CellNet: Inferring road networks from GPS trajectories

RADU MARIESCU-ISTODOR, University of Eastern Finland
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 periodical updates to provide reliable information. We propose an automatic method to generate the road network using a GPS trajectory dataset. The method, titled 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 three conceptually different state-of-the-art alternatives. The results show that CellNet provides better accuracy and is less sensitive to parameter setup. The generated road network occupies only 25% of the memory required for the networks produced by other methods.

• **Information systems→Information systems applications • Information systems→Information retrieval**.

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 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 utilized 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).

Figure 1. Aerial images of a city area (left panel) and countryside region (right panel).

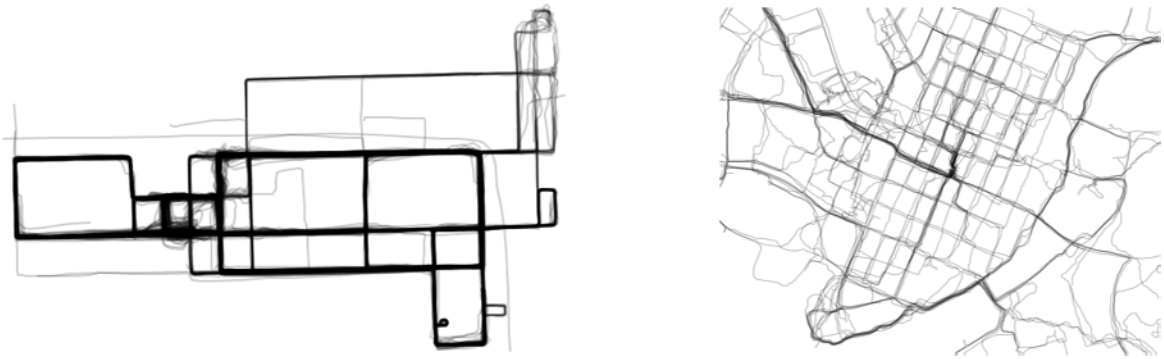

Figure 2. GPS routes in Chicago, USA (left panel) and Joensuu, Finland (right panel).

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] also use routes to form a binary image, which is then blurred and a density threshold is applied to filter out parts that contain too few routes. 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.

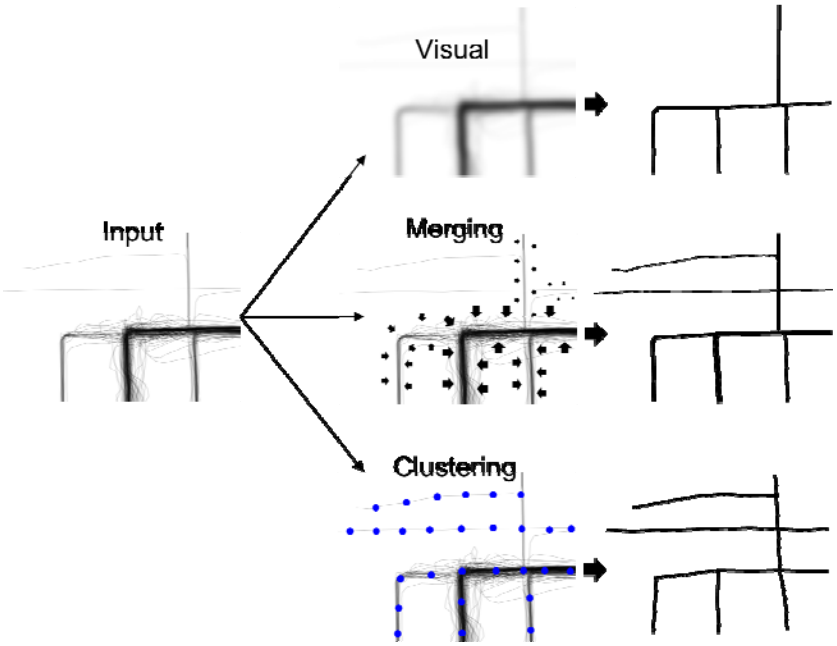

Figure 3. Three conceptually different road network generation techniques: visual, merging and clustering.

Route merging methods [Niehoefer 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 Schrödl 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 generating 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 using the routes.

In this paper 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 optimize the network to avoid redundant and overly complex roads.

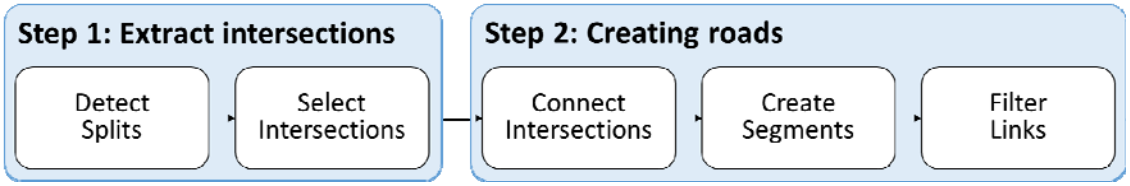

Figure 4. The steps performed by CellNet to infer a road network.

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

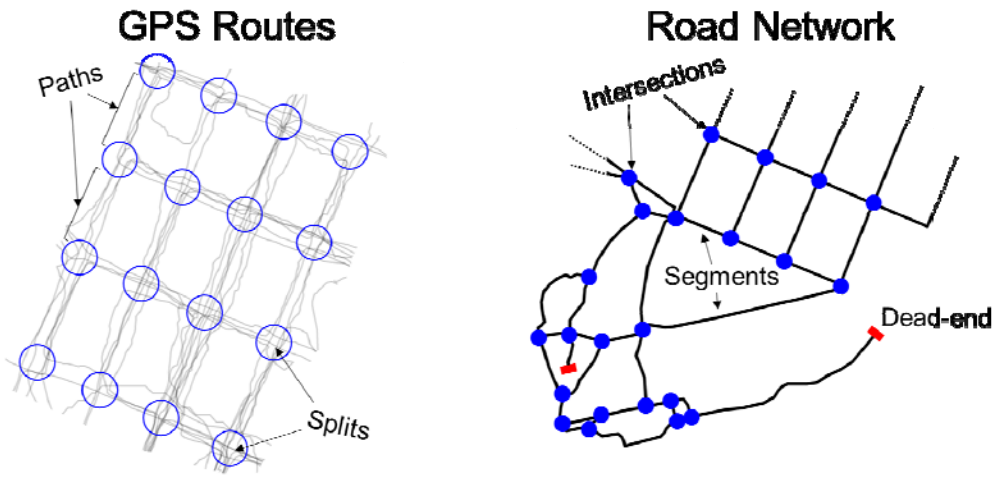

Figure 5. Diagram showing terminology used to discuss GPS routes and road networks.

