CellNet: Inferring Road Networks from GPS Trajectories

RADU MARIESCU-ISTODOR and PASI FRÄNTI, University of Eastern Finland

Road networks are essential nowadays, especially for people travelling to large, unfamiliar cities. Moreover, cities are constantly growing and road networks need periodic updates to provide reliable information. We propose an automatic method to generate the road network using a GPS trajectory dataset. The method, called CellNet, works by first detecting the intersections (junctions) using a clustering-based technique and then creating the road segments in-between. We compare CellNet against conceptually different alternatives using Chicago and Joensuu datasets. The results show that CellNet provides better accuracy and is less sensitive to parameter setup. The size of the generated road network is only 25% of the networks produced by other methods. This implies that the network provided by CellNet has much less redundancy.

CCS Concepts: • **Information systems** → *Location based services*; *Geographic information systems*;

Additional Key Words and Phrases: GPS routes, road network, grid, similarity, clustering

1 INTRODUCTION

In recent years, navigation- and location-based services have seen a rise in development. For these applications to work reliably, up-to-date road networks are essential. Maintaining the road networks requires extensive manual editing, which has led researchers to develop road network inference algorithms to automate this process. The goal is to create a directed graph that represents the connectivity and geometry of the underlying roads in a region. These algorithms can also be applied to update existing road networks or to be used in applications that road networks do not cover, such as pedestrian networks (Kasemsuppakorn and Karimi 2013).

Several different approaches exist for automatically constructing a road network. The earliest methods were based on aerial images (Tavakoli and Rosenfeld 1982). They extract edges and then group them into shapes, separating buildings from roads. To find the roads, the method in Hu et al. (2007) makes several initial guesses. A road tree is built for each initial guess by tracking along road segments in one or more directions. By merging the resulting trees, a road network is created. Barsi and Heipke (2003) focus on the task of finding road intersections by analysing the aerial images using a neural network.

The use of aerial images has limitations because roads possess varying features such as colour, intensity, shadows, and variable widths (Figure 1). In addition, obtaining the direction of travel

Authors' addresses: R. Mariescu-Istodor and P. Fränti, Machine Learning Group, School of Computing, University of Eastern Finland, Joensuu, Finland; emails: {radum, franti}@cs.uef.fi.

Fig. 1. Aerial images of a city area (left) and countryside region (right). Before GPS tracking became popular, roads were extracted from aerial images by image processing techniques. Visual methods have been inspired by this approach.
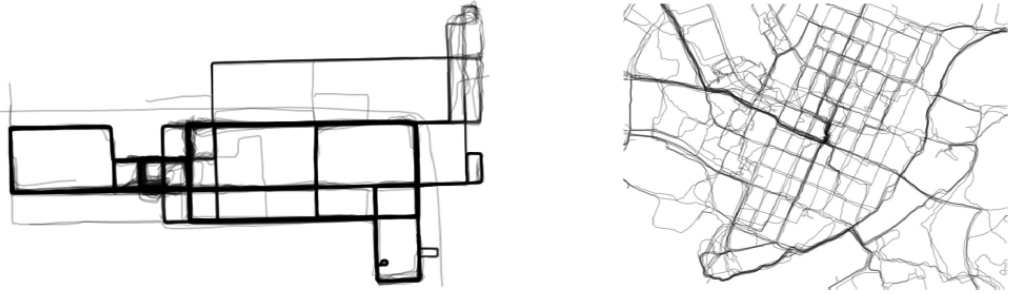

Fig. 2. GPS routes in Chicago, USA (left) and Joensuu, Finland (right). Modern extraction techniques rely merely on GPS data.

for roads is not possible using image data. Furthermore, collecting new aerial images after road construction work is costly. For these reasons, methods based on trajectories recorded using global positioning systems (GPS) have been developed. GPS technology provides a cheap alternative to aerial images owing to its built-in positioning capability, which is available in consumer devices such as smart phones, tablets, watches, and cameras. This technology is utilised in location-based services, navigation, and when tracking user movements. As a consequence, many GPS trajectories, referred to here as *routes*, have become available and can be used to obtain road network information (Figure 2).

Visual methods (Figure 3) use route data to form binary images, which are processed using image-processing techniques. In Chen and Cheng (2008), the routes are first converted to a binary image. Then the image is processed by morphological operations and a thinning operation to produce an image skeleton, which represents the road network. Davies et al. (2006) use kernel-density estimation (KDE) to produce a density map (image) of the routes. This image is then blurred and a density threshold is applied to filter out parts that contain too little data. The outlines are extracted using a contour-following algorithm, and the centre-lines of these outlines are computed using the Voronoi graph. These centre-lines are used to depict the underlying network. Biagioni and Eriksson (2012b) also use KDE; however, they apply multiple thresholds to compute a grey-scale skeleton. In this way, less-frequently travelled roads are also tolerated. More recently, Wang et al. (2015) demonstrate how Morse theory can be used to extract the road network from a KDE.
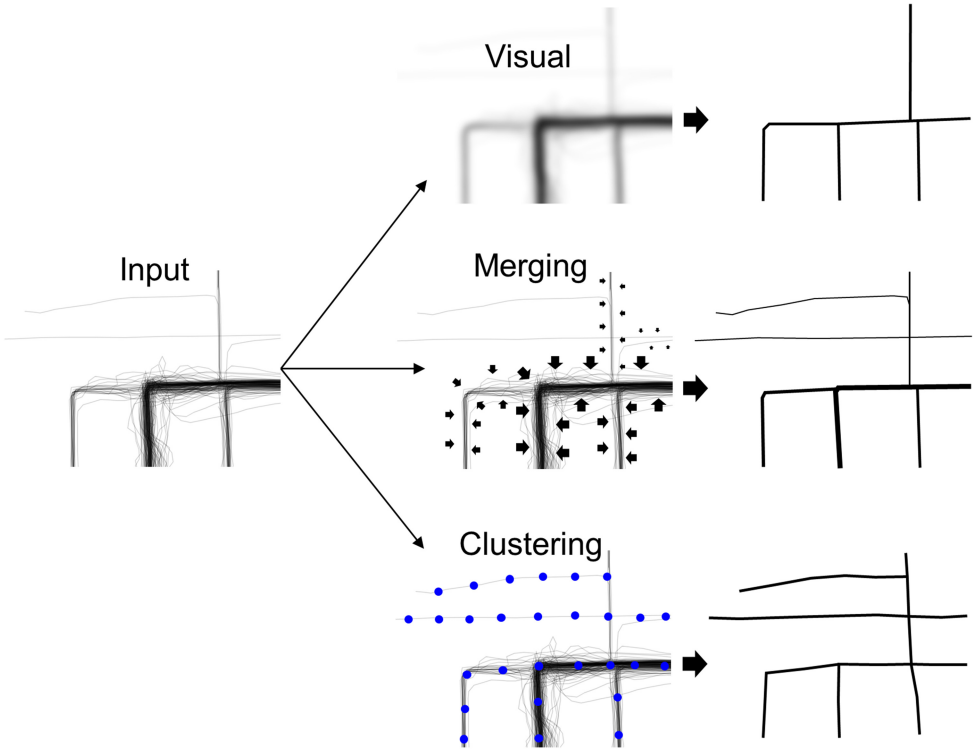
Fig. 3. Three conceptually different road network generation techniques: visual, merging, and clustering.

Route merging methods (Niehöfer et al. 2009; Cao and Krumm 2009) combine routes one-by-one to form a graph (Figure 3). If a route segment is already part of the graph, a weight corresponding to that particular segment is increased. Finally, segments with too low weights are removed from the network. Cao and Krumm (2009) perform a refining step on the routes prior to the merge, to reduce GPS inaccuracies. This step is an iterative process that uses an attractive physical force (Khanna 1999) between route points to obtain better representatives. A secondary attractive force is used to prevent the route points from moving too far from their original locations.

Clustering methods have also been used (Figure 3). In Edelkamp and Schrödl (2003), seed points (representatives) are first placed at a fixed distance over the routes in the dataset. Then these locations are fine-tuned by *k*-means algorithm. For roads that allow vehicles to move on several lanes, the authors also present a lane-finding strategy. In Schroedl et al. (2004), the bounding box of each intersection is analysed to compute the local turn-lane geometry.

The merging and clustering methods perform poorly in regions of high GPS error. In such regions, unwanted intersections and multiple spurious road segments are created. The visual methods work better in such situations if the density threshold is set high enough, but the drawback is that the parts containing few routes are omitted from the process and only a partial network is generated.

