Similarity Search based on Geo-footprints

TBA
Department of Computer Science and Engineering
University of Ioannina
Ioannina, Greece
TBA@cs.uoi.gr

ABSTRACT

Many applications track the movements of mobile users, especially in controlled (e.g., indoor) environments. The ultimate goal is to combine this information with other data (e.g., user profile) and use it for effective promotion marketing purposes or item recommendation. In this paper, we define the concept of geo-footprint, a concise representation of the visits by mobile users in supervised indoor spaces, which summarizes the potential interest of users in nearby points of interest. Then, we define similarity measures between users based on their footprints, extending popular models from Information Retrieval. Finally, we propose and evaluate similarity search algorithms which can be used as modules in recommender systems or data mining tasks (e.g., clustering).

ACM Reference Format:
TBA. 2019. Similarity Search based on Geo-footprints. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

There has been a lot of research and technology on managing the locations and movement of mobile objects. Most previous work focuses on handling (outdoor) user trajectories [17]. This includes trajectory data analytics systems [6, 14], trajectory cleaning and transformation [4], trajectory similarity measures [16], trajectory retrieval [13, 19], and trajectory clustering [18] and classification [2]. Targeted applications include traffic monitoring and analysis [20], navigation [7], trip recommendation [12]. Location-based services for indoor spaces are gaining popularity [15], as indoor location tracking can be achieved with good accuracy, either with the help of wireless technology (e.g., WiFi, RFID, UWB, BLT devices, etc.) [21], computer vision [9], or by using the smartphone's inertial sensors [10]. Users typically give consent to applications to track their locations, provided that their data will solely be used by the applications, they will not be shared with others, and they will be deleted after a certain period of time **TODO: read regulations, provide details**.

This work focuses on the capturing and use, for each mobile user, of the locations have been of *interest* to the user, such as a visit to the TVs section of a department store. We call this information *geo-footprints*. Instead of storing the entire history of user locations (i.e., the complete trajectories), we focus on the location information, which implies the interests or habits of users. Hence, the footprint of a user is a set of spatial regions, which characterize the behavior of a user, within a given context (e.g., in a department store).

Example 1. *The manager of department store Acme is interested in tracking, for each registered customer and for each visit of the customer to Acme, the areas within the store where the customer spent at least 5 minutes. These regions may relate the customer to products or product categories that are exhibited/promoted near the area at that particular period.*

By comparing the geo-footprints of users, we can infer whether two users have *similar behavior or habits*. Our goal is to define a similarity measure, which considers the spatial overlap of their footprints.

Example 2. *Bob visited Acme twice this month. During his first visit, he stayed for 30 minutes at the TV exhibition area and then for 45 minutes at the children books area. At his second visit, he stayed for about 30 minutes at the TV exhibition area. Alice visited Acme just once and she was at the TV area for 45 minutes. The footprints of Bob and Alice have a non-zero similarity ++* ***TODO: refine after definitions***

Computing the geo-footprints of users and their similarity finds use in market analysis applications and recommender systems. For instance, *customer segmentation* divides the customers of a company into clusters based on the similarity of their features (characteristics). Geo-footprints can be used together with other features (e.g., age, transaction records) to define clusters based on multivariate information. Similarly, in recommender systems or advertisement, apart from other features, footprints can be used to characterize users. This is especially useful in situations where there is insufficient information about other feature types of the target user (i.e., cold-start users). Products that are bought by users with similar footprints as the target user can be promoted to her/him. Another application is link recommendation in geo-social networks. The profiles of users of such networks include their location visits and their frequencies, which can be modeled as geo-footprints. Users with similar footprints have high probability to be socially linked, as they are expected to have similar interests.

