EFFICIENT MANAGEMENT AND SEARCH OF GPS ROUTES

Radu Mariescu-Istodor

EFFICIENT MANAGEMENT AND SEARCH OF GPS ROUTES

Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences
No 277

University of Eastern Finland
Joensuu
2017

Academic dissertation
To be presented by permission of the Faculty of Science and Forestry
for public examination in Louhela auditorium in Science Park at the
University of Eastern Finland, Joensuu, on August 25, 2017,
at 12 o'clock noon.

Grano Oy
Jyväskylä, 2017
Editors: Pertti Pasanen, Matti Vornanen,
Jukka Tuomela, Matti Tedre
Distribution: University of Eastern Finland / Sales of publications
www.uef.fi/kirjasto
ISBN: 978-952-61-2569-5 (print)
ISBN: 978-952-61-2570-1 (PDF)
ISSNL: 1798-5668
ISSN: 1798-5668
ISSN: 1798-5676 (PDF)

Author's address: Radu Mariescu-Istodor
University of Eastern Finland
School of Computing
P.O. Box 111
80101 JOENSUU, FINLAND
email: radum@cs.uef.fi

Supervisors: Professor Pasi Fränti, Ph.D.
University of Eastern Finland
School of Computing
P.O. Box 111
80101 JOENSUU, FINLAND
email: franti@cs.uef.fi

Reviewers: John Krumm, Ph.D.
Microsoft Research
Adaptive Systems and Interaction
14820 NE 36th Street, Building 99
Redmond, WA, 98052, USA
email: jckrumm@microsoft.com

Professor Hassan Karimi, Ph.D.
University of Pittsburgh
School of Computing and Information
135 North Bellefield Avenue
Pittsburgh, PA, 15260, USA
email: hkarimi@pitt.edu

Opponent: Professor Walter G. Kropatsch, Ph.D.
Vienna University of Technology
Institute of Computer Graphics and Algorithms
Pattern Recognition and Image Processing Group
Favoritenstraße 9/186-3
A-1040 Wien, Austria
email: krw@prip.tuwien.ac.at

Mariescu-Istodor, Radu
Efficient Management and Search of GPS Routes
Joensuu: University of Eastern Finland, 2017
Publications of the University of Eastern Finland
Dissertations in Forestry and Natural Sciences 2017; 277
ISBN: 978-952-61-2569-5 (print)
ISSNL: 1798-5668
ISSN: 1798-5668
ISBN: 978-952-61-2570-1 (PDF)
ISSN: 1798-5676 (PDF)

ABSTRACT

This research was focused on routes recorded using global positioning system (GPS) and examines methods of recording, storing, processing and visualizing those routes on a map. All of the methods discussed in this dissertation have been implemented in Mopsi, a location-based service with a collection of over 10,000 routes (11 million points).

I first discuss the creation of a system capable of working with a large amount of data, by applying two point-reduction methods. The two methods are cropping and polygonal approximation. These techniques allow users to load and visualize a large amount of data that would otherwise typically overload a browser.

Secondly, we used a grid to define four route measures: similarity, inclusion, novelty and noteworthiness. These measures feature in applications that deal with route search, ride-sharing and identifying taxi fraud. The similarity measure, C-SIM, allows real-time search on the Mopsi database. Our results showed that it is helpful for users who record their sports activities.

Navigation software is essential nowadays when visiting a large city. Our final contribution is CellNet, a method that uses the route database to infer the road network in an area, which is essential for navigation devices to function correctly. Using CellNet, we obtained higher quality results than those obtained by three conceptually different popular alternatives.

Universal Decimal Classification: 004.62, 004.93, 625.721, 629.052.9, 912.43

Library of Congress Subject Headings: Location-based services; Mobile geographic information systems; Global Positioning System; Orientation; Route choice; Roads; Digital maps; Information visualization; Similarity (Geometry)

Yleinen suomalainen asiasanasto: paikkatietojärjestelmät; mobiilisovellukset; satelliittipaikannus; suunnistautuminen; reitit; tiet; tieverkot; tiekartat; visualisointi

ACKNOWLEDGEMENTS

The work presented in this thesis was carried out in the Machine Learning Group at the School of Computing, University of Eastern Finland, Finland, between 2013 and 2017.

I would like to express my sincere gratitude to my supervisor, Professor Pasi Fränti, who has guided me throughout my studies. I am thankful to his continuous support and to the frequent conversations that spawned countless ideas for current and future research. I especially appreciate his dedication to doing sports, which has influenced me and has improved my overall wellbeing and ability to work.

I am grateful to the entire Machine Learning Group, especially to Andrei Tabarcea and Karol (and Katalin) Waga, who helped me to understand the concepts required in my field of study and showed me how to do good research. Special thanks also to Najlah Gali and Sami Sieranoja for commenting on my thesis.

I would like to thank my parents, Camelia and Liviu Popescu, who have supported me and my passions in life, even though this meant that I had to travel far from home. I often remember my grandmother, Constanta Istodor, and my aunt, Elena Vurfenescu, who both played a big role in raising me and forming my personality.

Lastly, I want to thank my girlfriend Iida Pirinen. I love her and all the things we do together, especially our travels – which have taught me a lot about the world. I'm also grateful to Iida's parents, Ritva and Risto, for showing me many nice things to do in Finland.

Joensuu, 23rd July 2017
Radu Mariescu-Istodor

LIST OF ABBREVIATIONS

API	Application Programming Interface
DTW	Dynamic Time Warping
EDR	Edit Distance on Real Sequence
ERP	Edit Distance with Real Penalty
GPS	Global Positioning System
LCSS	Longest Common Subsequence
MGRS	Military Grid Reference System
PNG	Portable Network Graphics
UPS	Universal Polar Stereographic
UTM	Universal Transverse Mercator
WGS	World Geodetic System
XML	Extensible Markup Language

LIST OF ORIGINAL PUBLICATIONS

This thesis is based on the following articles, referred to by the Roman Numerals I to V.

I Waga K., Tabarcea A., Mariescu-Istodor R. & Fränti P. 2013. Real Time Access to Multiple GPS Tracks, *International Conference on Web Information Systems & Technologies*, Aachen, Germany, pp. 293-299.

II Mariescu-Istodor R., Tabarcea A., Saeidi R. & Fränti P. 2014. Low complexity spatial similarity measure of GPS trajectories, *International Conference on Web Information Systems & Technologies*, Barcelona, Spain, pp. 62-69.

III Mariescu-Istodor R. & Fränti P. 2017. Grid-based method for GPS route analysis for retrieval, *ACM Transactions on Spatial Algorithms and Systems* (to appear).

IV Mariescu-Istodor R. & Fränti P. 2016. Gesture input for GPS route search, *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). Springer International Publishing*, pp. 439-449.

V Mariescu-Istodor R. & Fränti P. 2017. CellNet: Inferring road networks from GPS trajectories (submitted).

AUTHORS' CONTRIBUTIONS

I) The author implemented the system and performed the experiments together with his colleagues. The idea originated with Karol Waga. The co-authors wrote most of the paper and refined the methodology.

II) The idea originated with the author and was jointly refined through discussion with the co-authors. The author implemented the method, conducted the experiments and wrote the paper.

III) The idea was developed by both authors jointly. The author implemented the methods and performed the experiments, which were later refined by the co-author. The paper was written by the author.

IV) The idea originated with the author and was polished together with the co-author. The method was implemented by the author. The experiments and authorship of the paper were handled jointly by both authors.

V) The idea originated with the author and was later refined by the second author. Implementation and experiments were performed by the author. The two authors wrote the paper jointly.

CONTENTS

1 INTRODUCTION

In recent years, global positioning system (GPS) technology has become widely available. As a result of this increase in availability, there has been a boom in the amount of location-based data that are recorded, stored and downloaded on a daily basis. Such data include geo-tagged photos, videos, service locations and GPS trajectories. The trajectories are referred to as *routes*.

Location information often represents a point on the surface of the Earth. It is typically defined using the world geodetic system (WGS) coordinates: latitude and longitude. This information is obtained by GPS sensors available in many mobile devices nowadays, such as mobile phones, smart watches and tablets.

Location information can be used in many ways. Some examples include finding the location of lost or stolen items such as bicycles, cars and mobile phones. Pets or loved ones are also often tracked in case they go missing. Many people who play sports feel safer when sharing their location so that others know their whereabouts. Geo-tagged photo albums allow the grouping of large picture collections by location.

Sequences of GPS locations may be recorded to form routes. Various applications store, manipulate and display routes for several purposes – such as sports tracking, ski-track maintenance, vehicle tracking, fleet management, road maintenance and wildlife surveillance. Figure 1 shows some examples of these applications.

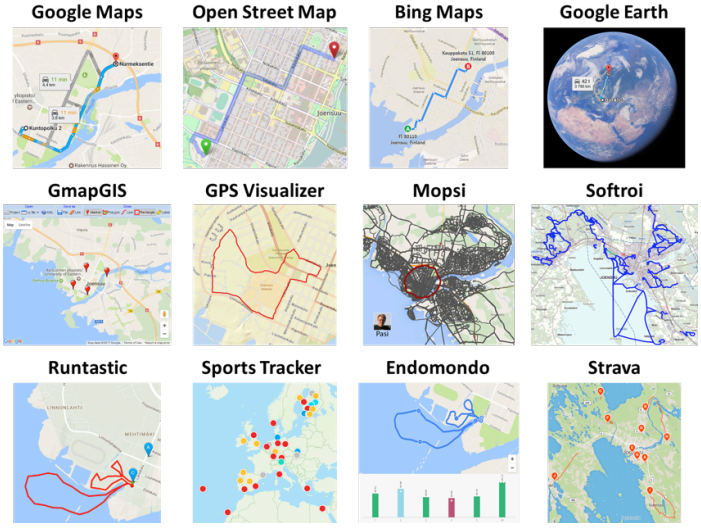


Figure 1. GPS routes displayed by various software.

Routes contain a lot of information and handling them is not trivial. In this thesis, I present efficient methods for storing, analysing, searching and visualizing routes recorded by GPS receivers on common mobile devices such as smartphones, tablets and smart watches. The methods have been implemented and tested inside Mopsi, a real-world environment. I use the concept of real-time to indicate processes which are expected to complete in less than 1 second.

1.1 MOPSI

Mopsi is a social network that helps people to discover who and what is around them. Its features include photo sharing, live tracking and chatting with friends. Mopsi can be found on the web at http://cs.uef.fi/mopsi and mobile applications exist for all major platforms (iOS, Android, Windows Phone, Symbian). They can be downloaded from the respective stores or from http://cs.uef.fi/mopsi/mobile.php.

Mopsi was developed by the Machine Learning Group, School of Computing at the University of Eastern Finland. It provides location-based services, such as search, recommendation, route tracking, geo-tagged photo collection and bus schedules. Mopsi has more than –

- 2,400 registered users
- 35,000 geo-tagged photos
- 400 points of interest
- 10,000 GPS routes.

The main topics of research to date involving Mopsi are as follows:

- route management and visualization **[I]**
- route search **[II, III, IV]**
- road network inference **[V]**
- transport mode detection [Waga et al. 2012]
- location-based recommendations [Fränti et al. 2011, Waga et al. 2011, Waga et al. 2012]
- web page summarization [Rezaei et al. 2015, Gali et al. 2015, Gali and Fränti 2016].

O-Mopsi [Tabarcea et al. 2013] is a mobile orienteering game built using data and modules in Mopsi.

1.2 RESEARCH CHALLENGES

Mobile users typically have many routes in their collection. Such route collections are difficult to manage. Certain challenges in processing routes are caused by GPS

inaccuracies, missing points, different recording intervals and varying movement speed. Another challenge is the large size; to store the routes for fast retrieval and to display them on a map is difficult without overwhelming the browser. In **[I]** we grappled with all of these challenges and provided methodological solutions.

The most popular route similarity measures are slow (quadratic time complexity) and unintuitive for the average user. Current approaches are point-based and borrow concepts from text matching, such as edit distance [Chen et al. 2005, Chen and Ng 2004], longest common subsequence [Vlachos et al. 2002] or time series analysis [Hamilton 1994, Berndt & Clifford 1994] such as dynamic time warping [Zheng and Zhou 2011]. Such point-based measures are unintuitive to typical sports tracking users, who understand routes as curves or shapes on the map. For example, to the user, a perfectly straight route consisting of 10 points is identical to the same route with 8 midpoints removed. However, the similarity as scored by the above measures is low.

Frechet [Eiter and Mannila 1994] is a similarity measure between two curves. It can be described as the minimum length of a leash an owner needs to walk a dog, when the owner travels on one curve and the dog on the other. While useful in applications such as route clustering, this type of measure is not what sports-tracking users expect. They are more interested in seeing whether the routes are recorded in the same area so they can objectively compare performances. Two routes belonging to two different users may be the same except for the start or end parts, which depend on the users' homes. Such two routes will have a low Frechet similarity even though they can be compared.

We defined a fast, linear time similarity measure called C-SIM **[III]**, which focuses on the spatial aspect of routes. C-SIM is inspired by the Jaccard set similarity coefficient; it measures the area two routes have in common as a proportion of the total space covered by the two routes. C-SIM is fast enough to allow real-time route similarity searches in large databases. To allow for fast computation, we used a grid to represent routes as sets of cells. We investigated and discussed methods of defining a good measure using the grid **[II]**.

Another challenge is searching for routes. Traditional solutions present routes as a time-ordered list or display the routes one-by-one on the map. Users often forget the date when a route was recorded. In this situation, users are forced to search in the list, one by one, to find a specific route. Showing the collection on the map successfully limits the data in the region a user is interested in; however, routes often overlap and become difficult to distinguish. We propose to use route similarity as a tool for efficient searching in large collections. As a result, users are able to search for routes based on the route shape. This shape input can be a similar route ob-

tained from the database **[III]** or a free-form shape **[IV]** drawn by the user on the map.

The final challenge we address is to automatically generate a road network using GPS routes. Current methods are based either on satellite (aerial) image analysis [Tavakoli and Rosenfeld 1982, Hu et al. 2007, Barsi and Heipke 2003] or on GPS route analysis. Many conceptually diverse methods use GPS routes; for example, route merging methods [Cao and Krumm 2009], clustering methods [Edelkamp and Schrödl 2003] and visual methods [Davies et al 2006]. These methods contain a list of parameters that need to be carefully chosen, depending on the properties of the route dataset. Optimizing these parameters is time consuming and can make a dramatic difference to the quality of the outcome, as demonstrated by Biagioni and Eriksson [2012]. Moreover, a road network generated by these methods is unnecessarily complex; relatively straight road segments have many points in their definition. To aid in these aspects, we developed a new two-step method called CellNet **[V]**. The method uses a grid to find road intersections and then connects the intersections to obtain the resulting network. It produces higher accuracy than other state-of-the-art methods. The network generated by CellNet is optimized in terms of size, requiring 75% less storage space than any other method.

All methods presented in this thesis were implemented and tested within the real datasets provided by Mopsi users. Figure 2 summarizes the components and methods used in this research.

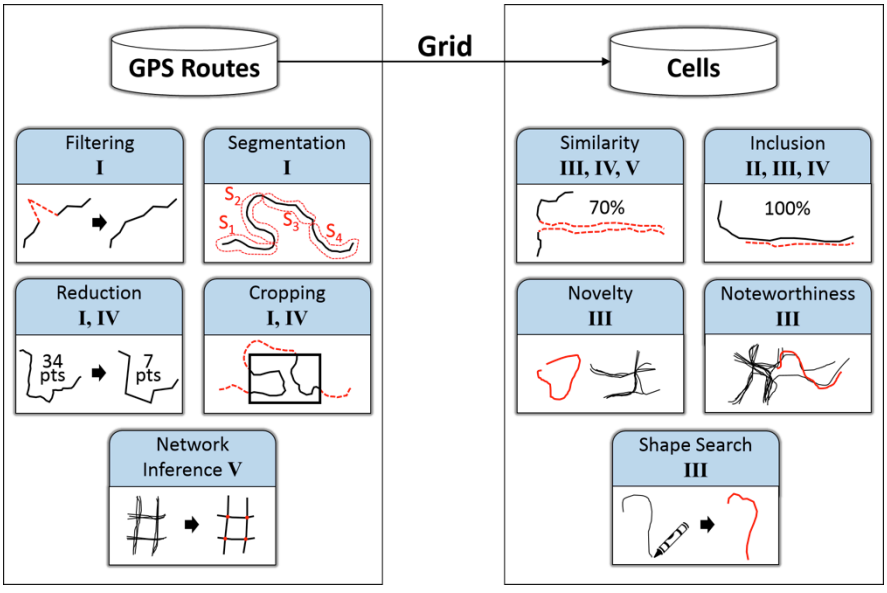


Figure 2. Different route operations available in Mopsi. They are grouped depending on whether they work with the cell approximation or with the routes themselves.

2 ROUTE HANDLING

Storing, accessing and visualizing large amounts of data on maps is computationally time-consuming. Several systems aim at providing such functionality [Alahakone and Ragavan 2009, Almer and Stelzl 2002, Follin et al. 2003, Horozov et al. 2006, Lehtimäki et al. 2008, Zheng et al. 2008]. StarTrack, described in Ananthanarayanan et al. [2009] and Haridasan et al. [2010], is most similar to the one we developed. StarTrack was tested with up to 10,000 routes. However, it does not address the problem of displaying the routes in real time; nor does it attempt to detect the transportation mode.

One of the most popular route collecting services are sport trackers, such as Sports Tracker, Endomondo, Runtastic and Strava. They allow users to record routes and evaluate their performance through comparison with past activities, or by comparing the user's performance to that of other users. In this chapter I describe current state-of-the-art methods for route management and compare these methods with the way routes are handled in Mopsi (Figure 3).

Figure 3. Mopsi user Pasi's route collection between 2008 and 2014, consisting of 915 routes with a total of 1,798,685 points. Pasi travelled a total of 11,775 kilometres, accumulating 500 hours of data. Map is centred in Joensuu, Finland.

2.1 ROUTE RECORDING

A GPS location is defined as a point **p** = {latitude, longitude, timestamp}, where the first two values represent the WGS coordinates on the surface of the planet and the last value is the unix timestamp at the moment the location is recorded. A sequence of such points forms a route $\mathbf{R} = \{\mathbf{p}_1, \mathbf{p}_2 \ldots \mathbf{p}_n\}$. The route points are usually presented in the order they were recorded in. This feature facilitates certain operations, such as drawing the points on a map.

In Mopsi, points are recorded at a fixed time interval. The interval is usually between 1 and 4 seconds but can be changed in the application settings. Recorded points are buffered in the device's internal memory and they are periodically uploaded to the server database if internet connection is available. Many applications (including Sports Tracker and Endomondo) do not allow changing this parameter, but their recording interval is usually within the same 1-to-4-second range.

2.2 ROUTE STORING

When new points are uploaded to the server, route objects are created or updated as follows. If the uploaded points are recorded within a 3-minute time period and within a distance of 1.5 km from the last point of a route, the route is updated with the new points; otherwise a new route object is created. Unlike other systems (Sports Tracker, Endomondo, Runtastic and Strava), this setup enables users to cope with battery limitations or a device or application error (requesting restart) by enabling the use of multiple devices to record a single route. Routes can be manually edited later in Mopsi, but the 3-minute, 1.5 km default segmentation is usually enough to cover typical situations. In case of such error, other applications allow manual processing – such as merging two routes – on the website. Routes can be stored efficiently using the method described by Chen et al. [2012].

Once uploaded, for every route we computed and stored the following features in a MySQL database:

- start and end point
- bounding box
- distance
- movement type
- polygonal approx.
- cell representation

(Figure 4).

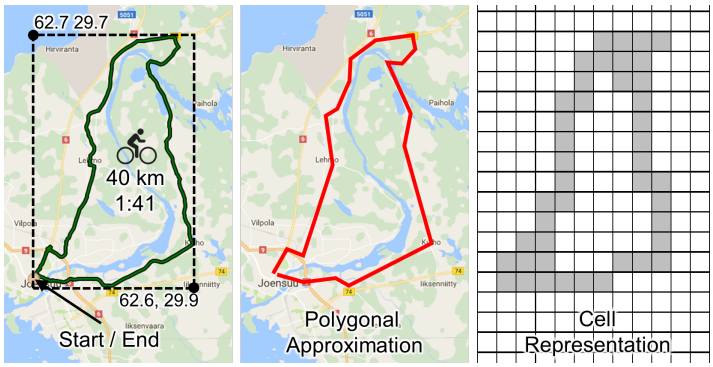


Figure 4. Example of route features from a sample route.

The MySQL table contains pointers to the files containing the route points. StarTrack uses XML files to store the route points. We used simple text files in which each row contained the latitude, longitude and timestamp. The XML file structure is useful to attach notes, photos or other information to a specific route as metadata. We stored the points in simple text files because these occupy about a third of the space relative to the XML formatting standard (see Figure 5). Additional metadata, such as the transportation mode, are stored in the MySQL database. The filtered polygonal approximated and segmented variants are also stored as files that MySQL points to.

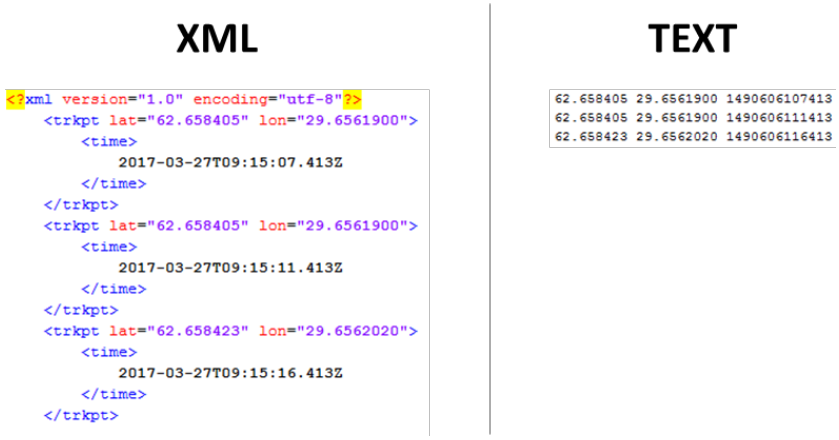


Figure 5. A route consisting of 3 points represented in XML format (left panel) and text format (right panel). The XML file contains 335 characters whereas the text file has only 104.

2.3 ROUTE ANALYSING

In Mopsi, any route can be analysed. Segmentation is performed together with automatic transport mode detection for each segment [Waga et al. 2012]. Each segment is coloured differently according to the transportation mode. All popular sports tracking programs allow to segment using a fixed length or duration. In Mopsi, the segments are defined in a way that minimizes the speed variance of each segment. Such segments are useful when displaying routes with multiple transportation modes. Figure 6 shows that the running segment in the middle is perfectly isolated and accurate statistics can be viewed for only that section. In Runtastic it is only possible to segment the route at fixed length or duration, therefore, the running portion falls under the fourth and fifth segments making it difficult to interpret.

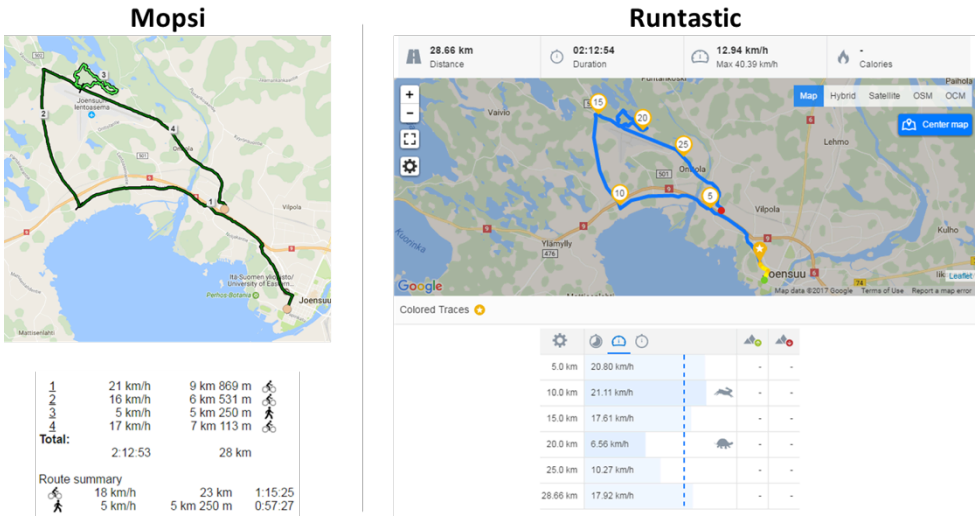


Figure 6. A route that features cycling, orienteering and cycling, shown in Mopsi (left panel) and Runtastic (right panel). The orienteering (running) segment is separated from the cycling segments in Mopsi.

Other features such as stop detection, roundness computation and showing photos taken by users on the route are also available in the analysis screen. Routes are filtered to remove outlier points caused by fluctuations in GPS accuracy. These points are computationally simple to identify and remove because they typically deviate away from the actual location, causing an impossibly high speed.

Mopsi lacks certain popular features, such as calorie and power output display, which other sports tracking software collects through additional pieces of hardware connected to the user's body or bicycle.

2.4 ROUTE VISUALIZING

We accessed routes in Mopsi by selecting a user and a time period (Figure 7). The time interval can be chosen to show the most recent day or last week, last month or last year's activity; showing the entire collection or choosing a user-defined time interval is also supported. This is efficiently done by querying the timestamp of the start point of the routes. This feature is available in all sports tracking software. Other programs (Sports Tracker and Runtastic) also allow the user to search or group the results based on transportation mode, distance travelled and duration.

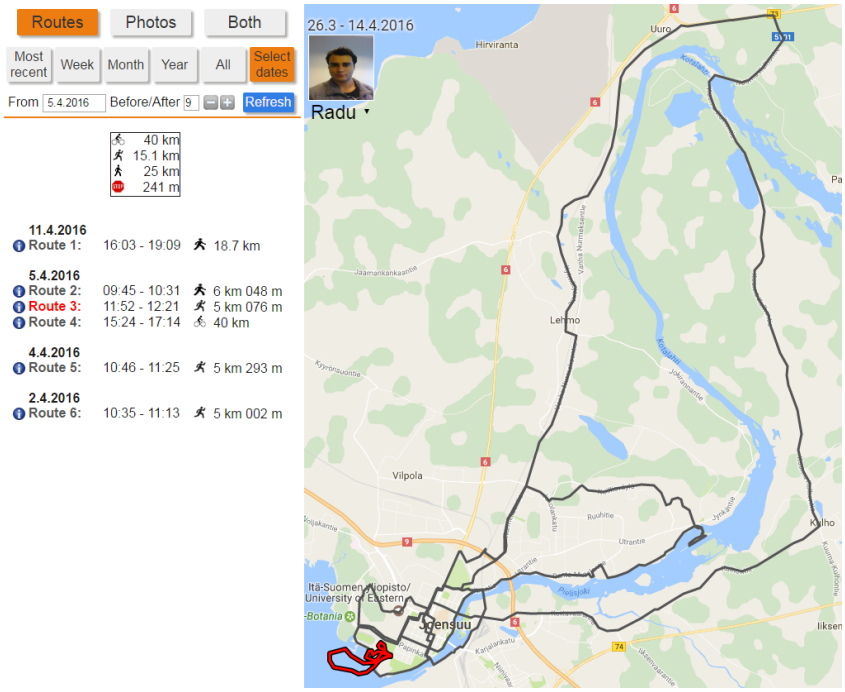


Figure 7. Mopsi tracking activity of user Radu over a few days. The entire route shapes are preserved. Routes are clickable inside the list or on the map.

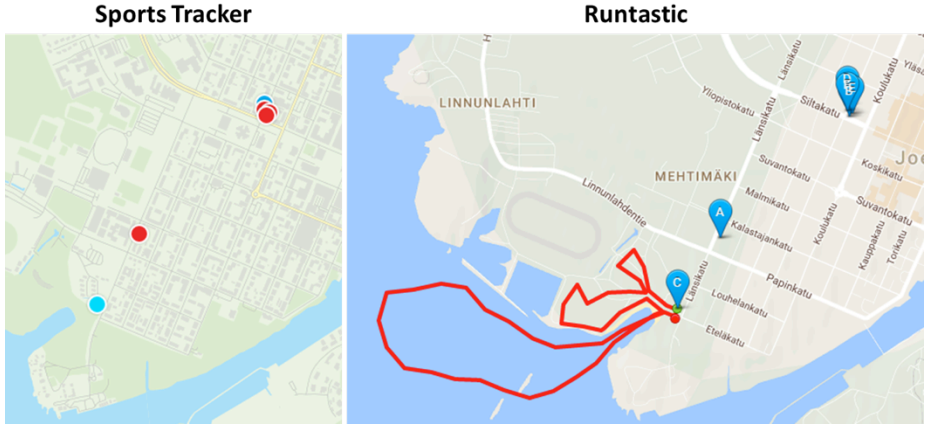


Figure 8. The same route collection as in Figure 7, displayed by two sports tracking programs; routes are represented by their start points. Sports Tracker colours the points based on the transportation mode. Runtastic can display one route at a time when clicking on the start point marker.

In Mopsi, the selected route collection is presented in two ways: in a list and on the map. The list is ordered with respect to time. Routes recorded on the same day are grouped together to enhance usability. On the map, routes are shown as grey lines. A route can be selected by clicking it on the map or in the list, and a selected route

is highlighted in red. The movement type and distance are shown for each route in the list. A summary of the data collection per transportation mode is shown at the top of the list.

Visualizing route collections on the map is a difficult task because of the large number of points they contain. Therefore, many other applications lack this feature and can display only one selected route at a time (Table 1). The map system used is typically Google Maps[1] or Open Street Map[2] (OSM). Runtastic also uses Open Cycle Map[3] (OCM). Mopsi allows for the display of specially designed orienteering maps as well, provided by Kalevan Rasti[4], as overlays on top of the Google map.

Table 1. Features of popular sports tracking applications.

	Mopsi	**Endomondo**	**Runtastic**	**Sports Tracker**	**Strava**
Map type	Google OSM Kalevanrasti	Google	Google OSM OCM	OSM	Google OSM
Displays collection	✓		✓	✓	
Displays single route	✓	✓	✓	✓	✓

To avoid overwhelming the map, some applications show the starting locations only (Figure 8). In contrast, Mopsi can display large route collections by showing the full shape of the routes, which enables users to better understand their collection (Figure 7). In this way, a user can see at a glance that the collection is not limited to the city centre, as suggested by Sports Tracker and Runtastic; several routes actually pass through different towns. When displaying collections, the problem of overlapping route segments also becomes apparent. It is common that the starting points of routes overlap near the user's home or workplace. Clustering these start points could help to improve the user's experience [Rezaei and Fränti, 2017].

Our solution is to limit the route points using two strategies. First, when users browse a collection on the map, they need to see the overall shape of the route but not every detail. The shape can be well preserved by applying polygonal approximation [Chen et al. 2012], which reduces the number of points. We used approximations with varying reduction levels, which are used at different zoom levels of the map (Figure 9).

[1] https://www.google.fi/maps
[2] http://www.openstreetmap.org
[3] https://www.opencyclemap.org
[4] http://wp.kalevanrasti.fi

Figure 9. Polygonal approximation at three different levels. The 10-point approximation is suitable for the current zoom level. Original route has 110 points.

The second strategy is to crop the collection according to the screen boundary. Only points within the current map borders are loaded, together with an immediate neighbourhood (50% extension of screen size). This allows panning the map by a small amount without the need to reload new data (Figure 10). The cropping process is hidden from the user and does not interfere with usability.

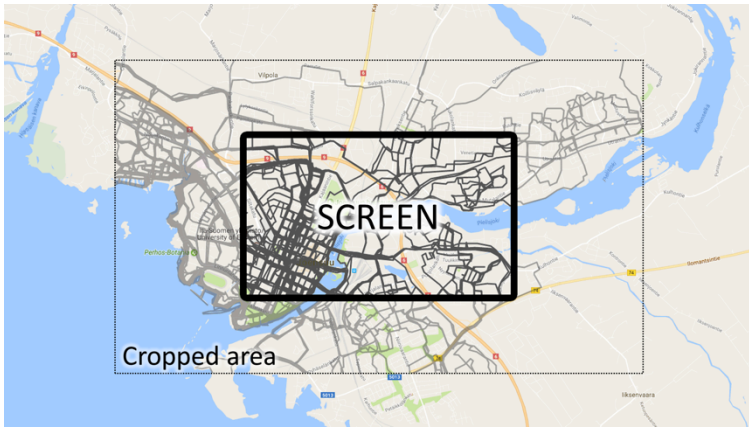


Figure 10. Cropping of a route collection. Only route segments inside the screen area and screen neighbourhood are plotted on the map.

An approach by Morris et al. [2004] aims to minimize the data displayed by combining the route segments that overlap. In Mopsi, we avoid this solution so that users are allowed to interact with each individual route by clicking it on the map.

To understand how effective the cropping and reduction process is for limiting the amount of data, we first investigated three highly active Mopsi users and their route collections. The data are shown in Table 2.

Table 2. Tracking statistics for 3 active users in Mopsi, grouped by time period.

User		Week	Month	Year	All
Pasi	routes	3	17	230	1,704
	points	2,030	43,635	548,379	3,648,923
	length (km)	33	230	8,040	27,384
	time (hours)	3	21	306	2,030
	size (MB)	221	1,719	22	145
Radu	routes	2	13	82	1,235
	points	650	14,376	100,542	1,383,318
	length (km)	8	140	1,258	23,138
	time (hours)	0.7	15	114	1,034
	size (MB)	24	566	3.8	53
Matti	routes	9	14	148	412
	points	2,086	5,875	98,237	293,207
	length (km)	39	90	1,642	4,160
	time (hours)	3	8	138	350
	size (MB)	74	210	3.5	11

Figure 11 illustrates the process of querying and displaying these collections in full, without any reduction. Figure 12 shows the process with polygonal approximation and cropping applied. We disregarded the time required to download the points from the server, as this duration varies depending on factors such as internet speed and bandwidth. For the display, we used the zoom level that allowed all routes to be visible on the map.

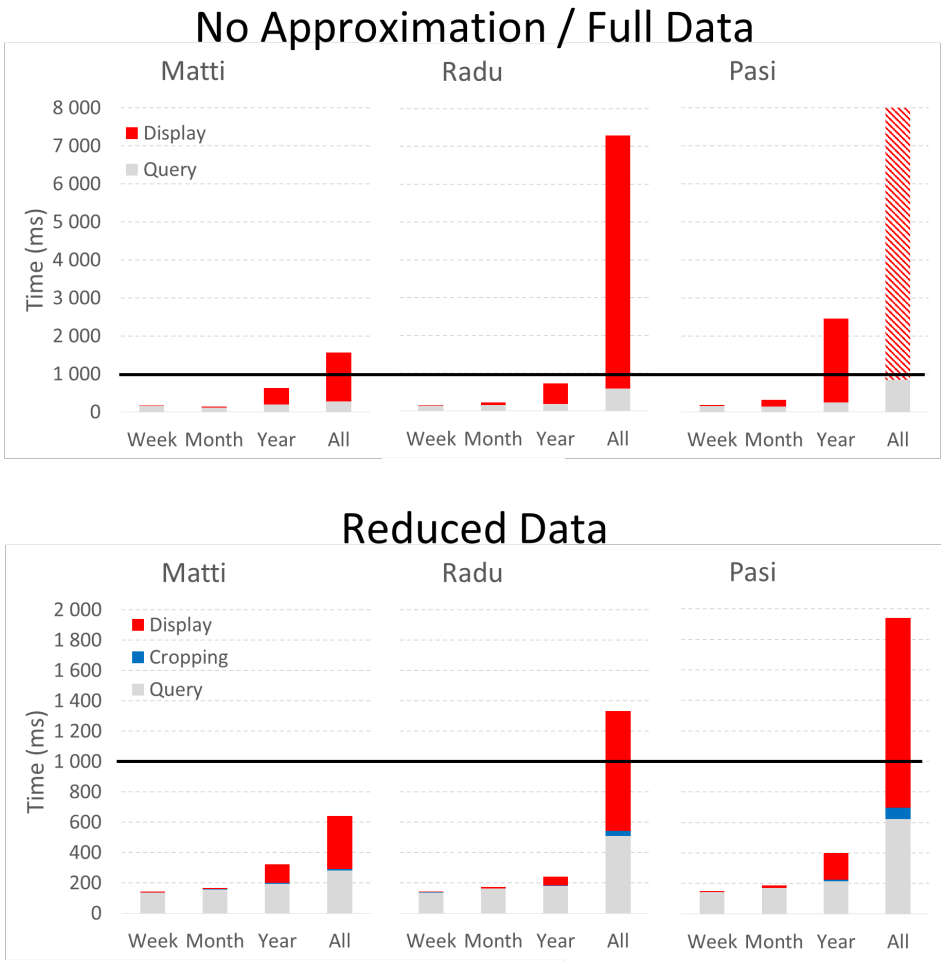


Figure 11. Query and display times for route collections of varying sizes, obtained for users Matti, Radu and Pasi. Pasi's entire collection crashed the browser.

The display time was most affected by reducing the number of points. In fact, the browser crashed when trying to display the entire data for user Pasi if no reduction was applied. After polygonal approximation, the points were reduced as shown in Table 3. We used five different approximations at different map zoom levels. In the experiment illustrated in Figure 11, R1 was used because it allowed all routes to be seen on the map. At this zoom level only ~1% of the points were required to preserve the route shape, making the download time a fraction of that needed for the unreduced data. The query processing time was 6% faster. This is because each route is stored in separate files, and forming a collection requires accessing multiple files. The process can be slow as it requires relocation of the read–write head of the hard-disk. The files are not large, therefore the required time depends mainly on the number of routes rather than the number of points. The cropping process is also

performed; however, the map shows all routes in this experiment, meaning that cropping would not be effective at all.

Table 3. Effectiveness of polygonal approximation.

User	R1	R2	R3	R4	R5
Pasi	0,8%	2%	4%	9%	22%
Radu	0,9	2%	4%	9%	21%
Matti	1,5%	3%	5%	15%	50%

Note: The values are measured as the proportion of points remaining after reducing all routes of a user. Values are shown for five different reduction levels (R1 to R5).

Table 3 shows that the efficiency of the reduction was similar for users Pasi and Radu. However, for Matti the reduction was less effective, especially at the higher zoom levels. Matti uses Android whereas Pasi and Radu use iPhone and Windows Phone respectively. Most Android devices do not represent GPS coordinates with sufficient accuracy, resulting in a zig-zag effect, as illustrated in Figure 12. The Android route therefore requires more points to be represented accurately.

Figure 12. Two walking routes recorded on different sides of a street. The top route was recorded using Windows Phone. The lower route was recorded using an Android device, which uses lower precision to represent coordinates.

The cropping step works in linear time with respect to the number of points in a collection. Using the polygonal approximation first causes the cropping step to process less data (Figure 13).

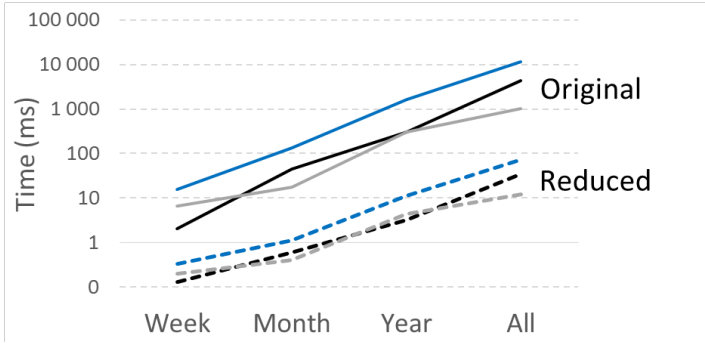


Figure 13. The speed of the cropping process on the original route (solid line) and reduced route (dotted line), for each of the three users: Matti (grey), Radu (black) and Pasi (blue).

After selecting the routes, the user can continue to look around by panning and zooming the map. Only the cropping and displaying operations are performed at this stage. To see how effective the system was in this situation, we performed the following experiment. For each of the three users, we loaded the entire route collection. Then we programmatically panned the map by matching the screen borders to the bounding box of every route. We recorded the number of points and the time required for the processing and the display. This experiment was designed to stress our method, by simulating a user moving the map to see the different regions where he or she expected to find routes.

The results are presented in Table 4, which shows that analysing a data collection requires to load on average, 1% to 5% of the data. Although Matti's collection was far smaller than Radu's, a similar amount of data was retrieved (on average). This is because Radu was recording routes in different cities, whereas Matti's data was mostly obtained in two cities, Kuopio and Tampere. This discrepancy resulted in their data density being roughly the same when zooming at the city level. The cropping time complexity was linear with respect to the total number of points. The time required by Google Map to display the points appears to be linear with respect to the number of points after cropping.

Table 4. Average amount of data and processing times when moving the map.

	Data		Processing Time	
User	**Size (KB)**	**Points**	**Cropping (ms)**	**Display (ms)**
Pasi	1,144	56,827 (2%)	407	210
Radu	325	15,989 (1%)	241	141
Matti	300	14,901 (5%)	93	128

To give more meaning to the amount of data being transferred, we compared the amount of data loaded by Google Maps to show the map tiles. Tiles were portable network graphic (PNG) images, typically 256 x 256 pixels, and their size varied considerably depending on the amount of information displayed. We recorded the size of each tile loaded in the panning experiment described earlier and the average was 10.5 KB per tile. A screen of 1920 x 1200 can load 36 tiles, equivalent to about 378 KB. This value is comparable to the amount of route data loaded when panning the map to browse Radu's and Matti's collections. To load Pasi's route data meant loading roughly three times the amount of data contained in the map tiles.

