No. 2022-014

ENHANCING DIGITAL ROAD NETWORKS FOR BETTER OPERATIONS IN DEVELOPING COUNTRIES

By

V.F. Stienen, D. den Hertog,
J.C. Wagenaar, J.F. de Zegher

8 June 2022

ISSN 0924-7815
ISSN 2213-9532

Enhancing digital road networks for better operations in developing countries

V.F. Stienen[1], D. den Hertog[2], J.C. Wagenaar[1], J.F. de Zegher[3]

[1] Tilburg University, Department of Econometrics and Operations Research, Tilburg, The Netherlands.

[2] University of Amsterdam, Amsterdam Business School, Amsterdam, The Netherlands.

[3] Massachusetts Institute of Technology, Sloan School of Management, Operations Management, Cambridge MA, USA.

June, 2022

Abstract Data scarcity in developing countries often significantly complicates the use of analytics to address development challenges. One of the most fundamental data structures needed in operations management is digitized road data; *e.g.*, a poorly digitized road network significantly reduces our ability to optimize trade of micro-enterprises (SDG 8) and placement of hospitals (SDG 3). Unfortunately, current methods to extend or create digital road networks are not well-adapted to regions with sparse geospatial data and, as a result, road networks are often poorly represented digitally in less-developed regions such as rural areas of developing countries. To address this, we propose a novel method to create digital road networks in regions with sparse geospatial data, by adapting existing methods to ensure they extract as much information as possible from the limited available data. Our proposed method combines projection-based incremental insertion methods that incrementally add new information to existing road networks when it becomes available, with a simple edge adjustment procedure that allows edge geometries to be improved when more information becomes available. This method is well-suited to either incrementally adjust a large existing road network (*e.g.*, OSM) or combine multiple sources of road networks in regions with sparse data (*e.g.*, OSM and eStrada, a dataset provided by the World Bank). Our method significantly improves the digital road network for smallholder farmers in Indonesia, where only 40% of the origin-destination pairs in our dataset were previously digitized. In a case study of optimizing geospatial accessibility to healthcare in Timor-Leste, we find that the improved road network detects an additional 5% of people to be in the vicinity of a hospital.

Keywords Map construction/extension · Digital road networks · Optimization · Data-scarcity · GPS trajectories · Algorithms · SDGs

1 Introduction

In 2015 the United Nations adopted the 17 Sustainable Development Goals (SDGs), which called for action to end poverty, protect the planet, and ensure that by 2030 people around the world enjoy peace and prosperity. Analytics plays a crucial role in achieving the SDGs but access to high-quality data is often a challenge (Peters et al., 2022; Ergun et al., 2014; De Vries and Van Wassenhove, 2020; Besiou et al., 2021; Besiou and Van Wassenhove, 2020).

In this paper, we try to address this issue for one of the most fundamental data structures needed in service operations management: digitized road data. An accurate representation of a road network is, amongst other things, important to optimize the trade of micro-enterprises (SDG 8) and determining the placement of hospitals in a region (SDG 3). We are used to relying on road networks from OpenStreetMap (OSM) or Google Maps, but unfortunately these networks can be highly inaccurate in developing regions. This can lead to significant flaws in, for instance, routing and facility location problems.

We study the use of GPS trajectories, collected from vehicles equipped with GPS trackers, to complement such existing road networks. Using GPS trajectories to generate road maps has become a relevant alternative to using specialized vehicles or using satellite imagery, especially in regions where satellite imagery is of lower quality or the landscape is heavily forested (*e.g.*, tropics), or where economic incentives to use specialized road mapping vehicles are limited.

We propose a novel method that efficiently incorporates data from such GPS trajectories to create digital road networks in regions with sparse geospatial data. The same method can also be used to merge two different datasets of road networks (*e.g.*, OSM and eStrada) by simply transforming one of the datasets into a set of GPS trajectories. We apply this method to two case studies, in collaboration with PemPem (a marketplace for micro-enterprises in upstream commodity markets) and the World Bank, respectively.

Given the setting, we focus on extending digital road networks under the following four conditions. First, we assume that only limited GPS trajectory data is available, meaning that there *may not be much redundancy* on each road segment. Second, the GPS trajectories that are available may have a *limited sampling frequency* (*e.g.* 10 or 30 seconds on average), which means that the density of the GPS trajectories may be highly unevenly distributed. Third, we focus on extending road networks for the use in *optimization* problems. Finally, we prefer to use a method that easily adjusts a current network when additional road trajectories become available. Typically, we work with large networks, and new trajectories become known every day that must be incorporated in the network without regenerating the whole network all over again. In short, we want to exploit all information given in the GPS trajectories. For example, if a road segment has been driven only a few times, we still include this road segment in the extended network. Moreover, as we have ordered GPS points, we know that a vehicle has been driving between two consecutive GPS points. We want to include this information in the extended network.

Literature survey

In the literature, there exist two categories of extending digital road networks using GPS trajectories: (global) map construction methods, and (local) update methods. As the name suggests, global map construction methods (re)construct the entire network based on GPS trajectories. For an extensive literature review and introduction of map construction algorithms, we refer to Ahmed et al. (2015). They give an overview of state-of-the-art map construction algorithms, in which a differentiation is made between between three main methods: point-clustering methods, intersection linking methods, and incremental track insertion methods.

Point clustering methods cluster points to obtain intersections or street segments that describe the road network. Examples of these methods can be found in Davies et al. (2006), Biagioni and Eriksson (2012), Guo et al. (2020), Zhang et al. (2020), Mariescu-Istodor and Fränti (2018), Huang et al. (2018). Point clustering algorithms typically need dense samples of all the input trajectories, making them inappropriate for data scarce environments.

Intersection linking methods generally first find all intersection points and then link these intersections together. Examples are the work of Karagiorgou and Pfoser (2012) and, more recently, Mariescu-Istodor and Fränti (2018) and Alsahfi et al. (2019). Note that the first step, identifying the intersections, often relies on clustering methods; when only a limited number of points between intersections exist, connections between these intersections may be missed. In short, finding the right intersections also requires dense input trajectories, making intersection linking method inappropriate for our setting as well.

Incremental insertion methods create a map from scratch or extend a map by considering individual trajectories one by one and adding the information of the trajectory to the current network. Most of the existing incremental insertion methods make use of a trajectory matching algorithm that is based on comparing entire trajectories or GPS points with each other (see Ahmed and Wenk, 2012, Tang et al., 2017, Ni et al., 2018). Ahmed and Wenk (2012) start by combining similar GPS trajectories in order to accurately map the entire geometry onto an existing graph using the Fréchet distance. Tang et al. (2017) map trajectories onto the existing graph based on its location, using Euclidean distances. Ni et al. (2018) develop an algorithm that, after pre-processing the GPS trajectories, first extracts representative points from the GPS points in order to obtain more uniform distances between GPS points of a trajectory. Then, they match these representative points with points in the existing network. After matching, the information is used to update the current network. Where Ahmed and Wenk (2012) use a simplification method when adding unmatched portions, Tang et al. (2017) and Ni et al. (2018) use Delaunay triangulation to create or update edges. All of these proposed matching algorithms require streets to be well-sampled (have a high trace coverage). The trajectories must capture every feature of the shape of the original streets; if they do not, as in data scarce environments, parallel edges may arise which might cause streets to not be accurately connected.

Instead of matching with GPS trajectories, Zhang et al. (2017) develop an incremental insertion method that is based on sequentially projecting points onto the edges of the network. When projecting, they incorporate the amount of nodes traversed from the previous (projected) point. If a point could not be projected, a new edge is added from the previous point to the current point. One disadvantage of this method is that no adjustment to geometries of edges, or locations of nodes, can be made when new information becomes available. In data-scarce environments, where geometries of edges are learned over time, this negatively impacts the simplicity and accuracy of the final graph. Moreover, due to the time lag between consecutive GPS points, nodes/intersections may be skipped, resulting in points that are not immediately projected but should. Using Zhang et al. (2017)'s method, this can result in parallel (non-existent) edges in the final digital road network and may cause road segments to not be accurately connected.

The second category of extending digital road maps, local update methods, focuses on changing only parts of the network rather than reconstructing the entire network. Examples are the work of Wu et al. (2016) and Tang et al. (2019). These methods first find changes to the road network (*e.g.* using matching algorithms), and then reconstruct these localized parts using similar methods as discussed above for generating the entire network. Such update methods are more efficient since only a part of the network has to be adjusted.

Contributions
The first contribution of this paper is the introduction of a novel incremental insertion, local update method for regions with sparse geospatial data. To cope with the sparse geospatial data, we combine a new projection procedure (incorporating information about the distance between GPS points) with a simple edge adjustment method that allows edge geometries to be improved when more information becomes known. We propose an algorithm that is minimally affected by the order of GPS trajectories examined, and that can be solved efficiently, which is essential when dealing with large road networks.

Second, we quantify the impact of poor road network data in two case studies, using our proposed method. In Indonesia, in a region covering 7,000 km^2, we improve the digital road network for smallholder farmers, where only 40% of the origin-destination pairs in our dataset were previously digitized. In Timor-Leste, in a region covering 15,000 km^2, our extended road map can detect an additional 5% of people to be in the vicinity of a hospital. This added information can lead to better decisions about (new) hospital placements.

Finally, we do note that all code is publicly available at `https://github.com/valentijnstienen/PemPem-paper`.

2 Mathematical approach

In this section, we describe our mathematical approach and algorithm development. We start with a brief overview of the general idea of the algorithm. Thereafter, we describe the assumptions used in the algorithm. These are characterized as algorithm *settings*, which can be adjusted when desired. Finally, we describe and discuss the main algorithm. Some of the exceptional situations are discussed in Appendix C.

2.1 Overview

We start by explaining the general idea of the algorithm. The starting point of the algorithm is an initial graph. In the algorithm we use GPS trajectories to add edges and nodes to this initial graph, in order to end up with an extended graph. The GPS trajectories look as in Table 1.

Date	Latitude	Longitude	Speed	Course	MLDC	MDC
2020-04-11 09:52:05	-0.40833	102.61859	55.0	-1	-	-
2020-04-11 09:52:16	-0.40826	102.61706	61.0	272	(170.3, 186.4)	244.4
2020-04-11 09:52:25	-0.40812	102.6158	48.0	280	(141.0, 152.5)	200.0
2020-04-11 09:52:36	-0.40785	102.61473	37.0	291	(122.7, 146.7)	244.4

Table 1: Sample of a GPS trajectory that is used as input for the algorithm. The course is the heading of the vehicle at that moment, MLDC represents the Most Likely Distance Covered from the previous GPS point, and MDC represents the Maximum Distance Covered from the previous GPS point in the trajectory.

In this table, MLDC represents the Most Likely Distance Covered from the previous GPS point, and MDC represents the Maximum Distance Covered from the previous GPS point in the trajectory. Note that the MLDC is an interval that includes a lower and upper bound. The MLDC and MDC can be computed from the GPS trajectory data (for details, we refer to Appendix B.

In order to extend a given graph, the algorithm will sequentially go through all the GPS points in such a trajectory. In this process, it checks whether a point can be *absorbed* in the network. This means that this point was likely received when a vehicle was driving on an existing road. If a point could not be absorbed, we may initiate to change the existing graph. For instance, by adding new edges. For these (new) edges, we keep additional information that serves the algorithm. An existing set of edges looks as in Table 2.

ID	highway	oneway	length	geometry	maxspeed	u	v	key
1	primary	False	601	`LINESTRING` (99.5122 -0.7382, 100.514...	40 mph	1	3	0
2	residential	False	700	`LINESTRING` (99.5122 -0.7382, 100.514...	20 mph	1	3	1
3	trunk	False	139	`LINESTRING` (99.5485 -0.7669, 100.549...	15 mph	2	3	0
6	track	True	651	`LINESTRING` (99.5557 -0.7722, 100.555...	15 mph	6	5	0

Table 2: Sample of the edges of a graph that represents a road network. Highway indicates the importance of the road within the road network, and u (v) represents the ID of the start (end) node of this edge. Note that there may be multiple different roads (and therefore edges) between the same start and end point. These are distinguished using a key in the last column.

For each new edge, added by the algorithm, we store information about its connection points to the network. This information is stored in `close_to_point_start` and `close_to_point_end`, corresponding to the connection point of the start and end point of the edge, respectively. More specifically, we save the latitude, longitude, course, MLDC and MDC of the single point that was used to connect the new edge to the network. As will become clear, a connection point is made based on the information of a single GPS point.

An important element of the algorithm is *projecting* a given GPS point, say p. In general, this means that we try to find a point on the existing graph from which point p could have been received. As an example, consider the situation in which we want to project p, but we know p succeeds another (projected) GPS point p^c. This means that we include the MDC and the MLDC from p^c (to the projection point). So, we only consider points to be a projection point for which the distance from p^c to the projection point is smaller than the MDC from p^c to p. If no such point exists, the point cannot be projected. If at least one candidate projection point exists, we have to decide which point is the best projection point. As a performance metric, we now include the distance to the MLDC interval. Any point that lies within this interval receives a score of zero. The final projection point is the point that minimizes the projection distance and the distance to the MLDC interval (both weighed equally). In Figure 1, we visualize these two metrics.

