Grid-Based Method for GPS Route Analysis for Retrieval

RADU MARIESCU-ISTODOR and 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 eight 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.

CCS Concepts: • **Information systems → Geographic information systems**; *Information retrieval*;

Additional Key Words and Phrases: GPS routes, grid, similarity, novelty, indexing, search

ACM Reference Format:
Radu Mariescu-Istodor and Pasi Fränti. 2017. Grid-Based Method for GPS Route Analysis for Retrieval. *ACM Trans. Spatial Algorithms Syst.* 3, 3, Article 8 (September 2017), 28 pages.
https://doi.org/10.1145/3125634

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*.

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

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

https://doi.org/10.1145/3125634

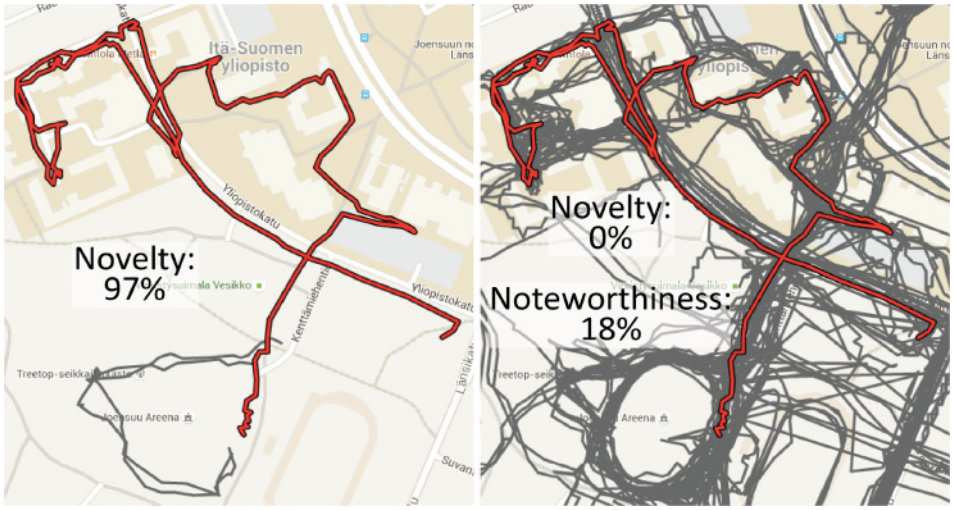

Fig. 1. Route novelty relative to the user himself (left) and relative to all users (right). The route is highly novel to the user, but not for to the rest of the database. Route noteworthiness is useful in the denser area.

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 driving. Stop points are also detected. In this article, we define four new route measures: *novelty, noteworthiness, similarity,* and *inclusion*.

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) and 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 a large density of routes so that novel regions rarely exist.

We define two routes as *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.

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. 2002a, 2002b) 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 speedup the proposed approach, we consider indexing. R-tree (Guttman 1984) has been used to improve efficiency of spatial queries in several route-searching problems (Yanagisawa et al. 2003; Frentzos et al. 2007a, 2007b; 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%

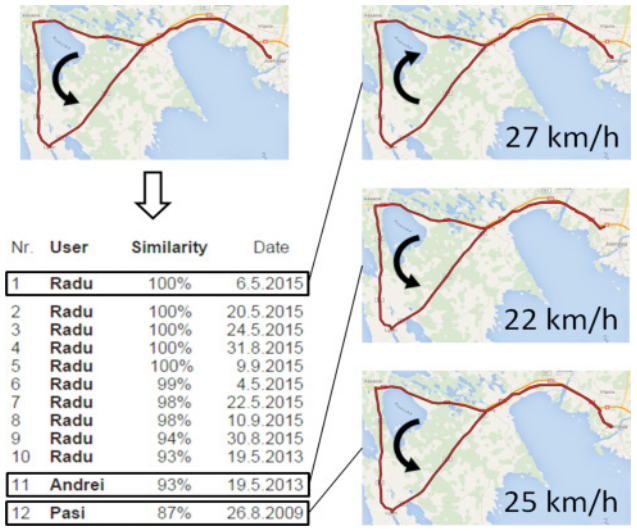

Fig. 2. Many similar routes for Joensuu – Liperi – Joensuu cycling track. We can see that several users have tried it with different outcome in terms of the average speed.

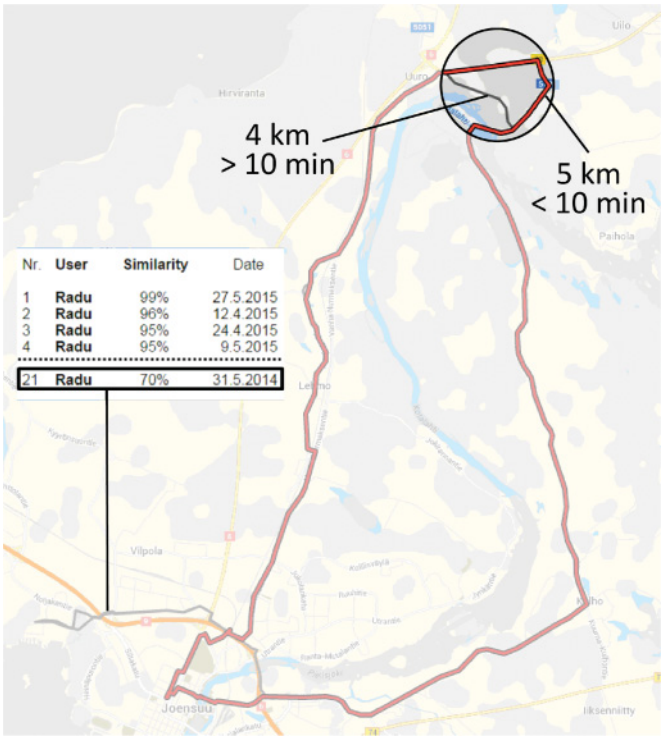

Fig. 3. Two 70% similar routes compared in Mopsi. One route takes a 1km shorter, off-road path. The other route is quicker, despite the extra kilometer.

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.

2 USING THE GRID

Grids have been used for representing geographical data in the past. In Pang et al. (2013) and 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) and 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 above-mentioned 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 purposes, 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. (2013), and 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.

