From Routes to Roads: Infer road network using GPS trajectories

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

Road networks are essential nowadays, especially for people traveling to large, unfamiliar cities. Moreover, cities are constantly growing and road networks need periodical updates in order to provide reliable information. We propose an automatic method to generate the road network using a GPS trajectory dataset. The method, entitled CellNet, works by first detecting the intersections (junctions) using a clustering-based technique and then creates the road segments in-between. We compare CellNet against three conceptually different state of the art alternatives and show that it provides better accuracy and is less sensitive to parameter setup. The generated road network occupies 25% of the memory of the networks produced by the other methods.

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

1. INTRODUCTION

In recent years, navigation and location based services (LBS) have seen a rise in development. For these applications to work reliably, it is important to have an up-to-date road network. Maintaining the road networks requires large amount of manual editing, and thus, researchers have developed 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.

There are several different approaches to automatically construct a road network. 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 sub-problem of finding road intersections by analyzing the aerial images using and using a neural network.

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

Using aerial images has limitations because roads have varying features such as color, intensity, shadows and variable widths (see Figure 1). Obtaining the direction of travel for the roads is also not possible using image data. Furthermore, collecting new aerial images after road construction work is costly. Because of these reasons, methods based on GPS trajectories have appeared. GPS technology is a cheap alternative to aerial images due to the built-in positioning capability available in consumer devices such as smartphones, tablets, watches and cameras. These are utilized in location-based services, navigation, and tracking user movements. As a consequence, a large amount of GPS trajectories, referred to here as *routes*, have become available and they may be used to obtain the road network information (see Figure 2).

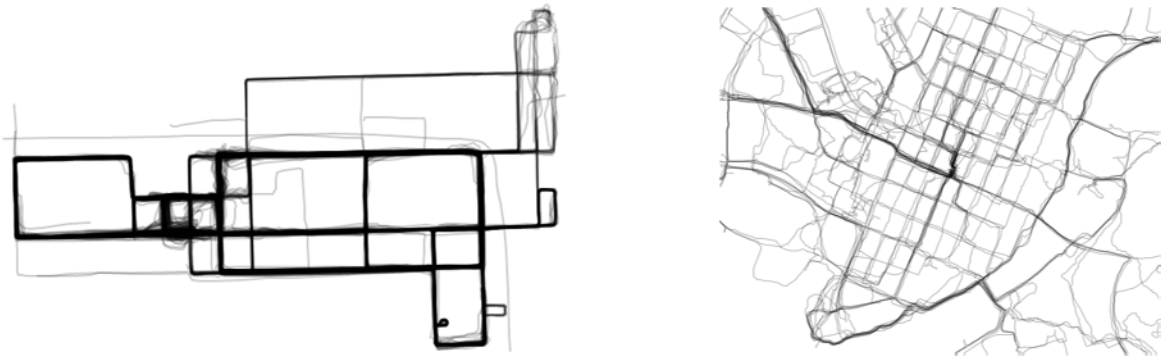


Fig. 2. GPS routes in Chicago, USA (left) and in Joensuu, Finland (right).

Visual methods (see Figure 3) use the 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 in order to produce an image skeleton, which represents the road network. Davies et al. [2006] also use the routes to form a binary image. The image is then blurred and a density threshold is applied to filter out parts with too few routes. The outlines are then extracted using a contour following algorithm and the centerlines of these outlines are computed using the Voronoi graph. These centerlines are used to form the underlying network.

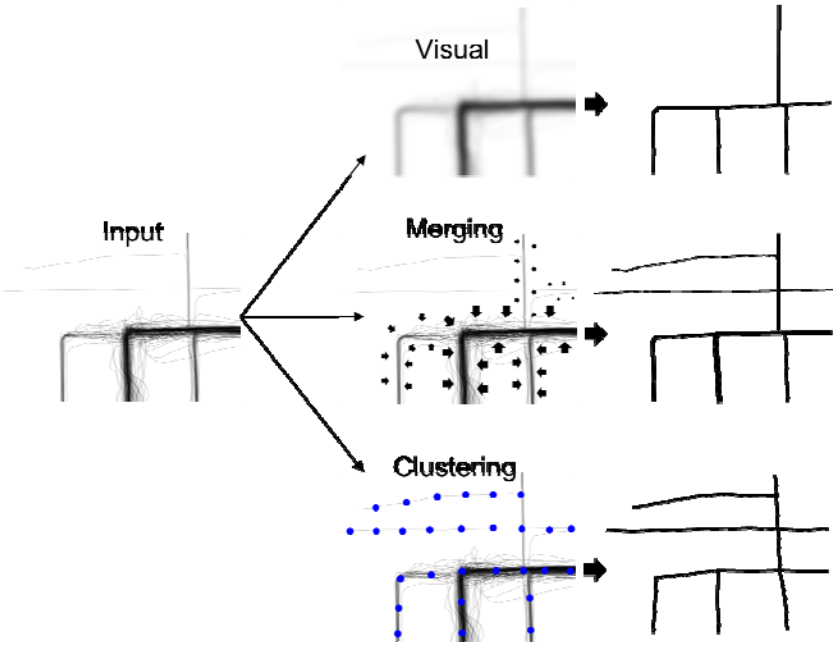


Fig. 3. Three conceptually different road network generation techniques: visual, merging and clustering.

Route merging (see Figure 3) methods [Niehoefer et al. 2009, Cao and Krumm 2009] combine routes one by one to form a graph. If a route segment is already part of the graph, a weight corresponding to that particular segment is increased. Finally, segments with too low weight 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 in order to obtain better representatives. A secondary attractive force is used to prevent the route points to move too far from their original location.

Clustering methods (see Figure 3) have also been used. 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 present also a lane finding strategy. In [Schrödl et al. 2004], the bounding box of each intersection is analyzed to compute the local turn-lane geometry.

The merging and clustering methods suffer in regions of high GPS error. There, unwanted intersections and multiple spurious road segments are created. The visual methods work better in this situation if setting the density threshold high enough, but the drawback is that the parts containing few routes will be 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 networks because in this way, GPS error only affects the shape of the roads, not the connectivity of the graph. Fathi and Krumm [2010] focus on this subproblem. 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.