We argue that finding the correct intersections (junctions) is the key to generate a high-quality road network, because this ensures that GPS error affects only the shape of the roads and not the connectivity of the graph. Fathi and Krumm (2010) focus on this challenge. They slide a circular shape descriptor over the GPS data; the descriptor is trained using positive and negative samples from known locations. After intersections have been obtained, road segments are generated
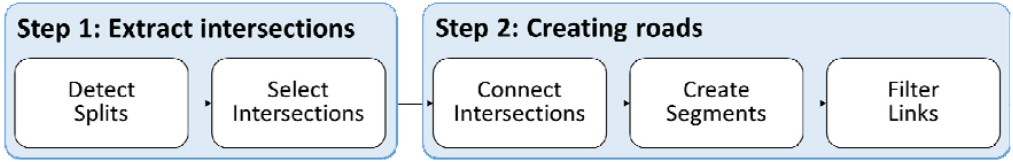
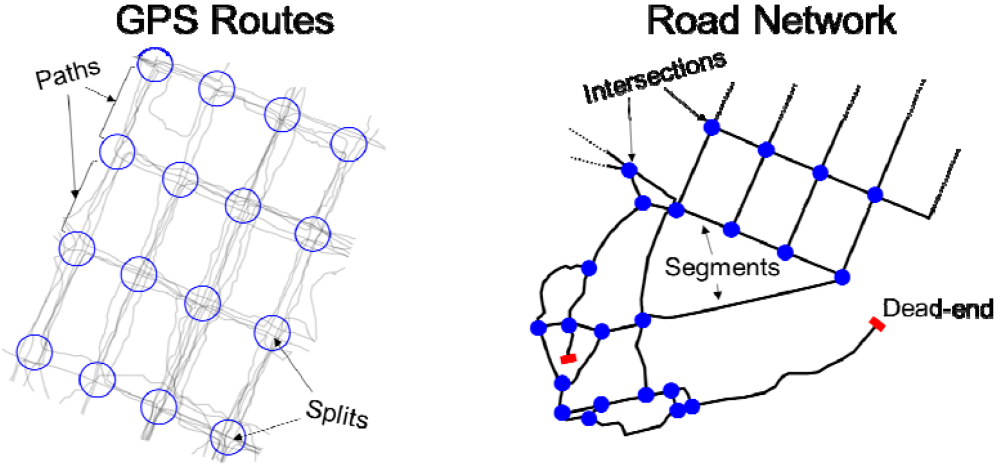Fig. 4. The steps performed by CellNet to infer a road network.

Fig. 5. Diagram showing terminology used to discuss GPS routes and road networks.

using the routes. Karagiorgou and Pfoser (2012) and Karagiorgou et al. (2013) also first find the intersections by detecting changes in movement patterns.

In this article, we present CellNet, a two-step method for inferring road networks (Figure 4). CellNet first identifies intersections by clustering the route points around the regions where routes split into several directions. Unlike other approaches (Barsi and Heipke 2003; Fathi and Krumm 2010), our method does not require the training of a classifier. In the second step, we generate the roads between the detected intersections using the route segments in the region. Finally, we optimise the network to avoid redundant and overly complex roads.

Figure 5 shows a graphic explanation of the terminology. The details of how to find the intersections are provided in Section 2 of this article. The steps in creating the roads are explained in Section 3. The proposed method is evaluated in Section 4 and is compared with four existing approaches: two visual methods (Davies et al. 2006; Biagioni and Eriksson 2012b), a merging method (Cao and Krumm 2009) and a clustering method (Edelkamp and Schrödl 2003). Biagioni and Eriksson (2012a) have implemented these methods and made them publically available.

2 EXTRACT INTERSECTIONS

Intersections are places in which more than two roads connect. To detect potential intersections from GPS routes, we apply the two processes shown in Figure 6. First, we analyse the neighbourhood of each point to detect *splits*. A split is defined as a point at which routes head off in more than two principal directions (Figure 7). Multiple splits are often found at the same intersection, especially if the intersection is large. From the detected splits, we measure the frequency of routes passing through the area. Splits having a higher frequency than their neighbours (local maxima) are selected as intersections.
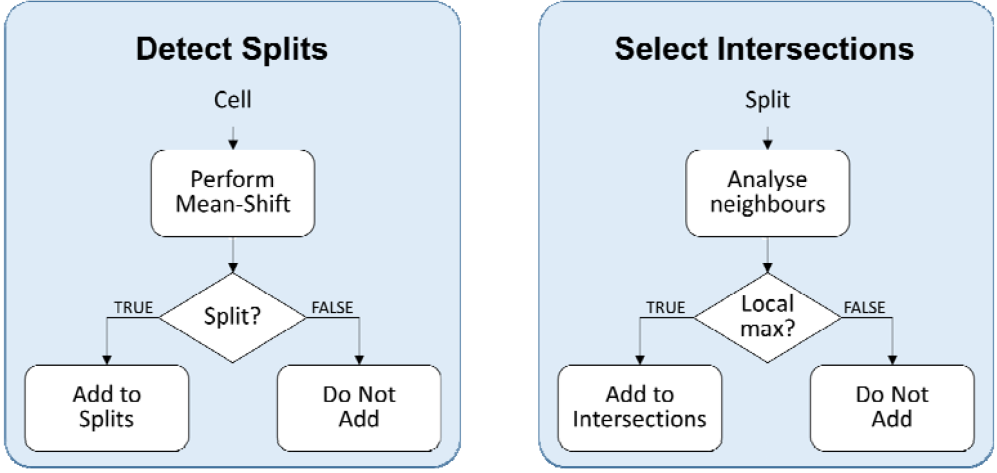
Fig. 6. Steps performed to detect splits (left panel) and to select local maxima (right panel).
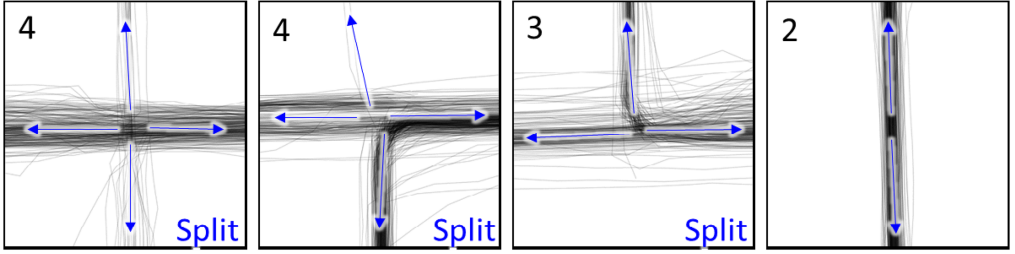

Fig. 7. Four examples of locations at which routes head off into several principal directions. The directions are highlighted by arrows. The first three examples are splits, according to our definition, whereas the last is not. The numbers (upper left corners) indicate the quantity of principal directions.

2.1 Detect Splits

To detect the splits, we analyse all locations through which the routes passed. To do this efficiently, we divide the space by a grid with cell length $L = 25$m (recommended). For every grid cell, we maintain information containing the cell's location, indexes of all routes passing through it and the total number of routes. We accumulate the evidence by processing the routes point-by-point. Gaps can appear in the cell representation in places where consecutive route points are further apart than L (Figure 8). Owing to such gaps, it is possible that the method might miss some intersections. We therefore use interpolation to handle this problem. A more detailed explanation on the use of the grid is given in Mariescu-Istodor and Fränti (2017).

After collecting the information, we process each cell only once. This approach makes the method much more scalable as the calculations depend far less on the number of routes than on the size of the area through which they pass. In this regard, our method resembles the visual-based approaches, but it uses route information and is not limited to image-processing methods.