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

MGRS is an alphanumeric two-dimensional coordinate system in which locations are identified independent of vertical position. Like **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 × 100km 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).

The remaining part consists of the numeric Easting and Northing values within the 100km square. MGRS allows 1 of 5 predefined precision levels when choosing the cell length: 1m, 10m, 100m, 1km, and 10km. 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 35VPK-1646-1774.

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

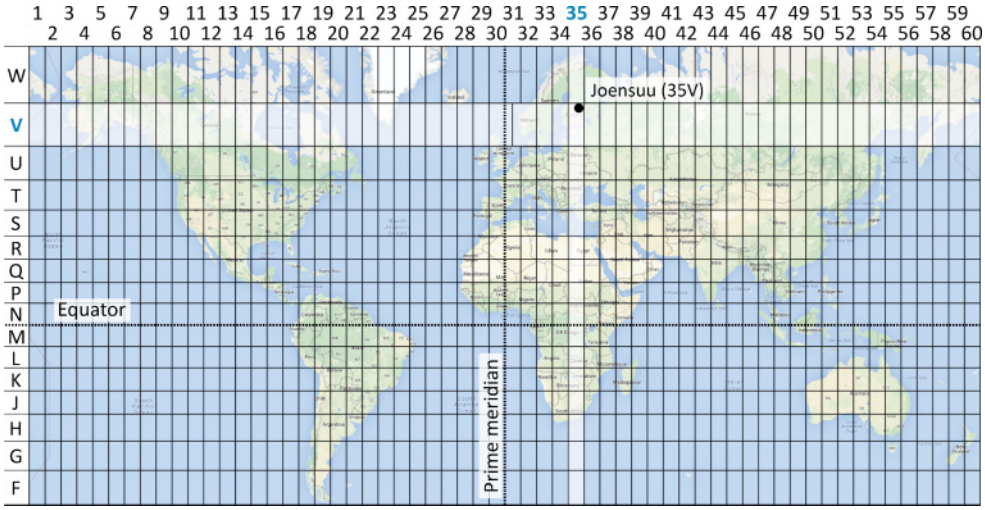

Fig. 4. UTM zones and latitude bands. Jonesuu is in MGRS grid zone 35 V.

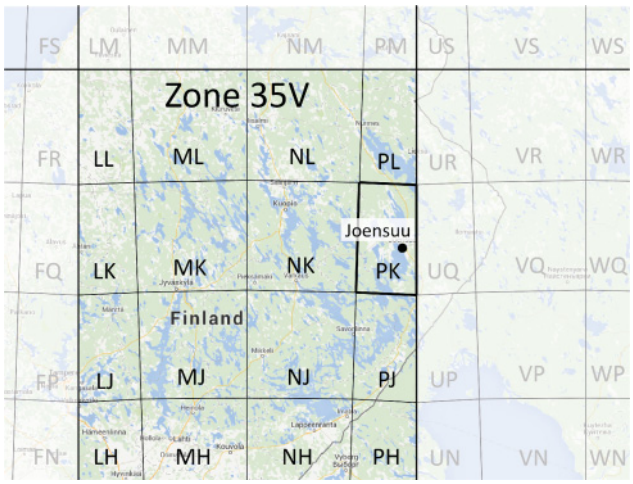

Fig. 5. 100 kilometer squares in zone 35 V. Joensuu is in square PK.

3 CELL REPRESENTATION

We define a point on the earth as $\mathbf{p} = (x,y)$ 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 × 100km 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 100km long and to reach the exact

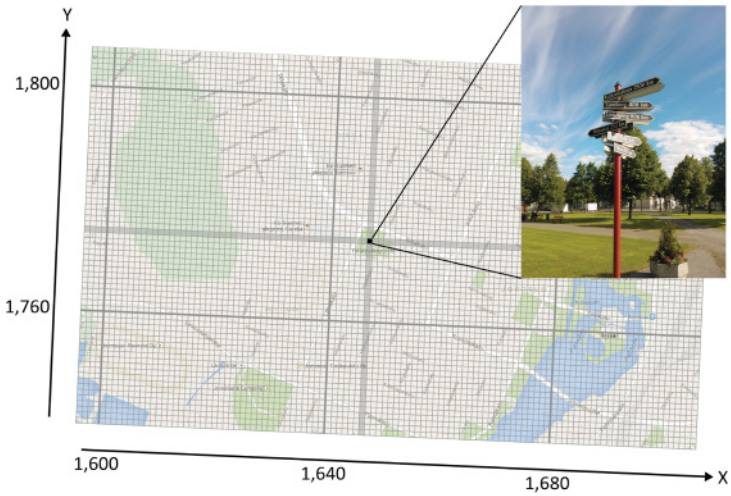

Fig. 6. 25 × 25 meter cells in Joensuu. The highlighted cell is in the center of a small park.

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 ← 4,000 × 4,000 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:

$$y = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) + y_1, \tag{1}$$

where (x_1,y_1) and (x_2,y_2) are the easting and northing values of the two cells inside a 100km 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 $O(\max(|x_2\text{-}x_1|,|y_2\text{-}y_1|))$ time.

It is possible that two consecutive cells do not lie inside the same 100km 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

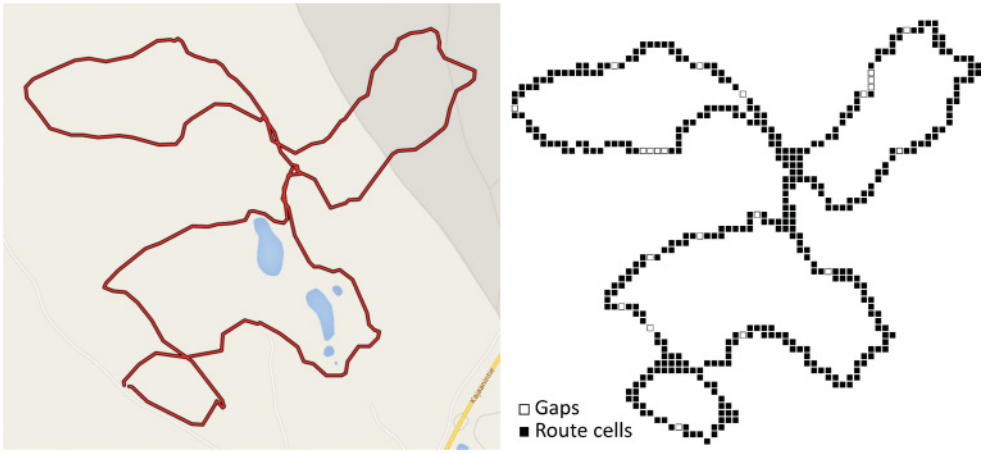

Fig. 7. A sample route (left) and the cell representation with cell size 25 × 25 meters (right).

