JOURNAL OF LOCATION BASED SERVICES
https://doi.org/10.1080/17489725.2025.2586277

Taylor & Francis
Taylor & Francis Group

Using iterated local alignment to aggregate trajectory data into a traffic flow map

Tarn Duong

Independent Researcher, Paris, France

ABSTRACT

Vehicle trajectories are a promising GNSS (Global Navigation Satellite System) data source to compute multi-scale traffic flow maps ranging from the city/regional level to the road level. The main obstacle is that trajectory data are prone to measurement noise. While this is negligible for city level, large-scale flow aggregation, it poses substantial difficulties for road level, small-scale aggregation. To overcome these difficulties, we introduce innovative local alignment algorithms, where we infer road segments to serve as local reference segments, and proceed to align nearby road segments to them. We deploy these algorithms in an iterative workflow to compute locally aligned flow maps. By applying this workflow to synthetic and empirical trajectories, we verify that our locally aligned flow maps provide high levels of accuracy and spatial resolution of flow aggregation at multiple scales for static and interactive maps.

ARTICLE HISTORY
Received 15 June 2025
Accepted 20 October 2025

KEYWORDS
GNSS; GPS; multi-scale; rasterisation; route finding

1. Introduction

One of the fundamental quantities in transport planning is a traffic flow map, i.e. a map of the traffic flow levels on the road segments in a road network (Ortúzar and Willumsen 2011). Whilst traffic flow maps are a rich source of information about vehicle mobility patterns, they are costly in terms of time and resources to compute for any reasonably sized road network. To alleviate this cost burden, most approaches restrict the spatial resolution of the flow map. One of the most common is to place sensors at fixed locations in the road network, whose results are then visualized as a traffic count map. Another common approach are trip intent/recall questionnaires to inform large-scale properties such as trajectory origins and destinations, whose results are then visualized as a desire line flow map/spider diagram (Tobler 1987). Both of these are simplified flow maps, since the detailed mobility patterns outside of the sensor locations or origin/destination pairs are not known. These unknown patterns can be inferred from other data sources, such as route assignment models (Evans 1976). While these model-assigned routes are highly detailed, the trade-off is that they are not guaranteed to correspond closely to empirical mobility patterns.

Trajectory data, derived from GNSS (Global Navigation Satellite Systems), combine the large-scale information of road sensor counts or questionnaires, with the small-scale details

CONTACT Tarn Duong tarn.duong@gmail.com

of model-assigned trajectories. Due to the prevalence of GNSS-enabled devices, such as vehicle navigation guides and mobile telephones, trajectory data can be acquired with low marginal cost whilst at the same time offering extensive spatial coverage and resolution of empirical mobility patterns (Andrienko and Andrienko 2013; Herrera et al. 2010). Our motivating example of trajectory data are the 1183 trajectories in EPSG 32632 (WGS 84/ UTM zone 32N) projected coordinates, collected from a GNSS-enabled mobile phone, from December 2017 to March 2019 by a single driver in Hannover, Germany, with an overall average sampling rate about 1 point per second (Zourlidou, Golze, and Sester 2022). In Figure 1(a), at the city level, the green circles appear to be aligned to the road network.

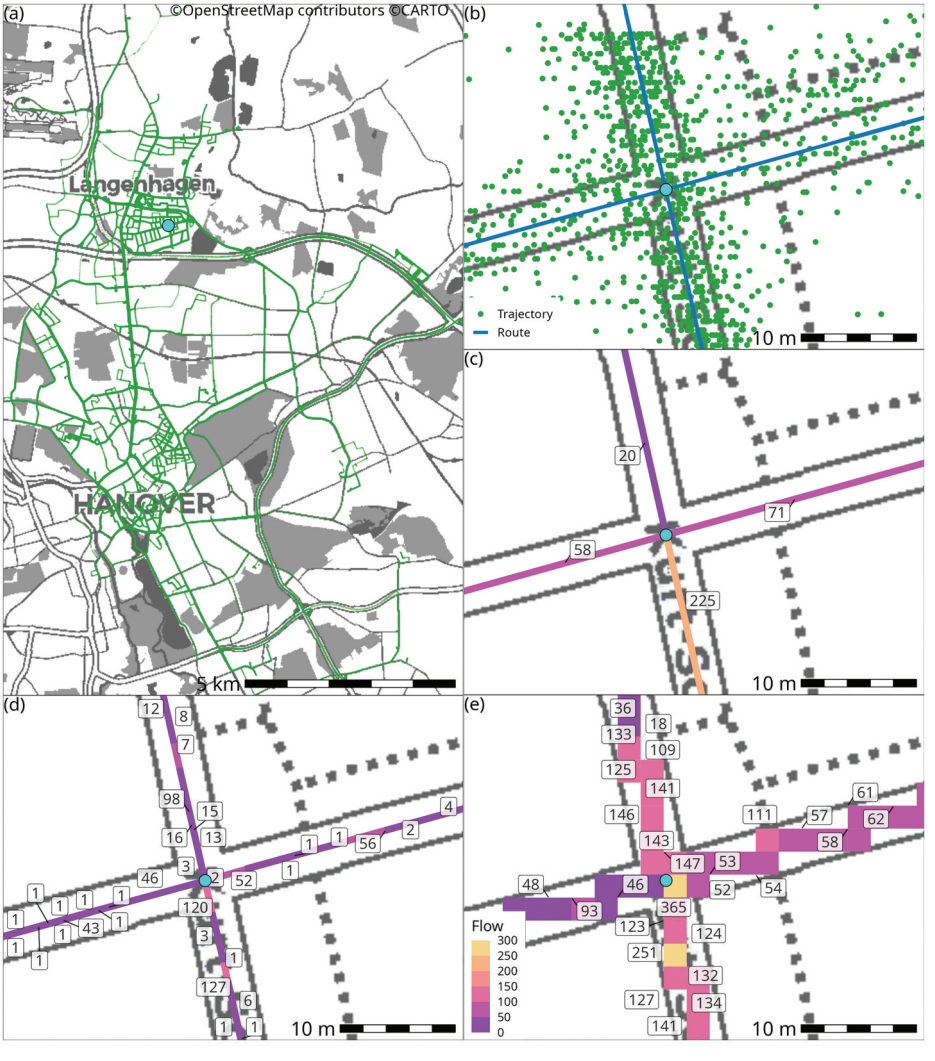

Figure 1. Trajectory data in Hannover, Germany. (a) Empirical trajectories (green circles), Loebensteinstraße (solid cyan circle). (b) Zoom of trajectories and map matched routes (blue lines). (c) Zoom of ideal flow aggregation. (d) Zoom of existing flow aggregation. (e) Zoom of 2 m × 2 m rasterised flow aggregation. Colour scale/label is traffic flow in road segments/grid cells (author's work).

If we zoom in on these trajectory points around Loebensteinstraße (solid cyan circle), formerly known as Hindenburgstraße, is a residential street near the A2 autobahn, then we observe that in Figure 1(b) the points deviate from the road network. These noisy trajectories are unsuitable for direct aggregation into a flow map, so the usual first step is to apply a map matching procedure (Chao et al. 2020; Quddus, Ochieng, and Noland 2007). The map matched routes are the blue lines in Figure 1(b). These follow the road network more closely. Map matching is a highly difficult problem and map matching with exact alignment to the underlying road network in a reasonable time frame is usually not feasible. The approach for fast computations taken by the map matching introduces some approximations which do not lead to exact alignment of empirical trajectories with the road network or to each other (Sghaier 2019).

The ideal flow aggregation from the empirical trajectories is in Figure 1(c). The colour scale for the flow ranges from 0 (violet) to 150 (pink) to 300 (yellow). Applying the state-of-the-art traffic flow aggregation API (Morgan and Lovelace 2021) to the inexactly aligned trajectories leads to an inaccurate flow map in Figure 1(d). The authors resolve this by rasterising the flow map to a raster grid: within each raster grid cell, the flow is the sum of the flows of all trajectories which intersect the grid cell. This rasterised flow map is displayed in Figure 1(e). The 2 m $\times$ 2 m raster grid obscures the road network at this scale. The resolution of a rasterised aggregation flow map is fixed by the resolution of the raster grid. If we require a higher resolution, then we must recalculate the flow map on a raster grid with a higher resolution.

Our goal is to produce a traffic flow map from empirical trajectories which is aligned to the road network and which can be utilised at any scale without recalculation, from the city level to the individual road level. In our proposed local alignment approach, we infer which road segments should serve as a local reference, and then proceed to align other nearby road segments to it. To accomplish this, we introduce a novel algorithm 'line blending' where road segments, which are near each other but do not share overlapping sub-segments, are aligned to maximise their overlapping sub-segments. Line blending is a 'fuzzy' aggregation method, which calculates accurate flow aggregation of noisy trajectories. This is in contrast to the current state-of-the-art flow aggregation methods which are accurate only for noise-free trajectories. We combine these novel algorithms into an iterative workflow which outputs a locally aligned flow map. This flow map is composed of road segments rather than raster grid cells, so its resolution is not limited by the raster grid resolution.

The outline of the paper is as follows. In Section 2, we construct an iterative workflow which computes a locally aligned flow map from noisy trajectories. In Section 3, we apply this workflow to synthetic trajectories from Los Angeles, USA and empirical trajectories from Hannover, Germany. We verify that, with a suitable tuning parameter choice, the locally aligned flow maps have high resolution and spatial accuracy. We then discuss some computational issues and end with some concluding remarks.

2. Local alignment of route segments for flow aggregation

We represent the road network by a graph $\mathcal{N} = \mathcal{N}(\mathcal{E}, \mathcal{V})$, where the $n_{\mathcal{E}}$ edges $\mathcal{E} = \{e_1, \ldots, e_{n_{\mathcal{E}}}\}$ are the road segments and the $n_{\mathcal{V}}$ nodes/vertices $\mathcal{V} = \{v_1, \ldots, v_{n_{\mathcal{V}}}\}$ indicate that different road segments are accessible to/from each other at this node

point. We set $\mathcal{N}$ to be the OpenStreetMap (OSM) road network. Each path in the network is composed of a connected sequence of edges, and we refer to this as a 'linestring' geometry. We denote a single trajectory $G = \{\boldsymbol{g}_1, \ldots, \boldsymbol{g}_{n_G}\}$ as a temporally ordered sequence of n_G points $g_i, i = 1, \ldots, n_G$. This is a 'multipoint' geometry. The points of a trajectory G are not necessarily coincident with the road network $\mathcal{N}$.

2.1. Map matching

We represent the output of a map matching algorithm M from a trajectory G as $M(G) = \{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n_M}\}$ which is a connected sequence of n_M edges, i.e. a linestring geometry. In comparison to a trajectory G, all boundary points of the edges of a map matched route $M(G)$ are aligned to the road network. There is a large body of research on this difficult problem of map matching. We focus on the popular class of Hidden Markov Models (HMM) map-matching algorithms. HMM methods iteratively build the map-matched route by selecting the most likely next segment to connect to the current route using a probabilistic model. According to a review of map-matching algorithms (Chao et al. 2020), HMM is a state-transition method. The other three classes are similarity, candidate-evolving, and scoring methods. Further details of these alternatives are found in Quddus, Ochieng, and Noland (2007), Chao et al. (2020). We leave this discussion here since our proposed methods are valid for any map-matching algorithm, and concentrate on HMM map matching due to its accuracy and computational efficiency. Even if we restrict ourselves to HMM algorithms on the OSM road network, there are many off-the-shelf map-matching APIs available. We focus on the Valhalla routing engine (https://valhalla.github.io/valhalla), which includes its highly recommended map matching API (Saki and Hagen 2022). The inputs to Algorithm 1 (ST_ROUTE) are the trajectory G and the map-matching API M. For readability, we defer the detailed description of the algorithm (and all subsequent algorithms) to Appendix A.

