A scalable method to construct compact road networks from GPS trajectories

Yuejun Guo[a], Anton Bardera[a], Marta Fort[a] and Rodrigo I. Silveira[b]

[a] Graphics and Imaging Lab, Universitat de Girona, Campus Montilivi, Girona, Spain
[b] Departament de Matemàtiques, Universitat Politècnica de Catalunya, Barcelona, Spain

ARTICLE HISTORY
Compiled November 12, 2020

ABSTRACT
The automatic generation of road networks from GPS tracks is a challenging problem that has been receiving considerable attention in the last years. Although dozens of methods have been proposed, current techniques suffer from two main shortcomings: the quality of the produced road networks is still far from those produced manually, and the methods are slow, making them not scalable to large inputs. In this paper, we present a fast four-step density-based approach to construct a road network from a set of trajectories. A key aspect of our method is the use of an improved version of the Slide method to adjust trajectories to build a more compact density surface. The network has comparable or better quality than that of state-of-the-art methods and is simpler (includes fewer nodes and edges). Furthermore, we also propose a split-and-merge strategy that allows splitting the data domain into smaller regions that can be processed independently, making the method scalable to large inputs. The performance of our method is evaluated with extensive experiments on urban and hiking data.

KEYWORDS
GPS trajectory; road network construction; split-and-merge strategy; Slide method

1. Introduction

Nowadays, more and more devices (smartphones, smartwatches, bracelets, etc.) are equipped with global navigation satellite systems (e.g., GPS) to track the position. The increasing popularity of GPS tracking devices leads to a massive amount of trajectory data generated every day. Constructing a route network from these data helps to understand the mobility of users and can provide an up-to-date basis for navigation and recommendation systems. Traditionally, producing a road network requires a large number of images from satellite or aerial photos from providers and platforms, or extensive field surveys, and intensive post-processing. This method is labor-expensive, time-consuming, and faces technical challenges (Chen and Cheng 2008, Li *et al.* 2016, Huang *et al.* 2018). Applying algorithms to generate road networks automatically is an attractive alternative that has received a lot of attention in the last decade. The automatic generation of road networks from GPS tracking data is also useful as a way

CONTACT Anton Bardera. Email: anton.bardera@imae.udg.edu
CONTACT Marta Fort. Email: mfort@imae.udg.edu
CONTACT Rodrigo Silveira. Email: rodrigo.silveira@upc.edu

to supplement existing maps with up-to-date information about road closures and other recent changes in the road network.

The literature on methods to generate road networks from GPS tracks, a problem known as *road network construction*, is extensive (see Section 1.1). However, most of them are still very time-consuming and are unable to scale to large inputs. For instance, existing methods handle all the input GPS data and analyze the whole covered area at once, which becomes infeasible when massive amounts of data need to be processed. Moreover, the quality of produced road networks is still far from that produced manually. For example, there are two recurring issues with automatically-generated road networks. The first is ignoring roads that are followed only by a few trajectories (considered unimportant) (Cao and Krumm 2009, Wang *et al.* 2015), which results in road networks that only partially cover the original data. The second is being too sensitive to incorrect input data, which results in road networks with nonexistent or erroneous roads.

Indeed, these problems are inherent to any sampling-based process, and there is no easy solution. Ignoring infrequent paths makes sense when dealing with redundant urban trajectories (i.e., data obtained from buses, cars, or trucks), considering that such paths would probably be non-passable or forbidden. However, infrequent paths may also be valuable when they represent valid movements that were not possible before. Additionally, in outdoor activity trajectory data, such as hiking or cycling, infrequent paths may indicate recently discovered or rarely-used routes, which may be of particular interest. GPS tracks of people walking, hiking, running, mountaineering, or trail running, are collected in several platforms such as Hiking Project (Project 2020), Komoot (Komoot 2020), AllTrail (AllTrail 2020), and Wikiloc (Wikiloc 2020). Some of them are collecting data from all over the world. It leads to another important aspect to take into account: the size of the geographical area makes it hard to obtain a road network covering the entire domain at once. Hence, designing a split-and-merge strategy to construct partial road networks for different areas, and lately merge them into one global road network, would be of great importance.

This paper presents a method to construct an accurate global road network from GPS data, from either vehicles or outdoor activities, that cover a large geographical area. To that end, we present a fast and simple density-based four-step approach to construct a road network. The three key points of our method are (i) Compacting the density distribution that defines a binary mask from which to obtain a preliminary road network. This is achieved based on a slightly improved version of the Slide method (Mach 2014b,a), which bundles trajectories following similar paths together. (ii) Determining the importance of each edge on the preliminary road network based on the input trajectory data, to detect and remove artifacts and to obtain a more accurate result. The importance of the edges can also give insights into the relevance of each route. (iii) Making the approach scalable, thanks to a split-and-merge strategy. We split the original data into different (slightly overlapping) geographical regions, then obtain an accurate road network for each region independently, and finally merge the road networks into a single consistent global road network. Moreover, the whole algorithm to obtain a road network is fast in terms of runtime, and several steps are parallelizable to make it even faster. In terms of road network quality, our method produces results that are comparable to or better than the state-of-the-art methods at the global scale, and it outperforms most existing methods when the road networks are analyzed locally.

The remainder of this paper is organized as follows. Section 1.1 provides an overview of the existing algorithms. Section 1.2 details the contributions of this paper. Section 2

introduces the Slide tool that is utilized in our algorithm. Section 3 presents the four-step approach in detail. Section 4 describes the split-and-merge strategy. Section 5 describes the experiments and comparative analysis of different GPS datasets. Finally, Section 6 gives the conclusions and future work.

1.1. *Previous work*

In the past two decades, the increasing availability of massive amounts of trajectory data has sparked the proposal of dozens of road network construction algorithms. Note that, the road network construction problem is also often referred as *map construction* and *map inference* (Ahmed *et al.* 2015a,b, Liu *et al.* 2012).

The usual setting is one where the input is a set of trajectories of moving objects, and the output is some form of a road network, often modeled as a graph embedded in the Euclidean plane. Throughout the paper, we will refer to this graph as the *route graph*.

Based on their algorithmic strategy, road network construction methods are classified into four main categories (even though some methods combine more than one strategy), point clustering, intersection linking, incremental track insertion, and density-based. As our approach is density-based, we present a general four-step pipeline for density-based methods (Ahmed *et al.* 2015b). We refer to Ahmed *et al.* (2015a) for more details on the other three categories. Note that in Mariescu-Istodor and Fränti (2018), some density-based methods are referred to as *visual methods*.

First, a density surface is produced from the input trajectories. Typically, this is done by overlaying a grid over the set of trajectories, computing a value for each grid cell (e.g., the number or total length of trajectories going through it), and applying some blurring functions to smooth out the surface (e.g., a Gaussian kernel) (Biagioni and Eriksson 2012, Davies *et al.* 2006).

Second, a skeleton is computed from the density function. To that end, most methods apply some kind of skeletonization or thinning algorithm. Often, the density function is first transformed into a binary image, from where later polygons and road centerlines are extracted, for example, see an example in (Davies *et al.* 2006). Alternatively, grayscale images can be used (Biagioni and Eriksson 2012).

The resulting skeleton is usually close to a road network but is still a raster representation, and it contains incomplete or excessive information. The third step consists of extracting a cleaner network representation out of the skeleton. This involves first distinguishing between the edge and vertex pixels. Then, actual vertex coordinates must be computed (e.g., by averaging out those of nearby vertex pixels), and, finally, edges connecting vertices must be produced, often after applying some kind of line simplification algorithm.

While the first steps outlined above result in a base network, its quality is usually low. Thus, most methods include a fourth phase, the refining phase. This phase improves the base network in terms of topology (e.g., removing spurious edges, adding missing connections), geometry (e.g., improving vertex locations), and even adding extra information to the network such as direction or even lane information (Biagioni and Eriksson 2012).

Note that although density-based methods are based on points, and, thus, do not treat the input trajectories as continuous objects, several algorithms do take into account the continuity of trajectories in the refinement phase. Next, we review the most relevant existing density-based road network construction methods.

One of the first road network construction algorithms was that of Davies *et al.* (2006). In the first step, it builds a histogram based on the total length of trajectory edges that go through each cell, which is blurred using a Gaussian filter. A binary road network is computed from the blurred histogram, by using a global threshold value. Then polygon boundaries for the road areas are extracted from the binary image, using a contour follower. In the third step, road centerlines are extracted, resulting in a base network. The fourth step removes spurious short segments.

One of the simplest approaches within this strategy is that of Chen and Cheng (2008). There, the density function is a binary image that captures all the cells where at least one trajectory goes through. Then morphological operations (such as dilation and closing) are used for smoothing, followed by a thinning algorithm that results in the skeleton. The base road network is constructed from this skeleton by connecting neighboring pixels. Redundant nodes are deleted by removing consecutive nodes that are too close and with a similar heading.

The algorithm by Shi *et al.* (2009) also processes the grid with morphological operators to fill gaps and smooth the initial density surface. A thinning algorithm is used to compute a skeleton. The road network construction is done through an algorithm called *combustion*, which through several iterations classifies pixels as road or intersection and groups them into connected components. These components form the basis of the vertices and edges of the base network.

The algorithm by Steiner and Leonhardt (2011) aims at low-frequency data (i.e., trajectories with large gaps between vertices). It uses a similar approach, but it uses a watershed transform on the blurred density function to detect the ridges of the surface, which will become edges of the base network. In a refinement phase, some of the incorrectly identified edges are removed, and intersection positions are adjusted.

The algorithm by Biagioni and Eriksson (2012) starts similarly, building a density surface based on how many trajectories pass through each cell. A thinning algorithm produces a gray-scale skeleton. This skeleton is processed with the combustion algorithm of Shi *et al.* (2009). Then each initially identified edge is simplified using the Douglas-Peucker algorithm (Douglas and Peucker 1973). The final refinement phase is rather involved, combining several steps of topological and geometric improvements. One of the steps applies map matching to each trajectory to identify the parts of the road network that represent it. With this information, edges with only one trajectory mapped through it are discarded. A second map matching phase is used to further remove spurious edges. For the geometric adjustments, a method based on k-means is used to tune the vertex positions.

A hybrid method that starts with a density-based strategy is the one by Kuntzsch *et al.* (2016). It starts following Davies *et al.* (2006). In a second step, a map matching phase uses the original trajectories to identify missing parts and junctions on the road network. A third step reconstructs the geometry of the missing parts, especially junctions, using a parameterized junction model designed for road networks.

Finally, also deviating from the classical density-based methods, but still closely related, there is the topological method by Wang *et al.* (2015). This algorithm applies discrete Morse theory and topological simplification to produce a road network. The initial density function is produced in the same way as previous methods, with a Gaussian kernel. The skeleton is produced by using Morse theory to extract ridges. The main advantage of their method is that it is adaptive, in the sense that it is based on local density. However, at the same time, it uses a global persistence-based criterion to filter out unimportant roads.

1.2. *Contributions*

We present a novel road network construction approach that improves over state-of-the-art methods in several aspects:

- It produces road networks of comparable and often better quality than state-of-the-art methods, with a considerably smaller size (fewer nodes and vertices). The key for this is using the Slide method to improve the initial density function, producing one that is more compact and reliable.
- In its refinement phase, it combines information on the length and frequency of the edges to filter those that are likely to be spurious, producing cleaner maps than most previous methods. This also allows it to identify relevant paths with only a few trajectories, which are missed by most existing methods.
- It is fast to generate networks. Besides, several parts of the method can run in parallel.
- The split-and-merge strategy allows it to scale to large datasets, by splitting the input into roughly disjoint geographical regions and building networks in each region independently. We are not aware of previous road network construction methods with this feature.

2. Preliminaries

In this section, we formally define what a trajectory is, and describe in detail the Slide method and mention some ways to improve it in Section 2.1.

In this paper, a trajectory is defined as a sequence of two-dimensional points, $(p_1, p_2, \ldots, p_n)$, that describes the movement of an object. We note that GPS trajectory data also contains a timestamp associated with each point. However, we ignore the time information and only regard the path described by the object, which is defined by the two-dimensional points. Hence, the points of a trajectory define a polyline, that may self-intersect, whose vertices are the trajectory points.

Moreover, we use the term *sub-trajectory* or *part* of a trajectory t, to refer to the trajectory t' defined by a continuous sub-sequence of the points defining t. Hence a sub-trajectory will be defined by two (or more) trajectory points.

2.1. *The Slide method*

Slide is a heuristic method that, given a set of trajectories defined by equidistant points, iteratively refines and adjusts the trajectories to optimize their alignment to the denser areas of the analyzed data. The method was proposed by Paul Mach from Strava labs (Mach 2014b,a) as a tool to adjust OpenStreetMap map geometry to the Strava global heatmap dataset. GPS trajectories are continuously collected to create a heatmap that essentially represents the density distribution of data. The key idea of Slide is to match and merge the input trajectories to the heatmap data, using tools from mathematical optimization.

Slide makes the input trajectory fall (or *slide*) into the surface valleys (high-density zones) by applying a force at each point. Since trajectories are usually not defined by equidistant points, the input trajectories are resampled to have equal distance between adjacent points and fulfill the Slide equidistance requirement. Such a trajectory defined by n points, $T = (p_1, p_2, \ldots, p_n)$, is iteratively adjusted by perturbing its interior points $p_i\,(1 < i < n)$ adding a correction vector, $\mathbf{cr}(p_i)$, to their current position. The

correction vector is defined as a weighted sum of the surface ($\mathbf{s_V}$), distance ($\mathbf{d_V}$), angle ($\mathbf{a_V}$) and momentum ($\mathbf{m_V}$) vector components (defined below) with different weights:

$$\mathbf{cr}\left(p_i\right) = \omega_1 \mathbf{s_V}\left(p_i\right) + \omega_2 \mathbf{d_V}\left(p_i\right) + \omega_3 \mathbf{a_V}\left(p_i\right) + \omega_4 \mathbf{m_V}\left(p_i\right), \tag{1}$$

where ω_1, ω_2, ω_3 and ω_4 are the weights of the corresponding components. According to the public implementation of Slide (Mach 2014a), the weights can be generally set to $\omega_1 = 0.5$, $\omega_2 = 0.2$, $\omega_3 = 0.1$ and $\omega_4 = 0.7$. The four vector components are defined as follows.