Using reduction and cropping not only improved the speed but also prevented the browser from crashing. For example, loading Pasi's entire collection without applying any reduction or cropping caused the browser to crash. Online utilities, such as GPSVisualizer[5], GmapGIS[6] and many sports tracking programs – which also use Google Maps (see Table 1) – are incapable of displaying these data, as they lack a similar data-reduction strategy.

[5] http://www.gpsvisualizer.com
[6] http://www.gmapgis.com

3 GRID-BASED OPERATIONS

Using a grid, we defined four route operations that were useful for solving different problems. These operations were:

- Similarity
- Inclusion
- Novelty
- Noteworthiness.

Similarity is probably the most common operation performed on routes. For instance, Ying et al. [2010] demonstrated that meaningful friend recommendations can be issued in social networks by analysing users' similar routes. Another case where route similarity is helpful is when giving trip recommendations. In Shang et al. [2012] a route is recommended when a set of intended places and textual attributes that describe the user's preferences is given as input. The similarity measure has also been used successfully to identify ideal places to build new bicycle paths [Evans et al. 2013]. Route similarity is used as an inverse distance function for clustering applications [Pelekis et al. 2010, McCullough et al. 2011, Ying et al. 2009] in various applications – for instance, to identify traffic congestion.

Finding similar route(s), also known as "k nearest neighbour search" in a database, is the most typical use for the similarity operation [Agrawal et al. 1993, Frentzos et al. 2007, Ni and Ravishankar 2007, Wang and Liu 2012, Yanagisawa et al. 2003]. In Mopsi, this feature enables users to find a similar route recorded in the past in order to compare the routes in terms of speed. The feature also allows comparison with the data of other users who have recorded similar routes.

Many measures for computing route similarity exist:

- *longest common subsequence* (LCSS) [Vlachos et al. 2002]
- *edit distance on real sequence* (EDR) [Chen et al. 2005]
- *dynamic time warping* (DTW) [Zheng and Zhou 2011]
- *edit distance with real penalty* (ERP) [Chen and Ng 2004]
- *Hausdorff distance* [Rockafellar and Wets 2009, Chen et al. 2011]
- *Frechet distance* [Eiter and Mannila 1994].

These measures typically require quadratic time to be computed. Some approximate and more complicated variants exist, such as *FastDTW* [Salvador and Chan 2004]. *Euclidean distance* (*L2-norm*) [Gradshteyn and Ryzhik 2000] is an example of a simple linear time approach to compute route similarity; however, the method

works well only if routes are aligned at their start and are of similar length. This degree of congruence happens rarely in a real database.

We considered two routes to be similar if they overlapped. The amount of overlap measured how similar the routes were. We defined a fast and linear time similarity measure (C-SIM), which focuses on the spatial aspect of the routes. C-SIM is inspired by the Jaccard set similarity coefficient, and measures the amount two routes have in common divided by the total space covered by the two routes. This space is measured by counting the number of distinct cells that the routes are passing through. The dilated region of each route is also obtained by using a 3×3 structural element on the original route cells. This dilated region is necessary to compensate for the arbitrary division of the grid, which might separate nearly identical routes that happen to be on different sides of a cell border. C-SIM is fast enough to allow real-time route similarity searches in large databases. The equation is:

$$S(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B{}^d| + |C_B \cap C_A{}^d|}{|C_A| + |C_B| - |C_A \cap C_B|}, \quad (1)$$

where C_A and C_B are the cell representations of two routes. $C_A{}^d$ and $C_B{}^d$ are the dilated regions of the two routes respectively.

The second operation is *Inclusion*. It measures how much one route is contained inside the other. The equation is:

$$I(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B{}^d|}{|C_A|}, \quad (2)$$

where C_A and C_B are the cell representations of two routes. $C_B{}^d$ is the dilated region of the second route.

Unlike similarity, inclusion is not symmetric. The measure is useful for solving drive-sharing problems, by identifying users who –

- can pick up somebody along the user's route, or
- can be picked up by somebody else on their route.

In Mopsi, inclusion is used to search for routes that pass through a region manually specified by the user on the map **[IV]**.

Novelty measures the amount of unique parts of a route compared with other routes in a database. Novelty can be useful in several applications. For instance, it may be

considered an alternative to iBAT [Zhang et al. 2011] with regard to identifying taxi fraud, namely when a taxi driver takes a longer route than necessary to arrive at the destination. The given route is compared with other past routes that start and end at the same locations. If the new route has a high novelty measure, the route is labelled as fraudulent. Alternatively taxi driver safety can be addressed as in [Karimi and Lockhart 1993]. Another application for novelty is to automatically update GPS navigation systems that exist in many cars nowadays. If a recent route shows novelty compared with the existing road network, the roads in the region have changed; in such instances, the database updating methods described by Fathi and Krumm [2010] and Cao and Krumm [2009] should be applied [**V**].

Noteworthiness is closely related to novelty. It measures the amount of rarely visited parts instead of focusing only on unique parts. This measure is useful in places that have a high density of routes that have extremely few novel regions. In Mopsi, novelty and noteworthiness are used to inform users when their route passes through places they have never visited before. It is also verified whether other users have frequented the area (Figure 14).

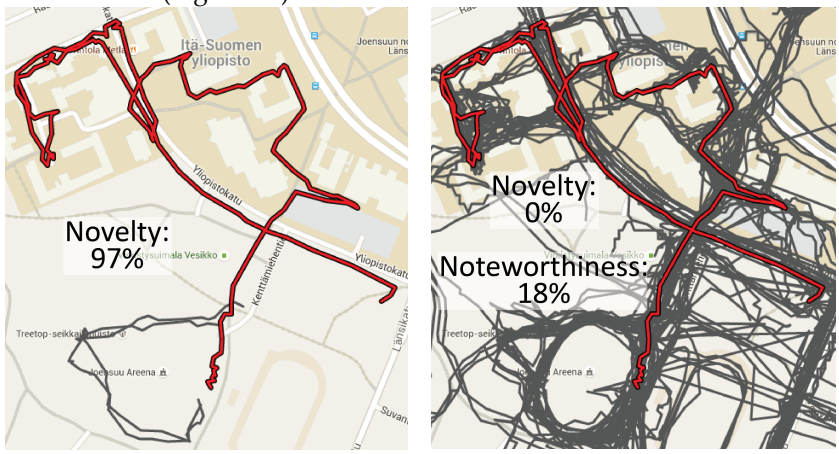


Figure 14. A route is 97% novel to the user (left panel). The same route is not novel at all with respect to all users (right panel), but 18% of the route is noteworthy. The selected route is shown in red and other routes in the collection are grey.

3.1 GRID

Grids have been used to represent geographical data in past studies. In Pang et al. [2012] and [Zhang et al. [2011], grids were used to find patterns in taxi data. In Wei et al. [2012], popular routes were constructed using the frequency information of grid cells. In Zheng et al. [2010] and Bao et al. [2012] the grid was used to infer stay areas, which in turn are used to detect points of interest. In Krumm and Horvitz [2006] grids were shown to be useful when predicting the destination of moving vehicles.

The abovementioned examples used grids to perform frequency analysis in sub-regions of a given area. We extended the use of grids to define a similarity measure between routes and to perform similarity-based retrieval in route databases. To enable this, we required a grid with equal cell size spanning the entire planet, which is not trivial to do [Kennedy and Kopp 2001]. The existing applications create grids by segmenting the latitude and longitude values [Bao et al. 2012] for which the cells gradually change size when one moves in the north–south direction. Another way grids have been defined is by focusing only on a small region, such as a city – as in Zhang et al. [2011], Krumm and Horvitz [2006], Pang et al. [2012] and Wei et al. [2012] – and dividing that region into equal-sized cells. When computing the similarity of routes, the grid needs to be finer than for other applications, which typically use cell sizes in the scale of 100 m to 1 km.

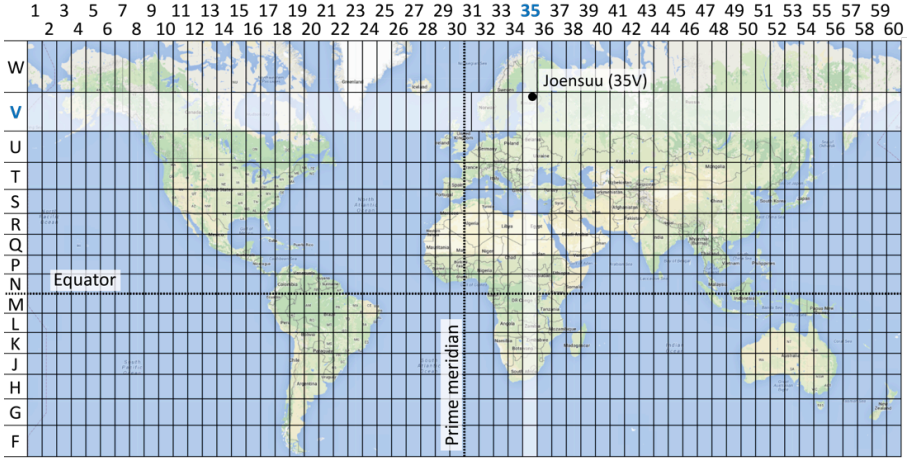


Figure 15. MGRS grid zones. Joensuu is in UTM zone 35 and latitude band V.

To generate the grid, we used the military grid reference system (MGRS[7]), in which cells of equal size fill up specially defined zones that cover the entire planet. These zones do not usually follow the north-south orientation. This aspect allows the zones to wrap around the planet. Then, each zone is divided into cells of the same size in square metres.

MGRS is an alpha–numeric two-dimensional coordinate system in which locations are identified independent of their elevation. MGRS divides Earth into projection zones and computes easting and northing in metres, within a designated zone. The Universal Transverse Mercator (UTM) is used to divide the planet into 60 zones, each being 6° of longitude wide. For the polar regions (above 84°N and below 80°S),

[7] http://builds.worldwind.arc.nasa.gov/worldwind-releases/1.4/docs/api/overview-summary.html

the Universal Polar Stereographic (UPS) convention is used instead of UTM. For the perpendicular segmentation, bands of latitude (8° high) are used.

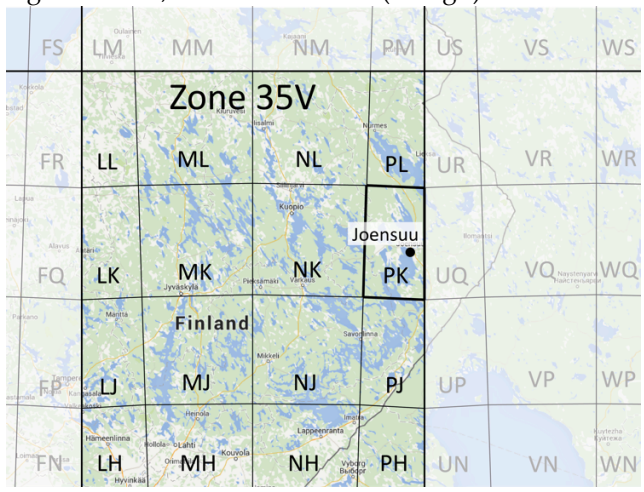


Figure 16. 100 kilometre squares. Joensuu is in square PK of zone 35V.

The first three characters of the MGRS value for the city of Joensuu, Finland, are 35V (Figure 15). The next pair of characters identifies a 100 km × 100 km square within each of the grid zones. Joensuu is located in region 35VPK (Figure 16). The remaining part consists of numeric easting and northing values within the 100-km square. MGRS allows one of five predefined precision levels when choosing the cell length: 1 m, 10 m, 100 m, 1 km or 10 km. However, any specific degree of precision can easily be obtained if the desired cell length can be perfectly divided into 100,000 (metres). Limited by the average GPS error, we chose a 25 m × 25 m cell size. As shown in Figure 17, we identified the centre of a small park as being 35VPK16461774.

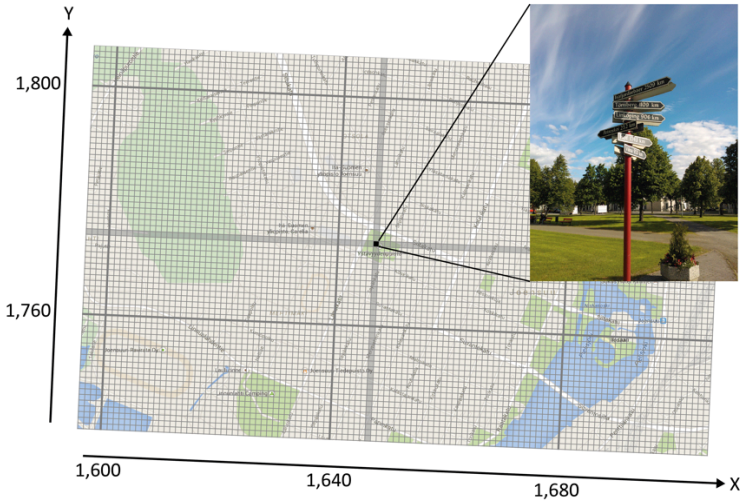


Figure 17. 25 m x 25 m cells in the Ystävyydenpuisto (Freedom Park) in Joensuu.

3.2 EVALUATION

We evaluated our proposed grid-based similarity measure, C-SIM, to see how it compares with other approaches. We used the Mopsi2014[8] dataset, which is a subset of all routes in the Mopsi database collected between 2008 and 2014. It contains 6,779 routes recorded by 51 users having 10 or more routes (Table 5). Some users have more data than others (Figure 18).

Table 5. Mopsi2014 dataset summary.

Routes	Points	Kilometres	Hours
6,779	7,850,387	87,851	4,504

The dataset consists of routes with a wide range of activities, including walking, cycling, hiking, jogging, orienteering, skiing, driving, and travelling by bus, train, or boat. Even though such ground truth is not available, using the method of Waga et al. [2012] we automatically computed the movement types and showed a distribution of these activities by transportation mode (Figure 19). Routes exist on every continent except Antarctica allowing a good test for MGRS, which seems to work well in all regions where test data is available. Most routes are in Finland, in the city of Joensuu, which creates a very dense area suitable for extensive testing of the methods.

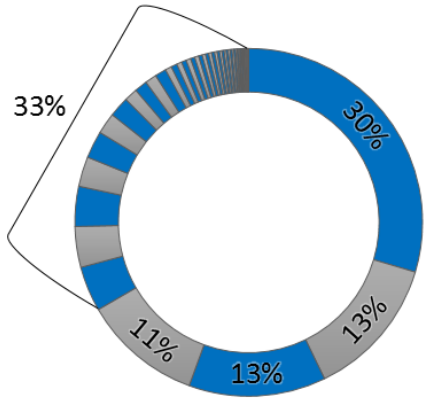


Figure 18. The distribution of the data per user. The four most active users had recorded approximately two thirds of the data.

We computed a 25 m × 25 m cell representation for all 6,779 routes using MGRS. The cell size was decided based on experimentation and by observing typical GPS inaccuracies. The cell database entries included cells obtained from interpolation and dilation [**II**], which are required for the operations. Statistics are shown in Table

[8] http://cs.uef.fi/mopsi/routes/dataset

6. Typically, point databases are indexed by using tree structures such as R-tree [Guttman 1948] to make range queries possible. As a comparison, if R-tree is applied, Mopsi2014 would require approximately 1 GB of space. Roughly the same space is required by the cell database when indexed using B-tree [Cormen 2009]. In **[III]** we showed that the Hash index [Cormen 2009] can also be used and is about 50% faster than the B-tree index, with a 40% increase in memory requirements.

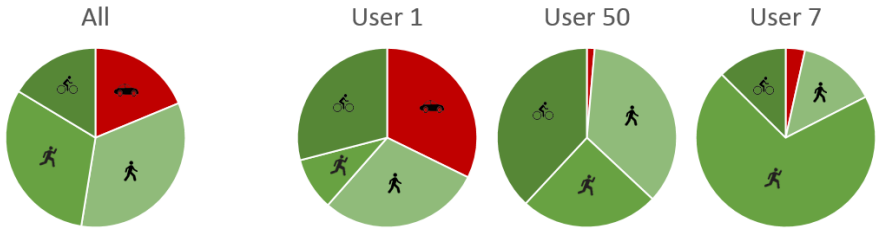


Figure 19. The distribution of all walking, running, cycling and car routes in Mopsi2014 dataset. The distributions for three sample users are also shown.

Table 6. Space requirements for Mopsi2014[9] dataset.

	Entries	Index	Total
Point Database	7,850,387 (329 MB)	R-tree (650 MB)	979 MB
Cell Database	11,477,506 (525 MB)	B-tree (429 MB)	954 MB
		Hash (788 MB)	1,313 MB

We investigated how various route similarity measures are affected by the following transformations:

- increasing sampling rate (adding points)
- decreasing sampling rate (removing points)
- adding noise
- random shifting of points
- synchronized shifting of points.

We extended the evaluation performed by Want et al. [2013] by adding C-SIM and a few other similarity measures to the comparison. We selected 1,000 random routes from Mopsi2014, and analysed the behaviour of the similarity measures when the five artificial transformations were applied. We assumed that these transformations might occur naturally in a route database due to the use of different devices, varying GPS accuracy and other influences. Therefore, the similarity between the transformed route and the original was expected to be high (100%); alternatively, the distance should be 0 for distance-based measures. The trends for the similarity measures are illustrated in Figure 20.

[9] http://cs.uef.fi/mopsi/routes/dataset

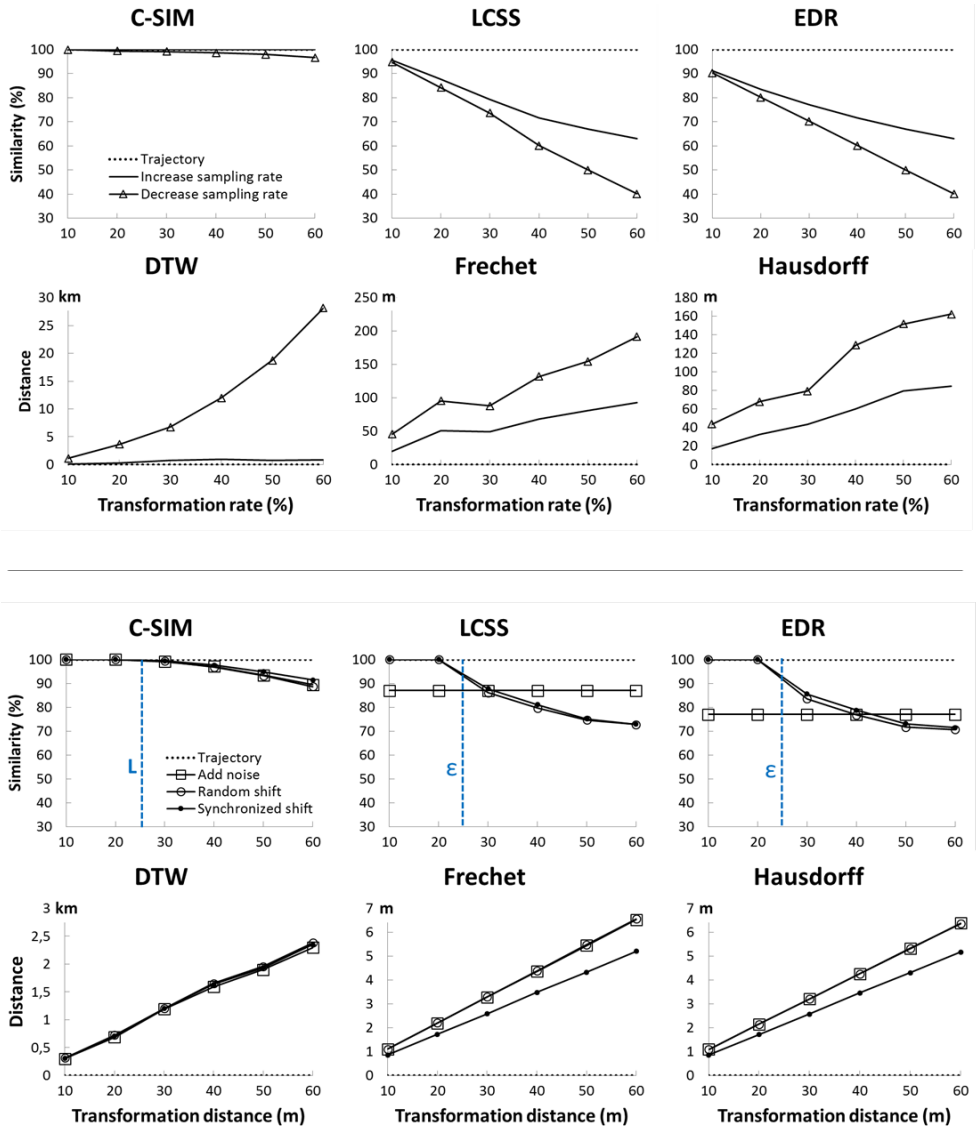


Figure 20. Six route similarity measures affected by sampling rate transformations (upper panel) and by noise and point shifting (lower panel).

C-SIM performed well when points were added or removed. The measure is not affected by increasing the sampling rate, because the cell representation remains identical due to the interpolation step. Decreasing the sampling rate had a minor effect on similarity, because of the inability of interpolation to correctly predict the missing parts of the route. However, the effect was far smaller than that of the other

methods. Among the measures, LCSS and EDR were the most sensitive to a decreased sampling rate, although an increase in sampling rate had a milder effect. The C-SIM, LCSS and EDR measures are not affected by point shifting if the transformation distance is small (L = ε = 25 m, in our experiments). For higher distances, C-SIM decreases in proportion to the transformation distance. LCSS and EDR similarities do not decrease proportionally to the distance; ε is a threshold when two points are considered identical. The similarity is higher when transformation distance is small, but will be above ε because points shifted only little more than ε metres away are still likely to match with other points in the vicinity. DTW did not vary with an increase of the sampling rate but was highly sensitive to a decrease. Hausdorff and Frechet were both sensitive to changes in sampling rate.

Table 7. Summary of the effectiveness of the 6 route similarity measures.

Function	Sampling rate		Add noise	Point shifting	
	Increase	Decrease		Random	Sync.
C-SIM	Robust	Robust	Fair	Fair	Fair
LCSS	Sensitive	Fair	Sensitive	Fair	Fair
EDR	Sensitive	Fair	Sensitive	Fair	Fair
DTW	Robust	Sensitive	Sensitive	Sensitive	Sensitive
Hausdorf	Sensitive	Sensitive	Sensitive	Sensitive	Fair
Frechet	Sensitive	Sensitive	Sensitive	Sensitive	Fair

Noise affected LCSS and EDR more than the other measures because it caused a change in the length of a transformed trajectory. DTW was sensitive to all transformations. Frechet and Hausdorff were sensitive to noise and point shifting, but less so if the points were shifted in the same direction (synchronized). The similarity depends linearly on the transformation distance. The results are summarized in Table 7.

4 ROUTE SEARCH METHODS

Routes can be searched for various reasons, such as finding, comparing and reviewing:

- find a past route in order to compare any progress
- compare one's effort with that of others on a similar track
- review statistics of a route recorded in the past
- memorize a specific tour to make revisiting a place easy

In large collections, finding a specific route is difficult. Traditionally, sports tracking applications offer a time-ordered listing and/or map plotting of the collection. Recently, thumbnail listing and segment-based searches have also become supported by certain applications. We introduced two novel means of searching for routes: similarity search **[III]** and gesture search **[II]**. These approaches are discussed in greater detail here.

4.1 TIME-ORDERED SEARCH

All sports tracking applications offer some kind of time-based ordering of a route collection. The options to display the information are a calendar, a list and – more recently – a list with route thumbnails (see Figure 21 and Table 8).

The *calendar* is familiar and intuitive to many users; however, it can show numerous empty locations, meaning the user must perform many clicks to access the data. In addition, calendars impose a limit on the number of activities per day. The calendar is large and wide and it cannot coexist with a map on a typical screen.

The *list* is more useful because it contains no blanks. In Mopsi, list items are grouped by the date. The duration, movement type and distance are shown. Other applications often include additional information, such as calorie burning and power output. In Mopsi the user lacks access to this information, which typically requires separate hardware in addition to the mobile phone.

Both the calendar and list formats have a weakness when searching for routes. They do not show the shape of the route although shape is a feature that users easily recognize. For this reason, all major applications now show a thumbnail list, which provides a greater amount of information but with the drawback that the list becomes longer. If the date is unknown, these methods are no longer useful and would imply sequential searching through every item in the list.

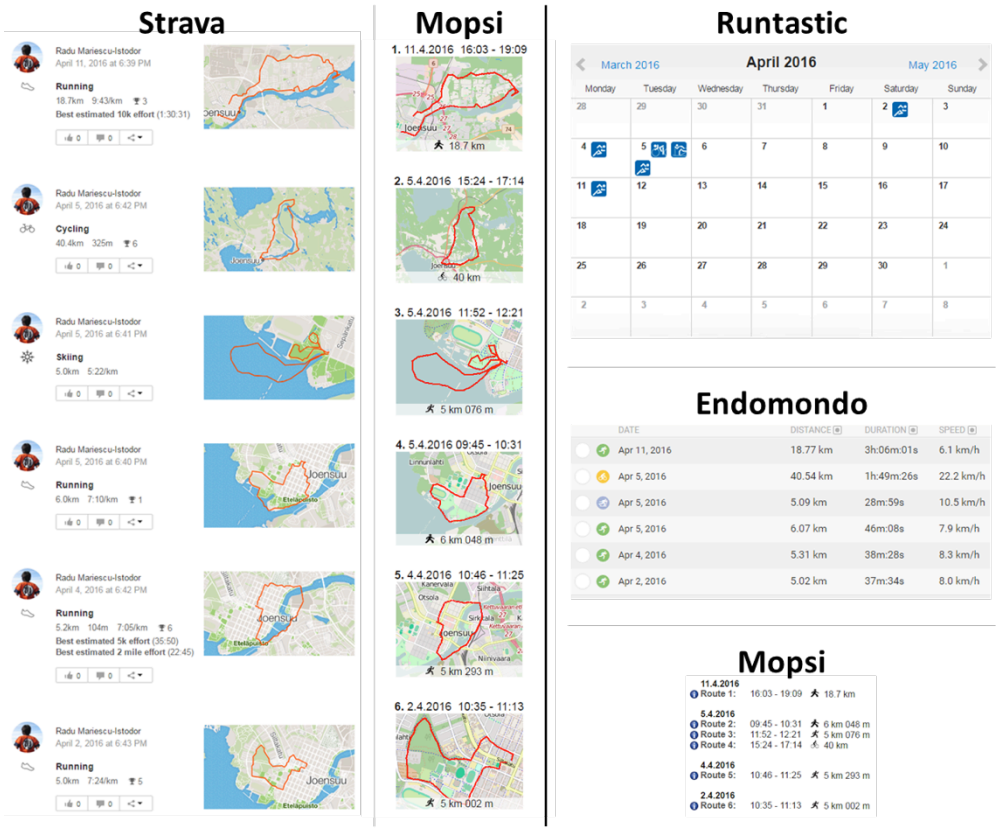


Figure 21. The same route collection visualized in different ways. Strava and Mopsi demonstrate the thumbnail view (left panel), Runtastic shows the calendar view (top right panel), and Endomondo and Mopsi show the list view (lower right panels).

Table 8. Time-based route visualization methods and their availability in sports tracking software.

	Mopsi	Endomondo	Runtastic	Sports Tracker	Strava
List	✓	✓	✓	✓	
Calendar		✓	✓	✓	✓
Thumbnails	✓	✓	✓	✓	✓

4.2 MAP-BASED SEARCH

Some applications show the collection on the map. In this way, routes can be identified by their location (Figure 22). The Sports Tracker application represents routes by their starting points so that the map is not overwhelmed by too many points. The route representatives are coloured with respect to their transportation mode. The disadvantage of this method is that it hides much of the information. Also, typically routes start at the same location – the user's home – for activities such as cycling and running. This commonality makes the three running routes hard to distinguish. In Mopsi, the entire route shapes are shown. The amount of data is minimized using the reduction. Problems occur if routes overlap so much that they become indistinguishable.

The benefit is that routes can be identified quickly, unless there is a massive amount of data for the region. In the latter scenario, the data should first be limited based on time.

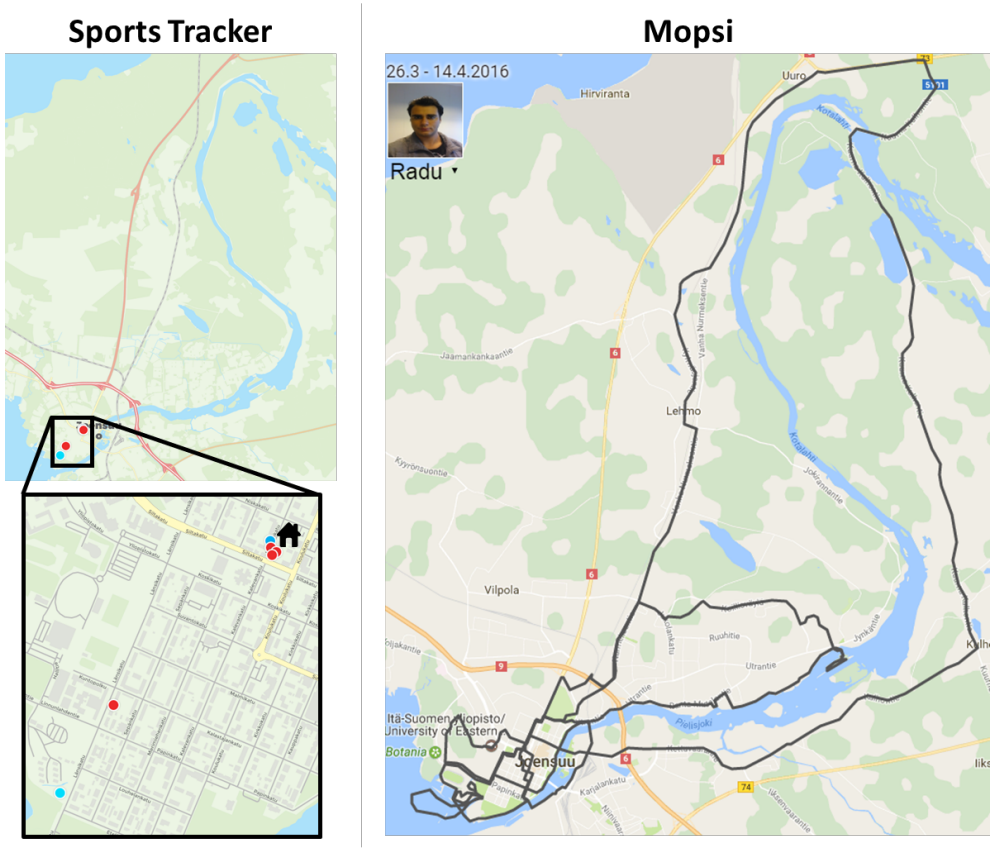


Figure 22. The same route collection displayed on a map by Sports Tracker (left panel) and Mopsi (right panel).

Strava does not display route collections on a map, but it does display a collection of user-defined segments of interest (Figure 23). This application provides an easy way for people to compete with others. The segments are manually defined by users via their start and end points and by the intermediate locations which can be selected from the user's routes. Once a segment is defined, users passing through that area will be clocked and ranked in a list, providing another way to search for routes. Because segments are manually defined, some regions may lack them and it is impossible for users to conduct a search in such areas.

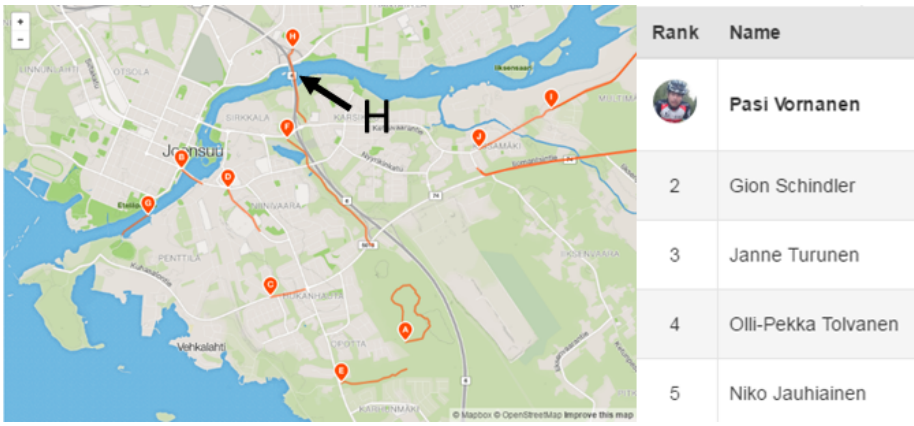


Figure 23. Strava segments in Joensuu (left panel) and the first 5 athletes on segment H (right panel).

4.3 SIMILARITY SEARCH

Route similarity can be used as a method to search the database (Figure 24). Starting with a reference route, Mopsi allows users to perform *route similarity ranking* (RSR). The application shows a list of routes that are spatially similar to the reference route, with results ranked in decreasing order of similarity. For each route, the ranking shows the user who recorded that route, the transportation mode used, the similarity value and the date. Figure 24 shows only the first 26 elements of the ranking whereas the full list contained 1196 routes.

The user can compare the reference route with any similar route in the list. The analysis can also be localized to a chosen segment of the reference route.

1	**Radu**	🚲	100%
2	**Radu**	🚲	99%
3	**Radu**	🚲	96%
4	**Radu**	🚲	96%
5	**Radu**	🚲	96%
6	**Radu**	🚲	96%
7	**Radu**	🚲	96%
8	**Radu**	🚲	96%
9	**Radu**	🚲	96%
10	**Radu**	🚲	95%
11	**Radu**	🚲	95%
12	**Radu**	🚲	95%
13	**matti**	🚲	95%
14	**Radu**	🚲	94%
15	**Radu**	🚲	92%
16	**Radu**	🚲	92%
17	**Radu**	🚲	90%
18	**Radu**	🚗	90%
19	**Radu**	🚲	89%
20	**Radu**	🚲	84%
21	**Radu**	🚲	84%
22	**Radu**	🚲	81%
23	**Radu**	🚲	77%
24	**Andrei**	🚲	77%
25	**Radu**	🚲	77%
26	**Radu**	🚲	76%

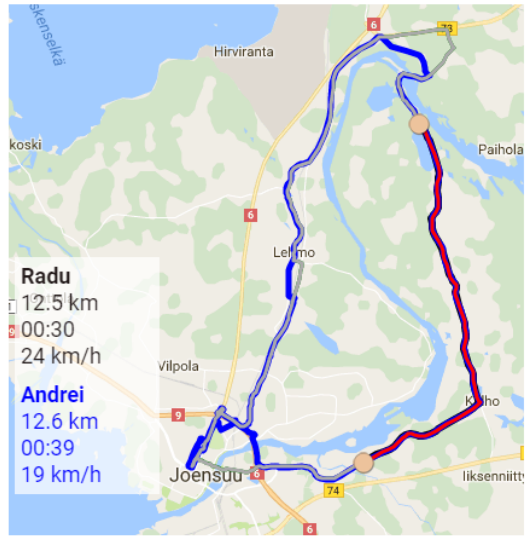


Figure 24. User Radu's reference route (grey) and a list of highly similar routes from the database. The user name, inferred movement type and similarity values are shown for each similar route. A route of user Andrei (blue) was 77% similar. A manually selected common segment (red) was selected and analysed, and Radu's segment was shown to be 5 km/h faster. The reason for the large speed difference was a strong tailwind from the north.

4.4 GESTURE SEARCH

Gestures have been used as a means to access menu items without the need to traverse large hierarchies [Kristensson and Zhai 2007, Li 2010] by providing users with various types of shortcuts. We proposed using gestures to access routes in large collections. The gesture represents hand-drawn input in the form of a free shape drawn on a map; the shape approximates the locations through which the targeted route passes. According to Cirelli and Nakamura [2014] and Karam and Schraefel [2015], this gesture is classified as symbolic and triggers a command, namely the search for spatially similar routes.

Typically, gesture-based systems require the user to learn a set of symbols [Cirelli and Nakamura 2014]. In our study, the user was expected to remember the spatial characteristics of the route and to be able to read maps, because roads, buildings and terrain elements (such as forests, lakes, and rivers) provide key information when giving the input. For example, a user can draw the input by following a river front, road, or other landmark visible on the map. Users who have a large route collection benefit most from the gesture search. It is therefore fair to assume that these users also have the necessary skills to understand maps.

Gesture search has two modes: similarity (Figure 25) and inclusion (Figure 26), which use the two operations respectively. To enter the gesture input mode, the user presses a hotkey (Ctrl for similarity, Shift for inclusion). While the key is pressed, the map changes colour to show which mode is active and further acts as a canvas on which to draw. The drawing is completed when the hotkey is released and search is then initiated with the drawn shape being used as the input.

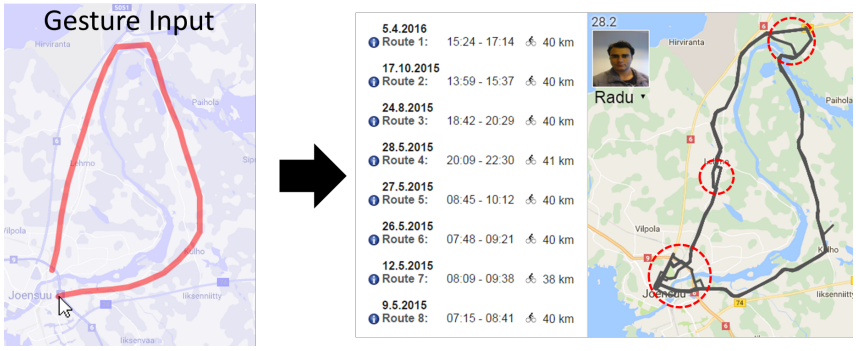


Figure 25. Gesture search using Similarity. Eight routes resembling the drawn shape were found and returned to the user. The eight routes overlapped perfectly on the map, except in three highlighted regions where the road network allowed variation.

The similarity search retrieves route candidates that are similar to the drawn shape, whereas the inclusion search retrieves candidates that contain or include the drawn shape. The latter is similar to the segments feature of Strava in the sense that routes passing through the drawn segment are retrieved. The benefit is that segments do not need to be created and stored in the system. Users can draw any segment at any time. The downside is that users do not become aware of places in which other people compete, as they do in Strava.

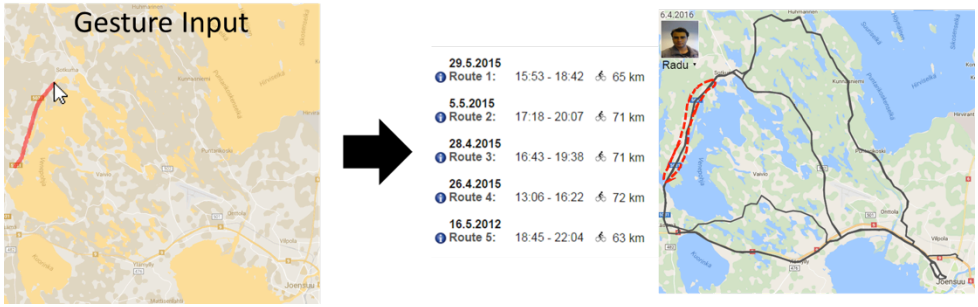


Figure 26. Gesture search using Inclusion. Five routes that pass through the drawn region are found and presented to the user.

The precision of drawing the gesture should be independent of the zoom level of the map. When the zoom level is decreased by one unit the content of the map be-

comes half of its previous size, and consequently the regions on the map become twice as difficult to read. We created 10 grids with different resolutions and stored the routes at each of these approximation levels (Table 9).

Table 9. A mapping from zoom level to the grid resolution. The statistics are for Mopsi2014 Route dataset using each of the grid resolutions.

Zoom level	≤ 6	7	8	9	10	11	12	13	14	≥15
Grid resolution	0	1	2	3	4	5	6	7	8	9
Cell size (km)	12.8	6.4	3.2	1.6	0.8	0.4	0.2	0.1	50 m	25 m
Number of cells	7×10^4	9×10^4	1×10^5	2×10^5	4×10^5	7×10^5	1×10^6	3×10^6	5×10^6	1×10^7
Memory (MB)	3.5	4.5	6.5	9.5	16.5	30.6	59.6	118.6	238	486
B-tree Index (MB)	8.5	9.5	13.5	21.5	35.6	66.7	131.8	263.1	526	1.1 GB

The finest grid has a cell size of 25 m × 25 m. Finer grids are not needed because at this level, GPS error becomes apparent and the route approximations become unreliable. The number of cells needed increases exponentially when finer grids are produced. Therefore, we did not compute unnecessary levels for no purpose. The sparsest grid had a cell length of 12.5 km. At lower levels (≥ 25 km) the cell size becomes so large that even the longest routes are represented by only a few cells.