2.2. Flow aggregation

To the road network graph $\mathcal{N} = \mathcal{N}(\mathcal{E}, \mathcal{V})$, we add the traffic flows on each of the network edges to form a flow map $\mathcal{F} = \{(f, \boldsymbol{f}) : \boldsymbol{f} \in \mathcal{N}, f > 0\} = \{(f_1, \boldsymbol{f}_1), \ldots, (f_{n_\mathcal{F}}, \boldsymbol{f}_{n_\mathcal{F}})\}$ composed of $n_\mathcal{F}$ flows, where each $\boldsymbol{f}_i$ is composed of edges in $\mathcal{E}$, with traffic flow f_i, $i = 1, \ldots, n_\mathcal{F}$. Our goal is to estimate the traffic flows $f_1, \ldots, f_{n_\mathcal{F}}$ and the flow network graph $\mathcal{F}$. For exactly aligned trajectories, to compute the flow aggregation, we deploy the overline function in the R package stplanr, which we refer to as ST_OVERLINE_PLANR (Lovelace and Ellison 2018). That is, starting with the map-matched routes $\mathcal{M} = \{M(G_1), \ldots, M(G_n)\}$, the flow map is $\mathcal{F} = \text{ST\_OVERLINE\_PLANR}(\mathcal{M})$.

To compute an accurate flow map from noisy trajectories relies on resolving the crucial problem of how to aggregate similar, but not exactly overlapping, road segments. Historically, most solutions have focused on eliminating the discrepancies in the map matching so that flow aggregation then becomes a straightforward procedure, drawing on the many map matching algorithms available. Despite this research effort, map matching algorithms which yield error-free output are limited to special cases, and none have the same availability, computational efficiency and geographical coverage as

the open source, global Valhalla map matching. Following the recommended best practice, we carry out some preprocessing to simplify the map-matched routes to improve the flow aggregation. We highlight two existing methods, which we call node snapping and node splitting.

Node snapping is where the boundary points of the traffic flow linestrings are snapped to each other. Since the former are also nodes of the flow map, this gives the name to the algorithm. We focus on snapping these nodes since the linestring misalignments are in part caused by the existence of nodes which are close to each other but not exactly equal. Since we are searching for points which are close to together, then this is well-suited to statistical clustering. There are many statistical clustering algorithms available, and we focus on hierarchical clustering. To resolve the computational bottleneck of hierarchical clustering, we approximate the 1-pass complete linkage clustering by a nested 2-pass clustering. We begin with an efficient single linkage clustering in the R package fastcluster (Müllner 2013). Since single linkage can result in chain-like clusters, we compute a subsequent complete linkage clustering to break these potential chains. In this nested approach, the complete linkage distance matrix is calculated only within each single linkage cluster, and so it is less likely to reach computational limits. This approximation is required since fastcluster does not implement an efficient complete linkage clustering. The inputs to Algorithm 2 (ST_SNAPNODE) are the flow linestrings $\mathcal{F}$ and the snap tolerance ε_S. The output are the node snapped linestrings $\mathcal{F}^*$. As its name suggests, all node points of $\mathcal{F}$ within ε_S distance of each other are snapped together. In comparison to the unsnapped linestrings $\mathcal{F}$, there are fewer snapped linestrings $\mathcal{F}^*$ which have higher flow values, are more interconnected, and share more exactly overlapping sub-segments.

Node snapping is focused near the boundary points of the flow linestrings, so it does not address misalignments far from the boundary points. Moreover, there remain some linestrings whose intersection is not correctly computed by ST_OVERLINE_PLANR. The solution to both of these problems is the explicit addition of the missing intersection nodes to these linestring, and then split these linestrings into simple linestring segments at these added nodes. This is known as node splitting. The inputs to Algorithm 3 (ST_SPLITNODE) are the flow linestrings $\mathcal{F}$ and the node splitting type S. Our method deploys for $S =$ 'subdivision' to spatial subdivision in the sfnetworks package (van der Meer et al. 2021), and for $S =$ 'unary' geos unary union in the geos package (Dunnington and Pebesma 2020). The first option tends to find fewer intersection nodes and is similar to the intersection computation performed by ST_OVERLINE_PLANR. The second option finds more missing intersection nodes, and leads to more comprehensive flow aggregation.

In Figure 2(a) is the flow aggregation of the map-matched routes without any node splitting or node snapping. There are missing intersection nodes, and there are intersection nodes (solid black circles), which are close to each other. In Figure 2(b) is the flow aggregation after node splitting ($S =$ 'unary') and then node snapping (with snap tolerance $\varepsilon_S = 2$ m). The result is that there are fewer flow segments with higher flow values. Due to the combined action of ST_SPLITNODE and ST_SNAPNODE, Figure 2(b) is closer to the ideal flow map topology than Figure 2(a).

Whilst the preprocessing does improve the flow aggregation, there remain some discrepancies, since Figure 2(b) is not the ideal flow map. To tackle directly to the problem of eliminating the discrepancies during flow aggregation, heuristic solutions have most

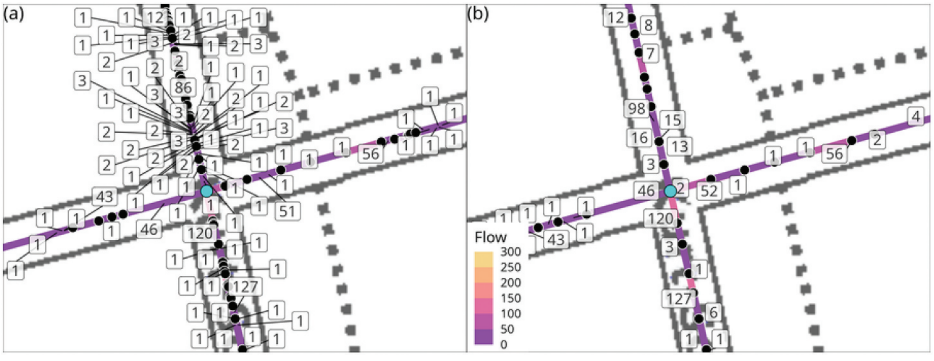

Figure 2. Unsplit/Unsnapped and node split/node snapped routes. (a) Unsplit and unsnapped routes, with flow aggregation. (b) Node split and node snapped, with snap tolerance $\varepsilon_S = 2$ m, and with flow aggregation. Colour scale/label is traffic flow in road segments (author's work).

likely been investigated, though few have been published so there is scarce literature on this problem. Among the few published solutions include edge bundling (Zhou et al. 2013) and rasterisation (Morgan and Lovelace 2021; Wood, Dykes, and Slingsby 2010). Edge bundling consists of clustering trajectory linestrings and replacing all cluster members with a single representative linestring. These (and subsequent) authors conclude that it performs poorly when applied to noisy trajectories at the road level, and remains mostly suited to large-scale aggregations, such as desire line maps (Tobler 1987). We do not consider edge bundling further since it is not suitable for our purposes. Rasterisation relies on converting the flow map into a raster matrix, and aggregating the flows within the same raster grid cell. Whilst rasterisation is able to improve flow aggregation, it depends highly on the raster grid resolution, and leads to a loss of spatial resolution.

We propose an alternative aggregation which solves both of these problems. We achieve this goal by local internal alignment between the map-matched routes. By internal alignment, we mean that we align the routes with each other, rather than to the external road network graph. By local alignment, we mean that we align segments of the routes, rather than the complete routes. Our proposal is developed within the R statistical analysis environment due to its integrated access to advanced statistical and geospatial analysis methods.

2.3. Line blending to align similar linestrings to local reference linestring

So far we have focused on improving the alignment of linestrings by resolving inconsistencies at their intersections. We now focus on aligning linestrings more generally. For this, we require a comparison relationship to establish an order of alignment of nearby linestrings. For two flow linestrings $(f_1, \boldsymbol{f}_1)$ and $(f_2, \boldsymbol{f}_2)$ from a flow map $\mathcal{F}$, we define that $\boldsymbol{f}_2$ is a candidate to be aligned to the reference $\boldsymbol{f}_1$ at threshold $\varepsilon \geq 0$ if

$$f_2 \subseteq \text{ST\_BUFFER}(f_1, \varepsilon), f_2 \leq f_1. \tag{1}$$

The condition on the flow values $f_2 \leq f_1$ means that we place a higher priority on the reference linestring $\boldsymbol{f}_1$ since its flow f_1 is larger than the flow f_2 on the candidate linestring $\boldsymbol{f}_2$. The buffer zone has flat edges, e.g. ST_BUFFER(endCapStyle='FLAT') in the R package sf

(Pebesma 2018), so the buffer zone ends at the boundary points of $\boldsymbol{f}_1$. This avoids erroneously considering neighbouring segments of $\boldsymbol{f}_1$, which are connected to $\boldsymbol{f}_1$ as a part of a longer sequence, to be candidates to be aligned to $\boldsymbol{f}_1$.

Let $\boldsymbol{f}_{ref}$ be a reference linestring from $\mathcal{F}$. The set of $n_{\mathcal{F}_{cand}}$ linestrings from $\mathcal{F}\backslash\{\boldsymbol{f}_{ref}\}$ which satisfy Equation (1) are the candidate linestrings $\mathcal{F}_{cand} = \{\boldsymbol{f}_{cand,1}, \ldots, \boldsymbol{f}_{cand,n_{\mathcal{F}_{cand}}}\}$, with the convention that if $n_{\mathcal{F}_{cand}} = 0$ then $\mathcal{F}_{cand}$ is the empty set. We call our approach 'line blending', since we blend the candidate linestrings onto the reference linestring. This line blending resolves the crucial problem of how to aggregate similar, but not exactly overlapping, road segments. This 'fuzzy' aggregation allows us to compute accurate flow aggregation from noisy trajectories.

The inputs in Algorithm 4 (ST_LINEBLEND) are the reference linestring $\boldsymbol{f}_{ref}$, the candidate linestrings $\mathcal{F}_{cand}$, and the blend tolerance ε. The output are the modified reference linestring $\boldsymbol{f}^*_{ref}$ and projected candidate flow linestrings $\mathcal{F}^*_{cand}$, all with added interior points for accurate calculation of exactly equal linestring segments. If there are many candidate linestrings, then this may lead to many sub-segments in the projected linestrings, each with their own flow values. So we assign the rounded value of the weighted mean flow, weighted by the sub-segment length, to all sub-segments.

In Figure 3(a), we illustrate this line blending with the flow linestrings $(7, AB)$ in blue and $(2, AC)$ in orange. Since $AD \subset \text{ST\_BUFFER}(AB, 2\text{m})$ (pale blue rectangle), and $2 < 7$, then AC is the reference $\boldsymbol{f}_{ref}$ and AD is the candidate linestring $\boldsymbol{f}_{cand}$, according to Equation (1). Line blending involves projecting AD to AC, so C is projected to C' (which lies exactly on the reference linestring). We also add C' to the reference linestring. The reference linestring becomes $\boldsymbol{f}^*_{ref} = (7, AC'B)$, and the candidate $\boldsymbol{f}^*_{cand} = (2, AC')$. Now $\boldsymbol{f}^*_{ref}, \boldsymbol{f}^*_{cand}$ share the sub-segment AC' exactly, and ST_OVERLINE_PLANR gives the aggregated flows as $(9, AC')$ and $(7, C'B)$. We take the rounded value of the weighted mean of these two flow linestrings. The result is a single linestring $(9, AC'B)$, as shown in Figure 3(b). Line blending is required to aggregate correctly the linestrings $(7, AC)$ and $(2, AD)$ since they do not share a common sub-segment.