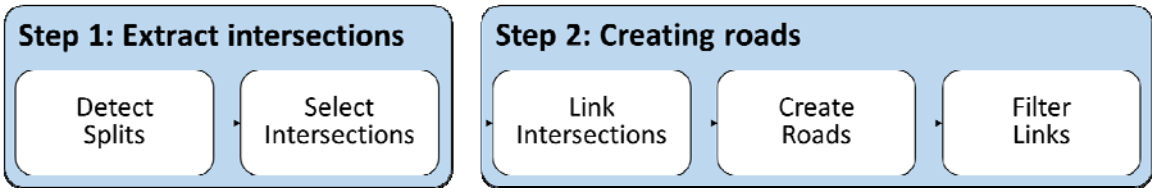


Fig. 4. The steps performed by CellNet to infer the road network.

In this paper we present CellNet, a two-step method for inferring the road network, see Figure 4. It first identifies intersections by clustering the route points around the regions where routes split into more directions. Unlike other approaches [Barsi and Heipke 2003, Fathi and Krumm 2010], our method does not require to train 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.

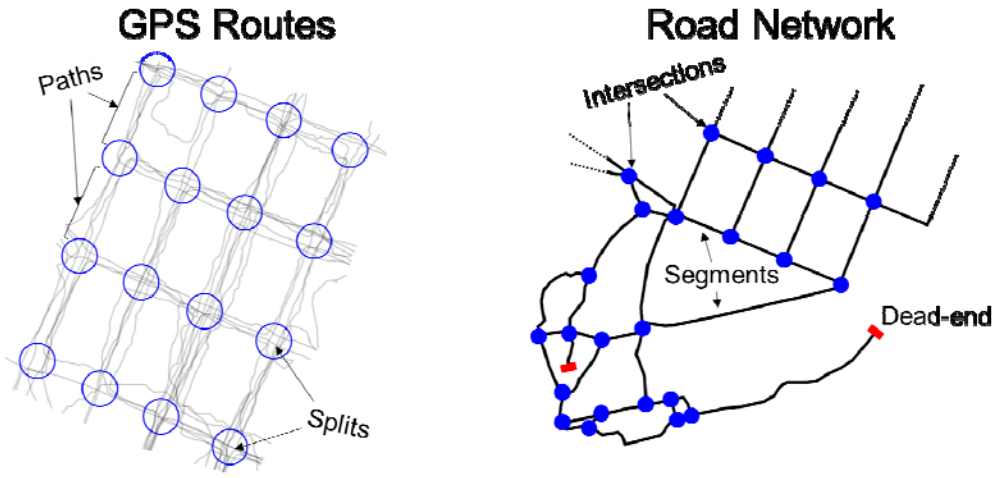


Fig. 5. The terminology used when discussing about GPS routes and a road network.

In Figure 5 we explain the terminology. The details of finding the intersections are given in Section 2. The steps of creating the roads are explained in Section 3. The proposed method is evaluated in Section 4 and compared with three existing approaches: a visual [Davies et al. 2006] a merging [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.

2. EXTRACT INTERSECTIONS

Intersections are places where more than two roads connect. To detect potential intersections from GPS routes we apply the two processes as shown in Figure 6. First we analyze the neighborhood of each point to detect *splits*. A split is defined as a point at which routes are heading to more than two principal directions (see Figure 7). It is likely that multiple splits will be found at the same intersection, especially if the intersection is large. From the detected splits, we measure the frequency of routes passing through. Splits having higher frequency than their neighbors (local maxima) are selected as intersections.

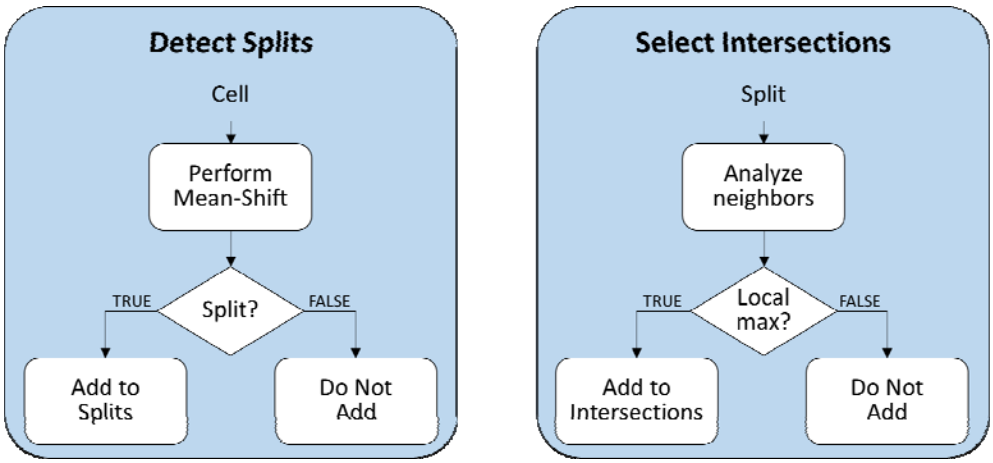


Fig. 6. Steps performed to detect the splits (left) and to select the local maxima (right).

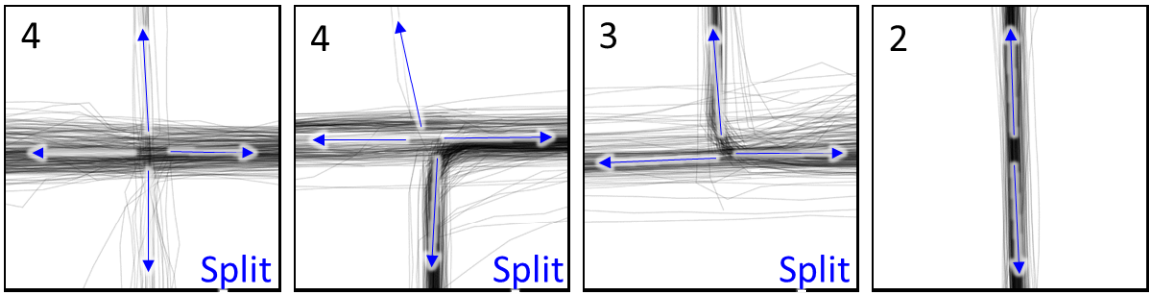


Fig. 7. Four examples of locations where routes head into a several multiple principal directions. The directions are emphasized by the arrows. The first three examples are splits according to our definition but the last one is not.

2.1 Detect Splits

To detect the splits, we analyze all locations where the routes pass through. To do this efficiently, we divide the space by a grid with cell length L = 25 meters (recommended). For every grid cell we maintain information containing its location, indexes of all routes passing through it and their total number. We accumulate the evidences 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 (see Figure 8). Due 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 and Fränti 2017].