2. EXTRACT INTERSECTIONS

Intersections are places in which more than two roads connect. To detect potential intersections from GPS routes, we applied the two processes shown in Figure 6. First we analyses 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 measured the frequency of routes passing through the area. Splits having a higher frequency than their neighbours (local maxima) were selected as intersections.

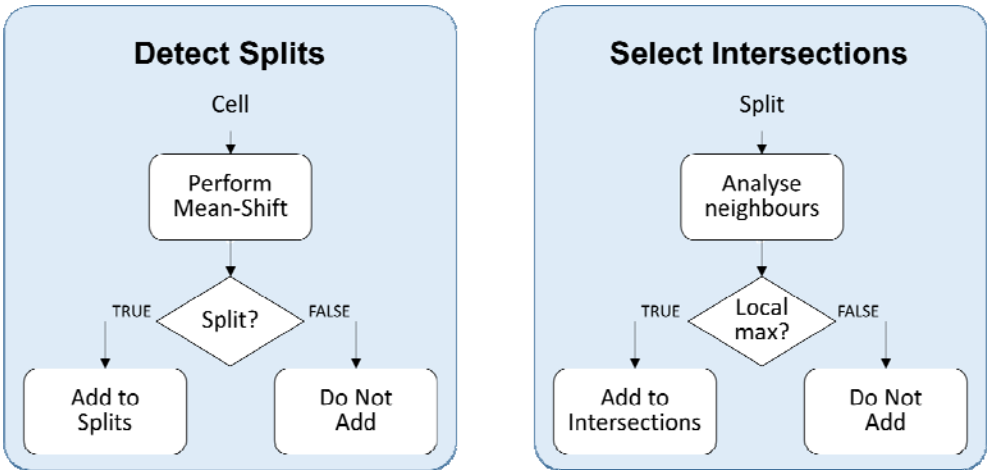

Figure 6. Steps performed to detect splits (left panel) and to select local maxima (right panel).

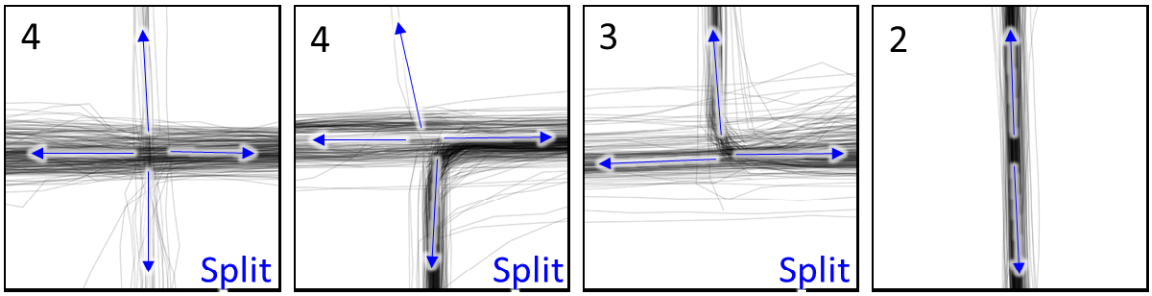

Figure 7. Four examples of locations at which routes head off into several principal directions. The directions are highlighted by arrows. The first three examples are splits, according to our definition, whereas the last is not. The numbers (upper left corners) indicate the quantity of principal directions.

2.1 Detect Splits

To detect the splits, we analysed all locations through which the routes passed. To do this efficiently we divided the space by a grid with cell length L = 25 m (recommended). For every grid cell, we maintained information containing the cell's location, indexes of all routes passing through it and the total number of routes. We accumulated 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 used interpolation to handle this problem. A more detailed explanation on the use of the grid is given in Mariescu and Fränti [2017].

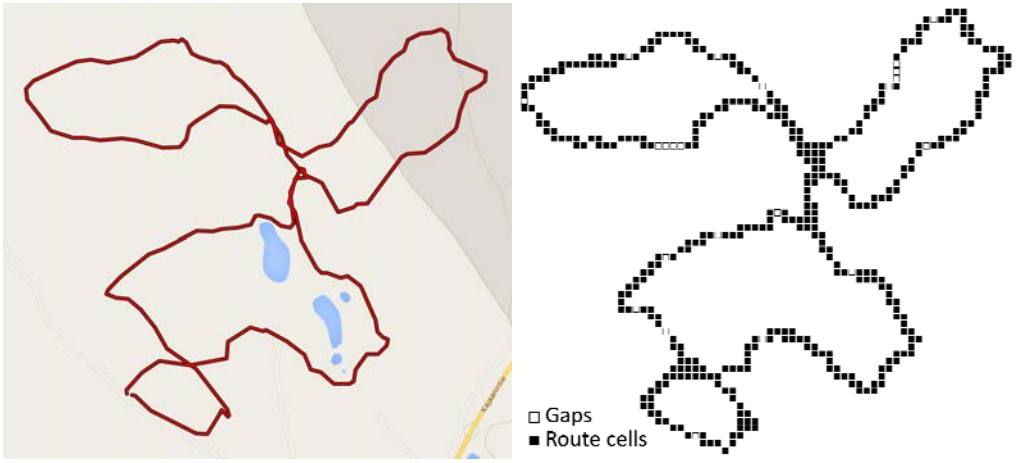

Figure 8. A sample route (top panel) and the cell representation with cell size 25 m × 25 m (lower panel). The gaps are filled using linear interpolation.

After collecting the information, we processed 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 was as follows. We first transferred 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 is repeated until the location

stabilizes. 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.