4.5 EVALUATION

We studied the efficiency of the gesture search from a usability point of view. We compared the average time a user spends on searching a randomly chosen route using the gesture search versus using the traditional system. Eleven volunteers were asked to search randomly selected routes using a tool[10] built for this purpose, as follows:

1. A target route was shown on the map but no date, length or duration were shown. The user could study and memorize the route for as long as he or she wanted to.
2. When the user pressed the *Start* button, he or she was (randomly) directed either to the traditional system or to the new gesture search. The timer was started.
3. The task was to find the route and input its date and then press the *Stop* button. If the date was correct the timer was stopped. If the user considered the task too difficult, he or she could press the *Give-up* button.

[10] http://cs.uef.fi/mopsi/routes/gestureSearch/qual.php

Each volunteer was asked to repeat the test at least 10 times or for as long as he or she found the task enjoyable.

In total, 82 routes were found using the traditional system, and 89 routes using the gesture search. Traditional searches were given up on 50% more often than the gesture search, with 24 traditional searches being abandoned, compared with only 16 gesture searches. Gesture search was 41% faster, on average. The individual performance differences are shown in Figure 27. Traditional searches were slower on average than gesture searches for all users except one.

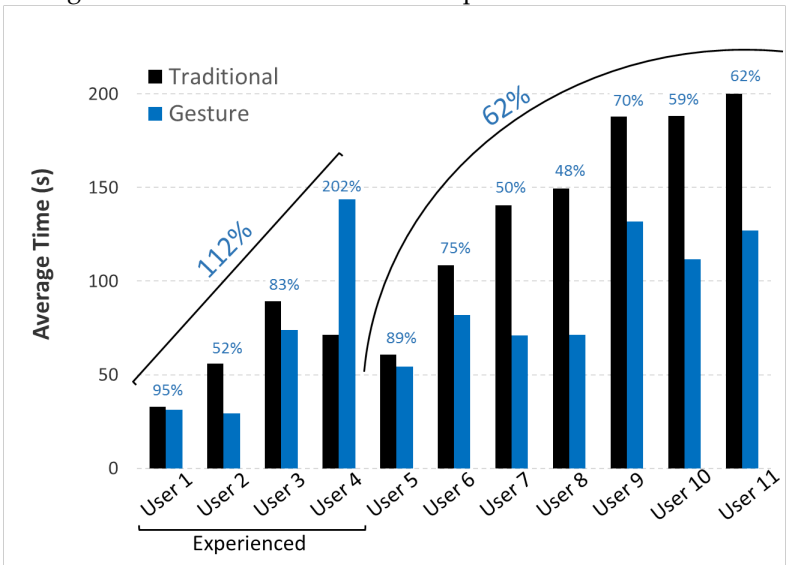


Figure 27. Average search times, showing the superiority of the gesture search relative to the traditional search. Results are shown for every user who participated in the experiment.

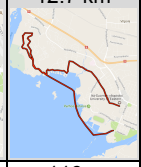
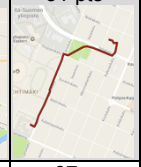
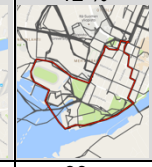
The search time was also affected by factors such as complexity and length of the route, and density of the areas the route passes through. We next grouped the results by these three factors. Complexity was calculated as the number of points used by the polygonal approximation [Chen et al. 2012] to represent the route at the maximum zoom level at which the route could still be seen in its entirety. Density was calculated as the proportion of cells that were frequented by many other routes; density values are the converse of the noteworthiness value in **[III]**. The results, shown in Table 10, indicated that although shorter, less complex routes in low-density areas were faster to find, the gesture search outperformed the traditional approach in all cases.

The volunteers were asked if they liked the gesture search and which method they would prefer to use for similar search tasks. Ten volunteers rated the gesture search as good and one as excellent. Most (nine volunteers) preferred the gesture search,

none preferred the traditional search, and two people said they would not use either method. Written comments included “I really liked it” and “It was fun”.

The four volunteers whose data are shown on the left-hand bars in Figure 27 had previously been familiar with the traditional search method. Even in that group, the gesture search yielded faster results for 75% of cases. This result was above our expectations because we assumed that previous experience in using the traditional method would bias the results. Less experienced users seem to find the routes faster using the gesture search than the traditional search. This result indicates that the gesture search is a more intuitive method.

Table 10. Average search times, grouped by various factors.

	Length		**Complexity**		**Density**	
	Short 2.7 km	Long 12.7 km	Low 31 pts	High 128 pts	Low 12 %	High 75 %
						
Traditional	90 s	116 s	87 s	120 s	90 s	117 s
Gesture	64 s	78 s	65 s	77 s	54 s	88 s
Reduction	30%	33%	25%	36%	30%	24%

5 INFERRING ROAD NETWORKS

In large cities, navigation using traditional means – a paper map – has become almost impossible. Road networks are becoming increasingly complex and large roads rarely offer the chance to pause and study the situation if one gets lost. As a result, navigators such as TomTom[11] and Garmin[12] are present in most cars nowadays and most smartphones have navigation capabilities. Road networks may also be used to offer personalized navigation such as safe routing [Krumm and Horvitz 2017] or accessible routing [Kasemsuppakorn and Karimi 2009]. For such applications, up-to-date and accurate information is essential.

The current acquisition and updating of road networks is characterized by a large amount of manual work, which is costly and slow. There have been two main ways of automatizing the process: aerial image processing [Tavakoli and Rosenfeld 1982, Hu et al. 2007, Barsi and Heipke 2003] and GPS route processing [Edelkamp and Schrödl 2003, Davies et al. 2006, Cao and Krumm 2009].

Using aerial images has limitations because roads have varying features such as colour, intensity, shadows and variable widths (Figure 28). Buildings cause further difficulties and this issue was addressed by Tavakoli and Rosenfeld [1982]. In that study, categorization was performed using edge features to separate roads from buildings and other structures. The method described by Hu et al. [2007] for finding roads begins with several initial guesses. A road tree is then 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. Obtaining the direction of travel for the roads is not possible using image data.

Figure 28. Aerial images showing part of a city (left) and a countryside area (right).

[11] https://www.tomtom.com
[12] http://www.garmin.com

GPS routes are easier and less costly to collect than aerial images. Route databases collected for various purposes, such as the OSM traces[13] and Mopsi2014[14], are already available and growing (Figure 29). The routes have fewer artefacts than the aerial images and the only issue is the error caused by tall buildings and other obstructions. Routes have the added benefit of preserving the direction of travel and can be used to produce a directed graph. Because of these advantages, inferring a road network from GPS routes has become an attractive area of research, and several conceptually distinct approaches have emerged. In addition to road networks, other types of networks, such as pedestrian networks [Kasemsuppakorn and Karimi 2013] which can be inferred from walking routes.

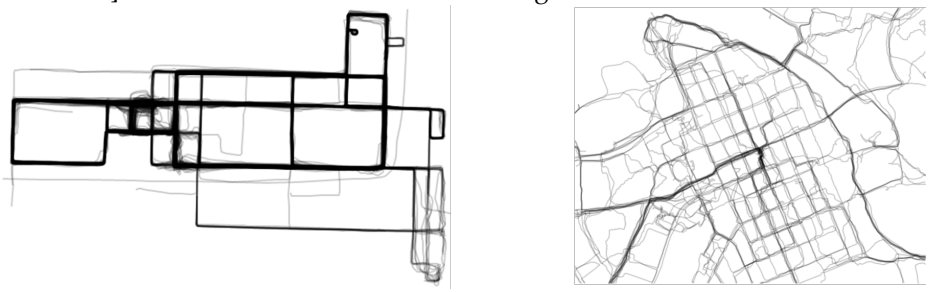


Figure 29. GPS routes recorded in Chicago (left) and Joensuu (right).

Visual methods [Chen and Cheng 2008, Davies et al. 2006] use route data to form binary images, which are processed using image-processing techniques such as contour finding, morphological operations, skeletonization and density-based thresholding.

Route merging 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 weights are removed from the network. Merging methods typically filter GPS data in order to better handle the noise.

Clustering-based methods also exist to infer road networks [Edelkamp and Schrödl 2003, Schrödl et al. 2004]. This approach typically begins by considering only the points of the GPS routes; connectivity is omitted. Then, equally spaced representatives are placed over the point data. The representatives are optimized using *k*-means, and finally the network is formed using the point connections from individual routes.

Some studies have focused on the task of locating the road intersections [Barsi and Heipke 2003, Fathi and Krumm 2010], and machine learning is used to achieve this

[13] https://www.openstreetmap.org/traces
[14] http://cs.uef.fi/mopsi/routes/dataset

goal. A classifier is trained using positive and negative samples obtained from data containing ground truth, typically OSM.

The visual, merging and clustering methods perform poorly in places where GPS accuracy is low. In those regions, numerous intersections are incorrectly found and many spurious segments disrupt the quality of the network. The filtering process employed by the merging methods is insufficient to handle abundant GPS error. Visual methods can handle the problem through setting a higher value for the density threshold parameter. The downside is that regions of the map having a low number of routes will also be omitted from the process. The existing clustering methods do not attempt to solve GPS errors at all.

We argue that accurately obtaining the location of road intersections is crucial for generating high-quality maps. Therefore, in **[V]** we proposed a new method entitled CellNet, which works in two steps:

1. finding the road intersections
2. generating the in-between road segments.

CellNet has two parameters: L and R, which can be interpreted respectively as the expected average GPS error in the dataset (L) and the minimum distance between two intersections (R). The method does not lead to substantial differences when these parameters are altered, and we expected it to work well with our recommended values of 25 and 80. A visual representation of the method output using these values is shown in Figure 30.

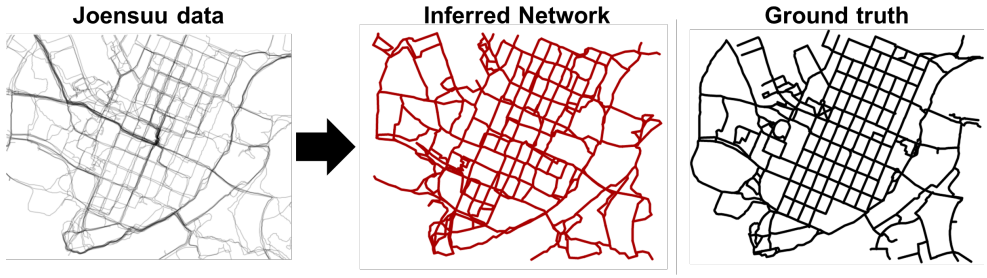


Figure 30. Joensuu road network as inferred by CellNet.

Unlike other intersection finding methods [Barsi and Heipke 2003, Fathi and Krumm 2010], CellNet does not require training data. It finds the intersections using a *split descriptor*, which checks to see whether at a certain location, routes head into more than two general directions. To check this a set of data points is created, as described in Figure 31. Then, clustering is performed separately with two and three clusters, using the random swap algorithm [Fränti and Kivijärvi 2000]. The two clusterings are inspected using the silhouette coefficient [Rousseeuw and

Kaufman 1990] to deduce the correct number of clusters. If three clusters provide the best solution, a split is concluded.

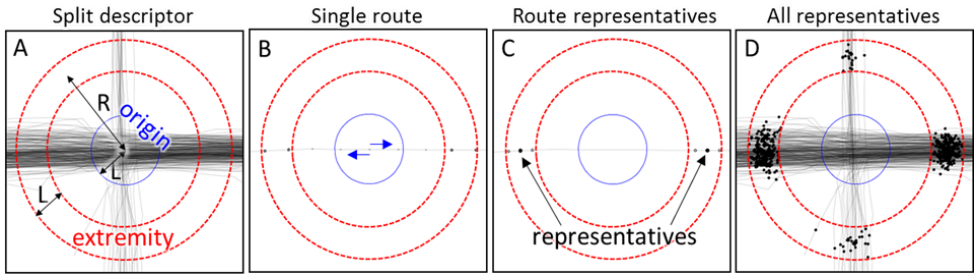


Figure 31. A, the split descriptor composed of the origin and the extremity. B, a sample route traversing through the point of interest; points inside the extremity are chosen. C, the selected points are averaged in each of the two directions to create the representatives. D, representatives of all routes passing through the point of interest.

Once the intersections are found, the in-between road segments are selected by checking every subsequent pair of intersections that every route passes through. If more routes link the same intersections, all segments are kept and are used in the following optimization step. We used the method in Hautamäki et al. [2008] to obtain a representative for all segments between every pair of intersections (Figure 32). We excluded segments that were not 100% spatially similar according to C-SIM similarity measure **[III]**. Unlike Hautamäki et al. [2008], we did not initialize the optimization method using the medoid. Instead, we used the shortest segment under the assumption that it has less GPS error, which would make it a good initial guess. In addition, we used the FastDTW algorithm [Salvador and Chan 2004]. Our results were no worse than those obtained when the medoid was used and the optimum DTW was calculated but the speed had improved by 99%.

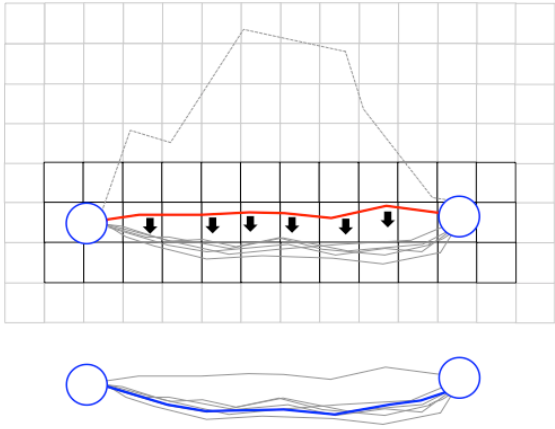


Figure 32. The minimum length segment (red) was improved by the averaging method using other segments that were spatially similar. The dashed-line segment (top) was not spatially similar and was therefore excluded from the process. The result was the fine-tuned blue segment (bottom).

Once the links were optimized, we noted that some became redundant. This was the result of a route missing one or more intersections due to GPS error. We removed these links using the following strategy. For every link segment, we found all segments that were contained inside its spatial region and marked them as valid. To do this, we used the inclusion measure from **[III]**. If a valid path existed between the two intersections, the direct link was removed because it was probably redundant. This strategy is an improvement over the one presented in Fathi and Krumm [2010], which takes into consideration only the physical length of the segments. Using only the length implies that the direct segment is removed in both situations presented in Figure 33.

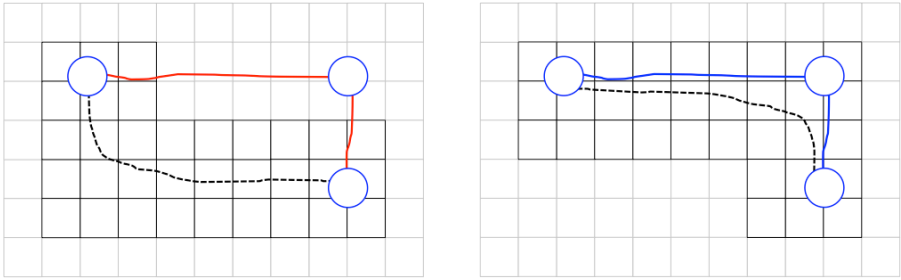


Figure 33. The specified segment (dashed line) is shown together with the dilated cell representation (dark cells). In the example on the right, two other road segments are included in the region defined by the dilated cells and are valid. The example on the left has no valid segments.

A question that has not yet been answered is how to score the quality of a generated road network. Virtually all studies to date have relied on visual inspection of the results, with generated maps being compared with existing maps or satellite imagery. We propose two novel ways of comparing a generated map with ground truth obtained from OSM. First, we evaluate the intersections using the same technique that is used to compute clustering quality in Fränti et al. [2014].

We next propose a way to evaluate the road segments connecting the intersections. We use grid cells to measure whether the ground truth segments are properly identified. The measure is also sensitive to redundant segments (Figure 34).

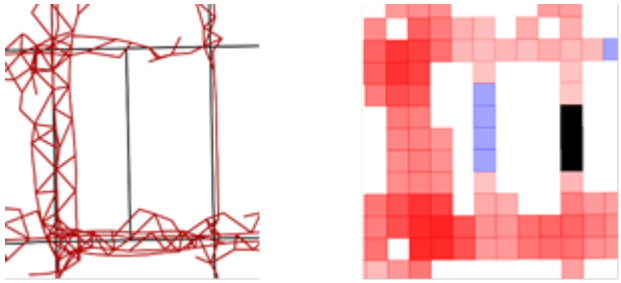


Figure 34. Left panel: ground truth segments (black) and inferred segments (red). Right panel: red cells show over-emphasis proportional to the colour intensity; blue cells show segments that were not represented at all; black cells mean correct representation.

Table 11 shows summary data on the quality of the road network produced by CellNet and a comparison with the three other methods. The visual method yielded the highest precision for the Chicago dataset, because the high density of routes in that dataset produced good visual features. The recall was low because the portion of the dataset that fell below the density threshold was omitted. The clustering and merging methods displayed high recall, because – unlike the visual approach – they did not drop out part of the dataset according to a threshold. However, the precision of the clustering and merging methods was low because they detected too many intersections in regions with many routes and low GPS accuracy. CellNet achieved the most balanced results in terms of precision and recall, and produced the best result in terms of the F-score.

Table 11. Comparison of CellNet with other popular road network inference methods.

Chicago

	Intersections			Links		
Method	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
Visual	97%	27%	42%	97%	27%	42%
Clustering	14%	94%	24%	17%	94%	28%
Merging	5%	90%	10%	7%	70%	10%
CellNet	77%	90%	84%	81%	68%	75%

Joensuu

	Intersections			Links		
Method	**Precision**	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
Visual	54%	63%	58%	56%	38%	46%
Clustering	22%	85%	36%	16%	92%	27%
Merging	22%	52%	31%	13%	28%	18%
CellNet	71%	68%	69%	68%	49%	58%

A potential challenge is the memory requirements of the generated network. The compared methods produce unnecessarily complex segments, which could be simplified by using polygonal approximation [Chen et al. 2009, Chen et al. 2012, Pikaz

and Dinstein 1995]. We used the technique in Chen et al. [2012] and obtained networks of only 25% of the size of those produced by other methods.

6 SUMMARY OF CONTRIBUTIONS

This chapter summarizes the contributions of our five publications. In publication **[I]** we studied how routes are recorded, stored and visualized. In publications **[II]** and **[III]** we explained how a grid-based representation is useful when implementing four commonly used functions: similarity, inclusion, novelty and noteworthiness. Publication **[IV]** presented an application of the fast grid-based similarity search, namely gesture search, which allows users to search for routes by drawing a free-form shape on the map. Publication **[V]** presents a novel way of generating a road network from a route dataset.

In **[I]**, we proposed a method for recording GPS routes which allows online and offline capability and live tracking, and is efficient in terms of internet and battery usage. Using the polygonal approximation and cropping strategies allows the Mopsi system to query and display routes consisting of over 3.5 million points in under 2 seconds. As far as we know, no other online system even exists to achieve this and all systems show only the start points or just a single route at a time.

In **[II]**, we presented a new fast and intuitive way of computing route similarity using a grid-based approximation of the routes and set-based operations. The method is equipped with interpolation and dilation of the grid cells in order to cope with missing points and to handle the arbitrary grid division into cells.

In **[III]** we introduced four grid-based route operations: similarity, inclusion, novelty and noteworthiness. The methods were analysed in terms of their space requirements, computational complexity and indexing strategy. In that work, the similarity measure presented in **[II]** was redefined as "inclusion" and a new, improved similarity measure was introduced. Using the new similarity measure, a route similarity search strategy was presented and was shown to work in real time on a real-world dataset. We built an interactive tool for comparing and understanding different similarity measures and offered an application programming interface (API) for calling our newly presented measure. The API also supports calls to the other similarity measures. These are available in the web page[15] of **[III]**.

In **[IV]** we showed that the similarity search method can be used to search for a route if the user remembers the approximate shape but not the time. This feature improves the user's experience when searching routes in large data collections, compared with the traditional interface described in **[I]**.

[15] http://cs.uef.fi/mopsi/routes/grid

In **[V]** we present a new method for road network extraction, CellNet, which produces accurate results without the need to optimize parameters. We show that CellNet produces higher quality results than three conceptually different state-of-the-art methods.

7 CONCLUSIONS

We presented efficient ways to record, store and visualize route collections and demonstrated their efficiency within the real-world environment of Mopsi. We showed that polygonal approximation and cropping are very useful in reducing the amount of data, and these techniques also allow the display of large route collections on the map. In addition, we showed that the system is capable of displaying routes consisting of over 3.5 million points in less than 2 seconds.

Many popular route similarity measures exist, inspired by methods based on various fields – such as string matching, time-series analysis, curve comparison and set matching. Most of these methods are slow and are not intuitive for average users, who perceive routes as being shapes on a map. Our proposed similarity measure, C-SIM, uses the grid-based representation of routes to output a fast and intuitive measure of similarity. It was combined with an indexing strategy, which was demonstrated to perform similarity searches in real time on a database containing over 5,000 routes.

Searching for routes is not easy in large collections. We proposed gestures to be used for this purpose. We built a working system that allows users to draw the approximate shape of a route on the map; then, spatially similar routes are retrieved. This method is preferred over the traditional approach when the user cannot remember the date of the searched route.

To date, managing road networks still requires intensive manual editing. Our proposed method, CellNet, provides more accurate results on different datasets compared with other popular approaches, without the need for parameter optimization. In addition, the size of the generated network is reduced by using polygonal approximation to produce maps that require a quarter of the storage space needed by other automatically generated maps.

Even though we have given solutions for many different applications, some problems remain open. Potential future research includes:

- To use the road network to predict user movements, recommend routes or give navigational instructions;

- To further reduce the amount of data when visualizing a route collection by keeping a single representative segment where multiple routes overlap.

BIBLIOGRAPHY

Agrawal R., Faloutsos C. & Swami A. 1993. Efficient similarity search in sequence databases. *International Conference on Foundations of Data Organization and Algorithms (FODO 1993)*, Chicago, Illinois, USA, pp. 69-84

Alahakone A. U. & Ragavan V. 2009. Geospatial Information System for tracking and navigation of mobile objects. *IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2009)*, Singapore, pp. 875-880.

Almer A. & Stelzl H. 2002. Multimedia visualisation of geo-information for tourism regions based on remote sensing data. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences, 34(4), pp. 436-441.

Ananthanarayanan G., Haridasan M., Mohomed I., Terry D. & Thekkath C. A. 2009. Startrack: a framework for enabling track-based applications. *ACM international conference on Mobile systems, applications, and services (MobiSys 2009)*, Kraków, Poland, pp. 207-220.

Bao T., Cao H., Yang Q., Chen E. & Tian J. 2012. Mining significant places from cell id trajectories: A geo-grid based approach. *IEEE International Conference on Mobile Data Management (MDM 2012)*, Bengaluru, India, pp. 288-293

Barsi, A. & Heipke, C. 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.

Berndt D. J. & Clifford J. 1994. Using dynamic time warping to find patterns in time series. *KDD workshop*, Vol. 10, No. 16, pp. 359-370.

Biagioni, J. & Eriksson, J. 2012. Inferring road maps from global positioning system traces. Transportation Research Record: Journal of the Transportation Research Board, 2291(1), pp. 61-71.

Cao L. & Krumm J. 2009. From GPS traces to a routable road map. *ACM SIGSPATIAL international conference on advances in geographic information systems (ACM SIGSPATIAL GIS 2009)*, Seattle, Washington, USA, pp. 3-12

Chen C. & Cheng Y. 2008. Roads digital map generation with multi-track GPS data. *IEEE In International Workshop on Education Technology and Training, 2008 and 2008 International Workshop on Geoscience and Remote Sensing. (ETT and GRS 2008)*, Vol. 1, pp. 508-511.

Chen J., Wang W., Liu L. & Song J. 2011. Clustering of trajectories based on Hausdorff distance. *IEEE International Conference on Electronics, Communications and Control (ICECC 2011)*, Ningbo, China, pp. 1940-1944.

Chen L., Özsu M. T. & Oria V. 2005. Robust and fast similarity search for moving object trajectories. *ACM SIGMOD international conference on Management of data and Symposium on Principles Database and Systems (SIGMOD/PODS 2005)*, Baltimore, MD, USA, pp. 491-502

Chen L. & Ng R. 2004. On the marriage of lp-norms and edit distance. *International Conference on Very Large Data Bases-Volume (VLDB 2004)*, Toronto, Canada, pp. 792–803

Chen M., Xu M. & Fränti P. 2012. A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Transactions on Image Processing*, 21(5), pp. 2770-2785.

Chen M., Xu M., & Fränti P. 2012. Compression of GPS trajectories. *IEEE Data Compression Conference (DCC 2012)*, pp. 62-71.

Chen Y., Jiang K., Zheng Y., Li C. & Yu N. 2009. Trajectory simplification method for location-based social networking services. *ACM International Workshop on Location Based Social Networks*, Seattle, USA, pp. 33-40.

Cirelli M. & Nakamura R. 2014. A Survey on Multi-touch Gesture Recognition and Multi-touch Frameworks. *ACM Conference on Interactive Tabletops and Surfaces (ITS 2014)*, Dresden, Germany, pp. 35-44.

Cormen T. H. 2009. Introduction to algorithms, *MIT press*.

Davies, J. J., A. R. Beresford & A. Hopper. 2006. Scalable, Distributed, Real-Time Map Generation. IEEE Pervasive Computing, Vol. 5, No. 4, pp. 47–54.

Edelkamp, S. & Schrödl S.. 2003. Route Planning and Map Inference with Global Positioning Traces. In Computer Science in Perspective, Springer Berlin Heidelberg, Vol. 2598, pp. 128–151.

Eiter T. & Mannila H. 1994. Computing discrete Fréchet distance. *Tech. Report CD-TR 94/64, Information Systems Department*, Technical University of Vienna

Evans M. R., Oliver D., Shekhar S. & Harvey F. 2013. Fast and exact network trajectory similarity computation: a case-study on bicycle corridor planning. *ACM SIGKDD International Workshop on Urban Computing (UrbComp 2013)*, Chicago, IL, USA, 9

Fathi A. & Krumm J. 2010. Detecting road intersections from gps traces. *International Conference on Geographic Information Science (GIScience 2010)*, Zurich, Switzerland, pp. 56-69

Follin J. M., Bouju A., Bertrand F. & Boursier P. 2003. Management of multi-resolution data in a mobile spatial information visualization system. IEEE International Conference on Web Information Systems Engineering Workshops, 2003. pp. 92-99.

Frentzos E., Gratsias K. & Theodoridis Y. 2007 .Index-based most similar trajectory search. *IEEE International Conference on Data Engineering (ICDE 2007)*, Istanbul, Turkey, pp. 816-825

Fränti P. & Kivijärvi J. 2000. Randomized local search algorithm for the clustering problem. *Pattern Analysis and Applications*, 3 (4), pp. 358-369.

Fränti P., Chen J. & Tabarcea A. 2011. Four Aspects of Relevance in Sharing Location-based Media: Content, Time, Location and Network. *In WEBIST*, pp. 413-417.

Fränti P., Rezaei M. & Zhao Q. 2014. Centroid index: Cluster level similarity measure, *Pattern Recognition*, 47 (9), pp. 3034-3045.

Gali N. & Fränti P. 2016. Content-based title extraction from web page. International Conference on Web Information Systems and Technologies (WEBIST 2016), Rome, Italy, pp. 204-210.

Gali N., Tabarcea A. & Fränti P. 2015. Extracting representative image from web page. International Conference on Web Information Systems & Technologies (WEBIST 2015), pp. 411-419.

Gradshteyn I. S. & Ryzhik I. M. 2000. Tables of Integrals, Series, and Products, 6th ed. San Diego, *CA: Academic Press*, pp. 1114-1125.

Guttman A. 1984. R-trees: a dynamic index structure for spatial searching. 1984. *ACM SIGMOD international conference on Management of data (SIGMOD 1984)*, New York, NY, USA, 47-57

Hamilton J.D. 1994. Time series analysis (Vol. 2). *Princeton: Princeton university press*

Haridasan M., Mohomed I., Terry D., Thekkath C. A. & Zhang, L. (2010, October). StarTrack Next Generation: A Scalable Infrastructure for Track-Based Applications.

ACM/USENIX Symposium on Operating Systems Design and Implementation (OSDI 2010), Vancouver, Canada, pp. 409-422.

Hautamäki V., Nykänen P. & Fränti P. 2008. Time-series clustering by approximate prototypes, *IAPR International Conference on Pattern Recognition (ICPR'08)*, Tampa, Florida, USA, December 2008, pp ???.

Horozov T., Narasimhan N. & Vasudevan V. 2006. Using location for personalized POI recommendations in mobile environments. IEEE International symposium on Applications and the internet, (pp. 6-pp). ???.

Hu, J., Razdan, A., Femiani, J. C., Cui, M. & Wonka, P. 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.

Karam M. & Schraefel M. C. 2015. A taxonomy of Gestures in Human Computer Interaction. *ACM Transactions on Computer-Human Interactions*, 2015. (in press)

Karimi H.A. & Lockhart J.T. 1993. GPS-based tracking systems for taxi cab fleet operations. *IEEE Conference on Vehicle Navigation and Information Systems*, pp. 679-682.

Kasemsuppakorn P. & Karimi H.A. 2009. Personalised routing for wheelchair navigation. *Journal of Location Based Services*, 3(1), pp.24-54.

Kasemsuppakorn P. & Karimi H.A. 2013. A pedestrian network construction algorithm based on multiple GPS traces. *Transportation research part C: emerging technologies*, 26, pp. 285-300.

Kennedy M. & Kopp S. 2001. Understanding Map Projections. *ESRI Press*.

Kristensson P. O. & Zhai S. 2007. Command strokes with and without preview: using pen gestures on keyboard for command selection. *SIGCHI Conference on Human Factors in Computing Systems (CHI 2007)*, New York, USA, pp. 1137-1146.

Krumm J. & Horvitz E. 2006. Predestination: Inferring destinations from partial trajectories. *In Proceedings of the 8th International Conference on Ubiquitous Computing (UbiComp '06)*, Orange County, CA, USA, pp. 243-260.

Krumm J. & Horvitz E. 2017. Risk-Aware Planning: Methods and Case Study for Safer Driving Routes. *In Twenty-Ninth Innovative Applications of Artificial Intelligence Conference*, pp. 4708-4714.

Lehtimäki T. M., Partala T., Luimula M. & Verronen P. 2008. LocaweRoute: an advanced route history visualization for mobile devices. ACM working conference on advanced visual interfaces, pp. 392-395.

Li Y. 2010. Gesture search: a tool for fast mobile data access. *ACM Symposium on User interface software and technology (UIST 2010)*, New York, USA, pp. 87-96.

McCullough A., James P. & Barr S. 2011. A Service Oriented Geoprocessing System for Real-Time Road Traffic Monitoring. *Transactions in GIS*, 15(5), 651-665.

Morris S., Morris A. & Barnard K. 2004. Digital trail libraries. *ACM/IEEE Conference on Digital Libraries (ICDL 2004)*, New Delhi, India, pp. 63-71.

Ni J. & Ravishankar C. V. 2007. Indexing spatio-temporal trajectories with efficient polynomial approximations. *IEEE Transactions on Knowledge and Data Engineering*, 19(5).

Niehöfer B., Lewandowski A., Burda R., Wietfeld C., Bauer F. & Lüert O. 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, 2009.

Pang L. X., Chawla S., Liu W. & Zheng Y. 2013. On detection of emerging anomalous traffic patterns using GPS data. *Data & Knowledge Engineering (DKE)*, 87, pp. 357-373.

Pelekis N., Kopanaki, I., Kotsifakos E. E., Frentzos E. & Theodoridis Y. 2011. Clustering uncertain trajectories. *Knowledge and information systems*, 28(1), pp. 117-147.

Pikaz A. & Dinstein I. 1995. An algorithm for polygonal approximation based on iterative point elimination, *Pattern Recognition Letters*, 16 (6), 557–563, Jun. 1995.

Rezaei M., Gali N. & Fränti P. 2015. ClRank:a method for keyword extraction from web pages using clustering and distribution of nouns", IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT 2015), pp. 79-84.

Rockafellar R. T. & Wets R. J. B. 2009. *Variational analysis (Vol. 317). Springer Science & Business Media*

Rousseeuw P. J. & Kaufman L. 1990. *Finding Groups in Data.* Wiley Online Library.

Rezaei M. & Fränti P. 2017. Clustering large geo-referenced data on maps for clutter removal (*submitted*)

Salvador S. & Chan P. 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 (SIGKDD 2004)*, Seatle, Washington, USA, pp. 70–80.

Shang S., Ding R., Yuan B., Xie K., Zheng K. & Kalnis P. 2012. User oriented trajectory search for trip recommendation. *ACM International Conference on Extending Database Technology*, Berlin, Germany, pp. 156-167.

Tabarcea A., Wan Z., Waga K. & Fränti P. 2013. O-mopsi: Mobile orienteering game using geotagged photos. CONFERENCE, pp. 300–303.

Tavakoli, M. & Rosenfeld, A. 1982. Building and road extraction from aerial photographs. IEEE Transactions on Systems, Man, and Cybernetics, 12, pp. 84-91.

Vlachos M., Gunopulos D. & Kollios G. 2002. Robust similarity measures for mobile object trajectories. *International Workshop on Database and Expert Systems Applications (DEXA 2002)*, Aix en Provence, France, pp. 721-726

Vlachos M, Kollios G. & Gunopulos D. 2002. Discovering similar multidimensional trajectories. *IEEE International Conference on Data Engineering (ICDE 2002)*, pp. 673-684

Waga K., Tabarcea A., Chen M. & Fränti P. 2012. Detecting movement type by route segmentation and classification. *IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012)*, Pittsburgh, USA, pp. 508-513.

Waga K., Tabarcea A. & Fränti P. 2011. Context aware recommendation of location-based data. *IEEE International conference on System Theory, Control, and Computing (ICSTCC 2011)*, pp. 1-6.

Waga K., Tabarcea A. & Fränti P. 2012. Recommendation of points of interest from user generated data collection. *IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012)*, Pittsburgh, USA, pp. 550-555.

Wang H. & Liu K. 2012. User oriented trajectory similarity search. *ACM SIGKDD International Workshop on Urban Computing (UrbComp 2012)*, Beijing, China, pp. 103-110

Wang H., Su H., Zheng K., Sadiq S. & Zhou X. 2013. An effectiveness study on trajectory similarity measures. *Australasian Database Conference (ADC 2013)*, Adelaide, Australia, pp. 13-22.

Wei L., Zheng Y. & Peng W. 2012. Constructing popular routes from uncertain trajectories. *In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '12)*, Beijing, China, pp. 195-203.

Yanagisawa Y., Akahani J. & Satoh T. 2003. Shape-based similarity query for trajectory of mobile objects. *International Conference on Mobile Data Management (MDM 2003)*, Melbourne, Australia, pp. 63-77

Ying J. J. C., Lu E. H. C., Lee W. C., Weng T. C. & Tseng V. S. 2010. Mining user similarity from semantic trajectories. *ACM SIGSPATIAL International Workshop on Location Based Social Networks (ACM SIGSPATIAL GIS 2010)*, San Jose, CA, USA, pp. 19-26

Ying X., Xu Z. & Yin W. G. 2009. Cluster-based congestion outlier detection method on trajectory data. *IEEE International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2009)*, Tianjin, China, pp. 243-247

Zhang D., Li N., Zhou Z. H., Chen C., Sun L. & Li S. 2011. iBAT: detecting anomalous taxi trajectories from GPS traces. *ACM international conference on Ubiquitous computing (UbiComp 2011)*, Beijing, China, pp. 99-108

Zheng V. W., Zheng Y., Xie X & Yang Q. 2010. Collaborative location and activity recommendations with gps history data. *In Proceedings of the 19th ACM International Conference on World Wide Web (WWW '10)*, New York, NY, USA, pp. 1029-1038.

Zheng Y. & Zhou X. 2011. Computing with spatial trajectories, *Springer Science & Business Media*

Zheng Y., Wang L., Zhang R., Xie X. & Ma W. Y. 2008. GeoLife: Managing and understanding your past life over maps. *IEEE International Conference on Mobile Data Management (MDM 2008)*, Beijing, China, pp. 211-212.

Paper I

Waga K., Tabarcea A., Mariescu-Istodor R. & Fränti P.
"Real Time Access to Multiple GPS Tracks"
International Conference on
Web Information Systems & Technologies
Aachen, Germany, pp. 293-299, 2013

Real Time Access to Multiple GPS Tracks

Karol Waga, Andrei Tabarcea, Radu Mariescu-Istodor and Pasi Fränti
Speech and Image Processing Unit, School of Computing, University of Eastern Finland, Joensuu, Finland
{kwaga, tabarcea, radum, franti}@cs.uef.fi

Keywords: Geographic Information Systems, GPS Tracks, Trajectory Storage and Visualization.

Abstract: Increasing availability of mobile devices with GPS receiver gives users the possibility to record and share a variety of location-based data, including GPS tracks. We describe a complete real-time system for acquisition, storage, querying, retrieval and visualization of GPS tracks. The main problems faced are how to store the data, how to access and how to visualize large amount of data. We propose to reduce the quantity of the data to be visualized, without affecting visualization quality. In order to achieve this, our system uses a fast polygonal approximation algorithm for different map scales along with a bounding box solution.

1 INTRODUCTION

Mobile devices with geo-positioning facilitate the acquisition of location-based data. This allows people to track their outdoor movements while performing physical exercises or when traveling. Companies can manage their geographical information in real-time (Martín et al., 2008) and track the movement of their own vehicles in order to solve problems such as fleet management (Jakobs et al., 2001) or traffic congestion (McCullough et al., 2011). The collected tracks are usually uploaded to an online system in order to be viewed, managed and analyzed. However, accessing and visualizing large amount of data is time consuming.

We present *MOPSI Routes*, a complete system for storage, retrieval and visualization of GPS tracks that overcomes the most common disadvantages of similar systems. For example, existing real-time web based systems, such as www.gmapgis.com and www.gpsvisualizer.com, do not have the possibility to plot large number of points and tracks on the map. In such cases, displaying becomes slow and visualizing overlapping tracks is difficult. Other solutions, such as TopoFusion (Morris et al., 2004), propose combining and intersecting GPS tracks in order to create trails and minimize the data needed to be displayed, although the goal, creating a GPS network of trails, is different. Our solution is to display all the recorded tracks in real time by reducing the number of points that are plotted. This is done by fast multi-resolution polygonal approximation algorithm described in (Chen et al., 2012), which achieves better approximation result than the existing competitive methods. Furthermore, we minimize the time needed for drawing by using a bounding box solution for plotting only the points that are visible to the user.

MOPSI Routes is available as a part of MOPSI services (cs.uef.fi/mopsi) and addresses the issues of storage, querying, retrieval and visualization of GPS tracks, first described in (Waga et al., 2012b). Users voluntarily upload their GPS tracks using our mobile application, which is available for most modern mobile operating systems (Android, Windows Phone, iOS and Symbian).

Similar research projects include GeoLife (Zheng et al., 2008), the system presented in (Alahakone et al., 2009) and StarTrack (Ananthanarayanan et al., 2009).

GeoLife (Zheng et al., 2008) is a project which focuses on visualization, organization, fast retrieval and understanding of GPS track logs. The main goal of the project is understanding people lives based on raw GPS data. The main contribution is visualizing GPS data over digital maps by indexing the GPS trajectories based on uploading behavior of users. Similarly to MOPSI Routes, tracks are searched using spatial range and time query.

The tool described in (Alahakone et al., 2009) is used for manipulating, integrating and displaying geographical referenced information. The main purposes for the tool are path planning and navigation of mobile objects. The tool can be used in

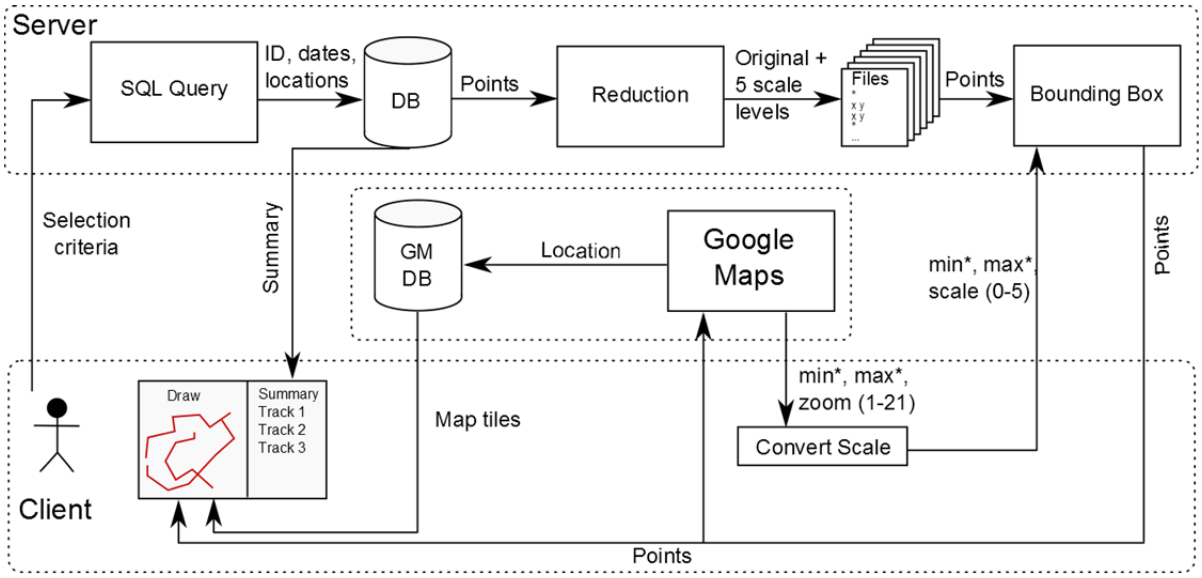


Figure 1: Typical Workflow of MOPSI Routes.

several applications such as: tracking, fleet management, security management and industrial robot navigation. Similarly to our system, a spatial database is used for storing tracks and points and Google Maps API to display the tracks. It presents a general approach in handling GPS data and it can be used in a variety of applications that use track recording, navigation and track planning. It requires that the user selects the points and defines the tracks, whereas our application automatically detects and segments the tracks.