The process is as follows: We first transfer the location of the cell closer to the stream of routes using the mean-shifting algorithm (Cheng 1995), which is basically a mode-seeking algorithm. At each step, it defines a fixed-radius neighbourhood and calculates the average location of the route points in this neighbourhood. The location is then updated to this average and the process
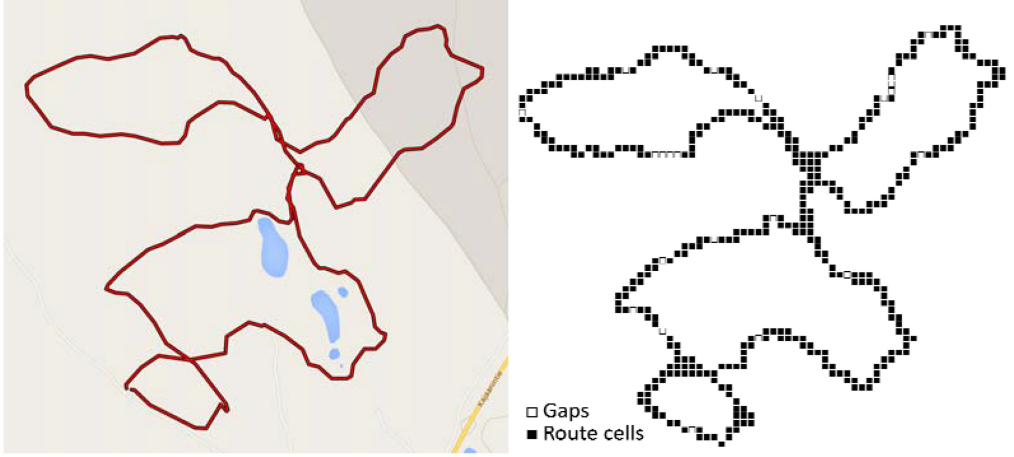
Fig. 8. A sample route (top panel) and the cell representation with cell size 25m × 25m (lower panel). The gaps are filled using linear interpolation.
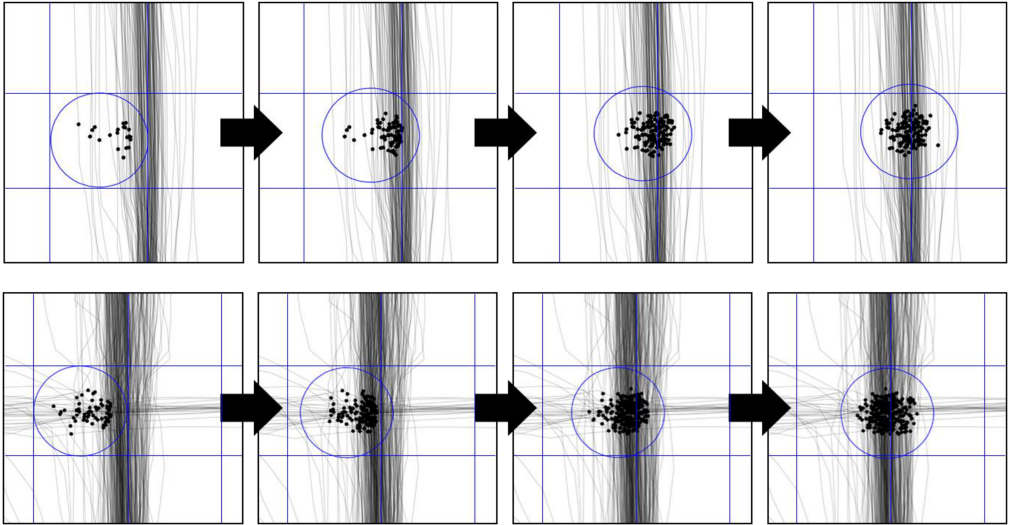
Fig. 9. Two examples of the mean-shift algorithm. The initial location gradually moves towards the centre of the routes. If an intersection is nearby, the location is likely to end up at its centre.

is repeated until the location stabilises. Figure 9 shows two examples of the mean-shift algorithm. Through this process, a location can sometimes end up in a different cell from the one where it started.

After the location has been tuned, we analyse the neighbourhood to detect the principal directions of movements. For this purpose, we define a *split descriptor*, which consists of two parts: the *origin* and the *extremity*. The origin is an L-radius circle around the tuned location. The extremity is a circular band of width L, situated at R metres from the origin (Figure 10). We recommend using the values $L = 25$m and $R = 80$m, although their exact choice is not critical.

Fig. 10. (A) The split descriptor composed of the origin and the extremity. (B) A sample route traversing through the point of interest; points inside the extremity are highlighted. (C) The points inside the extremity are averaged in each of the two directions to create the representatives. (D) Representatives of all routes passing through the point of interest.

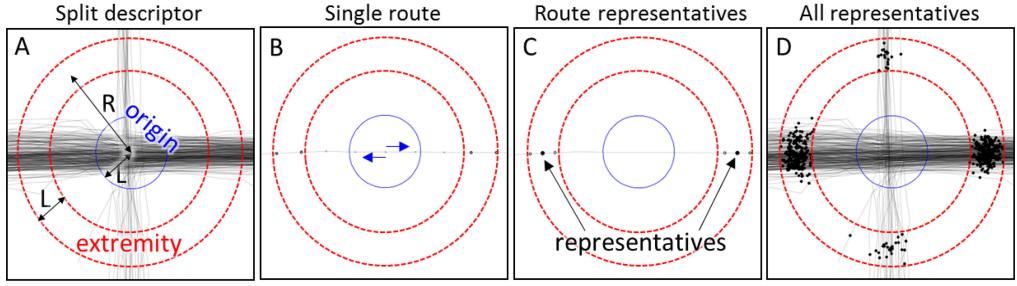From every route passing through, we select the points that are inside the extremity. Among the points inside the extremity, we select two representatives for each route by averaging the location of points inside the extremities, in each of the two directions (before and after the origin). Exceptions are routes that end inside the region, which pass through only once–or not at all if they also start in the same region (routes that contain no movement).

Averaging offers several benefits. First, it avoids problems caused by routes that traverse along the extremity, which could lead to false detection of a principal direction. Second, averaging reduces the amount of data to be processed by approximately 60%, which helps the next step (clustering). Third, we want each route to have equal impact in the calculations; otherwise, a route waiting at the location for an unusual amount of time would have too high an impact on the further analysis.

The representatives found by the descriptor are then clustered using the *random swap* algorithm (Fränti and Kivijärvi 2000); however, using repeated *k*-means might also suffice. To find the correct number of clusters, we cluster separately using two, three, and four clusters. The number of clusters that best models the data defines the number of directions. To detect the number of clusters, we use the maximum *silhouette coefficient* (*SC*) value according to the method of Rousseeuw and Kaufman (1990), which is the average value of all silhouettes belonging to every centroid:

$$s_x = \frac{b_x - a_x}{max\,\{a_x, b_x\}} \quad SC = \frac{1}{k} \sum_{i=1}^{k} s_i$$

Here, a_x is the average distance of centroid x to all other points in the same cluster, b_x is the minimum distance from x to the other clusters, and k is the number of clusters. The distance to the cluster is the average distance to all points within the cluster. The process is illustrated in Figure 11, which shows the cluster centroids, the corresponding partition and the silhouette coefficient. Here we consider only $k = 2$, 3 and 4 because our datasets do not include any crossroads with $k > 4$. In practice, it is enough to cluster using $k = 2$ and $k = 3$ clusters. If there is a crossing, the silhouette coefficient value is higher for any choice of $k > 2$ than it is for $k = 2$. In other words, the method is invariant to the actual number of roads in the intersection.

2.2 Select Intersections

After the splits are detected, we need to select a subset that captured all the intersections only once. It is possible that multiple split locations are found for an intersection, because the split descriptor detects any local maxima within the distance R from the intersection (Figure 12). The mean-shift
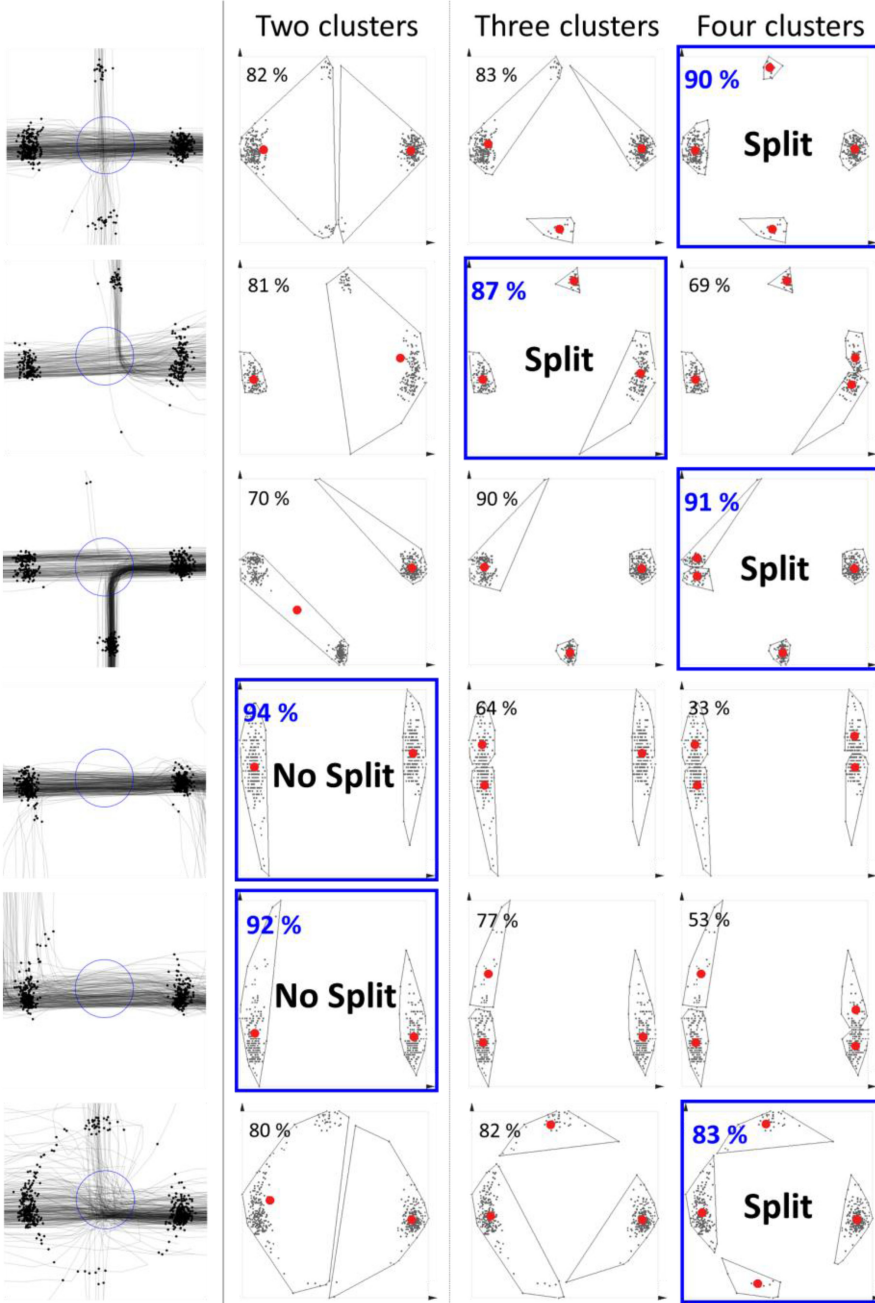
Fig. 11. Six locations investigated for splits. Each dataset is clustered by the random swap algorithm using two, three, and four clusters, respectively. The percentages represent the value of the silhouette coefficient. The occurrence of more than two clusters indicates a split.
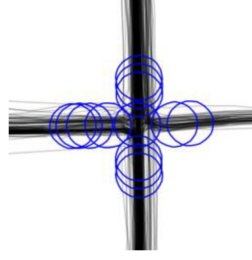
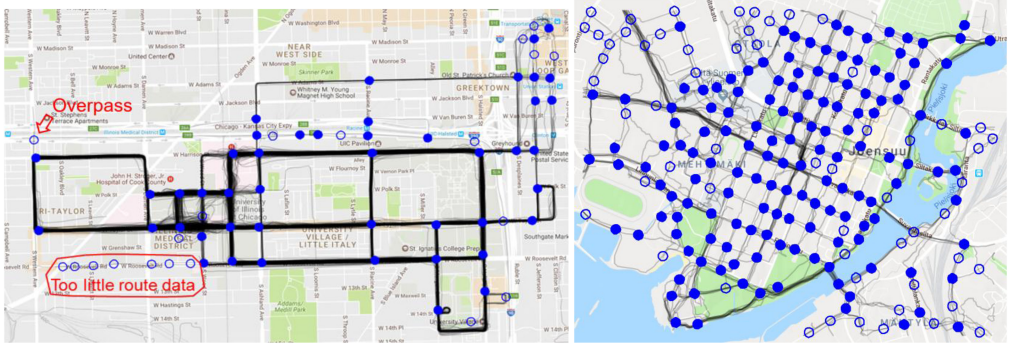Fig. 12. Multiple splits detected near the true intersection.