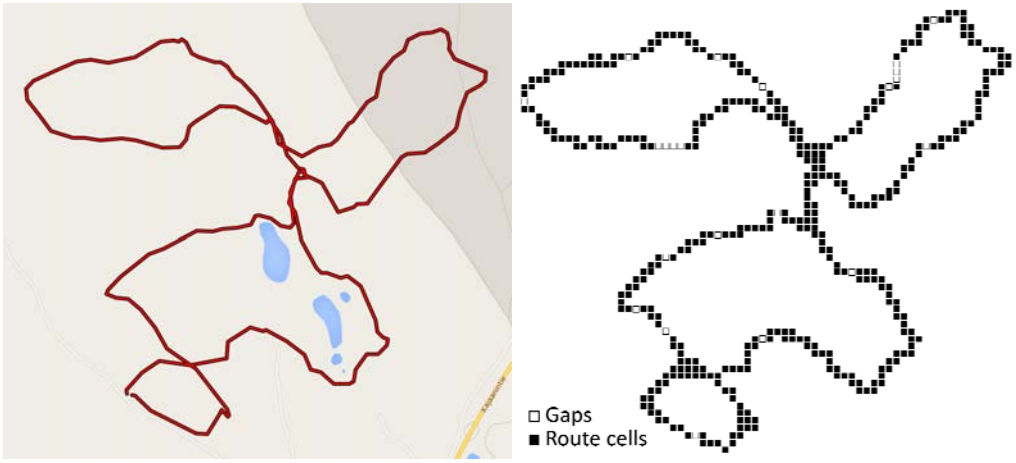


Fig. 8. A sample route (left) and the cell representation with cell size 25×25 meters (right). The gaps are filled using linear interpolation.

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

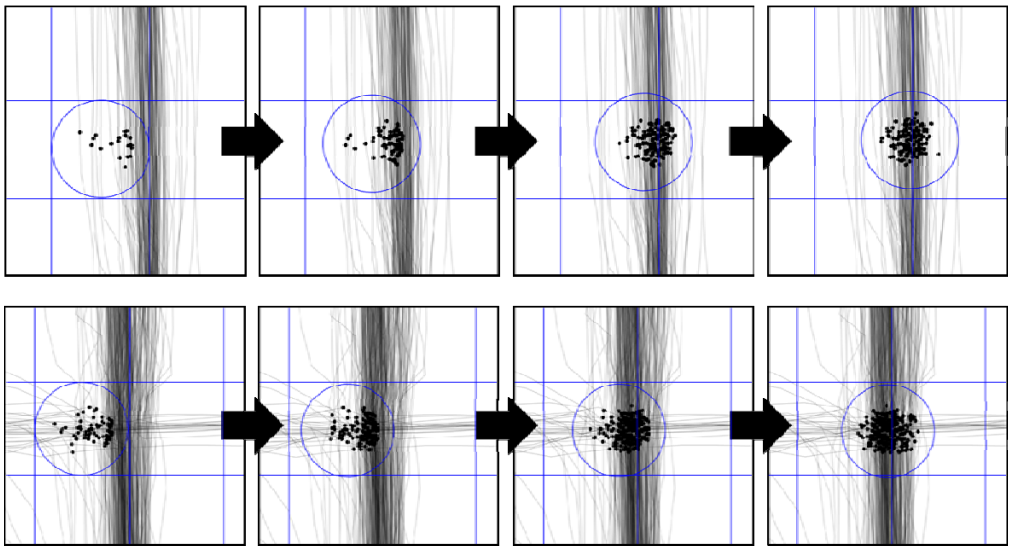


Fig. 9. Two examples of the mean-shift algorithm. The initial location gradually moves towards the center of the routes. If an intersection is nearby the location is likely to end up at its center.

The process is done as follows. We first transfer the location of the cell closer to the stream of routes by mean-shifting algorithm [Cheng 1995], which is basically a mode-seeking algorithm. At each step, it defines a fixed radius neighborhood and calculates the average location of the route points in this neighborhood. The location is then updated to this average and the process is repeated until the location stabilizes.

Figure 9 gives two examples of the mean-shift algorithm. After this process, it is possible for the location to end up in a different cell than where it started.

After the location has been tuned, we analyze the neighborhood 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 L-radius circle around the tuned location. The extremity is a circular band of width L situated at R meters from the origin, see Figure 10. We recommend to use the values L=25 meters and R=80 meters, although their exact choice is not critical.

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 the points inside the extremities in each of the two directions (before and after the origin). Exceptions are routes that end inside the region, which only pass once, or not even at all if they also start in the same region (routes that contain no movement).

Averaging has multiple benefits. First, it avoids problems caused by routes that traverse along the extremity, which could lead to false detection of 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 impact on the further analysis.

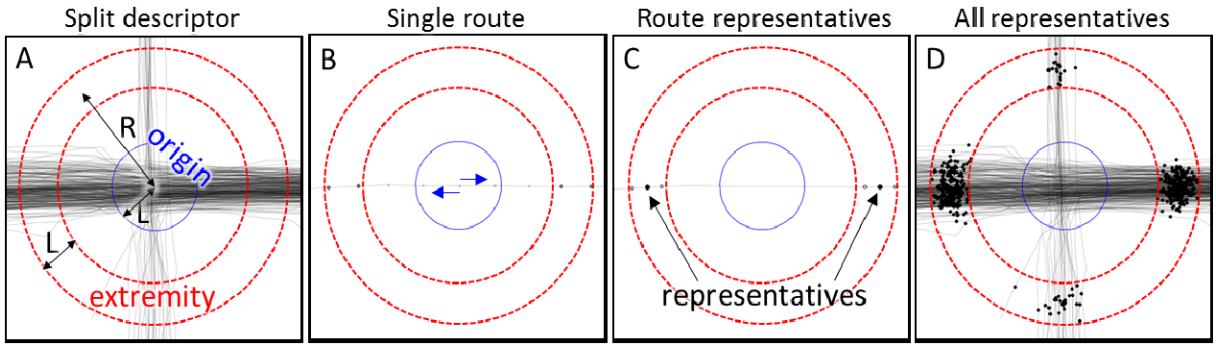


Fig. 10. A – the split descriptor composed of the origin and the extremity. B – a sample route traversing through the point of interest. Points inside the extremity are highlighted. C – The points inside the extremity are averaged in each of the two directions to create the representatives. D – Representatives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

The representatives found by the descriptor are then clustered using *Random Swap* algorithm [Fränti and Kivijärvi 2000] but using repeated k-means might also be good enough. To find the correct number of clusters, we cluster separately using 2, 3 and 4 clusters. The number of clusters that best models the data defines the number of directions. For detecting the number of clusters we use the maximum *Silhouette Coefficient* (SC) value according to [Rousseeuw and Kaufman 1990], which is the average value of all silhouettes belonging to every centroid:

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

$$SC = \frac{1}{k}\sum_{i=1}^{k} s_i$$

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. In practice, it is enough to

cluster using 2 and 3 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.