StarTrack (Ananthanarayanan et al., 2009) and its improved version (Haridasan et al., 2010) describe tracks of coordinates as high-level abstraction for various types of location-based applications. The system supports recording, comparison, clustering and querying tracks. Experimental results show that the system is efficient and scalable up to 10.000 tracks. The improved version was extended to operate on collections of tracks, delay query executions and permit caching of query results. Other improvements are canonicalization based on road networks, and use of track trees for similarity.

2 SYSTEM DESCRIPTION

MOPSI Routes can be accessed at cs.uef.fi/mopsi/routes. The typical workflow of the system is presented in Fig. 1, whereas Fig. 2 shows example of tracks collection from one user.

In the first step, user selects the tracks to be displayed by several criteria such as time, location, duration and length. Tracks that match the criteria are retrieved from database and processed before displaying to the user. During the processing phase, the points belonging to the retrieved tracks undergo approximation process that reduces the number of points needed for the specific map scales. Points that are outside the visible area of the map are omitted by applying a bounding box. In the final step, the remaining points are shown on the map and the user can browse through them using map view (panning and zooming) or using list view to see additional information and statistics of each route.

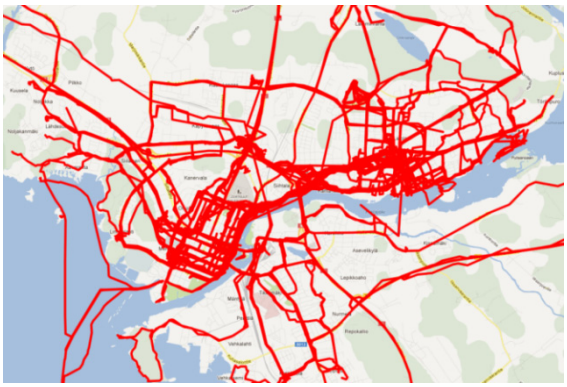


Figure 2: Example of user tracks collection.

2.1 Data Acquisition and Storage

MOPSI allows collecting tracking data using smartphones. The mobile application records the user's location and timestamp at a predefined

interval (usually 1-4 seconds). The data is saved to database on server immediately if internet connection is available, or buffered on the device if internet connection is not available or the application is in offline mode.

Tracks are first saved as individual points in the database, and track objects are created and updated real time when new points are received. Each track object contains not only the points but includes several basic statistics such as start and end time, bounding box and number of points. Segmentation and classification statistics are also stored. Analysis and classification of GPS tracks is described in details in (Waga et al., 2012a). Furthermore, each track is stored in its original and in a simplified form with reduced number of points. The approximated tracks are computed for 5 different zoom levels in order to speed up the visualization process. This limits the number of points drawn on the map without losing significant information about the shape of the GPS track. The analyzed and the approximated tracks are computed immediately when the points are uploaded.

The tracks are created and updated real time and tracking points are handled immediately after they have been uploaded. This process requires maintaining and updating track statistics and information constantly when user is recording a new track. To ensure this, there is a process running constantly on server that checks periodically (every 1 minute) if any track needs to be updated. When new tracking points are uploaded, they are either used for creating a new track object or merged with the existing points and inserted into list of the track's points in time order. The existing tracks are updated in the case that new tracking points belonging to an older track are received with significant delay caused, for example, by poor internet connection.

2.2 Different Map Scales

The tracks recorded in our system carry far more data than needed for visualization. Full data is needed for analysis, and therefore, complete GPS tracks must be stored. However, in the rendering process for a web browser, reduced number of points is sufficient to present the shape of track to user. We apply here a multi-resolution polygonal approximation algorithm described in (Chen et al., 2012). The algorithm is fast and achieves good quality approximation of the tracks. It is applied to every track and returns approximation of a track in 5 different map scales. The algorithm time complexity is O(N) (Chen et al., 2012) and the results are stored to avoid running algorithm repeatedly when the same track is displayed again.

Figure 3 shows an example of the original and approximated tracks. The original track contains 575 points and it is approximated in different map scales with 44, 13, and 6 points respectively. Suitable approximation error tolerance is selected for each map scale, and the visualization quality is not affected by the approximation, but rendering time is reduced significantly.

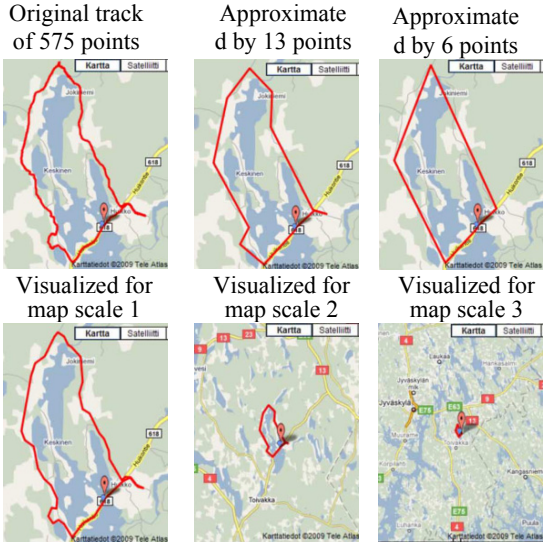


Figure 3: Visualization of a sample track.

2.3 Bounding Box

The purpose of the bounding box is to draw on the map only the points that are visible to user, see Fig. 4. Therefore, we select only points that user will see using the current map scale and location (*bounding box* of the map) at the moment of query. In addition, we draw also points that are outside the bounding box, but within immediate neighborhood (50% extension of screen size). In this way, we allow fast panning and zooming.

The bounding box is implemented as a function that gets coordinate of north, east, south and west of the map visible on the screen. Map scale is also passed, so that points from the correct approximation can be selected. The function is applied to every track and for every point it checks if the point lies inside the bounding box. Time complexity of the bounding box is linear and it is computed entirely on server.

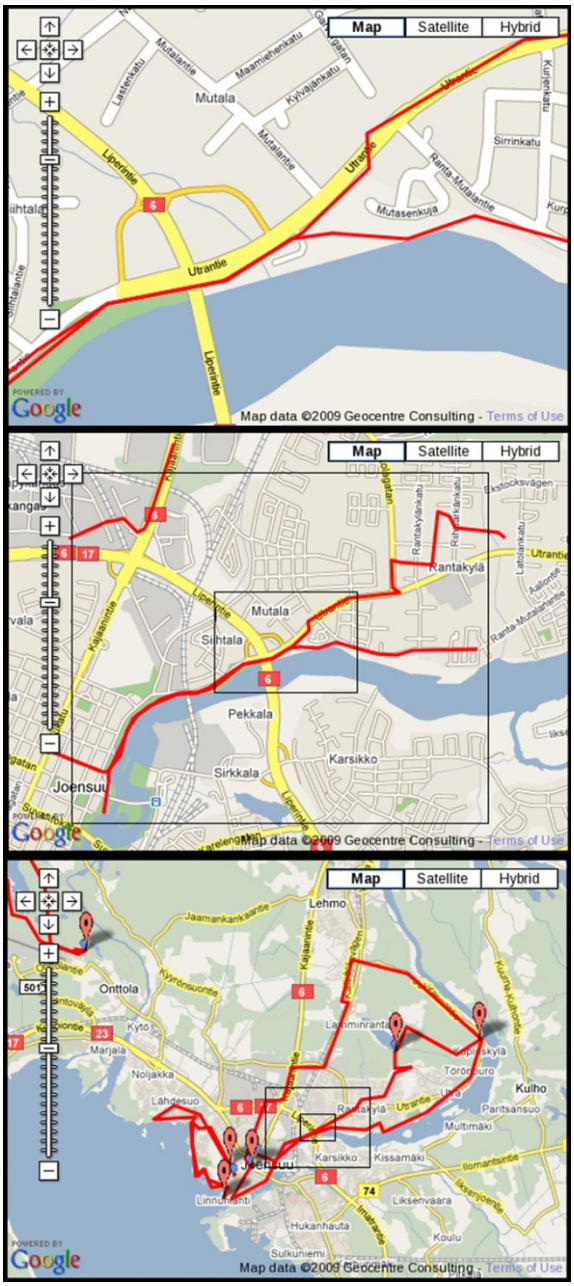


Figure 4: How the bounding box works (from top to bottom): what user sees on screen, what is drawn on map, all tracks selected.

2.4 Displaying Tracks on Map

In MOPSI, we use Google Maps to visualize the data (see Fig. 5). However, user can select different type of maps that are displayed as overlay over Google Maps. We support OpenStreetMaps and detailed orienteering maps in Joensuu area where most MOPSI users come from.

There are several search options available. The main search criterion is time, thus only tracks in the selected time period are shown. In addition, other criteria can be applied. For example, tracks can be filtered by minimum and maximum length and duration. Moreover, it is possible to search for tracks that start and end around a certain location.

Figure 5: Displaying tracks on the map.

3 RESULTS

We measure the time spent between sending request to the system and presenting the result to the user. The time elapsed from user's query to the time of displaying the tracks on the screen using our system is compared with the same system that does not have reduction.

In all measurements, we ignore the time needed for data transfer. However, in weak internet access this might become bottleneck, and therefore, we design the system so that it minimizes data transfer. That allows using the system on computers with slower internet connection as well as on tablets that usually have limited bandwidth.

Table 1: Collections used for our experiments.

User	Tracks	Points	Length (km)	Duration (h)
Pasi	784	1,216,039	8,535	669
Karol	650	1,015,939	9,655	442
Radu	429	613,684	4,604	188

We present measurements for 3 sample users from Table 1. The original tracks consist of large number of points. In MOPSI, there are users having over one million points, which shows the need for reducing the number of points being displayed as none of the browser could handle such large number of points (Chen et al., 2009). In Table 2, we show the number

of points from the original tracks within the selected time period and the number of points from the approximated tracks. The zoom level of the map is selected in such manner that all the tracks are visible on the map.

Figure 6 presents the time needed to display tracks in a selected period for three test users. The process is divided into three phases: querying database, computing bounding box and drawing in browser. Results show that the time needed for showing all the tracks of the user with the biggest collection is about 2.5 seconds.

Table 2: Number of points in original (left) and in the approximated tracks (right) in the selected time period for user Pasi.

	Original	**Approximated**
all	1,216,039	9,064
year	424,709	3,088
month	46,669	331
week	11,204	903
recent	3,328	141

Figure 7 shows average time percentages spent in each of the three phases. Querying data takes most of the time. Calculating bounding box is a fast process that additionally speeds up drawing tracks on map, so that it takes only 14% of time.

The approximation algorithm is necessary to reduce the number of points displayed. Without it, it is not possible to display all tracks because the web browser would crash. The number of points browsers can handle depends on available resources. Displaying thousands of points significantly slows down web browsers. Nevertheless, even if browser can display all the points in tracks, the time needed for the process increases.

Table 3: Size of files (in bytes) with original and approximated tracks for user Karol.

	Original	Approximated
week	14.000	148
month	346.000	2280
year	4.056.000	69.000
all	11.595.000	129.000

Bigger number of points slows down the bounding box algorithm and often leads to memory issues. Moreover, approximation algorithm reduces files sizes as shown in Table 3 and preserves bandwidth used to retrieve data from server.

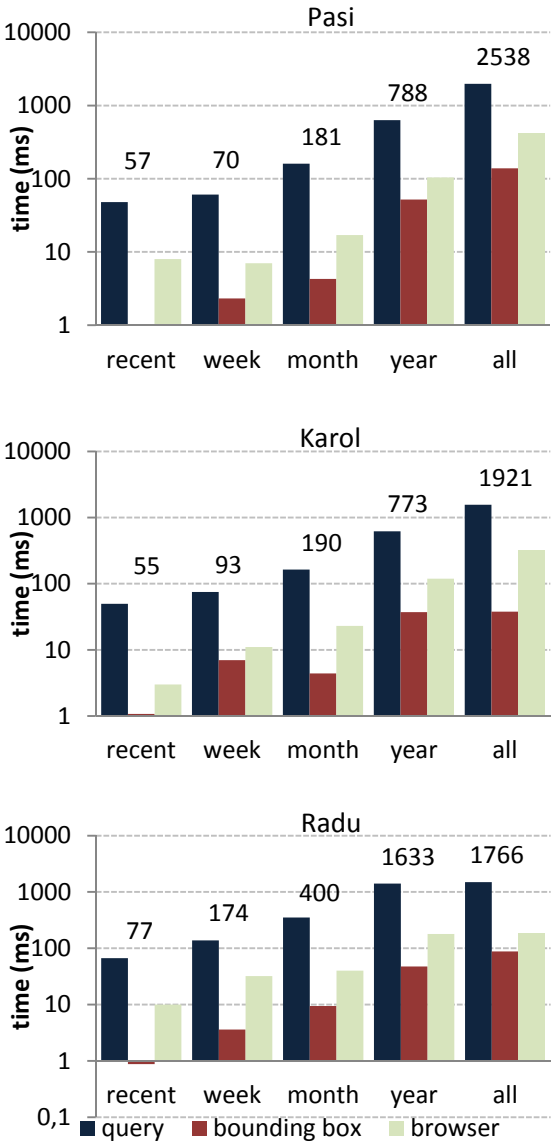


Figure 6: Display times of track collection for users Pasi, Karol and Radu.

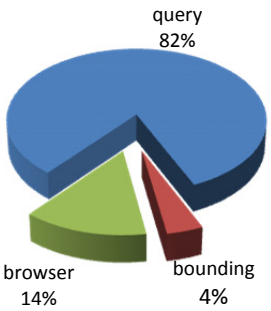


Figure 7: Average time percentage used for performing each operation of the system.

Experiments show that applying bounding box decreases time needed to draw tracks on map. Fig. 8 shows a sample case from the experiments. In this case the same set of tracks was requested at the same zoom level, but the map was focused in two different places, Finland and Poland. In Finland the collection of tracks is big, whereas in Poland there are only several tracks available. Because of applying the bounding box solution, not all the tracks have to be displayed and the time to show the tracks when map shows fewer tracks (Poland area) is significantly shorter. Figure 8 also shows how reducing number of points affects the display time.

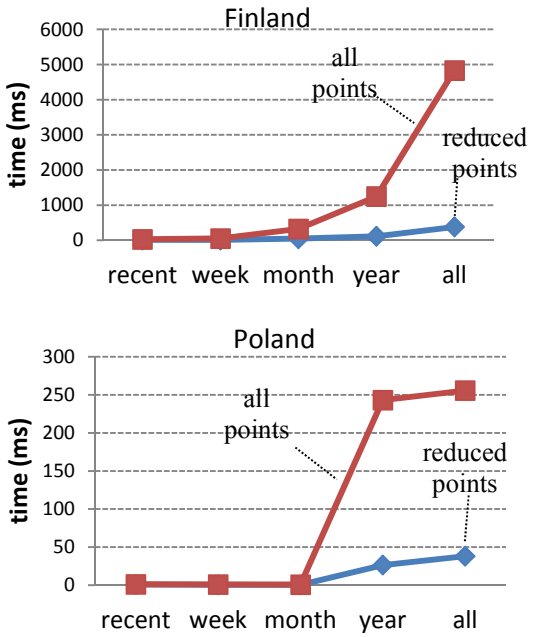


Figure 8: Example of querying the same track collection the same zoom level and focused in Finland (large collection, top) and Poland (small collection, bottom).

In comparison with the existing web based systems for visualizing GPS tracks, our system can display data consisting of significantly more points. For example, a track with about 10.000 points is displayed by our system in 1 second whereas GPS visualizer (www.gpsvisualizer.com) and GMapGis (www.gmapgis.com) need approximately 5 seconds. Moreover, user interaction is not slowed down in our system, when large number of points being is displayed.

4 SUMMARY

We presented a complete real time system to collect and visualize GPS tracks. Our motivation is to offer a system that is capable of handling large amount of GPS data so that user can access them in real time. The results show that our system is efficient even with large point collection. The most important part is the algorithm reducing the number of points to be displayed. Combined with a bounding box solution, the requested tracks can be accessed within about 2.5 seconds and the collection can be panned and zoomed with insignificant delay. The developed system can be used as a basis for more advanced analysis of GPS tracks, such as similarity and movement type detection.

Although, the system is efficient, there are still ways to improve it. For instance, now we reduce the number of points of one track only, but not when multiple tracks are overlapped. Further improvement could be achieved by clustering partial track segments. Moreover, the query phase should be optimized to minimize time needed to retrieve data.

REFERENCES

Alahakone, A. U., Ragavan, V. Geospatial Information System for Tracking and Navigation of Mobile Objects. *ICAIM 09. Singapore*, July, 2009.

Ananthanarayanan, G., Haridasan, M., Mohomed, I, Terry, D., Chandramohan, A. T. StarTrack: a Framework for Enabling Track-Based Application. *ICMAS 09. Kraków, Poland*, June 2009.

Chen, M., Xu, M., Fränti, P. A Fast O(N) Multi-resolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. *IEEE Trans. on Image Proc*. 21(5). 2012.

Chen, Y., Jiang, K., Zheng, Y., Li, Ch., Yu, N. Trajectory Simplification Method for Location-based Social Networking Services. Int. Workshop on *Location Based Social Network.* Seattle, USA, November 2009.

Haridasan, M., Mohomed, I., Terry, D., Chandramohan, A. T., Li, Z. StarTrack Next Generation: A Scalable Infrastructure for Track-Based Applications, 2010.

Jakobs, K., Pils, C., Wallbaum, M. Using the Internet in Transport Logistics - The Example of a Track & Trace System. *Networking ICN*, 194-203, 2001.

Martín S., Cristóbal E.S., Gil R., Díaz G., Oliva N., Castro M., Peire J. Finding the Way: Services for a Multi-View and Multi-Platform Geographic Information System. *WEBIST (2)*, pp.267-270, 2008.

McCullough, A., James, P., & Barr, S. (2011). A Service Oriented Geoprocessing System for Real Time Road Traffic Monitoring. *Transactions in GIS*, 15(5), 651-665, 2011.

Morris, S., Morris, A., Barnard, K. Digital Trail Libraries. *ACM/IEEE-CS Joint Conf. on Digital Libraries*, pp. 63-71, June, 2004.

Waga, K., Tabarcea, A., Chen, M., Fränti, P. Detecting

Movement Type by Route Segmentation and Classification. CollaborateCom, Pittsburgh, USA, October 2012.

Waga, K., Tabarcea, A., Mariescu-Istodor R., Fränti, P. System for Real Time Storage, Retrieval and Visualization of GPS Tracks. *ICSTCC*, Sinaia, Romania, October 2012.

Zheng, Y., Wang, L., Zhang, R., Xie, X., Ma, W.-Y. GeoLife: Managing and Understanding Your Past Life over Maps. *Int. Conf. on Mobile Data Mgmt*., Beijing, China, April 2008.

Paper II

Mariescu-Istodor R., Tabarcea A., Saeidi R. & Fränti P.
”Low complexity spatial similarity measure of GPS trajectories”
International Conference on
Web Information Systems & Technologies
Barcelona, Spain, pp. 62-69, 2014

Low Complexity Spatial Similarity Measure of GPS Trajectories

Radu Mariescu-Istodor, Andrei Tabarcea, Rahim Saeidi and Pasi Fränti
Speech and Image Processing Unit, School of Computing, University of Eastern Finland, Joensuu, Finland
{*radum, tabarcea, rahim, franti*}*@cs.uef.fi*

Keywords: GPS Trajectory, Spatial Similarity, MGRS, Cell Approximation, Sampling Frequency, Interpolation, Dilation.

Abstract: We attack the problem of trajectory similarity by approximating the trajectories using a geographical grid based on the MGRS 2D coordinate system. We propose a spatial similarity measure which is computationally feasible for big data collections. The proposed measure is based on cell matching with a similarity metric drawn from Jaccard index. We equip the proposed method with interpolation and dilation to overcome the problems missing data and different sampling frequencies when comparing two trajectories. The proposed measure is implemented online in the framework of Mopsi[a].

[a] cs.uef.fi/mopsi

1 INTRODUCTION

In recent years, GPS technology has been widely available in consumer devices, especially in smartphones[1], which count as more than a half on total mobile phone sales[2]. Furthermore, most of the users utilize their phone to find their location, amongst other services[3]. The wide availability of GPS-enabled smartphones that are also connected to the Internet has made the collection of large amount of location-based data possible. Such data includes geo-tagged photos, videos and geographical trajectories. Collecting geographical trajectories has practical applications in fleet management, sports tracking, recommending tourist trajectories, improving navigation and determining mobility patterns.

Having a large-scale collection of GPS trajectories raises the challenge of how to organize the data, how to present it in a meaningful way and how to filter out irrelevant data. Computing *trajectory similarity* is a tool that can be used in addressing those challenges (Agrawal et al., 1993). A problem in computing similarity of GPS trajectories is that the large amount of data does not permit processing raw trajectories in real time.

Time series analysis of one-dimensional data across the time has been used for analyzing stock changes, weather data and biomedical measurements (Hamilton, 1994; Chan and Fu, 1999; Worsley and Friston, 1995; Lange and Naumann, 2011). Despite the significant research output on time series analysis, the concept of computing similarity for traces of moving objects in the framework of spatio-temporal databases has been studied much less. Finding *k-nearest* trajectories, indexing and clustering of spatio-temporal data are among the recent directions of research with many applications to make queries in moving object databases (Frentzos et al., 2007a; Ni and Ravishankar, 2007; Frentzos et al., 2007b; Güting et al., 2010; Pelekis et al., 2011). These algorithms can be applied also for measuring the trajectory similarity (Hu and Steenkiste, 2006).

Using *Euclidean distance* is not practical for the case that the length of two trajectories are not equal (Yanagisawa et al., 2003). *Dynamic time warping* handles matching two sequences of different length but it is very sensitive to noisy data (Berndt and Clifford, 1994). Algorithms like *longest common subsequence* (LCS) (Vlachos et al., 2002b; Vlachos et al., 2002a) or *edit distance on real sequence* (EDR) (Chen et al., 2005) are designed to account for noisy and missing data but they are not perturbation free. Considering M trajectories of N points on average, the computational complexity of these algorithms is at minimum $O(M^2 \cdot N^2)$. Hence, these algorithms cannot provide real-time results when dealing with a large collection of data.

These algorithms do not utilize time stamps. By

[1] abiresearch.com/research/product/1005746-mobile-device-user-interfaces

[2] gartner.com/newsroom/id/2623415

[3] pewinternet.org/Reports/2012/Location-based-services.aspx

using the timing information a complete movement profile can be provided and the similarity of two trajectories can be used in trajectory clustering applications. The similarity measurement in LCS and EDR are based on point-to-point distance calculations. In the event of having two trajectories with different sampling frequency, LCS and EDR cannot provide correct similarity measure (Frentzos et al., 2007b). Although it is always possible to use a trajectory reduction or approximation algorithm to represent a trajectory with far less representatives for similarity calculation, the quality of such an approximation algorithm and overhead computational complexity is debatable (Ni and Ravishankar, 2007).

In this paper, we propose a fast method of computing trajectory similarity by approximating the trajectories using a geographical grid based on a 2D coordinate system. This process reduces a trajectory from points to cells with order of magnitude less details in representation and subsequently in distance calculations. We employ an asymmetric similarity metric inspired by Jaccard index. Dealing with GPS data collection, it is common to have bunch of data points lost or compare trajectories traveled by car with walking speed trajectories. We propose interpolation and dilation of trajectories represented as cells to overcome these difficulties. In the results section we simulate missing data and trajectory sampling frequency mismatch with two example trajectories and demonstrate the efficiency of the proposed approach. Conclusions are drawn after the discussion of results.

2 MOPSI

Mopsi is a research project location-based service developed at the University of Eastern Finland by Speech and Image Processing Group from the School of Computing. (Fränti et al., 2011) Mopsi offers multiple applications of location-aware systems, being a test-bed for various research topics that involve location-aware data. It contains tools for collecting, processing and displaying location-based data, such as photos or trajectories, along with social media integration. The main topics addressed in Mopsi are collecting location-based data, mining location data from web pages, processing, storing and compressing of GPS trajectories, detecting transportation mode from GPS trajectories, recommending *points of interest*, using location information in social networks, detecting users actions by using their location and building location-based games with the help of user-generated collections.

Location-based data is very common among web-

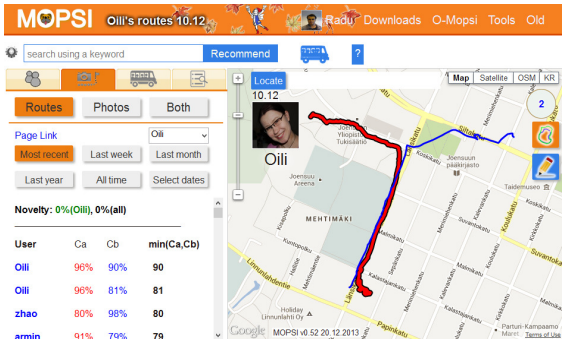


Figure 1: Mopsi application on web showing an example of two trajectories which display a common region.

pages, especially when their content describe commercial services, landmarks or public institutions. However, the location data is more commonly presented in a human-readable way and not as geographical coordinates, which are more accurate and easier to be automatically identified. We propose a method to automatically identify location information from web-pages by detecting postal addresses (Fränti et al., 2010).

Mopsi provides tools to collect GPS trajectories and it includes more than 9000 trajectories composed of over 7 million points by the end of 2013. Mopsi uses fast retrieval and displaying of the data (Waga et al., 2013) based on GPS trajectory polygonal approximation (Chen et al., 2012a). GPS trajectories are also compressed for optimizing storage space (Chen et al., 2012b). Transport mode information can be also retrieved by automatically analyzing GPS trajectories (Waga et al., 2012). The algorithm uses a second order Markov model to segment the trajectories and to detect car, bicycle, running or walking transportation modes.

The relevance of location-based media can be assessed by considering several aspects such as time, location, content or social network (Fränti et al., 2011), which are used to create a context for each user. A personalized recommender system can recommend relevant data based on user location and user context (Waga et al., 2011). Such data can be geotagged photos, services confirmed by administrators or GPS trajectories. Users can share their location in real-time by using mobile phone location-aware applications. This allows for the detection of various location-based actions such as meetings, visiting or passing-by *points of interest* (Mariescu-Istodor, 2013). Mopsi also includes location-based games, such as O-Mopsi (Tabarcea et al., 2013), which is an orienteering game using the data from a user-generated photo collection.

Mopsi provides tools for collecting location-based

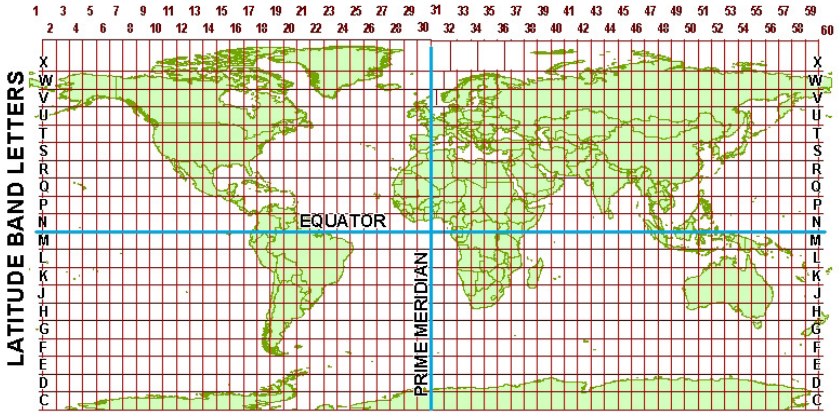


Figure 2: MGRS grid zones (source[4]).

data with mobile devices. It is available on most mobile operating systems (Android, iOS, Windows Phone, Symbian). The server-side processes the data collected by the user and displays the data collection. It also provides social features and integration which social media, with functionalities such as chatting, friends tracking and sharing data to Facebook. The Mopsi routes module provides tools for trajectory recording and displaying the large amount of data in reasonable time. Trajectory similarity is the newest addition to the Mopsi routes module.

3 TRAJECTORIES

In Mopsi we record a user's location at a certain time as a point $\mathbf{p}_k = (x_k, y_k, t_k)$, where x_k is the latitude, y_k is the longitude and t_k is the timestamp of point k. An ordered sequence of these points, defines a spatial trajectory $\mathbf{R} = (\mathbf{p}_1, \ldots, \mathbf{p}_K)$. We calculate the similarity between a reference trajectory $\mathbf{R}_a$ and all the other $M-1$ trajectories in the database, $\mathbf{R}_m$, $m = 1, \ldots, M$.

The similarity of two trajectories can be calculated as the Jaccard index:

$$J(\mathbf{R}_a, \mathbf{R}_m) = \frac{|\mathbf{R}_a \cap \mathbf{R}_m|}{|\mathbf{R}_a \cup \mathbf{R}_m|}, \quad (1)$$

Instead of this symmetric measure we want to find out if the reference trajectory is completely covered by another trajectory. Thus, we consider the following asymmetric similarity metric:

$$Sim(\mathbf{R}_a, \mathbf{R}_m) = \frac{|\mathbf{R}_a \cap \mathbf{R}_m|}{|\mathbf{R}_a|}, \quad (2)$$

$$Sim(\mathbf{R}_m, \mathbf{R}_a) = \frac{|\mathbf{R}_a \cap \mathbf{R}_m|}{|\mathbf{R}_m|}. \quad (3)$$

[4]earth-info.nga.mil/GandG/coordsys/grids/universal_grid_system.html

The first one shows what percentage of $\mathbf{R}_a$ is shared with $\mathbf{R}_m$ and the second shows what percentage of $\mathbf{R}_m$ is shared by $\mathbf{R}_a$. The way that we perform intersection operator is described in the following sections after we quantize the trajectories into cells.

3.1 Cell Approximation

In a preprocessing step, we generate a cell representation for a trajectory after it has been recorded. The Military Grid Reference Systems (MGRS) is an alpha-numeric system for expressing UTM/UPS coordinates. MGRS is used by NATO to locate points on earth. A single alpha-numeric value references a position that is unique for the entire earth (see Figure 2). MGRS is a projected coordinate system which uses a 2-dimensional Cartesian horizontal position orientation, so that locations are identified independently of vertical position. MGRS shares several characteristics with UTM such as the division of earth into projection zones and using easting and northing in meters within a designated zone. The main differences are that a MGRS zone is a 100km square within a UTM zone, whilst a UTM zone is usually 6 degrees in east-west and 8 degrees in north-south area and also that the notation of the areas is different. Based on the coordinate resolution, MGRS can define a grid with square cells with the length starting from 100km up to 10m or even 1m.

We approximate a trajectory $\mathbf{R} = \{(x_k, y_k)\}_{k=1}^{K}$ by a sparse binary matrix representation $\mathbf{C}$ where,

$$(\mathbf{C})_{ij} = \begin{cases} 1 & 0 < x_k - iL < L, 0 < y_k - jL < L \\ 0 & \text{Otherwise} \end{cases}, \quad (4)$$

where L stands for the cell length (25 meters in this paper) and indexes i and j span over in horizontal and vertical cells that trajectory R is residing inside. Figure 3 shows how the reference trajectory is approxi-

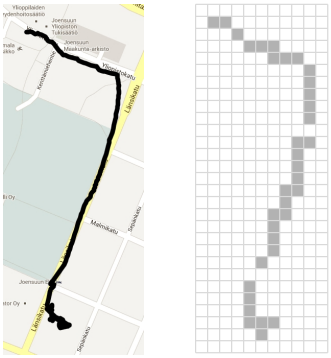


Figure 3: Example of a trajectory of 420 points being represented by 35 cells using the approximation in Equation 4. The cell representation is not continuous. The gaps appear because of the fixed cell size, variations in movement speed (or different sampling frequencies) and missing GPS locations. It is likely for such gaps to appear especially when users are moving by car, train or plane.

mated by cells. Generating the cell representation for a trajectory of average length of N points is done in $O(N)$ time.

3.2 Measuring Similarity

The similarity between two trajectories $\mathbf{R}_a, \mathbf{R}_m$ can now be calculated as:

$$Sim(\mathbf{R}_a, \mathbf{R}_m) = \frac{\|\mathbf{C}_a \odot \mathbf{C}_m\|_0}{\|\mathbf{C}_a\|_0}, \tag{5}$$

where $\mathbf{C}_a$ and $\mathbf{C}_m$ are the cell representations of $\mathbf{R}_a$ and $\mathbf{R}_m$, respectively, $\mathbf{C}_a \odot \mathbf{C}_m$ is a *Hadamard product* of two matrices $\mathbf{C}_a$ and $\mathbf{C}_m$ defined as $(\mathbf{C}_a \odot \mathbf{C}_m)_{ij} = (\mathbf{C}_a)_{ij} \cdot (\mathbf{C}_m)_{ij}$ and $\|\mathbf{C}\|_0$ represents the ℓ_0-quasinorm. In implementation, $\mathbf{C}_a$ and $\mathbf{C}_m$ are multiplied element by element and then we measure the number of non-zero elements. Figure 4 shows two sample trajectories being matched.

Assuming we have the cell representation $\mathbf{C}$ of a reference trajectory $\mathbf{R}$ we calculate the similarity for all trajectories in database in two steps. First, we find all the trajectories which share at least one cell with the reference trajectory. This has a time complexity of $O(N' \cdot (q + M'))$ where q represents the steps needed by the database system to perform the search ($N' \ll N$ and $M' \ll M$). In contrast to the average length N of a trajectory $\mathbf{R}$, we define $N' = \|\mathbf{C}\|_0$ as the number of non-zero elements in cell-approximated version of $\mathbf{R}$. In a similar way, M' indicates the number of other trajectories that share at least one cell with trajectory $\mathbf{R}$. Secondly, we calculate the trajectory similarity according to Equation (5) with a time complexity of $O(M' \cdot N')$. The overall complexity of the similarity

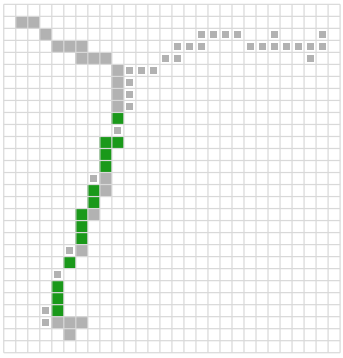


Figure 4: Matching two trajectories using the cell representation. The green cells denote the reference trajectory and the gray cells represent the other trajectory. The 'x' symbol is used to mark the cells shared by two trajectories; $Sim(\mathbf{C}_a, \mathbf{C}_m) = 40\%$ and $Sim(\mathbf{C}_m, \mathbf{C}_a) = 31\%$

scoring is $O(M' \cdot N')$ (assuming q constant by adding a proper indexing structure in the database).

In Figure 4 the straighforward application of the similarity scores yield similarity scores of 40% and 31% even though the trajectories seem to have more than 50% similarity by visual inspection. In the next subsections we analyze why this happens.

3.3 Interpolation

When the user is traveling fast or when recording frequency is low we notice gaps in the trajectory representation by cells. Gaps can also appear due to lack of GPS signal. Figure 5 shows three examples when different sized gaps appear in the cell representation of a trajectory. In cell approximation stage in Equation 4, we process the trajectory data points in the sequence they are recorded. In this way, the sequence of cells being detected as "1s" are used to determine if the next cell is connected to the current cell and find a potential gap in cell-approximation.

In order to fill the gap, the line equation between two cells is obtained from the start and end points as

$$j = f(i) = \frac{j_2 - j_1}{i_2 - i_1}(i - i_1) + j_1 \tag{6}$$

where i_1 and j_1 are the coordinates of one cell and i_2, j_2 are the coordinates of the other cell. The line in Equation 6 is then sampled by the cells that it is passing through and then set respective cell values as $(\mathbf{C}_{ij}) = 1$.

By performing interpolation, the trajectory similarity presented in Figure 4 is now updated as plotted in Figure 6. The similarity values are still below the visual expectations. The reason is that two cell representations may not overlap even though the trajectories are close to each other.

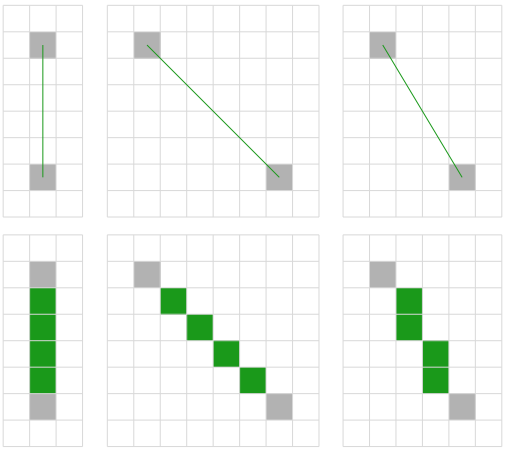


Figure 5: Interpolation between two cells in order to fill a gap; three example situations are depicted.

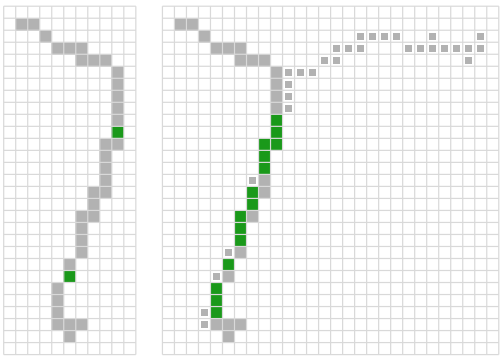


Figure 6: The trajectory having gaps is interpolated and the matching of the two trajectories becomes: $Sim(\mathbf{C}_a, \mathbf{C}_m) = 41\%$ and $Sim(\mathbf{C}_m, \mathbf{C}_a) = 33\%$.

3.4 Dilation

A frequent situation is that two nearby trajectory segments are evolving along each other in cell representation instead of overlapping. An example is provided in Figure 7 We solve this issue by applying *morphological dilation* on the trajectories and taking into account the neighbouring cells of a trajectory. We define $\mathbf{C}^d$ as a result of *binary dilation* of sparse binary representation $\mathbf{C}$ by *binary structure* $\mathbf{S}$ with

$$\mathbf{C}^d = \mathbf{C} \oplus \mathbf{S} = T(\mathbf{C} * \mathbf{S}), \tag{7}$$

where $\oplus$ defines the binary dilation and $*$ indicates the convolution operator. In the Equation 7, $T(\cdot)$ stands for *binarization transform* as

$$T((\mathbf{C} * \mathbf{S})_{ij}) = \begin{cases} 0 & 0 \leq (\mathbf{C} * \mathbf{S})_{ij} < 1 \\ 1 & \text{Otherwise} \end{cases} \tag{8}$$

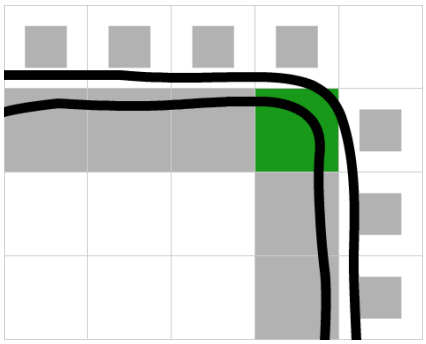


Figure 7: We see that two trajectories which are close enough to be considered similar can be represented by different cells. Only a single cell is shared by the cell representation of the two trajectories.

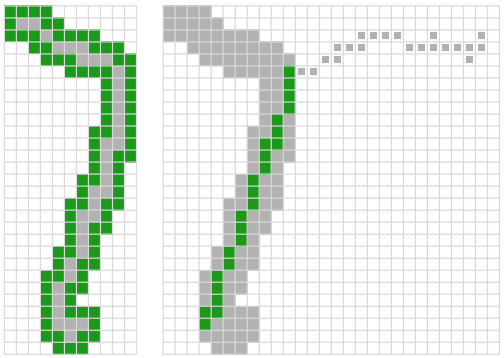


Figure 8: The reference trajectory is dilated and the matching of the two trajectories becomes: $Sim(\mathbf{C}_a, \mathbf{C}_m) = 64\%$ and $Sim(\mathbf{C}_m, \mathbf{C}_a) = 53\%$.

Figure 8 shows how a trajectory is dilated with the following structure

$$\mathbf{S} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}. \tag{9}$$

Then the two trajectories are matched when one of the trajectories is dilated. The similarity score is now calculated with $\mathbf{C}_a$ and $\mathbf{C}_m^d$ as in Equation 5. Typically the number of cells used in the trajectory representations increases by a factor of 3 when dilation is applied.