- The surface component calculates the movement of p_i with respect to the density surface. Intuitively, to make similar trajectories more compact, the point should move towards the densest part. In Slide, the definition is $\mathbf{s_V}(p_i) = gradientAt\left(p_i\right)$ which means the surface gradient at p_i. This correction vector drags p_i towards its closest valley. Bilinear interpolation (Smith 1981) is applied to approximately estimate the gradient at p_i. Figure 1 illustrates the effect of this component. Here, p_i will be pushed to p_i' where the higher density is.

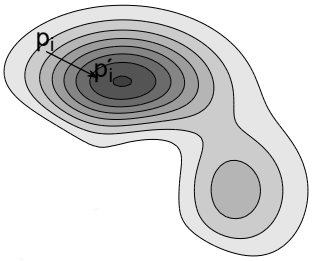


Figure 1. Effect of the surface component on a point p_i. The density surface is shown with contour lines, with darker areas indicating higher density.

- As required by Slide, initially, the trajectory should have equal distance between adjacent points. The distance component tries to ensure that p_i stays close to its previous position by maintaining the equal distance between p_i and its neighbors. To achieve this, the vector is computed by

$$\mathbf{d_V}(p_i) = \begin{cases} \mathbf{0} & p_{i-1} = p_{i+1} \\ \mathbf{m_1} + \mathbf{m_2}, & otherwise \end{cases} \tag{2}$$

  where $\mathbf{m_1} = p_{i-1} - \mathbf{center}$, $\mathbf{m_2} = p_{i+1} - \mathbf{center}$, $\mathbf{center} = \mathrm{p_{i-1}} + \mathbf{u}\left(\frac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{u}\cdot\mathbf{u}}\right)$. Here, the operation $\cdot$ denotes the scalar product, and $\mathbf{u}$ and $\mathbf{v}$ are defined for three consecutive points p_{i-1}, p_i and p_{i+1} as $\mathbf{u} = p_{i+1} - p_{i-1}$ and $\mathbf{v} = p_i - p_{i-1}$.
- The angle component maximizes the vertex angle and minimizes curvature. The calculation is given by

$$\mathbf{a_V}(p_i) = \begin{cases} 0 & |\mathbf{u} - \mathbf{v}| = 0 \ or \ \delta = 0 \\ \delta \frac{\mathbf{u}-\mathbf{v}}{|\mathbf{u}-\mathbf{v}|} \min\left\{|\mathbf{v}|, |\mathbf{u}|\right\} & otherwise \end{cases} \tag{3}$$

  where $\delta = 1 - \sqrt[3]{\mathbf{u}\cdot\mathbf{v}}$ and $|\ |$ is the Euclidean norm.
- The momentum component $\mathbf{m_V}$ is the correction vector used for p_i in the previous iteration ($\mathbf{m_V} = \mathbf{0}$ for the first iteration). Its goal is to speed-up the convergence of the process.

In each iteration, the Slide algorithm moves every interior point of a trajectory according to the corresponding correction vector, while maintaining the endpoints in their original positions. Thus, Slide prompts the interior part of the trajectory to a denser part of the surface. Slide terminates when the improvement between two consecutive iterations is smaller than a preset threshold (usually $5 \cdot 10^{-4}$). The improvement is computed as the difference of the sum of the density values for all the points according to the density surface.

For our purposes, Slide presents two main issues. Firstly, the geometrical meaning of the formula defining the distance component is not consistent with the goal of this component (see Section 3.3). In practice, it causes an excessive displacement of the points. Secondly, Slide does not move the endpoints of the trajectories, which is a problem that creates serious visual artifacts. We need the endpoints to move to a denser part of the surface in the same way the interior points do. The impact of these two important issues on the obtained results and how we fix them in our modified version of Slide is explained in Section 3.3.

3. The four-step approach

In this section, we present our four-step approach to construct the road network from the input data. This section starts by introducing the pre-processing step where some basic errors in the input data are removed, then describes the four steps of our approach. In concrete, the four steps are (i) *Building a density surface:* GPS data are driven into a grid to build a density surface; (ii) *Compacting the density surface via Slide:* trajectories are adjusted according to the density surface and the Slide tool, then a new and compact density surface is computed with the adjusted trajectories; (iii) *Constructing the initial road network:* the route graph is constructed from the density surface using a thinning algorithm followed by a polyline simplification method; (iv) *Refining the initial network with edge weights*: edge weights are computed to filter out wrong edges and to provide extra (visual) information on the popularity of the routes and the edges. Figure 2 illustrates the whole process.

3.1. *Pre-processing GPS data*

GPS data always includes noise. The usual causes of error are the inherent GPS error, which may range from 5 to 22 (m), and the sampling rate ranging from almost 3 to 30 (sec.) depending on the datasets (see (Duran *et al.* 2020) for further details). However, most datasets also contain incorrect values originated, fundamentally, from weak signal reception. This can be caused by the environment, a rapid change of position, signal distortion, due to dense canopy, or other typical sources of GPS errors. To decrease the impact of these incorrect points, we delete the points that define extremely long edges that can be visually identified as erroneous. Isolated trajectory points that define such edges are deleted from the dataset. Since we regard them as errors of signal absence, we split the trajectory into two parts by removing the corresponding erroneous point.

Besides, to apply Slide, we resample the trajectories so that they have consecutive points at equal distance. The resampling is done by using a linear chordal approximation interpolating the polygonal defined by the original trajectory (D'Errico 2020). The new points are placed at distance κ from each other (see Table 1 for further details).

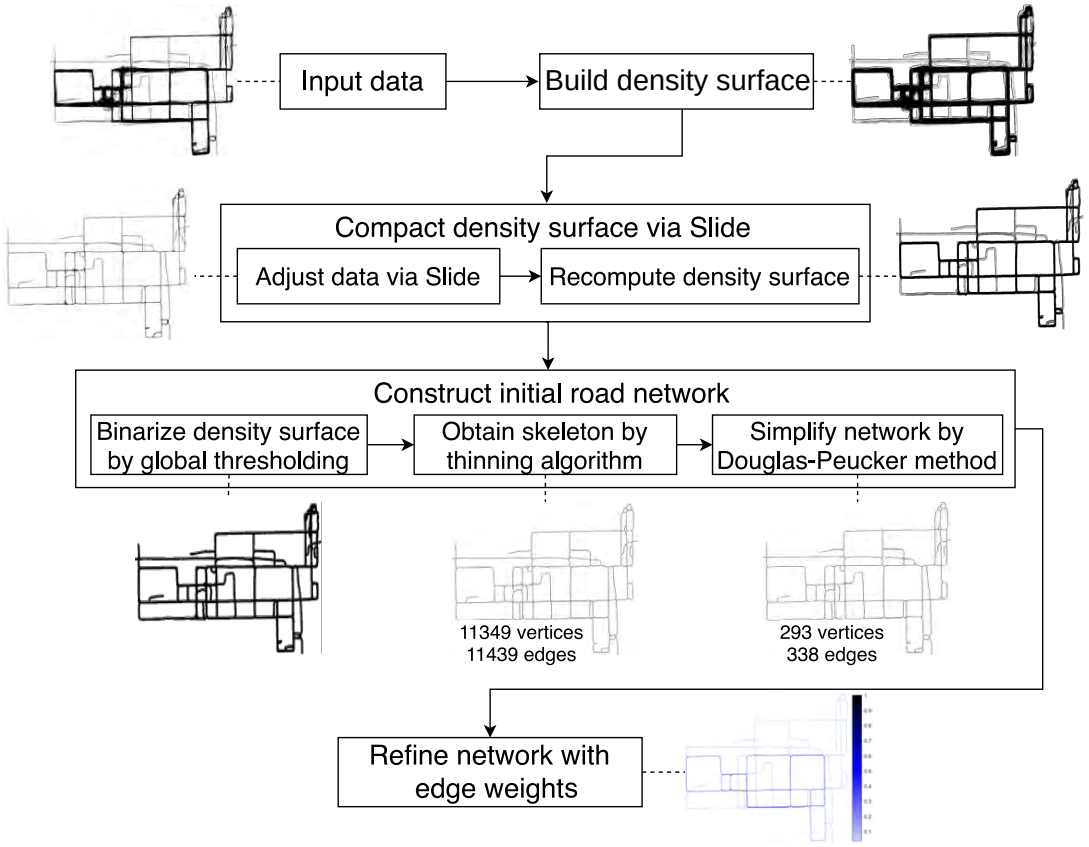


Figure 2. Block diagram representing our four-step road network construction algorithm.

3.2. *Building density surface*

To start with, we cover the area occupied by the given trajectories by a uniform grid with square cells of side length τ. Then we count the number of trajectories going through each cell to build the density surface. Furthermore, to reduce noise and artifacts, we apply a Gaussian blur method (Gonzalez and Woods 2006) on this initial density surface. After the Gaussian blur, a smoother density surface is obtained.

We note that since the mapping of each trajectory is independent of the others, we map each trajectory into the grid in parallel to speed up the process.

3.3. *Compacting density surface via Slide*

To be able to gather similar parts of different trajectories into a single edge of a graph, we apply an improved version of the Slide method described in Section 2 to each trajectory in the dataset. After that, we recalculate a more compact density surface using the adjusted trajectories.

As we mentioned in Section 2.1, in our experiments we found two limitations in Slide. On the one hand, the distance component of Slide aims to keep the distance between consecutive points equal, which helps to prevent points from deviating much from their original positions. Following the notation in Section 2, let p_{i-1}, p_i and p_{i+1} be three consecutive points of a trajectory, and $\mathbf{u} = p_{i+1} - p_{i-1}$, $\mathbf{v} = p_i - p_{i-1}$. Assuming that $p_{i-1} \neq p_{i+1}$ and using the notation in Figure 3 , we can rewrite the

correction vector of Equation 2 as:

$$\begin{aligned} \mathbf{d_V}(p_i) &= \mathbf{u} - 2\mathbf{u}\left(\frac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{u}\cdot\mathbf{u}}\right) \\ &= 2\mathbf{u}(\tfrac{1}{2} - \tfrac{|u_0|}{|u|}). \end{aligned} \tag{4}$$

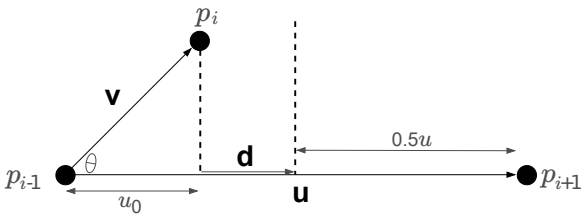


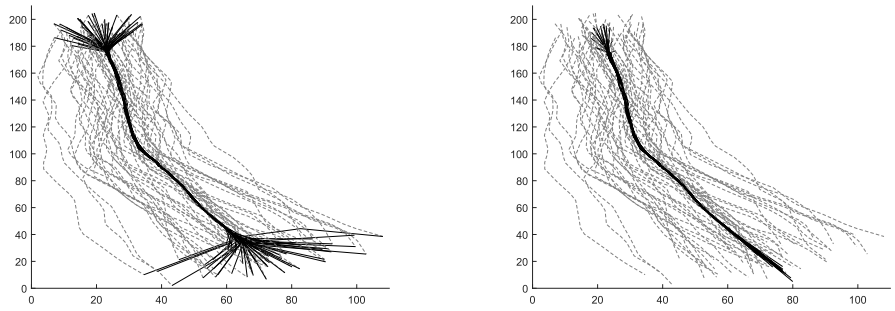
Figure 3. Geometric definition of the distance component of Slide.

where $\mathbf{u} \cdot \mathbf{v} = |u||v| \cos\theta$. From the geometric definition in Figure 3, it is clear that $\mathbf{u}(\frac{1}{2} - \frac{|u_0|}{|u|})$ is equal to vector $\mathbf{d}$. To keep an equal distance to p_{i-1} and p_{i+1}, p_i should move by $\mathbf{d}$. However, according to Equation 4 of Slide, p_i moves twice this amount. To preserve consistency between the intention and the used formula, we propose to define the correction vector of the distance component that p_i moves only by $\mathbf{d}$. Hence, we divide by a factor 2 the original formula. We have experimentally tested this change, and this new version gives better results. The same final effect could also be obtained by using half of the standard weight, i.e., $\omega_2/2$. However, for the mentioned consistency, we prefer to redefine $\mathbf{d_V}(p_i)$ as:

$$\mathbf{d_V}(p_i) = \begin{cases} \mathbf{0}, & p_{i-1} = p_{i+1} \\ \frac{1}{2}\left(\mathbf{m_1} + \mathbf{m_2}\right) & otherwise \end{cases} \tag{5}$$

The second change to Slide involves endpoints. The original version of Slide, presented in Section 2, does not move the endpoints of the trajectories because the correction vector only acts on the interior points. Since the position of a point has a strong influence on the distance and angle components of their neighboring points, the neighbors of the trajectory endpoints will not move properly. In practice, the parts of the trajectories near the endpoints present undesirable sharp changes. This aspect was already mentioned as an issue by the author of Slide (Mach 2014b). Figure 4(a) shows an example using a synthetic dataset provided by Piciarelli *et al.* (2008) showing the undesired shapes of the adjusted trajectories nearby their endpoints. The dataset includes 50 trajectories that are similar to each other, so ideally Slide should produce something that resembles a single trajectory. However, the end parts of the trajectories do not move to the densest area because they are anchored to the endpoints.

To solve this problem, we modify the method to move the endpoints, in each iteration, according to the movement of their neighbors. Let us consider p_1, p_2 and p_3, the first three points of a trajectory. The correction vector proposed by Slide is applied to the interior points of the trajectory, p_2 and p_3. Let p'_2 and p'_3 be the resulting points. Then, the endpoint p_1 is projected onto the line defined by $p'_2p'_3$ (see Figure 5). This orthogonal projection defines p'_1. Similarly, the other endpoint p_n is projected onto the line segment defined by p'_{n-2} and p'_{n-1}. Proceeding in this way the trajectories correctly meet at the denser part of the density surface (see Figure 4(b)). In our ex-



(a) Result by the original Slide. (b) Result by the improved Slide.

Figure 4. The original trajectories (dashed grey lines) and the adjusted ones (solid black lines) when using: (**a**) the original Slide; (**b**) our improved version of Slide.

periments, we also tried to guarantee the points to be equidistant by placing p'_1 on the line defined by p'_2 and p'_3 so that p'_2 would the midpoint of p'_1 and p'_3. However, the results obtained were not as good as expected.