Our contributions can be summarized as follows:

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- We define mobile user geo-footprints and show their application in market-analysis and recommendation applications.
- We propose a measure for the similarity between geo-footprints, which naturally extends document similarity measures from information retrieval. The challenge is to define a measure that considers the overlap and the weights of the locations in the compared footprints.
- We propose algorithms for efficient footprint-based similarity computation, which build upon plane-sweep and spatial join techniques. We investigate the indexing of geo-footprints for efficient similarity search.
- We experimentally evaluate the effectiveness of geo-footprints and the efficiency of similarity computation and search.

The rest of the paper is organized as follows. Section 2 presents related work.++

Finally, Section 8 concludes the paper.

2 RELATED WORK

Here goes related work

The problem of finding hot spots of moving vehicles, which finds application in traffic management and transportation analysis is studied in [11]. Sample objects are used as "sensors" that track the density of vehicles around them, based on their speed (e.g., high speed indicates light traffic). Time-parameterized hot spots and regions are identified.

++ examine previous work on the problem, when considering temporal information only.

3 FOOTPRINT EXTRACTION

This section presents an approach for extracting geo-footprints from user trajectories. We consider the scenario that the locations of users (e.g., customers in a department store) are tracked automatically and regularly and converted to trajectories, linked to the user identifiers. Formally:

Definition 1 (user trajectory). *A user trajectory T of length T is a sequence of locations $\{l_1, l_2, \ldots, l_{|T|}\}$, such that each location $l_i = \langle l.p, l.t\rangle$ is characterized by a spatial position p (i.e., a pair of x-y coordinates) and a timestamp t. For regularly tracked locations, $l_{i+1}.t - l_i.t$ equals a fixed time difference Δt, for each $i \in [1, |T|)$.*

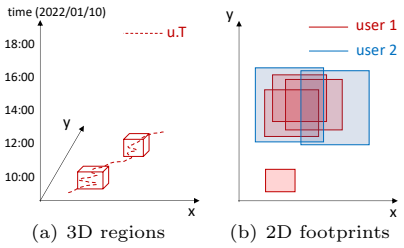
For the same user u, we typically have a sequence of trajectories $u.\mathcal{T} = \{u.T_1, u.T_2, \ldots u.T_{\mathcal{T}}\}$, where for each $i \in [1, |\mathcal{T}|)$, $u.T_i.l_{|T_i|}.t < u.T_{i+1}.l_1.t$, i.e., the trajectories are *temporally disjoint*. We may also refer to a trajectory in $u.\mathcal{T}$ as a *session* of user u. For example, a trajectory/session in $u.\mathcal{T}$ corresponds to the continuous time period during which customer u visited a store and was monitored by the store.

3.1 Regions of interest

Instead of the entire trajectories, we are interested in sub-trajectories, where the user is relatively immobile (e.g., the customer wanders around or stands near specific exhibition items). Two or more consecutive locations in a user trajectory belong to the same *region of interest*, if they are spatially close to each other. Formally:

Definition 2 (region of interest (RoI)). *A region of interest for a given user u is defined by the 3D minimum bounding box (MBB), defined by the set of consecutive locations $R = \{l_s, l_{s+1}, \ldots, l_e\}$ in a trajectory $u.T$ of $u.\mathcal{T}$, where $1 \le s < e \le |T|$, such that (i) $|l_i.p - l_j.p| \le \epsilon$ for each $i, j \in [s, e]$ and (ii) $e - s > \tau$.*

Parameter ϵ is a spatial extent constraint and τ is the minimum duration of a subtrajectory to qualify for a RoI. Quantity $|l_i.p - l_j.p|$ denotes the spatial distance between locations $l_i.p$ and $l_j.p$; typically an L_p norm is used to measure distance, like Euclidean distance (L_2). In our implementation, we use L_∞, i.e., we require that the distance between two locations at each dimension does not exceed ϵ. Intuitively, a RoI R includes all consecutive user locations in a trajectory and their timestamps, the extent of R does not exceed ϵ, and R includes a large enough number of locations, corresponding to a large enough time interval. For example, for a customer in a store who spends a long time near a particular category of items, we can infer that the spatial region around the items characterizes the interest of the customer. Figure 1(a) shows two regions of interest that can be extracted from a user trajectory $u.T$. For simplicity, we use the same symbol R to denote the sequence of locations that define a region and their MBB.