4 RESULTS

We implement our method in a real-world application, as a prototype using the Mopsi project route analysis module [5]. We investigate issues that may appear

[5] cs.uef.fi/mopsi/?tab=routes&userId=13&routeId=137882401 9381&similarity=true

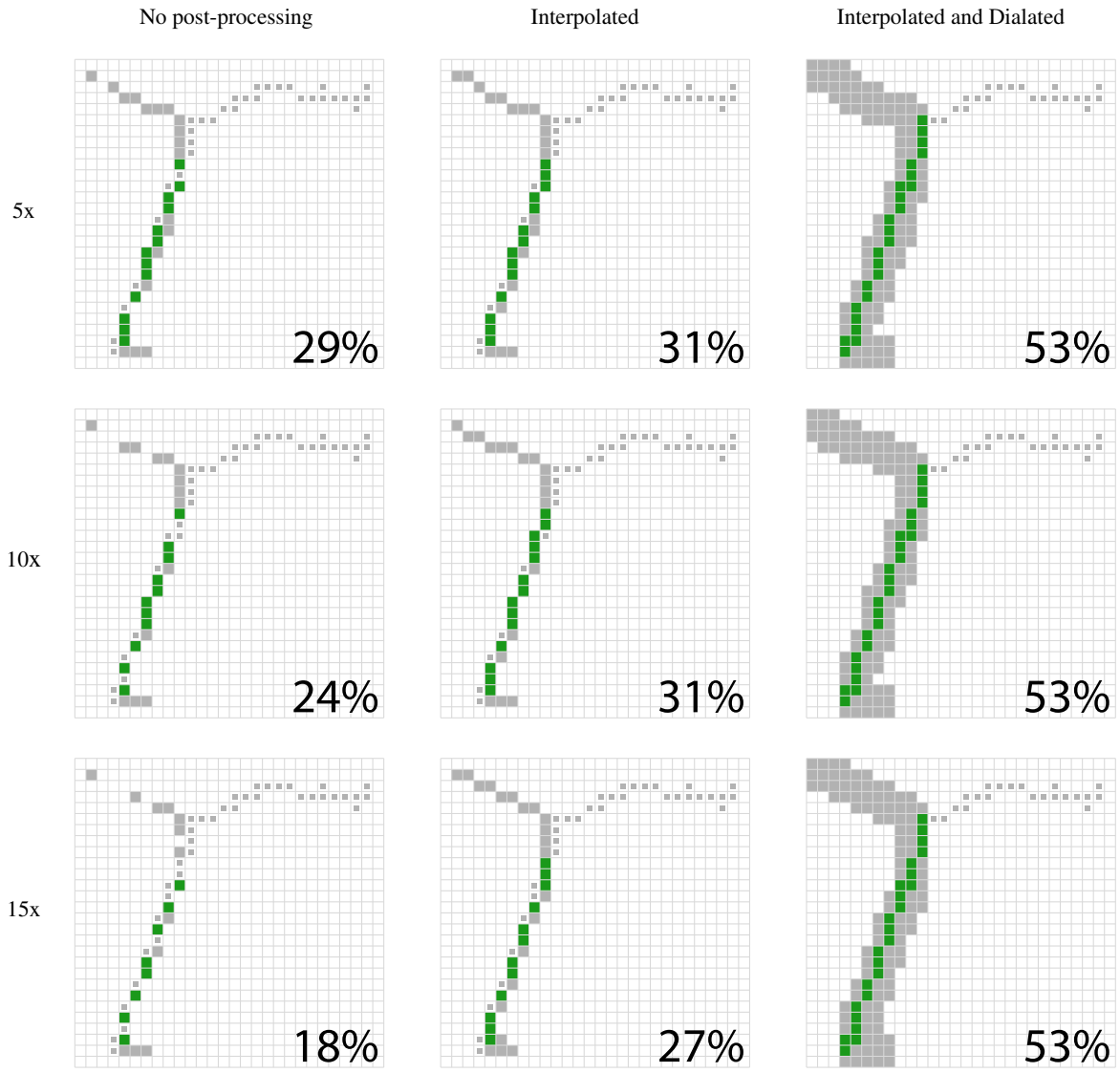


Figure 9: Simulating different sampling frequencies by subsampling the reference trajectory with a factor of 5x, 10x and 15x.

when collecting GPS trajectories in a practical application such as different sample rates, interpolation of collected points or breaks in the GPS signal caused by technical or environmental problems.

Firstly, as shown in Figure 9, we investigate how a different sampling frequency impacts the similarity score calculation. The reference trajectory is subsampled with factor f by only keeping every f^{th} element from the original trajectory. We notice that the interpolation step doesn't increase the similarity scores significantly. However, when followed by dilation, the similarity score indicates robustness against variations in sampling frequency which is a desired property for a trajectory matching procedure.

The other common issue while recording a trajectory is loss of location information for a brief period of time. This can happen, for example, if the user goes through a building, a tunnel or simply due to device software error. We simulate this behavior and see how the similarity scoring is affected in Figure 10. When removing 90 points we notice that the similarity score has dropped even when using interpolation and dilation. This happened because we removed a significant amount of subsequent points (20% of the trajectory). Interpolation does not have enough information to reconstruct the trajectory appropriately and consequently, loss of many data points in a trajectory is detrimental for similarity calculations.

The proposed method is implemented in two steps for real-world application: the preprocessing step,

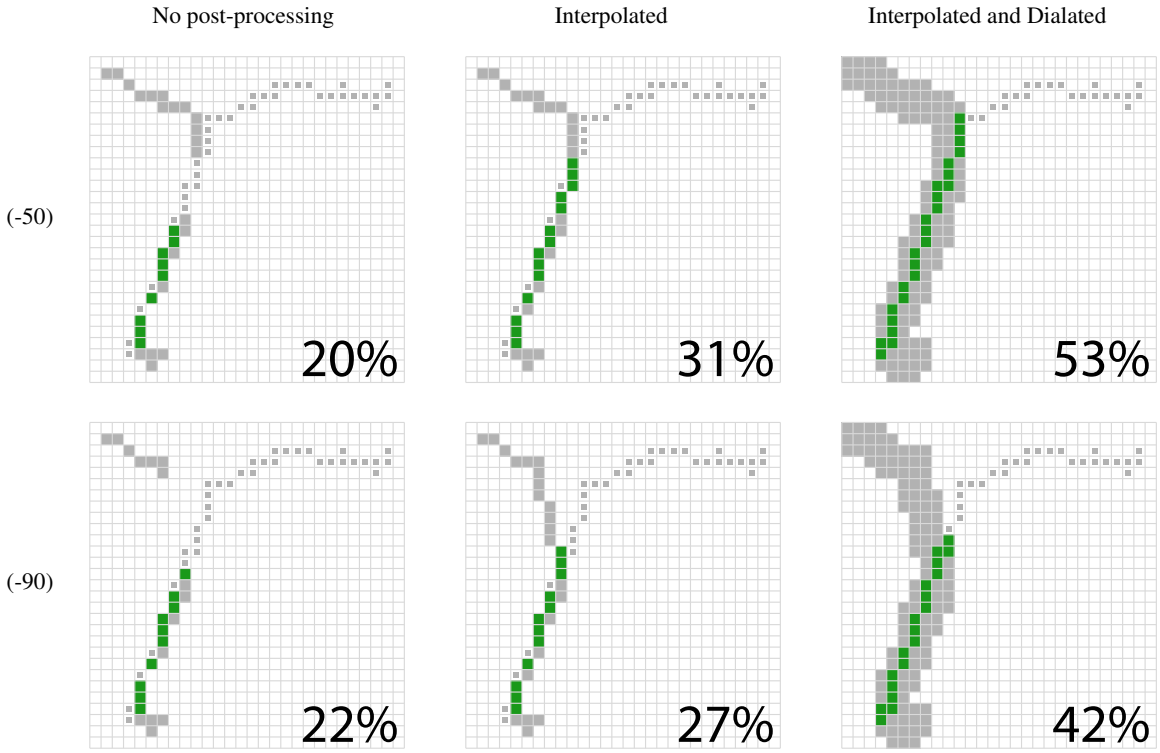


Figure 10: Simulating loss of GPS signal by removing 50 and 90 sequential points from the reference trajectory.

done when a new trajectory is added into the system and the similarity score calculation step, performed when searching all the similar trajectories of a given trajectory. When not using interpolation or dilation the time complexity for the preprocessing step is $O(M \cdot N)$ for M trajectories of average length N points. The similarity score calculation has a time complexity of $O(M' \cdot N')$. After interpolation is applied there will be an increase on the N' and M' parameters which increase, however, stay at the same order of magnitude. N' increases by the number of cells added trough interpolation and M' increases by the number of trajectories that share at least one cell with interpolated trajectory. The dilation stage increases the N' and M' parameters once more. N' typically increases by a factor of 3 and M' grows by the number of trajectories that share the cells that are added to the representation as a result of dilation. The overall complexity for M trajectories in the database is governed by $O(M \cdot N)$ for cell approximation and $O(\alpha \cdot M \cdot M' \cdot N')$ for similarity score calculation including interpolation and dilation ($\alpha \approx 6$, $M' \ll M$, $N' \ll N$). The similarity cell approximation complexity of $O(M \cdot N)$ is negligible compared to $O(\alpha \cdot M \cdot M' \cdot N')$ for score calculation. Hence, the overall computational complexity of the proposed approach is dominated by $O(\alpha \cdot M \cdot M' \cdot N')$ which is comparably much less than $O(M^2 \cdot N^2)$ for other similarity metrics presented in section 1.

5 CONCLUSIONS

We presented a method for computing similarity between trajectories in a large data collection. Because trajectories are likely to have different speed profile and missing points, interpolation and dilation techniques are employed before the scoring. We have demonstrated that the method is robust except when many points are removed and dramatically affect the structure of a trajectory. In that situation there is simply not enough information to rebuild the path and provide correct similarity values. The method was implemented in Mopsi, where for a given trajectory we display a list of similar paths in reverse order of the similarity scores.

REFERENCES

Agrawal, R., Faloutsos, C., and Swami, A. (1993). *Efficient similarity search in sequence databases*. Springer.

Berndt, D. J. and Clifford, J. (1994). Using dynamic time

warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA.

Chan, K.-P. and Fu, A. W.-C. (1999). Efficient time series matching by wavelets. In *Data Engineering, 1999. Proceedings., 15th International Conference on*, pages 126–133. IEEE.

Chen, L., Özsu, M. T., and Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM.

Chen, M., Xu, M., and Fränti, P. (2012a). Compression of gps trajectories. In *Data Compression Conference (DCC), 2012*, pages 62–71. IEEE.

Chen, M., Xu, M., and Fränti, P. (2012b). A fast $O(N)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Transactions on Image Processing*, pages 2770–2785.

Fränti, P., Chen, J., and Tabarcea, A. (2011). Four aspects of relevance in location-based media: content, time, location and network. In *Web Information Systems and Technologies (WEBIST'11), International Conference on*, pages 413–417.

Fränti, P., Tabarcea, A., Kuittinen, J., and Hautamäki, V. (2010). Location-based search engine for multimedia phones. In *Multimedia and Expo (ICME), 2010 IEEE International Conference on*, pages 558–563. IEEE.

Frentzos, E., Gratsias, K., Pelekis, N., and Theodoridis, Y. (2007a). Algorithms for nearest neighbor search on moving object trajectories. *Geoinformatica*, 11(2):159–193.

Frentzos, E., Gratsias, K., and Theodoridis, Y. (2007b). Index-based most similar trajectory search. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 816–825. IEEE.

Güting, R. H., Behr, T., and Xu, J. (2010). Efficient k-nearest neighbor search on moving object trajectories. *The VLDB Journal*, 19(5):687–714.

Hamilton, J. D. (1994). *Time series analysis*, volume 2. Cambridge Univ Press.

Hu, N. and Steenkiste, P. (2006). Quantifying internet end-to-end route similarity. In *Passive and Active Measurement Conference*, volume 2006, pages 101–110.

Lange, D. and Naumann, F. (2011). Efficient similarity search: arbitrary similarity measures, arbitrary composition. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1679–1688. ACM.

Mariescu-Istodor, R. (2013). Detecting user actions in MOPSI. Master's thesis, University of Eastern Finland.

Ni, J. and Ravishankar, C. V. (2007). Indexing spatio-temporal trajectories with efficient polynomial approximations. *Knowledge and Data Engineering, IEEE Transactions on*, 19(5):663–678.

Pelekis, N., Kopanakis, I., Kotsifakos, E. E., Frentzos, E., and Theodoridis, Y. (2011). Clustering uncertain trajectories. *Knowledge and Information Systems*, 28(1):117–147.

Tabarcea, A., Wan, Z., Waga, K., and Fränti, P. (2013). O-mopsi: Mobile orienteering game using geotagged photos. pages 300–303.

Vlachos, M., Gunopulos, D., and Kollios, G. (2002a). Robust similarity measures for mobile object trajectories. In *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on*, pages 721–726. IEEE.

Vlachos, M., Kollios, G., and Gunopulos, D. (2002b). Discovering similar multidimensional trajectories. In *Data Engineering, 2002. Proceedings. 18th International Conference on*, pages 673–684. IEEE.

Waga, K., Tabarcea, A., Chen, M., and Fränti, P. (2012). Detecting movement type by route segmentation and classification. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, pages 508–513. IEEE.

Waga, K., Tabarcea, A., and Fränti, P. (2011). Context aware recommendation of location-based data. In *System Theory, Control, and Computing (ICSTCC), 2011 15th International Conference on*, pages 1–6. IEEE.

Waga, K., Tabarcea, A., Mariescu-Istodor, R., and Fränti, P. (2013). Real time access to multiple GPS tracks. pages 293–299.

Worsley, K. J. and Friston, K. J. (1995). Analysis of fMRI time-series revisitedagain. *Neuroimage*, 2(3):173–181.

Yanagisawa, Y., Akahani, J.-i., and Satoh, T. (2003). Shape-based similarity query for trajectory of mobile objects. In *Mobile data management*, pages 63–77. Springer.

Paper III

Mariescu-Istodor R. & Fränti P.
”Grid-based method for GPS route analysis for retrieval”
ACM Transactions on Spatial Algorithms and Systems
(to appear) 2017

Grid-Based Method for GPS Route Analysis for Retrieval

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

Grids are commonly used as histograms to process spatial data in order to detect frequent patterns, predict destinations or to infer popular places. However, they have not been previously used for GPS trajectory similarity searches or retrieval in general. Instead, slower and more complicated algorithms based on individual point-pair comparison have been used. We demonstrate how a grid representation can be used to compute four different route measures: novelty, noteworthiness, similarity and inclusion. The measures may be used in several applications such as identifying taxi fraud, automatically updating GPS navigation software, optimizing traffic and identifying commuting patterns. We compare our proposed route similarity measure, C-SIM, to 8 popular alternatives including Edit Distance on Real sequence (EDR) and Frechet distance. The proposed measure is simple to implement and we give a fast, linear time algorithm for the task. It works well under noise, changes in sampling rate and point shifting. We demonstrate that by using the grid, a route similarity ranking can be computed in real-time on the Mopsi2014[1] route dataset, which consists of over 6,000 routes. This ranking is an extension of the most similar route search and contains an ordered list of all similar routes from the database. The real-time search is due to indexing the cell database and comes at the cost of spending 80% more memory space for the index. The methods are implemented inside the Mopsi[2] route module.

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

1. INTRODUCTION

In recent years, GPS technology has become widely available in consumer devices. Smartphones count as more than half of total mobile phone sales and many users utilize their phone location sensing capabilities. The wide availability of GPS-enabled smartphones makes it possible to collect large amount of location-based data. Such data includes geo-tagged photos, videos and geographical trajectories, which we will refer to as *routes*.

Mopsi is a location-based social network created by the School of Computing from the University of Eastern Finland. Mopsi users can find out who or what is around. They can track their movements, share photos and chat with friends. Mopsi includes fast retrieval and visualization of routes [Waga et al. 2013] using a real time route reduction technique [Chen et al. 2012]. Transport mode information is automatically inferred by analyzing speed variance of the route [Waga et al. 2012]. Movement is classified as either: walking, running, cycling or car. Stop points are also detected. In this paper we define four new route measures: *novelty*, *noteworthiness*, *similarity* and *inclusion*.

This work was supported by Nokia Scholarship grant.

Author's addresses: Radu Mariescu-Istodor and Pasi Fränti, Machine Learning Group, School of Computing, University of Eastern Finland, Joensuu, Finland.

DOI:http://dx.doi.org/10.1145/0000000.0000000

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

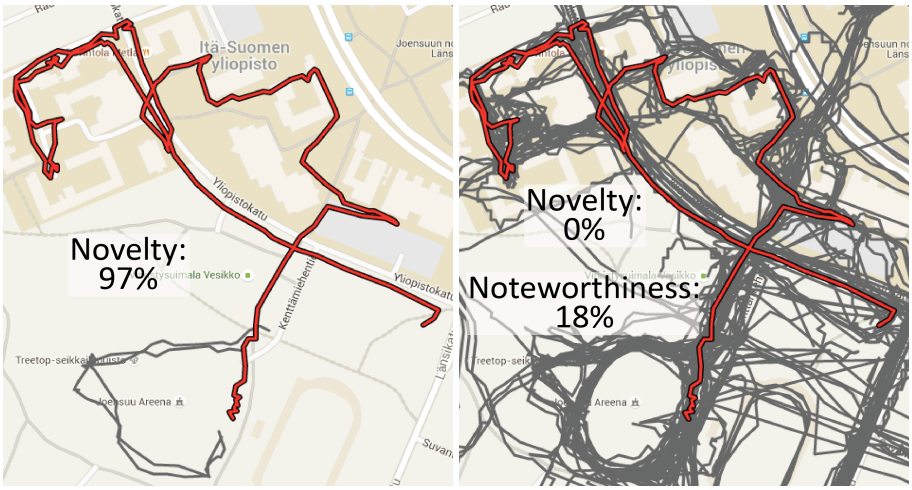


Fig. 1. Route novelty relative to the user himself (left) and relative to all users (right). The route is highly novel to the user, but not for to the rest of the database. Route noteworthiness is useful in the denser area.

Novelty measures the amount of unique parts of a route compared to other routes in the database. Novelty can be useful in several applications. In [Zhang et al. 2011], it is needed for identifying taxi fraud; when the taxi driver takes a longer route than necessary to arrive to the destination. The given route is compared to other past routes starting and ending at the same locations. If it has high novelty, the route is labeled as fraud. Another application for novelty is to automatically update GPS navigation systems existing in many cars nowadays. If a recent route has novelty compared to the exiting road network, it indicates that the roads in the region have changed and database-updating methods such as [Fathi and Krumm 2010, Cao and Krumm 2009] should be executed. In Mopsi, novelty is used to inform users when their route passes though places they have never visited before. We also verify if other users have frequented the area (see Figure 1). Route *noteworthiness* is closely related to novelty. It measures the amount of rarely visited parts instead of only focusing on the parts that are unique. This measure is useful in places with large density of routes so that novel regions rarely exist.

We define that two routes are *similar* if they overlap. The amount of overlap measures how similar the routes are. Many applications for route similarity exist. One example is to use route similarity as a component when measuring the similarity between users in a social network. In [Ying et al. 2010] it was suggested that meaningful friend recommendations could be issued in this way. Another case where route similarity is helpful is when giving trip recommendations. In [Shang et al. 2012] a route is recommended given a set of intended places and a set of textual attributes that describe the user's preferences. The similarity measure can also be used to identify ideal places where to build new bicycle paths. In [Evans et al. 2013] this is achieved by computing the similarity between all routes in a database using the network Hausdorff distance. Optimizing traffic is another task aided by route similarity. In [Ying et al. 2009] the similarity measure serves as a distance function for density-based clustering of segments in order to identify the congestion areas.

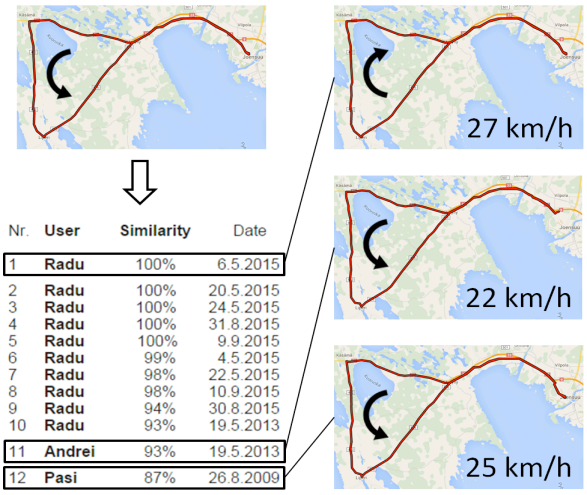


Fig. 2. Many similar routes for Joensuu – Liperi – Joensuu cycling track. We can see that several users have tried it with different outcome in terms of the average speed.

There are many methods for computing route similarity. General time series analysis methods have been used [Hamilton 1994, Agrawal et al. 1993]. Techniques based on longest common subsequence [Vlachos et al. 2002] and edit distance on real sequence [Chen et al. 2005] are more recent directions specifically aimed for analyzing routes. The advantages and disadvantages for each of these methods have been summarized in [Wang et al. 2013] where some methods are shown to be sensitive to noise while others to variations in sampling rate, the conclusion being that none of them is flawless but all can be useful, depending on the application. However, all these measures are implemented by dynamic programming with time complexity of $O(N_1N_2)$, where N_1 and N_2 are the number of points in the two routes. We will present a fast and simple approach by first representing the routes as cells of a 2D grid and then applying set operations. The proposed method is upper limited by $O(K_1+K_2)$ where K_1 and K_2 are the physical lengths of the two routes (in meters). The actual cost is smaller, depending on the cell size.

Many applications need to find for a given route the most similar one from the database. A naïve approach computes similarity with every other route. This results in $O(MN^2)$ complexity, where M is the number of routes in the database and N is the average number of points in a route. Many similarity computations can be omitted by calculating bounding box of the route. [Wang and Liu 2012] use polygonal approximation to first obtain a quick estimate of the similarity and then calculate exact measure for the top candidates only. Even then, the time complexity is far from real-time in areas with high route density. For this task, we present **R**oute **S**imilarity **R**anking (RSR) algorithm, which finds all similar routes in the database for a given input route. It is implemented in Mopsi where it is used as sport tracker such as jogging, cycling or cross-country skiing. The algorithm works real-time on a dataset of 6,700 routes. Users can easily compare stats and analyze their progress over time. The ranking shows also other users that have completed the same or a similar route (see Figure 2). Routes with lower similarity can also provide valuable insights. For instance, Figure 3 shows two 70% similar routes being compared in Mopsi and the knowledge gained from the comparison.

Finally, we define *inclusion* as the amount of one route contained inside the other. Unlike similarity, the inclusion is not symmetric. The measure is useful for solving the drive sharing problem by identifying users that:

A. can pick up somebody on his / her way,
B. can be picked up by somebody else on their way.

To speed-up the proposed approach, we consider indexing. R-tree [Guttman 1948] has been used to improve efficiency of spatial queries in several route-searching problems [Yanagisawa et al. 2003, Frentzos et al. 2007 and Güting et al. 2010]. GPS points are recorded with varying accuracy, which depends on several factors such as the device quality and the weather. Typically, when comparing GPS points a distance threshold must be set so that points closer than the threshold are considered identical. With R-tree it is possible to use range queries in order to obtain the points closer than the specified threshold. However, when using the grid, the cell size acts as a distance threshold and points that are mapped to the same cell are considered identical. Because of this, range queries are not necessary; we do, however, investigate B-tree and Hash [Cormen 2009] indexing methods to facilitate searching the cells.

The proposed measure uses only the spatial aspect of routes; this means that the order of traveling is ignored. For example, two cycling routes in the opposite directions would lead to 100% similarity. In applications where the order matters, a similar strategy as in [Chen et al. 2011] can be used by clustering the routes using the Hausdorff distance (a measure which ignores point order) and then adding the direction as a separate feature.

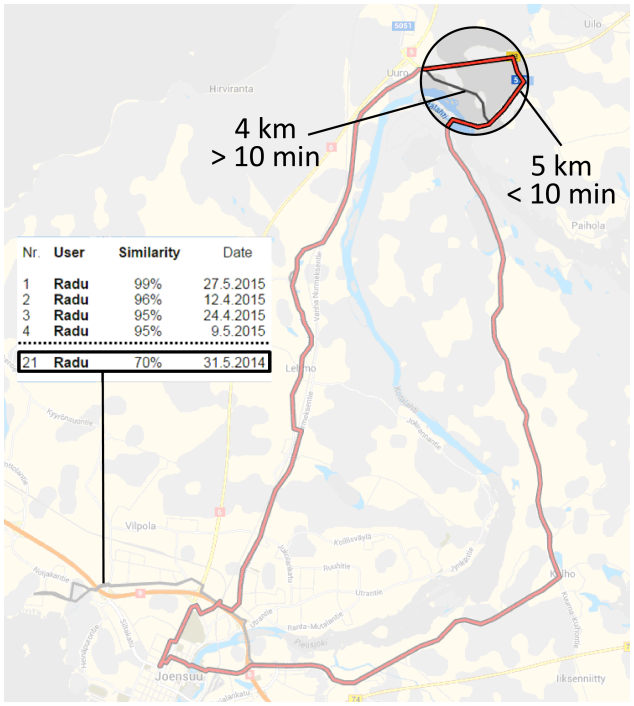


Fig. 3. Two 70% similar routes compared in Mopsi. One route takes a 1 kilometer shorter, off-road path. The other route is quicker, despite the extra kilometer.

2. USING THE GRID

Grids have been used for representing geographical data in the past. In [Pang et al. 2012, Zhang et al. 2011] grids are used to find patterns in taxi data. In [Wei et al. 2012], popular routes are constructed using the frequency information of grid cells. In [Zheng et al. 2010, Bao et al. 2012] the grid is used to infer stay areas, which are used to detect points of interest. In [Krumm and Horvitz 2006] grids are shown to be useful when attempting to predict the destination of moving vehicles.

The abovementioned examples use the grids to perform frequency analysis in sub regions of a given area. We extend the use of the grid to define a similarity measure between routes and to perform similarity-based retrieval in route databases. When computing similarity of routes, the grid needs to be finer than in other applications, which typically use cell sizes in the scale of 100 m – 1 km.

Constructing a grid with equal cell size on the planet surface is not trivial. In [Bao et al. 2012], the earth surface is partitioned in the scale of 0,001 latitude × 0,001 longitude which equals roughly 111 meters, however, the cells become smaller as they get further away from the equator. For our purpose we need a grid with equal size cells everywhere on planet. Otherwise, comparing two routes will give a different result depending on the latitude. Grids with equal cell sizes have been generated in [Zhang et al. 2011, Krumm and Horvitz 2006, Pang et al. 2012, Wei et al. 2012] but they all use the grid in a small region, typically in a single city. However, in our case the routes can be recorded anywhere on the surface of the Earth, land or water; the grid must therefore exist everywhere.

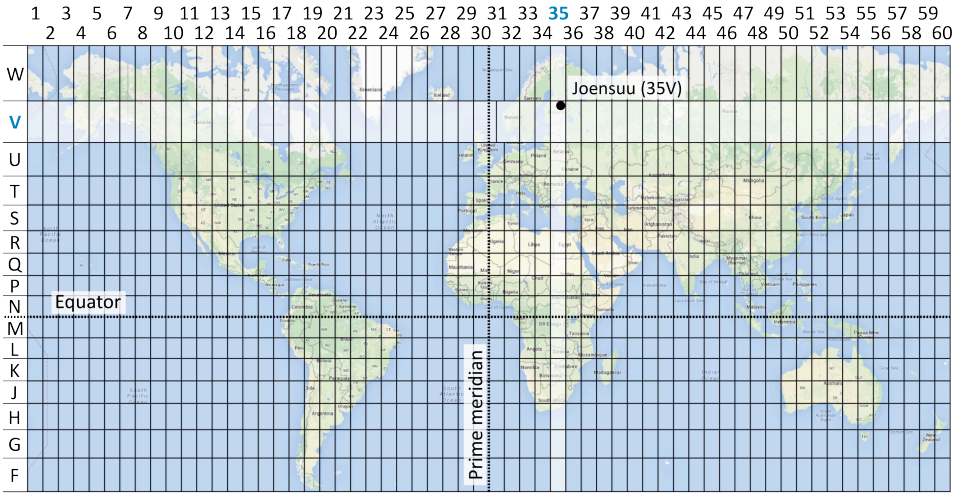


Fig. 4. UTM zones and latitude bands. Jonesuu is in MGRS grid zone 35 V.

Military **G**rid **R**eference **S**ystem (MGRS) has the required features: the cell size is constant and it is defined over the entire planet. It is a standard used by NATO to locate points on the earth. Open-source solutions[3] exist for different programming

[3] http://builds.worldwind.arc.nasa.gov/worldwind-releases/1.4/docs/api/overview-summary.html

languages. Time complexity for the conversion from WGS to MGRS and back is constant.

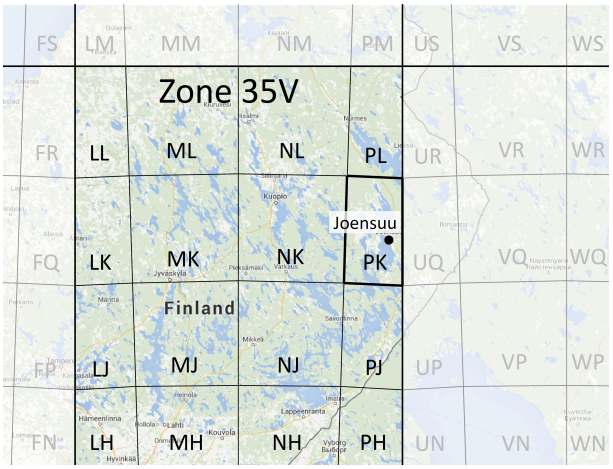


Fig. 5. 100 kilometer squares in zone 35 V. Joensuu is in square PK.

MGRS is an alpha-numeric two-dimensional coordinate system in which locations are identified independent of vertical position. Similarly to **U**niversal **T**ransverse **M**ercator (UTM), MGRS uses division of earth into projection zones and computes easting and northing in meters within a designated zone. UTM divides the planet into 60 zones, each being 6° of longitude in width. For the Polar Regions (above 84°N and below 80°S) the **U**niversal **P**olar **S**tereographic (UPS) convention is used instead of UTM. For the perpendicular segmentation, bands of latitude (8° high) are used. The first three characters of the MGRS value for the city of Joensuu, Finland are 35V (see Figure 4). The next pair of characters identifies a 100 × 100 kilometers square within each of the grid zones. Around the edge of a grid zone these squares are truncated in order to fit. This tedious process makes it possible to wrap the grid around the planet. Joensuu is located in region 35VPK (see Figure 5).

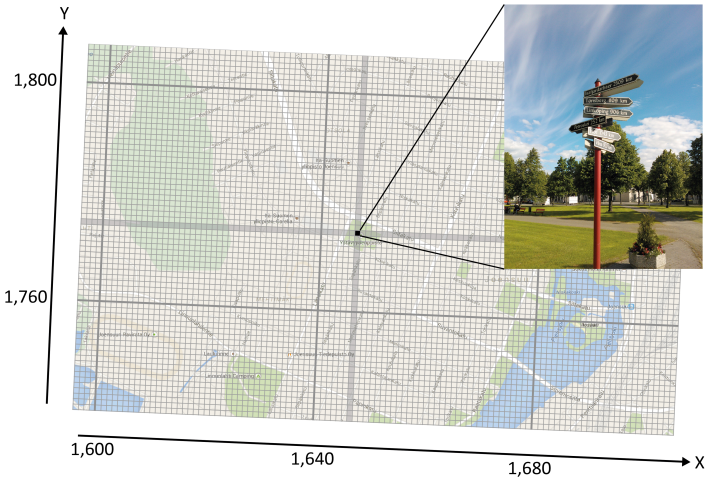


Fig. 6. 25 × 25 meter cells in Joensuu. The highlighted cell is in the center of a small park.

The remaining part consists of the numeric Easting and Northing values within the 100-kilometer square. MGRS allows one of 5 predefined precision levels when choosing the cell length: 1 m, 10 m, 100 m, 1 km, 10 km. However any precision can be easily obtained if the desired cell length perfectly divides 100,000 (meters). Based on our experiments and typical GPS device accuracy we chose a cell size of 25 × 25 meters. Larger values result in poorer approximation while smaller values are vulnerable to GPS error. In Figure 6, we identify a point as 35VPK16461774.

3. CELL REPRESENTATION

We define a point on the earth as **p**=(x,y) where x is the latitude and y is the longitude. An ordered sequence of these points defines a route $\mathbf{R}=(p_1,\ldots,p_N)$. The route can be approximated by set C, created by mapping each point to the MGRS grid. Algorithm Points-to-Cells shows how this representation can be calculated in O(N) time. The WGS-to-MGRS conversion works in constant time.

We use a hashing method to keep track of cells already existing in the representation. With our 25 × 25 meter sized cells, a 100 × 100 kilometer square results in a grid of size 4,000 × 4,000 cells. A route consists of pairs of Easting and Northing values that passes through the cells. The same route may pass through two cells with the same (x, y) but in a different square; for example MGRS coordinates 35VPK-1122-1122 and 35VPL-1122-1122. We store every Square ID in a linked list in the array at the (x, y) position. It is very unlikely that different cells of the same route have the same (x, y) coordinates. It would require the route to be at least 100 km long and to reach the exact same Easting and Northing values in an adjacent MGRS cell. Thus, the linked lists usually contain a single element.

Points-to-Cells: Finding the set of cells that approximate a given route.

Input: Route R containing N points, cell size L.
Output: Set C.

```
C ← empty list
H ← 4000 x 4000 array of empty lists
for i ← 1 to N do
    cell ← WGS-to-MGRS ( R [ i ], L )
    if H [ cell → x ] [ cell → y ] ]. contains ( cell → Square ID ) then
        // nothing to do, cell already exists in C
    else
        C . add ( cell )
        H [ cell → x ] [ cell → y ] ]. add ( cell → Square ID )
    end
end
```

Gaps can appear in the cell representation when a mobile user is traveling faster than the cell length divided by sampling interval, or when user moves and the device fails to update the location or for some other reason (see Figure 7). We generate the cells in the order that the route points were recorded; if two consecutively generated cells are not adjacent we fill the gap by using linear interpolation with equation:

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1, \tag{2}$$

where (x_1,y_1) and (x_2,y_2) are the easting and northing values of the two cells inside a 100 km square. To fill the gap, the line is sampled along the longer edge of the rectangular region whose opposite corners are the two cells (see Figure 8). This process requires $O(\max(|x_2-x_1|,|y_2-y_1|))$ time.

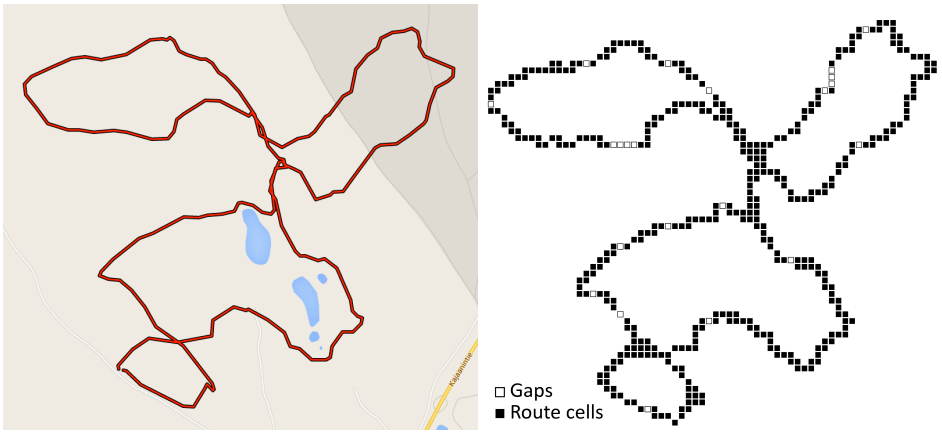


Fig. 7. A sample route (left) and the cell representation with cell size 25 × 25 meters (right).

It is possible that two consecutive cells do not lie inside the same 100 km square. Only in these rare situations we cannot perform the interpolation in MGRS space because the easting and northing values refer to different subspaces. Instead, we interpolate using the latitude and longitude values, which we gradually increment and compute the MGRS mapping along the way. Equation 3 gives an estimate for the number of cells a route contains after interpolation:

$$cellCount(R,L) = \frac{length(R)}{L}. \tag{3}$$

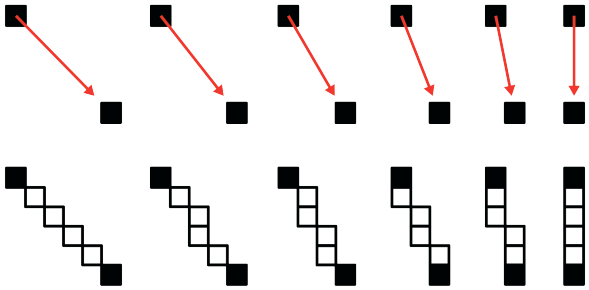


Fig. 8. Different examples where a four cell gap is filled through interpolation.

It is theoretically possible to double the amount of cells when moving along the cell border and slightly oscillating from one side to another. Fewer cells are possible when the route includes loops or overlaps itself. Theoretically, an infinitely long route can exist in a single cell just by moving around in circles within the cell. This kind of behavior is sometimes noticed when user is standing still but GPS signal fluctuates. In Figure 9 we show examples of routes with the same length but different number of

cells. In practice, when computing the cell representation for routes in Mopsi2014 dataset, a route passes through 37 cells per kilometer, on average (see Figure 12).

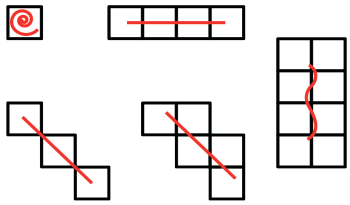


Fig. 9. Several 100 meter routes requiring between 1 and 8 cells of size 25 × 25 meters.

Often, points close to each other end up in different cells due to the arbitrary division of the grid. This can produce errors when comparing routes. In principle, two people walking hand in hand may never share a cell. We fix this problem with morphological dilation with square structural element (see Figure 10). The extra cells form a separate set C^d, which is treated as a buffer region when comparing two routes.

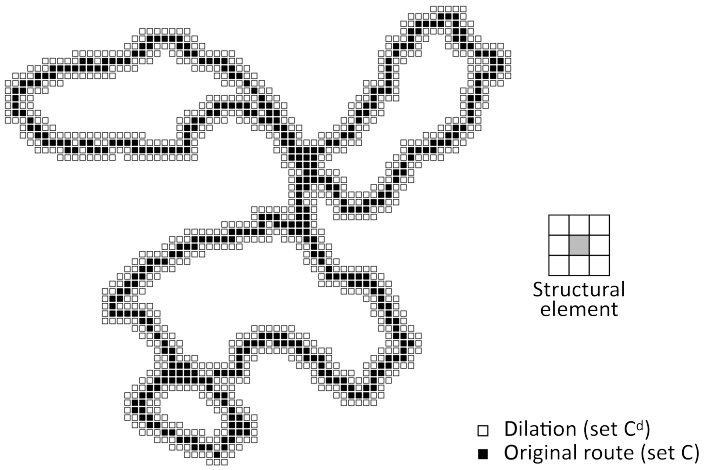


Fig. 10. Dilating the cell representation of a route with a square structural element.

When dilating a cell, the number of extra cells to be added depends on the direction of travel, see Figure 11. Moving diagonally adds 5 new cells while moving horizontally or vertically adds only 3. The direction is with respect to the orientation of the MGRS 100-kilometer square in the area (see Figure 5), and not to the cardinal direction. Mopsi2014 routes require 135 cells per kilometer, on average, when the dilated part is included (see Figure 12).

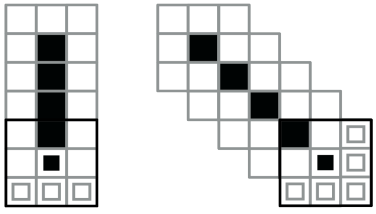


Fig. 11. Number of extra cells added from dilation depends on the direction of travel.

We use a square structural element because it guarantees that points with relative distance less or equal to L will be considered identical (a different structural element such as cross may not guarantee this property). The space required to store the complete cell representation, including the cells from interpolation and dilation, is proportional to the length of the route K. The time required is O(max(N, K / L)) where N is the number of points.

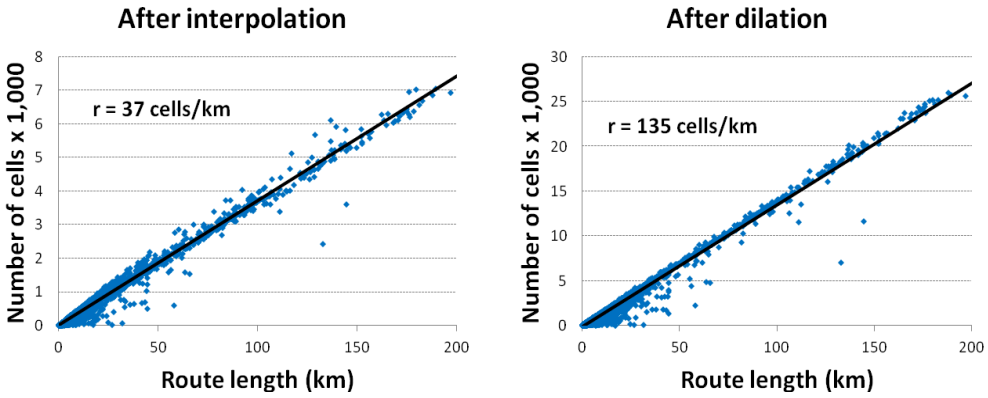


Fig. 12. Relationship between the length of the route and the number of cells when L=25 m.

