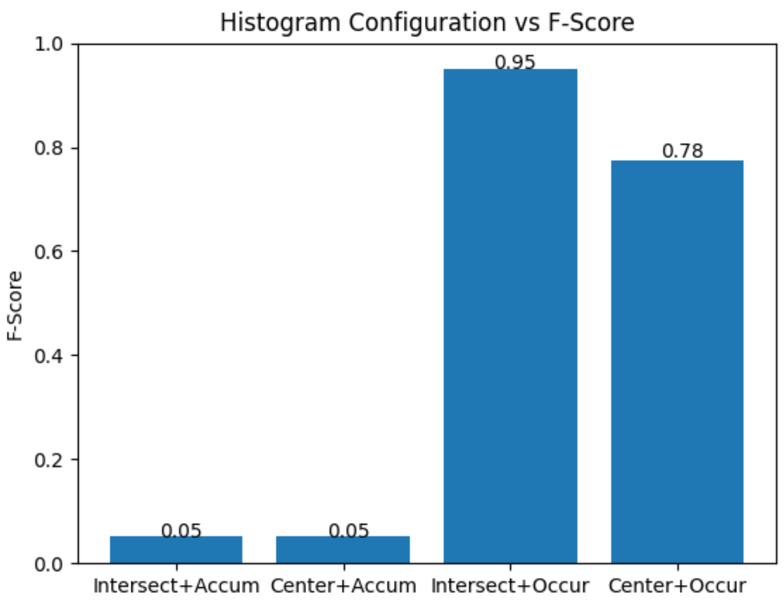
Table 4.1: Histogram Method Configurations

Histogram Configuration	**Description**
Intersect+Accum	Calculating the edge scores by taking the sum of all the cells in the histogram that intersect an edge, and filling in the histogram by accumulating the trajectories per cell by -1 for every trajectory.
Center+Accum	Calculating the edge scores by taking the sum of all the cells in the histogram in the center of the edge, and filling in the histogram by accumulating the trajectories per cell by -1 for every trajectory.
Intersect+Occur	Calculating the edge scores by taking the sum of all the cells in the histogram that intersect an edge, and filling in the histogram by only setting a cell to -1 at the occurrence of a trajectory, regardless of the number of trajectories that intersect the cell.
Center+Occur	Calculating the edge scores by taking the sum of all the cells in the histogram in the center of the edge, and filling in the histogram by only setting a cell to -1 at the occurrence of a trajectory, regardless of the number of trajectories that intersect the cell.

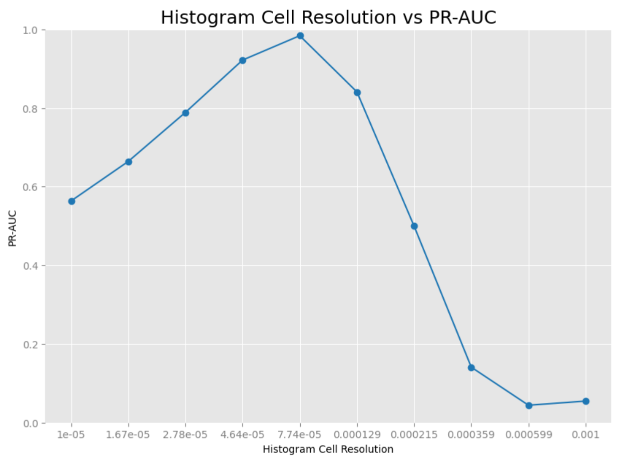
Another essential parameter for the histogram method is the resolution at which the histogram method runs. If the histogram resolution is too small, it may not be able to distinguish nearby roads from each other, resulting in bad scores. On the other hand, making the histogram resolution too big might decrease its performance in high noise scenarios. Ergo, an additional experiment on histogram resolution's impact on performance is performed. Table 4.2 shows what cell resolutions are used for the experiment. The cell resolution indicates how many degrees of latitude or longitude the width and height of a cell in the histogram represents. Thus, the lower the cell resolution, the higher the overall histogram resolution is. The results are plotted in figures 4.6b and 4.6c, showing the PR-AUC and F-Scores over the different cell resolutions. The results show that there is a optimal resolution to achieve a good score using the histogram method. While too low of a cell resolution, or too high of a histogram resolution, decreases the performance, this seems to be not as drastic as the decrease in performance when the cell resolution is too high, or when the histogram resolution is too low. Intuitively, this makes sense as a high cell resolution would result in a low resolution histogram and therefore roads will not be as distinguishable from one another as a high cell resolution would result in. The best performance is achieve when the cell resolution is set to $7.74\text{e}{-5}$.