(a) 3D regions (b) 2D footprints

Figure 1: Extracted regions and user footprints

According to Def. 2, two regions of interest R_1 and R_2 in the same trajectory can temporally overlap. For example, regions $R_1 = \{l_s, l_{s+1}, \ldots, l_{e_1}\}$ and $R_2 = \{l_s, l_{s+1}, \ldots, l_{e_2}\}$, where $e_2 = e_1 + 1$ may both be valid according to Definition 2. Hence, a huge number of regions could be extracted from the same trajectory. On the other hand, we would like to limit the number of regions that characterize a user, in order to make their representations concise and manageable, and to simplify the comparison between users. One step in this direction is to focus on maximal regions of interest only, according to the following definition.

Definition 3 (maximal region of interest). *A region of interest $R = \{l_s, l_{s+1}, \ldots, l_e\}$ is maximal if $\{l_{s-1}, l_s, \ldots, l_e\}$ and $\{l_s, l_{s+1}, \ldots, l_{e+1}\}$ are not regions of interest, according to Def. 2.*

Still, even the maximal regions could be too many; for example, both $\{l_s, l_{s+1}, \ldots, l_e\}$ and $\{l_{s+1}, l_{s+2}, \ldots, l_{e+1}\}$ can be maximal. Hence, when constructing the footprint of a user, we limit ourselves to extracting a set of regions which are *temporally disjoint*. This can be done by a greedy heuristic, which starts by verifying the conditions of Definition 2 on the first τ locations of the examined trajectory T (i.e., $s = 1, e = \tau$). If the first τ locations form then we keep s constant and expand e until the maximality condition is violated and finalize the extracted maximal region. The next candidate to start searching for a maximal region is $[e+1, e+\tau+1]$. If the first τ locations do not form a region of interest, we repeat with $s = 2, e = \tau + 1$, and so on, until the condition is satisfied for a given s and e. Algorithm 1 is a pseudocode of this heuristic, which also includes an optimization in order to avoid some redundant checks. Starting from the first location in the input trajectory T, the algorithm adds to the current region R the next location l_i, until condition ϵ is violated. When this happens, if R has enough locations, it is added to the profile of the user corresponding to trajectory R and a new R is initialized to hold l_i (Lines 6-8). If R does not have enough locations, then a new region $newR$ is initialized with just l_i; then, we add to $newR$ as many points from R as possible starting from the last point of R and going backwards. This guarantees that the maximal region which includes l_i will not be missed, while avoiding redundant operations.

Algorithm 1 Regions of interest extraction

Require: trajectory T; bounds ϵ, τ
1: $R \leftarrow$ null ▷ Current region
2: **for** each $l_i \in T$ **do**
3: **if** $R \cup l_i.p$ does not violate ϵ **then**
4: add l_i to R
5: **else**
6: **if** $|R| \geq \tau$ **then** ▷ Current region has enough points
7: add R to user profile
8: $R \leftarrow \{l_i\}$ ▷ Initialize current region
9: **else**
10: $newR \leftarrow \{l_i\}$ ▷ Initialize new region
11: **while** $newR \cup R.last$ does not violate ϵ **do**
12: $newR \cup R.last$; delete $R.last$;
13: **end while**
14: $R \leftarrow newR$ ▷ Initialize current region
15: **end if**
16: **end if**
17: **end for**
18: **if** $|R| \geq \tau$ **then** ▷ Last region has enough points
19: add R to user profile
20: **end if**

Since the above algorithm is of a greedy nature, there is no guarantee that we will find the set of maximal RoIs that represents best the behavior of the user. On the other hand, the algorithm is very fast, since it requires just a linear scan over the trajectory data. In addition, it is appropriate for real-time processing of streaming location data, as in this case the regions of interest should be determined fast without dependencies with future data. At the same time, we observe that in real applications there is no need for accurate detection of the regions and that in practice each region is well separated from others, especially when domain knowledge is used to define constraints ϵ and τ (e.g., in stores where movement is controlled and item categories or exhibitions are well separated).