Projecting a point onto an edge close to another point is a key element of the proposed algorithm. We illustrate the usefulness of this element by giving an example. If closeness is not incorporated, two consecutive points may be projected onto different edges in the network, while there is no (direct) possibility to go from the first point to the second point in the current network. Such a situation is sketched in Figure 2. When projecting p_n close to p_{n-1}, and not incorporating the distance covered information (M(L)DC), we would ignore the fact that there is no direct route possible between the two projected points. The consequence in this example is that p_n is either projected onto the edge on the left (if possible with a small enough distance), or it cannot be projected and a new edge has to

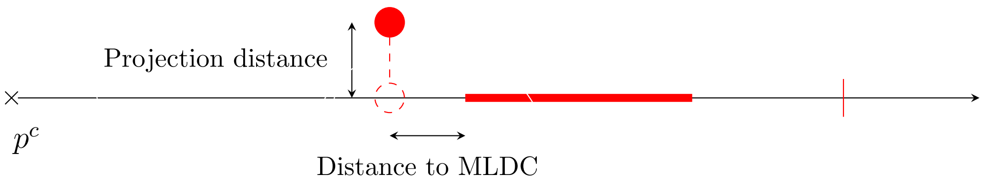


Figure 1: Illustration of the distance covered performance metric. When projecting a point (red dot) onto the network, we only consider candidate points that are reachable from p^c. In other words, the distance covered is smaller than the maximum distance covered (red mark). Moreover, we prefer to project within the region that is most likely reached since p^c (red thick line). The final projection point is the point that minimizes the sum of the projection distance and the distance to the MLDC region.

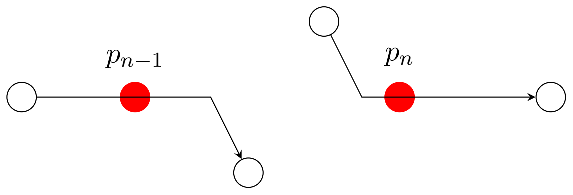


Figure 2: Example situation in which both p_{n-1} and p_n could be projected onto an edge in the network (using the maximum projection distance $\bar{d}$ and $\bar{\bar{d}}$), but there is no direct connection between the corresponding projection points.

be started. Both are convenient options. Note that incorporating the M(L)DC is especially valuable when there might be limited data available and a low frequency rate of receiving GPS points. In such cases, there is a larger possibility of missing (small) edges that occur between two consecutive GPS points.

We do note that there are other situations in which we want to *project* a point. For instance, when we do *not* need to project close to an other GPS point. For an elaborate explanation of all these scenarios, we refer to the definition and explanation of the `Project_point` procedure in Appendix C.1.1. As of this point, when projecting a point, we refer to this function; `Project_point`. Moreover, we will often make use of graphical visualizations to explain specific methods and procedures. For these visualizations, we use the consistent notation as summarized in Table 3.

2.2 Algorithm settings

The first setting of the algorithm that we define is the maximum distance for which we consider a point, say p, to be *close enough* to an edge for it to be considered as point on this edge. In the algorithm we will distinguish two types of edges. We have edges for which we assume the geometry is known (*e.g.*, edges that exist is OSM), and we have edges for which we are not sure about the geometry (*e.g.*, edges added by the algorithm). For both these types of edges, we have a separate maximum projection distance, $\bar{d}$ and $\bar{\bar{d}}$, respectively. Typically, $\bar{\bar{d}} \geq \bar{d}$, because if we are not (yet) sure about the geometry, points that are further away might still have been received when driving on that same road.

The second setting, $\bar{\alpha}$, represents the maximal angular difference between the bearing of a point p and the bearing of an edge onto which p may be projected. In other words, if a point is within $\bar{d}$ distance from an edge, and the bearing of this point differs by at most $\bar{\alpha}$ degrees from the bearing

Symbol	Definition
○	Node in the currently existing graph.
🔴	GPS point of a given trajectory.
⭕	The *projection* of a given GPS point onto the currently existing graph.
◌	Potential *projection* point of a given GPS point (projected onto the currently existing graph).
◌	End point of the geometry of an edge that is currently being constructed.
•	Interior (corner) point of the geometry of an existing edge (not one of the end nodes).
◌	Interior (corner) point of the geometry of an edge that is currently being constructed.
×	GPS point that is used as a close point when making a connection with the network (*i.e.*, `close_to_point_start` or `close_to_point_end`).
⟶	Edge in the currently existing graph.
⇢	Former edge (piece) that is now removed.
⇢	New edge that is currently being constructed.
- - - - -	Non-existent line (only used for illustration) between a GPS point and its (potential) projection point.

Table 3: The notation used for graph visualizations.

of this edge, this point is possibly received when driving on this edge. In Figure 3, we visualize the latter two algorithm settings.

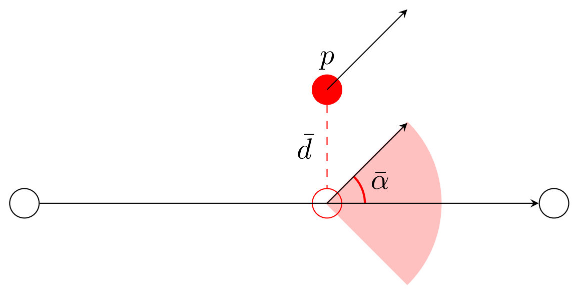


Figure 3: Illustration of thresholds used to check whether a point may be absorbed by an edge. $\bar{d}$ represents the maximum distance and $\bar{\alpha}$ represents the maximal angular difference.

The third setting of the algorithm is represented by $\bar{m}$, which indicates whether nodes are close enough to each other for them to be seen as one node. In other words, if two points are within $\bar{m}$ distance from each other, these points are considered to be a single point. As an example, consider the situation sketched in Figure 4. In this situation, we want to add an end node of a trajectory to the network. However, this end node has approximately the same coordinates as an existing node in the network, *e.g.*, a starting point of a previous GPS trajectory. In this example, we merge point p with a start point of an existing trajectory. Note that this ensures that there is no disconnection between the two trajectories.

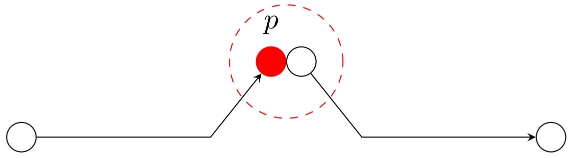


Figure 4: Example situation in which we merge point p with one of the existing nodes in the network.

The final setting of the algorithm, γ, indicates whether new edges are added in one direction or also in the opposite direction. For instance, we may want to assume that all roads that are traversed by trucks are bi-directional (set two-way indicator = `True`). All settings discussed are summarized in Table 4.

Setting	UoM	Symbol
Maximum projection distance, existing edges	m	$\bar{d}$
Maximum projection distance, new edges	m	$\bar{\bar{d}}$
Maximum difference in bearing	deg.	$\bar{\alpha}$
Merging distance	m	$\bar{m}$
Two-way indicator	`True/False`	γ

Table 4: Algorithm settings.

2.3 Algorithm

Next, we discuss the algorithm developed to create an accurate representation of the road network. We start with an initial graph and we sequentially consider GPS trajectories that may change this initial graph. We refer to Appendix A for details about creating the initial graph. Let G be the initial graph, a representation of the road network in a graph structure, where the edges represent roads and the nodes represent intersections or dead ends of roads. Let $\mathcal{P}$ be the ordered (with respect to time) set of GPS points. Then, the basic procedure of extending graph G, based on a single GPS trajectory, $\mathcal{P}$, is shown in Algorithm 1.

The first step of the algorithm is to select a part of the network that we may adjust (`line 2`). As an (extreme) example, we know that we are not adjusting parts of the network that are more than 100 km away from all of the points in the GPS trajectory. Considering only a part of the network per GPS trajectory makes it possible to apply the algorithm efficiently, even on large road networks. To do this, we start with finding the convex hull of all points in $\mathcal{P}$, with a *buffer* of 500 meters. All edges that have at least one endpoint inside this polygon are the edges that are relevant (and may therefore be adjusted), when processing GPS trajectory $\mathcal{P}$. Two examples are shown in Figure 5. After determining the relevant area, the algorithm sequentially evaluates the points of the (ordered)

Algorithm 1 Extending graph G using GPS trajectory $\mathcal{P}$

```
1: procedure ExtendGraph($G$, $\mathcal{P}$)
2:     $G' =$ Relevant_Area($G, \mathcal{P}$)
3:     $p_{-1} =$ None
4:     for $p \in \mathcal{P}$ do                                   ▷ For each GPS point in $\mathcal{P}$.
5:         if $p$ is StartPoint then
6:             Handle $p$ separately (see Appendix C.3)
7:             $p_{-1} = p$
8:             continue
9:         end if
10:        $p' =$ try_to_absorb($G'$, $p$, ($p_{-1}$, forward), connecting = False)
11:        if $p' \neq$ None then                  ▷ Point $p$ can be absorbed in the current network.
12:            Adjust_new_edges($p'$, situation = absorption)
13:            if $p$ is the last point of a new edge then
14:                Finish a new edge of $G'$ (see Section 2.3.2).
15:            end if
16:            $p_{-1} = p$
17:        else                              ▷ Point $p$ could not be absorbed in the current network.
18:            if $p$ is the first point of a new edge then
19:                Start a new edge of $G'$ (see Section 2.3.2).
20:                continue
21:            end if
22:            Adjust_new_edges($p'$, situation = creation)
23:            if $p$ is EndPoint then
24:                Handle $p$ separately (see Appendix C.3)
25:            else
26:                Incorporate $p$ into the current (newly created) edge.
27:            end if
28:        end if
29:    end for
30:    return $G$                                          ▷ Graph $G$ is now extended.
31: end procedure
```

GPS trajectory, $\mathcal{P}$ (`line 4`). As discussed before, the general idea of the algorithm is to try to *absorb* GPS points into the current network (`line 10`). Also recall that absorbing means that the GPS point under consideration could have been received while the vehicle was driving on the existing network. The two main events that may happen when, and after, a point could not be absorbed are starting and finishing a new edge. Both situations are visualized in Figure 6. In this figure, p_i represents the i^{th} GPS point after the current point p (indicated with a red dot). The second important part of the algorithm is the adjustment of geometries of edges for which we are uncertain about its geometry (`line 12,22`). For now, assume that all edges that we add to the network (all newly created edges) have an uncertain geometry. As an example, suppose that we have added a new edge to the network based on a single GPS trajectory. Then, we do not have any information about the geometry inbetween the GPS points that now form the interior points of the geometry. When we process another GPS trajectory that traverses this same road again, we want to incorporate the

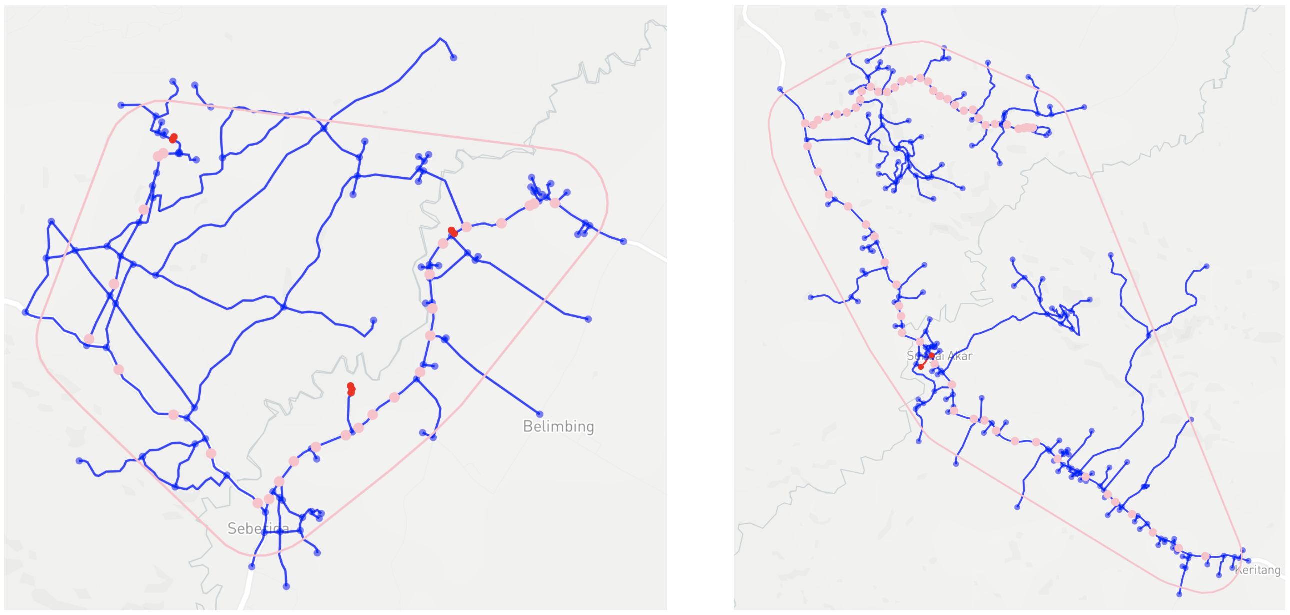


Figure 5: Polygon that is considered when processing the GPS trajectory containing the pink dots. Note that the blue edges are existing edges and the red edges represent edges that have previously been added by the algorithm.

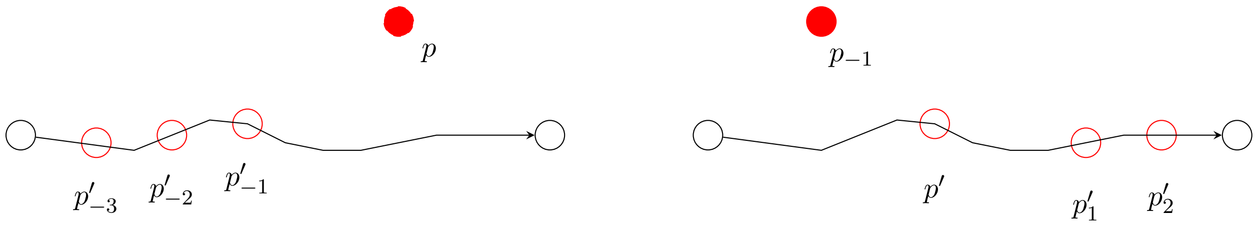


Figure 6: Left figure: point p is the first point that could not be absorbed, we start creating a new edge. Right figure: p is the first point that can be absorbed, after several other points could not. This means that we finish a newly created edge.

information of this new GPS trajectory into this edge (by adjusting its geometry). A possible edge adjustment is visualized in Figure 7. In the rest of this section, we will discuss the elements of the algorithm in more detail.