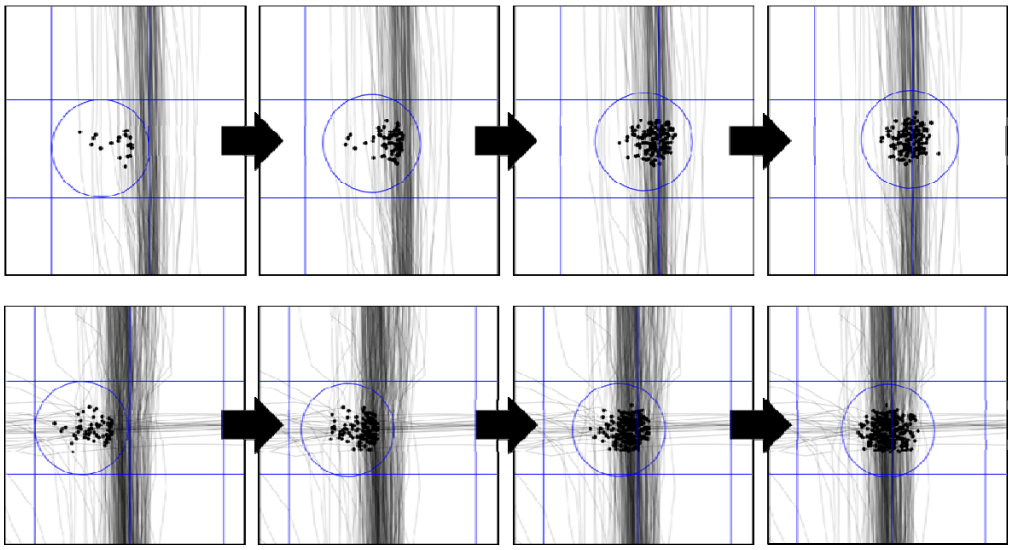

Figure 9. Two examples of the mean-shift algorithm. The initial location gradually moves towards the centre of the routes. If an intersection is nearby, the location is likely to end up at its centre.

After the location had been tuned, we analysed the neighbourhood to detect the principal directions of movements. For this purpose we defined 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.

From every route passing through, we selected the points that were inside the extremity. Among the points inside the extremity we selected 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 were 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).

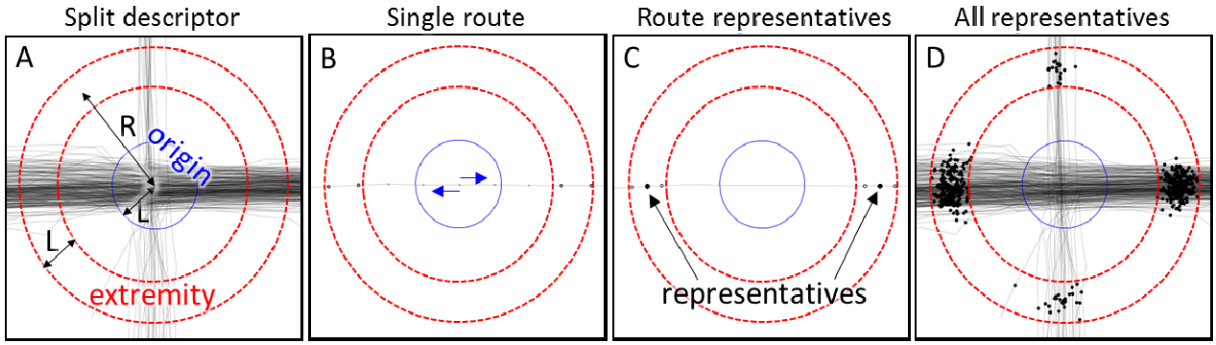

Figure 10. A, the split descriptor composed of the origin and the extremity. B, a sample route traversing through the point of interest; points inside the extremity are highlighted. C, the points inside the extremity are averaged in each of the two directions to create the representatives. D, representatives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

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 wanted 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.

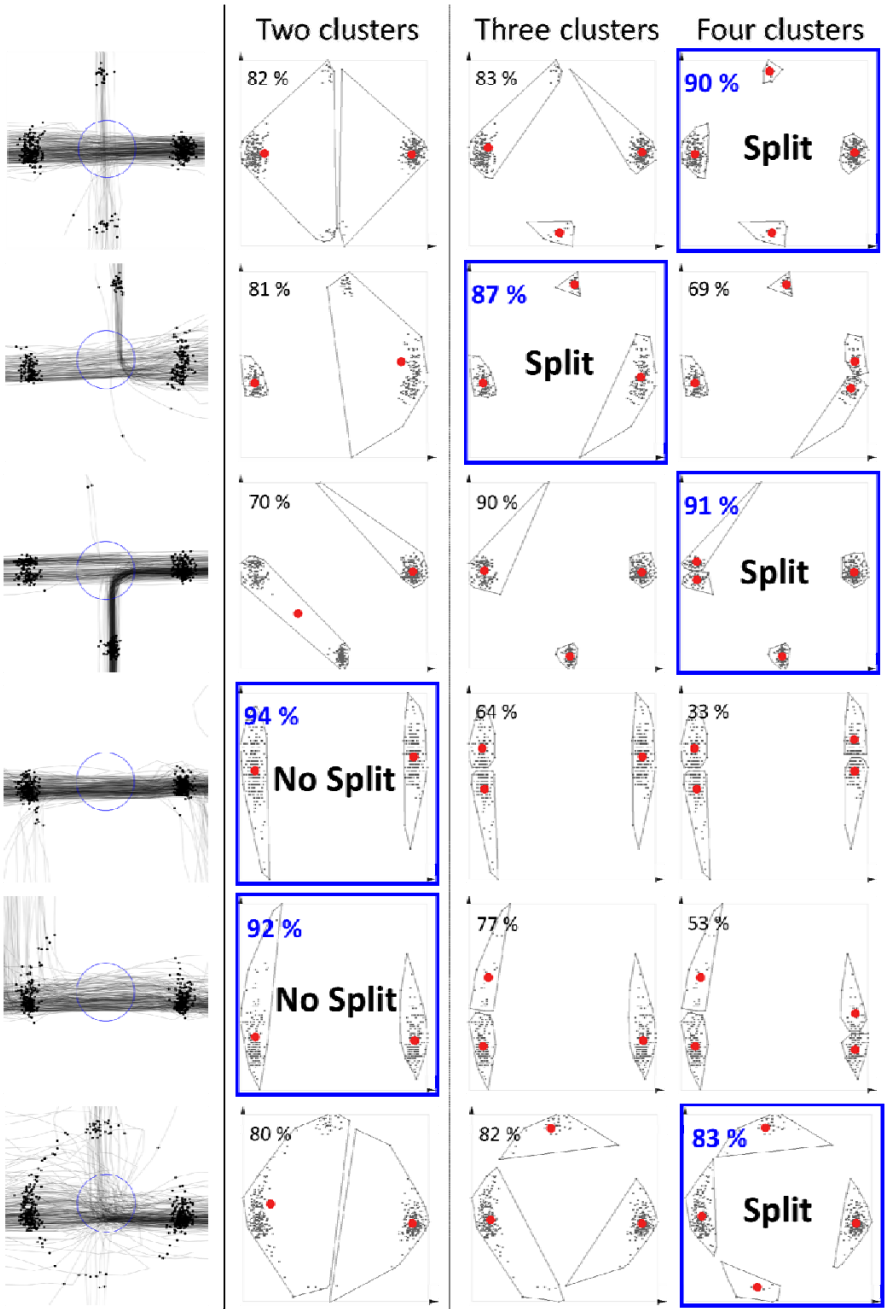

Figure 11. Six locations investigated for splits. Each dataset is clustered by the random swap algorithm using 2, 3 and 4 clusters respectively. The percentages represent the value of the silhouette coefficient. The occurrence of more than 2 clusters indicates a split.