4. CELL DATABASE

We compute the cell representation for the entire route database. In a dynamic system, the cell representation generation must be triggered when a new route is recorded. We create a *cell database*, in which we store entries ($Route_{id}$, $Cell_{id}$, Dilation); the first field is the route identifier, the second is a unique identifier for the cell (the MGRS cell id) and the last field is a Boolean value specifying if the cell was obtained from dilation or not. Some example entries are shown below:

Route	Cell (MGRS)	Dilation
3812	35VPK16491768	FALSE
3812	35VPK16481768	FALSE
3812	35VPK16471768	FALSE
3812	35VPK16471768	FALSE
3812	35VPK16461769	FALSE
3812	35VPK16451771	FALSE
3812	35VPK16441772	FALSE
3812	35VPK16441773	TRUE
3812	35VPK16441771	TRUE
6115	35VPK44122117	FALSE
6115	35VPK44122118	TRUE
6115	35VPK44022118	FALSE

We implement four elementary database operations, which will be used later to design our methods. We can apply B-tree but also hash index because all four operations rely only on strict equality checks. The time complexities of these four operations are summarized in Table 1. Parameter M is the number of routes in the database and Q is the average number of cells in a route; MQ equals to the number of

entries in the database. In Mopsi2014, M = 6,779 and Q = 1,693. The search using B-tree and hash has time complexities of O(log(MQ)) and O(1), respectively.

The time complexity for the Get-Routes operation depends on the number of routes that pass through the given cell. This number equals to the frequency of the given cell c in the database. We refer to this number as f(c). In places with a lot of routes, this number increases. For example, the frequency in the center of Joensuu is 400 whereas the average frequency in the entire Mopsi2014 dataset is 2.

Table I. Elementary database operations and their time complexities

Operation	No index	B-tree	Hash
Get-Cells	O(MQ)	O(log(MQ) + |C|)	O(|C|)
Get-Routes	O(MQ)	O(log(MQ) + f (c))	O(f (c))
Is-Novel	O(MQ)	O(log(MQ))	O(1)
Is-In-Subset	O(MQ|S|)	O(log(MQ)|S|)	O(|S|)

Get-Cells: Obtain the precomputed cells representing a given route.

Input: route r_{id}
Output: sets C and C^d

C ← empty set
C^d ← empty set

rows ← DB : SELECT ($Cell_{id}$, Dilation) WHERE $Route_{id} = r_{id}$

for i ← 1 **to** size (rows) **do**
    if rows [i] . Dilation = TRUE **then**
        C^d . add (rows [i] . $Cell_{id}$)
    else
        C . add (rows[i]. $Cell_{id}$)
    end
end

Get-Routes: Obtain the routes that pass through a given cell. R^d will contain the routes whose dilated region intersects the given cell.

Input: cell c_{id}
Output: arrays R and R^d

R ← new array
R^d ← new array

rows ← DB : SELECT ($Route_{id}$, $Cell_{id}$, Dilation) WHERE $Cell_{id} = c_{id}$

for i ← 1 to size (rows) **do**
    if rows [i] . Dilation = TRUE **then**
        R^d . add (rows [i] . $Route_{id}$)
    else
        R . add (rows [i] . $Route_{id}$)
    end
end

Is-Novel: Check if a given cell is novel in the database. A cell is novel if a single route passes through it. We stop the search immediately if a second route is found.

Input: cell c_{id}
Output: Boolean novel
novel ← FALSE
rows ← DB : SELECT ($Route_{id}$, $Cell_{id}$) WHERE $Cell_{id} = c_{id}$ LIMIT 2
if size (rows) = 1 **then**
    novel ← TRUE
end

Is-In-Subset: Check if a given cell is part of any route in a specified subset of the database.

Input: cell c_{id} and route subset S
Output: Boolean exists
exists ← FALSE
rows ← DB : SELECT ($Route_{id}$, $Cell_{id}$) WHERE $Route_{id}$ IN S AND $Cell_{id} = c_{id}$ LIMIT 1
if size (rows) = 1 **then**
    exists ← TRUE
end

5. MEASURES

In this section, we present four route measures: novelty, noteworthiness, similarity and inclusion. We give at least one algorithm for computing each measure and the time complexity is computed for every one of them.

5.1 Novelty

We define *novelty* as the proportion of a given route that does not overlap with any other route. Using the cell representation, it is computed by counting the amount of *novel* cells in the route. A cell is novel if it is not part of any other route from the database. We calculate the novelty by NOV algorithm, which performs Is-Novel check for every cell of the route. The novelty is then defined as the number of novel cells relative to the total number of cells.

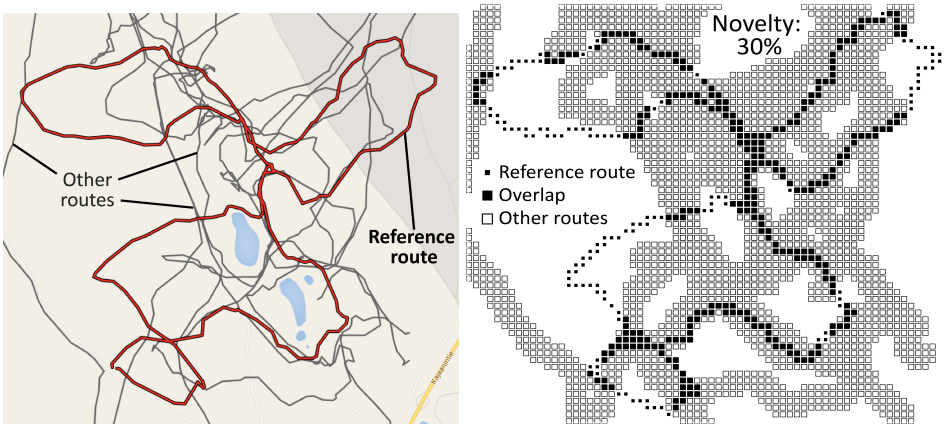


Fig. 13. The novel and overlapping cells of a route when compared against a route database. The dilated region is considered.

Figure 13 shows a route with 30% novelty. The Is-Novel function uses the dilated representation of the routes in the database; therefore, we do not need to dilate the reference route itself.

NOV: Compute the novelty of a route with respect to the entire database.

Input: route r_{id}
Output: novelty
C, C^d ← FindCells (r_{id})
novelCells ← 0
for i ← 1 **to** size (C) **do**
    if Is-Novel (C [i]) **then**
        novelCells ++
    end
end
novelty ← novelCells / size(C)

It may be needed to compute novelty with respect to a given subset S of the route database. This subset depends on the application. It can be routes that belong to a certain user, routes recorded in a given time period, routes starting and ending in specified locations, or routes with a certain movement type. S-NOV and NOV-S are two different ways for computing the *subset novelty*. S-NOV uses the Is-In-Subset operation. It is the same as NOV but limits the search to the subset S by using the SQL IN clause. The second one, NOV-S, gets all routes from database that pass through every cell of the input route by the while loop. Any cell that does not include any other routes, is labeled as novel. Other cells we refer as *active* cells. The number of active cells is counted as:

$$a(C) = \sum_{i=1}^{M} \left(\left| C \cap C_i \right| + \left| C \cap C_i^{\,d} \right| \right) \tag{4}$$

S-NOV: Compute the novelty of a route with respect to a subset of the database.

Input: route r_{id} and route subset S
Output: novelty
C, C^d ← FindCells (r_{id})
novelCells ← size (C)
for i ← 1 **to** size (C) **do**
    if Is-In-Subset (C [i], S) **then**
        novelCells --
    end
end
novelty ← novelCells / size(C)

NOV-S: Alternative way to compute the novelty of a route in a subset. Here S is an array of M elements with 1's indicating which routes are to be considered.

Input: route r_{id} and route subset S
Output: novelty
C, C^d ← Get-Cells (r_{id})
novelCells ← size (C)
for i ← 1 **to** size (C) **do**
    R_i, R^d_i ← Get-Routes (C [i])
    isNovel ← FALSE
    while isNovel = FALSE **and** items exist in R_i and R^d_i **do**
        r ← next item in R_i or R^d_i
        if S [r] = 1 **and** r ≠ r_{id} **then**
            isNovel ← TRUE
        end
    end
    if isNovel = TRUE **then**
        novelCells --
    end
end
novelty ← novelCells / size(C)

Table 2 contains the time complexities for the three novelty algorithms in case of B-tree, hash and without indexing. Algorithms S-NOV and NOV-S can both be useful, depending on the application. Using one or the other depends on the number of active cells in the region and the size of the route subset. S-NOV is more efficient in areas with many routes and smaller subsets; NOV-S is recommended otherwise. This is examined more in Section 6.3.

Table II. Time complexity for algorithms that compute novelty

Method	No index	B-tree	Hash
NOV	O(|C|MQ)	O(log(MQ) |C|)	O(|C|)
S-NOV	O(|C||S|MQ)	O(log(MQ) |C||S|)	O(|C||S|)
NOV-S	O(|C|MQ)	O(|C|(log(MQ)) + a(C))	O(|C|+ a(C))

5.2 Noteworthiness

Concluding that a cell is not novel because another route passes through it can be a too strict criterion. Figure 14 shows the tracking activity in a region. Some cells are *noteworthy* because they are traveled much less frequently than others. We define *noteworthiness* as the proportion of the cells that has little to no activity. We compute the noteworthiness using algorithm NTW. It first computes a histogram, which counts the frequency for each cell. The histogram is then normalized to the range [0, 1]. The cells with activity less or equal to a parameter p are counted as noteworthy. The noteworthiness of the route is then calculated as the ratio of noteworthy cells to all the cells in a route. The algorithm has the same time complexity as NOV-S. A parameter-free alternative can be constructed by calculating the average activity of the cells within the route and defining the cells with activity bellow this value as noteworthy.

NTW: Computing route noteworthiness.

Input: route r_{id}, threshold p
Output: noteworthiness
C, C^d ← FindCells (r_{id})
hist ← new array

//counting frequencies
for i ← 1 **to** size (C) **do**
    R_i, R^d_i ← Get-Routes (r_{id}, C [i])
    hist [i] ← size (R_i) + size (R^d_i)
end
normalize (hist)

// computing novelty
noteworthyCells ← 0
for i ← 1 **to** size (hist) **do**
    if hist [i] ≤ p **then**
        noteworthyCells ++
    end
end
noteworthiness ← size (noteworthyCells) / size (C)

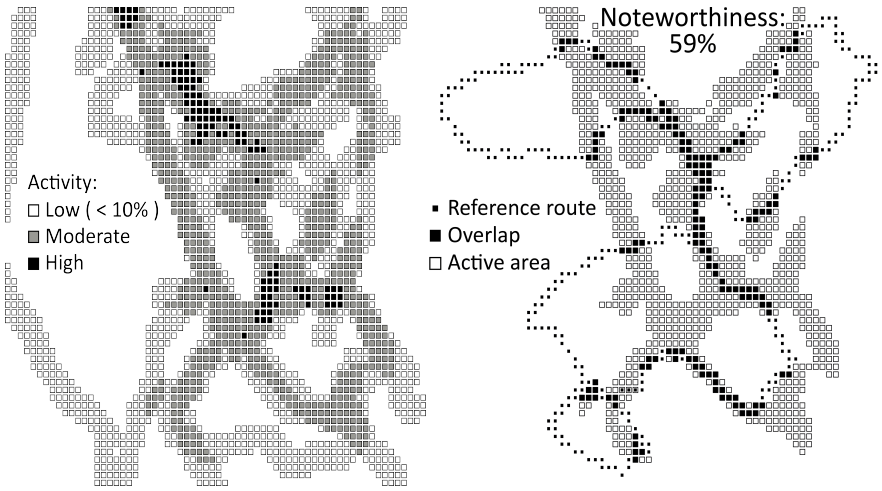


Fig. 14. The tracking activity in a region (left) and computing the noteworthiness of the route with respect to the active region (right). Parameter p = 10% is used in this example.

5.3 Similarity

We define that two routes are *similar* if they overlap. We use Jaccard index to measure the amount of similarity. It is calculated as the size of the intersection divided by the size of the union of two sets:

$$J(C_A, C_B) = \frac{|C_A \cap C_B|}{|C_A \cup C_B|}. \quad (5)$$

We consider the dilation in order to guarantee that points less than L meters away from each other are treated as identical. We do not dilate both routes simultaneously because it would double the distance threshold. Instead, we dilate each of the two routes separately, compute the two intersections with the original of the other route, and unite the results as follows:

$$S(C_A, C_B) = \frac{\left|(C_A \cap C_B) \cup \left(C_A \cap C_B{}^d\right) \cup \left(C_B \cap C_A{}^d\right)\right|}{\left|C_A \cup C_B\right|}. \quad (6)$$

Because $C_A \cap C_A{}^d = \{\}$ and $C_B \cap C_B{}^d = \{\}$ by definition, we union of the three sets in the numerator is equivalent to the sum of their individual sizes. The denominator can be computed simply by the sum of the elementary set sizes subtracted by the size of their intersection. Equation 7 shows an alternative formula after these observations. Figure 15 illustrates the necessary components for computing the similarity..

$$S(C_A, C_B) = \frac{\left|C_A \cap C_B\right| + \left|C_A \cap C_B{}^d\right| + \left|C_B \cap C_A{}^d\right|}{\left|C_A\right| + \left|C_B\right| - \left|C_A \cap C_B\right|}. \quad (7)$$

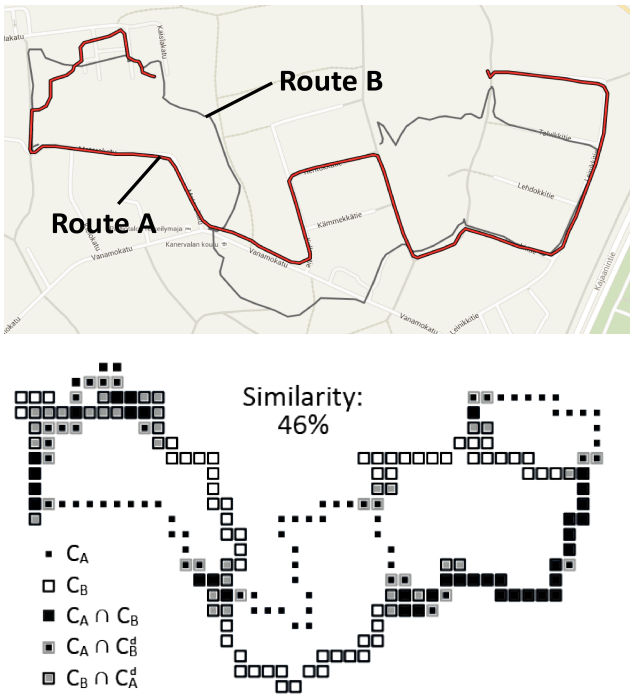


Fig. 15. Two routes that are 46% similar. Elementary sets for computing the similarity are depicted.

C-SIM algorithm computes the similarity between two given routes. It first retrieves the cell representation and then calculates the similarity measure using the cells.

Intersection algorithm computes the intersection between two sets efficiently in linear time using the hashing technique described earlier. The total time complexity of C-SIM is $O(N_A + N_B + |C_A| + |C_B|)$ where N_A and N_B are the number of points in the two routes.

C-SIM: Computing similarity between two routes.

Input: routes R_A and R_B
Output: similarity
C_A, C_A^d ← Points-to-Cells (R_A)
C_B, C_B^d ← Points-to-Cells (R_B)
cab ← Intersection (C_A, C_B)
cab^d ← Intersection (C_A, C_B^d)
cba^d ← Intersection (C_B, C_A^d)
similarity ← $(cab + cab^d + cba^d) / (|C_A| + |C_B| - cab)$

Intersection: Efficiently computing the intersection between two sets of cells.

Input: sets C_A and C_B
Output: intersection set X
H ← 4000 x 4000 array of empty lists
X ← new set
for each a **in** C_A **do**
    H [a → x] [a → y]. add (a → Square ID)
end
for each b **in** C_B **do**
    if H [b → x] [b→ y]. contains (b → Square ID]) **then** // O(1) time expected
        X . add (b)
    end
end

5.4 Inclusion

The *inclusion* measures what proportion of a given route is contained in another. Using the grid, we compute the inclusion between two routes, C_A and C_B, as:

$$I(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B{}^d|}{|C_A|}. \quad (8)$$

The inclusion is not symmetric and rarely gives the same result when switching the arguments (see Figure 16). We dilate the second route and normalize the result with respect to the original route. Algorithm INC has the same time complexity as C-SIM. Preliminary version of the inclusion measure has been presented in [Mariescu-Istodor et al. 2014].

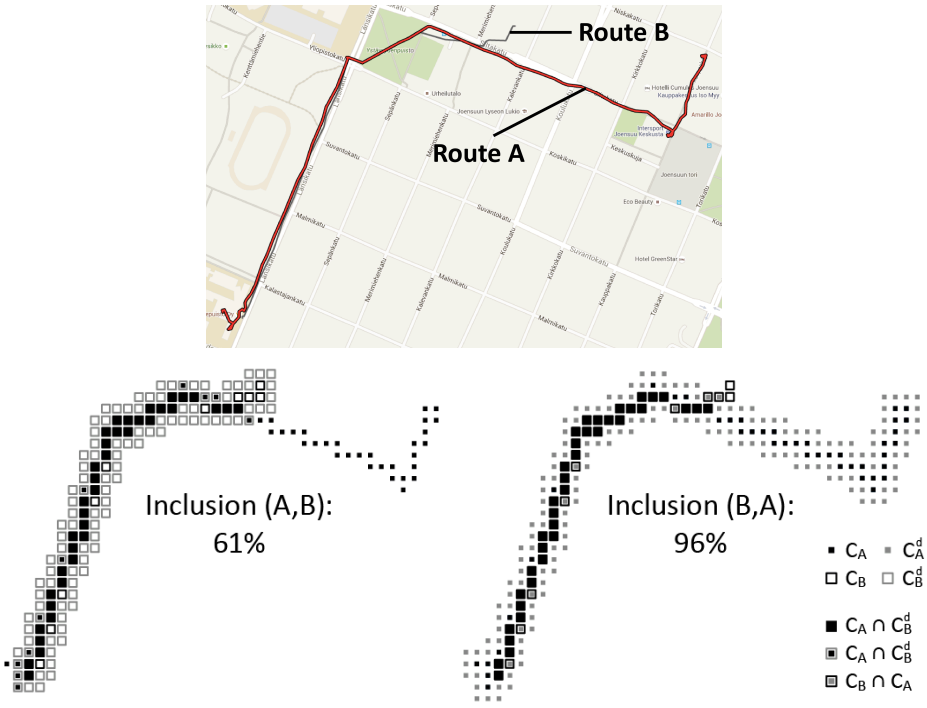


Fig. 16. Two routes A and B so that 61% of route A is included in B and 96% of route B is included in A.

INC: Computing inclusion of route R_A in R_B.

Input: routes R_A and R_B
Output: inclusion

C_A, C_A^d ← Points-to-Cells (R_A)
C_B, C_B^d ← Points-to-Cells (R_B)
cab ← Intersection (C_A, C_B)
cab^d ← Intersection (C_A, C_B^d)
inclusion ← cab + cab^d / $|C_A|$

5.5 Route Similarity Ranking

Route **S**imilarity **R**anking (RSR) is an algorithm that finds, for a given route, all similar routes in a database and ranks them in decreasing order of the similarity. RSR begins by computing the cell representation for the given route. It then iterates through every cell and finds what other routes are passing through. For each found route C_B it marks whether the cell belongs to $C_A \cap C_B$, $C_A \cap C_B^d$ or $C_A^d \cap C_B$. These numbers are later used for computing the similarity values according to Equation 7.

The time complexity of the RSR algorithm is $O((|C|+|C^d|)(\log(MQ))+a(C)+a(C^d))$ when B-tree index is used. If hash index is used, the time complexity is $O(|C|+|C^d|+a(C)+a(C^d))$. When no indexing is used, the time complexity is $O((|C|+|Cd|)MQ)$. It is difficult to predict how long the process will run for a given route exactly as it depends on the area where the route is: in a highly traveled area it

takes longer than in a less traveled area. A definite upper bound is when the database contains only identical routes; in this case the complexities become O(MQlog(MQ)) and O(MQ) for B-tree and hash respectively. RSR algorithm can be modified to compute the inclusion values for every route in the ranking since the necessary components: $|C_A \cap C_B|$ and $|C_A \cap C_B{}^d|$ already exist.

RSR: Computing the route similarity ranking.

Input: route r_{id}
Output: similarityList

```
C, C^d ← Get-Cells ( r_id )
SC ← initialize SetCounter array; // structure defined below
// process input route
for i ← 1 to size ( C ) do
        R_i, R^d_i ← Get-Routes ( C [ i ] )
        for j ← 1 to size ( R_i ) do
                SC [ R_i [ j ] ]. A ++;   SC [ R_i [ j ] ]. B ++;   SC [ R_i [ j ] ]. AB ++;
        end
        for j ← 1 to size ( R^d_i ) do
                SC [ R^d_i [ j ] ]. A ++;   SC [ R^d_i [ j ] ]. B ++;   SC [ R^d_i [ j ] ]. AB^d ++;
        end
end
// process dilated part
for i ← 1 to size (C^d ) do
        R_i, R^d_i ← Get-Routes ( C^d [ i ] )
        for j ← 1 to size ( R_i ) do
                SC [ R_i [ j ] ]. B ++;   SC [ R_i [ j ] ]. A^dB ++;
        end
        for j ← 1 to size ( R^d_i ) do
                SC [ R^d_i [ j ] ]. B ++;   SC [ R^d_i [ j ] ]. A^dB^d ++;
        end
end
similarityList ← new list;
for each rid in SC do
        S ← ( SC [ rid ] . AB + SC [ rid ] . A^dB + SC [ rid ] . AB^d ) /
                (SC [ rid ] . A + SC [ rid ] . B - SC [ rid ] . AB)
        similarityList.append ( rid, S )
end

SetCounter {
    A←0;   B←0;   AB←0;   A^dB←0;   AB^d←0;
}
```

6. EXPERIMENTS

We tested our methods with Mopsi2014[4] dataset, which is a subset of all routes in Mopsi database collected by the end of 2014. It contains 6,779 routes recorded by 51 users who each have 10 or more routes. Routes consist of wide range of activities including walking, cycling, hiking, jogging, orienteering, skiing, driving, traveling by bus, train or boat. Routes exist on every continent except Antarctic. This provides a good evaluation for MGRS, which works well in all regions where test data was available. Most routes are in Joensuu region, Finland, which creates a very dense area suitable for stressing the evaluated methods. Table 3 summarizes the Mopsi2014 dataset.

Table III. Mopsi2014 dataset summary

Routes	Points	Kilometers	Hours
6,779	7,850,387	87,851	4,504

We first computed the 25 × 25 meter cell representation for all 6,779 routes. The cell database entries include cells obtained from interpolation and dilation. Statistics are shown in Table 4. Typically, point databases are indexed with R-tree to make range queries possible. If R-tree is applied, Mopsi2014 would require approximately 1 GB of space. The cell database has similar space requirements when B-tree index is used but Hash index uses 80% more space than B-tree. In total, Mopsi2014 with hash index requires 1.3 GB of memory space.

Table IV. Database requirements

	Entries	Index	Total
Point Database	7,850,387 (329 MB)	R-tree (650 MB)	979 MB
Cell Database	11,477,506 (525 MB)	B-tree (429 MB)	954 MB
		Hash (788 MB)	1313 MB

Next we perform a set of experiments using no index, B-tree and hash respectively. All experiments were executed on Dell R920, 4 x E7-4860 (total 48 cores), 1 TB, 4 TB SAS HD. We use MySQL to store the data with B-tree and hash.

6.1 Effect of indexing on NOV algorithm

We evaluate the effect of indexing by computing the novelty of 3000 different routes. The novelty of each route is computed against the entire Mopsi2014 dataset. We focus only on routes with $|C| < 300$ cells (~8 km) because the process without indexing would be very slow.

Results are summarized in Figure 17, where a linear dependency on the number of cells can be observed. The time complexity for NOV algorithm depends on the number of cells and the size of the database. The reason for large variance when no indexing applied is that in order to conclude that a cell is novel we may need to search the entire database (worst case), or stop at the first other occurrence of the given cell and conclude that it is not novel and this first occurrence can appear at any

[4] http://cs.uef.fi/mopsi/routes/dataset

point. The variance increases with respect to the number of cells because the search is repeated for every cell. Based on the results, indexing of the database is essential, although the speed-up is obtained at the cost of increased space requirements.

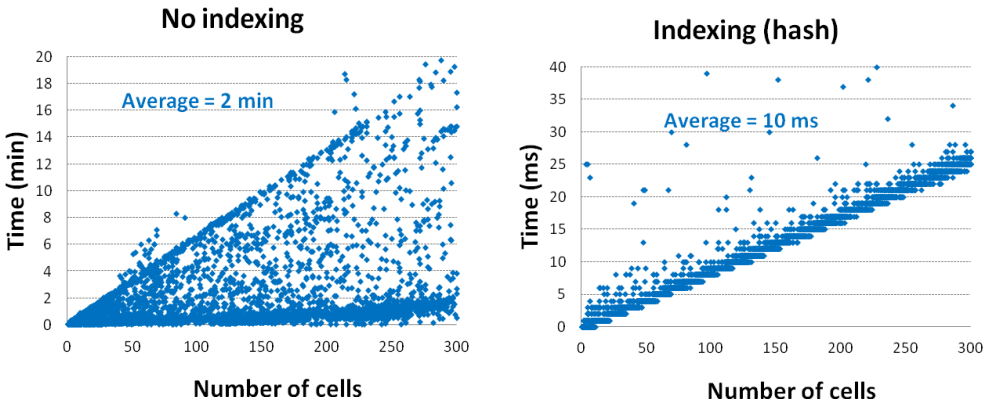


Fig. 17. Time required for calculating the novelty of 3000 routes plotted as a function of the length of the route (in cells).

6.2 Index type comparison on NOV

We compare the efficiency of B-tree and hash indexing when computing the novelty for all routes in Mopsi2014. We omit 22 routes with the highest number of cells in order to obtain statistically significant results. We divide the routes into 11 groups: first one has routes with less than 1000 cells, the second 1000-2000 cells, and so on. The average processing time and standard deviation for each group are shown in Figure 18. We observe that hash index is faster than B-tree. This is as expected from the complexity analysis. The average time for B-tree is 67 milliseconds and for hash is 43 milliseconds. Both methods are expected to work in real-time even for very large routes (300 km). Hash index requires about 50% of the time from that of the B-tree but requires about 40% more memory, see Table 4.

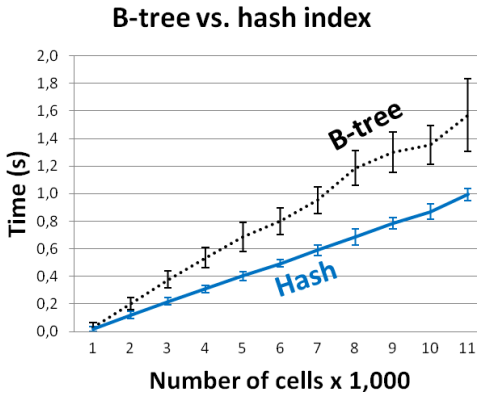


Fig. 18. Comparing B-tree and hash index by showing the average time and standard deviation for routes of different lengths (in cells).

6.3 S-NOV and NOV-S algorithms comparison

We compare the two algorithms for computing the novelty of a route with respect to a given route subset. We randomly select a route subset S of size |S|=15, and compute the novelty with respect to every route in Mopsi2014 (see Figure 19). The coefficient of determination (R^2) indicates that NOV-S is less dependent on the number of cells than S-NOV. This is expected because time complexity of NOV-S depends also on the amount of active cells involved in the computation.

We repeated S-NOV and NOV-S algorithms for 50 random route subsets of sizes 10 and 100. The average processing times are shown in Figure 20. Because S-NOV linearly depends on the size of the subset, it does not scale well for large subsets. NOV-S does not depend on the subset size. In reality, a small speedup is expected when dealing with large subsets because the chance for a cell to be categorized as not novel sooner increases with the subset size, however, the speedup is insignificant.

For small subsets (|S|< 20 routes in case of Mopsi2014) S-NOV is faster than NOV-S. This is because datasets with routes wide spread out will produce less active cells and NOV-S becomes more efficient. In areas with high density of routes S-NOV is expected to be useful for larger subsets.

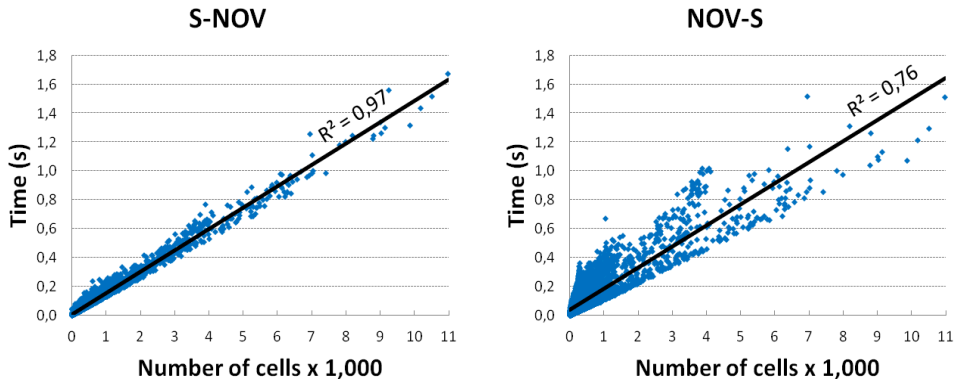


Fig. 19. Comparison of S-NOV and NOV-S running times when novelty is computed for every route in Mopsi2014 against a random subset S with |S| = 15 routes. Hash index is used.

Some applications may require many novelty computations with respect to a small size of subset. For example, by computing novelty of every route of a user with respect to his previous 5-10 routes we can measure how much variation exists in the user's movement. In Mopsi2014, such an application would require roughly half of the time with S-NOV than with NOV-S.

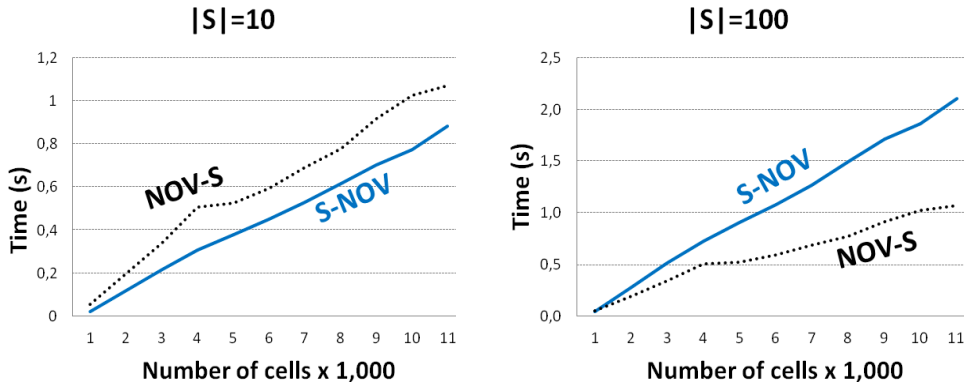


Fig. 20. Average time for both subset novelty methods when applied on subsets of sizes 10 and 100. The results are averaged for 50 random route subsets. Hash index is used.

6.4 Route similarity algorithms scalability comparison

We compare C-SIM algorithm against the following eight route similarity algorithms: **L**ongest **C**ommon **S**ubsequence (LCSS) [Zheng and Zhou 2011], **E**dit **D**istance on **R**eal sequence (EDR) [Chen et al. 2005], **D**ynamic **T**ime **W**arping (DTW) [Zheng and Zhou 2011], FastDTW [Salvador and Chan 2004], **E**dit distance with **R**eal **P**enalty (ERP) [Chen and Ng 2004], Euclidean (L_2-norm) [Gradshteyn and Ryzhik 2000], Hausdorff [Rockafellar and Wets 2009] and Frechet [Eiter and Mannila 1994]. Definitions for all the measures are given in Table 5. We consider all routes with less than 2000 points in Mopsi2014 and divide them into 19 groups. First group of routes have 100-200 points, the second group 200-300, and so on. Then we randomly pair the routes within each group, and compute the similarity for all the pairs. The average times in every group are shown in Figure 21.

LCSS, EDR, DTW, ERP and Frechet route similarity measures are implemented by dynamic programming and they require quadratic time. Hausdorff measure requires to check every point pair between the two routes; thus, its time complexity is also quadratic. Euclidean measure, C-SIM and FastDTW all work in linear time, and are therefore an order of magnitude faster than the others. Euclidean measure is fastest because it only computes a number of distance calculations equal to the size of the smallest of the two routes. It does not perform any kind of alignment of the two routes. FastDTW requires additional work for preparing the multi-resolution representation and processing of every resolution. In this experiment we set the radius parameter to 1. This achieves the poorest approximation, but provides the fastest result. Increasing the radius improves the quality of the solution. If the radius is set to be the size of one of the routes, path will be optimum, but computation time becomes quadratic again. C-SIM is not monotonously increasing because the time complexity is linear with respect to the number of cells of the route, which depends on the distance traveled, and less to the number of points. The average number of points in a route in Mopsi2014 is 1158 points. For routes of this length, C-SIM takes 0.5 s, which is about 10 % of the time taken by the slower method; for instance EDR takes 5.4 s.

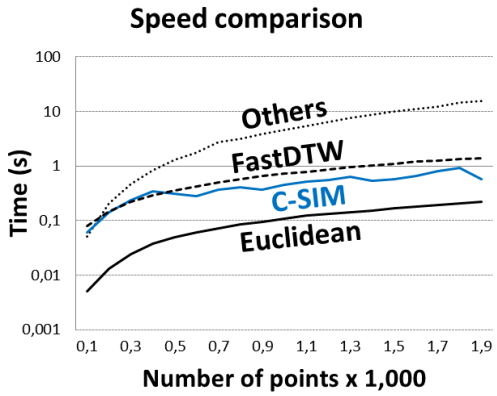


Fig. 21. Comparison of the nine route similarity measures in terms of speed.

6.5 Effectiveness of Route Similarity Measures

We repeat the effectiveness study of [Wang et al. 2013] by the following four measures: Frechet, Hausdorff, FastDTW and C-SIM (proposed). We investigate how the similarity measure is affected by the following transformations:

- increasing sampling rate (adding points)
- decreasing sampling rate (removing points)
- adding noise
- random shifting of points
- synchronized shifting of points.

All transformations use the rate parameter. The last three transformations use also a secondary distance parameter. The authors of [Wang et al. 2013] used 1,000 taxi routes from [Zheng et al. 2009]. We mimic their experiment by randomly selecting 1,000 routes from Mopsi2014, and analyze the behavior of the measures. We assume that these transformations may occur naturally in a route database due to the use of different devices, varying GPS weather and other influences. Therefore the similarity between the transformed route and the original is preferred to remain 100%, alternatively, the distance should be 0 for distance-based measures. We subjectively classify the measures either as *Sensitive*, *Fair* or *Robust*, depending on their ability to cope with these transformations.

In addition to this effectiveness experiment we created an interactive web environment where all nine similarity measures can be compared in terms of speed and effectiveness. We also provide an API supporting all nine similarity measures for researchers to use with their own data. Links are provided at the following address: http://cs.uef.fi/mopsi/routes/grid.

We first investigate the change in sampling rate (see Figure 22). C-SIM measure is affected the least by the two transformations. C-SIM is not affected at all by increasing the sampling rate because the cell representation is identical due to the interpolation step. Decreasing the sampling rate has minor effect on the similarity because of the inability of interpolation to correctly *guess* the missing parts of the route. However, the effect is much smaller than that of the other methods. In specific, LCSS and EDR are most sensitive to decreasing of the sampling rate though much

less on the increase of sampling rate. To obtain similarity values, we normalize the LCSS and EDR distances by dividing to the average length of the two routes. DTW and FastDTW are robust to the increase of the sampling rate but highly sensitive to the decrease. FastDTW distances are slightly higher than that of DTW because of the approximation errors but the difference is small. Euclidean distance is sensitive to both sampling rate transformations because the transformed route points become misaligned to the original trajectory. ERP is a combination of L_p-norms (such as Euclidean distance) and edit distance. ERP behaves similarly to Euclidean distance for the increase but is robust for the decreases in sampling frequency. Hausdorff and Frechet are both sensitive to changes in sampling rate.

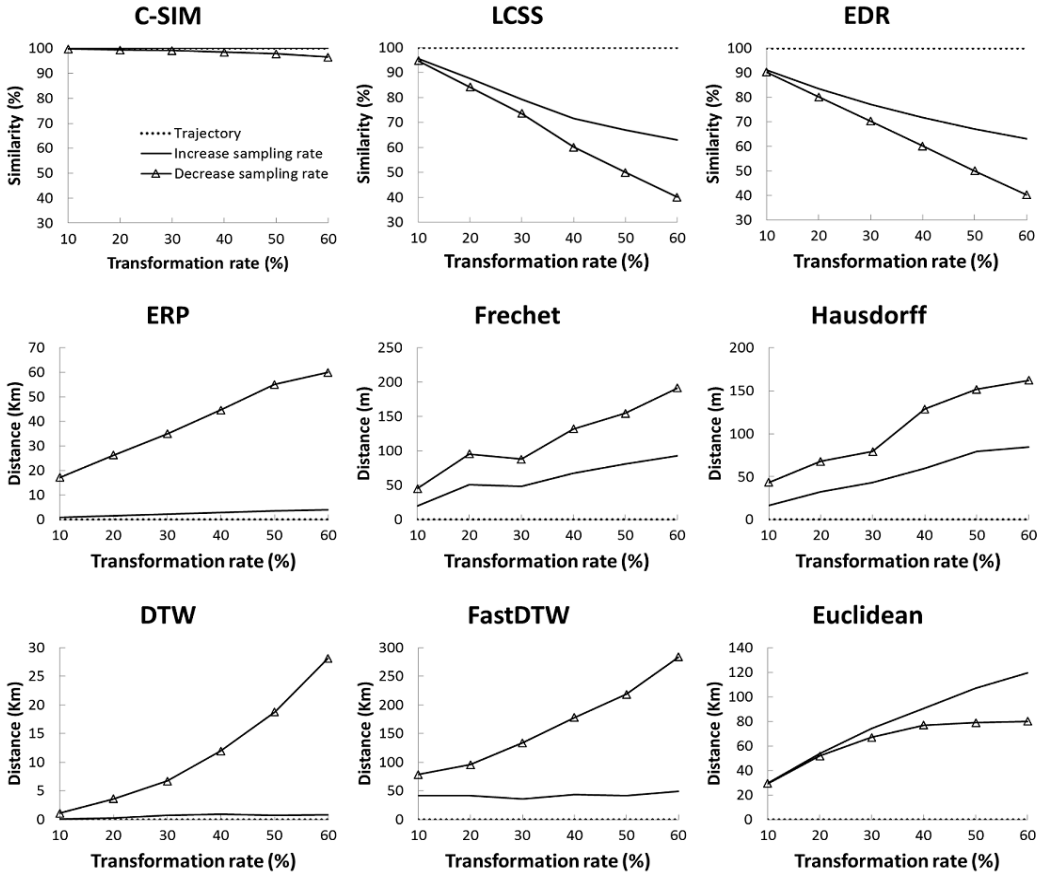


Fig. 22. Effect of changes in sampling rate on nine trajectory similarity measures.

We next examine how the measures behave when noise points are added, and when point locations are shifted (see Figure 23). These transformations depend on a distance parameter. C-SIM, LCSS and EDR measures are not affected by point shifting if the transformation distance is small (L = ε = 25 meters in our experiments). For higher distances, C-SIM decreases proportionally to the transformation distance. LCSS and EDR similarities will not decrease proportionally to the distance; ε is simply a threshold when two points are considered identical. The similarity is higher when transformation distance slightly above ε because points shifted little more than ε meters away are still likely to match with other

points in the vicinity. Noise affects LCSS and EDR more than the other measures because it causes a change in length of the transformed trajectory. DTW and FastDTW are sensitive to all transformations. ERP and Euclidean measures are highly sensitive to noise but they are robust for points shift. This is because when points are only shifted, the original alignment is not influenced much. Frechet and Hausdorff are sensitive to noise and point shifting, but less so if the points are shifted in the same direction (synchronized). The similarity depends linearly on the transformation distance. The results are summarized in Table 5.