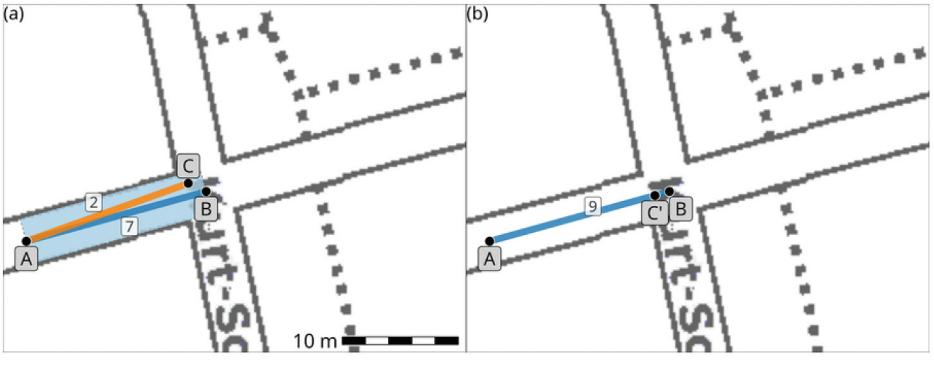

Figure 3. Line blending leads to correct flow aggregation. (a) Before line blending and flow aggregation. Reference linestring $(7, ABC)$ in blue, candidate linestring $(2, AD)$ in orange. (b) After blending candidate into reference linestring, with blend tolerance $\varepsilon = 2$ m, and flow aggregation. Blended linestring is $(9, AC'B)$ (author's work).

More complex situations arise when other linestrings touch the candidate linestring, but are not candidates themselves for blending. Since line blending projects the candidates to the reference linestring, then we have to also project these other linestrings to avoid leaving a gap in the updated flow map. The inputs to Algorithm 5 (ST_SNAP_CAND_TOUCH) are the reference linestring $\boldsymbol{f}_{ref}$, the candidate-touching linestrings $\mathcal{F}_{ct}$, and the snap tolerance ε_S. The output are the snapped candidate-touching flow linestrings $\mathcal{F}^*_{ct}$, which maintain connectivity between $\mathcal{F}^*_{ct}$ and $\boldsymbol{f}_{ref}$.

Figure 4(a) is an illustration of ST_SNAP_CAND_TOUCH with the reference $(7, AB)$ (blue), the candidate linestring $(2, AD)$ (orange), and the blending buffer zone with blend tolerance $\varepsilon = 4$ m (pale blue rectangle). The candidate-touching linestring $\boldsymbol{f}_{ct} = (5, CD)$ (purple) touches the candidate AC at C. The orange line is entirely within the pale blue buffer zone, whereas the purple line extends outside of the buffer zone and so is not a candidate for blending to the reference linestring. If we apply ST_LINEBLEND to blend the candidate linestring, then C is projected to C' to lie on the reference linestring, i.e. $\boldsymbol{f}^*_{cand} = (2, AC')$. If we apply ST_SNAP_CAND_TOUCH with snap tolerance $\varepsilon_S = 2$ m to the candidate-touching linestring, then the intersection point C is projected to the boundary point B, i.e. $\boldsymbol{f}^*_{ct} = (5, BD)$. Thus line blending, candidate-touching snapping, and flow aggregation yields that $\boldsymbol{f}^*_{ct} = (5, BD)$ remains connected to $\boldsymbol{f}^*_{ref} = (9, AC'B)$.

To illustrate line blending for the Hannover map-matched routes, Figure 5(a) is the flow segments without any line blending, and Figure 5(b) is the flow segments after line blending. Due to the action of ST_OVERLINE_LINEBLEND, each road segment in the line-blended flow map Figure 5(b) corresponds unambiguously to a road segment in the OSM road network, and thus approaches the ideal flow map. This is an improvement over the multiple road segments in the unblended flow map in Figure 5(a), and the pixelated road segments in the rasterised flow map in Figure 1(e).

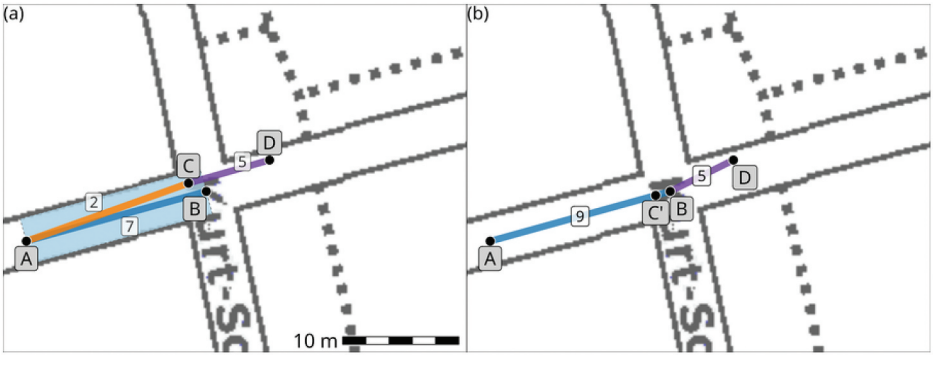

Figure 4. Snap candidate-touching linestrings to reference linestring after line blending. (a) Before line blending and snapping. Reference linestring $(7, AC)$ in blue, candidate linestring $(2, AC)$ orange, and candidate touching linestring $(5, CD)$ purple. (b) After line blending (blend tolerance $\varepsilon = 2$ m) and snapping (snap tolerance $\varepsilon_S = 2$ m). Reference linestring becomes $(9, AC'B)$, and candidate-touching linestring $(5, BD)$ (author's work).

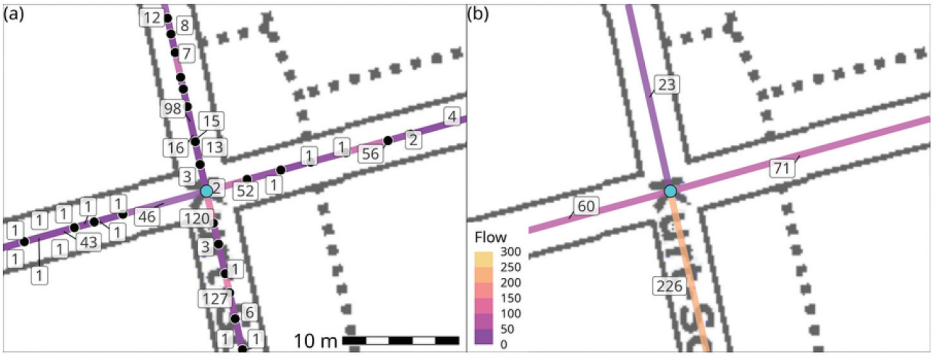

Figure 5. Line blending for Hannover map matched routes. (a) Flow segments without line blending. (b) Flow segments with line blending. Colour scale/label is traffic flow (author's work).

2.4. Workflow

Line blending for accurate flow aggregation at a single intersection in a road network is demonstrated in Figure 5. To extend line blending to cover the entire road network, we require two further generalisations. First, we generalise the reference linestring to be a sequence of k connected edges, each with their own flow. If we treat this edge sequence momentarily as a single linestring, with flow equal to the weighted mean flow, weighted by the length of the k edges, then we can apply Equation (1) to search for potential candidate linestrings. Due to computational limitations, we retain that candidates be single edges with single flow values. The generalisation to k-edge reference linestrings allows us to blend candidate linestrings which exceed the buffer zones of 1-edge reference linestrings.

Second, we determine the priority for line blending for a set of reference linestrings. We require that a reference linestring cannot be a candidate linestring to another reference linestring, and a candidate linestring is a candidate for one reference linestring only. These ensure that if two linestrings $f_2 \subseteq \text{ST\_BUFFER}(f_1, \varepsilon)$ and $f_1 \subseteq \text{ST\_BUFFER}(f_2, \varepsilon)$, then we select only one of them. The flow condition $f_2 \leq f_1$ usually distinguishes the reference linestring, except when $f_1 = f_2$. In this case, then we designate the longer linestring to be the reference.

The inputs to Algorithm 6 (ST_LINEBLEND_PRIORITY) are the flow linestrings $\mathcal{F}$, the number of connected edges for the reference linestrings k, and the blend tolerance ε. The output is a set of non-overlapping reference linestrings $\mathcal{F}_{ref}$, a set of unique candidate linestrings $\mathcal{F}_{cand}$, and a set of candidate-touching $\mathcal{F}_{ct}$ linestrings. By combining Algorithms 2–6, we obtain an algorithm for local alignment of flow segments, in Algorithm 7 (ST_OVERLINE_LINEBLEND). The inputs to ST_OVERLINE_LINEBLEND are the flow linestrings $\mathcal{F}$, the blend tolerance ε, and the snap tolerance ε_S. The output are the $\mathcal{F}^*$ aggregated aligned flow linestrings.

To compute a locally aligned flow map, we combine Algorithms 1–7 into an initial and an iterative phase. The inputs to Algorithm 8 (ST_OVERLINE_INITIAL) are the map-matched routes $\mathcal{M}$ and the split node type S. The output is an initial aligned flow map. With an initial flow map on hand, we proceed iteratively towards a locally aligned flow map. The inputs to this iterative step Algorithm 9 (ST_OVERLINE) are the initial flow map

$\mathcal{F}$, the split node type S, the blend tolerance ε, the snap tolerance ε_S, the number of connected edges $\boldsymbol{k}$, and the maximum number of iterations $j_{\max}$. After a suitable number of iterations, the output is a locally aligned flow map.

Our proposed workflow from an input of trajectory data to a locally aligned flow map is illustrated in Figure 6. The input are the trajectories. We select the blend tolerance tuning parameter ε, and set the maximum number of iterations $j_{\max} = 20$ and the snap tolerance $\varepsilon_S = \varepsilon$. We carry out map matching with ST_ROUTE. For the flow map aggregation, we begin with an iteration of ST_OVERLINE_INIITAL to create an initial flow map. For the subsequent step, we call ST_OVERLINE with #edges $\boldsymbol{k} = 1, 2$, and node split type $S =$ 'subdivision'. At this early stage, 'subdivision' node splitting provides more stable line blending priority. We follow with a call to ST_OVERLINE with 'unary' node splitting. The 'unary' node splitting is usually applied after 'subdivision' node splitting since the former can now add any missing intersection nodes without adversely affecting the line blending priority. We iterate ST_OVERLINE with $\boldsymbol{k} = 1, 2, S =$ 'unary', until convergence, and the flow map $\mathcal{F}$ is the output. In this workflow, the only unspecified tuning parameter is the blend tolerance ε, whose properties we investigate in the next section.

3. Results and discussion

3.1. Synthetic trajectories

Despite the ubiquity of GNSS-enabled devices, there is a lack of gold standard empirical trajectories and flow maps, due to privacy concerns and other reasons. Our solution is to rely on noise-free synthetic trajectories which are exactly aligned to a theoretical road network (Anastasiou, Kim, and Shahabi 2022a). These authors generate synthetic trajectories from an origin-destination matrix, and make available a set of 1.5 million synthetic trajectories for the metropolitan area of Los Angeles, USA (Anastasiou, Kim, and Shahabi 2022b). For illustrative purposes, we focus on a sample of $n = 1000$ trajectories, in EPSG 6339 (NAD83 (2011)/UTM zone 10N) coordinates, in Culver City (located about 16 km from downtown Los Angeles), which pass near the intersection of Main Street and Venice Boulevard, as indicated by the solid cyan line in Figure 7. We add a random uniform noise of ± 5 m to the noise-free trajectories to obtain the noisy trajectories, displayed as the green lines in Figure 7(a–b). This noise mimics the typical measurement error of 5 m observed in empirical trajectories from GNSS-enabled smartphones (Van Diggelen and Enge 2015). The reference noise-free flow map is in Figure 7(c), and the rasterised flow map in Figure 7(d). Morgan and Lovelace (2021) do not recommend a value for tuning the

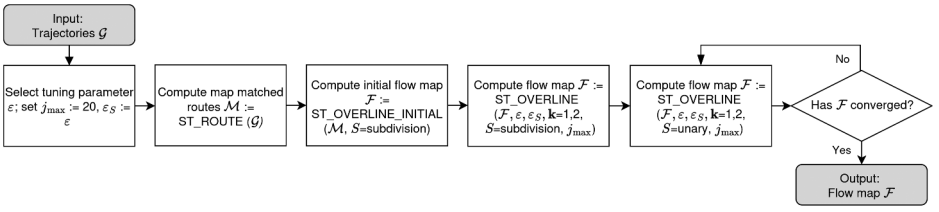

Figure 6. Workflow for locally aligned flow map. Input are trajectories $\mathcal{G}$, output is locally aligned flow map $\mathcal{F}$. Blend tolerance ε is tuning parameter (author's work).