The representatives found by the descriptor were 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 clustered 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 used 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:

$$s_x = \frac{b_x - a_x}{\max\{a_x, b_x\}}$$

$$SC = \frac{1}{k}\sum_{t=1}^{k} s_t$$

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. In practice, it is enough to cluster using two and three clusters. If there is a crossing, the silhouette coefficient value is higher both for $k = 3$ and $k = 4$ than it is for $k = 2$.

2.2 Select Intersections

After the splits were detected, we needed 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 algorithm eliminates redundant points in parallel to the route but not along it. To remove the redundant points along the routes, we kept only candidates that had more routes passing through them than any neighbouring candidates within radius R.

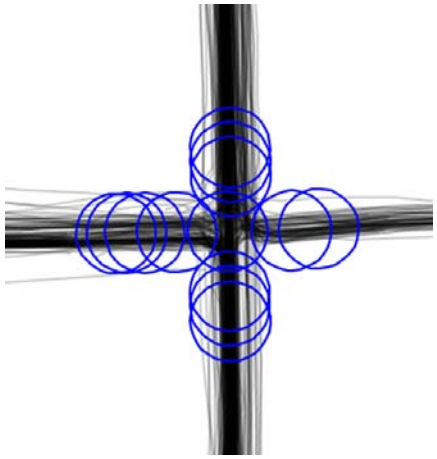

Figure 12. Multiple splits detected near the true intersection.

The two steps are shown in Figure 13, using the Chicago dataset[1] as an example. The split detector correctly found the intersections but also found several false positives. The selection step managed to remove most of these without losing any real intersection. The remaining false positives appeared mainly in areas that displayed high GPS error or insufficient data (Figure 13). Many false positives were detected

[1] http://cs.uef.fi/mopsi/routes/network

in areas in which only two routes ran 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). Because of this deficiency, we recommend that a dataset is checked to ensure that more than two routes exist in every region. However, this criterion should be a prerequisite for any road network inference method, because single observations can be the result of GPS error.

In Figure 13, the false positives in the region with too little route data did not affect the structure of the resulting network. After the road creation step, they resulted in a single long road.

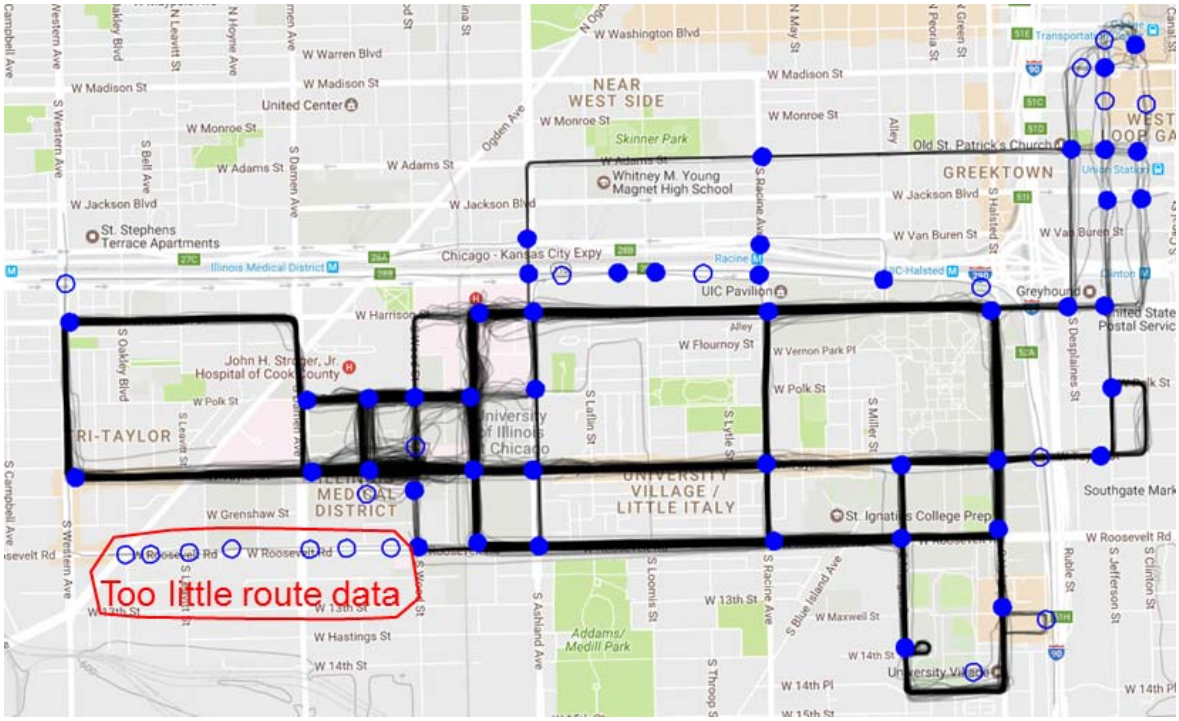

Figure 13. The intersections found in Chicago dataset. The filled circles represent correct detections (true positives) and empty circles represent incorrectly detected intersections (false positives).

3. CREATING ROADS

After the intersections had been found, we connected them. We examined each route in the dataset and linked any two intersections it passed through in sequential order. To create the roads, we used the route segments.