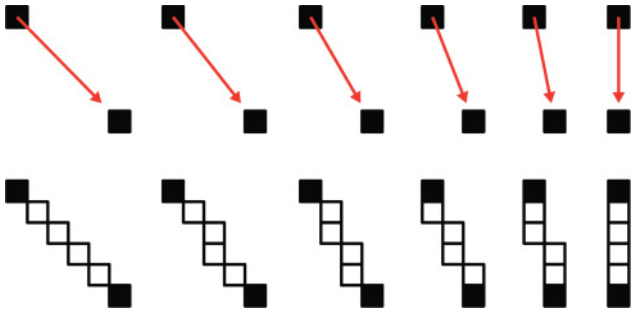

Fig. 8. Different examples where a four-cell gap is filled through interpolation.

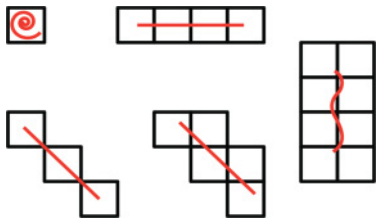

Fig. 9. Several 100-meter routes requiring between 1 and 8 cells of size 25 × 25 meters.

values, which we gradually increment and compute the MGRS mapping along the way. Equation (2) gives an estimate for the number of cells a route contains after interpolation:

$$cellCount\,(R, L) = \frac{length\,(R)}{L}. \tag{2}$$

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 the user is standing still but the 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).

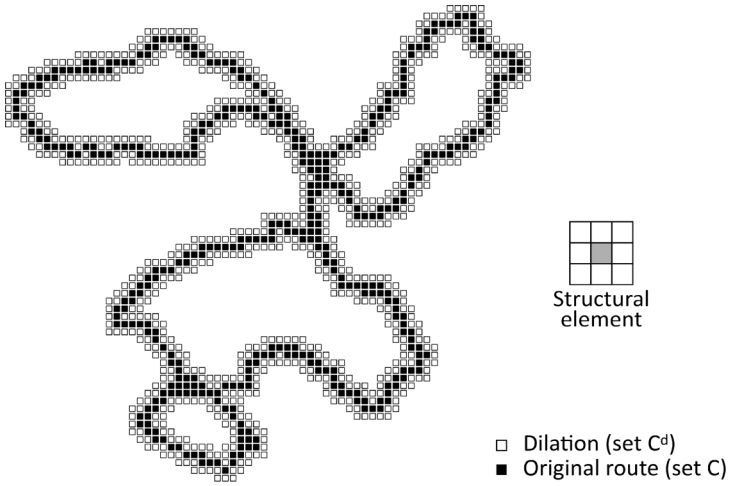

Fig. 10. Dilating the cell representation of a route with a square structural element.

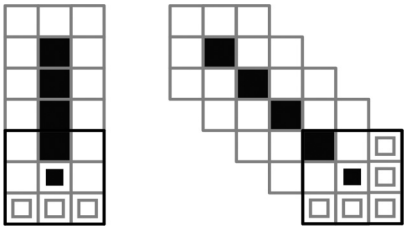

Fig. 11. Number of extra cells added from dilation depends on the direction of travel.

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.

When dilating a cell, the number of extra cells to be added depends on the direction of travel (see Figure 11). Moving diagonally adds five new cells while moving horizontally or vertically adds only three. The direction is with respect to the orientation of the MGRS 100km 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).

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.

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:

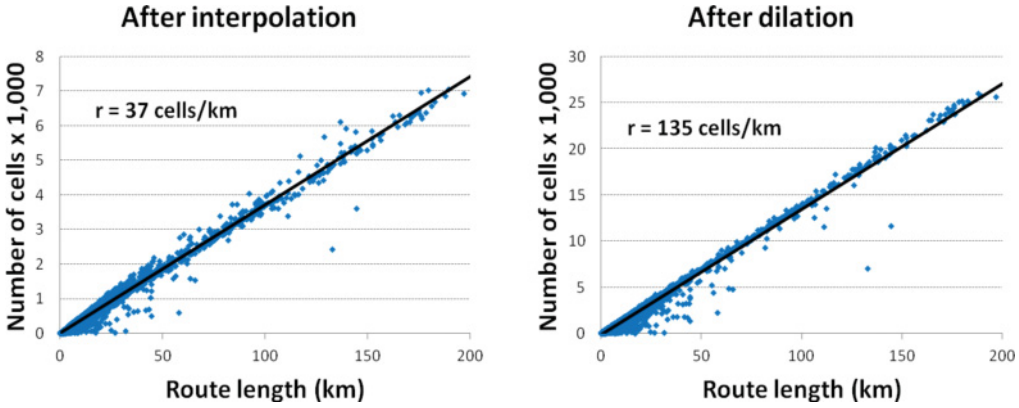

Fig. 12. Relationship between the length of the route and the number of cells when $L = 25$m.

Table 1. 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	$O(MQ\|S\|)$	$O(\log(MQ)\|S\|)$	$O(\|S\|)$

Route	Cell (MGRS)	Dilation
3812	35VPK-1649-1768	FALSE
3812	35VPK-1648-1768	FALSE
3812	35VPK-1647-1768	FALSE
3812	35VPK-1647-1768	FALSE
3812	35VPK-1646-1769	FALSE
3812	35VPK-1645-1771	FALSE
3812	35VPK-1644-1772	FALSE
3812	35VPK-1644-1773	TRUE
3812	35VPK-1644-1771	TRUE
6115	35VPK-4412-2117	FALSE
6115	35VPK-4412-2118	TRUE
6115	35VPK-4402-2118	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.