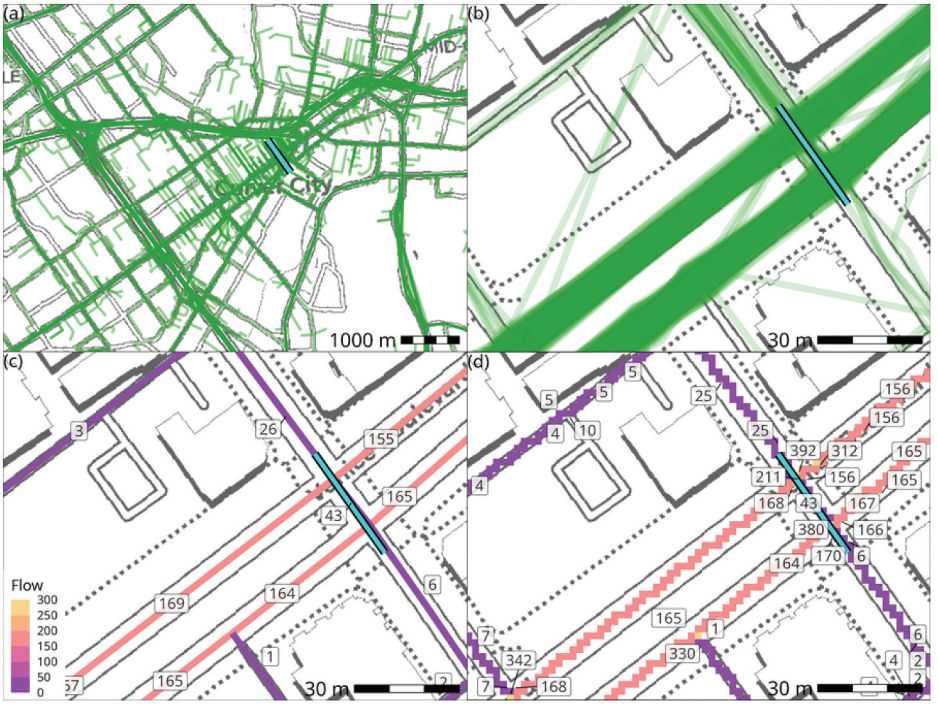

Figure 7. Synthetic Culver City trajectories ($n = 1000$). (a) Noisy trajectories (green lines). (b) Zoom of noisy trajectories near intersection of Main st and Venice bd (solid cyan line). (c) Reference noise-free flow map. (d) Rasterised flow map with $\varepsilon_R = 2$ m. Colour scale/label is traffic flow (author's work).

rasterisation grid resolution, so we select $\varepsilon_R = 2$ m subjectively so that the raster grid cells do not exceed the road widths.

For our locally aligned flow maps, the free tuning parameter is the blend tolerance ε. So we investigate the effect of ε on the accuracy of flow aggregation. For the blend tolerance $\varepsilon = 0$ m, this means that no line blending is performed, so the unblended flow map in Figure 8(a) comprises many small road segments with low flow values. For the blend tolerance $\varepsilon = 1, 5$ m, the flow maps in Figure 8(b–c) are similar and appear to be accurate flow aggregations. For the blend tolerance $\varepsilon = 10$ m, the purple flow segment (with flow 43), near the intersection of Venice Bd and Main St (solid cyan line), is poorly aligned to the road network, unlike the more accurate alignment in Figure 8(b–c).

We observe the usual sensitivity trade-off in the evolution of the values of the blend tolerance parameter. If this value is zero like in Figure 8(a), then line blending is not carried out. The input trajectories are insufficiently processed, so the output flow map contains too many road segments. Whilst each discrepancy to the true road network is small (low bias), whenever the input trajectories are perturbed slightly, this would lead to a different flow map (high variability). If the blend tolerance is large (e.g. 10 m in Figure 8(d)), the action of line blending is extended to a large radius. When the blend tolerance is larger than the distance between two distinct neighbouring road segments, then input trajectories which belong to different segments on the true road network can be erroneously

Figure 8. Effect of blend tolerance ε on synthetic Culver City flow maps. (a) Unblended flow map with $\varepsilon = 0$ m. (b) Flow map with $\varepsilon = 1$ m. (c) Flow map with $\varepsilon = 5$ m. (d) Flow map with $\varepsilon = 10$ m. Colour scale/label is traffic flow (author's work).

blended together, and the output flow map contains an estimated road segment with a large discrepancy from any true road segment. Whilst the discrepancy is large (high bias), whenever the input trajectories are perturbed slightly, this would lead to a similar flow map (low variability). If the blend tolerance is between these two extremes (e.g. 1 or 5 m in Figure 8(b–c)), then we achieve a trade-off between bias and variability.

To complement the above heuristic comparisons, we define a quantitative discrepancy between a flow map $\mathcal{F}$ and the reference flow map $\mathcal{F}_{ref}$. Whilst there are some promising discrepancy measures, such as graph edit distance (Zeng et al. 2009), none are able to satisfy both theoretical closeness and computational efficiency for general networks. So we provide ad hoc own discrepancy measures for our purposes. For each linestring $\boldsymbol{f}_i$ in $\mathcal{F}$, we begin by interpolating the point $\boldsymbol{f}_{i,\tau}$ at $\tau \text{len}(f_i)$ length along $\boldsymbol{f}_i$. That is, $\tau = 0.1$ yields an interpolated point $\boldsymbol{f}_{i,0.1}$ which is 10% of the length of f_i from the start point. We define the flow discrepancy to be $\text{Err}_{\text{Flow},i,\tau} = \left| f_{i,\tau} - f^*_{ref,i,\tau} \right|$ where $\boldsymbol{f}^*_{ref,i,\tau}$ is the closest reference linestring in $\mathcal{F}_{ref}$ to the interpolated point $\boldsymbol{f}_{i,\tau}$. We reverse the order of the comparison to obtain the errors $\text{Err}_{\text{Flow},j,\tau} = \left| f_{ref,j,\tau} - f^*_{j,\tau} \right|$, and concatenate these errors. In Figure 9, we display the flow errors for $\tau = 0.1, 0.9$. Figure 9(a) is the unblended flow map, Figures 9(b–d) are the locally aligned flow maps with blend tolerance $\varepsilon = 1, 5, 10$ m, and Figure 9(e) is the rasterised flow map. The unblended

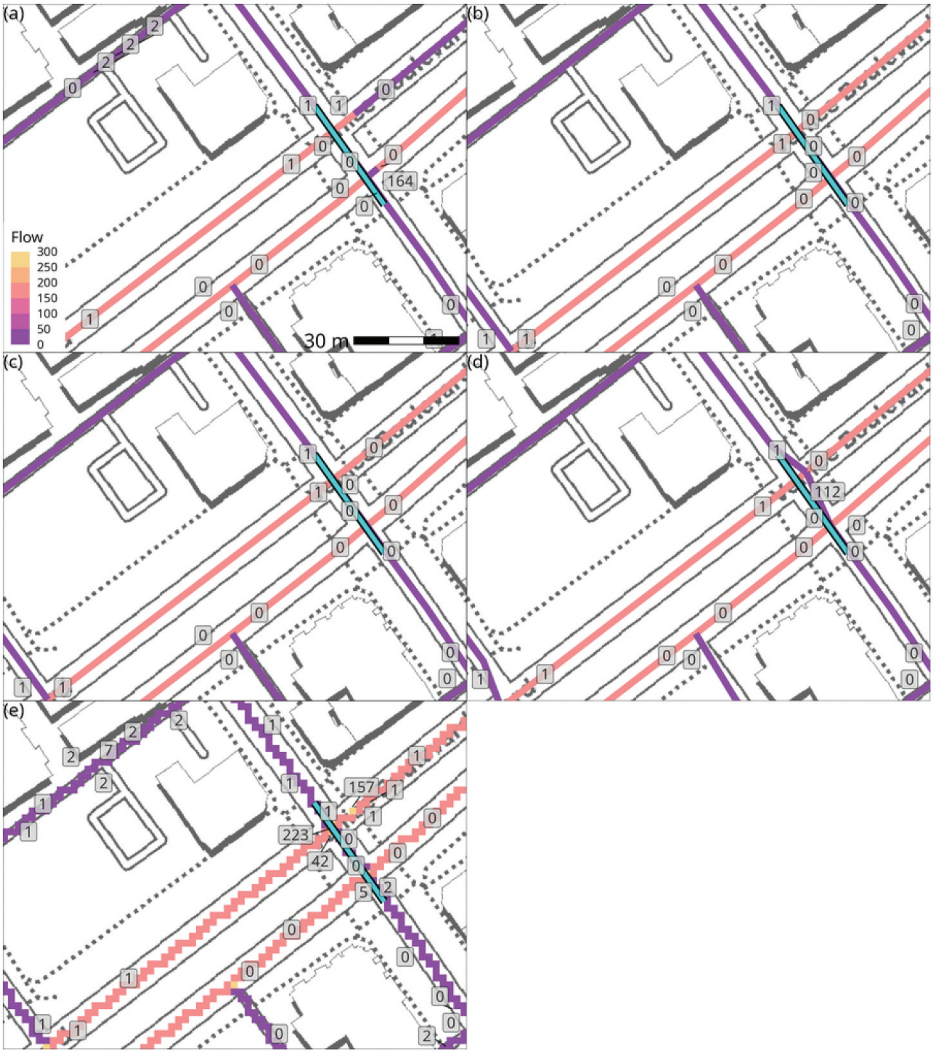

Figure 9. Flow errors for $\tau = 0.1, 0.9$ for synthetic Culver City flow maps ($n = 1000$). (a) Unblended flow map with $\varepsilon = 0$ m. (b) Flow map with $\varepsilon = 1$ m. (c) Flow map with $\varepsilon = 5$ m. (d) Flow map with $\varepsilon = 10$ m. (e) Rasterised flow map with $\varepsilon_R = 2$ m. Colour scale is traffic flow, label is traffic flow error (author's work).

flow map, the line-blended with $\varepsilon = 10$ m and the rasterised flow map lead to some high flow error values; whereas the flow errors remain low for the line-blended flow maps with $\varepsilon = 1, 5$ m.

In addition to this flow discrepancy, we define a topological discrepancy $\text{Err}_{\text{Node},i,\tau} = \left|\deg\left(v\left(f_{i,\tau}\right)\right) - \deg\left(v\left(f^*_{\text{ref},i,\tau}\right)\right)\right|$ where $\boldsymbol{v}(\boldsymbol{f}_{i,\tau})$ is the closest node in $\mathcal{F}$ to $\boldsymbol{f}_{i,\tau}$ and $\boldsymbol{v}(\boldsymbol{f}^*_{\text{ref},i,\tau})$ is the closest node in $\mathcal{F}_{\text{ref}}$ to $\boldsymbol{f}_{i,\tau}$; and $°(\cdot)$ is the degree of the node (i.e. the number of incident edges). We reverse the order of the comparison to obtain the errors $\text{Err}_{\text{Node},j,\tau} = \left|\deg\left(v\left(f_{\text{ref},j,\tau}\right)\right) - \deg\left(v\left(f^*_{j,\tau}\right)\right)\right|$, and concatenate these errors. The flow/

Table 1. Error discrepancies for flow maps, for $N = 10$ re-samples, sample sizes $n = 100, 1000, 10000$ and noise levels $\sigma = 5, 10$ m. Upper half is flow errors Err_{Flow}, lower half is topological errors Err_{Node}. Each value is 10% trimmed mean error $\pm$ sd, lowest mean values for each row are in bold. First to fourth columns are line blended flow maps $\varepsilon = 0, 1, 5, 10$ m, fifth column is rasterised flow map with $\varepsilon_R = 2$ m.