Fig. 13. The intersections found in Chicago (left) and Joensuu (right). The filled circles represent correct detections (true positives) and empty circles represent incorrectly detected intersections (false positives).

algorithm eliminates redundant points in parallel to the route but not along it. To remove the redundant points along the routes, we keep only candidates that have more routes passing through them than any neighbouring candidates within radius R.

The two steps are shown in Figure 13. The split detector correctly found the intersections but also several false positives. The selection step managed to remove most of these without losing any real intersection. The remaining false positives appear mainly in areas that displayed high GPS error or insufficient data (Figure 13). Many false positives are detected where only two routes run adjacent to each other. In such cases, the clustered dataset has only four points: two representatives for the two routes. This causes $SC = 1$ regardless of the point positions, because a_x is always 0 (one point in each cluster). Many of these false intersections, however, will not affect the structure of the network. Eventually, the highlighted intersections in Figure 13 will simply figure as components (points) of one long road segment. The current method will erroneously detect an intersection at an overpass. These situations may be successfully handled by analysing if routes turn to different clusters, however, if not enough data, this strategy causes many missed intersections: where routes do not turn.

3 CREATING ROADS

After the intersections have been found, we connect them. We examine each route in the dataset and link any two intersections it passes through in sequential order. To create the roads, we used the route segments.
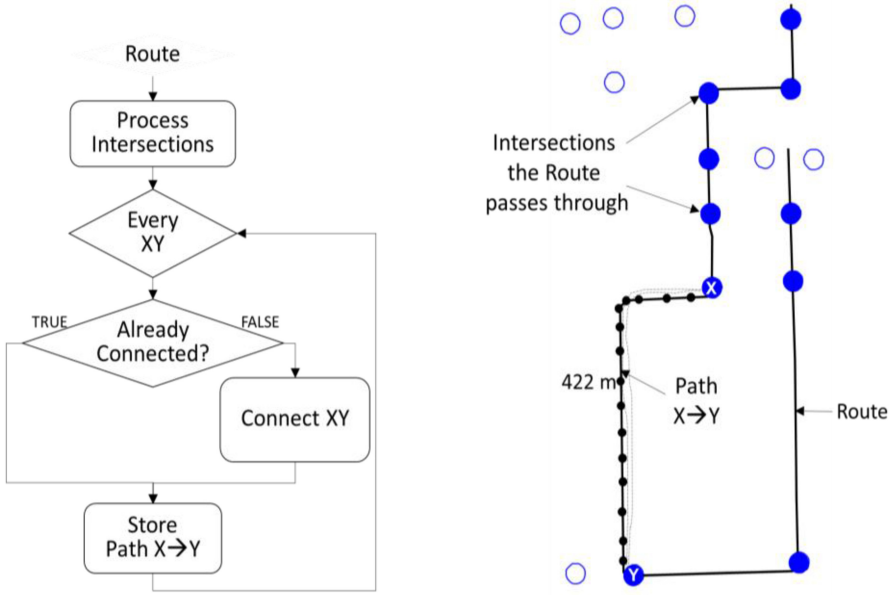
Fig. 14. Left panel: Algorithm for linking the intersections. Right panel: Example of a route passing through several intersections. Connections are formed between pairs of intersections in the order that the route passes through. For every connection, all paths are stored.

3.1 Connect Intersections

We analyse each route as shown in Figure 14. We first obtain the intersections that the route passes through and connected every subsequent pair. For each connection, paths are gradually collected from different routes to be used in the segment creation step described in Section 3.2.

3.2 Create Segments

To construct the road segments, we consider all paths between every two intersections. We choose the shortest path as an initial choice under the assumption that it has less GPS error. This strategy was proposed by Fathi and Krumm (2010) and seems to provide a good initial guess. However, if multiple paths exist, it is possible to find a better representative. In Figure 15, the grouped paths most likely indicate the correct road segment rather than the shortest path (shown in red). To create the segment, we first filter out paths that are not spatially similar to the initial choice; by so doing, we avoid paths that might have missed a third intersection. According to our experiments, such paths do more harm than good. The similarity function from Mariescu-Istodor and Fränti (2017) is used for this filtering:

$$S(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B{}^d| + |C_B \cap C_A{}^d|}{|C_A| + |C_B| - |C_A \cap C_B|},$$

where C_A and C_B are the cells that two paths A and B pass through, and $C_A{}^d$ and $C_B{}^d$ are the dilated cells obtained using the square structural element. Only paths that are 100% similar to the shortest path are accepted.

We compute the average for the similar paths using the method in Hautamäki et al. (2008), where the segment is iteratively improved using a strategy similar to k-means to optimise the dynamic time warping (DTW) distance. In Hautamäki et al. (2008), the medoid of the series is
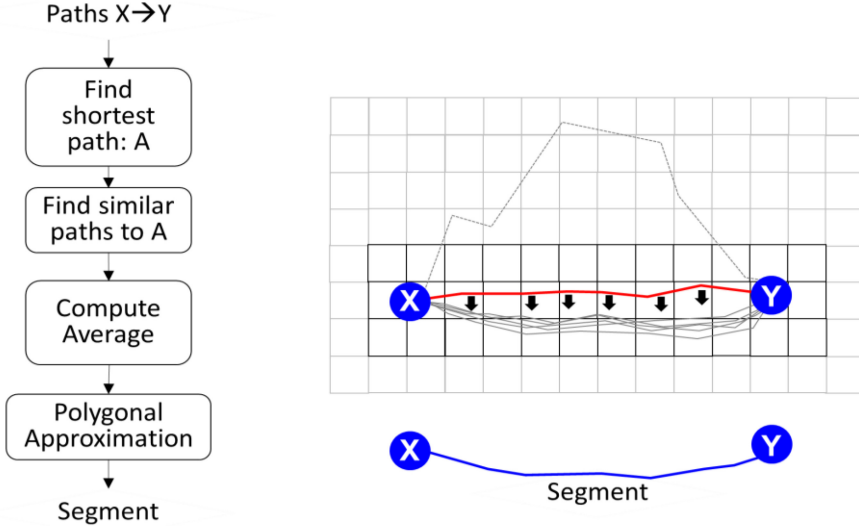
Fig. 15. Left panel: Algorithm for creating a segment. Right panel: Example where the initial guess is optimised using the similar paths. The dilated cells used by the similarity function are highlighted using the darker colour.

chosen as the initial representative. We have found that this initialisation does not improve the quality of the outcome and therefore we recommend keeping the shortest path as the initialisation. By not computing the medoid, the method is also much faster. We further sped the process up by applying the approximate FastDTW method (Salvador and Chan 2004), which works in linear time, rather than the typical DTW which has quadratic time complexity. Using these two modifications, the processing time reduces to about 1% of the original method. Alternative methods for averaging the paths, such as that of Schultz and Jain (2017), can also be used.