2.3.1 Absorption

Absorption of a point p into the existing network can, in general, be achieved in three ways: while *driving*, *turning*, or *merging*. All possibilities are visualized in Figure 8.

1. **Driving** In this case, we use the `Project_point` procedure (as described in Appendix C.1.1) to find a projection point. We set the `connect` parameter to `False`, because we do not want to establish a connection with the current network. The point can be absorbed when we do find a projection point that satisfies the requirements. Note that in the situation where we do not have a previous point, we do not incorporate the covered distance. The situation in which the previous point could be absorbed, is visualized in the left plot of Figure 8.

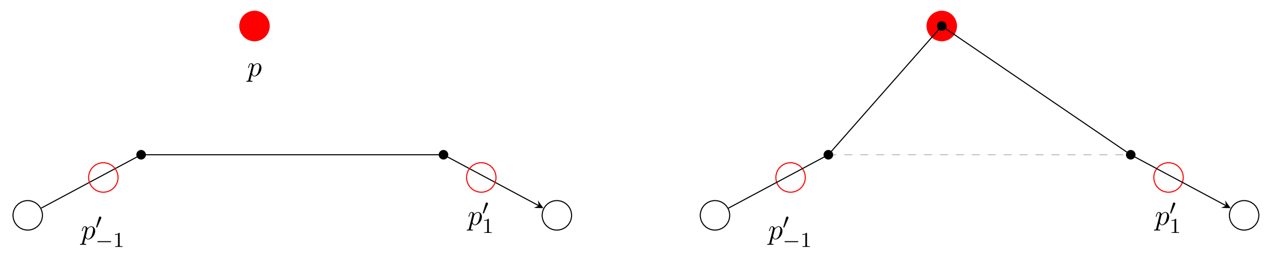


Figure 7: Adjusting the geometry of an existing newly created edge (black line). We are currently investigating GPS point p of a different GPS trajectory. Since the information of this point complements the information of the existing edge, we use it in the geometry of the existing edge.

2. **Turning** If p could not be absorbed while driving, we try to absorb it using *turning*. We find (if possible) the *opposite* edge, *i.e.*, the edge with the same geometry but in a reverse direction. We check whether we can project the point when we turn on the projection point of p_{-1} (which is the point to which p must be close, p^c). If so, we add a *turning* point on this road, and we consider point p absorbed. Note that turning only makes sense when we do incorporate the distance covered from the previous point. If not, turning is not used to absorb a point. An example situation is visualized in the middle plot of Figure 8.

3. **Merging** Finally, when we do not need to be close a (previous) point, and the point could not be merged while driving or turning, we check whether it is close enough ($\bar{m}$) to another node for it to be merged. Note that we do not allow for this type of absorption when we do need to be close to another point. For instance, when creating a new edge, we do not need to be close to a previous point. In this situation, we may absorb the point by merging. An example of such a situation is visualized in the right plot of Figure 8.

If, after checking these three options, we end up with a projection point, we conclude that point p can be absorbed ($p' \neq$ `None`, `line 11`).

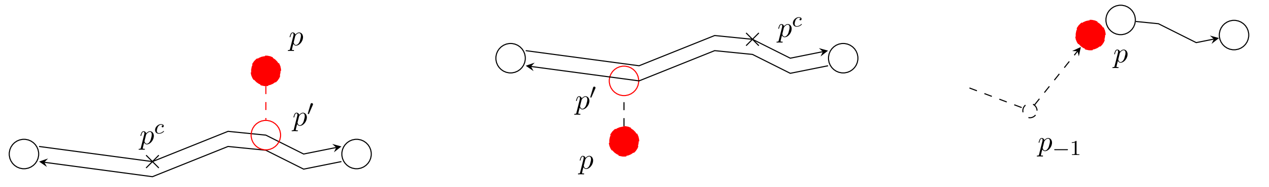


Figure 8: Absorption possibilities. Left figure: we can project p within the distance covered requirements while *driving*. Middle figure: we could not project p within the distance covered, but we can when we turn at p^c. Right figure: we are creating an edge (which means we do not need to be close to a point). We could not absorb p while driving (*e.g.*, due to the direction of p) or turning, but we can merge, and therefore absorb, p with an existing node.

2.3.2 Start/finish a new edge

Starting or finishing a new edge involves connecting edges to the network. We briefly discuss these two procedures below. First, We discuss how to start creating a new edge. In general, this happens, when a point cannot be absorbed in the network anymore (`line 18,19`). Assume that the previous point has been absorbed in the network. Instead of connecting p with the network by establishing an edge between p and p'_{-1}, we first project p onto an edge close to p'_{-1}, without incorporating the direction of this point p (using the `Project_point` procedure described in C.1.1). This is done in order to avoid (non-existing) parallel roads and increase the simplicity of the graph. Note that when projecting, we want to establish a connection with the current network. Therefore, we set the `connect` parameter in the `Project_point` procedure equal to `True`. Moreover, we want to be close to the previously projected point (that was absorbed). Therefore, we use the Most (Likely) Distance Covered (M(L)DC) of point p (that includes the distances covered between p and its previous point). This procedure is visualized in Figure 9, where we want to connect point p with the network, close to the point $p^c(= p'_{-1})$.

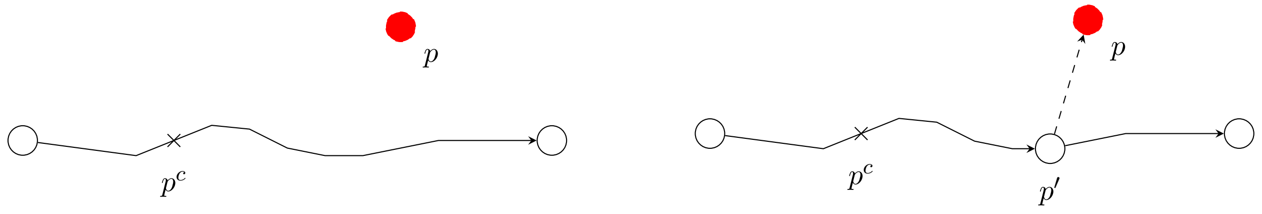


Figure 9: Starting a new edge by finding a connection point with the network based on p. Instead of using p^c as a connection point, we re-project p, using information about the distance covered, onto the network to obtain the connection of the new edge with the current network (right figure).

After adding/ensuring the connection point is in the network, we start creating a new edge. The first point of the geometry of this edge is this connection point. Next, we, again try to absorb the original GPS point (`line 20`). Now, there is no restriction on the distance covered from the previous point. This means that this point (that could not be absorbed before) may now be absorbed in this second phase.

Finishing an edge requires some more steps. Finishing an edge, in general, happens when a point, say p, is absorbed while creating a new edge (`line 13,14`). The first step in finishing an edge is to add the connection point to the network. Instead of connecting the previous point p_{-1} with the network by adding an edge between p_{-1} and p', we first project the previous point, p_{-1}, onto an edge close to p', without incorporating the direction of this point. As for the starting point, we want to establish a connection with the current network. Therefore, we set the `connect` parameter to `True`. We also want to be close to the next projected point (that was absorbed), but now in a `backward` manner. We again use the M(L)DC of point p. Once we find the right connection point, we add this point, p'_{-1}, as node and we split the edge onto which this node is projected into two edges. This procedure is visualized in Figure 10, where we want to connect p_{-1} with the network, close to the point $p^c(= p')$. Note that, as mentioned before, a point may be the first point that could not be absorbed, but also be the point that connects to the network again, due to the distance covered requirements were not met the first time. In this case, we do not re-project the previous point, but the start point of the new edge. For details about how this situation is treated we refer to Appendix C.2.

The next step, in finishing a new edge, is to incorporate the edge in the network. For this, we consider three possibilities: adjusting an existing newly created edge based on this new edge, adding the newly created edge, or doing nothing.

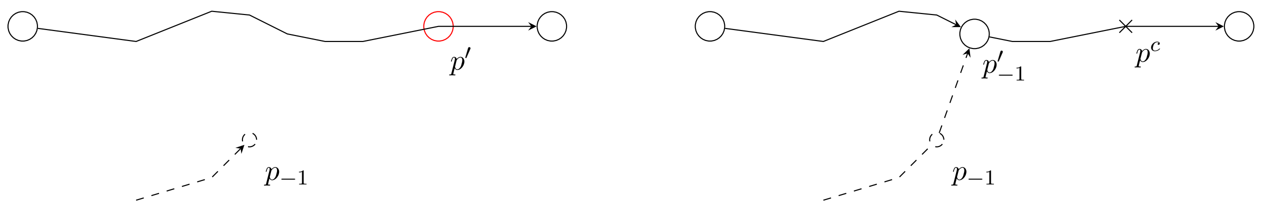
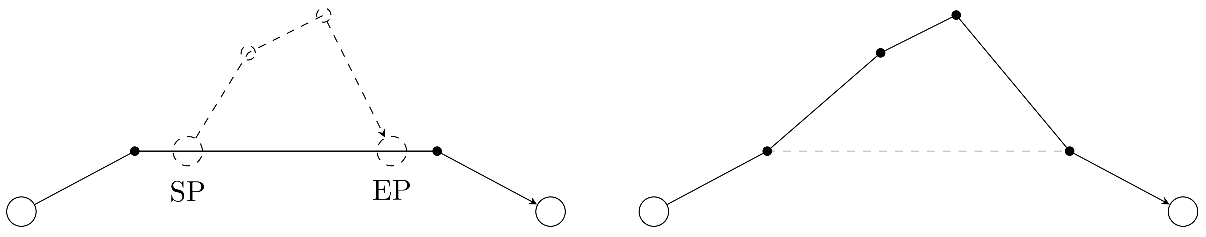


Figure 10: Finishing an edge by connecting it with the network. Instead of establishing a direct edge between p_{-1} and p', we re-project p_{-1} onto the network to obtain the connection of the new edge with the current network (right figure).

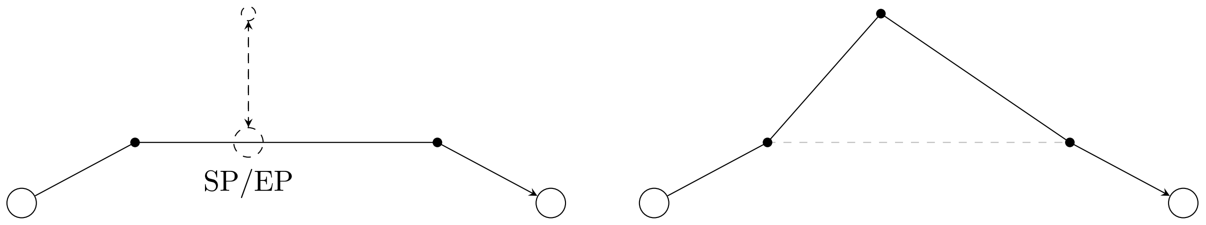
First, we check whether the new edge should adjust the geometry of an existing (newly created) edge e. This happens when the following conditions all hold:

- The potential new edge has a start **and** end point on edge e.
- Edge e has no interior points inbetween the start and end point of the new edge.
- The new edge contains at least 1 interior points. If it does not, any adjustments are not relevant.
- The new edge does not contain more than 5 interior points. If it does, we assume that adding another edge is more appropriate.
- The new edge is not a *self-edge* from (and to) an *existing* node. Note that a self-edge to a *newly added* node *is* used for adjusting existing (newly created) edges such as e.

We visualize two adjustment situations in Figure 11. Note that Figure 11b shows the last bullet point, in which a self-edge from (and to) a new node adjusts the geometry of a newly created edge. When the new edge is not used for adjusting the geometry of a newly created edge, we look at the (absolute) *difference* between the length of this new edge and the length of the shortest path between the start and end point of the new edge in the current graph (without using the potential new edge). If this difference is larger than two times the maximum projection distance, we add the new edge to the network. Otherwise, we do not add the edge. Note that there may be other (not shortest) paths between the start and end point of the new edge that have a similar length as the edge we want to add. As this may involve a lot of possible paths, we are not able to reliably tell if such an edge is similar as the edge we want to add. Therefore, we only compare the length of the new edge with the length of the shortest path between the start and end point of this new edge. Then, if we add the new edge to the network, we first determine the right key for this edge. We look for all edges that have a similar start *and* end point and look for the highest key. The new key will be this number plus one. In this way, we ensure that the new edge is an edge with a unique (u, v, *key*) pair. When the edge should not be added to the network, we remove the start point of this edge. The end point is kept as it is needed in the next iteration, in which we may want to project close to this point. Note that we will not remove the start point if it is also the end point of this edge or if this point was merged with an existing node.



(a) Example 1: Start and end point of the new edge differ.



(b) Example 2: Start and end point of the new edge are the same.

Figure 11: Two examples of adjusting newly created edges based on an edge that should be added to the network. The new edge (dashed) has a start and end point on the same line piece of a newly created edge (black solid line). We adjust its geometry and remove the start point of the new edge that was supposed to be added.

2.3.3 Adjusting new edges

Recall that there are two types of edges distinguished in the algorithm. We have edges for which we assume that the geometry is correct (*e.g.*, edges that exist is OSM), and we have edges added by the algorithm for which we are not sure about the geometry. Due to the limited data redundancy and low sampling frequency, it is likely that, initially, when we add a new edge, we may not approximate the actual geometry of the road very well. There is, at the time of adding the edge to the network, not enough information available to accurately predict the geometry of the road. When another GPS trajectory uses this road, we obtain more data points that may adjust the geometry of this edge. In this way, the order of examining GPS trajectories has a minimal influence on the final geometry. Recall that edges that are added by the algorithm, and which therefore have an uncertain geometry, are referred to as *newly created edges.*

In general, adjusting the geometry of an edge only involves adding a new point between the start and end point of a specific line piece of an edge. This procedure is visualized in Figure 12, where the gray polygon represents the actual road geometry, and the black dots represent interior points of the existing edge. This visualization also shows a situation in which adjustment of an edge improves the geometry of the edge.