3.1 Connect Intersections

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

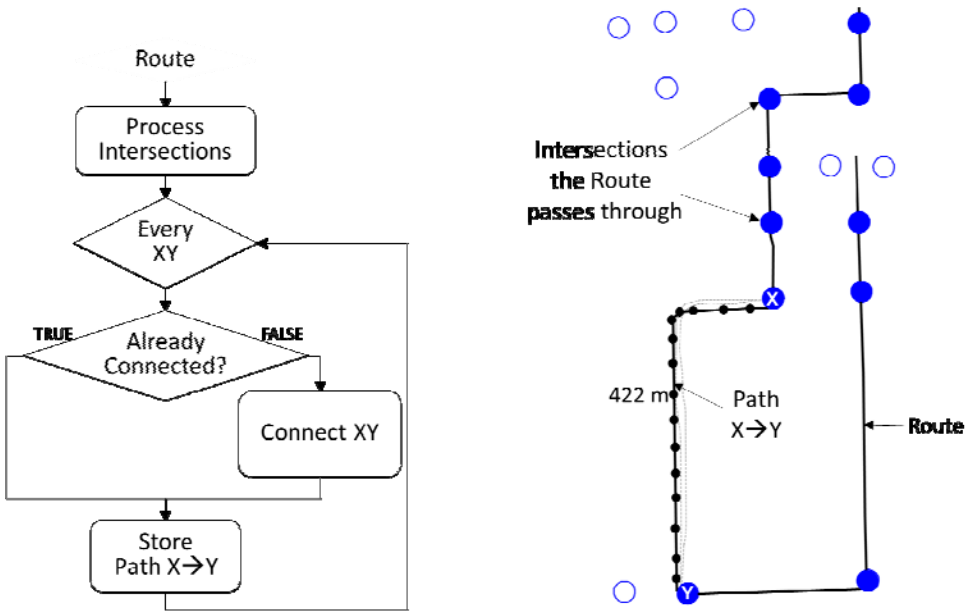

Figure 14. Left panel: Algorithm for linking the intersections. Right panel: Example of a route passing through several intersections. Connections are formed between pairs of intersections in the order that the route passes through. For every connection, all paths are stored.

3.2 Create Segments

To construct the road segments, we considered all paths between every two intersections. We chose 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 filtered out paths that were not spatially similar to the initial choice; by so doing we avoided paths that might have missed a third intersection. According to our experiments, such paths do more harm than good. The similarity function from Mariescu and Fränti [2017] was 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 computed 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 optimize the dynamic time warping (DTW) distance. In Hautamäki et al. [2008], the medoid of the series is chosen as the initial representative. We have found that this initialization does not improve the quality of the outcome and therefore we recommend keeping the shortest path as the initialization. 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 was reduced 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.

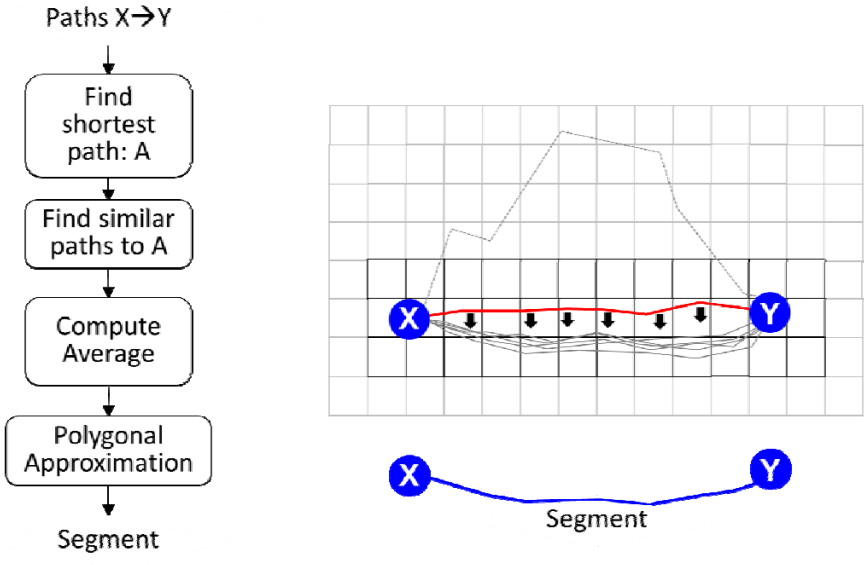

Figure 15. Left panel: Algorithm for creating a segment. Right panel: Example where the initial guess is optimized using the similar paths. The dilated cells used by the similarity function are highlighted using darker color.

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 was 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] propose 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.

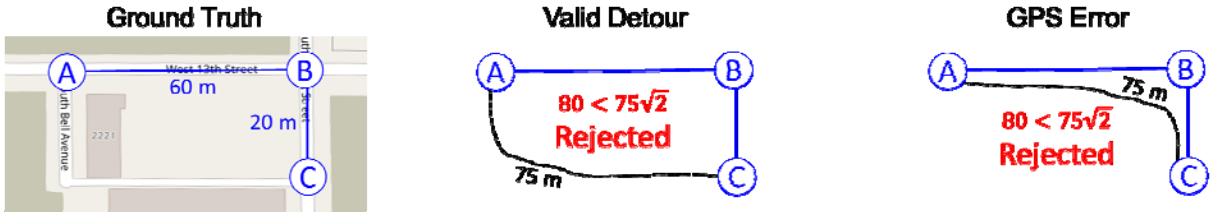

Fig. 16. Two examples where a segment is rejected according to the length rule. In the example on the left, the link should be kept because it represents a different road. On the right, the link should be removed because it is merely affected by GPS error.

To handle such problems, we present a filtering strategy based on spatial properties. For each segment, we first selected all other segments that were contained in the same region. These segments were used to form a subgraph. If a path existed in this subgraph, the segment was removed (Figure 17). We used the inclusion function from Mariescu 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.