		Line blended flow maps				Rasterised
Sample size	Noise	$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 5$	$\varepsilon = 10$	$\varepsilon_R = 2$
Flow errors						
$n = 100$	$\sigma = 5$	0.14 ± 0.48	0.09 ± 0.42	$\mathbf{0.09 \pm 0.44}$	0.13 ± 0.55	0.36 ± 0.96
	$\sigma = 10$	0.15 ± 0.47	$\mathbf{0.09 \pm 0.41}$	0.09 ± 0.41	0.13 ± 0.54	0.37 ± 0.96
$n = 1000$	$\sigma = 5$	0.30 ± 0.69	$\mathbf{0.24 \pm 0.68}$	0.24 ± 0.73	0.33 ± 0.98	0.66 ± 1.69
	$\sigma = 10$	0.35 ± 0.79	$\mathbf{0.27 \pm 0.74}$	0.29 ± 0.79	0.37 ± 1.04	0.72 ± 1.77
$n = 10000$	$\sigma = 5$	0.92 ± 1.80	0.85 ± 1.88	$\mathbf{0.81 \pm 1.84}$	0.96 ± 2.23	1.11 ± 2.49
	$\sigma = 10$	1.29 ± 2.66	0.94 ± 1.93	$\mathbf{0.93 \pm 1.95}$	1.06 ± 2.38	1.27 ± 2.64
Topological errors						
$n = 100$	$\sigma = 5$	0.41 ± 0.85	$\mathbf{0.01 \pm 0.23}$	0.04 ± 0.35	0.11 ± 0.45	1.37 ± 0.64
	$\sigma = 10$	0.41 ± 0.85	$\mathbf{0.01 \pm 0.20}$	0.05 ± 0.35	0.11 ± 0.45	1.36 ± 0.64
$n = 1000$	$\sigma = 5$	0.44 ± 0.96	$\mathbf{0.04 \pm 0.34}$	0.06 ± 0.36	0.10 ± 0.42	1.27 ± 0.61
	$\sigma = 10$	0.48 ± 0.97	$\mathbf{0.06 \pm 0.35}$	0.08 ± 0.38	0.12 ± 0.43	1.26 ± 0.60
$n = 10000$	$\sigma = 5$	0.83 ± 1.38	0.10 ± 0.38	$\mathbf{0.10 \pm 0.39}$	0.15 ± 0.45	1.21 ± 0.58
	$\sigma = 10$	0.94 ± 1.40	$\mathbf{0.12 \pm 0.39}$	0.13 ± 0.41	0.16 ± 0.46	1.20 ± 0.57

topological discrepancy is close to zero whenever the flow map $\mathcal{F}$ has flow values/node connectivities, which resemble those in the reference flow map $\mathcal{F}_{ref}$.

We draw $N = 10$ re-samples of $n = 100, 1000, 10000$ noise-free Culver City trajectories. To these trajectories, we add random uniform noise $\sigma = 5, 10$ m, and carry out map matching. From these map-matched routes, we compute line-blended flows maps with $\varepsilon = 0, 1, 5, 10$ m and rasterised flow maps with $\varepsilon_R = 2$ m. We calculate the flow and topological errors with $\tau = 0.1, 0.9$ for these flow maps with respect the noise-free flow map, as shown in Table 1. The flow errors are in the upper half and the topological errors in the lower half. Because these error values have a highly right-skewed distribution, each value in the table is the 10% trimmed mean error $\pm$ standard deviation, and the lowest values for each row are in bold. Due to the robustness of the map matching, the overall performance between the noise levels $\sigma = 5, 10$ m are similar so we focus on $\sigma = 5$ m. All rasterised maps, since they are limited by their rasterisation grid resolution, perform poorly in comparison to the line-blended flow maps, for both flow and topological errors. Among the line blended flow maps, $\varepsilon = 0, 10$ m have uniformly larger errors than $\varepsilon = 1, 5$ m. The optimal line blended flow maps with $\varepsilon = 1, 5$ m behave similarly for all sample sizes. So we tentatively recommend a value between 1 and 5 m for the blend tolerance parameter ε since it leads to optimal line blending in terms of the flow and topological discrepancies.

3.2. Empirical trajectories

We return to the empirical Hannover trajectories. From the 1183 trajectories, we keep 1180 trajectories with length greater than 0 m, and follow the workflow in Figure 6. After some trial and error, we set the blend tolerance $\varepsilon = 2$ m, which is consistent with the tentative recommendation of ε between 1 and 5 m from the synthetic trajectories

computations above. The workflow leads to a locally aligned flow map displayed in Figure 10(a). The corresponding unblended flow map is Figure 10(b) and the rasterised flow map is Figure 10(c). At the city level, all these flow maps visualise well the traffic flows.

If we zoom into Loebensteinstraße (solid cyan circle), then we observe that in Figure 11(a) the locally aligned map displays the aggregated flows on single linear road segments in comparison to the multiple road segments of the unblended flow map in Figure 11(b), and the pixelated road segments of the rasterised flow map in Figure 11(c). Whilst we do not have a reference flow map for these empirical trajectories, like we had for the synthetic Los Angeles trajectories, we are still able to compute proxy flows based on empirical counts of the map-matched routes with orthogonal line transects from the flow map road segments. Along Loebensteinstraße, the locally aligned flow map is composed to two road segments (in pink) with 0 and 2 flow errors corresponding to the proxy flows 71 and 58. For the unblended and rasterised maps, the flow errors vary from 0 to over 200. Without any line blending, near the cyan circle, there is an unblended line segment with estimated flow 3, whereas the proxy flow is 225, so the flow error is 222. Rasterisation some times leads to double counting, e.g. in the orange region near the cyan circle, the estimated flow is 365 whereas the proxy flow is 225, so the flow error is 140. The locally aligned map does not suffer from an insufficiently line-blended topology or from rasterisation effects, so its flow aggregation is more accurate.

For our purposes, the synthetic trajectories in Los Angeles, and the empirical trajectories collected from a single driver in Hannover, can be considered to represent the complete driver population. So the resulting flow maps describe the complete traffic flows. This is not the case for the usual crowd-sourced trajectory data which are drawn from a biased sample of the complete driver population, and so the resulting flow map is a biased estimate of the complete traffic flow. We leave it to future work to apply the necessary adjustments to compute an unbiased estimate of the complete traffic flow map (Iqbal et al. 2014; Miller, Vander Laan, and Marković 2020).

3.3. Computation

We report the execution times for the flow map computations for the synthetic and empirical trajectories. Since it is difficult to reliably reproduce execution times on different systems, we provide our execution times in Table 2 as a guide only. These are obtained on a stand-alone set-up running Ubuntu 24.04 with 16Gb RAM and 4 Intel i5 CPU @3.10 GHz. We deploy a local dockerised Valhalla routing engine API (https://github.com/gis-ops/docker-valhalla.) (GIS OPS 2023) rather than relying on a web-based service. The upper rows in Table 2 are for the synthetic Culver City trajectories, and the lower rows for the empirical Hannover trajectories. Each value is the mean execution time in seconds.

For the synthetic Culver City trajectories, in the first column of Table 2, the map matching carried out in ST_ROUTE executes in $\sim$14 s for $n = 100$, $\sim$90 s for $n = 1000$, and $\sim$850 s for $n = 10000$ which is approximately a 10-fold increase in execution time for each 10-fold increase in the number of trajectories n. So it is straightforward to infer a precise execution time of ST_ROUTE for large sample sizes from a smaller sample size. In the second column, the unblended flow map relies only on the fast flow aggregation of ST_OVERLINE_PLANR (Lovelace and Ellison 2018). The line-blended flow maps $\varepsilon = 1, 5, 10$

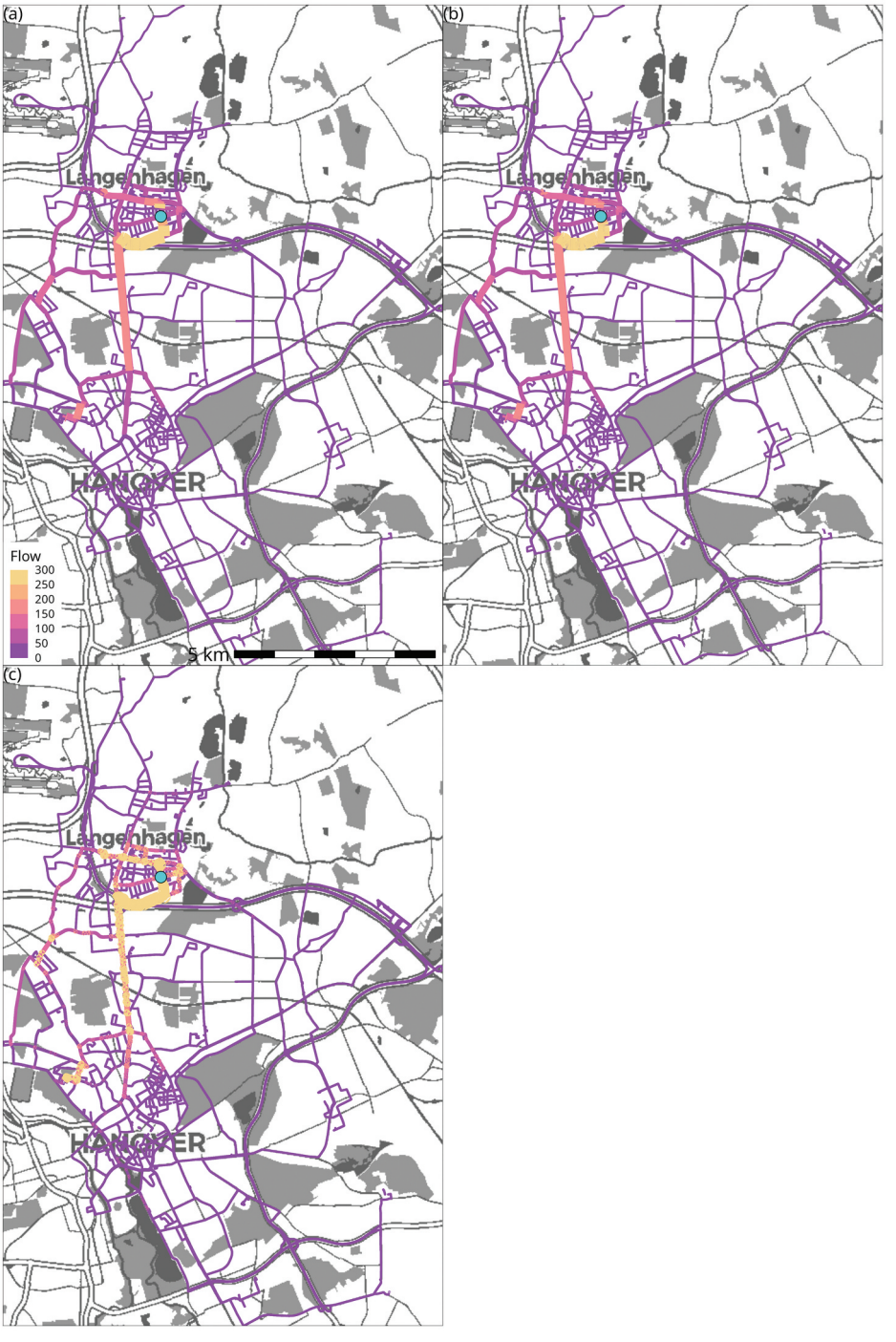

Figure 10. Hannover flow maps at city level. (a) Locally aligned flow map. (b) Unblended flow map. (c) Rasterised flow map. Solid cyan circle is Loebensteinstraße. Colour scale is traffic flow (author's work).

Figure 11. Hannover flow maps at road segment level. (a) Locally aligned flow map. (b) Unblended flow map. (c) Rasterised flow map. Solid cyan circle is Loebensteinstraße. Colour scale is traffic flow, label is traffic flow error (author's work).

Table 2. Execution times for map matching and flow map computations, for $N = 10$ re-samples, sample sizes $n = 100, 1000, 10000$ and noise levels $\sigma = 5, 10$ m. Upper table is for Culver City trajectories, lower table is for Hannover trajectories. Each value is mean execution time (seconds). First column is map matching, second to fifth columns are line blended flow maps, sixth column is rasterised flow map.