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

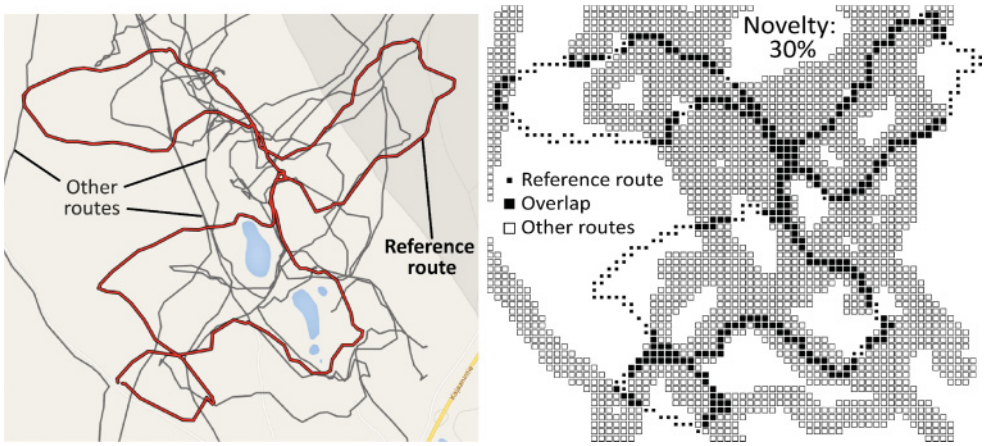

Fig. 13. The novel and overlapping cells of a route when compared against a route database. The dilated region is considered.

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 number 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.

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:

$$a(c) = \sum_{i=1}^{M} \left(|C \cap C_i| + \left| C \cap C_i{}^d \right| \right). \tag{3}$$

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 2. 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	$O(\|C\|MQ)$	$O(\|C\|(\log(MQ)) + a(C))$	$O(\|C\| + a(C))$

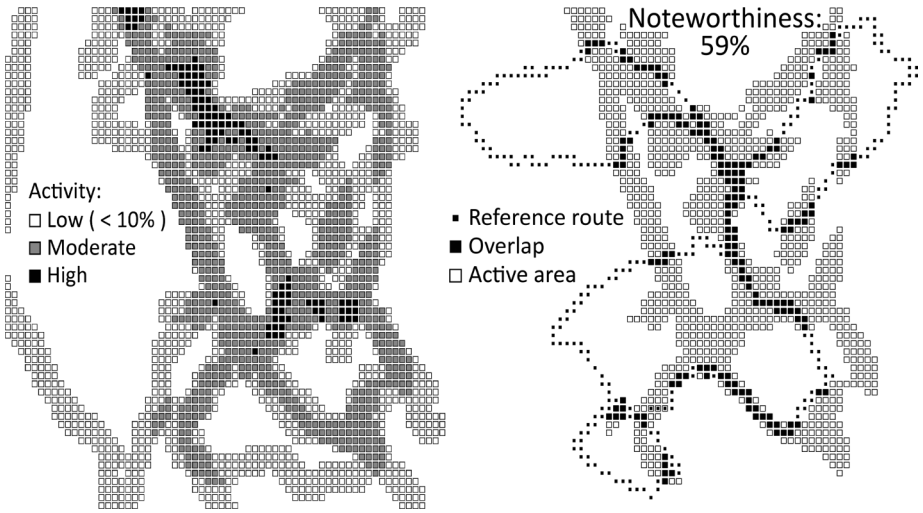

Fig. 14. The tracking activity in a region (left) and computing the noteworthiness of the route with respect to the active region (right). Parameter $p = 10\%$ is used in this example.

5.2 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 below 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)

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:

$$J(C_A, C_B) = \frac{|C_A \cap C_B|}{|C_A \cup C_B|}. \tag{4}$$

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:

$$S(C_A, C_B) = \frac{|(C_A \cap C_B) \cup (C_A \cap {C_B}^d) \cup (C_B \cap {C_A}^d)|}{|C_A \cup C_B|}. \tag{5}$$

Because $C_A \cap {C_A}^d = \{\}$ and $C_B \cap {C_A}^d = \{\}$ 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 (6) shows an alternative formula after these observations. Figure 15 illustrates the necessary components for computing the similarity.

$$S(C_A, C_B) = \frac{|C_A \cap C_B| + |C_A \cap {C_B}^d| + |C_B \cap {C_A}^d|}{|C_A| + |C_B| - |C_A \cap C_B|}. \tag{6}$$

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)

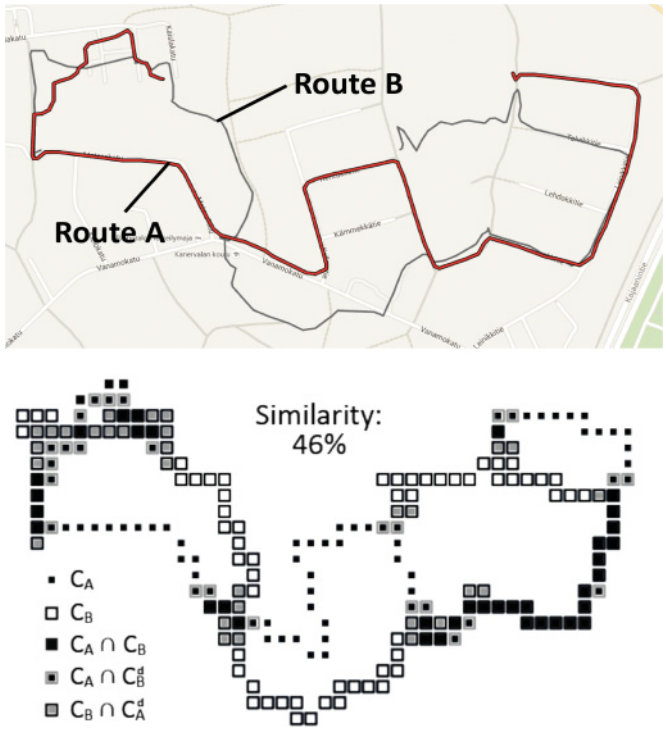

Fig. 15. Two routes that are 46% similar. Elementary sets for computing the similarity are depicted.

Intersection: Efficiently Computing the Intersection Between Two Sets of Cells.