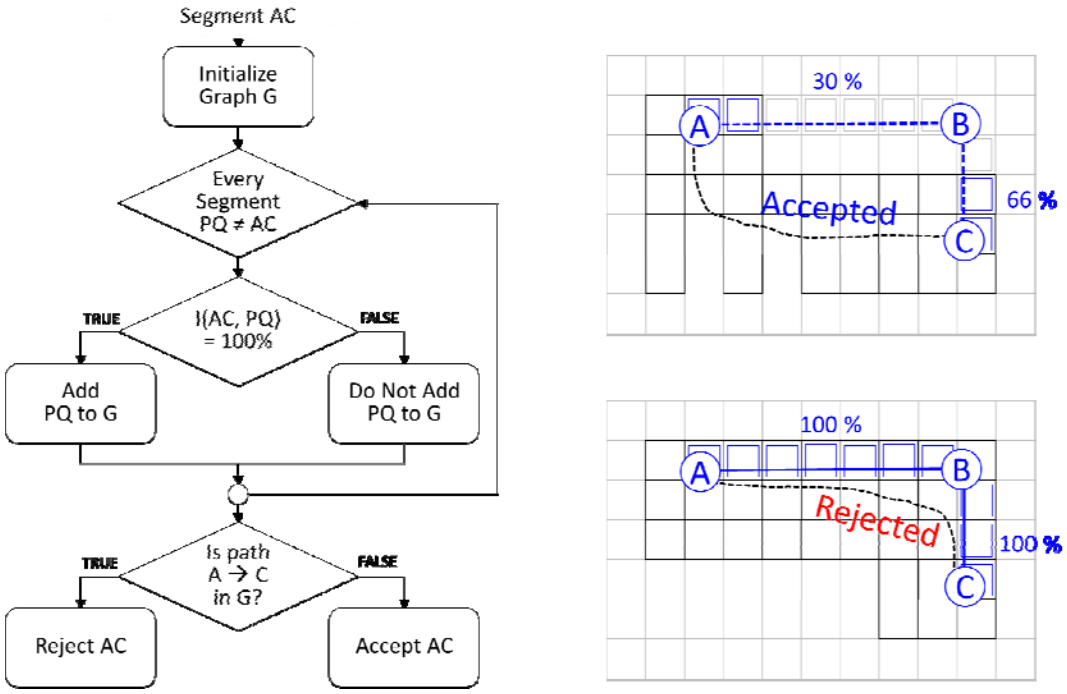

Fig. 17. Algorithm for filtering the segments (left panel). Examples where the segment is accepted (above, right) and rejected (bottom, right). The cell representations are shown. In the bottom right example, AB and BC are included in the region of AC and they form path A-B-C, which means the direct segment from A to C is redundant and rejected.

4. EVALUATION

We evaluated the proposed method using two datasets: Chicago and Joensuu[2], shown in Table 1 and Figure 18. The Chicago dataset is publically available [Biagioni and Eriksson 2012] and contains 889 routes of the campus shuttles at the University of Illinois at Chicago. 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 Joensuu by cropping the data to a square region covering most of the downtown area. Joensuu contains straight perpendicular roads in the centre and more complex curvy roads at the borders; the later are walking and cycling paths. The routes in Joensuu are collected while the user is jogging, usually along the sides of the streets.

[2] http://cs.uef.fi/mopsi/routes/network

Table. 1. Datasets used in the experiments.

Features	Chicago	Joensuu
Routes	889	108
Points	118,237	43,632
Intersections	52	228
Road segments	76	357
Points per segment (average)	6.6	4.8

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.

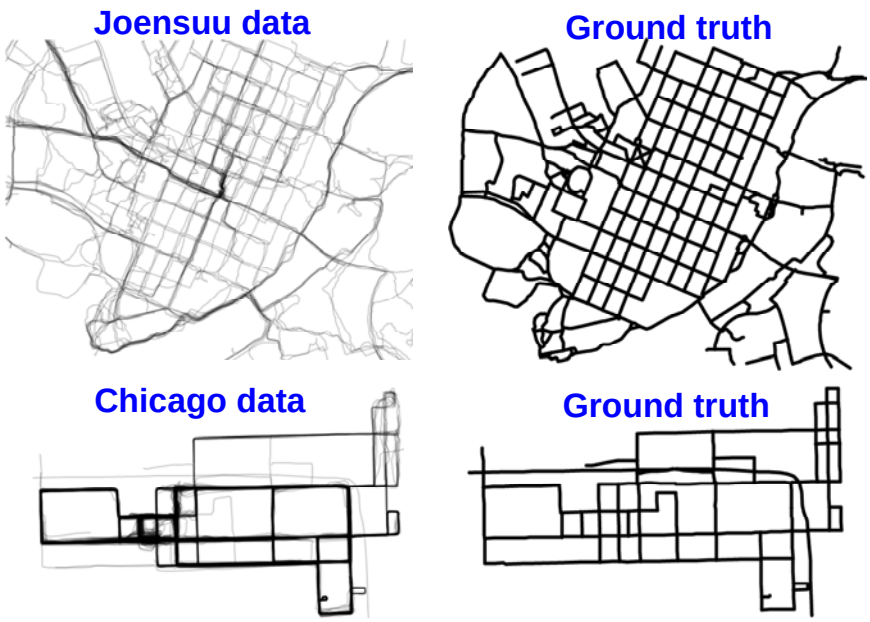

Figure 18. Joensuu and Chicago datasets, and the corresponding ground truth.

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 covered twice as large an area as Chicago's when counting the number of cells. The route density in Joensuu was lower: the average number of routes per cell was 5 compared with 91 in Chicago. The number of extracted segments per road was 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 was 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 was $O(Cmf)$. Overall, this step was one of two bottlenecks for the Chicago data and required 37% of the total processing time.

Table 2. Variables used and values obtained by CellNet for 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

Extracting the intersections depends on the number of splits found (S) in the previous step. Every split was compared against all others, leading to $O(S^2)$ time complexity. However, even if the number of splits was not small (2,118 in Joensuu), it merely needed simple thresholding and could be processed rapidly. Overall, this step required 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 checked if an intersection was close (<L) by analysing the cell it resided in and all its adjacent cells. These took $O(Np_rf)$ time in total. This step required about 2% of the total processing time.

Time complexity for the creation of the segments is linearly dependent 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 was large, they accumulated, and this step constituted 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 initialization; only rarely are substantially more iterations needed.

Filtering the segments requires computing the inclusion value between all segment pairs, which requires $O(R^2p_h)$. This step was the bottleneck for the Joensuu dataset, which had significantly more segments than the Chicago dataset. Then, for every segment, we checked if there existed a path linking the extremities in the subgraph. The subgraphs were small – fewer than 5 nodes – and any search strategy such as depth first search or breadth first search could be effectively applied. We used depth first search. In total, this step required 11% of the computation capacity for the Chicago dataset, and 71% for the Joensuu dataset.

The time complexities and observed processing times are summarized in Table 3. Overall, the algorithm required about 1 hour for the Joensuu dataset and 2 hours for Chicago.

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

4.2 Quality Comparison

We next compared the CellNet method with three conceptually different state-of-the-art approaches: a visual method [Davies et al. 2006], 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 [2012]. Visual outputs are shown for all these methods and CellNet in Figure 19, and a summary is provided in Table 4. The 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 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 was small for both the clustering and merging methods; however, the complexity of the overall network remained high owing to many spurious segments. Among the methods compared, the results from CellNet matched the ground truth most closely and the number of points used to represent the segments was optimized. In fact, this number was smaller than the ground truth, indicating that the ground truth itself (OSM) could be optimized.

Table 4. The number of intersections and segments obtained by various methods.