When applying this method, two special cases require more adjustments: when a new GPS point is explicitly added to the geometry of a newly created edge *before the first*, or *after the last* interior point of that geometry. In these cases, we may have to adjust the connection points. For details about these special cases, we refer to Appendix C.1.2. For the sake of clarity, we refer to this adjustment procedure as the `Add_point_to_new_edge` procedure.

Now, there are two situations in which we may want to adjust the geometry of a newly created edge.

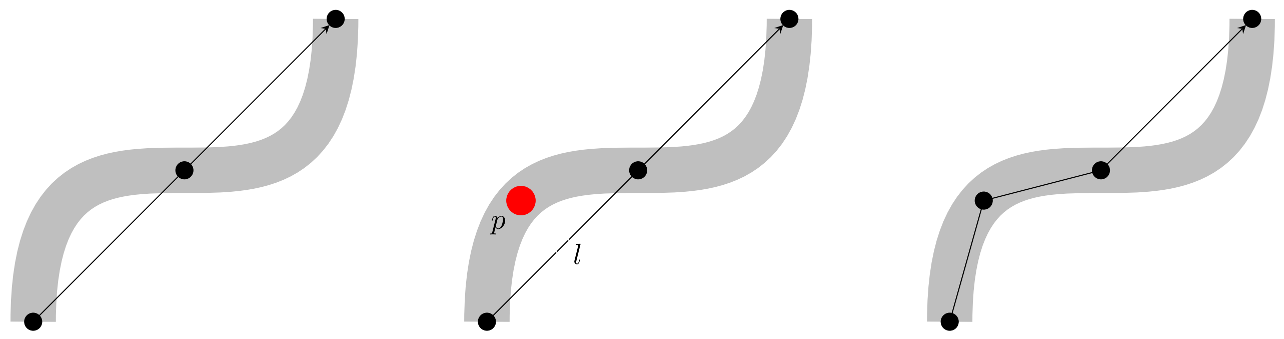


Figure 12: Adjusting the geometry of an edge with an uncertain geometry. We include point p explicitly in the geometry of the road, line piece l, which improves the geometry of the edge representing the road (gray polygon).

First, we want to adjust edges when **absorbing** a point in a newly created edge (`line 12`). More specifically, if we absorb a point in a newly created edge, we want to adjust the edge that absorbed the point. This situation is visualized in Figure 13a. Adjusting the geometry of this edge is done using the adjustment procedure `Add_point_to_new_edge`. Moreover, if there is an opposite edge, we also want to adjust, in a similar way, this opposite edge. To find the opposite edge, we find the edge that is closest to the original edge and within the maximum projection distance from this edge. Note that this way of adjusting is also performed in the special case of absorbing a starting point.

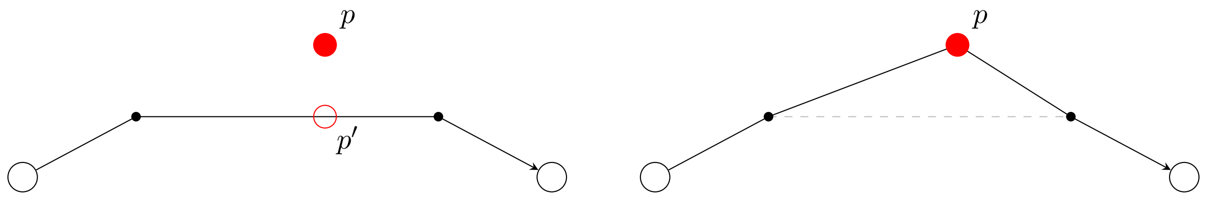
The second situation in which we may want to adjust the geometry of a newly created edge is when we are currently **creating** a new edge that is opposite to an existing edge (`line 22`). In this situation, we want to adjust the edge we are currently creating, but also the opposite edge (if there is one). Note that the opposite edge is assumed to be the closest edge to the current point (within the maximum projection distance). Let us first describe how to adjust the currently creating edge (based on the geometry of the opposite edge). When adjusting geometries, we exclude end points of edges to be used in the geometry of the currently creating edge. In other words, we only take over interior points from other edges to adjust a currently creating edge. Moreover, we only adjust a currently creating edge based on the geometry of a *single* (opposite) edge. We start by finding the projection point of the previous point and of the current point onto the opposite edge. Next, we look for any interior points inbetween these two projection points. If there are any, we add these in reverse order to the geometry of the currently creating edge. This situation is visualized in Figure 13b. After this, we adjust the opposite edge itself with the current point using the `Add_point_to_new_edge` procedure. Note that this situation boils down to the situation sketched in Figure 13a.

2.4 Final remarks

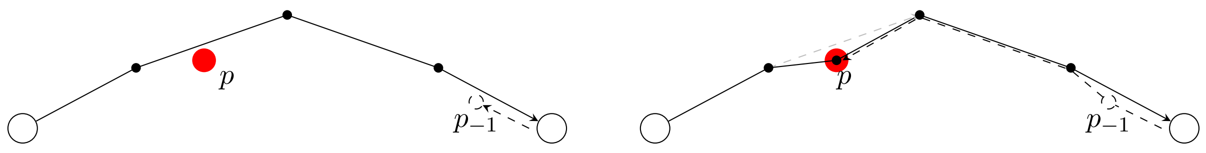
In the last part of this section, we describe two remarks about the algorithm.

2.4.1 Two-way setting

Recall that this setting means that we assume that all edges we add are two-way edges. If there exist one-way edges from the start (in OSM), we still treat them as one-way edges during the algorithm. When the two-way setting is activated, there are only a few things that change during the algorithm. First, when adding a node to the network, for instance adding connection points of new edges to



(a) Adjusting an edge based on the absorption of a single point.



(b) Adjusting an edge we are currently constructing (dashed) by including points of the opposite edge.

Figure 13: Adjusting an edge based on a single GPS point p.

the network, we now add this node in both directions (if possible). Secondly, when adding a new edge to the network, we also add the opposite edge to the network. Note that the opposite edge has therefore the same geometry, but in a reverse direction. Also, both added edges are classified as edges for which an opposite edge exists. Note that when edges are adjusted, these opposite edges will also be automatically adjusted by the algorithm.

2.4.2 Merging setting

When we are merging two existing road networks, we do have some additional information. As will be discussed, we know that each GPS trajectory represents a road. Since we know that there are no GPS trajectories of the same road, we will not adjust any existing newly created edges.

3 Numerical results and conclusions

In this section, we discuss the results of applying the proposed algorithm in two different case studies. The first case study is about PemPem (Section 3.1) and the second case study is about the World Bank (Section 3.2). The final results of the extended digital road networks of both the case studies can be viewed on `http://network-extension-app.herokuapp.com`. In this section, we present the results using metrics and we discuss some important observations. For a closer inspection of the final extended networks, we refer to this website. The calculations are done on a PC with a CPU that has a clock rate of 2.3 GHz and with 8GB RAM. Furthermore, the algorithm is implemented in Python.

3.1 PemPem

The first case study we discuss is done in cooperation with PemPem. The goal of PemPem is to make Enterprise Resource Planning (ERP) systems available to help smallholder farmers transition out of an informal cash-based economy, into a digital, cashless and technology-based economy (PemPem (2021)). This case study involves extending an existing road network, which can be used for routing decisions. In this study, we focus on a region in Sumatra (Indonesia) where PemPem is currently active. In this region, PemPem has access to the information in OSM, but this information turns out to be incomplete. Recently, they equipped trucks with GPS trackers, in order to generate streams of GPS traces of where a truck has been driving, at what time, and with what speed. The frequency rate of the trackers varies, in general, between 10, 30 and 60 seconds. We want to extend the current road network information (the information in OSM) with the information in the GPS trajectories. To do this, we first pre-process these trajectories. Pre-processing is done in order to create trajectories (which could be parts of current trajectories), that contain as few outliers as possible. Moreover, in this pre-processing algorithm, we determine additional information that is used in the algorithm, such as the most likely distance covered (MLDC), and the maximum distance covered (MDC). For details about the pre-processing algorithm, we refer to Appendix B. The result of this pre-processing algorithm consists of 14,846 final trajectories/trips. We now assume that if a truck has been driving on a particular location according to these trips, then this location represents a point on a road. So, if it is driving on a road that is not in our initial digital road network, we want to add this road to the network.

We start with an analysis about the quality of the initial graph. We perform a shortest path analysis for all the 14,846 origin destination pairs (OD pairs) in the GPS trajectory input. More specifically, we determine whether it is possible to find a shortest path for the OD pairs in the initial graph. There are two requirements for determining a shortest path between an OD pair. First, we should be able to project the starting point (O) and the end point (D) onto an edge. Naturally, this depends on the projection distance a point has to its edge. In this analysis, we distinguish three different maximum projection distances; 30, 50, and 70 meters. The higher this maximum projection distance, the higher the percentage of points that can be projected. Regarding the projection, we always project the origin and destination points onto the edges that are closest to these points. There is one exception, we also allow the point to be projected onto an edge that is not the closest, but which lies within 5 meters from the closest. In this way, we incorporate the possibility of projecting the point in the opposite direction (when possible). The second requirement for a shortest path between an OD pair to exist, is that it is possible to go from the origin to the destination point, using the roads in the graph. The results of the analysis are summarized in Table 5.

	Initial graph		
Maximum projection distance (meters)	30	50	70
O could not be projected	16.6	15.6	15.0
D could not be projected	14.8	14.2	13.4
O/D could not be projected	12.7	11.8	11.2
O/D could be projected, no SP exists	0.0	0.0	0.0
Total SPs that cannot be found	**44.1**	**41.6**	**39.6**

Table 5: The amount of shortest paths (SPs) that can be found in the initial network. The results are stated as percentages of the total number of shortest paths tried to be computed. Moreover, a distinction is made for the reasons why the SP could not be computed.

We indeed observe that when we increase the maximum projection distance, the amount of shortest paths that exists increases. However, even with a maximum projection distance of 70 meters, still only for 60.4% of the OD pairs can a shortest path be computed. In addition to this analysis, we examine whether having *similar* OD pairs has a large impact on the amount of shortest paths that can be computed. We examine the results when having only *unique* OD pairs. An OD pair is not unique if there exists another OD pair for which the origin and destination are close to the origin and destination of the original OD pair. We use a radius around the origin (destination) to determine whether another origin (destination) is close, referred to as the closeness radius (CR). We examine three different CRs, 5, 10 and 20 meters. Note that the higher the CR, the less OD pairs remain for shortest path computations. The results are shown in Table 6.

		Initial graph		
Maximum projection distance (meters)		30	50	70
All SPs	14,846 SPs	44.1	41.6	39.6
SPs (CR = 5m)	12,694 SPs	45.1	42.6	40.4
SPs (CR = 10m)	12,433 SPs	45.3	42.8	40.6
SPs (CR = 20m)	11,840 SPs	45.2	42.7	40.6

Table 6: The amount of shortest paths (SPs) that can be found in the initial network, now using different closeness radii (CR). The results are stated as percentages of the total number of shortest paths tried to be computed.

Based on this analysis, we conclude that having unique OD pairs does not change the results. In Figure 14, we visualize the locations of the points that can (not) be projected in the initial graph.

Next, we extend the initial graph with the information of the GPS trajectories (trips). The algorithm settings we use are summarized in Table 7.

Setting	UoM	Symbol	Value
Maximum projection distance, existing edges	m	$\bar{d}$	30
Maximum projection distance, new edges	m	$\bar{\bar{d}}$	50
Maximum difference in bearing	deg.	$\bar{\alpha}$	75
Merging distance	m	$\bar{m}$	10
Two-way indicator	`True/False`	γ	`False`

Table 7: Algorithm settings.

Besides these algorithm settings, we use different (random) orders in which we process the trips, each of these orders is referred to with a seed. Seed i refers to a specific order in which the trips are processed. There is one special order, namely Seed 00. This is the order in which the data was loaded. This means that for each day, the trips are in time order.