			Line blended flow maps				Rasterised
Sample size	Noise	Map match	$\varepsilon = 0$	$\varepsilon = 1$	$\varepsilon = 5$	$\varepsilon = 10$	$\varepsilon_R = 2$
Culver City							
$n = 100$	$\sigma = 5$	13.56	0.55	43.52	52.67	65.12	139.34
	$\sigma = 10$	14.12	0.56	44.40	53.50	64.71	138.52
$n = 1000$	$\sigma = 5$	88.88	1.92	237.08	194.75	246.64	294.90
	$\sigma = 10$	93.97	1.91	254.33	257.15	256.10	298.06
$n = 10000$	$\sigma = 5$	819.76	8.90	1900.01	986.88	855.36	606.88
	$\sigma = 10$	847.76	9.16	2244.08	1574.49	1101.33	608.63

Hannover				
Sample size	Map match	Line blended flow maps		Rasterised
		$\varepsilon = 0$	$\varepsilon = 2$	$\varepsilon_R = 2$
n = 1180	105.03	2.19	396.73	378.65

m require an iterative procedure which is around 100 times longer than the initial $\varepsilon = 0$ m flow map computation. As the number of the trajectories n increases, the execution time increases at about 5-fold, which is slower than the 10-fold increase for the map matching. For a 10-fold increase in the number of trajectories, many of them will overlap

each other so flow aggregation does not require to a 10-fold increase in execution time. For the sixth column, the execution time for rasterised maps increase more slowly than for line-blended flow maps, as the former depend more strongly on the rasterisation grid resolution than on the number of trajectories, at least up till $n = 10000$.

For the $n = 1180$ empirical Hannover trajectories, the map matching and initial flow map ($\varepsilon = 0$ m) have similar execution times to the $n = 1000$ synthetic Los Angeles trajectories. For the line-blended flow map with $\varepsilon = 2$ m, and the rasterised map with $\varepsilon_R = 2$ m, the execution times are higher due to the increased topological complexity of empirical trajectories as compared to synthetic trajectories.

For $n = 100, 1000, 10000$, map matching and line blending require ~1, ~5–10, ~30–50 min, respectively, which are feasible execution times for offline analyses of these small to moderate-sized data sets. For $n \gg 10000$, a promising approach to expedite the line blending computations is regionalisation. This consists of partitioning the study area into smaller regions, computing the flow map for each region, and then stitching these regionalised flow maps. This regionalisation is successfully employed for exact flow aggregation (Lovelace and Ellison 2018). Since all regionalised exact flow aggregations remain aligned to each other, it is straightforward to carry out the post hoc stitching. In contrast, different regionalised line-blended flow maps are not guaranteed to be aligned to each other, and it is an open question how to carry out this stitching. So we leave for future development the analysis of (a) data sets with a larger number of trajectories (like the full set of 1.5 million synthetic Los Angeles trajectories) and (b) online computations.

Our other future goal is the integration of the local alignment algorithms into an industry standard web GIS. In the mean time, we make available interactive web maps at https://mvstat.net/traj which allow the user to freely explore our proposed multi-scale traffic flow maps from the individual road to the city-wide level.

4. Conclusion

We have introduced an iterative workflow, which takes noisy GNSS vehicle trajectories as input and gives an accurate traffic flow map as output. Local alignment of flow segments is the key innovation. To carry out this local alignment, we defined a spatial relation to set up local reference flow segments, which then allowed us to align other nearby flow segments to this local reference segment. We presented solid evidence for the high level of spatial resolution and accuracy for our proposed locally aligned flow map for both synthetic and empirical trajectories. We demonstrated that they effectively reduce the noise of the input trajectories, and accurately aggregate the traffic flows on all road segments, and so they are a valuable addition to the toolkit for multi-scale traffic flow analysis.

Disclosure statement

No potential conflict of interest was reported by the author(s).

ORCID

Tarn Duong http://orcid.org/0000-0002-1198-3482

Data availability statement

The data that support the findings of this manuscript are openly available. The Hannover trajectories are hosted in the Leibniz University data depository https://doi.org/10.25835/9bidqxvl., and the Los Angeles trajectories in the Dryad depository https://doi.org/10.5061/dryad.4j0zpc8gf.

References

Anastasiou, C., S. H. Kim, and C. Shahabi. 2022a. "Generation of Synthetic Urban Vehicle Trajectories." In *2022 IEEE International Conference on Big Data (Big Data)*, edited by S. Tsumoto, Y. Ohsawa, L. Chen, D. Van den Poel, X. Hu, Y. Motomura, T. Takagi, L. Wu, Y. Xie, A. Abe, and V. Raghavan, 359–366. Osaka, Japan: IEEE. https://doi.org/10.1109/BigData55660.2022.10020237.

Anastasiou, C., S. H. Kim, and C. Shahabi. 2022b. *Synthetic Vehicle Trajectory Dataset for the Metropolitan City of Los Angeles Using DDTG*. Dryad. https://doi.org/10.5061/dryad.4j0zpc8gf.

Andrienko, N., and G. Andrienko. 2013. "Visual Analytics of Movement: An Overview of Methods, Tools and Procedures." *Information Visualization* 12:3–24. https://doi.org/10.1177/1473871612457601.

Chao, P., Y. Xu, W. Hua, and X. Zhou. 2020. "A Survey on Map-Matching Algorithms." In *Databases Theory and Applications*, edited by R. Borovica-Gajic, J. Qi, and W. Wang, 121–133. Vol. 12008. Cham: Springer. https://doi.org/10.1007/978-3-030-39469-1_10.

Dunnington, D., and E. Pebesma. 2020. *geos: Open Source Geometry Engine GEOS R API*. R Package Version 0.2.4. https://github.com/paleolimbot/geos.

Evans, S. P. 1976. "Derivation and Analysis of Some Models for Combining Trip Distribution and Assignment." *Transportation Research* 10:37–57. https://doi.org/10.1016/0041-1647(76)90100-3.

GIS OPS. 2023. *Valhalla Routing Engine Version 3.4.0 [Docker]*.

Herrera, J. C., D. B. Work, R. Herring, Q. Ban, X. Jacobson, and A. M. Bayen. 2010. "Evaluation of Traffic Data Obtained via GPS-Enabled Mobile Phones: The Mobile Century Field Experiment." *Transportation Research Part C: Emerging Technologies* 18 (4): 568–583. https://doi.org/10.1016/j.trc.2009.10.006.

Iqbal, M. S., C. F. Choudhury, P. Wang, and M. C. González. 2014. "Development of Origin–Destination Matrices Using Mobile Phone Call Data." *Transportation Research Part C: Emerging Technologies* 40:63–74. https://doi.org/10.1016/j.trc.2014.01.002.

Lovelace, R., and R. Ellison. 2018. "stplanr: A Package for Transport Planning." *The R Journal* 10 (2): 7–23. https://doi.org/10.32614/RJ-2018-053.

Miller, S., Z. Vander Laan, and N. Marković. 2020. "Scaling GPS Trajectories to Match Point Traffic Counts: A Convex Programming Approach and Utah Case Study." *Transportation Research Part E: Logistics & Transportation Review* 143:102105. https://doi.org/10.1016/j.tre.2020.102105.

Morgan, M., and R. Lovelace. 2021. "Travel Flow Aggregation: Nationally Scalable Methods for Interactive and Online Visualisation of Transport Behaviour at the Road Network Level." *Environment & Planning B: Urban Analytics & City Science* 48:1684–1696. https://doi.org/10.1177/2399808320942779.

Müllner, D. 2013. "fastcluster: Fast Hierarchical, Agglomerative Clustering Routines for R and Python." *Journal of Statistical Software* 53 (9): 1–18. https://doi.org/10.18637/jss.v053.i09.

Ortúzar, J. D., and L. G. Willumsen. 2011. *Modelling Transport*. 4th ed. Hoboken: Wiley.

Pebesma, E. 2018. "Simple Features for R: Standardized Support for Spatial Vector Data." *The R Journal* 10:439–446. https://doi.org/10.32614/RJ-2018-009.

Quddus, M. A., W. Y. Ochieng, and R. B. Noland. 2007. "Current Map-Matching Algorithms for Transport Applications: State-of-the Art and Future Research Directions." *Transportation Research Part C: Emerging Technologies* 15:312–328. https://doi.org/10.1016/j.trc.2007.05.002.

Saki, S., and T. Hagen. 2022. "A Practical Guide to an Open-Source Map-Matching Approach for Big GPS Data." *SN Computer Science* 3 (5): 415. https://doi.org/10.1007/s42979-022-01340-5.

Sghaier, T. 2019. "Développement d'un Outil de Calcul et de Visualisation des Flux de Conducteurs à Partir d'Enquêtes Origine-Destination [Development of Tools for the Computation and

Visualisation of Driver Flows from Origin-Destination Surveys]." Master's thesis, University of Orléans, France. In French.

Tobler, W. R. 1987. "Experiments in Migration Mapping by Computer." *The American Cartographer* 14:155–163. https://doi.org/10.1559/152304087783875273.

van der Meer, L., L. Abad, A. Gilardi, and R. Lovelace. 2021. *sfnetworks: Tidy Geospatial Networks*. R Package Version 0.6.4. https://luukvdmeer.github.io/sfnetworks.

Van Diggelen, F., and P. Enge. 2015. "The World's First GPS MOOC and Worldwide Laboratory Using Smartphones." Proceedings of the 28th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GNSS+ 2015), Tampa: 361–369.

Wood, J., J. Dykes, and A. Slingsby. 2010. "Visualisation of Origins, Destinations and Flows with OD Maps." *The Cartographic Journal* 47:117–129. https://doi.org/10.1179/000870410X12658023467367.

Zeng, Z., A. K. H. Tung, J. Wang, J. Feng, and L. Zhou. 2009. "Comparing Stars: On Approximating Graph Edit Distance." *Proceedings of the VLDB Endownment*, Lyon, France, 25–36. vol. 2. https://doi.org/10.14778/1687627.1687631.

Zhou, H., P. Xu, X. Yuan, and H. Qu. 2013. "Edge Bundling in Information Visualization." *Tsinghua Science and Technology* 18:145–156. https://doi.org/10.1109/TST.2013.6509098.

Zourlidou, S., J. Golze, and M. Sester. 2022. *Dataset: GPS Trajectory Dataset of the Region of Hannover, Germany*. Hannover, Germany: Leibniz University. https://doi.org/10.25835/9bidqxvl.

Appendix A. Details of algorithms

A1. *Map matched route*

The inputs to ST_ROUTE are the trajectory G and the map matching API M. The output is a map matched route $M(G)$. In Step 1, we compute the map matched route $M(G)$ from the trajectory G by calling the map matching API M. This map matched route $M(G) = \{\boldsymbol{m}_1, \ldots, \boldsymbol{m}_{n_M}\}$ is a linestring with n_M edges, with $n_M + 1$ points.

Algorithm 1. ST ROUTE – Map matched route

Input: G trajectory, M map matching API
Output: $M(G)$ map matched route
1: Compute map matched route $M(G)$ from empirical trajectory G

A2. *Snap node clustering of linestrings*

The inputs to ST_SNAPNODE are the $n_{\mathcal{F}}$ flow linestrings $\mathcal{F} = \{(f_1, \boldsymbol{f}_1), \ldots, (f_{n_{\mathcal{F}}}, \boldsymbol{f}_{n_{\mathcal{F}}})\}$ and the snap tolerance ε_S. The output are the snapped linestrings $\mathcal{F}^*$. In Step 1, we extract the boundary points of all flow linestrings. In Steps 2–3, we compute a single linkage clustering on all boundary points, and cut the dendrogram at height ε_S, resulting in C' clusters. In Steps 4–7, within each of these single linkage clusters, we compute a complete linkage clustering, and cut the dendrogram at height ε_S. This divides each single linkage cluster into C'' clusters, where all cluster members are at most ε_S distance from each other. In Steps 8–11, we compute the weighted spatial centroid in each of the C'' complete linkage clusters, weighted by the traffic flow values. In Step 12, we renumber the cluster labels from the above nested clusterings to approximate a 1-pass complete linkage clustering into C^* clusters. In Steps 13–16, within each of these C^* clusters, for all points of the corresponding flow linestrings which are closer than ε_S distance to the centroid, we snap them to the centroid. In Steps 17–18, we collate and sort all the $n_{\mathcal{F}^*}$ snapped linestrings $\mathcal{F}^* = \{(f_1^*, \boldsymbol{f}_1^*), \ldots, (f_{n_{\mathcal{F}^*}}^*, \boldsymbol{f}_{n_{\mathcal{F}^*}}^*)\}$.