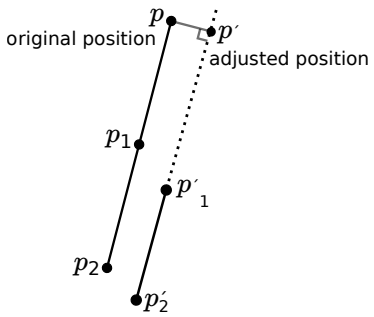


Figure 5. Determining the position of the moved endpoint.

Hence, we apply to each trajectory our improved version of Slide, which uses the new definition for the distance component and adjusts the trajectories endpoints. Finally, we recalculate the density surface with the adjusted trajectories following the procedure explained in Section 3.2. As shown in Figure 2, the density distribution becomes more compact. We use the new density function in the remaining phases.

We remark that since Slide works independently on each trajectory (in both the original and improved versions), our method adjusts all the trajectories in parallel to reduce computation time.

3.4. *Constructing the initial road network*

In this paper, we model the route network as an undirected graph. Each vertex has a geographical coordinate, and each edge represents a line segment in the network that connects its corresponding vertices. The density surface obtained based on Slide is first transformed into a binary image through global thresholding (Gonzalez and Woods 2006). Next, a thinning algorithm is applied to extract a one-pixel-wide skeleton of the binary image. Similarly to (Biagioni and Eriksson 2012) and (Shi *et al.* 2009), we obtain the initial graph from the skeleton, then use the line simplification (Douglas-Peucker) method by Douglas and Peucker (1973) to remove redundant vertices from each edge and to produce the final simplified preliminary route graph, see Figure 2. To speed up the computation, the simplification algorithm also runs in parallel on each

edge.

3.5. *Refining the road network with edge weights*

In the refinement step, we compute the edge weights. The weight of an edge measures its importance in the network, and it can help to filter unimportant edges, and in the visualization of a route graph—to give insight into the more or less frequent edges of the network. We define the weight of an edge as a pair of values based on two factors: length and frequency. The *length* of an edge is the Euclidean distance between its two vertices. The *frequency* of an edge is the number of trajectories passing through it. We compute it by counting the number of trajectories *mapped* to this edge, as follows.

To compute the edge frequency, we match the trajectories adjusted by Slide to the edges. When recomputing the density surface, we map the adjusted trajectories to the grid, in which each cell corresponds to a pixel of the skeleton image. We take advantage of this information to find the closest pixel of the skeleton image for each trajectory point. Such a pixel corresponds to an edge that is initially associated with the trajectory. Around the vertices with degree larger than two, wrong mappings usually happen. Figure 6 shows an example mapping a trajectory from the *Athens small* dataset. In Figure 6(b), the trajectory is initially matched to several edges. Note that this mapping is incorrect, as the top-right edge should be excluded. This occurs because some points of the trajectory near the intersection are close to this edge. To solve this problem, we remove the edges where the trajectory occupies less than half of the pixels defining the edge. This is done by extracting information from the skeleton image. Hence, after removing these underused edges, the whole trajectory is mapped to a set of edges that fit the trajectory more accurately, see Figure 6(c). The distance between pixels is measured by the Euclidean distance. To speed up the computation, we apply the Euclidean distance transform (Maurer *et al.* 2003) to the skeleton image. For each pixel, the distance transform assigns the nearest nonzero pixel.

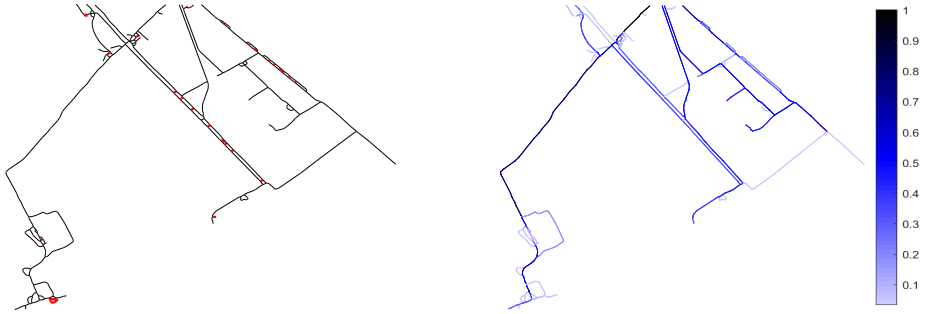
(a) Example trajectory (b) First mapping (c) Final result

Figure 6. Example of how a trajectory (dashed) is mapped to a route graph (solid edges). (**a**) Original data. (**b**) Route graph edges matched to the trajectory using 1-nearest neighbor. (**c**) Final result after removing underused edges.

The advantage of this method is three-fold. First, the adjusted trajectories and the skeleton image have been produced along with the road network construction, which saves time from data preparation. Second, utilizing the adjusted trajectories instead of the original ones makes the mapping more accurate. Third, the similarity between trajectories and the graph is computed by the Euclidean distance between pixels, which is simpler than measuring the distance between lines.

After obtaining the edge weights we refine the route graph by deleting the edges with the frequency and length smaller than prefixed thresholds f and l, respectively. This information is used for two purposes. First, although we have stated that we are interested in infrequent paths, short and infrequent edges are usually artifacts due to acquisition errors. Thus, their removal increases the final quality of the result, while

the long infrequent paths, which are the ones we are interested in, are kept. Second, the edge frequency information is used in the visualizing of the road network. Edges are visualized in a blue gradation according to their frequency. Darker color relates to higher frequency, and vice versa. In Figure 7, we can see the effects of this refining step. Figure 7(a) shows the initial road network that contains black and red edges. Meanwhile, the refined network painted in Figure 7(b) in a blue gradation according to the frequency of the edge, only includes the original black edges. The initial red edges are those deleted in this refining stage, and most of them are artifacts. This refined road network has been obtained considering $f = 2$ and $l = 30$ meters.



(a) Initial road network (b) Final road network

Figure 7. Map of the *Delta* dataset before (**a**) and after (**b**) the refining stage.

3.6. *Discussing parameters*

Our approach includes several parameters. Table 1 gives an overview of the parameters involved in our method, dividing them into two groups: fixed and variable settings. The fixed settings are parameters whose values are the same in all the datasets tested. In contrast, the variable settings have been adjusted for each considered dataset, and we give guidelines on the setting.

Table 1. Overview of parameters

	Parameter	**Symbol**	**Value**
Fixed setting	weights in Slide	ω_1,ω_2,ω_3,ω_4	0.5,0.2,0.1,0.7
	threshold of path score in Slide	u	$5 * 10^{-4}$
	standard deviation of Gaussian blur for initial density distribution	σ_1	5 pixels
	distance for Douglas-Peucker	d	2 meters
	threshold of edge weight (frequency and length)	f l	smallest edge frequency median length
Variable setting	resample distance	κ	set via experiments
	cell size	τ	
	standard deviation of Gaussian blur for re-computed density distribution	σ_2	
	threshold to convert binary image	ϵ	

Briefly discussing the fixed setting parameters, we remark that we take the values of the parameters of Slide from the original implementation. Concerning the distance threshold in Douglas-Peucker, it is the same meter resolution in all the cases because we are interested in obtaining route graphs with similar resolution. The threshold of edge weight includes the edge frequency f and edge length l. We set f as the smallest edge frequency and l as the median length of edges with frequency f. With higher values of f and l, the route graph will be simpler because more edges will be deleted.

Concerning the variable setting parameters, the values used in our experiments are the result of an experimental tuning. Since our algorithm is fast, the parameters are easy to tune experimentally taking into account the following considerations. Parameter κ specifies the distance between two consecutive points. As expected, a bigger κ value results in a faster computation, but faces the problem of over-smoothing, losing a lot of information. We set κ to be similar to the average distance between consecutive points in data. Parameter τ determines the resolution of the road network, and the smaller it is, the more details the network will record. For noisy data like urban data, it has to be bigger, and for the less-noisy data like hiking data smaller. In our tests, τ was limited to be 1, 2, or 3 meters. Concerning σ_2, it specifies how much the density surface should be smoothed. It is set between 2 to 5. To obtain good results, the noisier the data, the bigger σ. Finally, ϵ is the threshold used to convert the density surface to a binary image. Paths with density higher than ϵ are preserved, the others are ignored. High values of ϵ may cause breaks in paths of the road network. In any case, if the input data is similar to one of the datasets used in this paper in terms of size and error, the same settings can be adopted.

4. The split-and-merge strategy

In this section, we describe the proposed split-and-merge strategy in detail. The goal is to build the overall network by building smaller network pieces, which can be processed in parallel[1] and then merging them into a single consistent route graph. This becomes essential to deal with massive amounts of trajectories that occur in a continuous area. We remark that the challenge here is to guarantee that computing networks of smaller regions independently and then merging them produces a consistent overall route graph. Not all road network construction methods are suitable for this. For instance, incremental insertion methods (Ahmed and Wenk 2012, Cao and Krumm 2009, Niehofer *et al.* 2009, Quddus *et al.* 2007) produce route graphs that are very sensitive to the order in which trajectories are added. Thus it is hard to guarantee consistency between independently produced route graphs.

In this section, we will show that our method, with some adaptations, can be successfully implemented in a split-and-merge manner. First, we divide the geographical area into different regions with small overlapping areas. Then, we apply the four-step approach to each region to generate the respective graphs. Finally, we merge the generated route graphs to produce a route graph for the entire area. The edges and vertices in the overlapping areas are fixed to keep the continuity and consistency of the route graph.

4.1. *Splitting geographical area*

The first step is to evenly split the geographical area into small regions. To keep the consistency of the graph at the splitting boundaries, we enlarge the adjacent regions by an overlapping zone. Figure 8 shows a simple example where the geographical area is split into two regions (region 1 and region 2) by a vertical splitting line, the midline. To have an overlapping zone, the splitting boundary of region 1 is placed at a

[1]Matlab, the programming language used in our implementation, easily parallelizes the directly parallelizable loops with the *parfor* command. However, it cannot easily process the small regions in parallel, hence, the regions are processed sequentially one after the other.

certain distance to the right of the splitting line, and the splitting boundary of region 2 is located at the same distance to the left of the splitting line. The overlapping zone is the area between two splitting boundaries. This zone has to be big enough so that we have enough duplicated information to lately merge, appropriately, the obtained graphs along the splitting line. Indeed, the width of the overlapping zone is independent of the size of the two regions, so it only depends on the cell size of the grid, τ, and the number of overlapping grid cells, i.e., pixels. In our experiments, we obtained the best results when this zone had a width of 30 grid cells. Accordingly, the splitting boundaries are obtained by translating the splitting line 15τ meters in the respective direction. In the example, the original region is split by a single splitting line. If more than one line is used, we should proceed similarly for each of them.

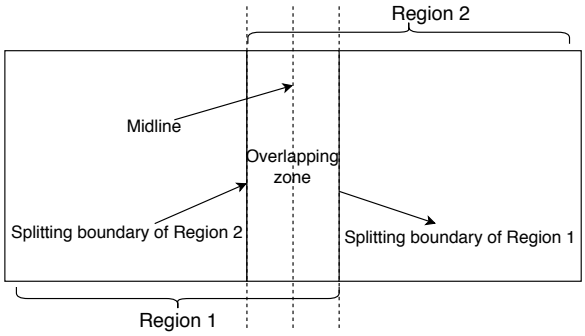


Figure 8. Splitting an geographical area into two regions.

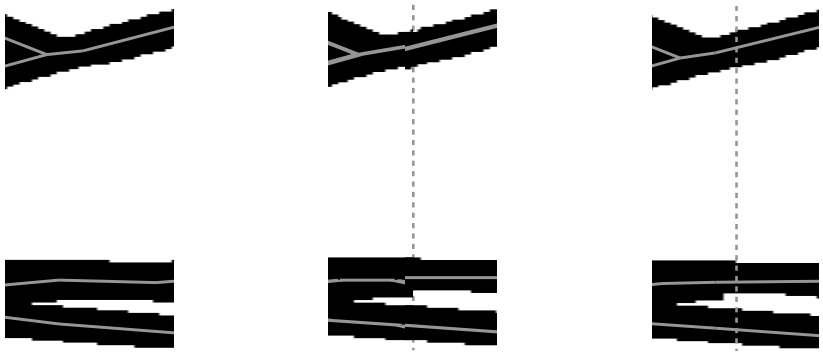
Then we split the trajectories according to the splitting boundaries. The splitting boundary cuts the passing trajectories into sub-trajectories. The intersection points of the trajectory and the splitting boundary, together with the trajectory endpoints, become the sub-trajectories endpoints. These sub-trajectories maintain the shape of the original trajectory, and each of them is entirely contained in a region.

Next, for each region, we apply the four-step algorithm defined in Section 3 to obtain a road network. In each region, the defined sub-trajectories are considered as independent trajectories. After obtaining the road network of each region, producing the whole final network requires merging the individual networks. We explain the merging of networks in the next section.

4.2. *Fixing boundaries and merging road networks*

In the merging step, we cut off the obtained route graphs along the splitting line (midline in Figure 8, blue dashed line in Figure 9) and add new vertices on this splitting line to join the edges that intersect the splitting line. Note that in the way we split the original data, all the sub-trajectories contained in the overlapping zone have been processed twice. However, these sub-trajectories are not adjusted in the same way when dealing with each region. This is because Slide works differently on interior points and endpoints. As a result, trajectories are not adjusted equally, and the re-computed density distribution changes slightly. This directly influences the skeleton image. Thus, simply cutting the graphs along the splitting line, adding vertices at the end of the edges, and combining the graphs according to the new vertices is insufficient. This does not ensure the connectivity and consistency of the global route graph in the splitting line. Next, we explain how to fix the route graph along the splitting lines, a process we call *boundary fixing*.

To fix the boundaries, we use the route graphs in the overlapping zones. Using Figure 8 as an example, we take the route graph of region 1 in the left 15τ-meter-width area and the route graph of region 2 in the right 15τ-meter-width area of the overlapping zone, then combine them as the initial route graph of the overlapping zone. Figure 9 gives an example of fixing boundaries using the *Athens small* dataset. The route graph in Figure 9(a) is generated by our algorithm without the split-and-merge strategy. Figure 9(b) shows the initial route graph of the overlapping zone. It can be seen that the edges from different regions in Figure 9(b) are not correctly connected, although they are close. Thus, we compute the intersections between the splitting line and the route graphs, then merge close-enough intersections to define new vertices. If the intersections are at a Euclidean distance smaller than a threshold, λ_{dis}, we create a new vertex located at their average position to represent these points. Finally, the edges are updated accordingly by replacing the intersection points with the corresponding new vertices. The fixed route graph in Figure 9(c) is almost the same as that in Figure 9(a).