By running Algorithm 1 for all trajectories T of a user u, we can extract all RoIs of the user. Note that a user may visit the same spatial region or overlapping spatial regions at different times. That is, two RoIs cannot overlap in time, but they may overlap in space. If the user has been at the same area multiple times (e.g., the TV exhibition area of a store), then the weight of that are should be higher in the user's profile. Based on this, we define the *geo-footprint* $F(u)$ of a user u as the collection of all RoIs of u, *disregarding their temporal dimension.* In other words, we consider the RoIs in a user footprint to be 2D MBRs, i.e., the 2D projections of the original 3D RoIs. For example, in recommender systems for department stores, the time when a customer visited an area in the store may not be important. Examples of such user footprints are shown in 1(b). In the rest of the paper, we will refer to the 2D projections of the maximal RoIs of a user as the geo-footprint of a user.

4 SIMILARITY BETWEEN FOOTPRINTS

We are now ready to define the similarity between users based on their geo-footprints. We follow an information retrieval approach, and derive the preference of a user u to a location l by the number of times l is in the RoIs of u. This is similar to the relevance of a text document to a term defined by the number of times the term appears in the document. However, since the spatio-temporal domain is continuous and infinite, as opposed to the domain of possible terms, we use the set of RoIs (instead of individual locations) in the user footprints to define and measure similarity.

Specifically, the footprint of a user can be modeled as a set of *disjoint* spatial regions and their frequencies. Figure 2(a) shows an example of a footprint with three RoIs (r_1, r_2, r_3). The regions divide the space into nine disjoint regions (A to I), such that the union of the disjoint regions is the space covered by all three RoIs. For each of the disjoint regions, Figure 2(a) shows in parentheses its frequency, i.e., the number of times it is included in the RoIs of the footprint. Hence, the footprint $F(r)$ of a user r can be defined as a set of (X, f_X) pairs, where X is a continuous spatial region (not necessarily rectangular) and f_X is the frequency of X; if regions X and Y are in the same footprint $F(u)$, then X and Y should be spatially disjoint.

We are now ready to define the similarity between two users r and s, based on their footprints $F(r)$ and $F(s)$. Inspired by the popular cosine similarity in IR, which measures the similarity between documents modeled as term frequency vectors, we define the similarity between two footprints as

follows:

$$sim(F(r), F(s)) = \frac{\sum_{(X,f_X)\in F(r),(Y,f_Y)\in F(s)} |X \cap Y| \cdot f_X \cdot f_Y}{||F(r)|| \cdot ||F(s)||} \quad (1)$$

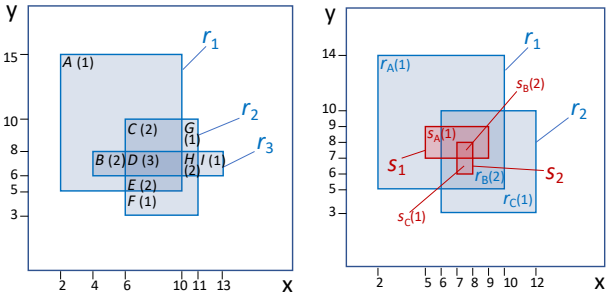
As in cosine similarity, the numerator in Eq. 1 aggregates the common locations in the footprints $F(r)$ and $F(s)$ and multiplies them with their frequencies in the footprints. That is, for each pair (X, Y) of regions that intersect and X is in $F(r)$, Y is in $F(s)$, their area $|X \cap Y|$ of their intersection $X \cap Y$ is computed and multiplied by f_X and f_Y; the result is added to the numerator. The numerator is equivalent to the dot product of two frequency vectors, which include all locations in space and their frequencies in the footprints of users r and s. Similarly, the denominator is the product of two quantities $||F(r)||$ and $||F(s||$ which are equivalent to the *Euclidean norms* of the footprints, considering all locations in space. Specifically:

$$||F(r)|| = \sqrt{\sum_{(X,f_X)\in F(r)} |X| * f_X^2}, \quad (2)$$

where $|X|$ denotes the area of region X. The denominator of Eq. 1 ensures that $sim(F(r), F(s))$ ranges from 0 to 1; two identical footprints have a similarity equal to 1 and two entirely disjoint ones have zero similarity. Figure 2(b) shows an example of two geo-footprints from two users r and s. Footprint $F(r)$ originally has two overlapping regions r_1 and r_2 which are converted to three disjoint regions r_A, r_B, r_C with their frequencies shown in parentheses. Hence, $F(r) = \{(r_A, 1), (r_B, 2), (r_C, 1)\}$. Similarly, $F(s) = \{(s_A, 1), (s_B, 2), (s_C, 1)\}$. The footprint similarity between r and s is:

$$\frac{|r_A \cap s_A| \cdot 1 \cdot 1 + |r_B \cap s_A| \cdot 2 \cdot 1 + |r_B \cap s_B| \cdot 2 \cdot 2 + |r_B \cap s_C| \cdot 2 \cdot 1}{\sqrt{52 \cdot 1^2 + 20 \cdot 2^2 + 22 \cdot 1^2} \cdot \sqrt{7 \cdot 1^2 + 1 \cdot 2^2 + 1 \cdot 1^2}}$$

which amounts to $2/\sqrt{77} \approx 0.228$.



(a) Disjoint-regions footprint (b) Regions of two footprints

Figure 2: Disjoint regions and frequencies

5 DATA REPRESENTATION AND SIMILARITY COMPUTATION

Given the set of RoIs of two users, a natural question is how we can compute efficiently the similarity of their footprints. The first thing to investigate is whether we should store a footprint in its original representation (as a set of RoIs) or convert it to a set of disjoint regions and their frequencies. Keeping the original representation is space-efficient, whereas the disjoint regions representation may require $O(2^n)$ space, where n is the number of RoIs. The reason is that any combination of RoIs may define one or more regions. Figure 3 shows an extreme example of 5 RoIs, which define 41 disjoint regions that have to be stored separately (together with their frequencies). Since it is typical for users to visit the same locations over and over again, we expect multiple pairs of RoIs to overlap each other. Another issue is that some of the disjoint regions may not be rectangles (e.g., see region A in Figure 2(a)), hence their representation and processing could be complex. This problem can be alleviated by splitting each such region to a set of rectangles (this is always possible) and replace the region by the rectangles paired with the original frequency of the unified region. This does not affect the correctness of similarity computation, but could greatly increase the number of regions that model a footprint. Overall, storing footprints in their original form (i.e., as RoI sets) is preferable.

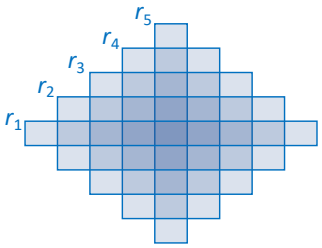


Figure 3: 41 disjoint regions defined by 5 RoIs

On the other hand, the original RoI representation may hinder the computation of similarity (Eq. 1), which is based on the disjoint regions representation. In this section, we propose algorithms that can compute the disjoint regions representation and its norm as well as similarity between two footprints, from the original set of RoIs.

5.1 Norm computation

We start by presenting Algorithm 2, a plane-sweep algorithm that can be used to compute the norm $||F(r)||$ of a footprint $F(r)$. The same algorithm also computes a set of disjoint regions that form $F(r)$ and their frequencies (if such a representation is preferred to RoIs). The algorithm can be used in a *preprocessing* phase, where the norms of all user footprints are computed and stored together with them, such that they can readily be used in Eq. 1 whenever we need to compute the similarity between two footprints.