Algorithm 2. ST SNAPNODE – Snap node clustering of linestrings

Input: $\mathcal{F}$ flow linestrings, ε_S snap tolerance
Output: $\mathcal{F}^*$ snap node clustered flow linestrings
1: Extract boundary points $B(\mathcal{F}) := \{\text{Start}(\boldsymbol{f}_1), \text{End}(\boldsymbol{f}_1), \ldots, \text{Start}(\boldsymbol{f}_{n_{\mathcal{F}}}), \text{End}(\boldsymbol{f}_{n_{\mathcal{F}}})\}$
2: Compute hierarchical clustering with single linkage on $B(\mathcal{F})$
3: Cut single linkage dendrogram at height ε_S to compute C' cluster labels
4: **for** $i := 1$ to C' **do**
5: Extract ith cluster of boundary points $B'_i := \{\boldsymbol{b}'_1, \ldots, \boldsymbol{b}'_{n_{B'}}\}$
6: Compute hierarchical clustering with complete linkage on B'_i
7: Cut complete linkage dendrogram at height ε_S to compute C'' cluster labels
8: **for** $j := 1$ to C'' **do**
9: Extract jth cluster of boundary points $B''_j := \{\boldsymbol{b}''_1, \ldots, \boldsymbol{b}''_{n_{B''}}\}$
10: Extract corresponding flows $\{\boldsymbol{f}''_1, \ldots, \boldsymbol{f}''_{n_{B''}}\}$ from $\mathcal{F}$
11: Compute $\boldsymbol{b}^*_j :=$ weighted centroid of $\{\boldsymbol{f}''_1, \ldots, \boldsymbol{f}''_{n_{B''}}\}$, weights $:= f''_1, \ldots, f''_{n_{B''}}$
12: Renumber collated complete linkage cluster labels to unique C^* labels
13: **for** $i := 1$ to C^* **do**
14: Extract corresponding flow linestrings $\mathcal{F}^*_i := \{(f^*_1, \boldsymbol{f}^*_1), \ldots, (f^*_{n_{B^*}}, \boldsymbol{f}^*_{n_{B^*}})\}$ from $\mathcal{F}$
15: Snap points of $\{\boldsymbol{f}^*_1, \ldots, \boldsymbol{f}^*_{n_{B^*}}\}$ within ε_S distance of cluster centroid to $\boldsymbol{b}^*_i$
16: $\mathcal{F}^*_i :=$ rejoin snapped and unsnapped points into linestrings
17: Collate snapped linestrings $\{\mathcal{F}^*_1, \ldots \mathcal{F}^*_{C'^*}\}$ into $\mathcal{F}^* := \{(f^*_1, \boldsymbol{f}^*_1), \ldots, (f^*_{n_{\mathcal{F}^*}}, \boldsymbol{f}^*_{n_{\mathcal{F}^*}})\}$
18: Sort $\mathcal{F}^*$ by descending flow and length

A3. *Split nodes at interior intersections of linestrings*

The inputs to ST_SPLITNODE are the flow linestrings $\mathcal{F}$ and the node splitting type S. The output are the split linestrings with added nodes $\mathcal{F}^*$. We deploy, for $S =$ 'subdivision' in Steps 1–3, to_spatial_subdivision in the sfnetworks package (van der Meer et al. 2021), and for $S =$ 'unary' in Steps 4–5, geos unary union in the geos package (Dunnington and Pebesma 2020).

Algorithm 3. ST SPLITNODE – Split nodes at interior intersections of linestrings

Input: $\mathcal{F}$ flow linestrings, S node splitting type
Output: $\mathcal{F}^*$ node split flow linestrings
1: Initialise local network $\mathcal{N}^*$ with linestrings $\mathcal{F}$
2: **if** $S ==$ "subdivision" **then**
3: $\quad \mathcal{F}^* :=$ `sfnetworks::to_spatial_subdivision`$(\mathcal{N}^*)$
4: **else if** $S ==$ "unary" **then**
5: $\quad \mathcal{F}^* :=$ `geos::geos_unary_union`$(\mathcal{N}^*)$

A4. *Blend candidate linestrings onto reference linestring*

The inputs to ST_LINEBLEND are the reference linestring $\boldsymbol{f}_{ref}$, the $n_{\mathcal{F}_{cand}}$ candidate linestrings $\mathcal{F}_{cand}$, and the blend tolerance ε. The output are the modified reference linestring and $n_{\mathcal{F}_{cand}}$ modified candidate flow linestrings, all with added interior points for accurate calculation of exactly equal linestring segments. In Step 1, we initialize a local network graph $\mathcal{N}^*$ with the reference linestring. In Steps 2–3, we extract all points of the candidate linestrings, and use the network_blend function in the sfnetworks package (which we denote as ST_NETWORK_BLEND) to blend efficiently these points into the reference linestring (van der Meer et al. 2021). ST_NETWORK_BLEND projects the candidate linestrings onto the reference linestring, and explicitly adds them to the network, thereby creating new edges in $\mathcal{N}^*$. The result is a local network graph with more, shorter edges and with nodes at the projected candidate points, and whose union is the reference linestring. In Steps 4–5, we extract the $n_{\mathcal{F}_{cand}}$ blended candidate linestrings with these added interior points by applying the shortest path search between the start and end point of each blended candidate linestring along the network graph $\mathcal{N}^*$. Step 6 involves collating the blended candidate linestrings. Step 7 is the equivalent for the blended reference linestring.

Algorithm 4. ST LINEBLEND – Blend candidate linestrings onto reference linestring

Input: $\boldsymbol{f}_{\text{ref}}$ reference, $\mathcal{F}_{\text{cand}}$ candidate flow linestrings, ε blend tolerance
Output: $\boldsymbol{f}^*_{\text{ref}}$ blended reference, $\mathcal{F}^*_{\text{cand}}$ blended candidate flow linestrings
1: Initialise local network graph $\mathcal{N}^*$ with $\boldsymbol{f}_{\text{ref}}$
2: Extract all points G_{cand} of candidate linestrings and flow values f_{cand} from $\mathcal{F}_{\text{cand}}$
3: Update network by blending candidate points $\mathcal{N}^* :=$ `ST_NETWORK_BLEND`$(\mathcal{N}^*, G_{\text{cand}}, \varepsilon)$
4: **for** $i := 1$ to $n_{\mathcal{F}_{\text{cand}}}$ **do**
5: $\quad \boldsymbol{f}^*_{\text{cand},i} :=$ shortest path from Start$(\boldsymbol{f}_{\text{cand},i})$ to End$(\boldsymbol{f}_{\text{cand},i})$ along network $\mathcal{N}^*$
6: Collate $\mathcal{F}^*_{\text{cand}} := \{\boldsymbol{f}^*_{\text{cand},1}, \ldots, \boldsymbol{f}^*_{\text{cand},n_{\mathcal{F}_{\text{cand}}}}\}$
7: $\boldsymbol{f}^*_{\text{ref}} :=$ shortest path from Start$(\boldsymbol{f}_{\text{ref}})$ to End$(\boldsymbol{f}_{\text{ref}})$ along network $\mathcal{N}^*$

A5. *Snap candidate-touching linestrings onto reference linestring*

The inputs to ST_SNAP_CAND_TOUCH are the reference linestring $\boldsymbol{f}_{ref}$, the $n_{\mathcal{F}_{cand}}$ candidate linestrings $\mathcal{F}_{cand}$, the $n_{\mathcal{F}_{ct}}$ candidate-touching linestrings $\mathcal{F}_{ct}$, and the snap tolerance ε_S. The output are $\mathcal{F}^*_{ct}$ the snapped candidate-touching flow linestrings. In Steps 1–8, we iterate over each candidate-touching linestring. In Steps 2–3, we compute the intersections between the candidate-touching

Algorithm 5. ST SNAP CAND TOUCH – Snap candidate-touching linestrings onto reference linestring

Input: $\boldsymbol{f}_{\text{ref}}$ reference, $\mathcal{F}_{\text{cand}}$ candidate, $\mathcal{F}_{\text{ct}}$ candidate-touching flow linestrings, ε_S snap tolerance
Output: $\mathcal{F}^*_{\text{ct}}$ snapped candidate-touching flow linestrings
1: **for** $i := 1$ to $n_{\mathcal{F}_{\text{ct}}}$ **do**
2: $\quad \boldsymbol{g}^*_{\text{ct},i} := \boldsymbol{f}_{\text{ct},i} \cap \{\text{Start}(\mathcal{F}_{\text{cand}}), \text{End}(\mathcal{F}_{\text{cand}})\}$
3: $\quad d^*_{i,\text{Start}} := \texttt{ST\_DIST}(\boldsymbol{g}^*_{\text{ct},i}, \text{Start}(\boldsymbol{f}_{\text{ref}}))$; $d^*_{i,\text{End}} := \texttt{ST\_DIST}(\boldsymbol{g}^*_{\text{ct},i}, \text{End}(\boldsymbol{f}_{\text{ref}}))$
4: $\quad$ **if** $(d^*_{i,\text{Start}} \leq \varepsilon_S$ and $d^*_{i,\text{Start}} \leq d^*_{i,\text{End}})$ **then**
5: $\quad\quad \boldsymbol{f}^*_{\text{ct},i} :=$ snap $\boldsymbol{f}_{\text{ct},i}$ to $\text{Start}(\boldsymbol{f}_{\text{ref}})$
6: $\quad$ **else if** $(d^*_{i,\text{End}} \leq \varepsilon_S$ and $d^*_{i,\text{End}} \leq d^*_{i,\text{Start}})$ **then**
7: $\quad\quad \boldsymbol{f}^*_{\text{ct},i} :=$ snap $\boldsymbol{f}_{\text{ct},i}$ to $\text{End}(\boldsymbol{f}_{\text{ref}})$
8: $\quad$ **else** $\boldsymbol{f}^*_{\text{ct},i} :=$ snap $\boldsymbol{f}_{\text{ct},i}$ to nearest point of $\boldsymbol{f}_{\text{ref}}$
9: Collate $\mathcal{F}^*_{\text{ct}} := \{\boldsymbol{f}^*_{\text{ct},1}, \ldots, \boldsymbol{f}^*_{\text{ct},n_{\mathcal{F}_{\text{ct}}}}\}$

linestring and the boundary points of the candidate linestrings, and the respective distances. In Steps 4–7, if this intersection point is within ε_S distance to the boundary points of $\boldsymbol{f}_{ref}$, then we snap the candidate-touching linestring to the closest boundary point. In Step 8, if the intersection point is not within ε_S distance, then we snap it to the nearest point on $\boldsymbol{f}_{ref}$. These snapping operations are similar to those in Steps 15–16 in ST_SNAPNODE in Algorithm 2, and ensure that we maintain connectivity between $\mathcal{F}_{ct}$ and $\boldsymbol{f}_{ref}$. In Step 9, we collate these snapped linestrings.