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

We next evaluated how well the algorithms performed at finding the intersections. Both the detected and the ground truth intersections were geographic locations (latitude, longitude). To compare the correctness of the extracted locations, we performed a nearest-neighbour search from each detected intersection to its nearest one in the ground truth. Then we counted how many real intersections were not found similarly, as 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 calculated three measures:

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

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

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

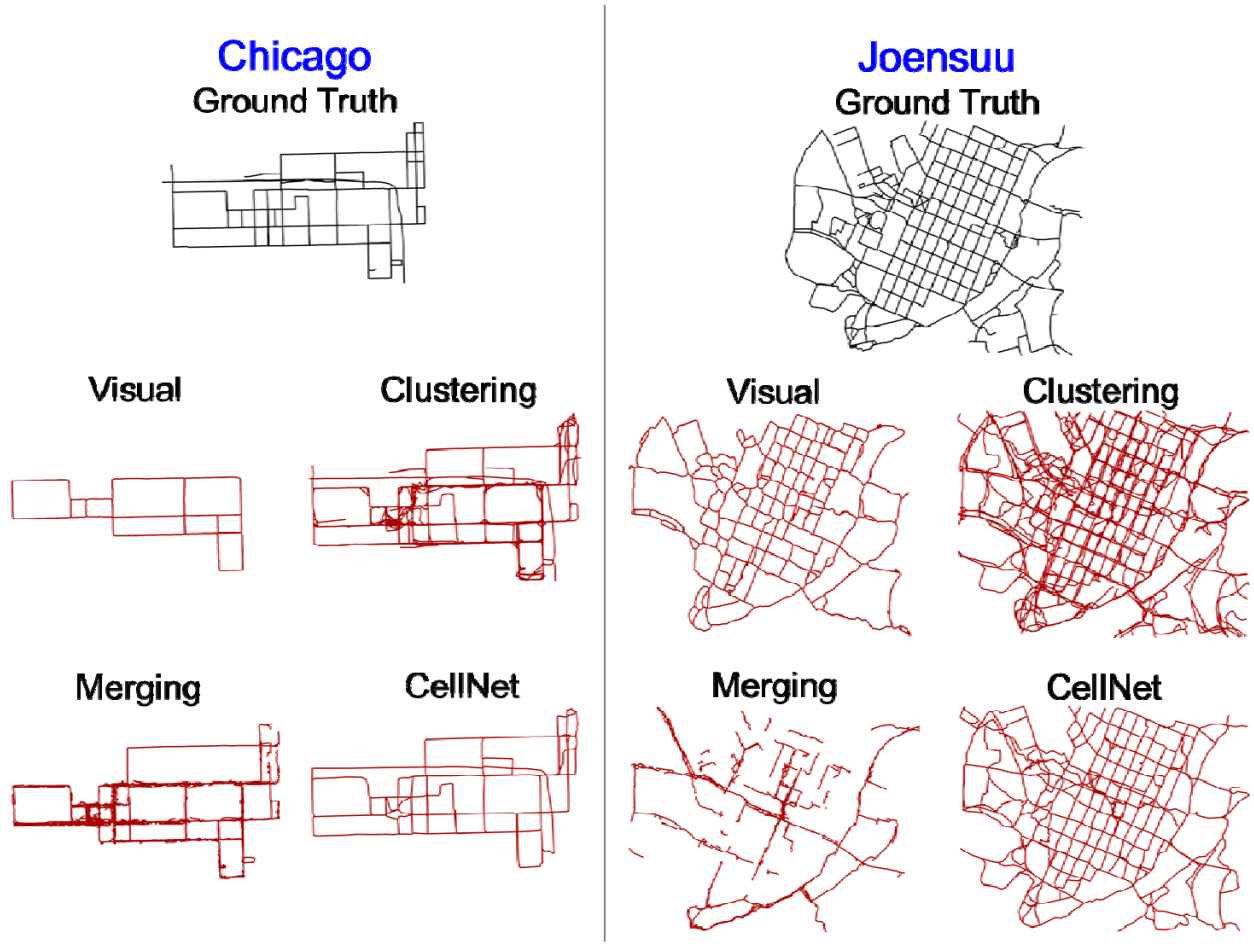

Figure 19. Visual output of the four methods for the Chicago and Joensuu datasets.

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 summarized in Table 5 as F-scores. The visual method displayed the highest precision for the Chicago dataset. This is partly because it detects only a few intersections (i.e. the method avoids false detections), and partly because the routes have high density in the region, which allows the visual-based method to work more accurately. However, the recall of the visual method is low because using a density threshold means that many intersections are missed. The clustering and merging methods have high recall, because – unlike the visual method – 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 was the most balanced in terms of precision and recall, and it produced the highest F-scores.

Table. 5. Quality of the intersections generated by the four measures.

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

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 converted them into cells. Then we created a second set from the extracted segments. To evaluate the success of a method, we calculated 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 computed precision, recall and F-score.

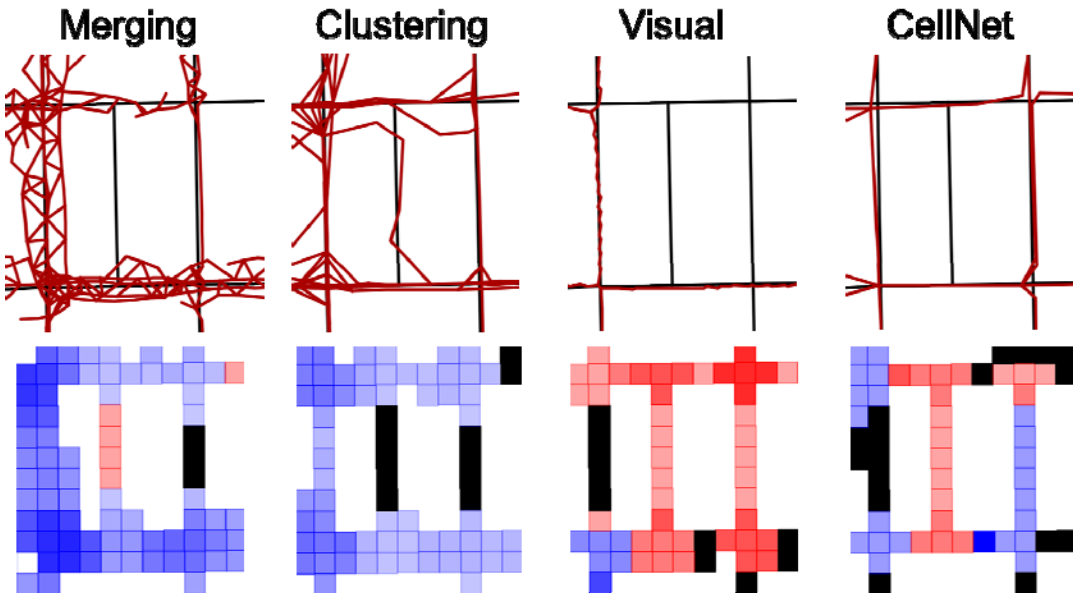

Fig. 20. Ground truth segments (black) and extracted segments (red) are shown at the top, and the corresponding cell frequency differences are shown at the bottom. Blue cells represent negative frequency (false detections), and red cells positive frequency (missed segments). Black cells have 0 frequency. The colour intensity is proportional to the frequency.