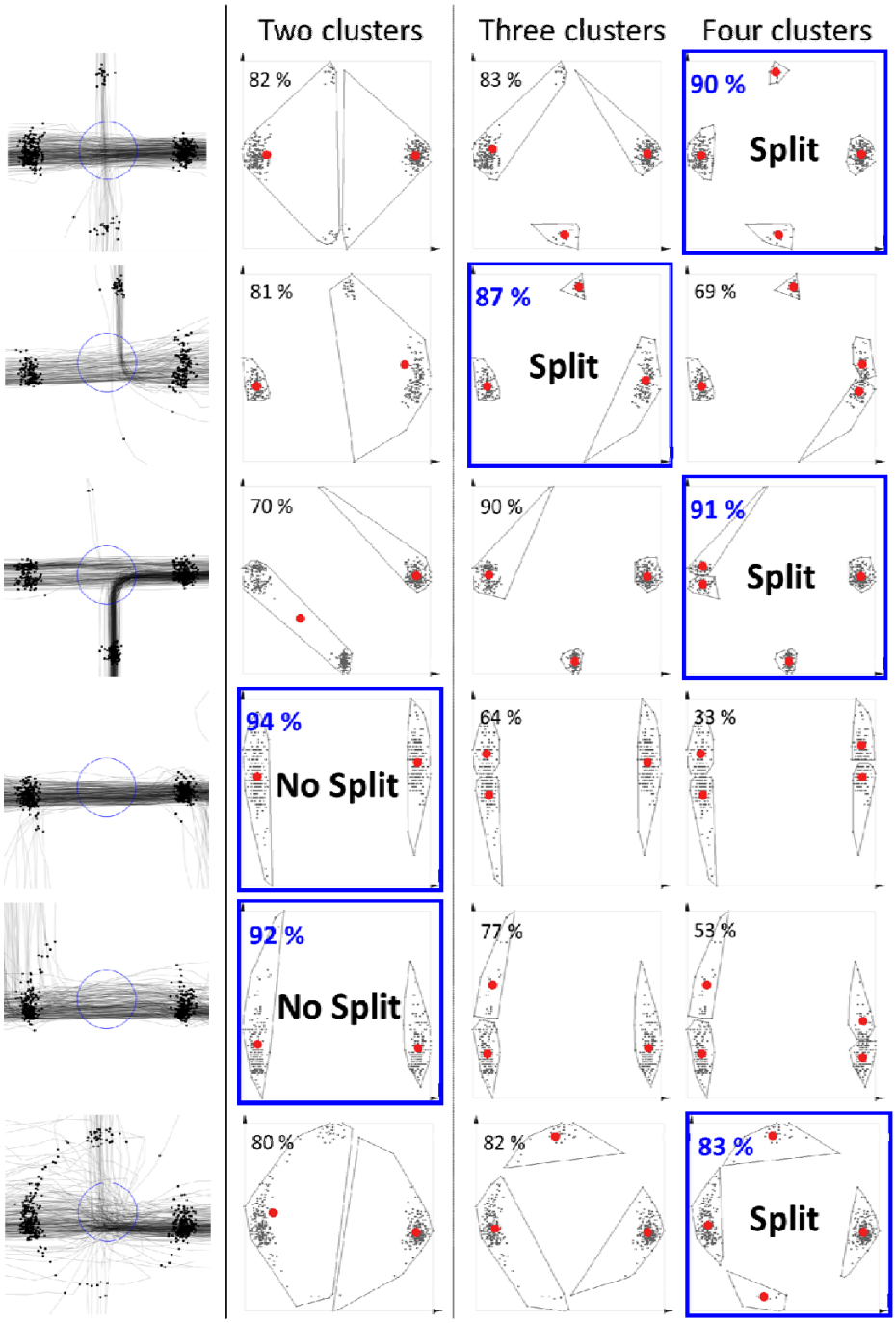


Fig. 11. Six locations investigated for splits. Each dataset is clustered by Random Swap algorithm using 2, 3 and 4 clusters respectively. The percentages represent the value of the Silhouette coefficient. More than 2 clusters indicate a split.

2.2 Select Intersections

After the splits have been detected, we need to select a subset that would capture all the intersections only once. It is possible that multiple split locations are found for an intersection because the split descriptor will detect any local maxima within the distance R from the intersection (see Figure 12). Mean-shift 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 it than any of its neighboring candidates within radius R.

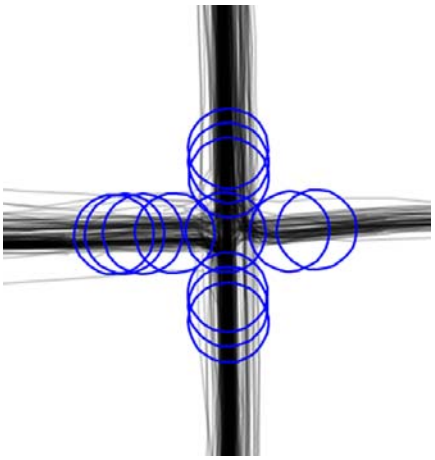
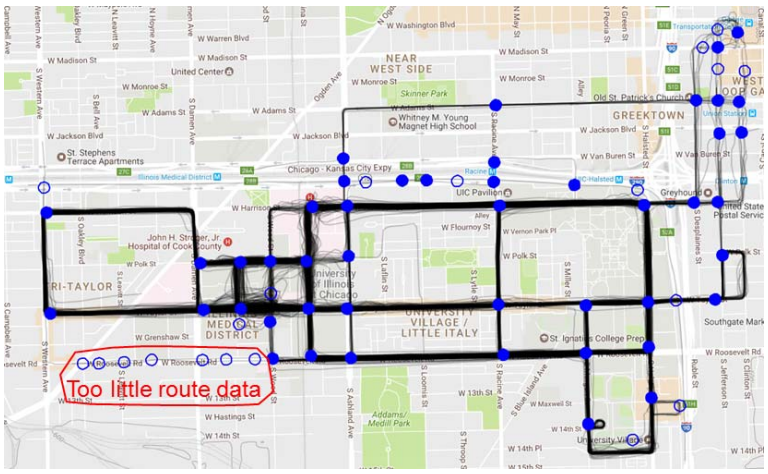


Fig. 12. Multiple splits detected near the true intersection.