Often the generated segments are overly complex. For instance, a straight line might be represented by tens of points, whereas only two would suffice. Excessive points can produce an unnecessarily complex network. We reduced the number of points in the segments by applying polygonal approximation. We used the algorithm in Chen et al. (2012), but simpler variants such as that presented by Pikaz and Dinstein (1995) could also be used. We reduced the number of points to 30% without any loss in accuracy. In fact, accuracy became slightly better because some noise is filtered out in the approximation.

3.3 Filter Segments

A route might miss one or more intersections because of GPS error. In such cases, two intersections will become connected incorrectly. To handle this issue, Fathi and Krumm (2010) proposed the following strategy: Remove any road segment with length l, if there is another path with length less than $\sqrt{2}l$. The segment is removed in this situation because it probably misses one or more intersections owing to GPS error. This strategy is effective; however, in certain situations, it does not work as intended. Figure 16 shows two scenarios in which this strategy rejects the road segment, even though in the example on the left the segment should be kept.

To handle such problems, we present a filtering strategy based on spatial properties. For each segment, we first select all other segments that are contained in the same region. These segments are used to form a subgraph. If a path exists in this subgraph, the segment will be removed
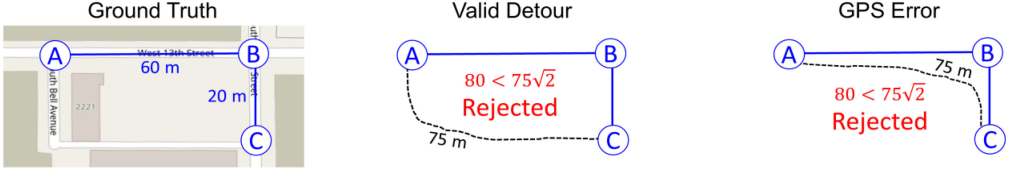
Fig. 16. Two examples where a segment is rejected according to the length rule. In the example on the left, the link should be kept because it represents a different road. On the right, the link should be removed because it is merely affected by GPS error.
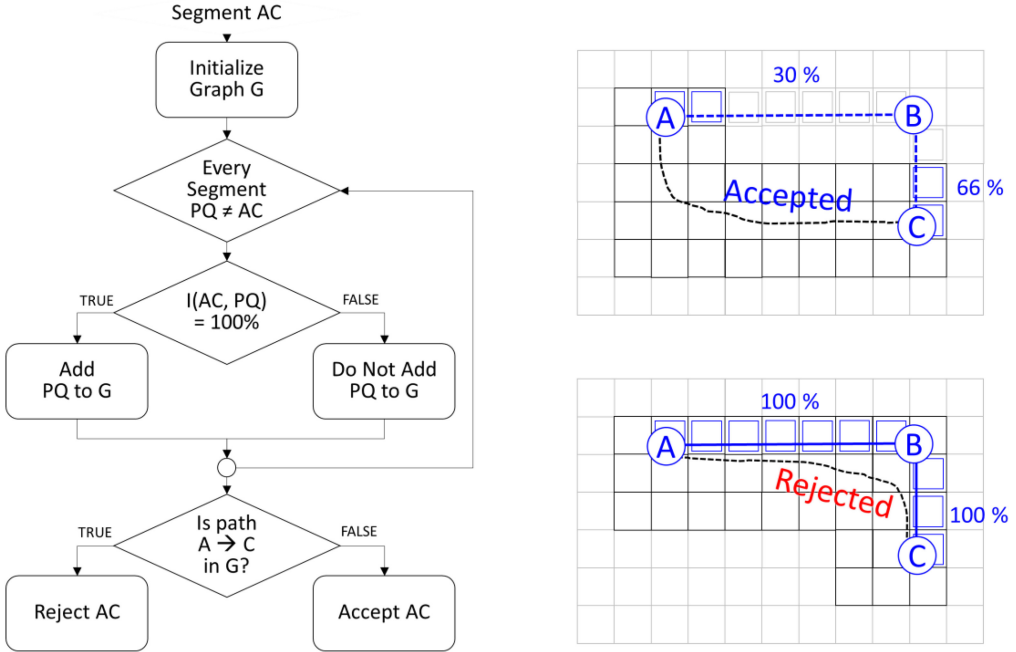
Fig. 17. Algorithm for filtering the segments (left panel). Examples where the segment is accepted (above, right) and rejected (bottom, right). The cell representations are shown. In the bottom right example, AB and BC are included in the region of AC and they form path A-B-C, which means the direct segment from A to C is redundant and rejected.

(Figure 17). We use the inclusion function from Mariescu-Istodor and Fränti (2017):

$$I(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B{}^d|}{|C_A|},$$

where A is a given segment and B is the segment to be tested if it is contained in A. The symbols C_A and C_B are cell representations of the two segments, and $C_B{}^d$ is the dilated cells of segment B.

4   EVALUATION

We evaluated the proposed method using two different datasets: Chicago and Joensuu,[1] shown in Table 1 and Figure 18. Both datasets are collected in areas of similar sizes, both including a clearly

---

Table 1. Datasets used in the experiments

Features	Chicago	Joensuu
Routes	889	108
Points	118,237	43,632
Total length (km)	2,869	250
Average speed (km/h)	33.1	9.1
Median sampling rate (s)	4	2
Intersections	52	228
Road segments	76	357
Points per segment (average)	6.6	4.8
Area (km^2)	4.5×2.5	2.8×2.2
Network length (km)	61	55
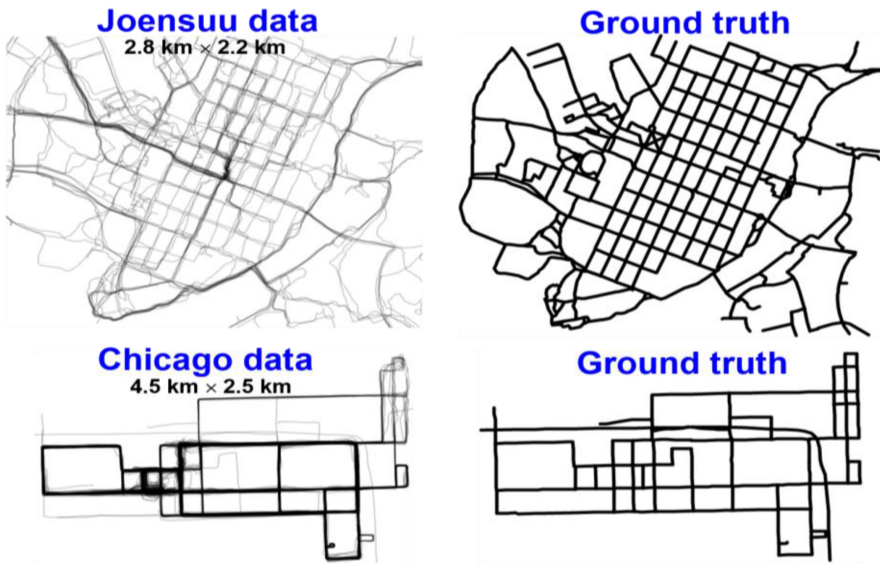
Fig. 18. Joensuu and Chicago datasets, and the corresponding ground truth.

visible city block structure. The Chicago dataset consists of regular campus bus routes with a high frequency of data points with limited spread on the streets. The Joensuu dataset consists of much fewer observations but with significantly wider spread across the streets in this small university town in Finland (76,000 inhabitants). There are only a few underpasses and traffic circles in these datasets.

The Chicago dataset is publically available (Biagioni and Eriksson 2012a) and contains 889 routes of the campus shuttles at the University of Illinois at Chicago, USA. The shuttles pass through main streets of the city. There are two areas that contain tall buildings which affect GPS precision. The second dataset contained tracks of a single user (Pasi) obtained from the Mopsi collection between 16.11.2014 and 25.4.2015. This collection included 102 routes in total, but we extracted only the 45 that are situated in the downtown of Joensuu, Finland, by cropping the data to a square region covering most of the downtown area. Joensuu contains straight perpendicular roads in the centre

Table 2. Variables Used and Values Obtained by CellNet for the Chicago and Joensuu Datasets

Symbol	Description		Chicago	Joensuu
N	Routes		889	108
p_r	Points per route	(average)	133	404
C	Cells		4,208	8,526
f	Routes per cell	(average)	91	5
S	Splits		368	2,118
X	Intersections		65	213
R	Road segments	(before filtering)	322	838
G	Paths per segment	(average)	37	3
p_h	Points per path	(average)	20	29
m	Mean-shift iterations	(average)	7.4	4.1
i	Time-series refining iterations	(average)	3.2	2.8
	Road segments	(after filtering)	102	349
	Points per segment		3.4	4

and more complex curvy roads at the borders; the latter are walking and cycling paths. The routes in Joensuu are collected while the user is jogging, usually along the sides of the streets.