Table 6 summarizes the results for the four methods when finding the road segments. Similar observations can be made as in the intersection evaluation. The visual method achieved the highest precision but had

the lowest recall, whereas clustering and merging displayed high recall but low precision. In the noisy regions, the clustering and merging methods produced many spurious segments, as shown in Figure 20.

Table. 6. Quality of the roads generated by the four measures.

Chicago				
Method	**Reference**	**Precision**	**Recall**	**F-score**
Visual	Davies et al. 2006	97%	27%	42%
Clustering	Edelkamp and Schrödl 2003	17%	94%	28%
Merging	Cao and Krumm 2009	7%	70%	10%
CellNet	Proposed	92%	83%	87%
Joensuu				
Method	**Reference**	**Precision**	**Recall**	**F-score**
Visual	Davies et al. 2006	56%	38%	46%
Clustering	Edelkamp and Schrödl 2003	24%	87%	38%
Merging	Cao and Krumm 2009	13%	33%	19%
CellNet	Proposed	68%	49%	58%

4.3 Discussion of the Parameter Setup

The three compared methods were implemented by Biagioni and Eriksson [2012], who closely followed the descriptions in their respective papers, except for the clustering method [Edelkamp and Schrödl 2003]. Biagioni and Eriksson [2012] did not implement the intersection refinement process for the clustering method. The visual method [Davies et al. 2006] uses three parameters: cell size, density threshold and kernel bandwidth. 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 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.

Table 7. Parameters used by the different methods.

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

Note: Optimized values are shown for Chicago and Joensuu.

We optimized the parameters of the methods using a trial-and-error approach and the observations of Biagioni and Eriksson [2012]. It is possible that better quality can be achieved; however, the optimization task is tedious and time consuming. For CellNet, we optimized 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%). Optimized 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 optimized for the Chicago dataset on the Joensuu dataset directly (Table 8). The visual method [Davies et al. 2006] 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 optimizing 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 had already produced good results before optimization – even better than other methods after optimization. This finding suggests that parameter optimization is not required by CellNet, which is expected to work with the recommended values (L = 25, R = 80).

Table 8. Results when using the parameters from Chicago dataset on the Joensuu dataset.

Method	References	Chicago parameters		Optimized parameters	
		Intersections	Segments	Intersections	Segments
Visual	Davies et al. 2006	-	-	58%	46%
Clustering	Edelkamp and Schrödl 2003	46%	35%	54%	38%
Merging	Cao and Krumm 2009	27%	9%	31%	19%
CellNet	Proposed	63%	56%	69%	58%

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 of visual methods 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 [2012]. The route merging method suffers in the presence of high GPS noise. It is also far slower than the other approaches, as shown in Biagioni and Eriksson [2012]. 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 optimized values.

Table 9. Running times for the different methods using the two datasets.

Method	Chicago	Joensuu
Visual	15 min	14 min
Clustering	54 min	15 min
Merging	2.5 days	3 h
Proposed	1.9 h	1.1 h

We compared the memory requirements for each of the networks; the results are shown in Table 10. Because of the point reduction step, the size of the network produced by CellNet was small at less than 25% of the networks produced by any 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.

Table 10. Size of the networks represented as total number of points of all detected roads.

Method	Chicago	Joensuu
Visual	1,309	4,752
Clustering	2,119	5,366
Merging	4,749	6,097
Proposed	331	1,215

5. CONCLUSIONS

We present a new road network inference method, called CellNet, consisting of two steps: first, it finds the road intersections and then it creates the in-between segments. CellNet works well on different route datasets, without the need for time-consuming parameter optimizations. It produced higher accuracy (F-scores) than three conceptually distinct state-of-the-art methods when tested on two different real route datasets. The memory requirements of the resulting networks were considerably smaller – roughly 25% – compared with the size of networks generated by other methods we tested. The speed was only mid-range. Perhaps a more efficient algorithm could be used to improve the segment optimization step.

REFERENCES

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), pp. 113-118.

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

Lili Cao and John Krumm. 2009. From GPS traces to a routable road map. *In Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, Seattle, Washington, USA, pp. 3-12.

Chen Chen and Yinhang Cheng. 2008. Roads digital map generation with multi-track GPS data. *IEEE International Workshop on Education Technology and Training, 2008. and 2008 International Workshop on Geoscience and Remote Sensing*. Vol. 1, pp. 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), pp. 2770-2785.

Yizong Cheng. 1995. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8), pp. 790-799.

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

Stefan Edelkamp and Stefan Schrödl. 2003. Route planning and map inference with global positioning traces. *In Computer Science in Perspective*, pp. 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*, Zurich, Switzerland, pp. 56-69.

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

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

Ville Hautamäki, Pekka Nykänen and Pasi Fränti. 2008. Time-series clustering by approximate prototypes. *IAPR International Conference on Pattern Recognition*, Tampa, Florida, USA, pp. 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), pp. 4144-4157.

Piyawan Kasemsuppakorn and Hassan A. Karimi. 2013. A pedestrian network construction algorithm based on multiple GPS traces. *Transportation research part C: emerging technologies*, 26, pp. 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 (submitted).

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*, Volume 2, Number 4.

Arie Pikaz. 1995. An algorithm for polygonal approximation based on iterative point elimination. *Pattern Recognition Letters*, 16 (6), pp. 557–563.

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. ACM International Conference on Knowledge Discovery and Data Mining Workshop on Mining Temporal and Sequential Data, Seatle, Washington, USA, pp. 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), pp. 59-87.

David Schultz and Brijnesh Jain. 2017. Nonsmooth Analysis and Subgradient Methods for Averaging in Dynamic Time Warping Spaces

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