Input: sets C_A and C_B
Output: intersection set X
$H \leftarrow$ 4,000 $\times$ 4,000 array of empty lists
$X \leftarrow$ new set
for each a **in** C_A **do**
    H [a $\leftarrow x$] [a $\leftarrow y$]. add (a$\leftarrow$ Square ID)
end
for each b **in** C_B **do**
    if H [b $\leftarrow x$] [b$\leftarrow y$]. contains (b $\leftarrow$ 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:

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

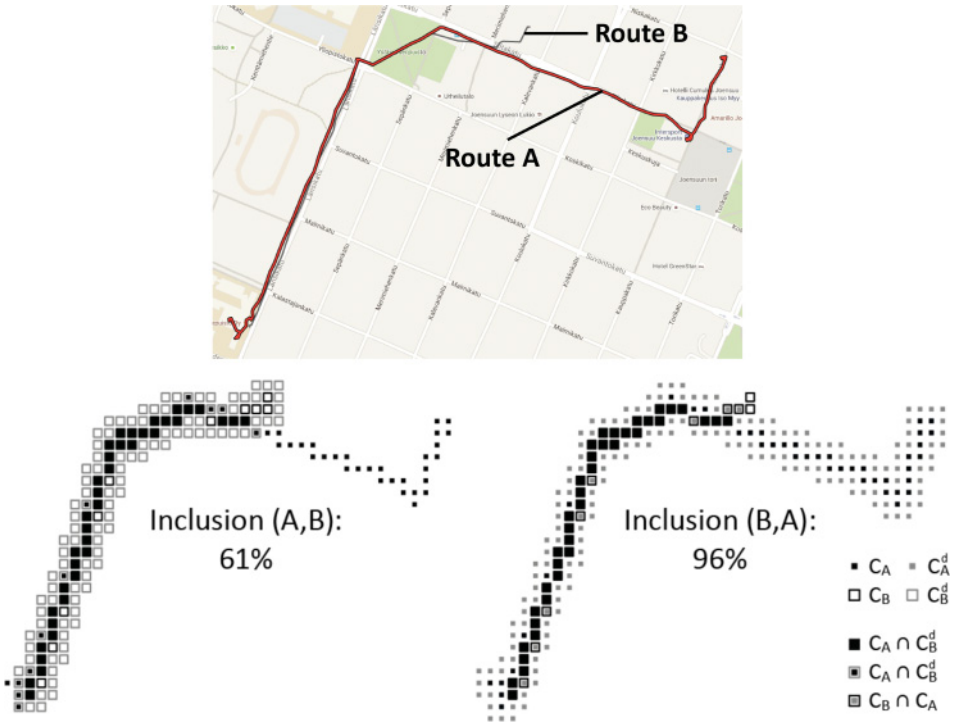

Fig. 16. Two routes A and B so that 61% of route A is included in B and 96% of route B is included in A.

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. A preliminary version of the inclusion measure has been presented in Mariescu-Istodor et al. (2014).

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 $O((|C| + |C^d|)MQ)$. 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 $O(MQ\log(MQ))$ 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

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

Table 3. Mopsi2014 Dataset Summary

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

Table 4. Database Requirements

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

more routes. Routes consist of wide range of activities including walking; cycling; hiking; jogging; orienteering; skiing; driving; and 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.

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 1GB 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 1.3GB of memory space.

Next, we perform a set of experiments using no index, B-tree and hash, respectively. All experiments were executed on Dell R920, 4 $\times$ 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 3,000 different routes. The novelty of each route is computed against the entire Mopsi2014 dataset. We focus only on routes with $|C| < 300$ cells (~8km) 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 the 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 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.

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: the first one has routes with less than 1,000 cells, the second 1,000–2,000 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 67ms and for hash is 43ms. Both methods are expected to work in real-time even for very large routes (300km). Hash index

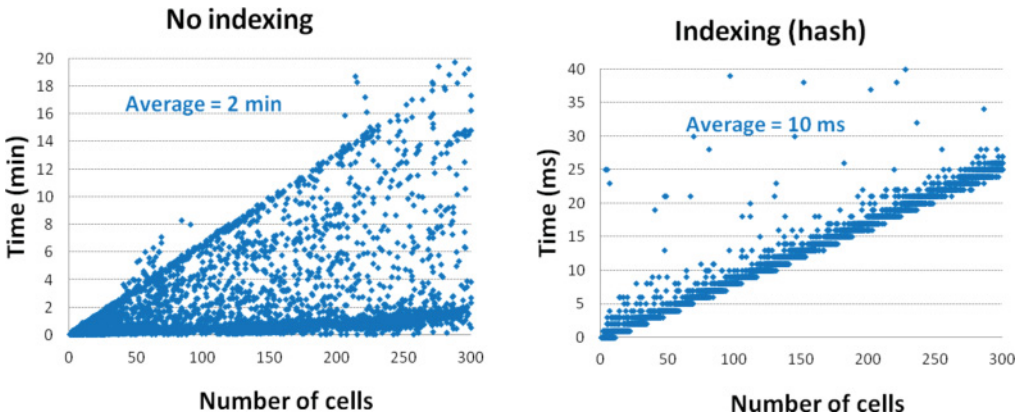

Fig. 17. Time required for calculating the novelty of 3,000 routes plotted as a function of the length of the route (in cells).

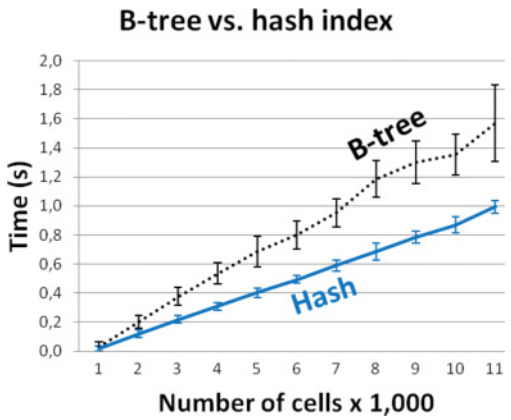

Fig. 18. Comparing B-tree and hash index by showing the average time and standard deviation for routes of different lengths (in cells).

requires about 50% of the time from that of the B-tree but requires about 40% more memory (see Table 4).

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.