4.3.2 Rule-based Method Ablation Study

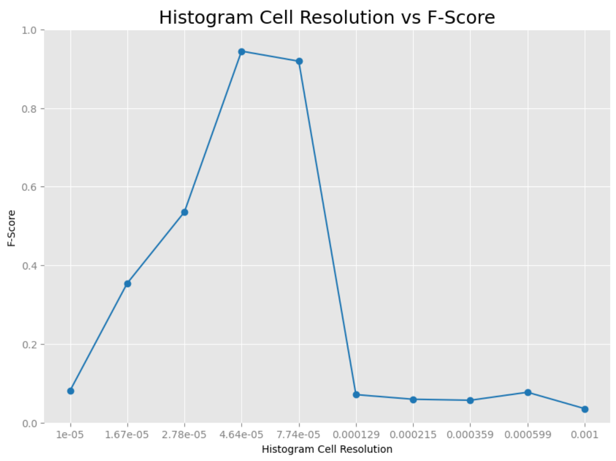
The last ablation study is performed on the rule-based method. The rule-based method takes inspiration from local map-matching techniques, where every timed point is matched to an edge based on the distance and heading. This ablation study aims to answer how important the heading is to getting a good score on map invalidation, as the distance is considered to be the most important to consider to match a point to an edge. The heading weight, C_h, is changed while the distance weight, C_d, is kept constant at 1.0. Figure 4.7 shows how the PR-AUC and F-Scores vary as the heading weight is changed. According to the PR-AUC curve, it seems that heading weight only decreases the performance on map invalidation, while the F-Score curve shows that a tiny amount of heading could be considered, however, this is due to the



(a)



(b)



(c)

Figure 4.6: The results of the ablation studies on the histogram method. Figure 4.6a shows the results of the first ablation study for the histogram method, comparing the different configurations of calculating and extracting the scores from the histogram. Figures 4.6b and 4.6c show the results from the ablation study on the different resolutions for the histogram, and how they impact the PR-AUC and F1-Score

Table 4.2: Histogram Cell Resolutions

Histogram Cell Resolution
1e−5
1.67e−5
2.78e−5
4.64e−5
7.74e−5
1.29e−4
2.15e−4
3.59e−4
5.99e−4
1e−3