We generated ground truth from OSM by querying all road segments in the respective areas of Joensuu and Chicago. We then manually excluded road segments that were not travelled in the data (Figure 18). In this way, it is theoretically possible to achieve 100% accuracy by a perfect algorithm. The Joensuu dataset had about four times as many intersections, and almost five times as many road segments, as the Chicago dataset. The number of points per segment did not differ significantly.

4.1 Processing Time

To obtain the time complexity of our method, we analysed each step using the variables shown in Table 2. The table contains values experimentally observed from both datasets. In the Joensuu dataset, the routes cover twice as large an area as Chicago's when counting the number of cells. The route density in Joensuu is lower: the average number of routes per cell is 5 compared with 91 in Chicago. The number of extracted segments per road is also lower, with 3 for Joensuu versus 37 for Chicago.

The time complexity of the split detection step depends on the size of the area covered by the routes, specifically the number of non-empty cells. For every cell, mean-shift is performed once and clustering three times, using the random swap algorithm with a fixed number of iterations (100) and a varying number of clusters (2, 3, and 4). Mean-shift requires $m \cdot f$ steps and clustering $100 \cdot (2+3+4) \cdot f$ steps. Total time complexity is O(Cmf). Overall, this step is one of two bottlenecks for the Chicago data and requires 37% of the total processing time.

Extracting the intersections depends on the number of splits found (S) in the previous step. Every split is compared against all others, leading to O(S^2) time complexity. However, even if the number of splits is not small (2,118 in Joensuu), it merely needs simple thresholding and can be processed rapidly. Overall, this step requires just a fraction of the total processing time (0.01% for Chicago and 0.2% for Joensuu).

Connecting the intersections depends on the number of routes and on the number of points in a route. Essentially, every point of every route must be processed. For every point, we check if an

Table 3. Time Complexity and Processing Time for Each Step of the Method

Step	Time complexity	Processing time (s)	
		Chicago	Joensuu
Detect splits	$O(Cmf)$	2,640	703
Select intersections	$O(S^2)$	0.8	8.3
Connect intersections	$O(Np_rf)$	116	64
Create segments	$O(RGp_hi)$	3,630	370
Filter segments	$O(R^2p_h)$	809	2,738
Total	$O(Cmf + S^2 + Np_rf + RGp_hi + R^2p_h)$	1.9 hours	1.1 hours

intersection was close (<L) by analysing the cell it resides in and all its adjacent cells. These take $O(Np_rf)$ time in total. This step requires about 2% of the total processing time.

Time complexity for the creation of the segments depends linearly on the number of splits (S), the number of points (p_h), and the number of iterations (i) in the path-averaging method. The total time complexity is $O(RGp_hi)$. Although none of the values is large, they accumulate, and this step constitutes the second bottleneck of the algorithm for the Chicago dataset–requiring 50% of the total processing time. The value of i remains small because the shortest segment is usually a good initialisation; only rarely substantially more iterations are needed.

Filtering the segments requires computing the inclusion value between all segment pairs, which requires $O(R^2p_h)$. This step is the bottleneck for the Joensuu dataset, which has significantly more segments than the Chicago dataset. Then, for every segment, we check if there existed a path linking the extremities in the subgraph. The subgraphs are small–fewer than five nodes–and any search strategy such as depth-first search or breadth-first search can be effectively applied. We use depth-first search. In total, this step requires 11% of the computation capacity for the Chicago dataset, and 71% for the Joensuu dataset. The time complexities and observed processing times are summarised in Table 3. Overall, the algorithm required about 1 hour for the Joensuu dataset and 2 hours for Chicago.

4.2 Quality Comparison

We next compare the CellNet method with four different approaches: two visual methods (1) (Davies et al. 2006) and (2) (Biagioni and Eriksson 2012b); a merging method (Cao and Krumm 2009); and a clustering method (Edelkamp and Schrödl 2003). The compared methods were all implemented by Biagioni and Eriksson (2012a). Visual outputs for all these methods and CellNet are shown in Figure 19, and a summary is provided in Table 4. The first visual method found too few segments from the Chicago dataset; that is, parts having too few data were missed. This did not happen to the same degree for the Joensuu data, because the route density there was more constant. The segments obtained by the visual method were very complex when looking at the number of points. The second visual method was able to handle the areas with less route data in Chicago. In Joensuu, however, due to lower density of routes the method did not perform as well.

The clustering method found too many intersections and spurious road segments, especially in regions with high GPS error. The merging method also found too many intersections and segments. In Joensuu, it produced a disconnected map because some regions have too little route data. The number of points per segment is small for both the clustering and merging methods; however, the complexity of the overall network remains high owing to many spurious segments. Among the methods compared, the results from CellNet matched closest to the ground truth and the number of
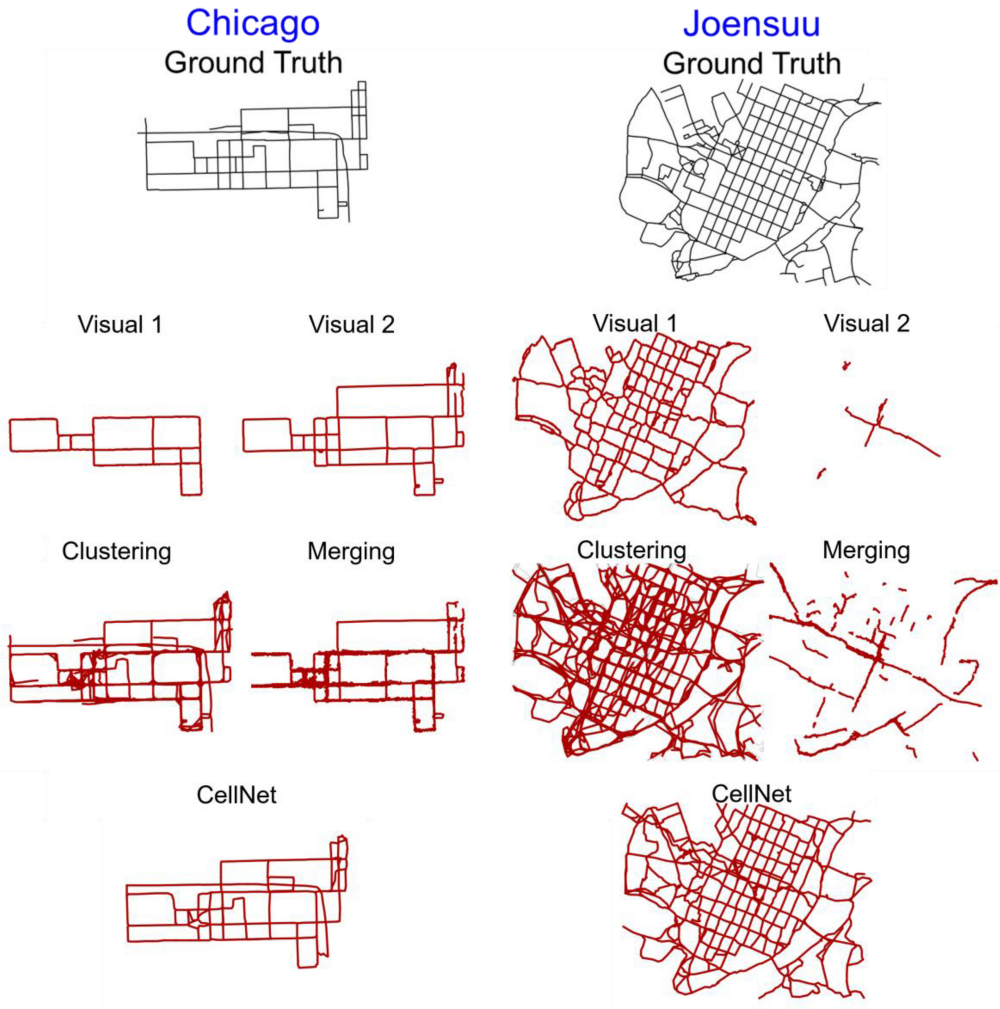
Fig. 19. Visual output of the five methods for the Chicago and Joensuu datasets. We provide an interactive evaluation of the methods at the following address: http://cs.uef.fi/mopsi/routes/network.

Table 4. The number of Intersections and Segments Obtained by Various Methods

	Chicago					
Features	**Visual 1**	**Visual 2**	**Clustering**	**Merging**	**CellNet**	**Ground Truth**
Intersections	16	38	363	916	65	52
Segments	24	66	831	1,859	102	76
Points per segment (average)	54	5.7	2.5	2.5	3.4	6.6
	Joensuu					
Features	**Visual 1**	**Visual 2**	**Clustering**	**Merging**	**CellNet**	**Ground Truth**
Intersections	278	6	844	558	213	228
Roads	420	15	1,551	1,154	349	357
Points per segment (average)	11.2	17.5	3.5	5.3	4	4.8

Table 5. Accuracy of the Intersection Detection by the Five Methods

Method	Chicago			Joensuu		
	Precision	Recall	F-score	Precision	Recall	F-score
Visual 1	97%	27%	42%	54%	63%	58%
Visual 2	86%	63%	73%	100%	1%	2%
Clustering	14%	94%	24%	42%	76%	54%
Merging	5%	90%	10%	22%	52%	31%
CellNet	77%	90%	84%	71%	68%	69%

points used to represent the segments was optimised better. In fact, this number was smaller than the ground truth, indicating that even the ground truth itself (OSM) could be optimised further.