The two steps are demonstrated in Figure 13 for Chicago dataset[1]. The split detector correctly finds the intersections but also several false positives. The selection step manages to remove most of these without losing any real intersection. The remaining false positives appear mainly in the areas with very high GPS error or insufficient data (see Figure 13). There are many false positives detected where there are only 2 routes traveling alongside each other. In this situation, the clustered dataset only has 4 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, it is recommended that the dataset is checked to have more than 2 routes in every region. This, however, should be a prerequisite for any road network inference method, because single observations can be the result of GPS error.



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

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

In Figure 13, the false positives in the region with too little route data do not affect the structure of the resulting network because after the road creation step they will result in a single, long, road.

3. CREATING ROADS

After the intersections have been found, we connect them. We take each route in the dataset and link any two intersections it passes through in sequential order. In order to create the roads we use the route segments.

3.1 Connect Intersections

We analyze each route as shown in Figure 14. We first get the intersections that the route passes through and connect 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.

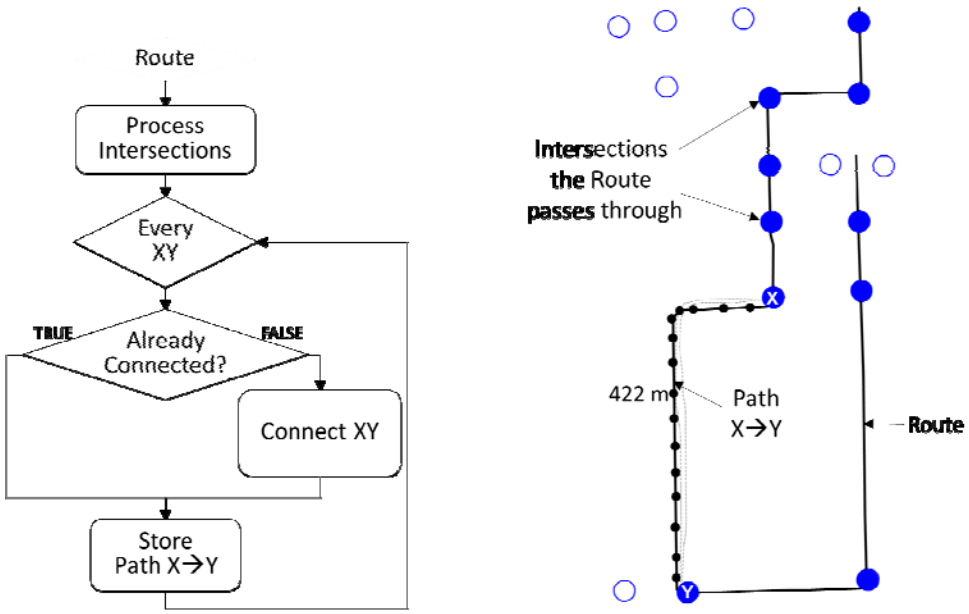


Fig. 14. Algorithm for linking the intersections (left). Example of a route passing through several intersections (right). Connections are formed between pairs of intersections in the order that the route passes through. For every connection, all paths are stored.

3.2 Create Segments

To construct the road segments we consider all paths between every two intersections. We choose the shortest path as an initial choice under the assumption that it has lower GPS error. This strategy was earlier proposed by Fathi and Krumm [2010] and seems to be 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 (red). To create the segment we first filter out paths that are not spatially similar to the initial choice. In this way, we avoid paths which may have missed a third intersection. According to our experiments, such paths do more harm than good. The similarity function from [Mariescu 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 square structural element. Only paths that are 100% similar to the shortest path are accepted.

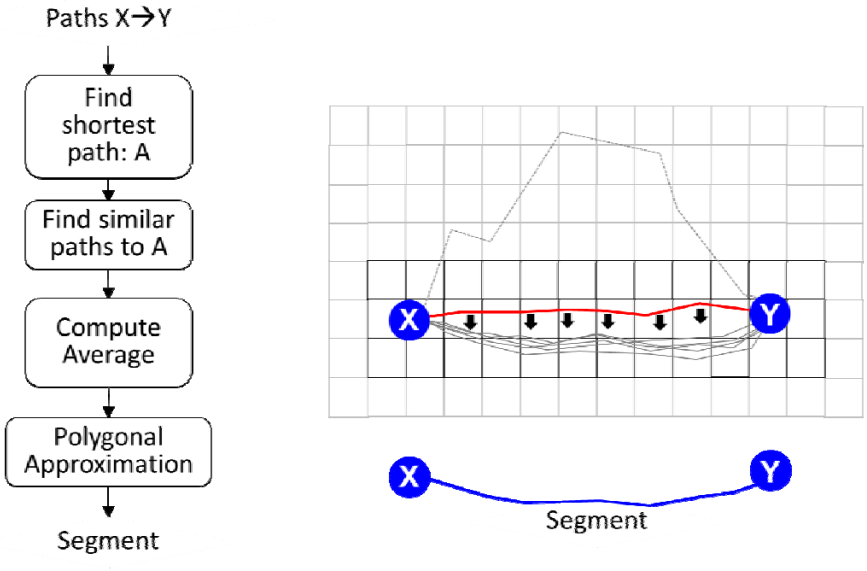


Fig. 15. Algorithm for creating a segment (left). Example where the initial guess is optimized using the similar paths (right). The dilated cells used by the similarity function are also highlighted.

We compute the average of the similar paths using the method described 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 find that this initialization does not improve the quality of the outcome and recommend keeping the shortest path as the initialization. By not computing the medoid the method is also much faster. We gain an additional speedup by applying the approximate FastDTW method [Salvador and Chan 2004] which works in linear time as opposed to the typical DTW which has quadratic time complexity. Using these two modifications, the processing time is reduced to about 1%. Alternative methods for averaging the paths such as [Schultz and Jain 2017] may 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. This can produce an unnecessarily complex network. We reduce the number of points of the segments by applying polygonal approximation. We use the algorithm in [Chen et al. 2012] but some simpler variant like in [Pikaz and Dinstein 1995] could be used as well. We reduced the number of points to 30% without any loss in accuracy. In fact, accuracy becomes slightly better because some noise is filtered out in the approximation.

3.3 Filter Segments

A route might miss one or more intersections due to GPS error. In such case, two intersections will become connected incorrectly. To handle this issue, Fathi and Krumm [2010] propose the following strategy: to remove a 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 due to GPS error. This

strategy is effective, however there are some situations when this does not work as intended. In Figure 16 we see two different scenarios where the before mentioned strategy rejects the road segment, even though in the example on the left the segment should be kept.