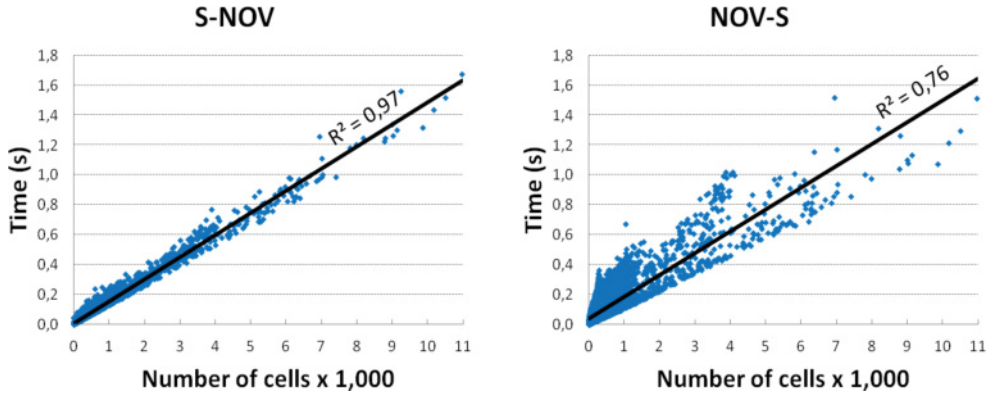

Fig. 19. Comparison of S-NOV and NOV-S running times when novelty is computed for every route in Mopsi2014 against a random subset S with |S| = 15 routes. Hash index is used.

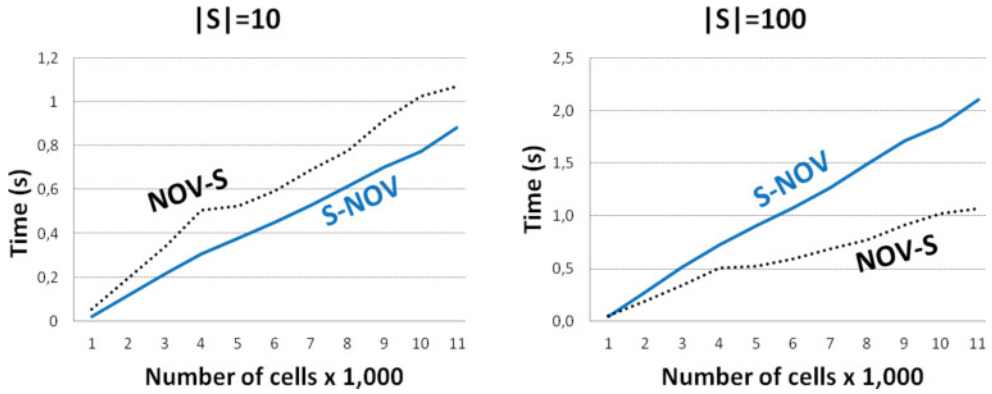

Fig. 20. Average time for both subset novelty methods when applied on subsets of sizes 10 and 100. The results are averaged for 50 random route subsets. Hash index is used.

For small subsets (|S| < 20 routes in case of Mopsi2014), S-NOV is faster than NOV-S. This is because datasets with routes widely 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.

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.

6.4 Route Similarity Algorithms Scalability Comparison

We compare C-SIM algorithm against the following eight route similarity algorithms: **L**ongest **C**ommon **Subs**equence (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

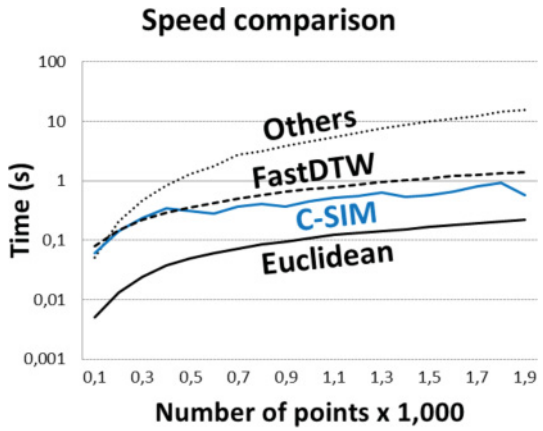

Fig. 21. Comparison of the nine route similarity measures in terms of speed.

less than 2,000 points in Mopsi2014 and divide them into 19 groups. The 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. The Hausdorff measure requires checking 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 1,158 points. For routes of this length, C-SIM takes 0.5s, which is about 10% of the time taken by the slower method; for instance. EDR takes 5.4s.

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. Wang et al. (2013) used 1,000 taxi routes from Zheng et al. (2009). We mimic