(a) No splitting (b) Before fixing (c) After fixing

Figure 9. The graph (solid gray lines) and the splitting line (dashed gray line) in the binary image of the overlapping zone.

Since the change of the density surface in the overlapping zone exists but is very small (see Figure 9), we set λ_{dis} to be related to the standard deviation of Gaussian Blur σ. By extensive experiments, we found that $\lambda_{dis} = 3\tau\sigma$ achieves good results.

5. Results and discussion

This section presents extensive experiments on different trajectory datasets. First, we begin by comparing the four-step approach to other road network construction algorithms. Second, we check the runtime improvement of utilizing the split-and-merge strategy. Our method was implemented and ran in Matlab 2018a, on a Debian GNU/Linux machine with AMD Ryzen Threadripper 1950X 16-Core Processor and 32 GB RAM. The *Parallel Computing Toolbox* of Matlab was used to execute for-loop iterations with the *parfor* command. This command allows exploiting parallelism almost without any change to the sequential code, just identifying the corresponding loops.

5.1. *Comparison between our approach and other algorithms*

In this section, we test our approach by using four urban datasets (*Athens small*, *Athens large*, *Chicago* and *Berlin*), four hiking datasets (*Delta*, *Aiguamolls*, *Garraf* and *Montseny*) and one jogging dataset (*Joensuu*). The *Joensuu* dataset was made publicly available by Mariescu-Istodor and Fränti (Mariescu-Istodor and Fränti 2019) and used by intersection linking algorithm, CellNet (Mariescu-Istodor and Fränti 2018). The trajectories in the *Joensuu* dataset take place in the city, mostly along streets, with only a few stretches going through parks or other pathways banned to vehicles. For this reason, we consider it more similar to the urban data sets above than to the hiking data sets. Table 2 summarizes the specifications of these datasets. Besides, it provides the runtime of our algorithm for each of them, without applying the split-and-merge strategy. Column *Runtime* indicates the runtime with the parallelizable loops parallelized, and *Seq. Runtime* presents the runtime of the sequential version of our algorithm (i.e., without using the parfor instruction). From this table, we can see that our algorithm generates road networks rather fast. These datasets have been previously used in the literature to evaluate road network construction algorithms, for example, in (Ahmed *et al.* 2015a), (Duran *et al.* 2020) and (Mariescu-Istodor and Fränti 2018). (Ahmed *et al.* 2015a) gives a detailed comparison of several algorithms based on different distance measures using urban datasets. Duran *et al.* (2020) presents an analysis of local artifacts that usually appear in route graphs generated when using both urban and hiking data. We also compare our approach with one additional algorithm that which we call WWL (Wang *et al.* 2015). WWL is a recent density-based method but not evaluated in previous work, but we consider worth including it in our study for several reasons. Firstly, it is a density-based method that uses tools from computational topology, which is very different from previously studied methods. Secondly, the results obtained, as presented by the authors, seem to improve over many previous methods. Thirdly, it has a public implementation (Wang 2016) that we could use to evaluate their algorithm. The parameters of this algorithm have been tuned to obtain the best results.

Table 2. Statistics of GPS trajectory datasets and runtime of our approach to generate road networks.

Dataset	**Area**	**# Trajectories**	**# Points**	**Runtime (seconds)**	**Seq. Runtime (seconds)**
Athens small	$2.6km \times 6km$	129	2839	4.21	26.81
Athens large	$12km \times 14km$	482	32745	113.99	595.28
Chicago	$3.8km \times 2.4km$	889	118360	10.77	119.13
Berlin	$6km \times 6km$	27189	192223	112.33	1253.13
Joensuu	$2.8km \times 2.2km$	109	43891	14.05	19.68
Delta	$2.9km \times 2.8km$	161	38029	6.97	19.67
Aiguamolls	$9.6km \times 5.9km$	101	46116	21.88	38.34
Garraf	$6.7km \times 4.6km$	630	288472	31.47	259.67
Montseny	$7km \times 4.7km$	101	128181	30.08	168.69

This section is organized as follows. First, a general comparison similar to the one by Ahmed *et al.* (2015a) is presented. It comprises road network complexity, length, path-based, and direct Hausdorff distance comparison. Second, a visual inspection, including the data coverage on the *Chicago* and *Joensuu* datasets, and ten common artifacts in hiking data analyzed by Duran *et al.* (2020), is given. Finally, we give a summary of the results. We note that the results of the algorithms AW (Ahmed and Wenk 2012), BE (Biagioni and Eriksson 2012), CK (Cao and Krumm 2009), DBH (Davies *et al.* 2006), ES (Edelkamp and Schrödl 2003), and KP (Karagiorgou and Pfoser 2012) for urban and hiking datasets are taken from (Ahmed *et al.* 2015a)

and (Duran *et al.* 2020), respectively. The results of BE, CK, DBH, ES, and CellNet for *Joensuu* dataset were calculated by using the output networks made available by (Mariescu-Istodor and Fränti 2019).

5.1.1. General comparison

In this general comparison, we evaluate our road networks with some standard measures used in (Ahmed *et al.* 2015a).

5.1.1.1. Road network complexity. The road network complexity refers to the number of vertices and edges in the network and the total length of edges. The length of an edge is the Euclidean distance between the two vertices defining the edge. Tables 3 and 4 present the network complexities for the different algorithms using urban and hiking datasets, respectively. In fact, Table 3 does not contain results for all the algorithms when considering the larger sets, *Athens large* and *Berlin*. According to Ahmed *et al.* (2015a), this is because the missing algorithms could not cope with the size of the input datasets.

We note first that, in terms of the number of vertices and edges, our algorithm generates less complex graphs than most algorithms, while WWL and DBH produce the most complex route graphs for urban and hiking data, respectively. Concerning the length, the results vary a lot with the different algorithms, especially with the hiking datasets. The difference in map complexity comes from how the algorithms proceed. In fact, in the intersection-linking-based (KP) and point-clustering-based (ES) algorithms, the first step identifies intersections or vertices from all the points, and the second step connects them by associating them with the trajectories, which often results in redundant vertices (Duran *et al.* 2020). Incremental methods (AW, CK) insert a trajectory into an existing (initially empty) road network incrementally. If a part of a trajectory is different from the existing network, the algorithms add this part as a new path in the network. Due to the noise in data, trajectories following the same road are not always matched to each other, so the algorithms often create redundant edges. The noise in the data also affects the density-based algorithms (DBH, WWL), resulting in zigzagging shapes that increase the overall network length. In most of the road networks obtained when dealing with the hiking data, the existence of redundant, unnecessary, or incorrect edges increases the total length of the network. This can be easily seen in the networks obtained with the existent algorithms, see Figure 11 for urban, Figure 12 for jogging, or Figure 14 for hiking data.

5.1.1.2. Distance measures. We evaluate the global quality of the resulting route graphs using two distance measures presented in (Ahmed *et al.* 2015a), the *path-based* and the *directed Hausdorff* distances. These measures compute the distance between a ground truth road network and the generated networks. Since the ground truth is only available for urban data, we only consider the urban datasets in this comparison. To compute the distance measures for our networks, we used the public implementation by Pfoser and Wenk (2016). Tables 5 and 6 list the results. Small distance values indicate similarity to the ground truth, and hence, good performance. The computation of these measures is computationally intensive, which caused issues in the largest route graphs: The graphs of *Athens Large, Chicago* and *Berlin* generated by WWL resulted too large, with the program computing the path-based distance aborted after one week of computations. According to (Ahmed *et al.* 2015a), out of the seven algorithms

Table 3. Road network complexity for urban datasets. The results of AW, BE, CK, DBH, ES, and KP are taken from (Ahmed *et al.* 2015a). The result of CellNet was produced based on the output networks made available by Mariescu-Istodor and Fränti (2019).

Methods	Vertex Amount	Edge Amount	Length (km)
Athens small			
AW	344	378	35
BE	391	398	22
CK	20	14	3
DBH	209	227	2
ES	526	1037	197
KP	660	637	35
WWL	7104	8260	44
Ours	**433**	**473**	**39**
Athens large			
AW	7067	7960	1358
KP	6584	5280	252
WWL	56456	80377	430
Ours	**4364**	**4937**	**413**
Chicago			
AW	1195	1286	34
BE	303	322	24
CK	2092	2948	78
DBH	1277	1310	14
ES	828	1247	83
KP	596	558	26
WWL	3234	3549	36
CellNet	306	214	35
Ours	**272**	**307**	**33**
Berlin			
AW	1322	1567	164
KP	2542	2262	161
WWL	20747	25711	277
Ours	**1650**	**1867**	**160**
Joensuu			
BE	105	90	2
CK	4015	2970	24
DBH	4732	4312	46
ES	2303	1502	136
CellNet	1419	1065	65
Ours	**833**	**977**	**57**

Table 4. Road network complexity for hiking datasets. The results of AW, CK, DBH, ES, and KP are taken from (Duran *et al.* 2020).

Methods	Vertex Amount	Edge Amount	Length (km)
Delta			
AW	2362	2459	395
CK	2667	2436	1810
DBH	10229	10197	45
ES	1028	1756	11029
KP	6787	4817	446
WWL	4994	5467	19
Ours	**443**	**466**	**20**
Aiguamolls			
AW	13454	13516	2179
CK	10621	5308	2208
DBH	39786	39206	121
ES	4147	4918	40849
KP	21690	21810	1990
WWL	22228	24258	131
Ours	**1537**	**1608**	**112**
Garraf			
AW	7827	7898	2005
CK	13565	8172	4345
DBH	88009	87162	363
ES	5295	9320	30763
KP	36487	36574	2229
WWL	7922	8720	95
Ours	**1290**	**1338**	**66**
Montseny			
AW	8893	8940	1721
CK	19323	11625	2809
DBH	83025	81783	214
ES	4610	7774	9661
KP	24329	24492	4478
WWL	17519	19143	103
Ours	**1678**	**1754**	**82**

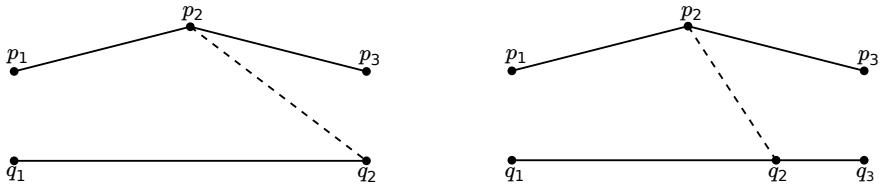
evaluated, the ones performing better, in general, were the ones by KP and by BE. In Table 5, the result of our algorithm does not seem to be as good as these two algorithms but is better than the other methods. On the other hand, according to Table 6, our algorithm behaves better with the directed Hausdorff distance measure.

There are two main reasons for the relatively poor performance of our method according to the measures above. First, the number of vertices of the generated road networks influences the distance measures, since they are based on measuring the distance between vertices. The path-based and Hausdorff distances are based on the discrete Fréchet and discrete Hausdorff distances, respectively. Both are very sensitive to vertex density, as illustrated in Figure 10. The figure shows two simple trajectories, P and Q. Q draws a straight line which is defined by 2 points in Figure 10(a) and 3 points in Figure 10(b). The Fréchet and Hausdorff distances between Q and P are given by the length of the dashed line. Hence, using three points, instead of two, to represent Q benefits these distance measures. Therefore, more compact networks (Figure 10(a)), which are desirable for memory reasons, perform worse in these measures than net-

Table 5. Comparison of path-based distance measure of urban datasets. The results of AW, BE, CK, DBH, ES, and KP are taken from (Ahmed *et al.* 2015a).

Methods	**Path-based Distance(m)**							
	min	**max**	**median**	**average**	**2%**	**5%**	**10%**	**15%**
Athens small								
AW	9	224	45	52	101	101	81	72
BE	5	73	35	36	67	66	61	57
CK	The road network is too small to perform this measure.							
DBH	4	38	11	11	38	18	14	14
ES	2	229	36	39	89	72	68	61
KP	7	229	32	38	113	68	59	57
WWL	8	304	117	127	247	229	222	208
Ours	**5**	**229**	**41**	**45**	**106**	**76**	**75**	**72**
Athens large								
AW	7	849	70	85	250	164	132	114
KP	2	175	25	32	109	80	63	53
WWL	The road network is too large to perform this measure.							
Ours	**3**	**465**	**35**	**40**	**121**	**87**	**71**	**62**
Chicago								
AW	7	201	35	42	127	100	85	76
BE	3	71	15	18	71	38	27	26
CK	1	126	24	27	79	61	49	42
DBH	2	92	12	14	57	24	22	21
ES	1	205	29	37	99	84	72	66
KP	3	89	15	23	72	72	65	51
WWL	The road network is too large to perform this measure.							
Ours	**4**	**109**	**23**	**29**	**91**	**73**	**62**	**42**
Berlin								
AW	9	540	66	74	207	147	120	107
KP	4	306	28	37	120	85	65	52
WWL	The road network is too large to perform this measure.							
Ours	**5**	**395**	**32**	**39**	**104**	**81**	**67**	**60**

works with many redundant points (Figure 10(b)). Note that this is an artifact caused by using discrete measures (vertex-based), as opposed to the original continuous version of Fréchet and Hausdorff distance, which would not have this bias. In practice, if we compute the Hausdorff measure before the Douglas-Peucker simplification, our road network is much better evaluated (see the results in the *Our-no-simplify* row of the tables). In this case, our method becomes the best or second-best for all datasets, concerning both median and average distances. Second, these two measures evaluate the similarity between the generated network and the ground truth without taking into account whether they cover the same area or not. Smaller networks containing fewer paths will easily be better evaluated. Indeed, identifying paths that exist in the input data but do not appear in the ground truth could result in a large distance. Since our algorithm tries to keep all the paths followed by trajectory data, some of these paths are kept. However, the other algorithms regard them as unimportant and eliminate them.



(a) Trajectory Q has 2 points. (b) Trajectory Q has 3 points.