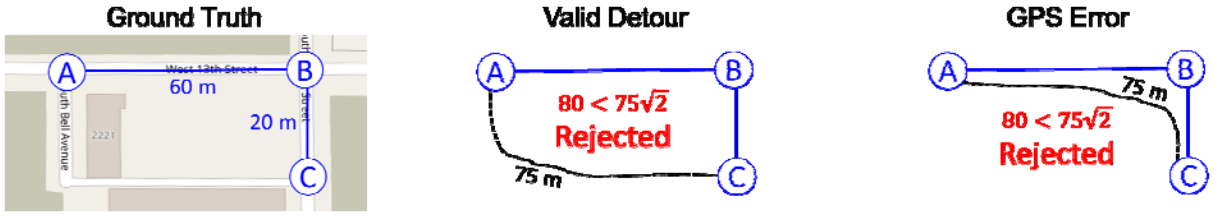


Fig. 16. Two examples where a segment is rejected according to the length rule. In the example on the left, the link should be kept because it represents a different road. On the right, the link should be removed because it is just affected by GPS error.

To handle such problems, we present a different 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 is removed (see Figure 17). We use 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. C_A and C_B are the cell representations of the two segments. $C_B{}^d$ are the dilated cells of segment B.

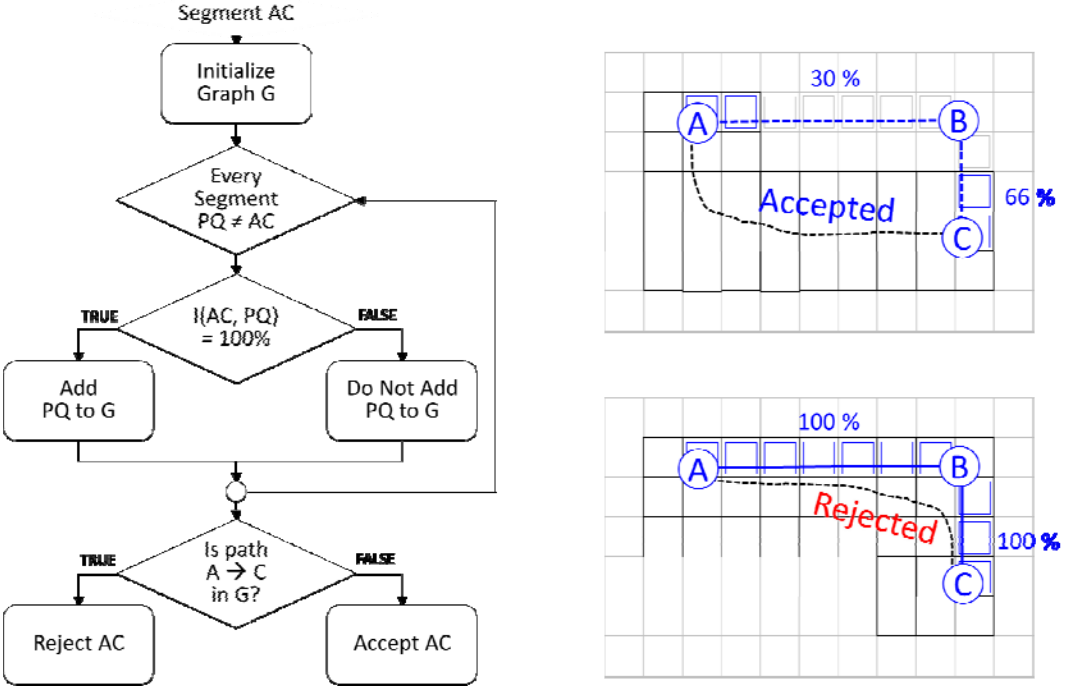


Fig. 17. Algorithm for filtering the segments (left). Example where the segment is accepted (above) and rejected (bottom). The cell representations are shown. In the example on the bottom, AB and BC are included in the region of AC and they form path A-B-C which means the direct segment from A to C is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

4. EVALUATION

We evaluate the proposed method using two datasets: Chicago and Joensuu[2], see Table 1 and Figure 18. The Chicago dataset is publically available [Biagioni and Eriksson]. It contains 889 routes of the campus shuttles in the University of Illinois at Chicago. The shuttles pass through major streets of the city. There are two areas that contain tall buildings which affect the GPS precision. The second dataset contains tracks of a single user (Pasi) from Mopsi collection during 16.11.2014 - 25.4.2015. This collection includes 102 routes in total but we extract only the 45 that are in Joensuu area by cropping the data to a square region covering most of downtown. Joensuu contains straight perpendicular roads in the center and more complex, curvy roads on the borders. The later are walking and cycling paths. The routes in Joensuu are collected while jogging, usually on the sides of the streets.

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 generate ground truth from open street map by querying all road segments in the respective areas: Joensuu and Chicago. We then manually excluded road segments that were not travelled in the data (see Figure 18). In this way, it would be theoretically possible to achieve 100% accuracy by a perfect algorithm. Joensuu dataset has about 4 times as many intersections and almost 5 times as many road segments as Chicago dataset. The number of points per segment do not differ significantly.

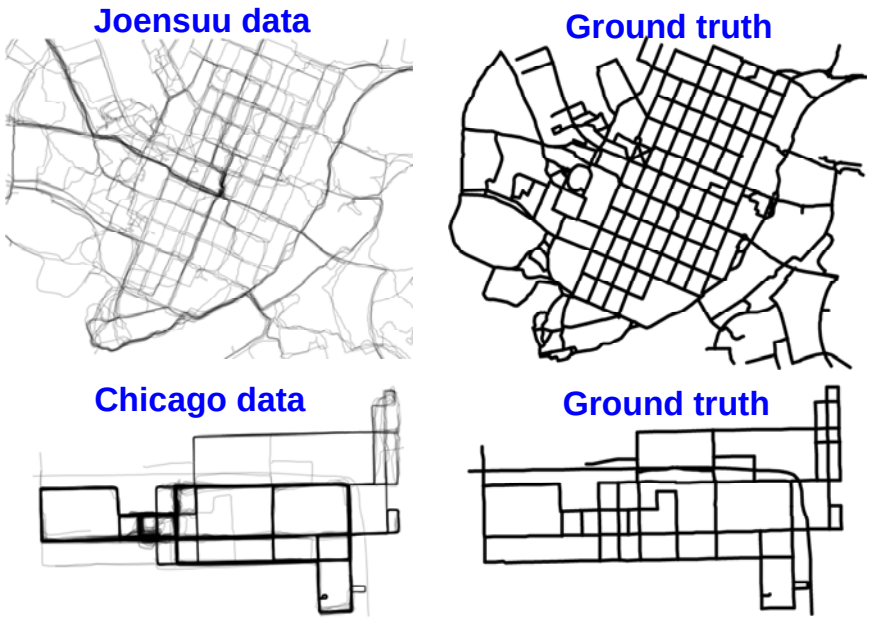


Fig. 18. Joensuu and Chicago datasets, and the corresponding ground truth.