First, we compare the extended and original network based on the additional roads. Here, we distinguish two types of added roads. First, we add roads to the network that already existed in OSM, but that were not *driveable*. We assume that we can only route trucks over roads (or edges) that are currently classified as a "road" or a "road link" (see `https://wiki.openstreetmap.org/wiki/Key:highway`). These are, for instance, roads that are currently classified as walking paths and roads for

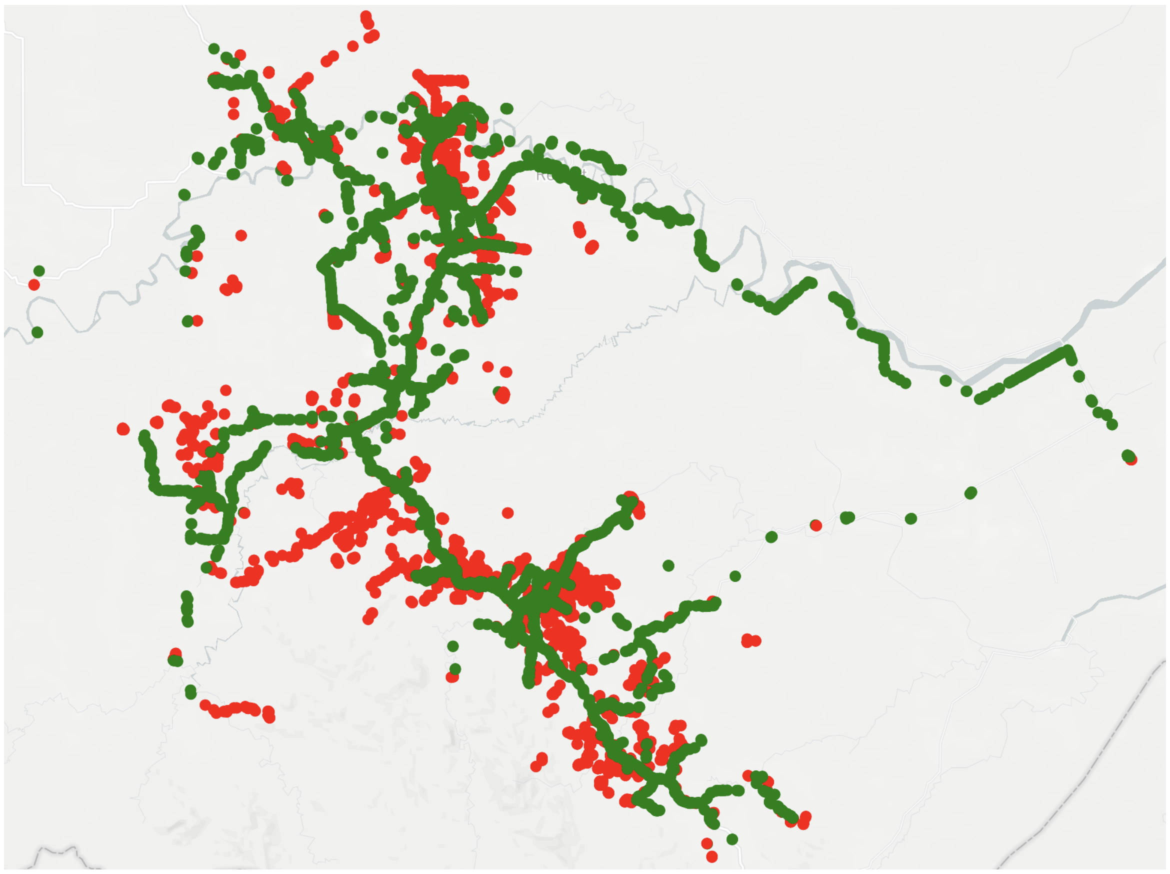


Figure 14: All start and end point of the OD pairs projected onto the initial graph. Green indicates that the point can be projected, red means that it can not. A maximum projection distance of 30m is used.

which the type is unknown. If a truck drove such a road, we classify it as driveable, which means that we can now use it for shortest path computations. The second type of road that we add are roads that are not in OSM at all, also not as a walking path. In Table 8, we summarize the results for different extended networks (based on different seeds).

	Initial graph	**Extended graph** Seed 00	Seed 0	Seed 1	Seed 2
Total amount of km	6,712	8,006	8,006	8,003	8,004
# of edges	24,811	31,381	31,292	31,345	31,337
Total amount of km added	-	1,294	1,294	1,291	1,292
Geometry in OSM	-	*927*	*928*	*927*	*927*
Geometry not in OSM	-	*367*	*366*	*364*	*365*

Table 8: Kilometers/edges added by the algorithm.

We observe that, for all different seeds, we have similar results. We increase the amount of driveable road kilometers with approximately 19.3%. More specifically, 13.8% is added using existing (non-driveable) roads in OSM and 5.5% is newly added. Based on this analysis, we conclude that the order of the trajectories does not affect the results significantly. In the rest of this section, we focus on the results that are obtained when using Seed 00. In Figure 15, we visualize the added roads for this seed. For a more interactive visualization, we again refer to http://network-extension-app.herokuapp.com. Note that Figure 15 aligns with the observations made in Figure 14, in the sense

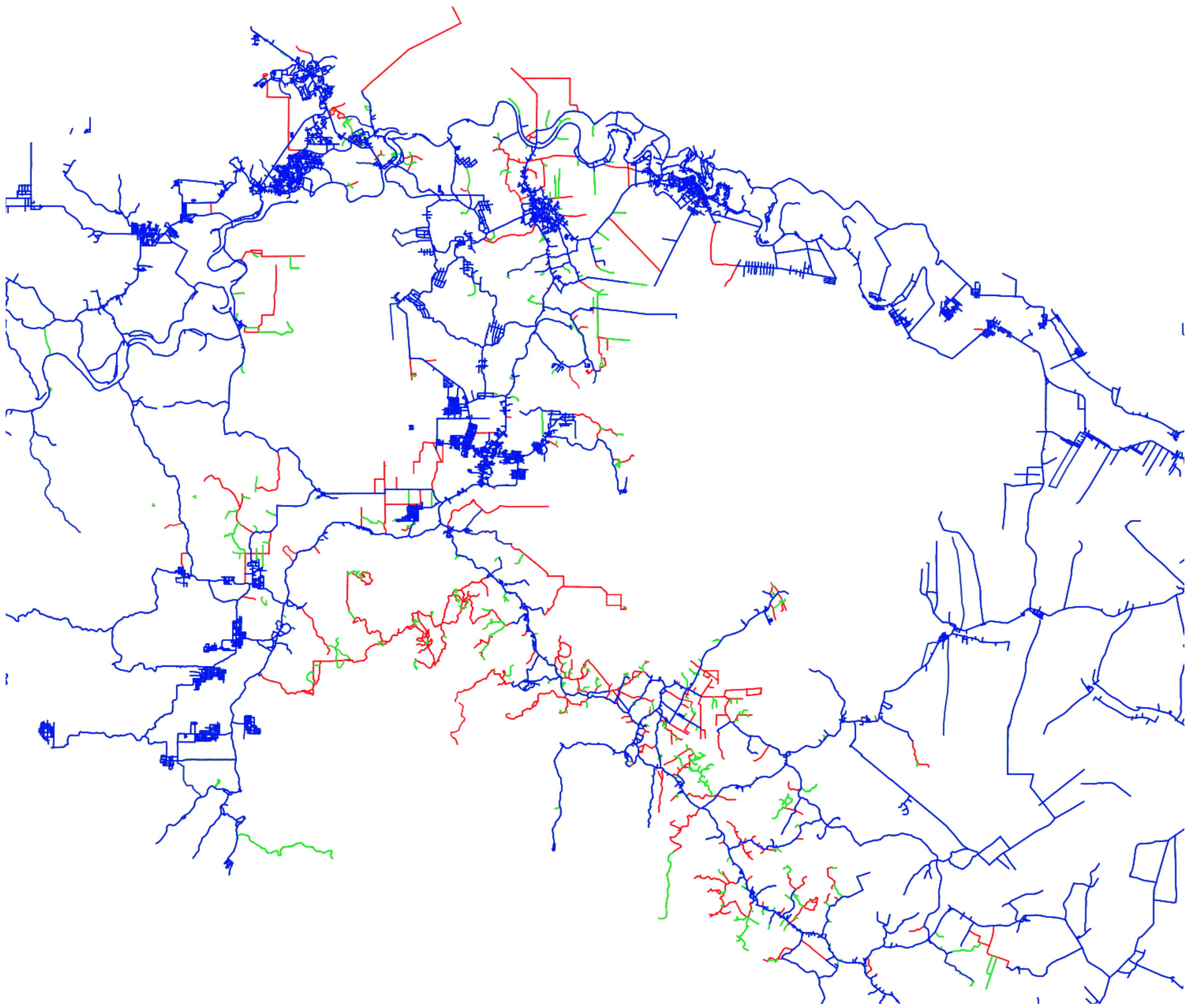


Figure 15: The red/green lines represent the roads that are added by the algorithm (Seed 00). The red lines already existed in OSM (but were not driveable) and the green lines did not exist in OSM.

that regions where points could not be projected, now have roads added by the algorithm.

Secondly, note that, by design of the algorithm, we know that, in the extended network, we will be able to find a shortest path for *almost* all OD pairs. Experiments show that using a projection distance of 30 meters, over 99% of the shortest paths can be found. Recall that when a point could not be projected, the algorithm started a new edge to make this projection possible. Therefore, it is logical that most of the points can be projected in the extended graph. A few exceptions exist, such as edges that may have its geometry changed over time, or starting points that, in the algorithm, were projected onto a newly created edge for which a higher maximum projection distance was used. Moreover, situations may arise in which both the origin and destination can be projected, but no shortest path can be found. Note that this may happen when points are projected onto different edges than when processed in the algorithm. Recall that, in these calculations, a point is projected onto the closest edge (or an edge within 5 meters from this closest edge). Therefore, in these cases, it may also happen that there does not exist a shortest path between the projected origin and the projected destination. As there are more edges in the extended graph, this is also more likely to happen in the extended graphs.

Thirdly, we examine the *reduction* in shortest path length when using the extended network. For the shortest paths, we again use the 14,846 origin destination pairs (OD pairs) from the GPS trajectory input that were used to extend the network. However, now, we only look at the OD pairs for which a shortest path exists in the old *and* in the new network. Note that an SP may exist in the old graph, whereas in the new graph it does not. This may result from other (closer) projection points. Moreover, we only look at the shortest paths that have their start and end points projected at the same spot, within a 5 meter radius. In Figure 16, we show a frequency plot of the reduction in shortest path length when using the extended network (Seed 00) over the initial network. In this plot, we only show the reductions that are larger than 1%. Note that a lot of the paths may be similar as before, having a 0% reduction. For reference, the average shortest path length in the initial graph is approximately 5.2 kilometers.

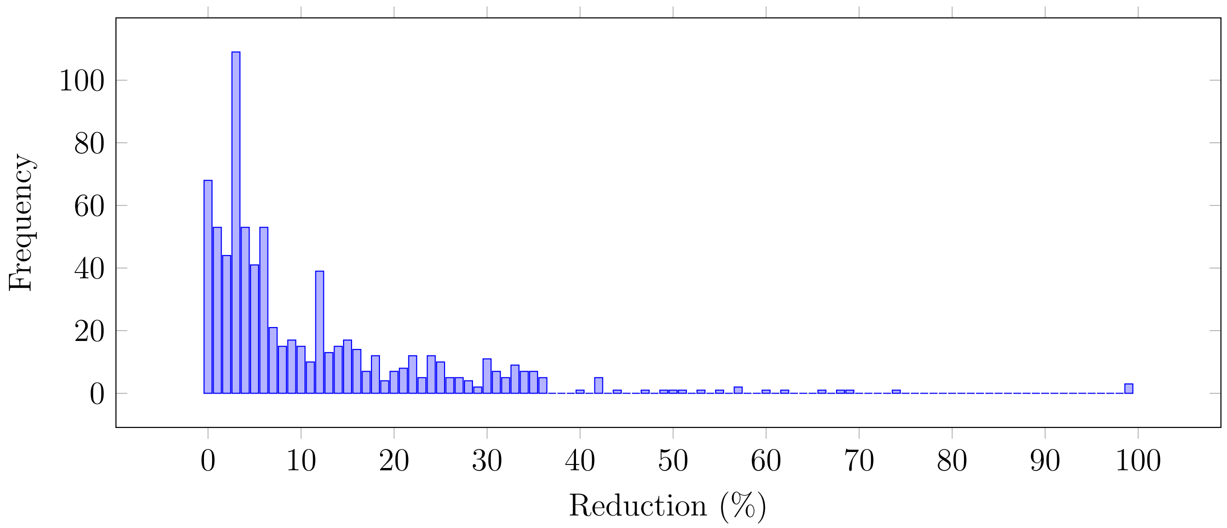


Figure 16: Percentage decreases of shortest path length (minimum of 1%). Bins have a width of 1%.

We observe that *if* a reduction is realized, the magnitude of this reduction is most likely between the 0 and 35 percent.

We conclude this subsection with a few words about the computation time. The computation time of the algorithm depends on the amount of trips that has to be processed. It takes approximately 2.0-2.5 seconds per trip. Moreover, there are, on average, 27.3 GPS points in a trip.

3.2 World Bank

The second case study we discuss is done in cooperation with the World Bank. This case study involves merging two existing road networks, in order to accurately predict distances to resources such as hospitals and schools. This case study is about the country Timor-Leste, located in South

East Asia. The World Bank has access to a database called *eStrada*, that contains information about the road network of Timor-Leste. However, this database is missing some important roads and road connections. Moreover, Open Street Map contains information about the road network, but with similar issues: parts of the country are not covered in the OSM database. Therefore, we would like to merge the two networks into one network. The merged network should contain all roads that are in either one of the two databases. Moreover, we want each road to occur just once. Note that it may be the case that the roads in both networks have (slightly) different geometries. If two roads represent the same actual road, we only want it to occur once in the merged network.

To do this, we take one of the road networks as the starting graph and we transform the other network into a set of GPS trajectories (see Appendix B for details about this procedure). Now, there are two possibilities: extending the OSM network with the roads in the eStrada network *or* extending the eStrada network with the roads in the OSM network. We compare both possibilities. In both cases, the starting graph contains roads in Timor-Leste, which covers an area of over 15,000 km^2. The settings that are used in the algorithm are summarized in Table 9.

Setting	UoM	Symbol	Value
Maximum projection distance, existing edges	m	$\bar{d}$	30
Maximum projection distance, new edges	m	$\bar{\bar{d}}$	30
Maximum difference in bearing	deg.	$\bar{\alpha}$	75
Merging distance	m	$\bar{m}$	10
Two-way indicator	`True/False`	γ	`True`

Table 9: Algorithm settings.

We set the two-way indicator to `True`, because we assume that all roads are bi-directional and each road in both digital road networks appears only once. Finally, we use the merging setting, which means that we do not adjust any edges during the algorithm. In Table 10, we state the main results of running the proposed algorithm to obtain the final merged network. For reference, we also include some numbers about the original non-merged networks themselves.