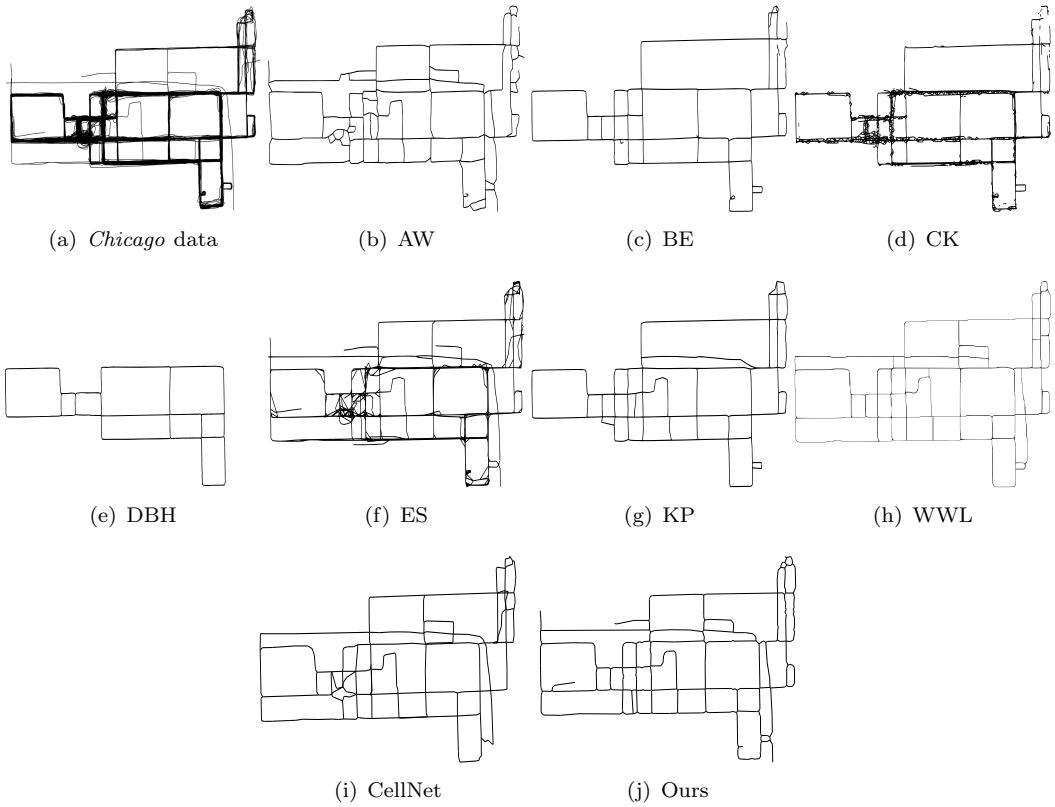
Figure 10. Two different distance values (length of the dashed segment) between trajectories P and Q depending on the number of points of Q.

Table 6. Comparison of directed Hausdorff distance measure of urban datasets. The results of AW, BE, CK, DBH, ES, and KP are taken from (Ahmed *et al.* 2015a).

Methods	**Directed Hausdorff Distance(m)**							
	min	**max**	**median**	**average**	**2%**	**5%**	**10%**	**15%**
Athens small								
AW	1	82	25	26	82	54	46	40
BE	3	74	19	20	47	43	31	31
CK	5	25	13	13	25	25	25	22
DBH	2	13	7	6	13	13	13	11
ES	1	86	18	21	63	50	42	37
KP	2	84	14	17	54	40	33	30
WWL	1	106	9	12	56	38	26	21
Ours	**1**	**79**	**14**	**18**	**54**	**43**	**32**	**29**
Ours-no-simplify	**1**	**79**	**8**	**11**	**45**	**31**	**23**	**19**
Athens large								
AW	1	269	30	33	84	67	56	50
KP	1	200	10	13	46	35	26	22
WWL	1	1143	7	12	47	28	20	17
Ours	**1**	**130**	**14**	**17**	**49**	**40**	**32**	**28**
Ours-no-simplify	**1**	**130**	**8**	**10**	**38**	**28**	**22**	**18**
Chicago								
AW	1	81	14	19	72	59	43	35
BE	2	53	9	11	29	25	23	17
CK	1	78	9	12	44	35	28	25
DBH	2	20	8	7	20	14	13	12
ES	1	93	8	13	57	48	35	25
KP	1	48	7	8	41	23	15	13
WWL	1	111	6	10	63	38	17	13
CellNet	9	1085	106	162	816	453	343	256
Ours	**1**	**60**	**9**	**11**	**40**	**29**	**20**	**17**
Ours-no-simplify	**1**	**60**	**5**	**6**	**23**	**16**	**11**	**10**
Berlin								
AW	1	219	30	33	95	70	60	53
KP	1	232	14	18	59	42	34	30
WWL	1	334	14	25	141	111	61	32
Ours	**1**	**162**	**14**	**18**	**51**	**42**	**34**	**29**
Ours-no-simplify	**1**	**165**	**8**	**11**	**39**	**29**	**22**	**18**
Joensuu								
BE	1	211	20	26	70	59	48	43
CK	1	50	7	8	19	16	13	12
DBH	1	55	10	11	28	22	20	17
ES	1	475	89	91	233	172	132	126
CellNet	8	241	51	61	163	138	114	102
Ours	**3**	**256**	**46**	**58**	**168**	**147**	**113**	**100**
Ours-no-simplify	**3**	**18**	**3**	**4**	**4**	**4**	**4**	**4**

5.1.2. Visual inspection

The previous measures provide global indicators of the produced road networks. In this section, we analyze the generated networks locally by visual inspection. We visually inspect and compare the networks obtained with our approach to those obtained using some of the existent algorithms. We focus on two important factors: (i) data coverage, i.e., we analyze if the generated network covers all the data in the input dataset, and (ii) local road network quality, based on the presence of visual artifacts. The latter follows Duran *et al.* (2020), who performed a local evaluation of road network construction algorithms generated by five previous methods. Their analysis identified several common artifacts present in the networks generated for the four hiking datasets described earlier. We study the networks generated by our method in the same areas studied by Duran *et al.* (2020), to determine to what extent our method suffers from the same limitations.



(a) *Chicago* data (b) AW (c) BE (d) CK

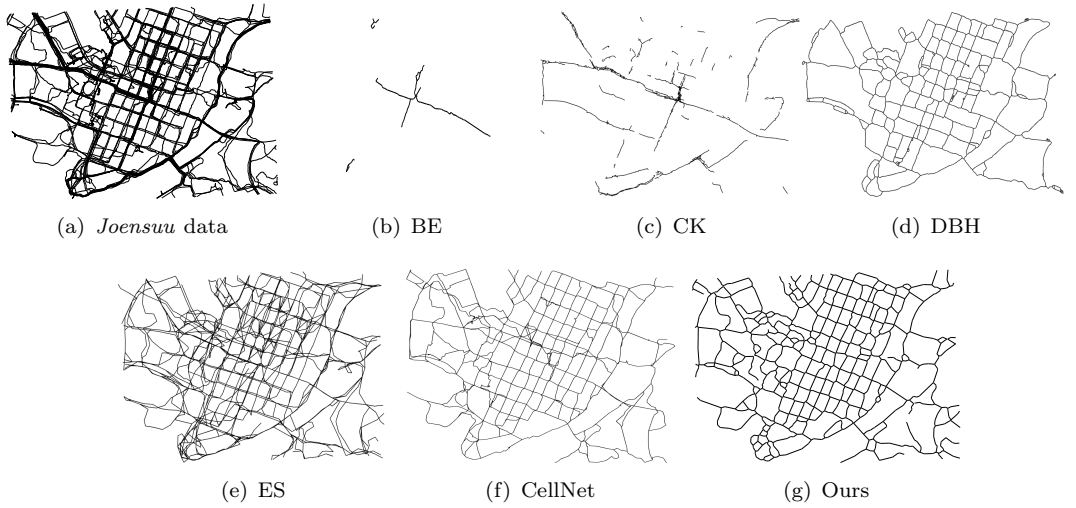
(e) DBH (f) ES (g) KP (h) WWL

(i) CellNet (j) Ours

Figure 11. Road networks of (**a**) *Chicago* dataset by (**b**) - (**i**) several methods. Figures (**b**) - (**g**) are produced by the results in (Pfoser and Wenk 2016).

5.1.2.1. Data coverage. To analyze the coverage of the generated road network on the original data, we use the *Chicago* and *Joensuu* dataset and evaluate the results by eight different algorithms, see Figure 11 and Figure 12, respectively. Figures 11(d) and 11(f) include obviously redundant edges around the same paths, which results in a large number of edges, as listed in Table 3. Figure 11(b) has multiple broken edges on the very right side. Figures 11(c) - 11(e) have low data coverage where only a few edges are produced compared to the other route graphs. Figure 11(g) misses the left bottom and left up paths, which also occurs in some other route graphs. WWL (Figure 11(h)) and CellNet (Figure 11(i)) generate road networks quite similar to ours, but the networks still do not cover all the paths. Our method (Figure 11(j)) has the highest coverage of the paths, and at the same time, low road network complexity. Appendix A gives the route graphs obtained using the other datasets. On the other hand, Figure 12(b) and 12(c) shows how BE and CK miss most of the edges and present several disconnected parts. DBH, shown in Figure 12(d), produces many edges but misses some paths, mainly in the top-left and bottom-right part of the road network, while ES generates many redundant edges. Finally, CellNet and ours are comparable.

5.1.2.2. Local artifacts. The evaluation of five road network construction algorithms by Duran *et al.* (2020)) found that although most algorithms can produce reasonable road networks at a global scale, they fail to represent the route graph accurately at a local scale. This lack of low-level accuracy, not detected by distance measures, has a high impact on the perceived quality of the generated route graph.



(a) *Joensuu* data (b) BE (c) CK (d) DBH

(e) ES (f) CellNet (g) Ours

Figure 12. Resulting road networks for the *Joensuu* dataset (shown in **a**)) by several methods (**b**) - (**i**). Figures (**b**) - (**f**) was produced based on the output networks made available by Mariescu-Istodor and Fränti (2019).

In this section, we evaluate the performance of our method in the same situations that Duran *et al.* (2020) identified as most challenging for existing algorithms, which correspond to ten different *common artifacts*, denoted $[C1]$ to $[C10]$. (Duran *et al.* 2020) evaluated each artifact at a specific location of one of the four hiking datasets, where the input trajectories and the terrain generated a situation that is particularly difficult for most road network construction algorithms.

The presence of each artifact in the resulting route graphs was quantified with an ad-hoc score that measures to what extent the artifact is present. That is, (Duran *et al.* 2020) defines one different score function for each of the ten artifacts. For instance, artifact $[C7]$, illustrated in Figure 14(i) - (l), is concerned with the creation of an excessive amount of map edges in areas with high GPS error. The score function for this artifact computes the ratio of the total length of the produced map to that of the ground truth. We refer to (Duran *et al.* 2020) for the exact definition of the score functions. Table 7 presents, for each artifact, the scores obtained by the algorithms according to the score functions described in (Duran *et al.* 2020). The scores for AW, CK, DBH, ES, and KP are taken from (Duran *et al.* 2020).

The table shows that our method obtains good results, ranking first in five cases, second in one case, and around average for the remaining four. None of the other evaluated algorithms obtains such a good result, being the one by Cao and Krumm (2009) the second-best in this regard. It is interesting to take a look at some of these artifacts, to understand the type of situations evaluated. We present a snapshot of our results in Figure 14. The figure exhibits partial results for four of the artifacts. For each artifact, we show four route graphs corresponding to (i) the worst one among the five evaluated by Duran *et al.* (2020)), (ii) the best one among those evaluated by Duran *et al.* (2020), (iii) the one produced by WWL, and (iv) the one generated by our method. For a detailed description of each artifact and the measures used to quantify them, we refer to (Duran *et al.* 2020).

Next, we comment briefly on each of these four examples.

Artifact $[C8]$, *Excessive number of connections along single path*, shows a simple but fundamental challenge: a single path with a large number of trajectories and high

noise (in this case, due to dense forest). Many methods fail to get a single path in such a situation, producing parallel partials paths connected, see Figure 14(a). While the best method evaluated by Duran *et al.* (2020) produces a single path, and its geometry presents issues towards the right end. WWL captures roughly the correct shape, but with severe wiggling. Our method produces a path with almost perfect geometry, obtaining maximum score according to the measure used by Duran *et al.* (2020).

Artifact [$C3$], *Artificial bridges*, is shown in the second row of the table. It depicts the problematic situation of having two parallel paths at a relatively small distance. Most algorithms tend to merge or connect such paths by creating artificial connections. While both our method and DBH succeed in not adding artificial connections, and thus obtain a maximum score, the paths produced by our method are much closer to the ground truth, as can be verified visually (and based on the ground truth provided by Duran *et al.* (2020)).

The third row illustrates artifact [$C7$], *Excessive number of connections in area.* This area is particularly challenging due to the high noise produced by the canopy and a large number of paths crossing multiple times. Most algorithms generate an excessive number of connections for this input, similar to the result shown in Figure 14(m). In contrast, WWL and our algorithm obtain much cleaner route graphs. The score reflects the total amount of edge length in the route graph concerning the ground truth. Our method misses only a couple of links, covering 88% of the route graph. Even though the graph by WWL visually is worse than ours, its slightly better score comes from the fact that it includes several nonexistent links that increase the total edge length.

The last row of Figure 14 shows artifact [$C2$], *Shortcuts at intersections*, which illustrates one of the most challenging situations for road network construction methods: bifurcations with high noise. When a path forks into two, most methods identify the bifurcation point too late, producing shortcuts in the route graph. In this case, (Duran *et al.* 2020) computes the score for each method as the distance between the first and second bifurcations in the route graph. From that point of view, the best result is the one shown in Figure 14(p), which gets the bifurcation point close to the real one, but produces a lot of noise around it, and even later. In contrast, our method (Figure 14(p)) computes the bifurcation a bit further, thus obtains a worse score, but produces a cleaner and more clear route graph, with the correct topology.

These four artifacts give a good general idea of the behavior of our method for the ten artifacts identified by Duran *et al.* (2020). The results for the remaining six artifacts can be found in Appendix B. Overall, our experiment shows that, at the local level, our method produces better results than all previously analyzed methods.

5.1.2.3. Visual use of frequency information. Finally, we illustrate in Figure 7(b) how assigning a weight to each edge proportional to the frequency information provides results into road networks that show insights into the popularity of the different parts of the network. In the figures, the frequencies of edges are visualized with colors ranging from light blue (lowest frequency) to black (highest frequency). Further examples, corresponding to the datasets of *Chicago* and *Montseny*, are presented in Figure 13.