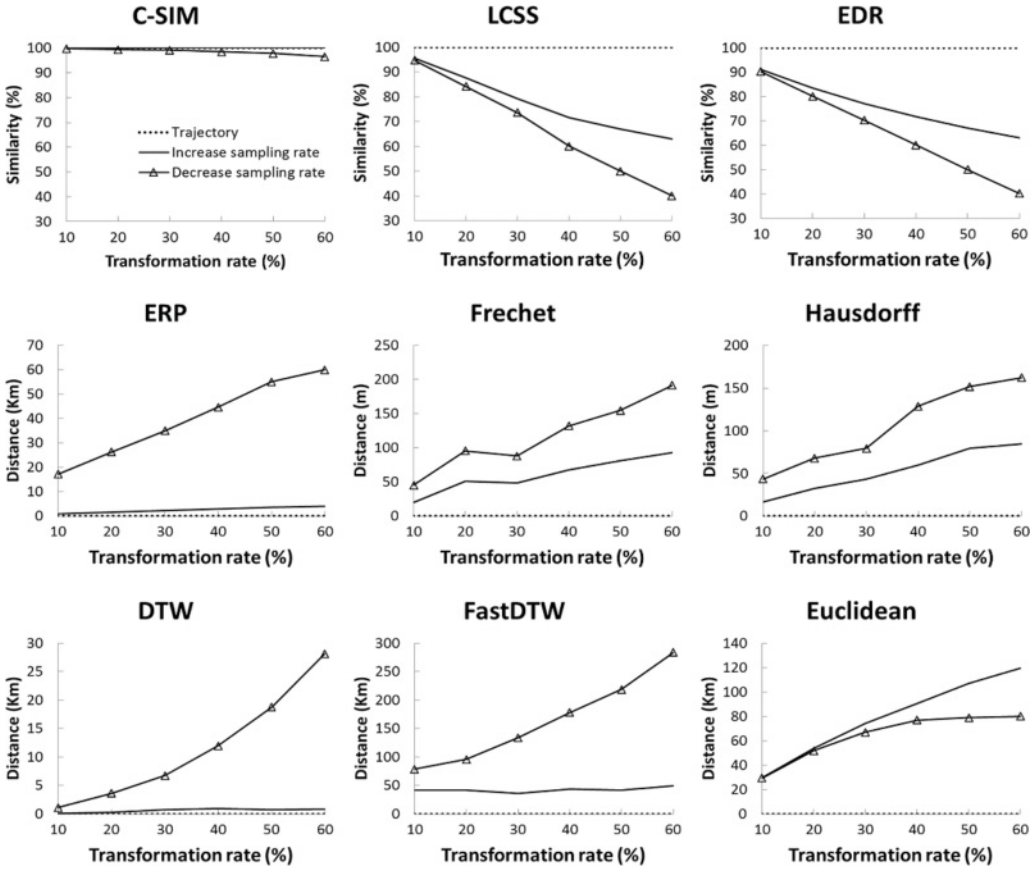

Fig. 22. Effect of changes in sampling rate on nine trajectory similarity measures.

their experiment by randomly selecting 1000 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. Specifically, 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

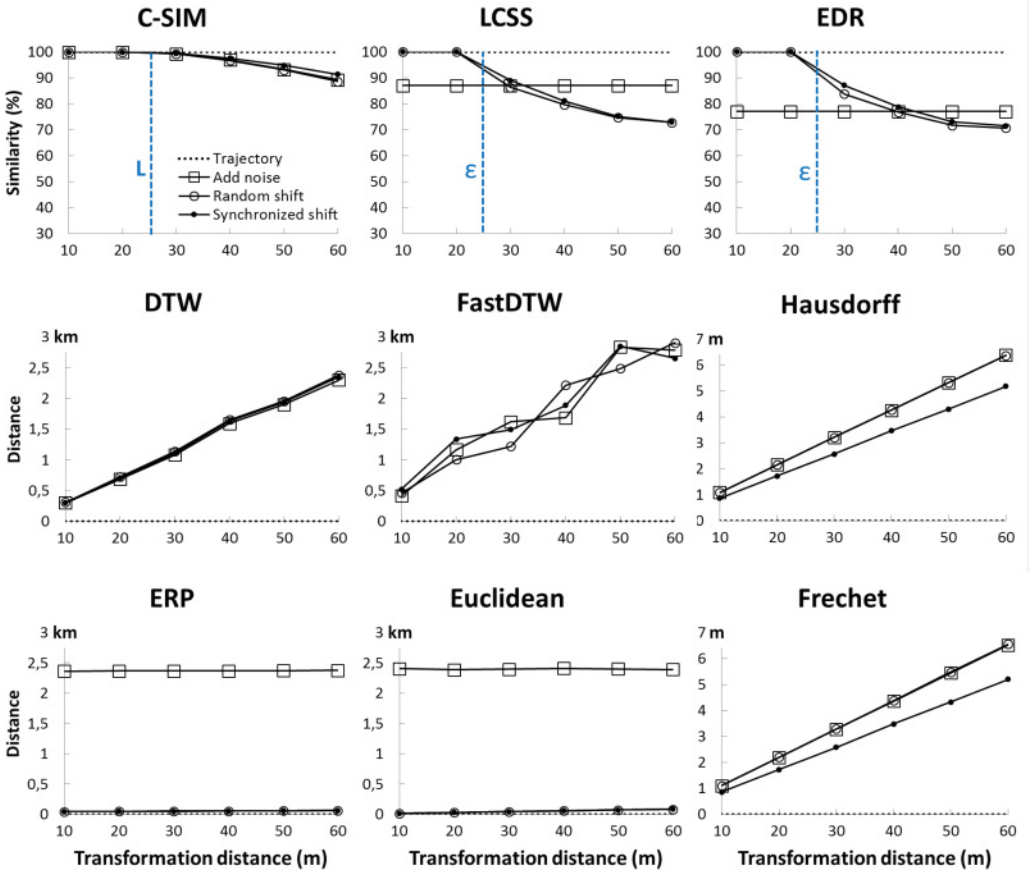

Fig. 23. Effect of nine different similarity measures when adding noise or shifting point locations as a function of transformation distance. In this experiment, 30% of the points are altered.

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.

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 ($L = \varepsilon = 25$ 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

Table 5. Summary of the Effectiveness of the Nine Route Similarity Measures

Definition	Sampling rate		Add noise	Point shifting	
	Increase	Decrease		Random	Sync.
$Grid(C_A, C_B) = \frac{\lvert C_A \cap C_B \rvert + \lvert C_A \cap C_B^d \rvert + \lvert C_B \cap C_A^d \rvert}{\lvert C_A \rvert + \lvert C_B \rvert - \lvert C_A \cap C_B \rvert}$	Robust	Robust	Fair	Fair	Fair
LCSS (A, B) $= \begin{cases} 0 & \text{, if } n = 0 \text{ or } m = 0 \\ 1 + \text{LCSS}(\text{Rest}(A), Rest(B)) & \text{, if } d(\text{Head}(A), \text{Head}(B)) \le \varepsilon \\ \max \begin{cases} \text{LCSS}(\text{Rest}(A), B) \\ \text{LCSS}(A, \text{Rest}(B)) \end{cases} & \text{, otherwise} \end{cases}$	Sensitive	Fair	Sensitive	Fair	Fair
EDR $(A, B) = \begin{cases} n & \text{, if } m = 0 \\ m & \text{, if } n = 0 \\ \min \begin{cases} \text{EDR}(\text{Rest}(A), \text{Rest}(B)) + \text{Subcost} \\ \text{EDR}(\text{Rest}(A), B) + 1 \\ \text{EDR}(A, \text{Rest}(B)) + 1 \end{cases} & \text{, otherwise} \end{cases}$ 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
DTW (A, B) $= \begin{cases} 0 & \text{, if } n = m = 0 \\ \infty & \text{, if } n = 0 \text{ or } m - o \\ d(\text{Head}(A), \text{Head}(B)) + \min \begin{cases} \text{DTW}(A, \text{Rest}(B)) \\ \text{DTW}(\text{Rest})(A), B) \\ \text{DTW}(\text{Rest}(A)), \text{Rest}(B)) \end{cases} & \text{, otherwise} \end{cases}$	Robust	Sensitive	Sensitive	Sensitive	Sensitive
FastDTW(A, B) = approximation of DTW(A, B)	Fair	Sensitive	Sensitive	Sensitive	Sensitive
ERP $(A, B) = \begin{cases} \sum_1^n \lvert a_i - g \rvert & \text{, if } m = 0 \\ \sum_1^m \lvert b_i - g \rvert & \text{, if } m = 0 \\ \max \begin{cases} \text{ERP}(\text{Rest}(A), \text{Rest}(B)) + d(a_1, b_1) \\ \text{ERP}(\text{Rest}(A), B) + d(a_1, g) \\ \text{ERP}(A, \text{Rest}(B)) + d(b_1, g) \end{cases} & \text{, otherwise} \end{cases}$	Fair	Sensitive	Sensitive	Robust	Robust
Eulidean$(A, B) = \sqrt{\sum_{i=1}^{\min(n,m)} d(a_i, b_i)}^2$	Sensitive	Sensitive	Sensitive	Robust	Robust
Hausdorff$(A, B) = \max \begin{cases} d(A, B) \\ d(B, A) \end{cases}$ $d(A, B) = \sup_{a \in A} \inf_{b \in B} d(a, d)$	Sensitive	Sensitive	Sensitive	Sensitive	Fair
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 $[0, 1] \to [0, 1]$.	Sensitive	Sensitive	Sensitive	Sensitive	Fair