	OSM	eStrada	OSM + eStrada	eStrada + OSM
Total kilometers	6,858	6,009	8,515	8,356
# of edges	13,495	5,694	17,730	19,453
Total amount of km added	-	-	1,657	2,340

Table 10: Characteristics of the union network, compared to the input networks.

First, we observe that it does matter whether we choose to extend the OSM network with the roads in eStrada, or extend the eStrada network with the roads in OSM. If we choose to extend the OSM graph with the roads in eStrada, we end up with almost 160 km less than extending the eStrada graph with the roads in OSM. To explain this difference, we start with the observation that if starting with the OSM network, we add 1,100 roads of the eStrada network to the OSM network.

On the contrary, when starting with the eStrada network, we add 8,000 roads of the OSM network to the eStrada network. eStrada is a digital road network that does not contain all roads in dense areas (see Figure 17). This means that, when adding OSM to eStrada, we also add all roads in these dense areas. At the same time, new edges that do not satisfy the minimum (absolute) *difference* restriction for adding new edges are *not* added (see Figure 18). This means that when adding OSM to eStrada, we are likely to end up with a network that has less road kilometers than the network created when adding eStrada to OSM.

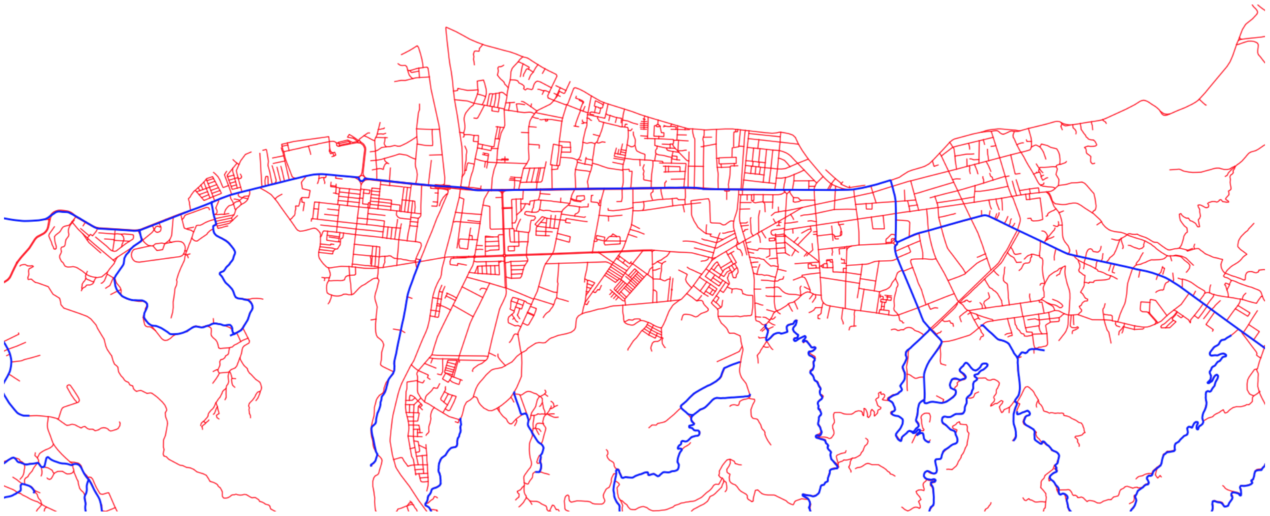


Figure 17: The OSM network (red edges) contains many roads in a dense area. The eStrada network (blue edges) is more focused on global connections, without incorporating all roads in dense areas.

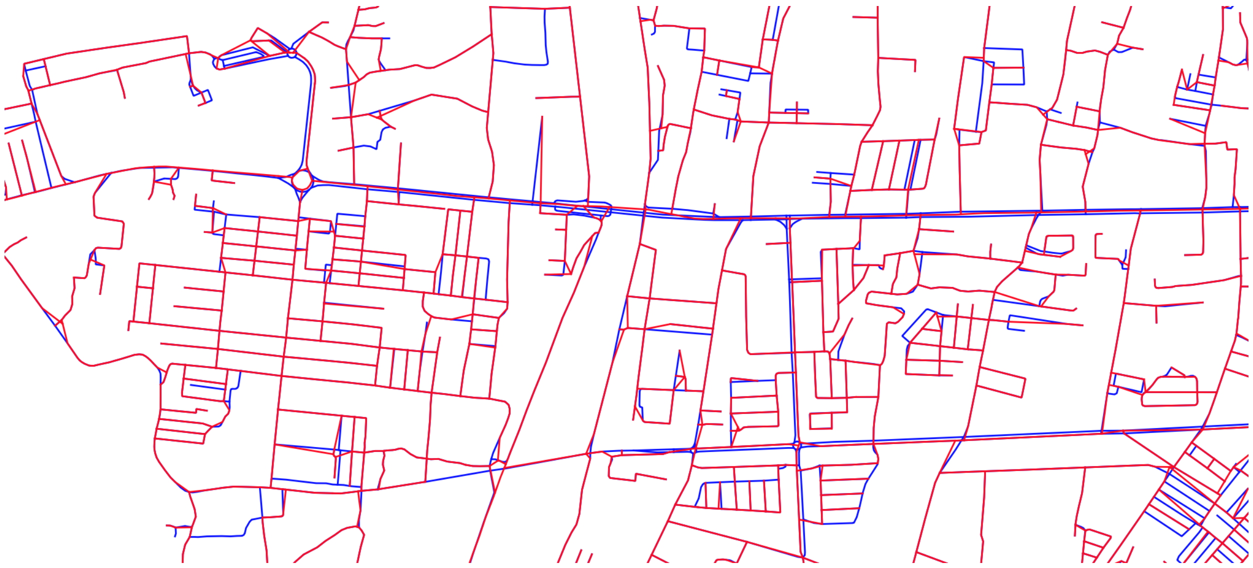


Figure 18: When adding the OSM network to the existing eStrada network, we obtain a new network (red edges) that does not include all roads in dense areas. Note that the edges that are in OSM, but were not added during the algorithm are colored blue.

Next, using analyses done by the World Bank, we show some results about the total *coverage* that a network of hospitals has when using all (merged) digital road networks. The coverage is defined as the percentage of households that has to travel less than z kilometers to their nearest hospital, where z is a given threshold. In Table 11, we show the coverage of the different networks for z equal to 2 and 5 kilometers.

z	OSM	eStrada	OSM + eStrada	eStrada + OSM
Current hospital coverage (2 km) in %	46.7	46.4	48.4	48.1
Current hospital coverage (5 km) in %	73.6	75.8	81.4	79.8

Table 11: Coverage of a network of hospitals when using the different road networks.

We observe that using both extended networks results in a higher coverage. For $z = 5$ km, we can achieve approximately a 8% increase in coverage by extending the OSM map with the information in eStrada. This means that a more accurate prediction can be given which households are in the vicinity of a hospital and which not. This, in turn, means that better decisions can be made about where to build additional hospitals in order to maximize the coverage. For example, we can avoid situations in which we recommend building new hospitals near households that are already in the vicinity of another hospital (using unknown, but existing roads).

To conclude, our method significantly improved the digital road network representations for smallholder farmers in Indonesia, where only 40% of the origin-destination pairs in our scarce dataset were previously digitized. Moreover, in a case study of optimizing geospatial accessibility to healthcare in Timor-Leste, we find that the improved road network detects an additional 5% of people to be in the vicinity of a hospital. Our results therefore demonstrate the importance of efficiently incorporating scarce geospatial datasets into digital road network representations in order to facilitate optimal trade and resource allocation in lower-income regions. Further research includes the estimation of travel times on roads based on the same GPS trajectories that are used to extend the digital road networks. Moreover, the pre-processing of the raw GPS data can be improved. In this paper, we assume that the GPS points are accurate. However, we still do observe several outliers that have a (large) impact on the final digital road networks. Reducing the amount of outliers present will improve the accuracy of the extended network.

Acknowledgement We would like to thank PemPem and the World Bank, for the collaboration and for providing us with the data/information for the case studies.

References

Ahmed, M., Karagiorgou, S., Pfoser, D., and Wenk, C. (2015). *Map Construction Algorithms*. Springer.

Ahmed, M. and Wenk, C. (2012). Constructing street networks from GPS trajectories. In *European Symposium on Algorithms*, pages 60–71. Springer.

Alsahfi, T., Almotairi, M., Elmasri, R., and Alshemaimri, B. (2019). Road map generation and feature extraction from GPS trajectories data. In *Proceedings of the 12th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, pages 1–10.

Besiou, M., Pedraza-Martinez, A. J., and Van Wassenhove, L. N. (2021). Humanitarian operations and the UN sustainable development goals. *Production and Operations Management*, 30(12):4343–4355.

Besiou, M. and Van Wassenhove, L. N. (2020). Humanitarian operations: A world of opportunity for relevant and impactful research. *Manufacturing & Service Operations Management*, 22(1):135–145.

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

Boeing, G. (2017). OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139.

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

De Vries, H. and Van Wassenhove, L. N. (2020). Do optimization models for humanitarian operations need a paradigm shift? *Production and Operations Management*, 29(1):55–61.

Ergun, Ö., Gui, L., Heier Stamm, J. L., Keskinocak, P., and Swann, J. (2014). Improving humanitarian operations through technology-enabled collaboration. *Production and Operations Management*, 23(6):1002–1014.

Guo, Y., Bardera, A., Fort, M., and Silveira, R. I. (2020). A scalable method to construct compact road networks from GPS trajectories. *International Journal of Geographical Information Science*, pages 1–37.

Huang, J., Deng, M., Tang, J., Hu, S., Liu, H., Wariyo, S., and He, J. (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 *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 89–98.

Mariescu-Istodor, R. and Fränti, P. (2018). CellNet: Inferring road networks from GPS trajectories. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 4(3):1–22.

Ni, Z., Xie, L., Xie, T., Shi, B., and Zheng, Y. (2018). Incremental road network generation based on vehicle trajectories. *ISPRS International Journal of Geo-Information*, 7(10):382.

PemPem (2021). `https://www.pempem.org`. Accessed: 2021-12-17.

Peters, K., Silva, S., Wolter, T. S., Anjos, L., Van Ettekoven, N., Combette, É., Melchiori, A., Fleuren, H., Den Hertog, D., and Ergun, Ö. (2022). UN world food programme: Toward zero hunger with analytics. *INFORMS Journal on Applied Analytics*, 52(1):8–26.

Tang, J., Deng, M., Huang, J., Liu, H., and Chen, X. (2019). An automatic method for detection and update of additive changes in road network with GPS trajectory data. *ISPRS International Journal of Geo-Information*, 8(9):411.

Tang, L., Ren, C., Liu, Z., and Li, Q. (2017). A road map refinement method using Delaunay triangulation for big trace data. *ISPRS International Journal of Geo-Information*, 6(2):45.

Wu, T., Xiang, L., and Gong, J. (2016). Updating road networks by local renewal from GPS trajectories. *ISPRS International Journal of Geo-Information*, 5(9):163.

Zhang, Y., Liu, J., Qian, X., Qiu, A., and Zhang, F. (2017). An automatic road network construction method using massive GPS trajectory data. *ISPRS International Journal of Geo-Information*, 6(12):400.

Zhang, Y., Zhang, Z., Huang, J., She, T., Deng, M., Fan, H., Xu, P., and Deng, X. (2020). A hybrid method to incrementally extract road networks using spatio-temporal trajectory data. *ISPRS International Journal of Geo-Information*, 9(4):186.

Appendices

A Initial graph

To create an initial graph, we can use information of roads that exist in OpenStreetMap (OSM). Using the work of Boeing (2017), we are able to initialize a road network (graph) that has the OSM features (*e.g.*, type of road, oneway, length,..). He develops a method to easily extract the information from OSM into graph structures in Python. In Table 12, we show a sample of how the edges of the graph are represented.

ID	highway	oneway	length	geometry	maxspeed	u	v	key
1	primary	False	601	LINESTRING (99.5122 -0.7382, 100.514...	40 mph	1	3	0
2	residential	False	700	LINESTRING (99.5122 -0.7382, 100.514...	20 mph	1	3	1
3	trunk	False	139	LINESTRING (99.5485 -0.7669, 100.549...	15 mph	2	3	0
6	track	True	651	LINESTRING (99.5557 -0.7722, 100.555...	15 mph	6	5	0

Table 12: Sample of the edges of a graph that represents a road network. Highway indicates the importance of the highway within the road network, and u (v) represents the ID of the start (end) node of this edge. Note that there may be multiple different roads (and therefore edges) between the same start and end point. These are distinguished using a key in the last column.

B Pre-processing algorithm

As discussed in the paper, a (large) set of GPS trajectories, that consists of GPS points, is used to extend a graph. The algorithm works best when the trajectories do not contain any outliers or *useless* points (*e.g.*, duplicates). This pre-processing algorithm is used to obtain trajectories (which could be parts of current trajectories), that contain as few outliers as possible. We refer to these new trajectories as *trips*. In other words, we want to create trips for which we can assume that the GPS points in a trip do not contain outliers, and are received in a consecutive order while the vehicle was driving on a road. Moreover, in this pre-processing algorithm, we want to obtain information that we can use to compute the Most Likely Distance Covered (MLDC), and the Maximum Distance Covered (MDC).

We distinguish two situations in which we run the pre-processing algorithm. First, we discuss the situation in which we have a set of consecutive GPS points (consisting of latitude/longitude pairs) that includes a time stamp and a speed/course registration of the vehicle at all times. Afterwards, we discuss the pre-processing algorithm that should be used when merging two existing road networks.