5.1.3. Summary

To sum up, in terms of road network complexity, our approach generates the simplest road networks for most datasets, while still offering good data coverage. The majority

[C1]		score
(best: 0)		
1	AW	0.04
2	CK	0.15
3	DBH	0.32
4	ES	0.52
4	**Ours**	**0.52**
5	WWL	0.56
6	KP	0.74

[C2]		score
(best: 0)		
1	CK	0.02
2	DBH	0.06
3	ES	0.14
4	**Ours**	**0.38**
5	WWL	0.59
6	AW	0.77
7	KP	0.99

[C3]		score
(best: 0)		
1	DBH	0.00
1	KP	0.00
1	**Ours**	**0.00**
2	WWL	0.11
3	AW	0.21
4	CK	0.56
5	ES	0.69

[C4]		score
(best: closest to 1)		
1	CK	1.01
2	DBH	0.95
3	WWL	1.12
4	**Ours**	**0.87**
5	ES	0.55
6	AW	0.29
7	KP	0.05

[C5]		score
(best: closest to 1)		
1	**Ours**	**0.99**
2	AW	1.02
2	KP	1.02
2	WWL	1.02
3	DBH	1.05
4	CK	2.00
5	ES	5.35

[C6]		score
(best: closest to 1)		
1	**Ours**	**1.05**
2	WWL	1.12
3	DBH	1.17
4	KP	1.46
5	CK	1.66
6	AW	1.98
7	ES	2.19

[C7]		score
(best: closest to 1)		
1	WWL	1.02
2	**Ours**	**0.88**
3	AW	1.13
4	KP	1.14
5	DBH	0.82
6	CK	2.16
7	ES	2.74

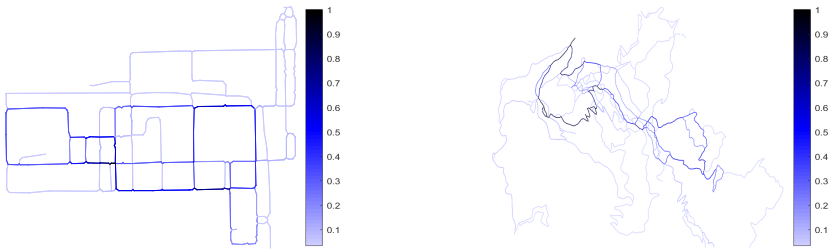
[C8]		score
(best: closest to 1)		
1	**Ours**	**1.00**
2	KP	1.06
3	AW	1.08
4	WWL	1.12
5	ES	3.01
6	DBH	3.12
7	CK	4.01

[C9]		score
(best: closest to 1)		
1	**Ours**	**1.01**
2	KP	0.98
3	AW	0.93
4	WWL	1.14
5	DBH	0.80
6	CK	0.72
7	ES	2.99

[C10]		score
(Yes/No (best: No))		
1	CK	No
1	DBH	No
1	WWL	No
2	KP	Yes
2	AW	Yes
2	ES	Yes
2	**Ours**	**Yes**

Method	Average rank
Ours	**2.1**
WWL	2.9
DBH	3.1
AW	3.5
CK	3.7
KP	3.7
ES	5.0

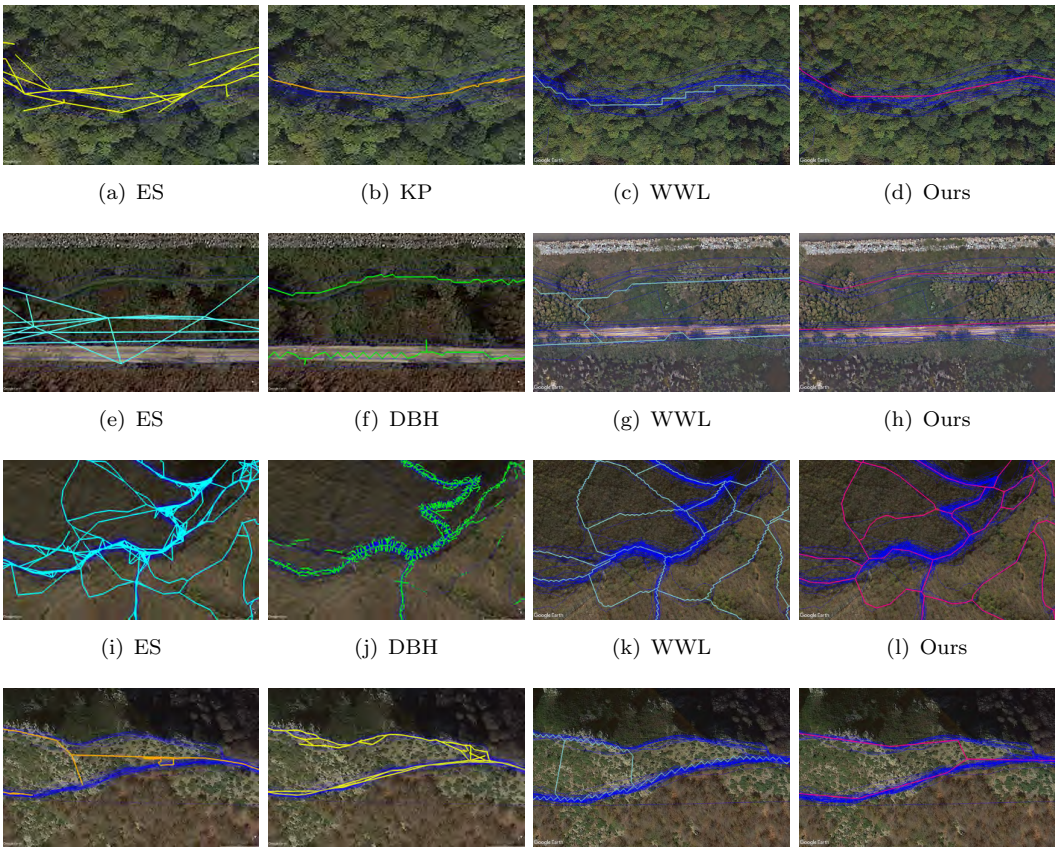
Table 7. Scores of the compared algorithms for the ten common artifacts identified by Duran *et al.* (2020). Algorithms presented sorted from best (top) to worst (bottom). The bottom-right table gives the average rank of each method over the ten common artifacts.



(a) *Chicago* (b) *Montseny*

Figure 13. Road networks of *Chicago* and *Montseny* obtained by our four-step method.

of the other algorithms produce redundant vertices and spurious edges, often caused by wrongly identifying intersections and by the influence of noisy trajectories. Considering the evaluation based on the two distance measures, our results are competitive but somewhat penalized by the fact that our approach sometimes keeps paths that are followed by trajectories but do not exist in the ground truth (as explained in Section 5.1.1.2), although we consider that in most cases this results in meaningful maps. Regarding the data coverage and the presence of local artifacts, our approach performs particularly well, clearly ahead of the other evaluated methods. Also, taking advantage of edge weights to visualize the road network gives extra insights into the route popularity from the given data, and to filter spurious edges in the initial network improves the quality of the route graph. Finally, in terms of performance, our approach can produce road networks for all datasets tested in a matter of a few minutes using a standard laptop computer, making it a practical tool for map construction.



(a) ES (b) KP (c) WWL (d) Ours

(e) ES (f) DBH (g) WWL (h) Ours

(i) ES (j) DBH (k) WWL (l) Ours

(m) KP (n) CK (o) WWL (p) Ours

Figure 14. Artifacts in the generated road networks. Rows from top to bottom: $[C8]$, $[C3]$, $[C7]$, and $[C2]$. Columns left to right: worst and best result in (Duran *et al.* 2020), WWL and our approach. Figures in the first two columns are taken from (Duran *et al.* 2020).

5.2. *Road network evaluation by utilizing the split-and-merge strategy*

To validate our split-and-merge strategy, we compare the road networks obtained with and without it by visual inspection, in terms of road network complexity, distance measures, and runtime. For simplicity, we use "single" and "split-and-merge" to indicate these two ways of handling the input data. Datasets are split into two regions based on the midline, and the two resulting regions of the split-and-merge strategy are processed sequentially and independently one after the other.

According to the results presented in Table 8, the split-and-merge strategy leads to road networks with almost the same complexity and total length than those obtained without splitting. Concerning the runtime, we can see how the split-and-merge strategy also generally saves time. Note that the runtimes of *Athens small*, *Chicago* and *Berlin* datasets slightly increase. There are two reasons for this raise in execution time. First, for simple datasets like *Athens small*, computing the edge weight in each region takes more time than computing it in the whole dataset. Second, fixing the boundaries takes more time if the graph in the overlapping zone is complicated. Since the trajectories in the *Berlin* dataset are densely distributed, especially in the overlapping zone, both the splitting and fixing take a relatively long time. This shows that the choice of the splitting line is important since the difficulty and time needed in the merging process depend on the number of trajectories intersecting the splitting line. The vertical central

splitting line is not necessarily the best option. How to find an optimal splitting line, resulting in few intersection points with the trajectories, is an interesting line to further study.

Next, we study how different the route graphs with and without split-and-merge are. Ideally, we would like the distortion caused by split-and-merge to be minimal. In Table 9 we provide the path-based and directed Hausdorff distance measures for the urban data for both strategies, single and split-and-merge. Note that the *min, max, median* and *average* values almost coincide for all the urban datasets, and in the percentiles, there are only small changes. From this, we can conclude that there are no significant differences between the two resulting route graphs at the global level.

Finally, we present a visual inspection of the route graphs obtained with both strategies in Appendix C. There we can visually confirm that the split-and-merge road networks (colored in blue) do not differ much from those obtained with single (in red).

Overall, the split-and-merge strategy makes our approach scalable to huge datasets without a major impact on the produced route graphs. In addition, it often improves the computational cost of the approach for the datasets studied.

Table 8. Road network complexity and runtime with and without the split-and-merge strategy.

Dataset	**Single**				**Split-and-merge**			
	Vertex Amount	**Edge Amount**	**Length (km)**	**Runtime (seconds)**	**Vertex Amount**	**Edge Amount**	**Length km)**	**Runtime (seconds)**
Athens small	433	473	39	4.21	445	487	38.80	4.46
Athens large	4364	4937	413	113.99	4481	4934	408.32	88.79
Chicago	272	307	33	10.77	288	323	33.35	11.55
Berlin	1650	1867	160	112.33	1763	1947	164.23	147.68
Delta	443	466	20	6.97	441	466	19.55	6.24
Aiguamolls	1537	1608	112	21.88	1550	1616	112.25	14.53
Garraf	1290	1338	66	31.47	1443	1485	71.38	29.40
Montseny	1678	1754	82	30.08	1702	1773	82.29	25.21

Table 9. With and without the split-and-merge strategy comparison: road network distance measures.

Path-based Distance(m)

Datasets	**Methods**	**min**	**max**	**median**	**average**	**2%**	**5%**	**10%**	**15%**
Athens small	single	5	229	41	45	106	76	75	72
	split-and-merge	5	229	41	45	106	75	73	69
Athens large	single	3	465	35	40	121	87	71	62
	split-and-merge	3	465	35	42	127	92	73	66
Chicago	single	4	109	23	29	91	73	62	42
	split-and-merge	4	140	23	28	89	71	58	41
Berlin	single	5	395	32	39	104	81	67	60
	split-and-merge	6	368	32	39	105	89	67	60

Directed Hausdorff Distance(m)

Datasets	**Methods**	**min**	**max**	**median**	**average**	**2%**	**5%**	**10%**	**15%**
Athens small	single	1	79	14	18	54	43	32	29
	split-and-merge	1	79	14	18	54	45	35	29
Athens large	single	1	130	14	17	49	40	32	28
	split-and-merge	1	139	14	17	50	40	32	28
Chicago	single	1	60	9	11	40	29	20	17
	split-and-merge	1	60	9	11	45	32	20	17
Berlin	single	1	162	14	18	51	42	34	29
	split-and-merge	1	162	14	18	51	42	34	28

6. Conclusions

In this paper, we have presented a four-step road network construction approach that is fast, robust, scalable, and partially parallelizable. It combines a density surface smoothed by a Gaussian filter to reduce noise; the Slide tool to adjust trajectories to the higher density zones making the density surface more compact; a thinning algorithm followed by a Douglas-Peucker simplification algorithm to construct the route graph that records the transited zones; and a refinement step assigning and taking into account the weight, combining length and frequency, of the edges. Besides, we have proposed two solutions to solve some defects of Slide, which have been demonstrated to be effective. Overall, the road networks obtained in our experiments provide a simple and good representation of the initial trajectory data. The local analysis shows that in most cases, the generated networks are better than those obtained with the best previously evaluated methods, according to the measures proposed by Duran *et al.* (2020). A remarkable feature of the produced networks is that they present very few redundant edges while including both frequent and infrequent paths. Moreover, the visualization of edge weights in the generated road network gives insight into the importance or popularity of different routes, providing more information than regular networks.

Considering the limitations of storage and computational cost of previous methods, we have designed a split-and-merge strategy that allows handling large-scale data in a distributed manner. The geographical area is split into small regions, with an overlapping zone being kept between adjacent regions to improve the consistency of the combined road network. For each region, the four-step approach is applied to generate a route graph. The individual graphs are merged into a global single graph taking advantage of the overlapping zone to fix the adjacent networks. Experiments on real datasets demonstrate that this strategy succeeds in producing an accurate global road network, very similar to the network that would be obtained if the input data was handled entirely. Indeed, even in the case that the input data can be handled at once, our experiments on real urban and hiking data show that applying the split-and-merge strategy may accelerate the process while keeping the accuracy.

However, there is still room for improvement. First, in the split-and-merge strategy, it would be interesting to study what is the best way to split the geographical area, so that the computation time is reduced and the global road network accuracy is preserved. This could be achieved using the density distribution to decide how to split the area. Moreover, the individual route graphs associated with the small areas could be obtained in parallel. Second, the Slide tool always smooths the sharp turns considering the route is straight, which in some cases causes excessive simplification in turns and wrong merging in narrow curves. A possible solution would be segmenting the trajectories into small sub-trajectories based on the direction change.

Data codes and availability statement

The urban data and codes that support the findings of this study are available with the identifier (https://doi.org/10.6084/m9.figshare.12199541). The hiking data cannot be made publicly available due to the data privacy, but we have made a mocked example on the *Garraf* dataset to give insights into the format of hiking data.