and FastDTW are sensitive to all transformations. ERP and Euclidean measures are highly sensitive to noise but they are robust for the 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.

6.6 Effect of Indexing on RSR Algorithm

We demonstrate the efficiency of indexing by computing the similarity ranking using RSR algorithm for 1,500 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.

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

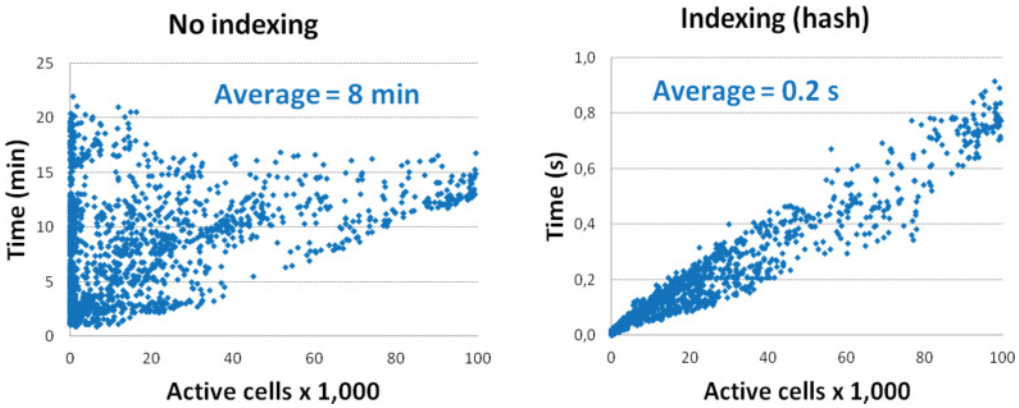

Fig. 24. The time required for calculating the route similarity ranking for 1500 routes plotted as a function of the amount of active cells.

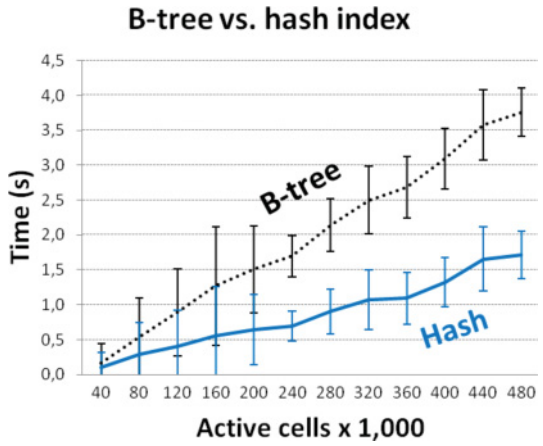

Fig. 25. Comparing B-tree and hash index efficiency for performing the route similarity ranking by showing the average time and standard deviation for routes with different amounts of active cells.

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).

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 the 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)*. 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)*. 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)*. 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)*. 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)*. 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)*. 792–803.

Minjie Chen, Mantao Xu, and Pasi Fränti. 2012. A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Trans. Image Process.* 21, 5, 2770–2785.

Thomas H. Cormen. 2009. *Introduction to Algorithms*. MIT Press.

Thomas Eiter and Heikki Mannila. 1994. *Computing Discrete Fréchet Distance*. Tech. Rep. 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)*. 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)*. 56–69.

Elias Frentzos, Kostas Gratsias, Nikos Pelekis, and Yannis Theodoridis. 2007a. Algorithms for nearest neighbor search on moving object trajectories. *Int. J. Adv. Comput. Sci. Geograph. Inf. Syst. (Geoinformatica)* 11, 2, 159–193.

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

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

Ralf Hartmut Güting, Thomas Behr, and Jianqiu Xu. 2010. Efficient *k*-nearest neighbor search on moving object trajectories. *Int. J. 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)*. 47–57.

James D. Hamilton. 1994. *Time Series Analysis (Vol. 2)*. Princeton University Press, Princeton, NJ.

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

Radu Mariescu-Istodor, Andrei Tabarcea, Rahim Saeidim, 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)*. 62–69.

Linsey Xiaolin Pang, Sanjay Chawla, Wei Liu, and Yu Zheng. 2013. On detection of emerging anomalous traffic patterns using GPS data. *Data Knowl. Eng. (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)* (Seattle, WA), 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.

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

Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002b. 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, PA), 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.

Haibo Wang and Kuien Liu. 2012. User oriented trajectory similarity search. In *Proceedings of the ACM SIGKDD International Workshop on Urban Computing (UrbComp'12)*. 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)*. 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)*. 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)*. 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)*. 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)*. 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)*. 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)*. 1029–1038.

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.

Yu Zheng and Xiaofang Zhou. 2011. *Computing with Spatial Trajectories*. Springer Science & Business Media.

Received September 2016; revised April 2017; accepted July 2017