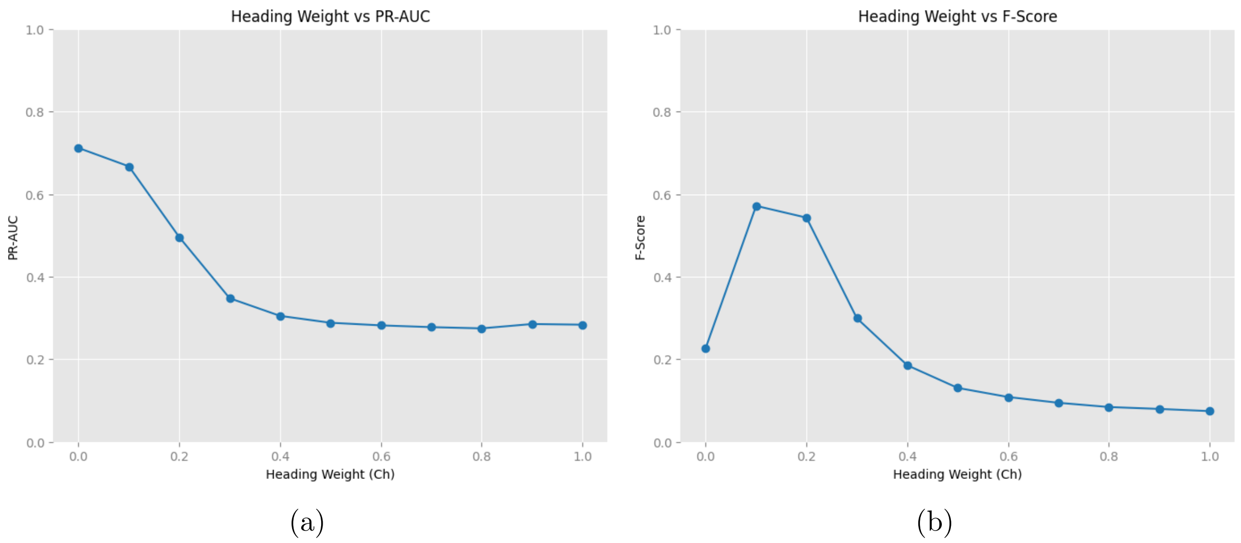


Figure 4.7: The results from the ablation study on the rule-based method.

thresholding function used. Therefore, it can be concluded that the heading mostly decreases the performance of the rule-based method on map invalidation.

Chapter 5

Conclusions

In this paper, we introduced a synthetic GPS probe trace dataset for SD map invalidation called *Synthetic Highways*, with fake data extracted from the game *Cities: Skylines*, for which we introduce two methods based on the already well-established area of *map-matching*, called the rule-based and HMM methods, and one purpose-built method for map invalidation, called the histogram method. Three experiments designed to test the robustness of the methods to noise, trajectory sparsity/density, and trajectory temporal sparsity, produce results that favour the purpose-built histogram method. In all experiments, with the exception of the trajectory temporal sparsity experiment, the histogram method shows high performance on both PR-AUC and F-Score than all other methods. The rule-based method, based on local map-matching techniques, seems to perform reasonably well when trajectory data has a high temporal sparsity, and outperforms the purpose-built histogram method.

Further ablation studies on the histogram and rule-based method highlight the essential parameters for each of the methods. Namely, the cell size in the histogram method has a significant negative impact on its performance on map invalidation if the cell size is too small. With the rule-based method, ablation studies investigating the impact in performance when matching trajectory points based on their distance alone, as opposed to including heading information with increasing weights. The results exemplify that heading may often times hurt performance.

Although SD map invalidation is a research area with industry interest, attaining a reliable real-world dataset is nearly impossible, due to the lack of available information on the current real-time statues of roads. Our dataset provides a benchmark for possible map invalidation methods. Furthermore, a synthetic dataset allows for trajectory augmentation from no-noise, high temporal density trajectories, allowing more detailed analyses into how the methods differ in robustness to different kinds of augmentations, which is quintessential to knowing what method to use when knowing what kind of real-world trajectory data map-makers have available. Besides the *Synthetic Highways* dataset, a mod of the same name is available for download, in case others want to generate more synthetic GPS data for map invalidation.

Our analysis shows that using map-matching based techniques only performs best in few scenarios, highlighting the need for a purpose-built map invalidator, as the histogram method in this paper illustrates. Further research into improving the histogram method could include several post-processing steps after producing the histogram, before extracting scores for each road. The use of filters, such as Gaussian or Canny filters, may improve scores even further, especially in high trajectory noise scenarios. More advanced techniques supervised learning techniques could also be employed, such as a convolution neural network that transforms a noisy histogram built from the trajectories to a histogram built from the map of the second snapshot, G', which essentially is the ground-truth map.

Chapter 6

Appendix

Listing 1: Clingo Code for Suggesting Road Removals

```
% Every edge can be a new edge
{{ new_edge(EN,N1,N2) }} 1 :- edge(EN,N1,N2).

% Maximize the number of roads to be removed
numedge_old(OC) :- OC = #count {{ EN,edge(EN,N1,N2) : edge(EN,N1,N2) }}.
numedge_new(NC) :- NC = #count {{ EN,new_edge(EN,N1,N2) : new_edge(EN,N1,
    ↪ N2) }}.
#maximize {{ DIFF : numedge_old(OC), numedge_new(NC), DIFF = OC - NC}}.

% Define reachability over old network
reachable(N1,N1) :- node(N1).
reachable(N1,N2) :- node(N1), node(N2), edge(_,N1,N2).
reachable(N1,N3) :- node(N1), node(N2), node(N3), N1!=N2, N2!=N3,
    ↪ reachable(N1,N2), reachable(N2,N3).

% Define reachability over new network
new_reachable(N1,N1) :- node(N1).
new_reachable(N1,N2) :- node(N1), node(N2), new_edge(_,N1,N2).
new_reachable(N1,N3) :- node(N1), node(N2), node(N3), N1!=N2, N2!=N3,
    ↪ new_reachable(N1,N2), new_reachable(N2,N3).

% Contraint: All nodes that were reachable previously must remain
    ↪ reachable
:- reachable(N1,N2), not new_reachable(N1,N2).

% Define removed edge as being an edge that does not have a new_edge
removed_edge(EN,N1,N2) :- edge(EN,N1,N2), not new_edge(EN,N1,N2).
```