Acknowledgement(s)

We acknowledge Ahmed *et al.* for releasing the four urban datasets: *Athens small, Athens large, Chicago* and *Berlin* available online, Duran *et al.* for providing the four hiking datasets: *Delta, Aiguamolls, Garraf* and *Montseny*, and Mariescu-Istodor and Fränti for the *Joensuu* dataset. Besides, we acknowledge Paul Mach for sharing the implementation of Slide, Ahmed *et al.* for sharing the implementation of different road network construction algorithms and evaluation measures, and Wang *et al.* for making the code of their road network construction method public in GitHub. We also acknowledge the people from Wirkiloc (Jordi *et al.*) for discussing the methodology.

Disclosure statement

The authors declare no conflict of interest.

Funding

This work was supported by the Spanish Government under Grants PID2019-106426RB-C31 and PID2019-104129GB-I00/AEI/10.13039/501100011033; the Catalan Government under grants 2017-SGR-1101 and 2017-SGR-1640; the Universitat de Girona under grant PONTUdG2019/11; and the Chinese Academy of Sciences President's International Fellowship Initiative under grant 2021VTB0004. Yuejun Guo acknowledges the support from Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya and the European Social Fund.

Biographical note

Yuejun Guo got the PhD from the UdG in May 2020. She got the master degree in computer science and technology from Tianjin University in 2016 and the bachelor degree in computer science and technology from Civil Aviation University of China in 2013. Her main research interests focus on trajectory data analysis including clustering, anomaly detection and map construction. She also worked on image processing.

Anton Bardera is an associate professor at the department of Informàtica, Matemàtica Aplicada i Estadística of the Universitat de Girona. He received the PhD from the UdG in November 2008. His research interests are medical imaging and information theory. He is doing the research at the Graphics and Imaging Laboratory (GILab), specifically, in the Medical Imaging Group. He collaborates on the development of the Starviewer application.

Marta Fort is an associate professor at the department of Informàtica, Matemàtica Aplicada i Estadística of the Universitat de Girona. She got the PhD from the UdG in 2008 and bachelor degree in Mathematics from the Universitat Politècnica de Catalunya (UPC) in 2003. The general area of her research is computational geometry, and the application fields are computer graphics, geographic information systems, facility location.

Rodrigo I. Silveira is an associate professor at the Department of Mathematics of Universitat Politècnica de Catalunya (UPC). He works at the UPC research group

on Discrete, Combinatorial and Computational Geometry, and he is the head of the UPC group on Computational Geometry and Applications (CGA). He finished the PhD at Utrecht University, and before the PhD, he did the study at the Computer Science Department of the Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires. He works mainly on Computational Geometry. He has worked on several different areas of Computational and Combinatorial Geometry, and he is particularly interested in problems that are motivated from Geographic Information Science (GIS) and Graph Drawing.

References

Ahmed, M., *et al.*, 2015a. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19 (3), 601–632. Available from: `https://doi.org/10.1007/s10707-014-0222-6`.

Ahmed, M., *et al.*, 2015b. *Map construction algorithms*. Cham: Springer International Publishing.

Ahmed, M. and Wenk, C., 2012. Constructing street networks from gps trajectories. *In*: L. Epstein and P. Ferragina, eds. *Algorithms – ESA 2012*, Berlin, Heidelberg. Springer Berlin Heidelberg, 60–71.

AllTrail, 2020. Alltrail. `https://www.alltrails.com/`. Online; accessed on 28 February 2020.

Biagioni, J. and Eriksson, J., 2012. Map inference in the face of noise and disparity. *In*: *International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, New York, NY, USA. ACM, 79–88. Available from: `http://doi.acm.org/10.1145/2424321.2424333`.

Cao, L. and Krumm, J., 2009. From gps traces to a routable road map. *In*: *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '09, New York, NY, USA. ACM, 3–12. Available from: `http://doi.acm.org/10.1145/1653771.1653776`.

Chen, C. and Cheng, Y., 2008. Roads digital map generation with multi-track gps data. *In*: *International Workshop on Education Technology and Training, International Workshop on Geoscience and Remote Sensing*. 508–511.

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

D'Errico, J., 2020. interparc. (https://www.mathworks.com/matlabcentral/fileexchange/34874-interparc. Online; accessed on 10 March 2020.

Douglas, D.H. and Peucker, T.K., 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*, 10 (2), 112–122.

Duran, D., Sacristán, V., and Silveira, R.I., 2020. Map construction algorithms: a local evaluation through hiking data. *GeoInformatica*, to appear. Available from: `https://www.doi.org/10.1007/s10707-019-00386-7`.

Edelkamp, S. and Schrödl, S., 2003. Route planning and map inference with global positioning traces. *In*: R. Klein, H.W. Six and L. Wegner, eds. *Computer Science in Perspective: Essays Dedicated to Thomas Ottmann*, Berlin, Heidelberg. Springer Berlin Heidelberg, 128–151.

Gonzalez, R.C. and Woods, R.E., 2006. *Digital image processing (3rd edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc.

Huang, J., *et al.*, 2018. Automatic generation of road maps from low quality gps trajectory data via structure learning. *IEEE Access*, 6, 71965–71975.

Karagiorgou, S. and Pfoser, D., 2012. On vehicle tracking data-based road network generation. *In*: *International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, New York, NY, USA. ACM, 89–98.

Komoot, 2020. Komoot. `https://www.komoot.com/`. Online; accessed on 28 February 2020.

Kuntzsch, C., Sester, M., and Brenner, C., 2016. Generative models for road network reconstruction. *International Journal of Geographical Information Science*, 30 (5), 1012–1039. Available from: https://doi.org/10.1080/13658816.2015.1092151.

Li, H., Kulik, L., and Ramamohanarao, K., 2016. Automatic generation and validation of road maps from gps trajectory data sets. *In*: *ACM International on Conference on Information and Knowledge Management*, CIKM '16, New York, NY, USA. ACM, 1523–1532. Available from: http://doi.acm.org/10.1145/2983323.2983797.

Liu, X., *et al.*, 2012. Mining large-scale, sparse gps traces for map inference: Comparison of approaches. *In*: *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, New York, NY, USA. ACM, 669–677. Available from: http://doi.acm.org/10.1145/2339530.2339637.

Mach, P., 2014a. The slide method. https://github.com/paulmach/slide. Online; accessed on 09 December 2019.

Mach, P., 2014b. Strava slide tool. https://labs.strava.com/slide/. Online; accessed on 28 February 2020.

Mariescu-Istodor, R. and Fränti, P., 2018. Cellnet: inferring road networks from gps trajectories. *ACM Trans. Spatial Algorithms Syst.*, 4 (3). Available from: https://doi.org/10.1145/3234692.

Mariescu-Istodor, R. and Fränti, P., 2019. Web of cellnet. http://cs.uef.fi/mopsi/routes/network/. Online; accessed on 13 July 2020.

Maurer, C.R., Qi, R., and Raghavan, V., 2003. A linear time algorithm for computing exact euclidean distance transforms of binary images in arbitrary dimensions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25 (2), 265–270.

Niehofer, B., *et al.*, 2009. Gps community map generation for enhanced routing methods based on trace-collection by mobile phones. *In*: *International Conference on Advances in Satellite and Space Communications*, SPACOMM '09, Washington, DC, USA. IEEE Computer Society, 156–161. Available from: http://dx.doi.org/10.1109/.30.

Pfoser, D. and Wenk, C., 2016. Web of map construction algorithms. http://mapconstruction.org/. Online; accessed on 09 December 2019.

Piciarelli, C., Micheloni, C., and Foresti, G.L., 2008. Synthetic trajectory dataset. https://avires.dimi.uniud.it/papers/trclust/. Online; accessed on 09 December 2019.

Project, H., 2020. Hiking project. https://www.hikingproject.com/. Online; accessed on 28 February 2020.

Quddus, M.A., Ochieng, W.Y., and Noland, R.B., 2007. Current map-matching algorithms for transport applications: state-of-the art and future research directions. *Transportation Research Part C: Emerging Technologies*, 15 (5), 312–328. Available from: http://www.sciencedirect.com/science/article/pii/S0968090X07000265.

Shi, W., Shen, S., and Liu, Y., 2009. Automatic generation of road network map from massive gps, vehicle trajectories. *In*: *International IEEE Conference on Intelligent Transportation Systems*, October. 1–6.

Smith, P., 1981. Bilinear interpolation of digital images. *Ultramicroscopy*, 6 (2), 201–204. Available from: http://www.sciencedirect.com/science/article/pii/0304399181900619.

Steiner, A. and Leonhardt, A., 2011. Map generation algorithm using low frequency vehicle position data. *In*: *TRB 90th Annual Meeting of the Transportation Research Board*, January. 1–17.

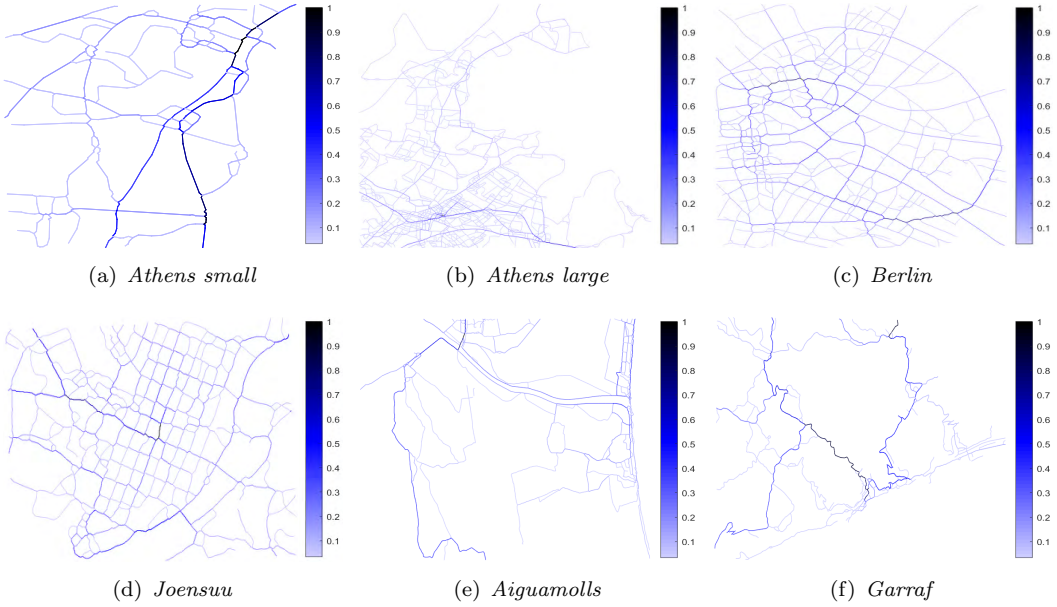
Wang, J., 2016. Graph reconstruction. https://github.com/wangjiayuan007/graph_recon_DM. Online; accessed on 09 December 2019.

Wang, S., Wang, Y., and Li, Y., 2015. Efficient map reconstruction and augmentation via topological methods. *In*: *SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, New York, NY, USA. ACM, 25:1–25:10. Available from: http://doi.acm.org/10.1145/2820783.2820833.

Wikiloc, 2020. Wikiloc. https://www.wikiloc.com/. Online; accessed on 28 February 2020.

Appendix A. Generated road networks by our approach

This section shows the generated route graphs in Figure A1. The edges with higher weight are darker.



(a) *Athens small* (b) *Athens large* (c) *Berlin*

(d) *Joensuu* (e) *Aiguamolls* (f) *Garraf*

Figure A1. Road networks generated with our four-step approach.

Appendix B. Remaining artifacts in local analysis

Figure B1 provides the route graphs obtained with our approach for the rest of the common artifacts analyzed by Duran *et al.* (2020), $[C1]$, $[C4]$, $[C5]$[2], $[C6]$, $[C9]$ and $[C10]$. As before, each row presents, from left to right, the worst and best road network according to (Duran *et al.* 2020), the one by WWL, and our route graph. Recall that the scores obtained by each algorithm for each of the artifacts are given in Table 7.

For artifact $[C1]$, Figure B1(d) shows a closed turn with a pointed shape similar to that of the bifurcation of Figure 14(p). In this situation, our algorithm behaves similarly, producing a neat route graph but with the actual turning point detected a bit too early. The best algorithm analyzed by Duran *et al.* (2020) in this situation is AW. The score here is based solely on how well the turning point is computed, something that AW does accurately here. However, the route graph by AW presents several incorrect edges that make the overall route graph much noisier than ours. Artifact $[C4]$ consists of two close parallel paths that many algorithms tend to merge incorrectly. In this situation, our method generates arguably the best route graph (Figure B1(h), with two almost straight lines without bridges. The route graphs in Figure B1(g) and Figure B1(f) manage to identify the two parallel paths but suffer from some erroneous connections. The route graph in Figure B1(e) shows two paths that almost overlap halfway from the original trajectories. We note that our method obtains a worse score than CK and WWL here because the bottom path is detected

[2]We note that the location of this artifact provided in (Duran *et al.* 2020) is incorrect. The correct location is 41° 18'16.13"N, 2° 7'28.64"E.

a bit too high compared to the ground truth, and the score function used by Duran *et al.* (2020) picks that up. However, we consider that the effect of this displacement is negligible compared to the more notorious issues in the route graphs produced by CK and WWL.

Artifact [$C5$] evaluates if the algorithms generate duplications for one path due to noisy trajectories. All of CK (best in the previous comparison), WWL, and ours manage to avoid duplicating this stretch. However, WWL shows its typical wiggling that deteriorates the result, and KP also has, to a much lesser extent, artificial oscillations. In contrast, our method produces a straight path, which scores best among the seven algorithms compared.

Artifact [$C6$] focuses on places where a large number of trajectories go back and forth. This creates issues for many methods, which fail to identify this and produce duplicates of what should be a single path. The best method in the previous comparison (DBH), WWL, and ours do a rather good job of avoiding this duplication. However, our road network is best in this setting, producing a simple path with the correct topology (Figure B1(p)). Not surprisingly, our route graph scores best, using the measure by Duran *et al.* (2020), over the seven methods compared.