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

4.1 Processing Time

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

Table. 2. Variables used by CellNet and values obtained when running it on 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

The time complexity of the split detection step depends on the size of the area covered by the routes, more precisely on the number of non-empty cells. For every cell, mean-shift is performed once and clustering three times using random swap algorithm with a fixed number of iterations (100) and different 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 the two bottlenecks with Chicago data requiring 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 to each other, leading to $O(S^2)$ time complexity. However, even if the number of splits is not small (2,118 in Joensuu) it needs just simple thresholding and can be done fast. Overall, it requires just a fraction of the total processing time (0.01% in Chicago and 0.2% in 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 intersection is close (<L) by analyzing the cell it resides in and all its adjacent cells. These take $O(Np_rf)$ time in total. This step takes about 2% of the total processing time.

Time complexity for the creation of the segments has linear dependency 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 very large, they cumulate so that this step is the second bottleneck of the algorithm with Chicago dataset requiring 50% of the total processing time. The value of i remains small due to the fact that the shortest segment is usually a very good initialization so that only rarely significantly 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 of the Joensuu dataset, which has significantly larger number of

segments compared to Chicago dataset. Then, for every segment, we check if there exists a path linking the extremities in the subgraph. The subgraphs are small (less than 5 nodes) and any search strategy can be effectively applied. We used depth first search. In total, this step requires 11% of the computation with Chicago dataset, and 71% with the Joensuu dataset.

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

Table. 3. Time complexity and processing time of each step of the method.

Step	Time complexity	Processing time (s)	
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	$O(Sp_hi)$	3,630	370
Filter segments	$O(R^2p_h)$	809	2,738
Total	$O(Cmf + S^2 + Np_rf + Sp_hi + R^2p_h)$	1.9 hours	1.1 hours

4.2 Quality Comparison

We next compare the CellNet method with three conceptually different state of the art approaches: a visual [Davies et al. 2006] a merging [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 given for all these methods and CellNet in Figure 19 and a summary is provided in Table 4. We see that the visual method finds too few segments from the Chicago dataset, namely, the parts with too few data are missed. This does not happen as much in Joensuu where the route density is more constant. The segments obtained by the visual method are very complex when looking at the number of points. The clustering method finds too many intersections and spurious road segments especially in regions with high GPS error. The merging method also finds too many intersections and segments. In Joensuu, it produces 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 is still high due to many spurious segments. Among the methods compared, CellNet gives results closest matching to the ground truth and the number of points used to represent the segments is also optimized. In fact, this number is smaller than the ground truth, indicating that the ground truth itself (open street map) could be optimized.

Table. 4. The number of intersections and segments obtained by the 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

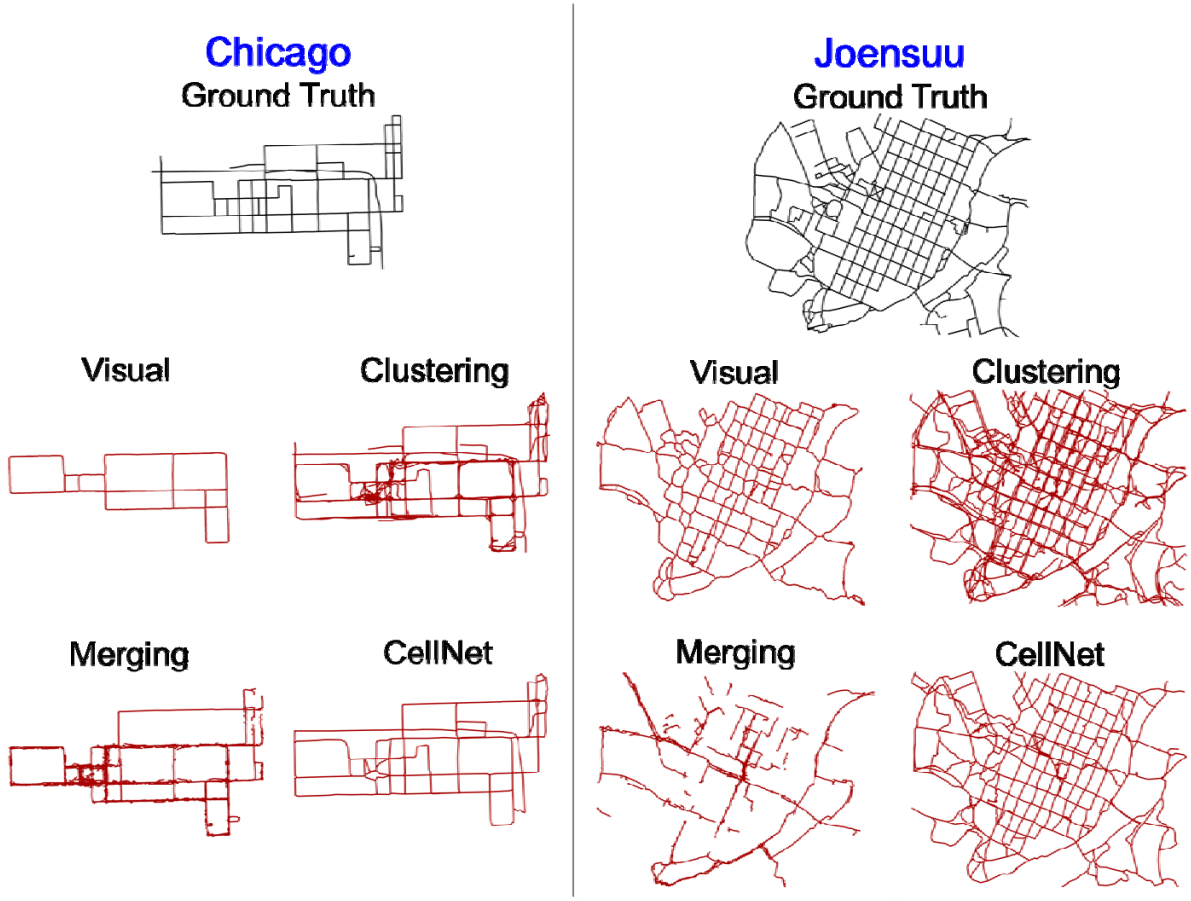


Fig. 19. Visual output of the four methods on Chicago and Joensuu datasets.