We next evaluate how well the algorithms performed at finding the intersections. Both the detected and the ground truth intersections are geographic locations (latitude, longitude). To compare the correctness of the extracted locations, we perform a nearest-neighbour search from each detected intersection to its nearest one in the ground truth. Then we count how many real intersections were missed, similarly as was done with cluster centroids in Fränti et al. (2014). The number of these *orphan* intersections counts as missed (*false negatives*). The process is then repeated in the other direction: from ground truth to detected intersections. The unmapped intersections count as false detection (*false positives*)–that is, a detected segment that does not have a match in the ground truth. Using these values, we calculate three measures:

$$precision = \frac{correct}{correct \ + false\,detected}.$$

$$recall = \frac{correct}{correct + missed}$$

$$f-score = 2 \cdot \frac{precision \cdot recall}{precision + recall}.$$

Although some of the methods do not specifically detect intersections, intersections do exist where two or more road segments connect. It is therefore possible to evaluate them. The results are summarised in Table 5. The visual methods have the highest precision for the Chicago dataset. This is partly because they detect fewer intersections (i.e., the methods avoid false detections), and partly because the routes have high density in the region, which allow the visual-based methods to work more accurately. However, the recall is low because many intersections are missed. The second visual method performs better because unlike Visual 1, it uses multiple thresholds, which enables it to generate the network where less data is available. The clustering and merging methods have high recall, because–unlike the visual methods–they do not intentionally drop out parts of the dataset. However, the precision is low because they detect too many intersections in regions with many routes and low GPS accuracy. Our method is the most balanced in terms of precision and recall, and it has the highest F-score values.

We next introduce a novel approach to evaluate the correctness of the road segments. First, we obtained all the segments from the ground truth and convert them into cells. Then we create a second set from the extracted segments. To evaluate the success of a method, we calculate the difference between the two sets. If the generated network is flawless, the difference is an empty set (all cells have frequency 0). Otherwise, some cells will have a positive frequency (missed segments) and other cells will have a negative frequency (false segments). Cells with 0 frequency are the desired result (correct detection), as shown in Figure 20. We compute precision, recall and F-score.
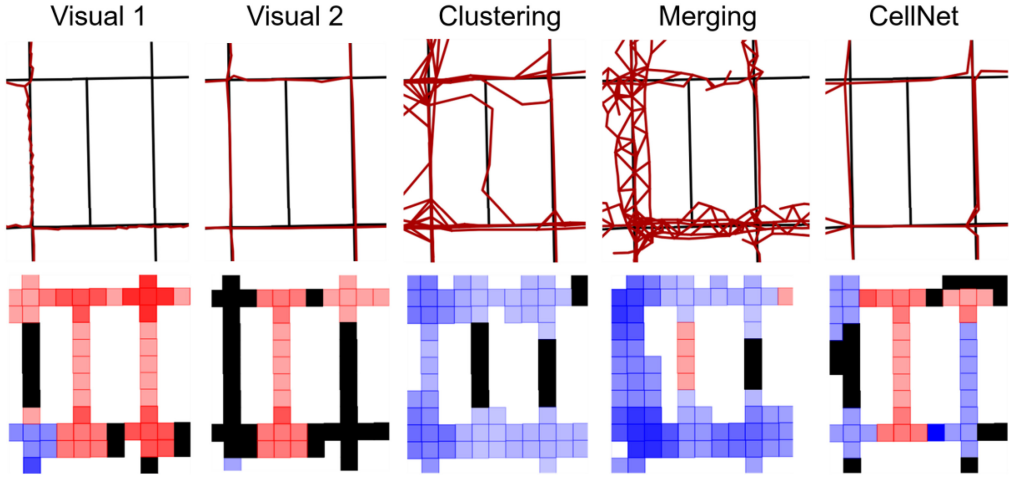
Fig. 20. Ground truth segments (black) and extracted segments (red) are shown at the top, and the corresponding cell frequency differences are shown at the bottom. Blue cells represent false detections (negative frequency), and red cells missed segments (positive frequency). Black cells have 0 frequency. The colour intensity is proportional to the frequency.

Table 6. Accuracy of the Road Detection by the Five Measures

Method	Reference	Chicago			Joensuu		
		Precision	Recall	F-score	Precision	Recall	F-score
Visual 1	Davies et al. (2006)	97%	27%	42%	56%	38%	46%
Visual 2	Biagioni and Eriksson (2012b)	92%	55%	69%	82%	0.6%	1%
Clustering	Edelkamp and Schrödl (2003)	17%	94%	28%	24%	87%	38%
Merging	Cao and Krumm (2009)	7%	70%	10%	13%	33%	19%
CellNet	Proposed	92%	83%	87%	68%	49%	58%

Table 6 summarises the results for the five methods when finding the road segments. Similar observations can be made as in the intersection evaluation. The visual methods achieved the highest precision but have the lowest recall, whereas clustering and merging display high recall but low precision. In the noisy regions, the clustering and merging methods produce many spurious segments, as shown in Figure 20.

4.3 Discussion of the Parameter Setup

The four compared methods were implemented by Biagioni and Eriksson (2012a), who closely follow the descriptions in their respective papers, except for the clustering method (Edelkamp and Schrödl 2003). Biagioni and Eriksson (2012a) did not implement the intersection refinement process for the clustering method. The first visual method (Davies et al. 2006) uses three parameters: cell size, density threshold, and kernel bandwidth. The second visual method (Biagioni and Eriksson 2012b) uses two main parameters: cell size and density threshold. The clustering method has three parameters: cluster seed interval, intracluster bearing difference, and intracluster distance. The merging method (Cao and Krumm 2009) has three parameters: edge volume, location distance limit, and location bearing difference. The merging method uses several other parameters in the route clarification step; however, this step is separate from the method itself

Table 7. Parameters Used by the Different Methods

Method	Parameter	Chicago	Joensuu
Visual 1	cell size	2	2
(Davies et al. 2006)	density threshold	100	3
	kernel bandwidth	17	15
Visual 2	cell size	1	2
(Biagioni and Eriksson 2012b)	density threshold	50	20
Clustering	cluster seed interval	50	70
(Edelkamp and Schrödl 2003)	intracluster bearing difference	45	45
	intracluster distance	20	22
Merging	edge volume	3	2
(Cao and Krumm 2009)	location distance limit	20	25
	location bearing difference	45	45
CellNet	origin radius (L)	30	24
(Proposed)	distance to extremity (R)	100	80

Note: Optimised values are shown for Chicago and Joensuu.

Table 8. Results When Using the Parameters from Chicago Dataset on the Joensuu Dataset

Method	References	Chicago parameters		Optimised parameters	
		Intersections	Segments	Intersections	Segments
Visual 1	Davies et al. (2006)	-	-	58%	46%
Visual 2	Biagioni and Eriksson (2012b)	-	-	2%	1%
Clustering	Edelkamp and Schrödl (2003)	46%	35%	54%	38%
Merging	Cao and Krumm (2009)	27%	9%	31%	19%
CellNet	Proposed	63%	57%	69%	58%

and is not presented here. All methods also have a fourth parameter, namely the number of routes to be used. We disregarded this parameter because it is essentially a sub-sampling of the dataset, which can be performed as a separate pre-processing step if the dataset is excessively large.

We optimised the parameters of the methods using a trial-and-error approach and the observations of Biagioni and Eriksson (2012a) and Ahmed et al. (2015). It is possible that better quality can be achieved; however, the optimisation task is tedious and time consuming. For CellNet, we optimised the two parameters by grid search using the Chicago dataset in the scale L in (20, 40) and R in (50, 150). The results showed only slight variations: the lowest F-score achieved in these ranges was only slightly worse than the highest achieved score (highest, 84%; lowest, 75%). Optimised parameter values for the two datasets are shown in Table 7.