For this pre-processing, we use the following settings. First, a vehicle is assumed to be idle when its velocity is below threshold $v^{\min}$. In this situation, we consider the truck to be down/idle. Secondly, we define $t^{\max}$ as the maximum time a truck is down while driving the same trip. If a truck, for instance, has to wait for a traffic light, then we want the trip to continue. However, if the down time exceeds $t^{\max}$, we are not sure whether the truck has changed its location inbetween (while not sending its GPS location). Thirdly, $v^{\max}$ is the maximum speed that a truck may drive inbetween two GPS points. An obvious choice might be to use the speed limit of the region. This number is used to determine an upper bound on the distance covered inbetween two GPS points. Finally, we use three settings that determine how much information is kept in the trips. First, $\delta^{\min}$ is the minimum time inbetween GPS points. Occasionally, a GPS tracker might return its location after only a few seconds. Since this does not improve the final result of the algorithm, we exclude GPS points that are returned within $\delta^{\min}$ time from the last point saved in the trip. Secondly, we exclude GPS points that are within a radius of $d^{\min}$ from the last point saved in the trip. This slightly smoothens the trip, because close points do not occur anymore. Thirdly, we only add trips to the final dataset that contain at least $l^{\min}$ points.

Using these settings, we can preprocess a GPS trajectory into a set of trips. The proposed algorithm is visualized below in Algorithm 2.

Algorithm 2 Pre-processing GPS trajectory $\mathcal{T}$

```
1:  procedure PreProcess($\mathcal{T}$)
2:      Find subtrips, $\mathcal{S}$, that comprise consecutive GPS points (based on $v^{\min}$ and $t^{\max}$)
3:      $R$, $T' = \emptyset$, $\emptyset$
4:      for $s \in \mathcal{S}$ do                                        ▷ For each subtrip in $\mathcal{S}$
5:          for $p \in s$ do                                         ▷ For each GPS point in subtrip $s$
6:              if $T' = \emptyset$ then
7:                  $T' = T' \cup \{p\}$
8:                  $p_{-1} = p$
9:              else
10:                 LDC = distance($p$, $p_{-1}$, using location)          ▷ Linear distance covered (LDC)
11:                 if LDC $> d^{\min}$ then
12:                     MDC = distance($p$, $p_{-1}$, using velocity)   ▷ Maximum distance covered (MDC)
13:                     if (LDC < MDC) and (timediff($p, p_{-1}$) $> \delta^{\min}$) then
14:                         $T' = T' \cup \{p\}$.
15:                     else if LDC > MDC
16:                         if $|T'| \geq l^{\min}$ then
17:                             $R = R \cup T'$
18:                         end if
19:                         $T' = \{p\}$
20:                     end if
21:                 end if
22:             end if
23:         end for
24:         if $|T'| \geq l^{\min}$ then
25:             $R = R \cup T'$
26:         end if
27:     end for
28:     return $R$                                                          ▷ Final set of trips
29: end procedure
```

In Algorithm 2, we start with pre-processing a GPS trajectory (dataset) $\mathcal{T}$. We extract the *driving pieces* of the dataset and remove the points at which the vehicle is idle. This is done using $v^{\min}$ and

$t^{\max}$. Note that waiting for a time less than $t^{\max}$ is not considered as being in an idle state. The result is a set of subtrips $\mathcal{S}$ (`line 2`).

Next, let R be the final set of trips. We split the current set of subtrips even more as will become clear. We start with R being an empty set, after which we add trips to this final set in a sequential manner. Let T' be a set of (consecutive) GPS points that will form a trip that will be added to the final set R. Note that this set also starts as an empty set.

We start processing all GPS points that occur in $\mathcal{S}$. If we are creating a new trip ($T' = \emptyset$, `line 5`), we always add the current point to the trip that is being created (`line 6`). When having at least one point in the current trip, we check if we could add the next point to trip T'. We first check whether this point is not too close to the the last point in T' (use $d^{\min}$). More specifically, we compute the linear distance covered (LDC). This is the linear distance between the last point in T', and the current point that may be added to T' (`line 10`). If it is too close, we do not add the point and we start examining the next point in s. If it is not too close (`line 11`), we add the point if it could have been reached from the last point in T'. For this constraint, we compute the maximum distance covered (MDC, `line 12`), which is an upper bound on the distance that the vehicle had covered. This MDC is based on the distance that a vehicle could drive (using $v^{\max}$) within the amount of time between receiving this GPS point and receiving the last point of T'. Now, if the LDC is larger than the MDC, we could not reach the current point from the last point in T'. The truck most likely changed location without sending its GPS location(s). If the cardinality of T' is at least $l^{\min}$, we add the current trip, T', to R (`line 16-18`). Then, we start a new (temporary) trip T'. We immediately add the current point as the starting point of this temporary trip T' (`line 19`). If MDC is larger than LDC, which means that the point may be received during the same trip, we check whether the time between receiving this point and the last point in T' is larger than $\delta^{\min}$. If so, we assume that this point is received in the currently creating trip. We add the point to T' (`line 14`). If not, we start examining the next GPS point. After going through all the GPS points in s, we add the final temporary trip, T', to R, if the cardinality of T' is at least $l^{\min}$ (`line 24-26`).

During the pre-processing algorithm we also save additional information in R that is needed in the main algorithm discussed in Section 2 of the paper: the course of GPS points and the distance covered inbetween GPS points. First, if the course with a GPS point is unknown or uncertain we re-compute the course between this point and the next point (if possible). Alternatively, we can set the course equal to the course of the previous or next point. Secondly, we ideally want to know the distance covered inbetween all GPS points. As this is difficult to obtain using only the location and speed of the two GPS points, we use an interval: the *most likely* distance covered (MLDC) interval. The lower bound of this interval, the minimum most likely distance that is covered, is assumed to be the linear distance (LDC) between the current GPS point and the last point in T'. The upper bound of this interval, the maximum most likely distance covered, is based on the velocity and the timestaps of the current GPS point and the last point in T'. We have two speed registrations, one for this last point and one for the current point. There may exist points with an uncertain speed record. For these points, we can set the speed equal to the mean of the previous and next speed registration. We also have the time between these two GPS points. Therefore we can compute the distance covered when driving the speed of the last point, or the current point, the whole time. Using the maximum of these two speed registrations, we can compute the maximum most likely distance covered. Next to the MLDC, we also save the maximum distance covered (MDC). This number can be used to check whether it is possible to project on an edge. Note that this upper bound is different than the upper bound of the MLDC. The MLDC is an expectation, but it may be exceeded. The MDC is supposed to be certain. In Table 13, we show a sample of a trip from a pre-processed GPS trajectory.

Date	Latitude	Longitude	Speed	Course	MLDC	MDC
2020-04-11 09:52:05	-0.40833	102.61859	55.0	-1	-	-
2020-04-11 09:52:16	-0.40826	102.61706	61.0	272	(170.3, 186.4)	244.4
2020-04-11 09:52:25	-0.40812	102.6158	48.0	280	(141.0, 152.5)	200.0
2020-04-11 09:52:36	-0.40785	102.61473	37.0	291	(122.7, 146.7)	244.4

Table 13: Sample of a trip from a pre-processed GPS trajectory.

Note that for the first point of a trip, no distance computations can be made, as there is no previous point in the trip.

Finally, consider the situation in which we want to merge two existing networks (which do not rely on GPS trajectories). In this case, we slightly adjust the pre-processing algorithm. One of the graphs represents the initial (starting) graph. The other graph is transformed into a set of trips. To do this, we extract all corner points of the geometry of a road. This new list of points (or coordinates) is seen as a *trip*. Note that this means that there are no outliers in these trips, each point is a point on the road. Moreover, LDC, both bounds of MLDC and the MDC are considered to be the same; the linear distance between the corresponding GPS points. The course between two GPS points, which is also used in the final algorithm, can also be computed by using the location of both points. Note that since the speed of the vehicle is not used in the final algorithm, we do not (try to) include this quantity in the dataset.

C Algorithm

In this section, we start with elaborating on two specific (preliminary) functions that are used in the proposed algorithm: the `Project_point` and the `Add_point_to_new_edge` (Appendix C.1). Then, we discuss two exceptions used in the algorithm (Appendix C.2 andC.3).

C.1 Preliminary functions

First, we discuss two preliminary functions that are used in the algorithm. We describe how we project a point onto a given network and we describe how to adjust the geometry of an existing edge.

C.1.1 `Project_point(graph` $= G$, `point` $= p$, (`close_to_point` $= p^c$, `direction` $= d$), `connect` $= c$, `settings` $= s$)

We discuss how to project a point p onto graph G. There are multiple situations in which a point can be projected. Therefore, we include additional parameters that determine how the projection should be performed. First, the point may need to be projected close to another point, p^c, in a particular direction. Either, p^c is the previous point and we project p *after* p^c ($d =$ `forward`), or p^c is the next point and we project p *before* p^c ($d =$ `backward`). In both cases, we include information about the maximum distance covered (MDC) and the most likely distance covered (MLDC), from p^c to p ($d =$ `forward`), or from p to p^c ($d =$ `backward`). Secondly, we specify the goal of projection by indicating whether we want to use the projection point to connect it with the network ($c =$ `True`), or we use the projection point for absorption in the existing network ($c =$ `False`). Finally, we include all algorithm settings that are introduced the paper. For clarity, we show the used settings in Table 14. We distinguish three situations in which we may want to project a point.

Setting	UoM	Symbol
Maximum projection distance, existing edges	m	$\bar{d}$
Maximum projection distance, new edges	m	$\bar{\bar{d}}$
Maximum difference in bearing	deg.	$\bar{\alpha}$

Table 14: Algorithm settings used in the `Project_point` procedure.

First, we want to project p onto G when we do not need to be close to a point and we do not want to make a new connection with the network. To do this, we consider all edges that are within $\bar{d}$ distance from p (without incorporating the direction of the point and edge). For each of these edges, we find the best candidate point for being the projection of p *incorporating the direction of the point and edge.* The best point, and therefore the projection point, is the candidate point with the smallest distance to p while satisfying the maximal difference in direction, $\bar{\alpha}$.

Secondly, consider the situation in which we want to project a point when we do need to be close to another point with direction d, but we do not want to make a new connection with the network ($p^c \neq$ `None`, $c =$ `False`). In this case, we perform the same operations as for the case when we do not need to be close to a point. As before, the projection point must satisfy the maximal difference in heading, $\bar{\alpha}$, and it must be within $\bar{d}$ (or $\bar{\bar{d}}$) distance from p. However, it does not need to be the point with the smallest projection distance. In this case, we also use the MLDC and MDC.

To illustrate this usage, we show an example in which we need to project p in a `forward` direction. This means that we include the MDC and the MLDC from p^c (to the projection point). The situation in which we need to project p in a `backward` direction can be done in a similar manner (we do note that this latter situation is not used in the proposed algorithm). So, when projecting p in a `forward` direction, we only consider points to be a projection point for which the distance from p^c to the projection point is smaller than the MDC from p^c to p. If no such projection point exists, the point cannot be projected onto the graph G. If at least one candidate projection point exists, we have to decide which point is the best projection point. As a performance metric, we now include the distance to the MLDC interval. Any point that lies within this interval receives a score of zero. The final projection point is the point that minimizes the projection distance and the distance to the MLDC interval (both weighed equally). In Figure 19, we visualize these two metrics.

Note that this means that there could be multiple candidate projection points on one edge if these points satisfy the maximum projection distance and the maximum difference in heading. The best candidate point, or projection point, then, depends on the performance metric as described above. In Figure 20, we visualize how projection close to a point p^c, in a `forward` direction, is performed when we do (right plot) and when we would not (left plot) incorporate the distance covered.

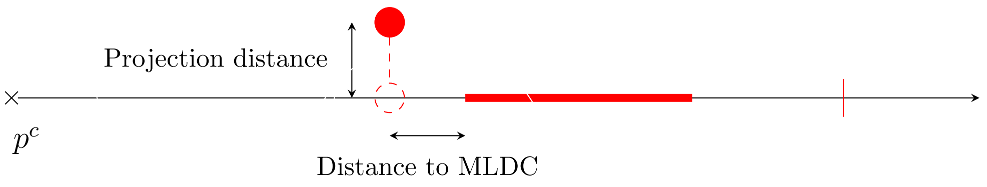
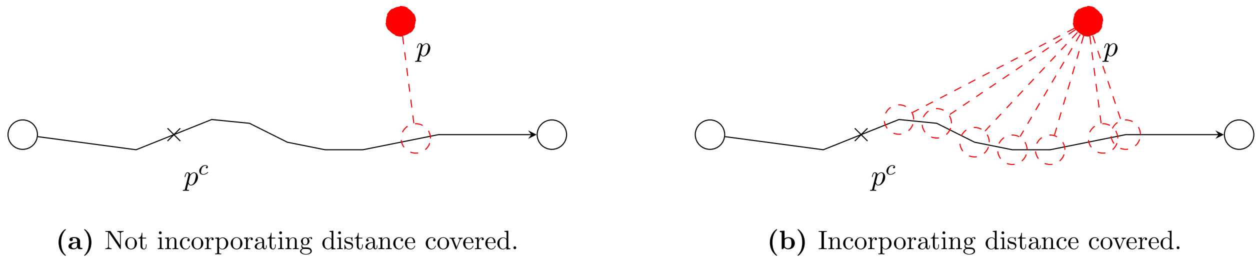


Figure 19: Illustration of the distance covered performance metric. When projecting a point (red dot) onto the network, we only consider candidate points that are reachable from p^c. In other words, the distance covered is smaller than the maximum distance covered (red mark). Moreover, we prefer to project within the region that is most likely reached since p^c (red thick line). The final projection point is the point that minimizes the sum of the projection distance and the distance to the MLDC region.



(a) Not incorporating distance covered. **(b)** Incorporating distance covered.

Figure 20: Projecting point p on the network, close to point p^c, in a `forward` direction. When not incorporating distance covered, the closest point to the edge is the possible projection point. When incorporating distance covered, we have for each line piece a possible projection point. The best one depends on the projection distance and the distance to the MLDC.