We next evaluate how well the algorithms found 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 nearest neighbor search from each detected intersection to its nearest one in the ground truth. Then we count 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 to the other direction (from ground truth to detected intersections). The unmapped intersections count as false detection (*false positives*), i.e. detected segment that does not have a match in the ground truth. Using these values we calculate three measures:

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

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

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

Even though some of the methods do not specifically detect the intersections, the 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 their f-scores. The visual method has the highest precision with Chicago dataset. This is partly because it only detects few intersections (avoiding false detections), and partly because the routes have high density in the region allowing the visual-based method to work more accurately. However, its recall is low because using a density threshold means that many intersections will be 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 is the most balanced in terms of precision and recall, and it produces the highest f-score.

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 introduce next a novel approach to evaluate the correctness of the road segments. First, we take all the segments from the ground truth, and convert them into the 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 with frequency 0). Otherwise, some cells will have positive frequency (missed segments) while some other cells will have negative frequency (false segments). Cells with 0 frequency are the desired result (correct detection), see Figure 20. We compute Precision, Recall and F-score.

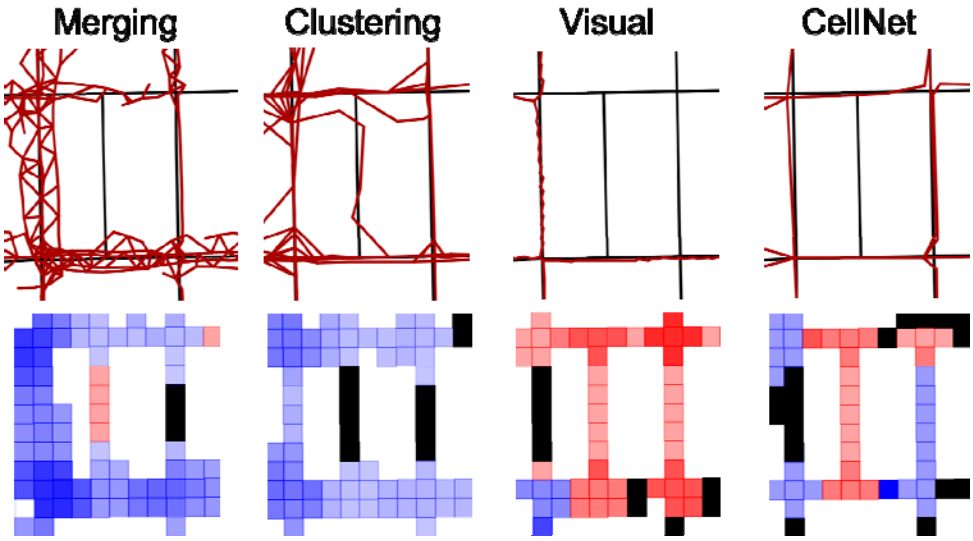


Fig. 20. Ground truth segments (black) and extracted segments (red) are shown on the top, and the corresponding cell frequency differences are shown on the bottom. Blue cells represent negative frequency (false detections), and red cells positive frequency (missed segments). Black cells have 0 frequency.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

Table 6 summarizes the performance of the different methods when finding the road segments. Similar observations can be made as in the intersection evaluation. The visual method has highest precision but lowest recall, whereas clustering and merging have high recall but low precision. In the noisy regions, the clustering and merging methods produce 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] closely following the description in their respective papers, except for the clustering method [Edelkamp and Schrödl 2003] where they did not implement the intersection refinement process. The visual method [Davies et al. 2006] has three parameters: the cell size, the density threshold and the kernel bandwidth. The clustering method has three parameters: the cluster seed interval, the intracluster bearing difference and the intracluster distance. The merging method [Cao and Krumm 2009] has three parameters: the edge volume, the location distance limit and the location bearing difference. It uses several other parameters in the route clarification step, however, it is a separate part from the method itself and we will not present them here. All methods have also a forth parameter: the number of routes to be used. We disregard this parameter because it is

essentially a sub-sampling of the dataset which can be done as a separate preprocessing step when the size of the data is too big.

Table. 7. Parameters used by the different measures. Optimized values are shown for Chicago and Joensuu.

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 (path length?)	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

We optimized the parameters of the methods using 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 a tedious and time consuming process. For CellNet, we optimize the two parameters by grid search using the Chicago dataset in the scale L in [20, 40] and R in [50, 150]. The result had only slight variations: the lowest f-score achieved in these ranges was only slightly worse (best = 84%, worst = 75%). Optimized parameter values for the two datasets are shown in Table 7.

In order to see the importance of optimizing parameters, we tried to use the values optimized for the Chicago dataset on the Joensuu dataset directly (see 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 produces the highest f-scores. By optimizing on Joensuu data, the visual method started to work and produced the second best result. The clustering method improves by 17% (intersections) and 6% (segments) and the merging method improves by 15% (intersections) and 111% (segments). CellNet does not improve by much: 9% (intersections) and 4% (segments), however, it already produces good result without optimization (even better than other methods after optimization). This 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 on Joensuu dataset using parameters optimized using Chicago data and optimizing using Joensuu data.

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 fast when compared to the other methods because the data usually contains many overlapping routes, which are processed jointly. Their drawback is that the direction of the travel is lost in the image representation, and it 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 much slower when compared to the other approaches as shown in [Biagioni and Eriksson 2012]. CellNet running time is

moderate. The time complexity of the method is slow when dataset has high route density, or when the number of roads is high. Processing times are shown in Table 9; however, they can vary significantly when changing parameters. The times are shown for the optimized values.

Table. 9. Running time of the different methods on 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 compare the memory requirements for each of the networks in Table 10. Because of the point reduction step, the size of the network produced by CellNet is small, < 25% of any of the others. The visual method uses too many points to describe the roads; this artifact can be seen in Figure 20. The clustering and merging methods produce 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 presented 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. It works well on different route datasets without the need for time-consuming parameter optimizations. It produces higher accuracy (f-score) than three, conceptually different, state of the art methods when tested on two different real route datasets. The memory requirements of the resulting network are smaller, roughly 25% of the size of the networks generated by the other methods. The speed is 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.

M. P. Khanna. 1999. Introduction to particle physics. *PHI Learning Pvt. Ltd.*

R. Mariescu-Istodor and P. Fränti, "Grid-based method for GPS route analysis and retrieval", *ACM Trans. on Spatial Algorithms and Systems*. (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.