Bibliography

[1] James Biagioni and Jakob Eriksson. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *Transportation Research Record*, 2291(1):61–71, 2012.

[2] Lili Cao and John Krumm. From gps traces to a routable road map. In *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2009), November 4-6, 2009, Seattle, WA*, pages 3–12, November 2009.

[3] J.J. Davies, A.R. Beresford, and A. Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4):47–54, 2006.

[4] Alexey Dosovitskiy, Germán Ros, Felipe Codevilla, Antonio M. López, and Vladlen Koltun. CARLA: an open urban driving simulator. *CoRR*, abs/1711.03938, 2017.

[5] Stefan Edelkamp and Stefan Schrödl. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*, 2003.

[6] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. *CoRR*, abs/1605.06457, 2016.

[7] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. 1 potassco: The potsdam answer set solving collection. *AI Commun.*, 24:107–124, 01 2011.

[8] Josh Greenfeld. Matching gps observations to locations on a digital map. 01 2002.

[9] Lei He, Shengjie Jiang, Xiaoqing Liang, Ning Wang, and Shiyu Song. Diff-net: Image feature difference based high-definition map change detection. *CoRR*, abs/2107.07030, 2021.

[10] Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. Roadrunner: Improving the precision of road network inference from gps trajectories. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '18, page 3–12, New York, NY, USA, 2018. Association for Computing Machinery.

[11] Minhyeok Heo, Jiwon Kim, and Sujung Kim. Hd map change detection with cross-domain deep metric learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10218–10224, 2020.

[12] Xuemei Liu, James Biagioni, Jakob Eriksson, Yin Wang, George Forman, and Yanmin Zhu. Mining large-scale, sparse gps traces for map inference: Comparison of approaches. 08 2012.

[13] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. Map-matching for low-sampling-rate gps trajectories. pages 352–361, 01 2009.

[14] Paul Newson and John Krumm. Hidden markov map matching through noise and sparseness. In *17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2009), November 4-6, Seattle, WA*, pages 336–343, November 2009.

[15] David Pannen, Martin Liebner, and Wolfram Burgard. Hd map change detection with a boosted particle filter. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2561–2567, 2019.

[16] Jan-Hendrik Pauls, Tobias Strauss, Carsten Hasberg, Martin Lauer, and Christoph Stiller. Can we trust our maps? an evaluation of road changes and a dataset for map validation. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 2639–2644, 2018.

[17] Jan Pinos, Vit Vozenilek, and Ondrej Pavlis. Automatic geodata processing methods for real-world city visualizations in cities: Skylines. *ISPRS International Journal of Geo-Information*, 9(1):17, Jan 2020.

[18] Andy W. R. Soundy, Bradley J. Panckhurst, Phillip Brown, Andrew Martin, Timothy C. A. Molteno, and Daniel Schumayer. Comparison of enhanced noise model performance based on analysis of civilian gps data. *Sensors*, 20(21), 2020.

[19] Rade Stanojevic, Sofiane Abbar, Saravanan Thirumuruganathan, Sanjay Chawla, Fethi Filali, and Ahid Aleimat. Kharita: Robust map inference using graph spanners. *CoRR*, abs/1702.06025, 2017.

[20] Meert Wannes and Mathias Verbeke. Hmm with non-emitting states for map matching. *ECDA*.

[21] Pan Zhang, Mingming Zhang, and Jingnan Liu. Real-time hd map change detection for crowdsourcing update based on mid-to-high-end sensors. *Sensors*, 21(7), 2021.