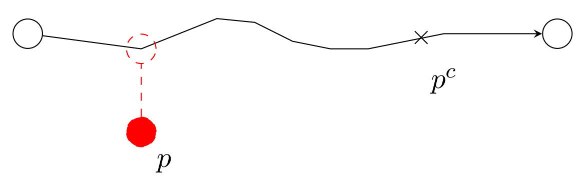
Thirdly, we consider the situation in which we want to project p onto the network in order to *connect* p with the network. Note that when connecting a point to the network, we *must* project the point, since we know that a connection exists. Therefore, we do not incorporate the maximum projection distance $\bar{d}$. We assume that if we establish a connection to the network, we have to be close to a point (p^c is defined) in either a `forward` or `backward` direction. Therefore, we again incorporate the distance covered from p^c, and we choose the projection point that minimizes the projection distance and the distance to the MLDC interval. In this case, projecting close to a point p^c, in a `forward` direction, is performed in a similar way, as when not making a connection with the network (see Figure 20). However, now, we do not include the maximum projection distance or difference in heading (the vehicle may have entered a road in a different direction). Moreover, now, the distance covered is the distance from p^c to the p (note that p is not absorbed). That means that the projection distance is now included in the computation for the distance covered.

For the sake of completeness, in Figure 21, we visualize the situation in which we project a point close to a point p^c in a `backward` direction.

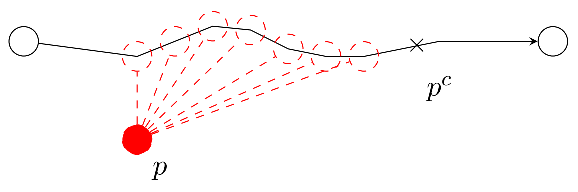
C.1.2 `Add_point_to_new_edge(edge` $= e$, `point` $= p$, `line_piece` $= l$)

This method is used to adjust the geometry of edge e, by including point p in this edge on line piece l. First, we only adjust edge e when p is not *close* to one of the existing interior points. In this way, we keep the geometry of edges as smooth as possible.

In general, adjusting the geometry of edge e only involves adding point p between the start and end point of line piece l. This procedure is visualized in Figure 22, where the gray polygon represents



(a) Not incorporating distance covered.



(b) Incorporating distance covered.

Figure 21: Connecting point p with the network close to point p^c in a `backward` direction. When not incorporating distance covered, the closest point to the edge is the possible projection point. When incorporating distance covered, we have for each line piece a possible projection point. The best one depends on the projection distance and the distance covered.

the actual road geometry, and the black dots represent interior points of the existing edge. This visualization also shows a situation in which adjustment of an edge improves the geometry of the edge.

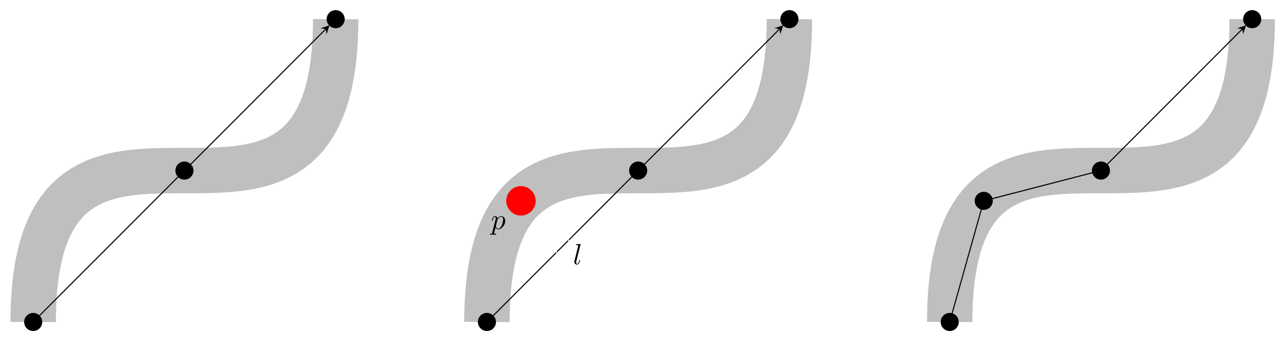
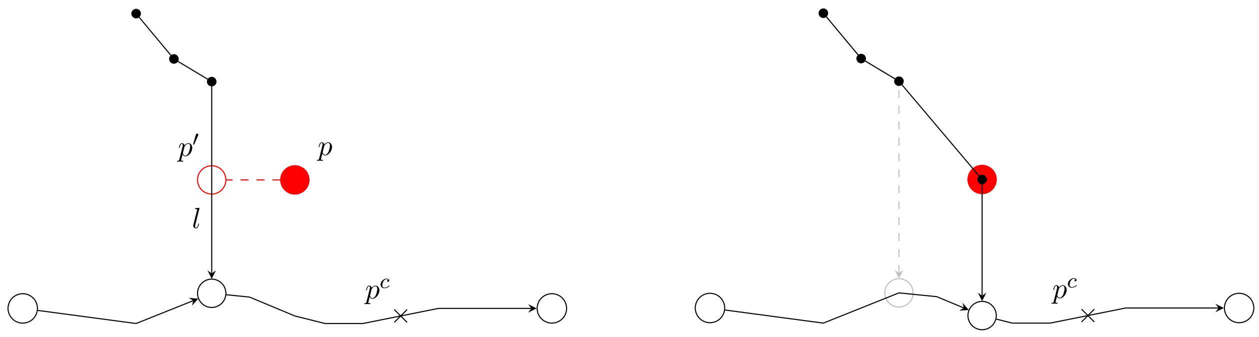


Figure 22: Adjusting the geometry of an edge with an uncertain geometry. We include point p explicitly in the geometry of the road, which improves the geometry of the edge representing the road (gray polygon).

When applying this method, two special cases require more adjustments: when a new GPS point is explicitly added to the geometry of a newly created edge *before the first*, or *after the last* interior point of that geometry. Note that the connection of a newly created edge to the network is determined based on the first or last interior point of a geometry, depending on the connection of the start or end point of the edge. Therefore, in both special cases, the connection point of the edge may need to be adjusted due to the addition of this new GPS point. We will illustrate this adjustment procedure for a point that is explicitly added *after* the last interior point of the geometry of a newly created edge. Consider the situation in Figure 23a. The existing edge that is coming from above is an edge that has previously been added by our algorithm. Note that the last interior point of this edge has been used to connect the edge with the current network (edge below). When connecting, the restriction was to project this last point close to p^c (in a backward direction) using the distance covered information at that time. Note that this information and p^c are both stored in the `close_to_point_end` attribute of this (newly created) edge e. We want to include point p in the geometry of the existing (newly created) edge on the piece between the last interior point and the connection point (see Figure 23a). We start with removing the edge piece between the last interior point of the geometry and the connection point. Then, we add point p as a new (last) interior point to this geometry, and we reconnect this edge onto the existing network. For this, we use the distance covered information that was used when the edge was created. In other words, we want to project p close to p^c (in a backward direction) using the distance covered information from the `close_to_point_end` attribute. One remark is that we do not use the lower bound of the most likely distance covered in this case, as we are now (most likely) getting closer to p^c. Therefore, we set this lower bound equal to zero in this case. The result is illustrated in Figure 23b.



(a) Original situation. **(b)** Adjusted situation.

Figure 23: Adjustment procedure when a point p is absorbed by an existing newly created edge, on the edge piece between the last interior point and the connection point.

The adjustment procedure for a point that is explicitly added *before* the first interior point of the geometry of a newly created edge is similar. In this case, we add point p as a new (first) interior point to this geometry, and we reconnect this edge onto the existing network with a restriction that we must be close to point p^c, in a forward direction. There is one additional difference. In this case, we can update the `close_to_point_start` from edge e. To find the new connection point, we can now use the previous point that was absorbed (instead of the `close_to_point_start` from e). After that, we can also store this information as the new `close_to_point_start` of e (as this point is, most likely, closer to the "real" connection point). When adding a point *after* the last GPS point of an edge, we do not know anything about the connection point of that edge. Therefore, only when we add a point before the first interior point, we use the previous point and update the `close_to_point_start` information.

C.2 Point that can and can not be absorbed

Note that, as discussed in the paper, a GPS point may be the first point that could not be absorbed, but also be the point that connects to the network again, due to the distance covered requirements were not met the first time. In this case, we do not re-project the previous point, but we project the start point of the new edge. As this start point was created based on p, and we want the end point to be again close to p, we expect that a minimal distance is covered inbetween these points. Therefore, we use zero as lower and upper bound in the MLDC interval, when projecting the start point close to p. The new projected point is added as node and we split the edge onto which this node exists into two edges. This situation is sketched in Figure 24.

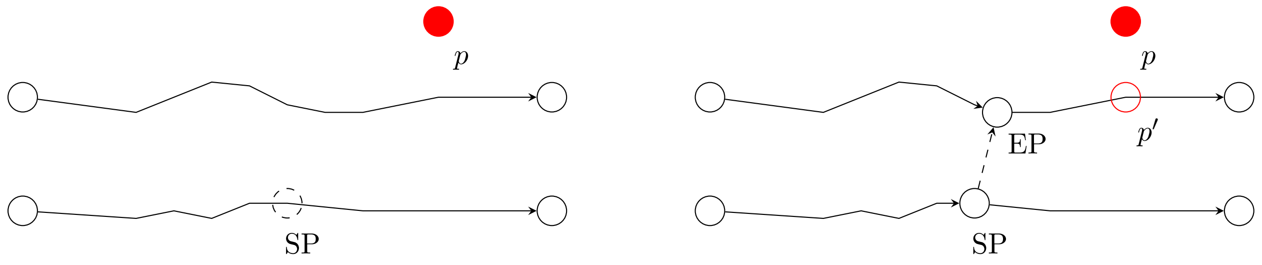


Figure 24: Finishing an edge that is based on a point that is the first point that could not be absorbed, but also be the point that connects the edge to the network again. SP (EP) is the start (end) point of the new edge.

C.3 Handling the start/end point

In the paper, we discuss how a GPS trajectory is used to extend a given graph. We do this using examples of GPS points that often have at least one predecessor and one successor. However, obviously, this is not the case for the start and end point of a GPS trajectory. Next, we discuss the procedure for handling these two special points. This differs only slightly from the regular situation with at least one predecessor and successor.

We start with discussing the starting point. As for other points, we try to absorb this point in the existing network. Note that this is done by *driving* or *merging*, and that we do not incorporate the distance covered, as there is no previous point. This also means that absorption by turning does not make sense. If the start point could be absorbed, we store the projection information and move on to the next point. If the start point could not be absorbed, we add this point explicitly. We first check if we can project the point onto an existing edge, without incorporating the direction of the point. If the start point can now be projected, we add the projected start point to the network, either explicitly or by merging with an existing point. If we add the point explicitly, we make sure that it is added in two ways (if possible), because we do not have any relevant information about the direction of this point (note that it could not be absorbed). We store the projection information and move on to the next point. Note that this means that a connection is established with the existing network. If the start point could not be projected onto an existing edge, we add the start point as a new node and start a new edge (with no connection to the existing network).

For end points, we require additional steps. When an endpoint could not be absorbed, we know that we were creating a new edge. The end point was not absorbed, but, now, we still do need to finish the edge. We first check if we can project the end point onto an existing edge (without incorporating the direction of the last point). Since we do not use the distance information when creating a new edge (we do not need to be close to a previous point because the previous point could not be absorbed), we just look at the maximum projection distance. If the end point could be projected, we add the projected end point to the network, either explicitly or by merging with an existing point. If we add the point explicitly, we make sure that it is added in two ways (if possible), because we do not have any relevant information about the direction of this point. If the end point could not be projected onto an existing edge, we add the end point as a new node.

Next, we check, as for finishing a regular new edge, whether the edge must be added or ignored. Note that this edge will not be used to adjust the geometry of an existing newly created edge, as this end point was not absorbed. Therefore, the requirements for adjustment will never be met. So, we look at the (absolute) difference between the length of this new edge and the length of the shortest path between the start and end point of the new edge in the current graph (without the new edge). If this difference is larger than two times the maximum projection distance, we add the new edge to the network. Otherwise, we do not add the edge.

D Results

When discussing the PemPem case study, we used 14,846 trips to extend the initial graph. These trips are obtained using the pre-processing process, as described in Appendix B. For this case study (PemPem), we use data of 2,047 files. Each file registers the GPS trajectory of a vehicle on one specific day. The average time inbetween consecutive GPS points ranges from 10 to 60 to 90 seconds, depending on the tracking device used. We use the pre-processing algorithm to create trips from these files. Note that this involves removing potential outliers of GPS trajectories and duplicate consecutive GPS points. The pre-processing algorithm settings used are summarized in Table 15. Note that these settings are chosen in cooperation with PemPem. They can be adjusted to a specific case, as is explained in Appendix B.

Setting	UoM	Symbol	Run 1
Down speed	km/h	$v^{\min}$	3
Maximum down time	s	$t^{\max}$	125
Maximum speed	km/h	$v^{\max}$	80
Minimum linear distance covered	m	$d^{\min}$	10
Minimum time between points	s	$\delta^{\min}$	5
Minimum trip length	points	$l^{\min}$	5

Table 15: Pre-processing algorithm settings.

In Table 16, we state the main results of applying the pre-processing algorithm to the set of GPS trajectories. Out of the 14,960 OD pairs, 114 are beyond the considered region that we would like

	Full dataset	Pre-processed dataset
# GPS points	4,143,392	408,475
Number of trips	-	14,960

Table 16: Characteristics of the 2,047 files

to extend. Therefore, these 114 trips are disregarded, which means that 14,846 trips are used in the extension algorithm. Ideally, each of the trips now consists of a set of consecutive GPS points that represent locations on the road onto which the corresponding vehicle was driving. In other words, a road exists between two consecutive GPS points.