Artifact [$C9$] focuses on the missing parts of the single path due to having very few trajectories going through it. Many algorithms produce disconnected paths in this situation, as CK in Figure B1(q). The best algorithm in the previous comparison, KP, WWL and ours (Figures B1(r) - B1(t)) cover the input data continuously. For this artifact, considering that the length of path produced by our method is most similar to that of the ground truth, our method obtains the best score among the seven methods.

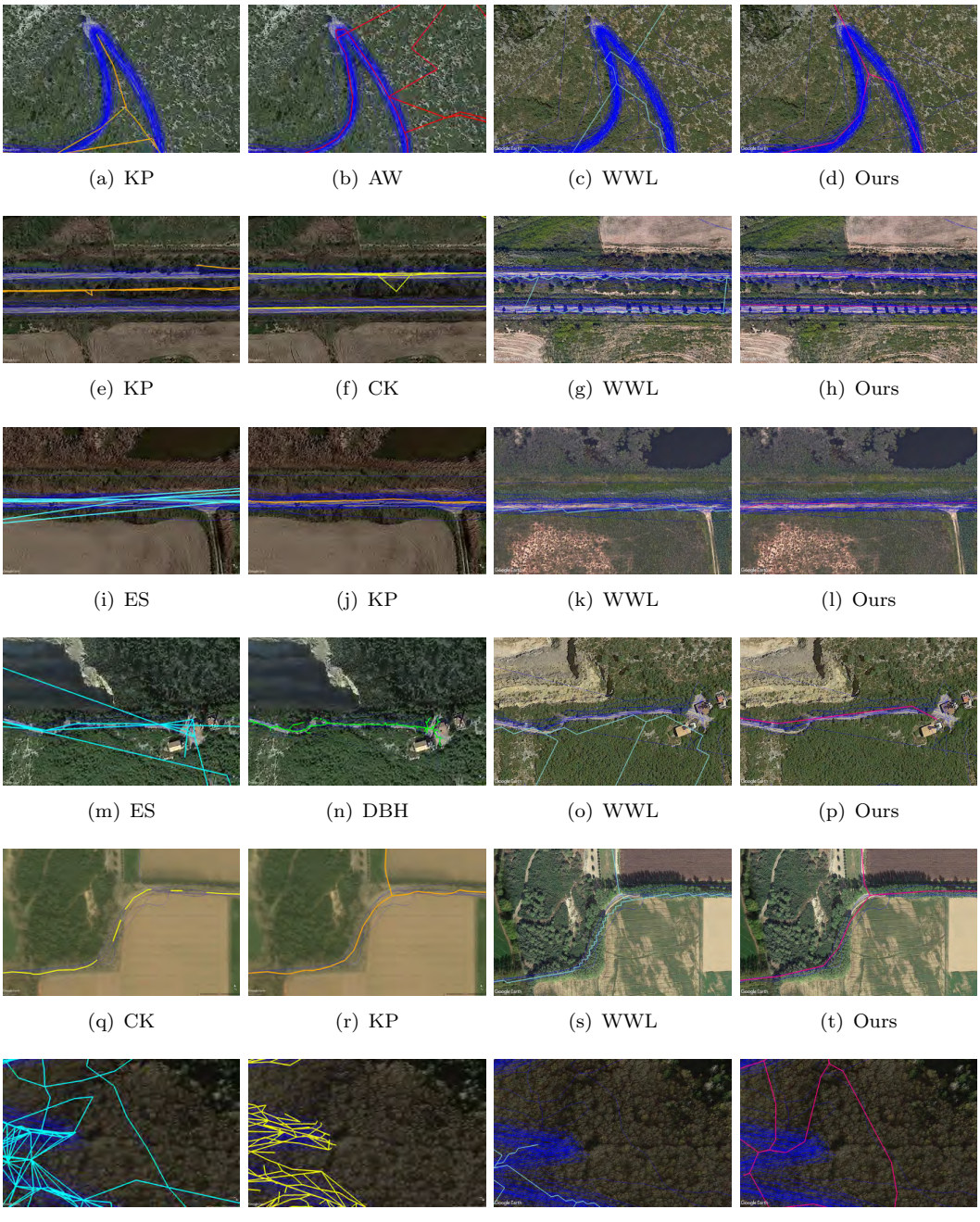
Finally, the last row contains the results for artifact [$C10$]. This situation depicts an area with a high density of trajectories that also contains a separated outlier trajectory that runs top-down. Several methods, including ours, fail to realize that this central trajectory is an outlier. While our method produces a reasonable road network (Figure B1(x)), it does include the outlier trajectory, since it is long and different enough from nearby ones. This issue would have been avoided if we had used a higher frequency threshold f.

B.1. *Specific artifacts*

In addition to the ten common artifacts, we also analyze the performance of our method (and WWL) for the seven specific artifacts identified by Duran *et al.* (2020), see Figure B2. These are artifacts that appear in only one of the five algorithms compared by Duran *et al.* (2020). Thus they are not as common as the previous ones. Nevertheless, it is interesting to see how our method behaves in such situations since most of them are challenging in one way or another.

Artifact [$S1$] (first row in Figure B2) contains a highway underpass that some algorithms miss to present given its relatively short length. Our algorithm not only includes the connection in the route graph but catches the geometry. In contrast, AW (Figure B2(a)) misses the connection, while WWL captures the pass, but with highly zigzagging geometries.

Artifact [$S2$] (second row in Figure B2) considers a wide turn with high number of trajectories, where methods like CK (Figure B2(d)) tend to shortcut and reduce the turn curvature. WWL (Figure B2(e)) also suffers a bit from this issue, while our method (Figure B2(f)) gives the best result, producing a curve that stays all the time within the bundle of trajectories, thus reducing the curvature only marginally.



(a) KP (b) AW (c) WWL (d) Ours

(e) KP (f) CK (g) WWL (h) Ours

(i) ES (j) KP (k) WWL (l) Ours

(m) ES (n) DBH (o) WWL (p) Ours

(q) CK (r) KP (s) WWL (t) Ours

(u) ES (v) CK (w) WWL (x) Ours

Figure B1. Artifacts in generated road networks. Rows from top to bottom: [$C1$], [$C4$], [$C5$], [$C6$], [$C9$], [$C10$]. Columns left to right: worst and best result in (Duran *et al.* 2020), WWL and our approach. Figures in the first two columns are taken from (Duran *et al.* 2020).

Artifact [$S3$] (third row in Figure B2), shows an area where some methods produce spurious short segments or turns. In this case, Figure B2(g) includes a "hair" along the path. WWL and our algorithm avoid this issue. However, the route graph in Figure B2(h) has a somewhat zigzagging geometry.

Artifact [$S4$] (fourth row in Figure B2) focuses on the zigzagging or wiggling of generated paths. This is a typical artifact in density-based methods. Indeed, both DBH and WWL (Figure B2(j) and Figure B2(k)) have this problem. In contrast, our algorithm, despite being also density-based, does not suffer from this issue (see Figure B2(l)).

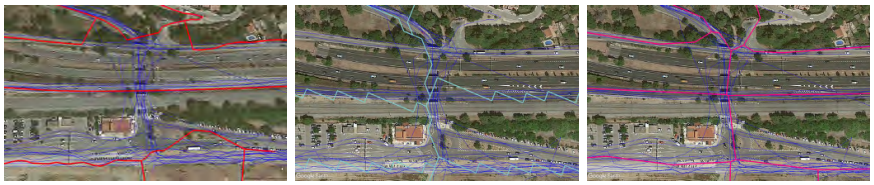
Artifact [$S5$] (fifth row in Figure B2) focuses on paths that end up over-sampled in the resulting road networks. For instance, the route graph by CK includes dozens of vertices for the stretch showed alone (Figure B2(m)). By comparison, Figure B2(n) and Figure B2(o) show better results, and our approach generates the best route graph concerning the similarity with the ground truth.

Artifact [$S6$] (sixth row in Figure B2), shows the issue of excessively simplifying zigzagging paths. Here the three methods suffer from this problem. However, we observe that the route graph produced by our method is still better than that of KP and WWL (Figure B2(p) and Figure B2(q)), in the sense that it does not include any erroneous connection.

Finally, artifact [$S7$] (last row in Figure B2) checks if the resulting road networks are missing the beginning or ends of paths. Both WWL and our method don't have this problem. However, WWL still misses the paths on the right side. Our algorithm produces the best route graph that includes all the paths followed by the trajectories (note that the short connection between both paths is not followed by any trajectory in the data).

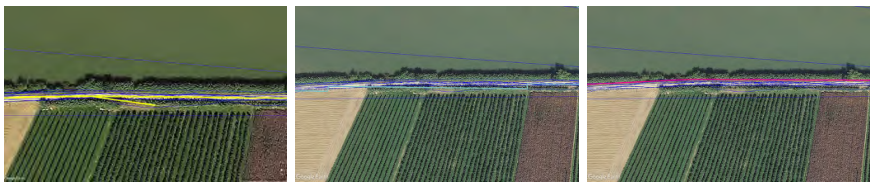
Appendix C. Road networks generated by using the split-and-merge strategy

Next, we provide the route graphs obtained without and with the split-and-merge strategy depicted in red and blue, respectively, in Figure C1.

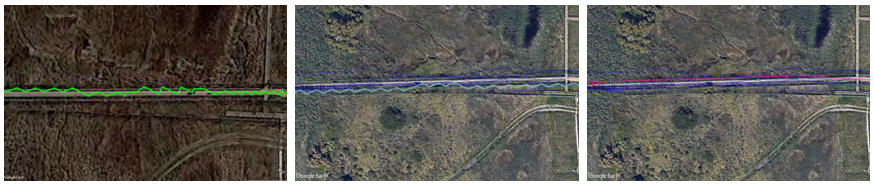


(a) AW (b) WWL (c) Ours

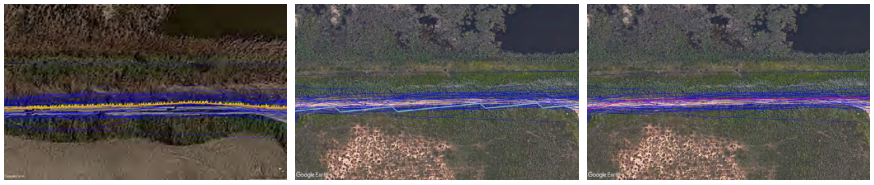
(d) CK (e) WWL (f) Ours



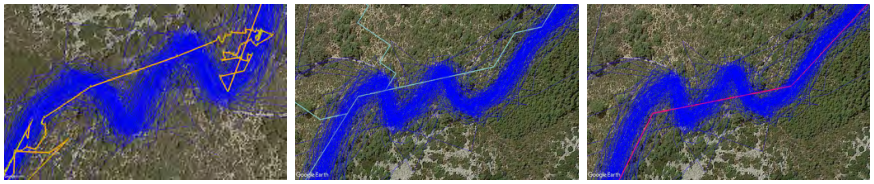
(g) CK (h) WWL (i) Ours



(j) DBH (k) WWL (l) Ours



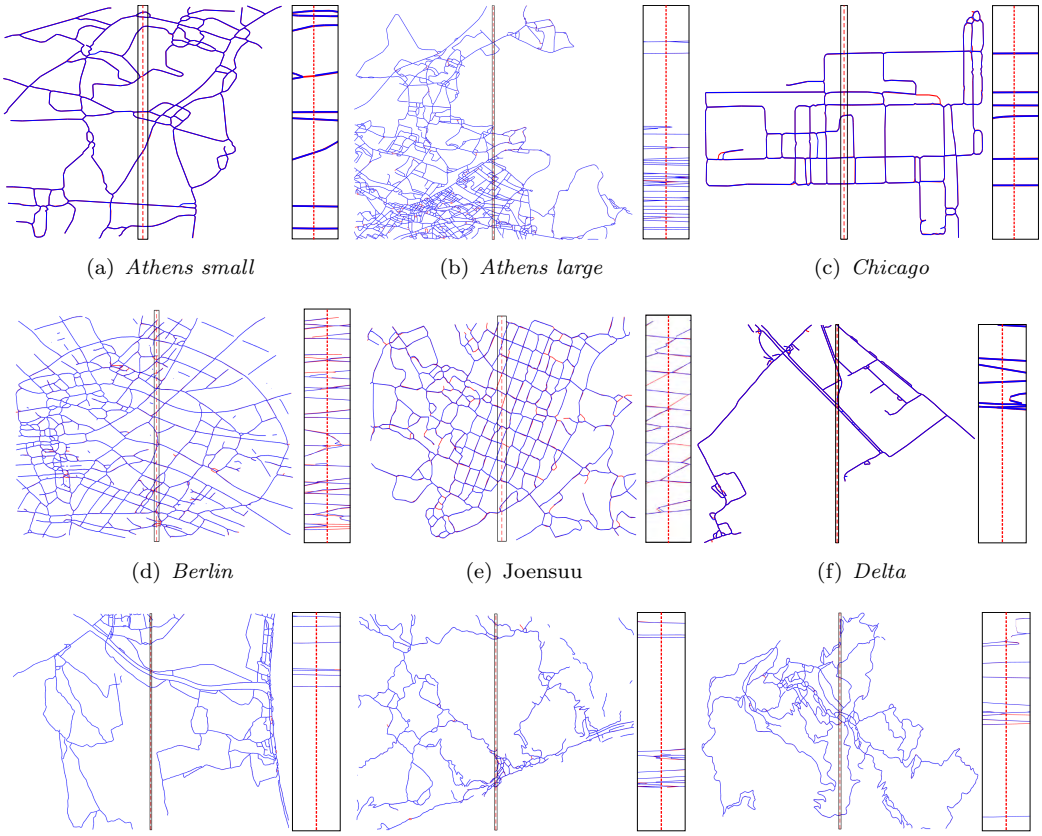
(m) KP (n) WWL (o) Ours



(p) KP (q) WWL (r) Ours

(s) KP (t) WWL (u) Ours

Figure B2. Artifact [$S1$] - [$S7$] (from up to down) in generated road networks. Columns left to right: result in (Duran *et al.* 2020), WWL, and our approach. Figures in the first column are taken from (Duran *et al.* 2020).



(a) *Athens small* (b) *Athens large* (c) *Chicago*

(d) *Berlin* (e) Joensuu (f) *Delta*

(g) *Aiguamolls* (h) *Garraf* (i) *Montseny*

Figure C1. Road network generated with our approach for different datasets. The single (in red) and split-and-merge (in blue) obtained route graphs are overlaid. The overlapping zone (in black) and the splitting line (red dotted) are marked and zoomed.