To evaluate the importance of optimizing the parameters, we tried to use the values optimised for the Chicago dataset on the Joensuu dataset directly (Table 8). The visual methods (Davies et al. 2006; Biagioni and Eriksson 2012b) crashed because the density threshold was too high to produce any contours. The clustering method (Edelkamp and Schrödl 2003) worked fairly well. The merging method (Cao and Krumm 2009) produced a low F-score. CellNet produced the highest F-scores. By optimising the Joensuu data, the visual method produced the second-best result. The clustering method improved the intersection aspect by 17% and the segment aspect by 6%, and the merging method improved intersections by 15% and segments by 111%. CellNet did not improve by much, at 9% for intersections and 4% for segments; however, this method already produces

Table 9. Running Times and the Resulting Size of the Networks
(Number of Points of All Detected Roads)

Method	Running time		Network size	
	Chicago	Joensuu	Chicago	Joensuu
Visual 1	15 min	14 min	1,309	4,752
Visual 2	30 min	15 min	381	80
Clustering	54 min	15 min	2,119	5,366
Merging	2.5 days	3 h	4,749	6,097
CellNet	1.9 h	1.1 h	331	1,215

good results before optimisation–even better than other methods after optimisation. This finding suggests that parameter optimisation is not required by CellNet, which is expected to work with the recommended values ($L = 25$, $R = 80$).

4.4 Speed and Space Requirements

The visual methods are computationally faster than the other methods because the data usually contain many overlapping routes, which are processed jointly. The drawback is that the direction of travel is lost in the image representation and must be handled separately. Visual methods also perform poorly if the density of the routes varies inside the dataset, as demonstrated by Biagioni and Eriksson (2012a). The route-merging method suffers in the presence of high GPS noise. It is also much slower on the Chicago dataset because it has high frequency of points per street. The reason is that the merging approach processes every trajectory individually regardless their location. The other methods manage this issue more efficiently.

CellNet running time is moderate. The time complexity of the method is slow when a dataset has high route density or the number of roads is high. Processing times are shown in Table 9; however, they can vary substantially when parameters are changed. The times are shown for the optimised values. The memory requirements are also shown in Table 9. Because of the point-reduction step, the size of the network produced by CellNet is at most 25% of the networks produced by the other methods. The visual method uses too many points to describe the roads; this artefact is evident in Figure 20. The clustering and merging methods produced many spurious roads.

5 CONCLUSIONS

We present a new road network inference method, called CellNet, which works well on different route datasets. It produced higher accuracy (F-scores) than conceptually distinct methods when tested on two different real route datasets. The memory requirements of the resulting networks are considerably smaller–roughly 25%–compared with the size of networks generated by the other methods we tested. The speed is only mid-range. Perhaps a more efficient algorithm could be used to improve the segment optimisation step.

Two more recent methods were left out from the comparisons because both implementations have certain technical difficulties. Visual comparison is therefore provided in Figure 21 showing that these methods appear to be close to CellNet. However, the main benefit of CellNet is that it almost completely avoids the time-consuming parameter optimisations. The same setup ($L = 30$, $R = 100$) worked well for both Joensuu and Chicago data, whereas all other methods require significantly different parameter setup.

CellNet is designed for pedestrian users in small towns like Joensuu. However, it is also expected to be able to handle more complex road structures like roundabouts, overpasses, and intersections
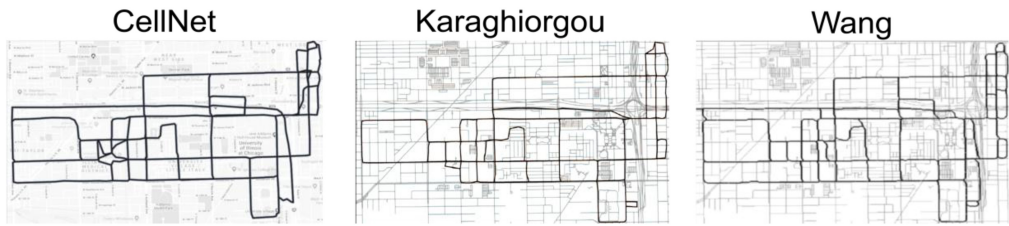
Fig. 21. Visual comparison of two other methods: (Karagiorgou and Pfoser 2012; Wang et al 2015).

with more than four exits. It would simply detect an intersection at a roundabout or multiple intersections if the exits were R meters apart and one intersection otherwise. The method would also ignore overpass intersections, as no trajectory would make a turn there. A limitation is that, with small amount of data, normal crossroads might be missed if no user made a turn at a particular location.

The method might also be improved by filtering the input data before processing to remove GPS noise. For instance, in Mariescu-Istodor et al. (2018), impulsive types of noise were removed by using four simple rules. However, noise around tall buildings can have systematic bias and should be studied further.

REFERENCES

Mahmuda Ahmed, Sophia Karagiorgou, David Pfoser, and Carola Wenk. 2015. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica* 19, 3, 601–632.

Arpad Barsi and Christian Heipke. 2003. Artificial neural networks for the detection of road junctions in aerial images. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences* 34(3/W8), 113–118.

James Biagioni and Jakob Eriksson. 2012a. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *Transportation Research Record: Journal of the Transportation Research Board* (2291), 61–71.

James Biagioni and Jakob Eriksson. 2012b. Map inference in the face of noise and disparity. In *ACM SIGSPATIAL*. 79–88.

Lili Cao and John Krumm. 2009. From GPS traces to a routable road map. In *ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 3–12.

Chen Chen and Yinhang Cheng. 2008. Roads digital map generation with multi-track GPS data. In *IEEE International Workshop on Education Technology and Training, 2008 and 2008 International Workshop on Geoscience and Remote Sensing* 1, 508–511.

Minjie Chen, Mantao Xu, and Pasi Fränti. 2012. A fast O(N) multi-resolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Transactions on Image Processing* 21, 5, 2770–2785.

Yizong Cheng. 1995. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17, 8, 790–799.

Jonathan Davies, Alastair R. Beresford, and Andy Hopper. 2006. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing* 5, 4, 47–54.

Stefan Edelkamp and Stefan Schrödl. 2003. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*. 128–151.

Alireza Fathi and John Krumm. 2010. Detecting road intersections from GPS traces. In *Proceedings of the 6th International Conference on Geographic Information Science*. 56–69.

Pasi Fränti and Juha Kivijärvi. 2000. Randomised local search algorithm for the clustering problem. *Pattern Analysis & Applications* 3, 4, 358–369.

Pasi Fränti, Mohammad Rezaei, and Qinpei Zhao. 2014. Centroid index: Cluster level similarity measure. *Pattern Recognition* 47, 9, 3034–3045.

Ville Hautamäki, Pekka Nykänen, and Pasi Fränti. 2008. Time-series clustering by approximate prototypes. In *IAPR International Conference on Pattern Recognition*. 1–4.

Jiuxiang Hu, Anshuman Razdan, John C. Femiani, Ming Cui, and Peter Wonka. 2007. Road network extraction and intersection detection from aerial images by tracking road footprints. *IEEE Transactions on Geoscience and Remote Sensing* 45, 12, 4144–4157.

Sophia Karagiorgou and David Pfoser. 2012. On vehicle tracking data-based road network generation. In *ACM SIGSPATIAL*. 89–98.

Sophia Karagiorgou, Dieter Pfoser, and Dimitrios Skoutas. 2013. Segmentation-based road network construction. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 460–463.

Piyawan Kasemsuppakorn and Hassan A. Karimi. 2013. A pedestrian network construction algorithm based on multiple GPS traces. *Transportation Research Part C: Emerging Technologies* 26, 285–300.

M. P. Khanna. 1999. *Introduction to Particle Physics*. PHI Learning Pvt. Ltd.

Radu Mariescu-Istodor and Pasi Fränti. 2017. Grid-based method for GPS route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems* 3, 3, 8.

Radu Mariescu-Istodor, Por Heng, and Pasi Fränti. 2018. Roundness measure for GPS routes. *International Conference on Location Based Services (LBS), Adjunct proceedings*. 81–86.

Brian Niehöfer, Andreas Lewandowski, Ralf Burda, Christian Wietfeld, Franziskus Bauer, and Oliver Lüert. 2010. Community map generation based on trace-collection for GNSS outdoor and RF-based indoor localization applications. *International Journal on Advances in Intelligent Systems* 2, 4.

Arie Pikaz and Its'hak Dinstein. 1995. Optimal polygonal approximation of digital curves. *Pattern recognition* 28, 3, 373–379.

Peter J. Rousseeuw and L. Kaufman. 1990. *Finding Groups in Data*. Wiley Online Library.

Stan Salvador and Philip Chan. 2004. FastDTW: Toward accurate dynamic time warping in linear time and space. In *ACM International Conference on Knowledge Discovery and Data Mining Workshop on Mining Temporal and Sequential Data*. 70–80.

Stefan Schroedl, Kiri Wagstaff, Seth Rogers, Pat Langley, and Christopher Wilson. 2004. Mining GPS traces for map refinement. *Data Mining and Knowledge Discovery* 9, 1, 59–87.

David Schultz and Brijnesh Jain. 2017. Nonsmooth analysis and subgradient methods for averaging in dynamic time warping spaces. *Pattern Recognition* 74, 340–358.

Mohamad Tavakoli and Azriel Rosenfeld. 1982. Building and road extraction from aerial photographs. *IEEE Transactions on Systems, Man, and Cybernetics* 12, 84–91.

Suyi Wang, Yusu Wang, and Yanjie Li. 2015. Efficient map reconstruction and augmentation via topological methods. In *ACM SIGSPATIAL*. 25.