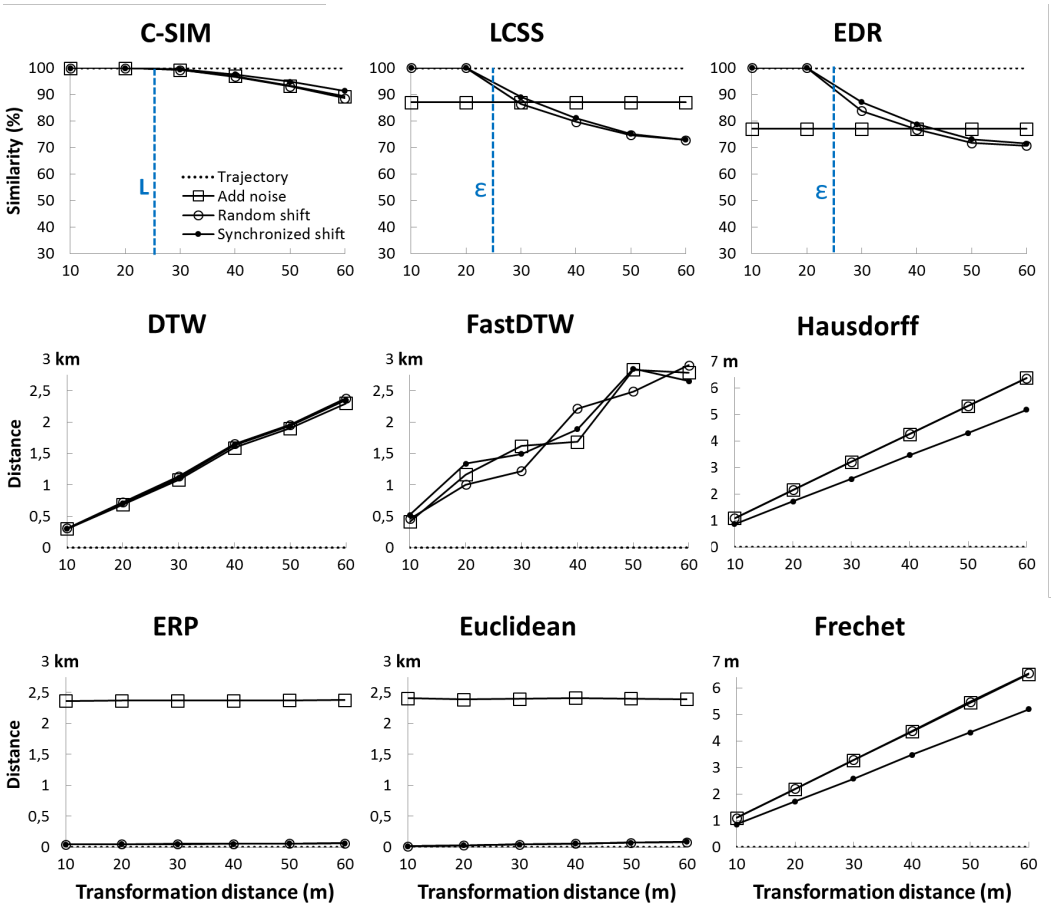


Fig. 23. Effect of nine different similarity measures when adding noise or shifting point locations as a function of transformation distance. In this experiment 30% of the points are altered.

Table V. Summary of the effectiveness of the nine route similarity measures

Definition	Sampling rate		Add noise	Point shifting	
	Increase	Decrease		Random	Sync.
$\text{Grid}(C_A, C_B) = \frac{\lvert C_A \cap C_B \rvert + \lvert C_A \cap C_B^{\,d} \rvert + \lvert C_B \cap C_A^{\,d} \rvert}{\lvert C_A \rvert + \lvert C_B \rvert - \lvert C_A \cap C_B \rvert}$	Robust	Robust	Fair	Fair	Fair
$\text{LCSS}(A,B) = \begin{cases} 0 & \text{, if } n = 0 \text{ or } m = 0 \\ 1 + \text{LCSS}(\text{Rest}(A), \text{Rest}(B)) & \text{, if } d(\text{Head}(A), \text{Head}(B)) \le \varepsilon \\ \max\begin{cases} \text{LCSS}(\text{Rest}(A), B) \\ \text{LCSS}(A, \text{Rest}(B)) \end{cases} & \text{, otherwise} \end{cases}$	Sensitive	Fair	Sensitive	Fair	Fair
$\text{EDR}(A,B) = \begin{cases} n & \text{, if } m = 0 \\ m & \text{, if } n = 0 \\ \min\begin{cases} \text{EDR}(\text{Rest}(R), \text{Rest}(S)) + \text{subcost} \\ \text{EDR}(\text{Rest}(R), S) + 1 \\ \text{EDR}(R, \text{Rest}(S)) + 1 \end{cases} & \text{, otherwise} \end{cases}$ $\text{subcost} = \begin{cases} 0 & \text{, if } d(\text{Head}(A), \text{Head}(B)) \le \varepsilon \\ 1 & \text{, otherwise} \end{cases}$	Sensitive	Fair	Sensitive	Fair	Fair
$\text{DTW}(A,B) = \begin{cases} 0 & \text{, if } n = m = 0 \\ \infty & \text{, if } n = 0 \text{ or } m = 0 \\ d(\text{Head}(A), \text{Head}(B)) + \min\begin{cases} \text{DTW}(A, \text{Rest}(B)) \\ \text{DTW}(\text{Rest}(A), B) \\ \text{DTW}(\text{Rest}(A), \text{Rest}(B)) \end{cases} & \text{, otherwise} \end{cases}$	Robust	Sensitive	Sensitive	Sensitive	Sensitive
FastDTW *(A,B)* = approximation of DTW *(A,B)*	Fair	Sensitive	Sensitive	Sensitive	Sensitive
$\text{ERP}(A,B) = \begin{cases} \sum_1^n \lvert s_i - g \rvert & \text{, if } m = 0 \\ \sum_1^m \lvert r_i - g \rvert & \text{, if } n = 0 \\ \min\begin{cases} \text{ERP}(\text{Rest}(R), \text{Rest}(S)) + d(r_1, s_1) \\ \text{ERP}(\text{Rest}(R), S) + d(r_1, g) \\ \text{ERP}(R, \text{Rest}(S)) + d(s_1, g) \end{cases} & \text{, otherwise} \end{cases}$	Fair	Sensitive	Sensitive	Robust	Robust
$\text{Eulidean}(A,B) = \sqrt{\sum_{i=1}^{\min(n,m)} d(a_i, b_i)^2}$	Sensitive	Sensitive	Sensitive	Robust	Robust
$\text{Hausdorff}(A,B) = \max\begin{cases} d(A,B) \\ d(B,A) \end{cases}$ $d(A,B) = \sup_{a \in A} \inf_{b \in B} d(a,b)$	Sensitive	Sensitive	Sensitive	Sensitive	Fair
$\text{Frechet}(A,B) = \inf_{\alpha,\beta} \max_{t \in [0,1]} \{d(A(\alpha(t)), B(\beta(t)))\}$ α, β - are non-decreasing surjections from [0, 1] → [0, 1].	Sensitive	Sensitive	Sensitive	Sensitive	Fair

6.6 Effect of indexing on RSR algorithm

We demonstrate the efficiency of indexing by computing the similarity ranking using RSR algorithm for 1500 different routes. We choose routes consisting of < 100 cells (including dilations) because the process is very slow when no indexing is applied. In Figure 24, we notice a linear dependency on the number of active cells. This corresponds to the time complexity analysis.

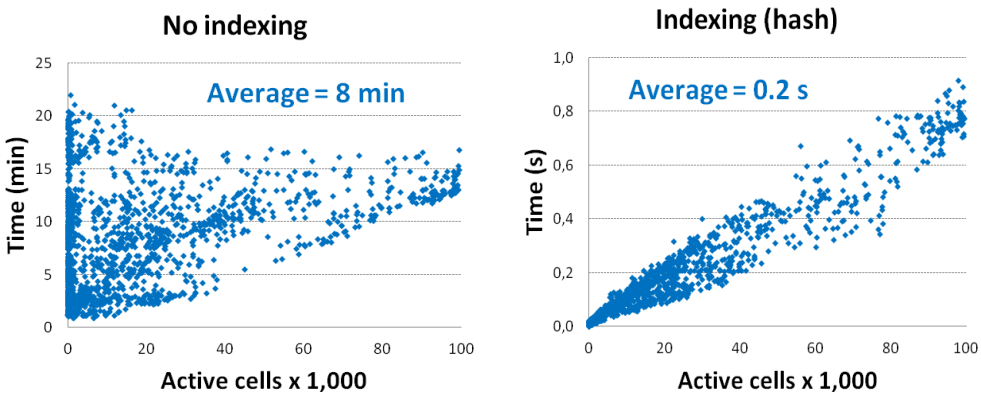


Fig. 24. Time required for calculating the route similarity ranking for 1500 routes plotted as a function of the amount of active cells.

6.7 Comparison of indexing on RSR

We compare the efficiency of RSR algorithm by computing the similarity ranking for every route in Mopsi2014 when using B-tree and hash indexing methods. Routes with large number of active cells were excluded because they were too few to provide significant statistics. We divide the routes into 12 groups: first group routes have less than 40 thousand active cells, the second group between 40 and 80 thousand active cells, and so on. The average processing time and standard deviation for each group are shown in Figure 25. The hash index performs better than B-tree as expected from the time complexity analysis. Hash index takes less than half the time required by B-tree. The average time for B-tree is 2.1 seconds and for hash it is 0.9 seconds. With hash index, RSR performs in real-time (under one second).

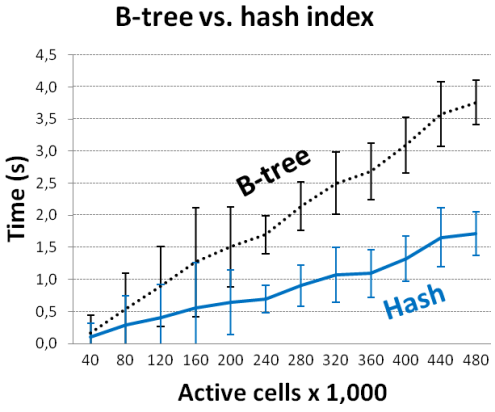


Fig. 25. Comparing B-tree and hash index efficiency for performing the route similarity ranking by showing the average time and standard deviation for routes with different amounts of active cells.

7. CONCLUSIONS

We showed that representing GPS routes as cells of a 2D grid provides efficient computation of different route measures. We presented algorithms for computing four distinct route measures using the proposed grid-based approach. Time complexity was derived for each algorithm and a wide array of experiments were performed on a real route dataset: Mopsi2014.

We compared the new cell based similarity measure C-SIM to existing measures in terms of scalability and effectiveness. In terms of speed it outperforms all compared measures with the exception of Euclidean, however, Euclidean measure is only suitable when routes have the same length, which is not often the case. From an effectiveness point of view, C-SIM is the least affected by changes in sampling rate and performs fairly well under noise and point shifting.

We demonstrated the efficiency of B-tree and hash indexing methods when computing route novelty, noteworthiness and the route similarity ranking. All algorithms perform real-time with the Mopsi2014 dataset. The hash index takes roughly 50% of the time of B-tree but requires 80% more space. We also presented two strategies for computing novelty and noteworthiness in respect to a subset of the database, and concluded that NOV-S is better of these two in most typical use scenarios.

Future research could be done to improve different aspects of the methods. For instance, interpolation may be done by using the underlying road network. Navigation may be implemented by using the cells and the past activity information from inside each cells. This way, popular routes should become apparent. Finally, experimenting with different sizes of cell, and computing cell separate representations at different zoom levels might provide faster processing. These are left as future studies.

REFERENCES

Rakesh Agrawal, Christos Faloutsos and Arun Swami. 1993. Efficient similarity search in sequence databases. In Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms (FODO '93), Chicago, Illinois, USA, 69-84.

Tengfei Bao, Huanhuan Cao, Qiang Yang, Enhong Chen and Jilei Tian. 2012. Mining significant places from cell id trajectories: A geo-grid based approach. In Proceedings of the 13th IEEE International Conference on Mobile Data Management (MDM '12), Bengaluru, India, 288-293.

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 (ACM SIGSPATIAL GIS '09), Seattle, Washington, USA, 3-12.

Jinyang Chen, Rangding Wang, Liangxu Liu and Jiatao Song. 2011. Clustering of trajectories based on Hausdorff distance. In Proceedings of the IEEE International Conference on Electronics, Communications and Control (ICECC '11), Ningbo, China, 1940-1944.

Lei Chen, M. Tamer Ozsu and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In Proceedings of the 2005 ACM SIGMOD international conference on Management of data and Symposium on Principles Database and Systems (SIGMOD/PODS '05), Baltimore, MD, USA, 491-502.

Lei Chen and Raymond Ng. 2004. On the marriage of lp-norms and edit distance. In Proceedings of the 30th International Conference on Very Large Data Bases-Volume (VLDB '04), Toronto, Canada, 792–803.

Minjie Chen, Mantao Xu, and Pasi Fränti. 2012. A Fast Multiresolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. In IEEE Transactions on Image Processing, 21(5), 2770-2785.

Thomas H. Cormen. 2009. Introduction to algorithms. MIT press.

Thomas Eiter and Heikki Mannila. 1994. Computing discrete Fréchet distance. Tech. Report CD-TR 94/64, Information Systems Department, Technical University of Vienna.

Michael R. Evans, Dev Oliver, Shashi Shekhar and Francis Harvey. 2013. Fast and exact network trajectory similarity computation: a case-study on bicycle corridor planning. In Proceedings of the 2nd ACM SIGKDD International Workshop on Urban Computing (UrbComp '13), Chicago, IL, USA, 9.

Alireza Fathi and John Krumm. 2010. Detecting road intersections from gps traces. In Proceedings of the 6th International Conference on Geographic Information Science (GIScience '10), Zurich, Switzerland, 56-69.

Elias Frentzos, Kostas Gratsias, Nikos Pelekis and Yannis Theodoridis. 2007. Algorithms for nearest neighbor search on moving object trajectories. In the International Journal on Advances of Computer Science for Geographic Information Systems (Geoinformatica), 11(2), 159-193.

Elias Frentzos, Kostas Gratsias and Yannis Theodoridis. 2007 .Index-based most similar trajectory search. In Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE '07), Istanbul, Turkey, 816-825.

Ralf Hartmut Güting, Thomas Behr and Jianqiu Xu. 2010. Efficient k-nearest neighbor search on moving object trajectories. In the International Journal on Very Large Data Bases (VLDB), 19(5), 687-714.

Antonin Guttman. 1984. R-trees: a dynamic index structure for spatial searching. In Proceedings of the 1984 ACM SIGMOD international conference on Management of data (SIGMOD '84), New York, NY, USA, 47-57.

James D. Hamilton. 1994. Time series analysis (Vol. 2). Princeton: Princeton university press.

Radu Mariescu-Istodor, Andrei Tabarcea, Rahim Saeidi and Pasi Fränti. 2014. Low complexity spatial similarity measure of GPS trajectories. In Proceedings of the 10th International Conference on Web Information Systems and Technologies (WEBIST'14), Barcelona, Spain, 62-69.

Linsey Xiaolin Pang, Sanjay Chawla, Wei Liu and Yu Zheng. 2013. On detection of emerging anomalous traffic patterns using GPS data. Data & Knowledge Engineering (DKE), 87, 357-373.

Tyrrell R. Rockafellar and Roger J.-B. Wets. 2009. Variational analysis (Vol. 317). Springer Science & Business Media.

Stan Salvador and Philip Chan. 2004. FastDTW: Toward accurate dynamic time warping in linear time and space. In Prcoeedings of the 10th ACM International Conference on Knowledge Discovery and Data Mining Workshop on Mining Temporal and Sequential Data (SIGKDD '04), Seatle, Washington, USA, 70–80.

Shuo Shang, Ruogu Ding, Bo Yuan, Kexin Xie, Kai Zheng and Panos Kalnis. 2012. User oriented trajectory search for trip recommendation. In Proceedings of the 15th ACM International Conference on Extending Database Technology, Berlin, Germany, 156-167.

Izrail Solomonovich Gradshteyn and Iosif Moiseevich Ryzhik. 2000. Tables of Integrals, Series, and Products, 6th ed. San Diego, CA: Academic Press, 1114-1125.

Michail Vlachos, Dimitrios Gunopulos and George Kollios. 2002. Robust similarity measures for mobile object trajectories. In Database and Expert Systems Applications, 2002. In Proceedings of the 13th IEEE International Workshop on Database and Expert Systems Applications (DEXA '02), Aix en Provence, France, 721-726.

Michail Vlachos, George Kollios and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In Proceedings of the 18th IEEE International Conference on Data Engineering (ICDE '02), 673-684.

Karol Waga, Andrei Tabarcea, Minjie Chen and Pasi Fränti. 2012. Detecting movement type by route segmentation and classification. In Proceedings of the 8th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom '12), Pittsburgh, USA, 508-513.

Karol Waga, Andrei Tabarcea, Radu Mariescu-Istodor and Pasi Fränti. 2013. Real Time Access to Multiple GPS Tracks. In Proceedings of the 9th International Conference on Web Information Systems and Technologies (WEBIST '13), Aachen, Germany, 293-299.

John Krumm and Eric Horvitz. 2006. Predestination: Inferring destinations from partial trajectories. In Proceedings of the 8th International Conference on Ubiquitous Computing (UbiComp '06), Orange County, CA, USA, 243-260.

Haibo Wang and Kuien Liu. 2012. User oriented trajectory similarity search. In Proceedings of the ACM SIGKDD International Workshop on Urban Computing (UrbComp '12), Beijing, China, 103-110.

Haozhou Wang, Han Su, Kai Zheng, Shazia Sadiq and Xiaofang Zhou. 2013. An effectiveness study on trajectory similarity measures. In Proceedings of the 24th Australasian Database Conference (ADC '13), Adelaide, Australia, 13-22.

Ling-Yin Wei, Yu Zheng and Wen-Chih Peng. 2012. Constructing popular routes from uncertain trajectories. In Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '12), Beijing, China, 195-203.

Yutaka Yanagisawa, Jun-ichi Akahani and Tetsuji Satoh. 2003. Shape-based similarity query for trajectory of mobile objects. In Proceedings of the 4th International Conference on Mobile Data Management (MDM '03), Melbourne, Australia, 63-77.

Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wang-Chien Lee, Tz-Chiao Weng and Vincent S. Tseng. 2010. Mining user similarity from semantic trajectories. In Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks (ACM SIGSPATIAL GIS '10), San Jose, CA, USA, 19-26.

Xia Ying, Zhang Xu and Wang Guo Yin. 2009. Cluster-based congestion outlier detection method on trajectory data. In Proceedings of the 6th IEEE International Conference on Fuzzy Systems and Knowledge Discovery (FSKD '09), Tianjin, China, 243-247.

Daqing Zhang, Nan Li, Zhi-Hua Zhou, Chao Chen, Lin Sun and Shijian Li. 2011. iBAT: detecting anomalous taxi trajectories from GPS traces. In Proceedings of the 13th ACM international conference on Ubiquitous Computing (UbiComp '11), Beijing, China, 99-108.

Vincent W Zheng, Yu Zheng, Xing Xie and Qiang Yang. 2010. Collaborative location and activity recommendations with gps history data. In Proceedings of the 19th ACM International Conference on World Wide Web (WWW '10), New York, NY, USA, 1029-1038.

Yu Zheng and Xiaofang Zhou. 2011. Computing with spatial trajectories, Springer Science & Business Media.

Yu Zheng, Xing Xie and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from gps trajectories. In Proceedings of the 18th ACM International Conference on World Wide Web (WWW '09), Madrid, Spain, 791–800.

Paper IV

Mariescu-Istodor R. & Fränti P.
”Gesture input for GPS route search”
Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR). Springer International Publishing
pp. 439-449, 2016

Gesture Input for GPS Route Search

Radu Mariescu-Istodor(✉) and Pasi Fränti

University of Eastern Finland, Joensuu, Finland
{radum, franti}@cs.uef.fi

Abstract. We present a simple and user-friendly tool for an efficient search from a spatial database containing GPS tracks. The input is a sketch of a route drawn by a user on a map by mouse, hand or other means. This type of interaction is useful when a user does not remember the date and time of a specific route, but remembers its shape approximately. We evaluate the efficiency of the retrieval when the shape given by the gesture is simple or complex, and when the area contains either a small or large number of routes. We use the Mopsi2014 route dataset to demonstrate that the search works in real time.

Keywords: GPS · Route · Gesture · Matching · Touchscreen · Draw

1 Introduction

GPS-enabled smartphones allow users to collect large amounts of location-based data such as geo-tagged notes, photos, videos and geographical trajectories hereafter referred to as *routes*. Mobile users track routes for reasons like: recording travel experiences, recommending a certain path and keeping track of personal statistics in sports such as hiking, running, cycling and skiing. A sample route collection is shown in Fig. 1. From a large collection like this, it is difficult to find a specific route unless user remembers the date when it was recorded. Otherwise the amount of data is overwhelming to perform systematic search from among all the records.

Many applications exist that allow users to track their movement; some of these are: *Sports Tracker*[1], *Endomondo*[2], *Strava*[3] and *Mopsi*[4]. Mopsi is a location-based social network created by the School of Computing at the University of Eastern Finland. Mopsi users can find out who or what is around. They can track their movements, share photos and chat with friends. Mopsi includes fast retrieval and visualization of routes [1] using a real-time route reduction technique [2]. Transport mode information is automatically inferred by analyzing the speed variance of the route [3]. Movement is classified as either: walking, running, cycling or car. Route similarity, novelty, inclusion and noteworthiness [4, 5] are computed by using cell representations of the routes created by a grid which covers the planet. Searching for spatially similar routes is done efficiently by indexing these cells.

[1] http://www.sports-tracker.com.
[2] https://www.endomondo.com.
[3] https://www.strava.com.
[4] http://cs.uef.fi/mopsi.

A. Robles-Kelly et al. (Eds.): S+SSPR 2016, LNCS 10029, pp. 439–449, 2016.
DOI: 10.1007/978-3-319-49055-7_39

Fig. 1. Route collection of user Pasi over the city of Joensuu, Finland is shown on left. The collection spans from 2008 to 2014. A circle-shaped route that we want to find is emphasized. Four attempts (all failed) to find this route by clicking the map are shown right.

We propose a real-time search for routes in the Mopsi collection by using gestures and pattern matching. The gesture is a hand-drawn input in the form of a free shape done on a map. The shape approximates the locations where the targeted route passes through. According to [6], this gesture can be classified as of the *symbolic* type, implying that it has no meaning when performed in other contexts (not using a map). Referring to the taxonomy in [7] the result of the gesture is to trigger a command: search for route(s) with given spatial characteristics. This search works by computing the similarity between the input gesture and every route in the database. The most similar route candidates are provided to the user.

Gestures have been used as a means to access menu items without the need to traverse large hierarchies. In [8], gestures are continuous pen traces on top of a stylus keyboard. This soft keyboard can be inconvenience as it wastes screen space unnecessarily. In our method, we use the underlying map as a canvas for drawing the gestures. On desktop computers, the gesture mode is explicitly activated by holding a hotkey while drawing the gesture by mouse. On touchscreens, we need to distinguish the gesture from normal map interaction (panning and zooming). In [9], it was discovered that it is possible to distinguish gesture from other touch events such as scrolling or tapping by buffering the touch events and analyzing the queue to determine if the sequence is a gesture or not. We use this method to activate the gesture mode, and neither designated area nor activation button are therefore needed.

Typically, symbolic gesture-based systems require the user to learn a set of symbols [6]. Our method is simpler as no learning of symbols is required. However, the user is expected to understand and be able to read maps because the roads, buildings and terrain elements such as forests, lakes and rivers are the key information used when giving the input. For example, user may draw the input by following a river front, road, or other landmarks visible on map. Users who have a large route collection benefit most from the gesture search. It is therefore fair to assume that these users have also the necessary skills to understand maps.

2 User Interface

2.1 System Overview

Let us assume that Mopsi user Pasi wants to review the statistics of a specific route from his collection but he does not recall the date. Pasi knows that the route is in Joensuu, Finland so he proceeds to move the map to this location. Figure 1 shows that Pasi has a large route collection in Joensuu. Let us further assume that he wishes to find the highlighted circular route. Exhaustive search among all the routes would not be reasonable so best change is to try to distinguish the route on map. In Mopsi, this is possible by clicking any individual route on the map. However, this is also difficult because the targeted route overlaps with many others.

Gesture search enables a user to search routes by drawing the sample shape of the desired route over the map. Figure 2 shows how Pasi's route is found by drawing a circular gesture on the map around the center of Joensuu. The search returns four possible candidates, including the one he was looking for.

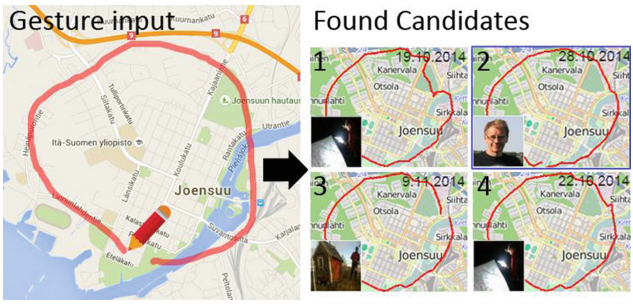


Fig. 2. A circle-shaped gesture surrounding the center of Joensuu reveals four circular route candidates. Pasi's route is number two in the list.

2.2 Map Handling

Mopsi uses Google Maps and OpenStreetMap. They both offer several built in functions for user interaction. A user can pan the map by clicking and dragging it in the desired direction. Zooming in can be done by double left-click and zooming out is done by double right-click. Zooming can be also done using the mouse wheel or by the pinch gesture.

To start the gesture search on a computer, user presses a hotkey (Ctrl). When pressed, the built-in map handling functions are disabled and the gesture input mode is enabled. In this mode, a user can draw on the map by clicking, holding and moving the mouse around while keeping the hotkey pressed. Releasing the hotkey causes two things to happen simultaneously: the input gesture is sent to the server for processing the query, and default map behavior is reactivated.

Majority of touchscreens nowadays do not have a keyboard and existing buttons serve for different purposes such as exiting applications, changing volume levels or enabling the camera. It is possible to implement a soft button on the screen to toggle the

gesture input mode however this wastes screen space which makes drawing more difficult, especially on small screens.

Instead, we activate the gesture first by a click (tap) and then, immediately, touch the screen again to draw the shape. We denote this event as *Tap-and-Draw*. The *Draw* event works similarly as panning the map, however, the preceding *Tap* event triggers gesture input mode. When the *Draw* gesture is complete, the input gesture is sent to the server and the search is initiated; default map behavior is reactivated.

2.3 Real-Time Route Search

The search returns the route(s) that are most similar to the shape of the gesture input. For the matching, we use the method in [5]. It computes the spatial similarity between routes by first representing them as cells in a grid and then using the Jaccard similarity coefficient:

$$J(C_A, C_B) = \frac{|C_A \cap C_B|}{|C_A \cup C_B|}, \tag{1}$$

where C_A and C_B are two sets of cells. However, because of the arbitrary division of the grid, route points may end up in different cells even though the points are close to each other. This problem is solved by applying morphological dilation with square structural element and using the additional cells as a buffer region when computing the similarity. The similarity is then formulated as:

$$S(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap C_B^d| + |C_B \cap C_A^d|}{|C_A| + |C_B| - |C_A \cap C_B|}, \tag{2}$$

where C_A^d and C_B^d are the dilated regions of routes C_A and C_B respectively. To make the search efficient we pre-compute the cell representation and use B-tree index [12] on the cell database. With this setup the search works real-time.

To perform the search, the input shape is converted into cells. The similarity between this cell set and all routes is then computed using (2). The result is similarity ranking which often contains a multitude of results with varying levels of similarity to the given shape. To the user we present only the most significant candidates.

2.4 Map Projection and the Grid

Most online maps (*Google Maps*, *OpenStreetMap*, *Yahoo! Maps*, *Bing Maps*) use a variant of the *Mercator* projection [10]. In Mopsi, we use Google Maps or OpenStreetMap as the map interface. *Mercator* is a cylindrical map projection which preserves the angles, however, the linear scale increases with latitude. The parallels and meridians are straight and perpendicular to each other. The meridians are equidistant, but the parallels become sparser as they further themselves from the equator.

Creating a grid by choosing a fixed cell size (in degrees) will cause the cells to appear vertically stretched when viewed on the Mercator projection. The amount cells stretch increases the farther away they are from the equator. In Joensuu, Finland the cells appear twice as tall as they are wide.

2.5 Multi-resolution Grids

The precision of drawing the gesture should be independent on the zoom level of the map. When the zoom level is decreased by one unit the content of the map becomes half of its previous size, and consequently, the regions on the map become twice as difficult to read. We create 10 grids with different resolutions and store the routes at each of these approximation levels (see Table 1).

The finest grid has a cell size of 25 × 25 meters. Finer grids are not needed because at this level, GPS error becomes already apparent and the route approximations become unreliable. The amount of cells needed increases exponentially when finer grids are produced. Therefore, we do not compute unnecessary levels in vain. Sparsest grid has cell length of 12.5 km. At lower levels (≥ 25 km) the cell size becomes so big that even the longest routes are represented by only a few cells.

Table 1. A mapping from zoom-level to the grid resolution. The statistics are for Mopsi2014 Route dataset using each of the grid resolutions.

Zoom level	≤ 6	7	8	9	10	11	12	13	14	≥15
Grid resolution	0	1	2	3	4	5	6	7	8	9
Cell size (km)	12,8	6,4	3,2	1,6	0,8	0,4	0,2	0,1	50 m	25 m
Amount of cells	7× 10^4	9× 10^4	1× 10^5	2× 10^5	4× 10^5	7× 10^5	1× 10^6	3× 10^6	5× 10^6	1× 10^7
Memory (MB)	3,5	4,5	6,5	9,5	16,5	30,6	59,6	118,6	238	486
B-tree Index (MB)	8,5	9,5	13,5	21,5	35,6	66,7	131,8	263,1	526	1,1 GB

3 Route Search

We present next our algorithm for performing the gesture-based route search. The algorithm (*GSearch*) first extracts the cells that the input shape passes through using the *Find-Cells* function. This function chooses the correct grid resolution based on the zoom level using the mapping presented in Table 1. Every point is then mapped to the cell it resides in. At the Equator, one degree is roughly 111 km and the smallest cell length we support is 25 × 25 meters. We dilate the input route C_A with 3 × 3 square structural element to obtain C_A^d.

GSearch: Searches for route candidates matching a given gesture.

Input: gesture shape G, zoom level z
Output: candidates list L
C, C^d ← **Find-Cells** (G, z)
ranking ← **RSR** (C, C^d, z)
Top-Cluster ← **Cluster** (ranking. similarities)
for i ← 1 to size (ranking) **do**
  if ranking [i] is in Top-Cluster **then**
    add ranking [i] to L. append (ranking [i])

Find-Cells: Obtains the cells that a given shape passes through at a specific zoom level.

```
Input: shape S, zoom level z
Output: cell set C, dilated region C^d
r               ← Get-Resolution ( z ) // according to Table 1
min-cell-size   ← 25 // meters
degree-size     ← 111 // km
dividing-factor ← min-cell-size * degree-size / pow( 2, r )
for i ← 1 to size ( S ) do
    x ← round ( S [ i ]. latitude * dividing-factor)
    y ← round ( S [ i ]. longitude * dividing-factor)
    add ( x, y ) to C
C^d             ← Dilate ( C )
```

Route Similarity Ranking (*RSR*) algorithm is then applied to find all similar routes in the database. *RSR* iterates through every cell in C_A and C_A^d, and finds what other routes pass through the same cells. For each found route C_B, it checks if the cell belongs to $C_A \cap C_B$, $C_A \cap C_B^d$ or $C_A^d \cap C_B$. The algorithm maintains counters for each type and uses them for computing the similarity values using (2). Time complexity is $O((|C| + |C^d|)(\log(MQ)) + a(C) + a(C^d))$ where M is the number of routes in the database, Q is the average route size in cells and $a(C) = \sum \left(|C \cap C_i| + |C \cap C_i^d|\right)$, $i = \overline{1, M}$.

The similarity ranking usually results in a large number of routes, of which only few are relevant to the user. It might be possible to filter out routes below a given threshold, but then we might get no result in some cases; the other extreme is when searching for a very common route. Then there can be too many results above the threshold. Therefore, we limit the number of results using clustering as follows.

RSR: Computing the route similarity ranking.

```
Input: cells C, dilated part C^d, zoom level z
Output: ranking of routes according to similarity values
SC ← initialize SetCounter array; // structure defined below
// process input route
for i ← 1 to size ( C ) do
    R_i, R^d_i ← Get-Routes ( C [ i ] )
    for j ← 1 to size ( R_i ) do
        SC [ R_i [ j ] ]. A ++; SC [ R_i [ j ] ]. B ++; SC [ R_i [ j ] ]. AB ++;
    for j ← 1 to size ( R^d_i ) do
        SC [ R^d_i [ j ] ]. A ++; SC [ R^d_i [ j ] ]. B ++; SC [ R^d_i [ j ] ]. AB^d ++;
// process dilated part
for i ← 1 to size (C^d ) do
    R_i, R^d_i ← Get-Routes ( C^d [ i ] )
    for j ← 1 to size ( R_i ) do
        SC [ R_i [ j ] ]. B ++; SC [ R_i [ j ] ]. A^dB ++;
    for j ← 1 to size ( R^d_i ) do
        SC [ R^d_i [ j ] ]. B ++; SC [ R^d_i [ j ] ]. A^dB^d ++;
ranking ← new list;
for each r_id in SC do
    sim ← ( SC [ r_id ] . AB + SC [ r_id ] . A^dB + SC [ r_id ] . AB^d ) /
          ( SC [ r_id ] . A + SC [ r_id ] . B - SC [ r_id ] . AB)
    add ( r_id, sim ) to ranking
SetCounter {  A←0;  B←0;  AB←0;  A^dB←0;  AB^d←0;  }
```

We cluster the threshold values by *Random Swap* (*RS*) algorithm [11] with 10,000 iterations with 16 clusters. The algorithm alternates between K-Means and random relocation of centroids in order to avoid getting stuck in a local optimum. The algorithm converges to the final result in few hundreds of iterations, on average. However, since *Random Swap* is fast, we can afford to use 10,000 iterations to increase the probability of finding optimal partitioning.

From the clustering result, we take the cluster having the routes of highest similarities. The idea is that the clustering will find the size of this set automatically by fitting the clustering structure to the distribution of the similarities.

Cluster: Limits the ranking to the most likely candidates.

Input: similarities S
Output: cluster with highest similarities
T ← 10.000 // number of iterations
M ← 16 // number of clusters
P ← **RS** (S, T, M) // Random Swap clustering
Top-Partition ← 0; Top-Cluster ← empty set
for i ← 1 to size (S) **do**
  if S [i] = max (S) **then**
    Top-Partition ← P [i]
for i ← 1 to size (S) **do**
  if P [i] = Top-Partition **then**
    add S [i] to Top-Cluster

4 Experiments

We perform experiments using the Mopsi2014[5] dataset, which is a subset of all routes from the Mopsi database collected by the end of 2014. It contains 6,779 routes recorded by 51 users who have 10 or more routes. Routes consists of a wide range of activities including walking, cycling, hiking, jogging, orienteering, skiing, driving, traveling by bus, train or boat. Most routes are in Joensuu region, Finland, which creates a very dense area suitable for stressing the method. A summary of the dataset is shown in Table 2. All experiments were performed on Dell R920, 4 x E7-4860 (total 48 cores), 1 TB, 4 TB SAS HD.

Table 2. Statistics of Mopsi2014 route dataset.

Routes	Points	Kilometers	Hours
6,779	7,850,387	87,851	4,504

[5] http://cs.uef.fi/mopsi/routes/dataset.

4.1 Efficiency of the Search

The efficiency of the search is proportional to the size of the database, and to the resolution of the grid. The grid to be chosen depends on the zoom level required to view the targeted route on the map: small routes are best viewed using a higher zoom-level. The grid depends also on the size of the screen. To get a better understanding of this we computed the zoom-level for the best-view of each route in the Mopsi2014 dataset. We consider the best-view as the maximum zoom-level that shows the entire route on screen. The results in Fig. 3 show that lowest zoom levels are rarely used. Routes in such zoom levels should span across multiple countries or even continents, and thus, are rare in the dataset. The highest zoom levels (20–21) are also not often used because they cover only very short routes, usually non-movement records.

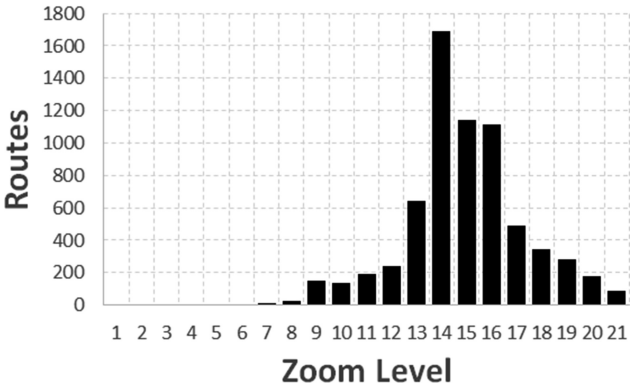


Fig. 3. Histogram showing what zoom-levels are used more often when viewing routes.

When computing the histogram from Fig. 3, we assumed a screen size of 1366 × 768, which, according to the free statistics provided by W3Counter[6], was the most used screen size during March 2016.

We next compute the efficiency of the G-Search algorithm by taking every route in Mopsi2014 as the target route. The best-viewed zoom level for them is first found. A perfect gesture is then simulated for the route by selecting the cells it travels through. Search is then performed using the default screen size of 1366 × 768. The results are summarized in Fig. 4. As expected, the time required is small (0.2–0.8 s) at small zoom levels. At the largest zoom levels the time is also small, but this is against expectations. The reason for the low execution times is the fact that for zoom level 15 and above, the same grid is used and, as a result, the number of cells required to represent each route is lower. Only the middle level routes can take slightly more than 1 s.

This experiment shows that, given a random target route, the expected search time is about 1 s or less, thus, it can be considered real-time. In practice, a smaller zoom level is used by the user than the best-fitting one is selected. Thus, < 1 s result happens

[6] https://www.w3counter.com/globalstats.php.

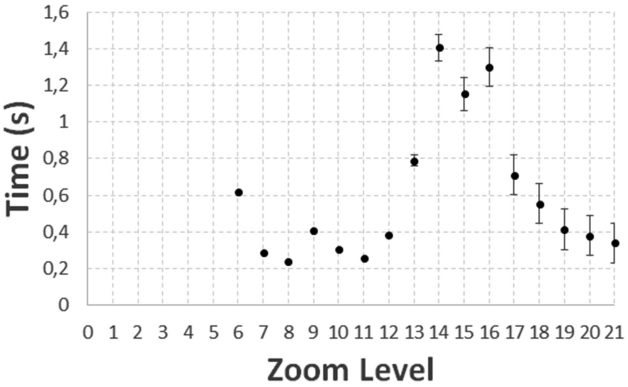


Fig. 4. Times required by G-Search when searching all routes in Mopsi2014 dataset. The results are grouped by the zoom-level and averaged. The average of all searches is 0.9 s.

more often. The reason is that often at zoom levels just below the best-fitting one it is easier to see the landmarks on the map. Furthermore, it is possible that at the best-fitting level the gesture implies drawing on the edges of the map, which are more difficult to target than the central area. Another reason is that the 1366 × 768 screen size is large, and using a smaller screen implies a finer grid will be used. Processing times with default screen size of 320 × 658 yields even smaller processing time of about 0.2 s.

The search time also depends on the density of the routes. In low density areas (< 200 routes), the search time is 0.14 s, on average. In very dense areas (> 1000 routes) the search time is 2.2 s, on average. There is also minor dependency on the size of the gesture. A gesture passing through 50 cells takes 0.7 s time on average, whereas as gesture passing through 200 cells takes 0.7 s, on average. The upper limit is the number of cells that can fit on the screen (3600 with the 1366 × 768 screen size).

4.2 Usability Evaluation

We study next the efficiency of the gesture search from usability point of view. We compare the average time user spends on searching a randomly chose route using the gesture search and using the previous (traditional) system. Eleven volunteers were asked to search randomly selected routes using a tool[7] built for this purpose as follows:

A target route was shown on map but no date, length or duration were shown. User can study and memorize the route as long as wanted.

When user pressed the *Start* button, user was (randomly) directed either to the traditional system or to the new Gesture search. Timer was started.

The task was to find the route and input its date and then press *Stop* button. If the date was correct the timer was stopped. If the user considered the task too difficult he was allowed to press *Give-up* button.

[7] http://cs.uef.fi/mopsi/routes/gestureSearch/qual.php.

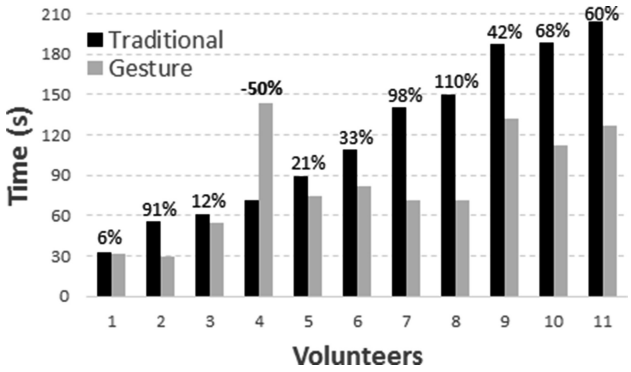


Fig. 5. Average search times and the relative difference between traditional and gesture search.

Each volunteer was asked to repeat the test at least 10 times, or as long as he/she found it fun to do.

In total, 106 routes were searched using the traditional system, and 98 using the gesture search. The searched routes were found 77 % of the time using traditional search compared to 91 % when using gestures. Gesture search was 41 % faster, on average. The individual performance differences are shown in Fig. 5. Traditional search is slower on average than gesture search for all except one user.