A6. *Compute line blending priority*

The inputs to ST_LINEBLEND_PRIORITY are the flow linestrings $\mathcal{F}$, the number of connected edges for the reference linestrings k, and the blend tolerance ε. The output is a set of non-overlapping reference linestrings $\mathcal{F}_{ref}$, a set of unique candidate linestrings $\mathcal{F}_{cand}$, and a set of (potentially repeated) candidate-touching $\mathcal{F}_{ct}$ linestrings. In Steps 1–2, we set up a network graph from the linestrings $\mathcal{F}$, and then extract $\mathcal{K}$, all simple paths (i.e. all paths composed of connected, unique edges) with k-edges. In Steps 3–4, we compute the

Algorithm 6. ST LINEBLEND PRIORITY – Compute line blending priority

Input: $\mathcal{F}$ flow linestrings, k #edges, ε blend tolerance
Output: $(\mathcal{F}_{\text{ref}}, \mathcal{F}_{\text{cand}}, \mathcal{F}_{\text{ct}})$ reference, candidate, candidate-touching linestrings
1: Initialise network graph $\mathcal{N}^*$ from linestrings $\mathcal{F}$
2: Extract all simple paths of length k edges $\mathcal{K} := \{\boldsymbol{k}_1, \ldots, \boldsymbol{k}_{n_{\mathcal{K}}}\}$ from $\mathcal{N}^*$
3: **for** $i := 1$ to $n_{\mathcal{K}}$ **do**
4: $\quad f_i :=$ weighted mean of k flows, weight $:=$ len(Edge($\boldsymbol{k}_i$))
5: Sort $\mathcal{K}$ by descending flow and combined length
6: Initialise $\mathcal{K}^* := \{\boldsymbol{k}_1\}$
7: **for** $i := 2$ to $n_{\mathcal{K}}$ **do**
8: $\quad$ **if** $(\boldsymbol{k}_i \cap \mathcal{K}^* = \{\})$ **then** $\mathcal{K}^* := \mathcal{K}^* \cup \{\boldsymbol{k}_i\}$
9: Initialise $\mathcal{Q} := \mathcal{F}_{\text{ref}} := \mathcal{F}_{\text{cand}} := \mathcal{F}_{\text{ct}} := \{\}$
10: **for** $i := 1$ to $n_{\mathcal{K}^*}$ **do**
11: $\quad$ **if** $(\text{Edge}(\boldsymbol{k}^*_i) \notin \mathcal{F}_{\text{cand}})$ **then**
12: $\quad\quad$ Set reference linestring $\boldsymbol{k}_{\text{ref}} := \boldsymbol{k}^*_i$
13: $\quad\quad$ Search $\mathcal{F}^*_{\text{cand}} := \{\boldsymbol{f} \in \mathcal{F} \backslash (\text{Edge}(\mathcal{F}_{\text{ref}}) \cup \{\boldsymbol{k}_{\text{ref}}\} \cup \mathcal{F}_{\text{cand}})\colon \boldsymbol{f} \subseteq \texttt{ST\_BUFFER}(\boldsymbol{k}_{\text{ref}}, \varepsilon)\}$
14: $\quad\quad$ **if** $\mathcal{F}^*_{\text{cand}} \neq \{\}$ **then**
15: $\quad\quad\quad$ Update $\mathcal{F}_{\text{ref}} := \mathcal{F}_{\text{ref}} \cup \{\boldsymbol{k}_{\text{ref}}\}, \mathcal{F}_{\text{cand}} := \mathcal{F}_{\text{cand}} \cup \mathcal{F}^*_{\text{cand}}$
16: $\quad\quad\quad$ Search $\mathcal{F}_{\text{ct}} := \{\boldsymbol{f} \in \mathcal{F} \backslash \{\text{Edge}(\mathcal{F}_{\text{ref}}) \cup \mathcal{F}_{\text{cand}}\}\colon \texttt{ST\_TOUCHES}(\mathcal{F}^*_{\text{cand}}, \boldsymbol{f})\}$
17: $\quad\quad\quad$ Update $\mathcal{Q} := \mathcal{Q} \cup \{(\{\boldsymbol{k}_{\text{ref}}\}, \mathcal{F}^*_{\text{cand}}, \mathcal{F}^*_{\text{ct}})\}$
18: Extract $\mathcal{F}_{\text{cand}} := \{\mathcal{F}^*_{\text{cand},1}, \ldots, \mathcal{F}^*_{\text{cand},n_{\mathcal{Q}}}\}, \mathcal{F}_{\text{ct}} := \{\mathcal{F}^*_{\text{ct},1}, \ldots, \mathcal{F}^*_{\text{ct},n_{\mathcal{Q}}}\}$ from $\mathcal{Q}$

weighted mean of the k flow values, weighted by the length of individual edges, and assign this to the entire k-edge path. The function Edge$(\cdot)$ extracts the component edges from a k-edge path. In Step 5, we sort the flow linestrings, in descending order of their flow values and length. In Steps 6–8, we construct a maximal set of non-overlapping k-edge paths $\mathcal{K}^*$. We initialise $\mathcal{K}^*$ to contain the first path. We iterate through $\mathcal{K}$ and, if this path in $\mathcal{K}$ does not overlap the current $\mathcal{K}^*$, then we add it to $\mathcal{K}^*$. For $k = 1$, we bypass Steps 2–8. In Steps 9–18, we step through the $\boldsymbol{k}_i^* \in \mathcal{K}^*$, starting from the linestring with the highest flow and length. In Steps 11–13, if the current flow linestring $\boldsymbol{k}_i^*$ has no incident edges in the candidate linestrings $\mathcal{F}_{cand}$, then we set it to be a reference linestring. In Step 13, we search for candidate linestrings for $\boldsymbol{k}_i^*$, from those linestrings which are not already reference or candidate linestrings, according to Equation (1). In Steps 14–17, if there is at least one candidate linestring, then we update $\mathcal{F}_{ref}, \mathcal{F}_{cand}$, and $\mathcal{F}_{ct}$. In Step 18, we extract the candidate $\mathcal{F}_{cand}$ and candidate-touching $\mathcal{F}_{ct}$ linestring sets.

A7. *Locally align segment flows using line blending*

The inputs to ST_OVERLINE_LINEBLEND are the flow linestrings $\mathcal{F}$, the split node type S, the number of connected edges k, the blend tolerance ε, and the snap tolerance ε_S. The output are the modified reference linestring $\boldsymbol{f}_{ref}^*$ and projected candidate flow linestrings $\mathcal{F}_{cand}^*$, all with added interior points. In Step 1, we node split the flow linestrings $\mathcal{F}$. In Step 2, we node snap the flow linestrings. In Step 3, we set up the priority for the line blending, and in Step 4 we store all the linestrings that will not be modified by the line blending in $\mathcal{F}^c$. In Steps 5–8, we iterate over each reference linestring. In Step 6, we update the reference and candidate flow linestrings by blending the candidate linestrings. In Step 7, we apply ST_OVERLINE_PLANR within each of the k-edges of the updated reference linestrings. In Step 8, we compute the weighted mean flow, weighted by the length of the corresponding sub-segments, and assign it as the flow value to all sub-segments. The result from Steps 6–8 is a modified k-edge reference linestring with k aggregated flows. In Step 9, we snap the candidate-touching linestrings to the reference linestring to maintain connectivity. In Steps 10–11, we collate the modified linestrings from Steps 5–9, with the unmodified linestrings $\mathcal{F}^c$ from Step 4, and sort them in descending flow and length.

Algorithm 7. ST OVERLINE LINEBLEND – Locally align segment flows using line blending

Input: $\mathcal{F}$ flow linestrings, S split node, k #edges, ε blend tolerance, ε_S snap tolerance
Output: $\mathcal{F}^*$ aggregated aligned flow linestrings

1: $\mathcal{F} := \texttt{ST\_SPLITNODE}(\mathcal{F}, S)$
2: $\mathcal{F} := \texttt{ST\_SNAPNODE}(\mathcal{F}, \varepsilon_S)$
3: $\{\mathcal{F}_{\text{ref}}, \mathcal{F}_{\text{cand}}, \mathcal{F}_{\text{ct}}\} := \texttt{ST\_LINEBLEND\_PRIORITY}(\mathcal{F}, k, \varepsilon)$
4: $\mathcal{F}^c := \mathcal{F} \backslash (\text{Edge}(\mathcal{F}_{\text{ref}}) \cup \mathcal{F}_{\text{cand}} \cup \mathcal{F}_{\text{ct}})$
5: **for** $i := 1$ to $n_{\mathcal{F}_{\text{ref}}}$ **do**
6: $\quad \{\boldsymbol{f}_{\text{ref},i}^*, \mathcal{F}_{\text{cand},i}^*\} := \texttt{ST\_LINEBLEND}(\boldsymbol{f}_{\text{ref},i}, \mathcal{F}_{\text{cand},i}, \varepsilon)$
7: $\quad \boldsymbol{f}_{\text{ref},i}^* := \texttt{ST\_OVERLINE\_PLANR}(\boldsymbol{f}_{\text{ref},i}^* \cup \mathcal{F}_{\text{cand},i}^*)$
8: $\quad f_{\text{ref},i}^* :=$ weighted mean of $\boldsymbol{f}_{\text{ref},i}^*$, weight $:=$ len(Edge($\boldsymbol{f}_{\text{ref},i}^*$))
9: $\quad \mathcal{F}_{\text{ct},i}^* := \texttt{ST\_SNAP\_CAND\_TOUCH}(\boldsymbol{f}_{\text{ref},i}^*, \mathcal{F}_{\text{cand},i}, \mathcal{F}_{\text{ct},i}, \varepsilon_S)$
10: $\mathcal{F}^* := \mathcal{F}_{\text{ref}}^* \cup \mathcal{F}_{\text{ct}}^* \cup \mathcal{F}^c$
11: Sort $\mathcal{F}^*$ in descending order of flow and length

A8. *Compute initial locally aligned flow map*

The inputs to ST_OVERLINE_INITIAL are the map matched routes $\mathcal{M}$, the split node type S. The output is an initial aligned flow map. In Step 1, we apply ST_OVERLINE_PLANR to the map matched routes to produce a flow map $\mathcal{F}$. In Step 2, we apply ST_SPLITNODE to this flow map.

Algorithm 8. ST OVERLINE INITIAL – Compute initial locally aligned flow map

Input: $\mathcal{M}$ map matched routes, S split node, ε_D simplify tolerance
Output: $\mathcal{F}$ initial aligned flow map
1: $\mathcal{F} :=$ ST_OVERLINE_PLANR($\mathcal{M}$)
2: $\mathcal{F} :=$ ST_SPLITNODE($\mathcal{F}, S$)

A9. *Compute iterated locally aligned flow map*

The inputs of ST_OVERLINE are the initial flow map $\mathcal{F}$, the split node type *S*, the blend tolerance ε, the snap tolerance ε_S, the number of connected edges $\boldsymbol{k}$, and the maximum number of iterations $j_{\max}$. The output is an iteration of a locally aligned flow map. Steps 1–6 is the iteration of ST_OVERLINE_LINEBLEND for the *k*-edge reference linestrings. For each *k* in $\boldsymbol{k}$, we iterate until $j_{\max}$ is reached or two consecutive flow maps are identical. Within each iteration, we search for the *k*-edge reference linestrings, blend the candidate linestrings, and snap the candidate-touching linestrings, and compute the flow aggregation. In Steps 7–10 is some housekeeping in ST_OVERLINE_PRUNE, where we remove pseudo nodes, remove leaf edges with low flow values, and replace thin loops with non-looped linestrings.

Algorithm 9. ST OVERLINE – Compute iterated locally aligned flow map

Input: $\mathcal{F}$ initial flow map, S split node, $\boldsymbol{k}$ #edges, $j_{\max}$ max #iterations, ε blend tolerance, ε_S snap tolerance
1: **for** k **in** $\boldsymbol{k}$ **do**
2: $\quad\mathcal{F}_{\text{prev}} := \{\}$; $j := 0$
3: $\quad$**while** $j < j_{\max}$ and $\mathcal{F}_{\text{prev}} \neq \mathcal{F}$ **do**
4: $\quad\quad\mathcal{F}_{\text{prev}} := \mathcal{F}$; $j := j + 1$
5: $\quad\quad\mathcal{F} :=$ ST_OVERLINE_LINEBLEND($\mathcal{F}, k, \varepsilon, \varepsilon_S$)
6: $\quad\quad\mathcal{F} :=$ ST_OVERLINE_PLANR($\mathcal{F}$)
7: $\mathcal{F}_{\text{prev}} := \{\}$; $j := 0$
8: **while** $j < j_{\max}$ and $\mathcal{F}_{\text{prev}} \neq \mathcal{F}$ **do**
9: $\quad\mathcal{F}_{\text{prev}} := \mathcal{F}$; $j := j + 1$
10: $\quad\mathcal{F} :=$ ST_OVERLINE_PRUNE($\mathcal{F}$)