The search time is affected also by other factors such as complexity and length of the route, and density of the areas the route passes through. We next group the results by these three factors. The complexity is calculated as the number of points used by the polygonal approximation [2] to represent the route at its best-fit zoom level. Density is calculated as the proportion of cells that are overloaded by other routes; it is the opposite to the noteworthiness value in [5]. Results in Table 3 show that although shorter and less complex routes in low density areas are faster to find, the Gesture search outperforms the traditional approach in all cases.

The volunteers were also asked if they liked the Gesture search and which one they would prefer for such search task. They all rated Gesture search as good (10) or excellent (1). Most (9) preferred Gesture search, none (0) preferred the traditional search, and some (2) would not use either. Written comments included "*I really liked it*" and "*It was fun*".

Table 3. Average search times when grouped by different factors.

	Length		Complexity		Density	
	Short 2.7 km	Long 12.7 km	Low 31 pts	High 128 pts	Low 12 %	High 75 %
Traditional	90 s	116 s	87 s	120 s	90 s	117 s
Gesture	64 s	78 s	65 s	77 s	54 s	88 s
Reduction	30 %	33 %	25 %	36 %	30 %	24 %

5 Conclusion

We showed that gestures can be successfully used as input for searching routes from large data collections. We solved all the components of the search including user input, database optimization, pattern matching, and selecting threshold by clustering to show only the most significant results. The effectiveness of the method was demonstrated by run time analysis showing that it works real time, and by usability experiments showing that it outperforms traditional search.

References

1. Waga, K., Tabarcea, A., Mariescu-Istodor, R., Fränti, P.: Real time access to multiple GPS tracks. In: International Conference on Web Information Systems and Technologies (WEBIST 2013), Aachen, Germany, pp. 293–299 (2013)
2. Chen, M., Xu, M., Fränti, P.: A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. IEEE Trans. Image Process. **21**(5), 2770–2785 (2012)
3. Waga, K., Tabarcea, A., Chen, M., and Fränti, P.: Detecting movement type by route segmentation and classification. In: IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2012), Pittsburgh, USA, pp. 508–513 (2012)
4. Mariescu-Istodor, R., Tabarcea, A., Saeidi, R., Fränti, P.: Low complexity spatial similarity measure of GPS trajectories. In: International Conference on Web Information Systems and Technologies (WEBIST 2014), Barcelona, Spain, pp. 62–69 (2014)
5. Mariescu-Istodor, R., Fränti, P.: Grid-based method for GPS route analysis and retrieval. Manuscript (2016, submitted)
6. Cirelli, M., Nakamura, R.: A survey on multi-touch gesture recognition and multi-touch frameworks. In: ACM Conference on Interactive Tabletops and Surfaces (ITS 2014), Dresden, Germany, pp. 35–44 (2014)
7. Karam, M., Schraefel, M.C.: A taxonomy of Gestures in Human Computer Interaction. ACM Transactions on Computer-Human Interactions (2015, in press)
8. Kristensson, P.O., Zhai, S.: Command strokes with and without preview: using pen gestures on keyboard for command selection. In: SIGCHI Conference on Human Factors in Computing Systems (CHI 2007), New York, USA, pp. 1137–1146 (2007)
9. Li, Y.: Gesture search: a tool for fast mobile data access. In: ACM Symposium on User Interface Software and Technology (UIST 2010), New York, USA, pp. 87–96 (2010)
10. Kennedy, M., Kopp, S.: Understanding Map Projections. ESRI Press, Redlands (2001)
11. Fränti, P., Kivijärvi, J.: Randomized local search algorithm for the clustering problem. Pattern Anal. Appl. **3**(4), 358–369 (2000)
12. Cormen, T.H.: Introduction to Algorithms. MIT press, Cambridge (2009)

Paper V

Mariescu-Istodor R. & Fränti P.
”CellNet: Inferring road networks from GPS trajectories”
(submitted) 2017

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).

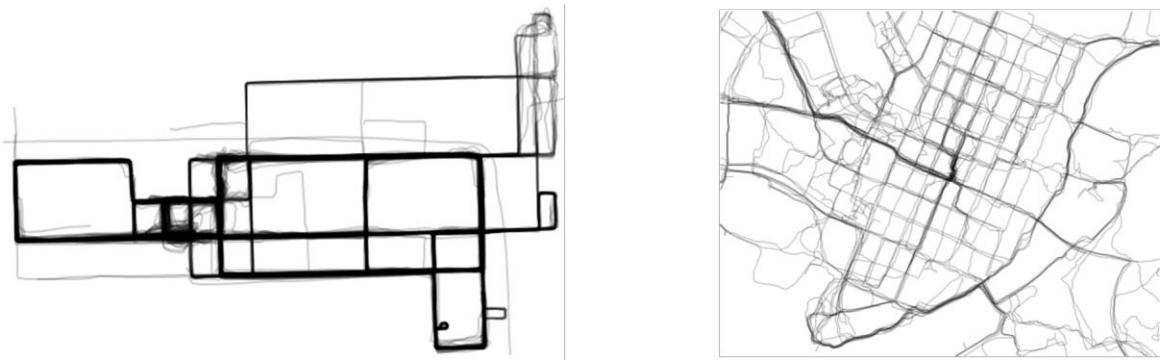


Figure 2. GPS routes in Chicago, USA (left panel) and Joensuu, Finland (right panel).

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.

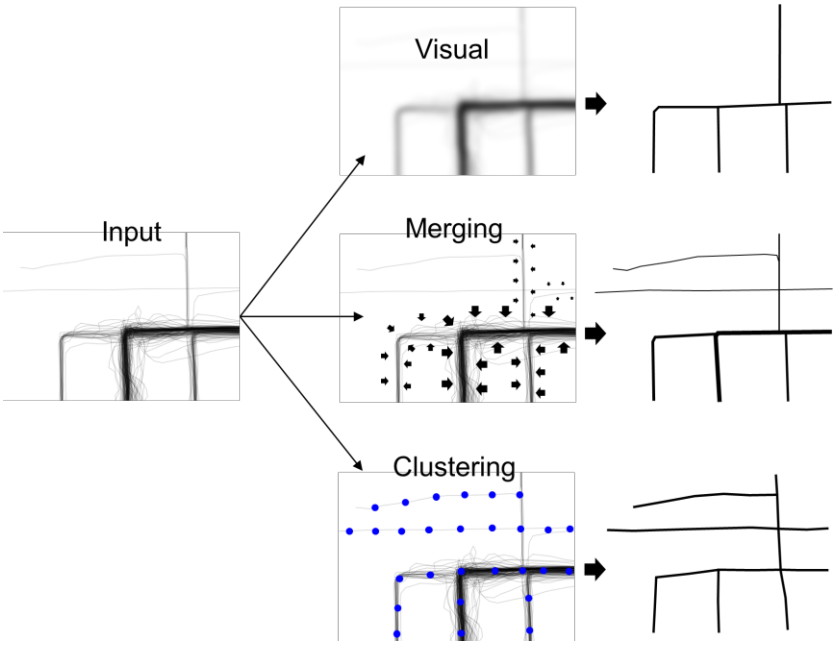


Figure 3. Three conceptually different road network generation techniques: visual, merging and clustering.

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.

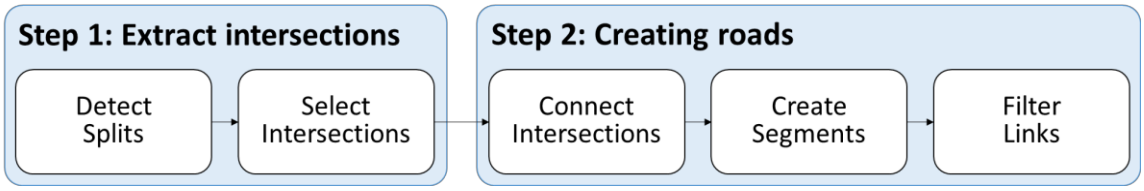


Figure 4. The steps performed by CellNet to infer a road network.

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.

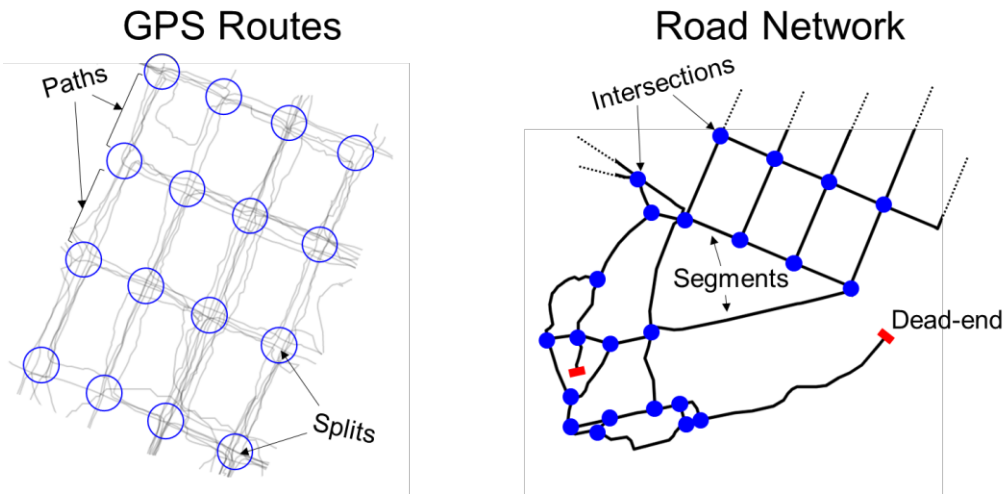


Figure 5. Diagram showing terminology used to discuss GPS routes and road networks.

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.

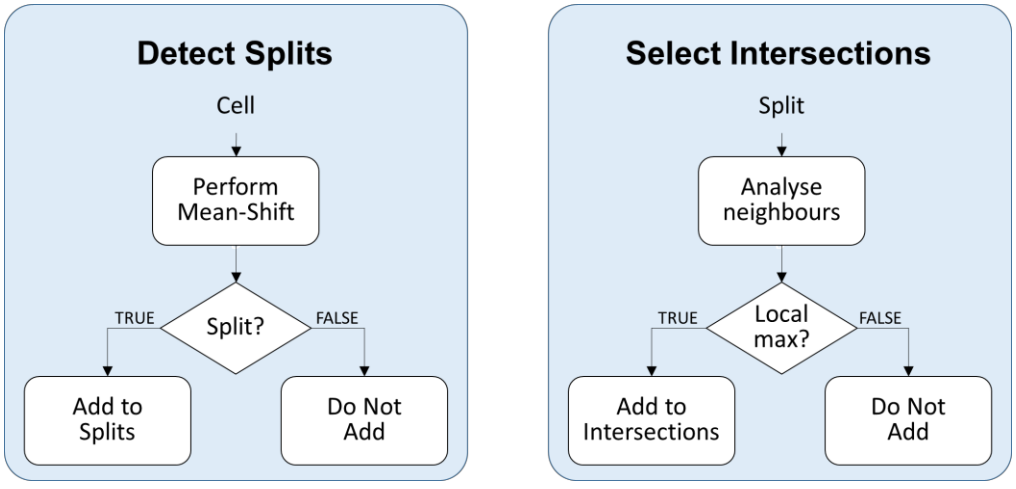


Figure 6. Steps performed to detect splits (left panel) and to select local maxima (right panel).

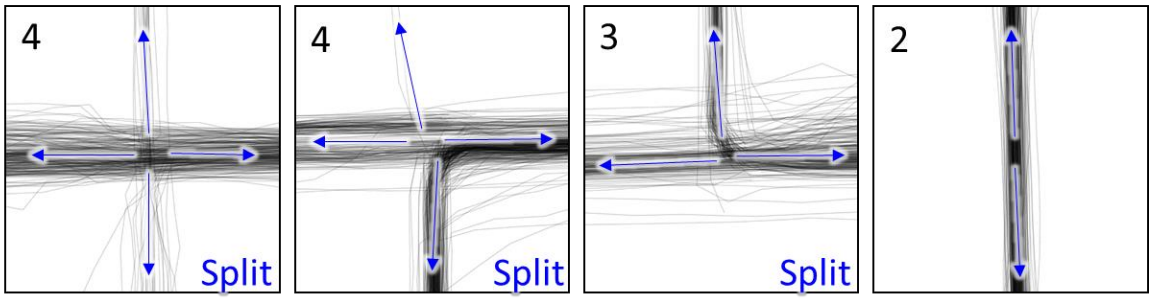


Figure 7. Four examples of locations at which routes head off into several principal directions. The directions are highlighted by arrows. The first three examples are splits, according to our definition, whereas the last is not. The numbers (upper left corners) indicate the quantity of principal directions.

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].

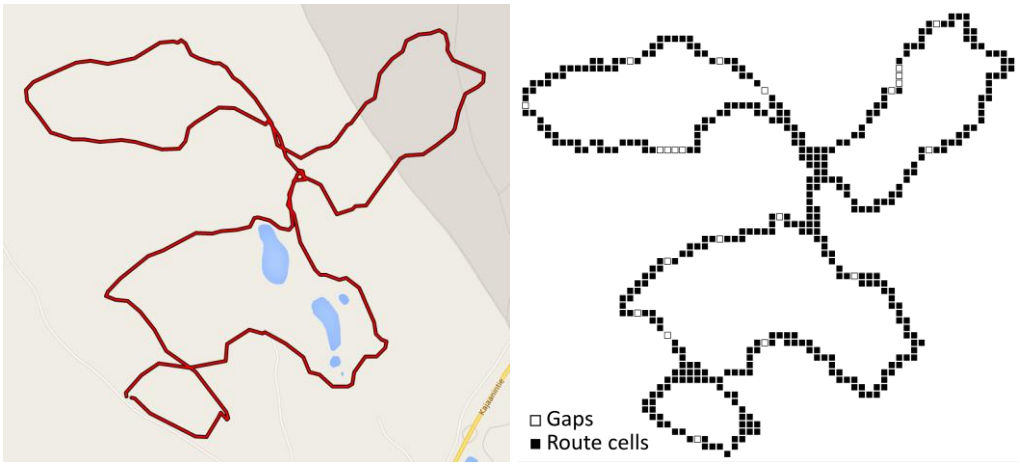


Figure 8. A sample route (top panel) and the cell representation with cell size 25 m × 25 m (lower panel). The gaps are filled using linear interpolation.

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.

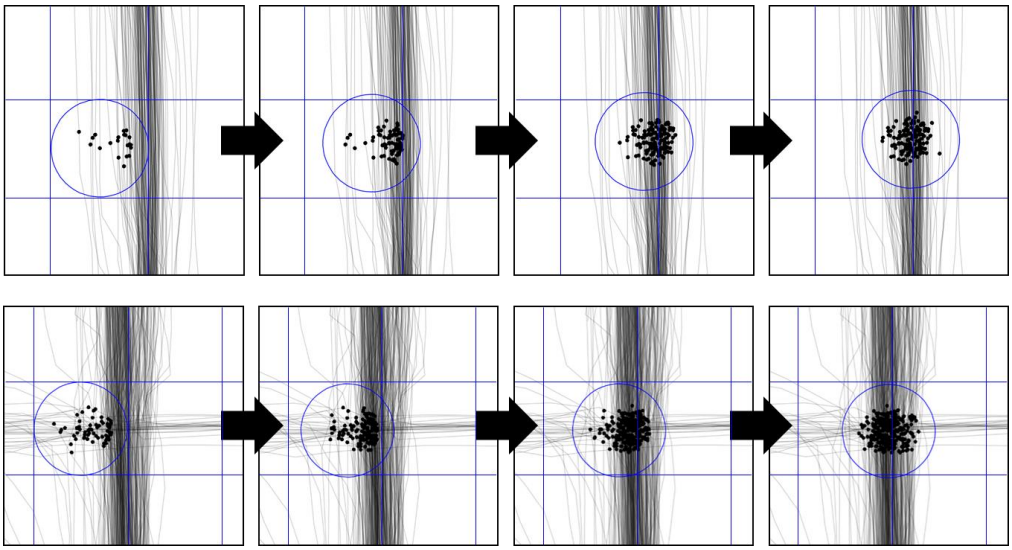


Figure 9. Two examples of the mean-shift algorithm. The initial location gradually moves towards the centre of the routes. If an intersection is nearby, the location is likely to end up at its centre.

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).

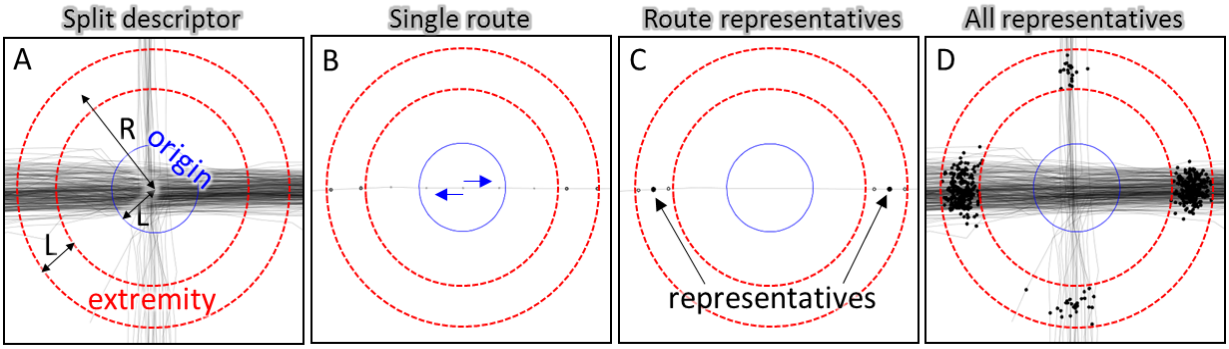


Figure 10. A, the split descriptor composed of the origin and the extremity. B, a sample route traversing through the point of interest; points inside the extremity are highlighted. C, the points inside the extremity are averaged in each of the two directions to create the representatives. D, representatives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

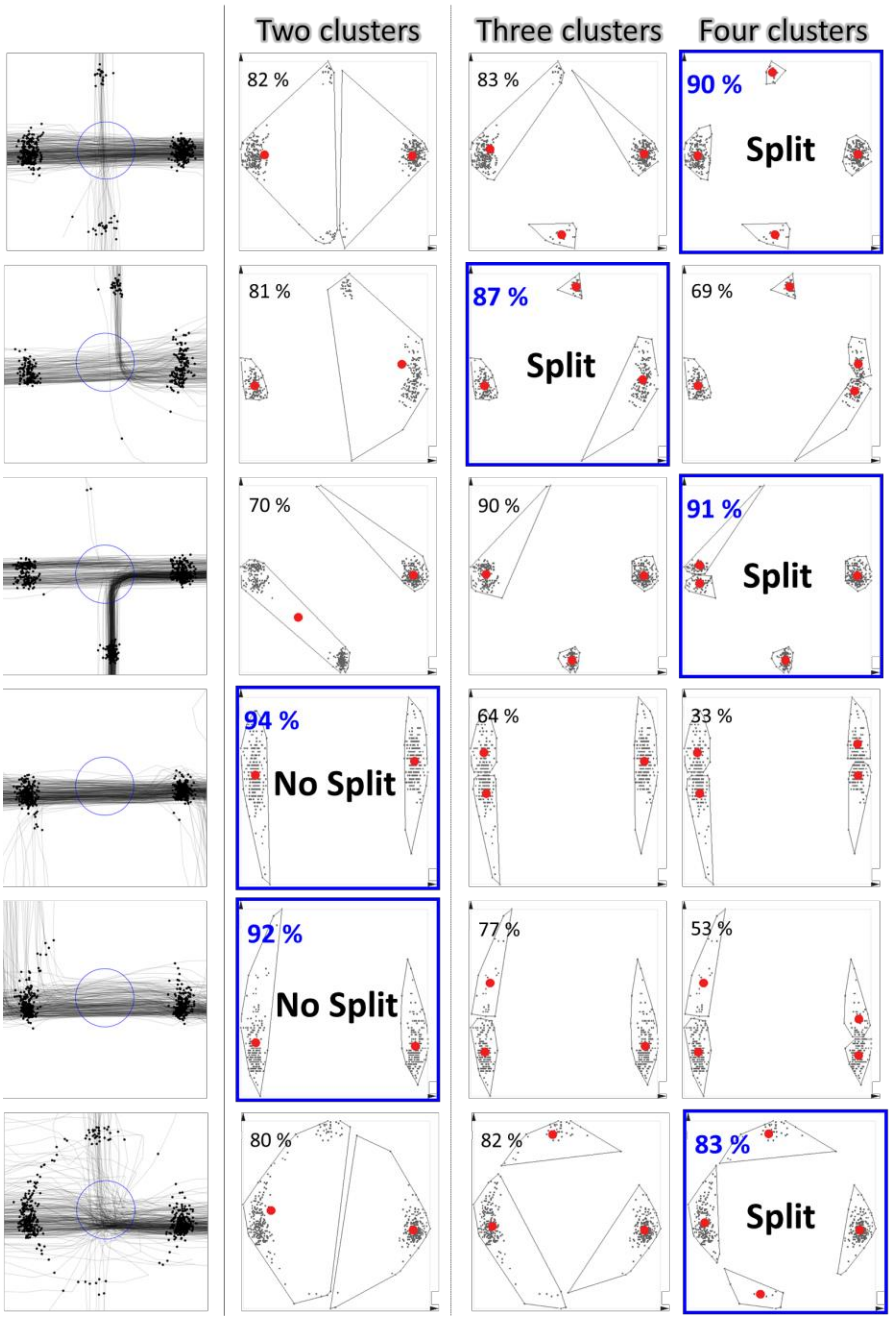


Figure 11. Six locations investigated for splits. Each dataset is clustered by the random swap algorithm using 2, 3 and 4 clusters respectively. The percentages represent the value of the silhouette coefficient. The occurrence of more than 2 clusters indicates a split.

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:

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

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

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 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.

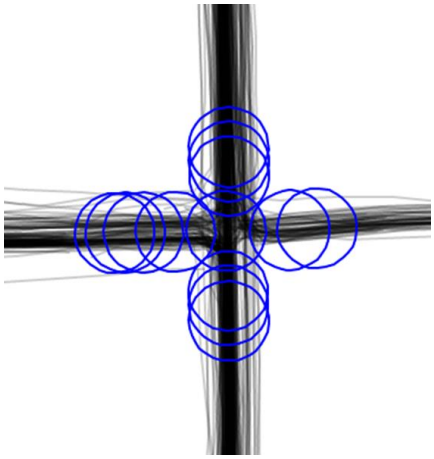


Figure 12. Multiple splits detected near the true intersection.

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 in areas in

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

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.

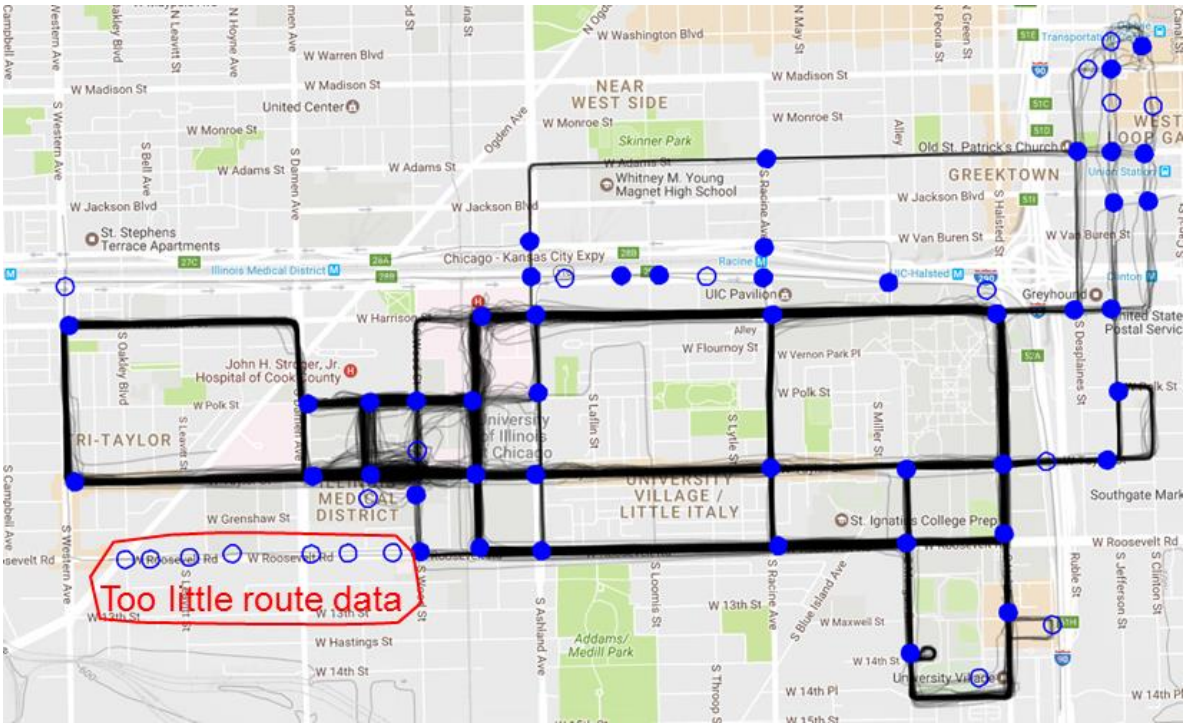


Figure 13. The intersections found in Chicago dataset. The filled circles represent correct detections (true positives) and empty circles represent incorrectly detected intersections (false positives).

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.

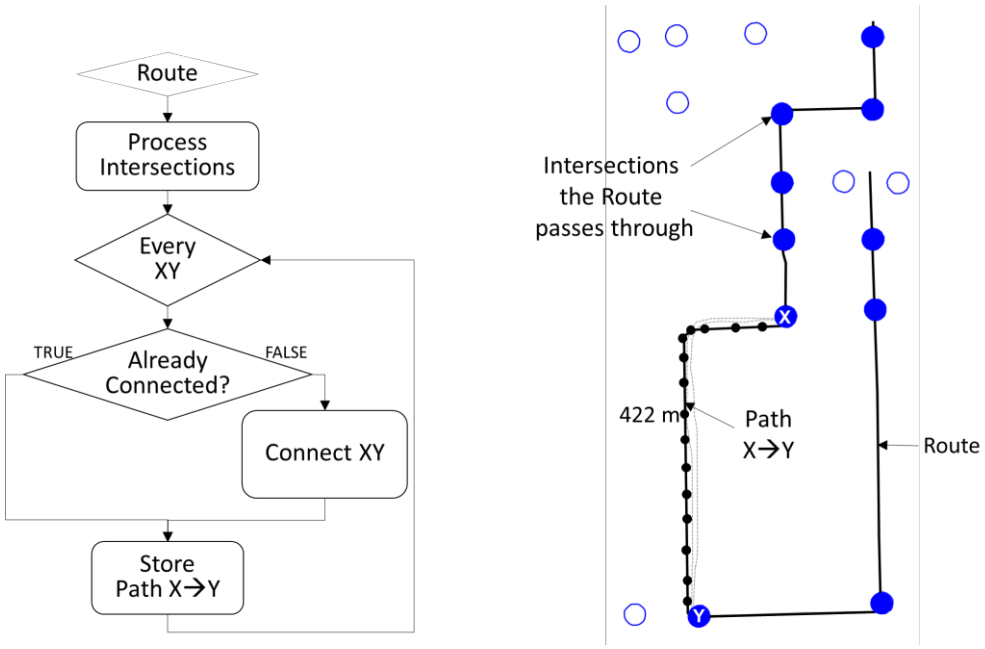


Figure 14. Left panel: Algorithm for linking the intersections. Right panel: Example of a route passing through several intersections. Connections are formed between pairs of intersections in the order that the route passes through. For every connection, all paths are stored.

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.

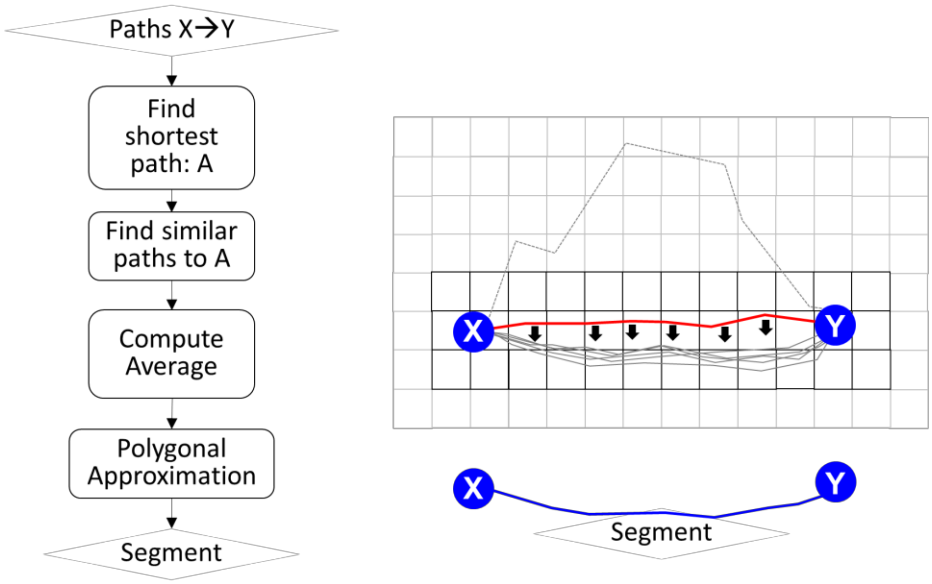


Figure 15. Left panel: Algorithm for creating a segment. Right panel: Example where the initial guess is optimized using the similar paths. The dilated cells used by the similarity function are highlighted using darker color.

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.

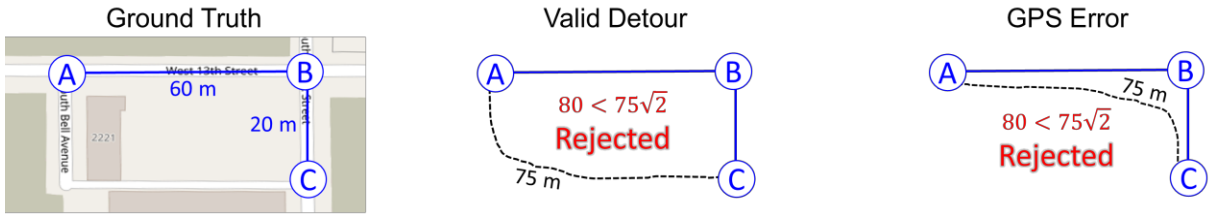


Fig. 16. Two examples where a segment is rejected according to the length rule. In the example on the left, the link should be kept because it represents a different road. On the right, the link should be removed because it is merely affected by GPS error.

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.

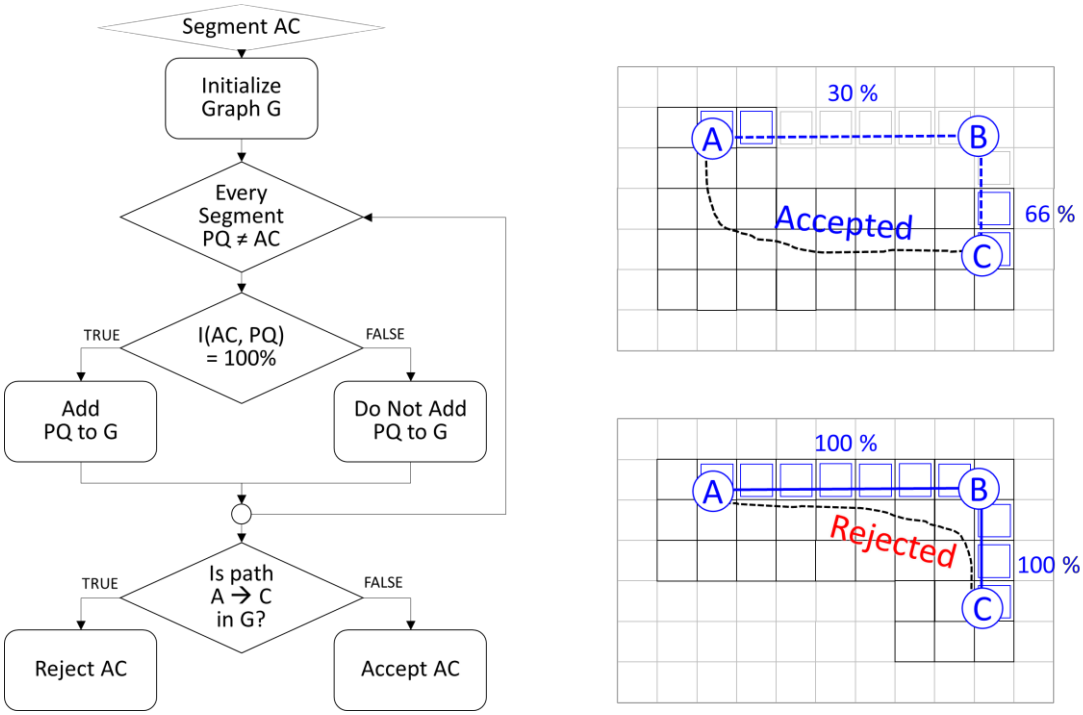


Fig. 17. Algorithm for filtering the segments (left panel). Examples where the segment is accepted (above, right) and rejected (bottom, right). The cell representations are shown. In the bottom right example, AB and BC are included in the region of AC and they form path A-B-C, which means the direct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

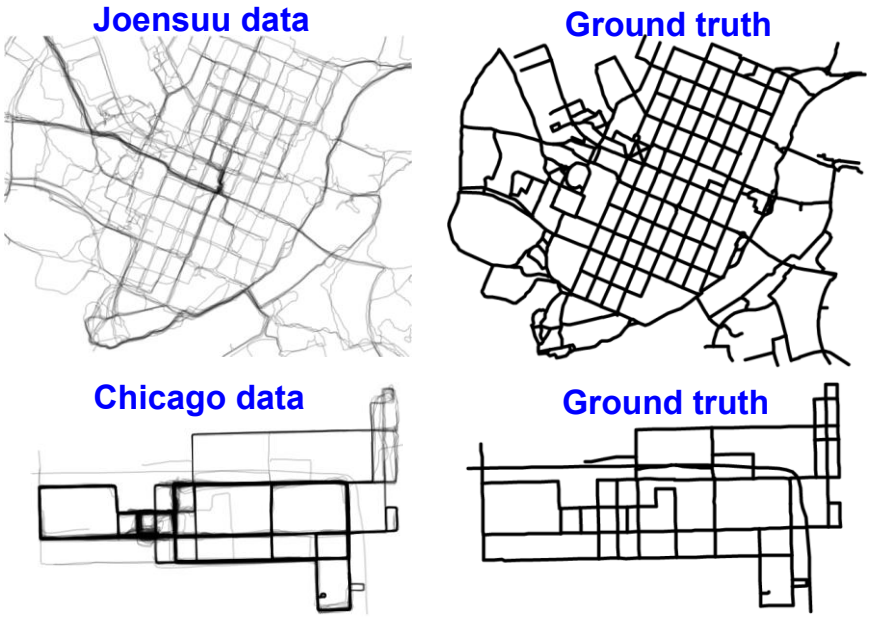


Figure 18. Joensuu and Chicago datasets, and the corresponding ground truth.

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:

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

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

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

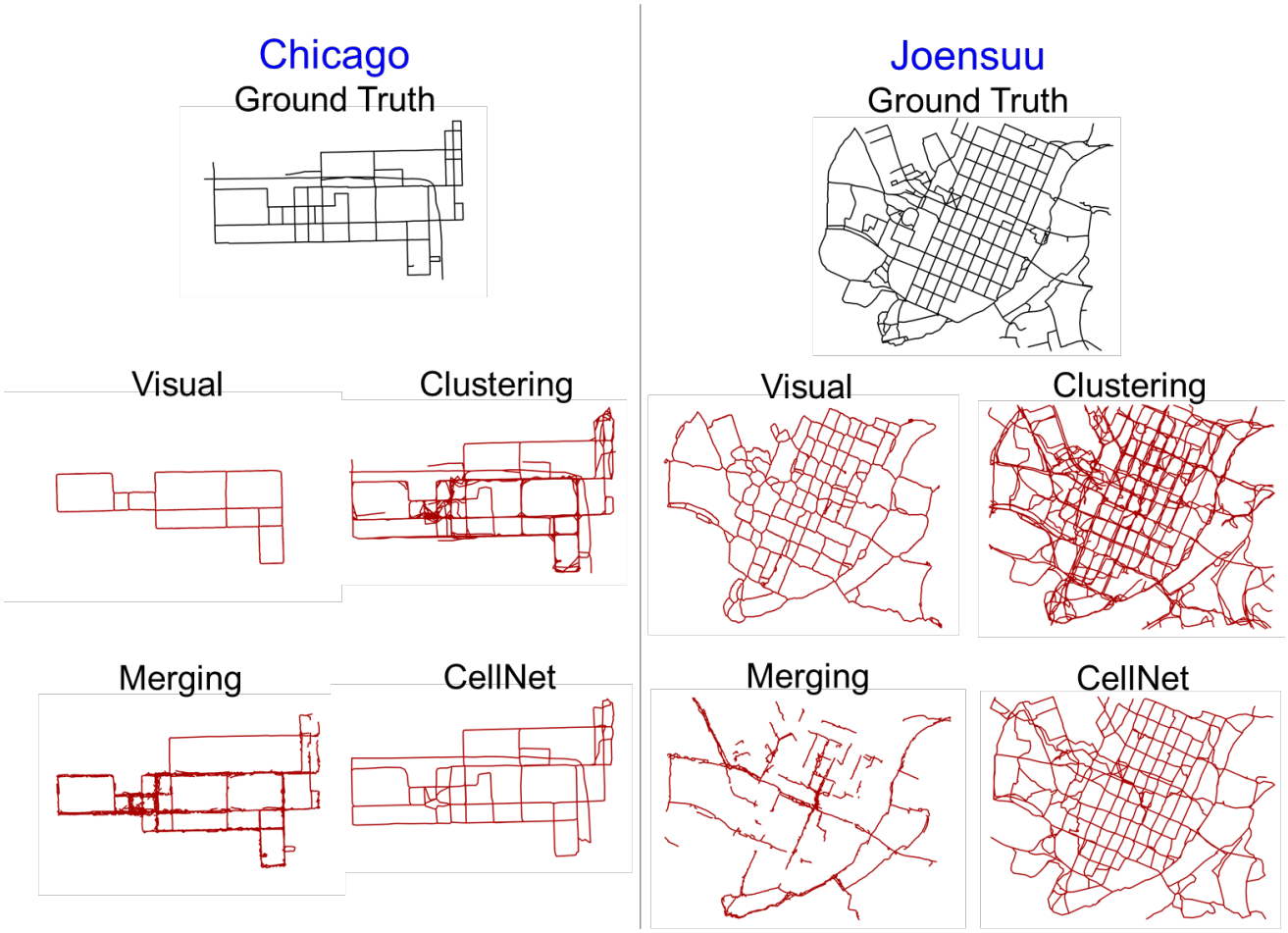


Figure 19. Visual output of the four methods for the Chicago and Joensuu datasets.

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.

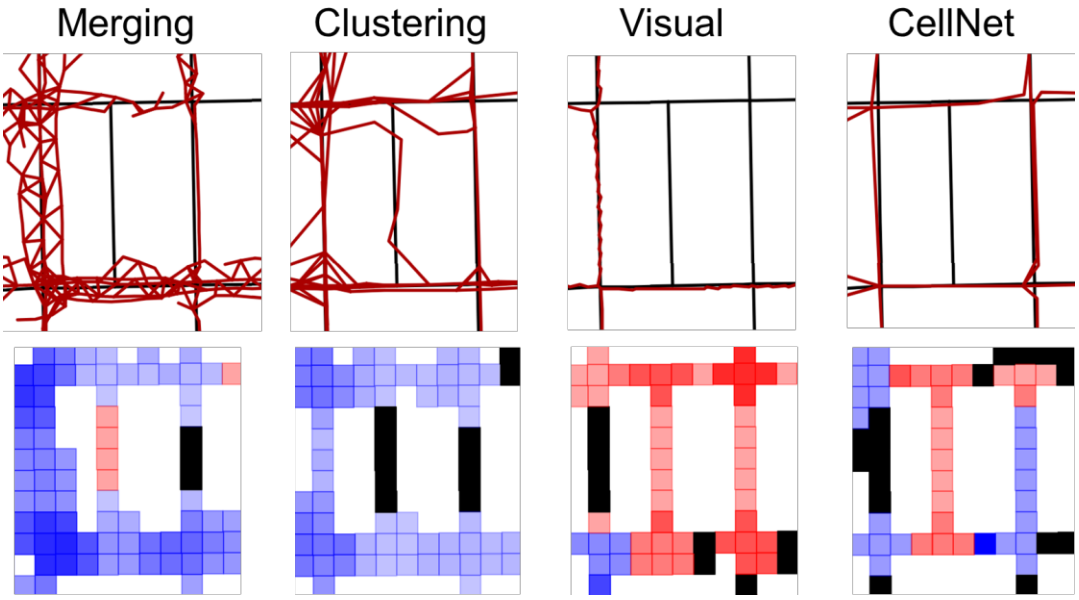


Fig. 20. Ground truth segments (black) and extracted segments (red) are shown at the top, and the corresponding cell frequency differences are shown at the bottom. Blue cells represent negative frequency (false detections), and red cells positive frequency (missed segments). Black cells have 0 frequency.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