In a nutshell, Algorithm 2 computes the disjoint regions from the set of RoIs and their frequencies and incrementally constructs the norm by summing the contribution of each such region. Recall that each RoI is a rectangle. Initially, we pick a *sorting* dimension (in the pseudocode, we assume that

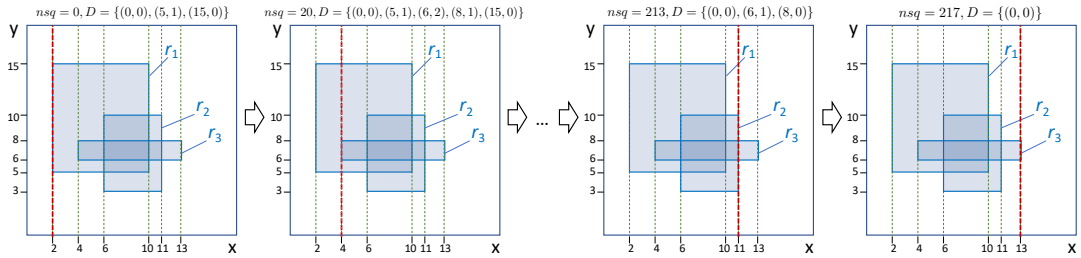


Figure 4: Steps of the norm computation algorithm

this is the x-axis) and take the projection of each RoI r_i on that dimension, which is an interval $[a, b]$; we create two triples $\langle a, r_i.id, Start\rangle$ and $\langle b, r_i.id, End\rangle$. All triples are then sorted using the first column. The algorithm then accesses the triples in order, which model the stops of a plane-sweep line. We initialize a data structure D, which manages the disjoint regions at each stop of the sweep line. D divides the line into intervals; each interval corresponds to a continuous empty space or occupied space by a disjoint region. D keeps a set of $(start, count)$ pairs ordered by $start$. For the first entry in D, $start$ is the smallest value of the non-sweep axis (e.g., y-axis). The $start$ value of each subsequent entry in D defines the end value of the interval of the previous entry. When the sweep line moves from one position to the next one, D is used to compute the areas of the disjoint regions and their contribution to the norm (lines 4-6); we keep in a variable nsq the sum of squares computed as the line progresses. In addition, we keep in a variable $prev$ the previous position of the sweep line (used to compute the areas of the regions between the previous position of the line to the next one).

When the line moves from value $prev$ to v (i.e., the currently accessed projection endpoint is $\langle v, r_i.id, type\rangle$), the first thing to do is to compute the contribution of the disjoint regions in the stripe from x=$prev$ to x=v (lines 4-6). For this, we scan the entries of D in ascending $start$ order and, for each $e \in D$, compute the area of the region defined by y-range $[e.start, e.next.start]$ and x-range $[prev, v]$ and multiply them with $e.count^2$. The result is added to nqo. $e.next$ is the next entry to e in D (if there is no $e.next$, then $e.next.start = \infty$ and $e.count$ should be 0).

After updating nsq, the algorithm updates D to include the correct intervals and their counts. Specifically, if the line stops to the beginning of a RoI r_i, two new entries are added to D, otherwise (r_i ends), two entries that correspond to r_i are removed from D, updating the corresponding counters. When plane sweep finishes, the algorithm computes and returns the square root of nsq which is the norm of the input set of RoIs.

TODO: D should be a data structure that maintains its entries sorted and can access them sequentially (i.e., it could be a balanced binary search tree).

Figure 4 illustrates some of the steps of the norm computation algorithm for the set of RoIs shown in Figure 2(a). The updates nsq and D after each step are shown at the top. At the first stop of the sweep line (v=2), no regions before (D just contains $(0, 0)$), so nsq remains 0. Note that D now contains three elements: $(0, 0)$ corresponding to y-interval $[0, 5]$ with counter 0 (no regions), $(5, 15)$ for y-interval $[5, 15]$ with counter 1 (single region), and $(15, 0)$ for y-interval $[15, \infty]$ with counter 0. At the next stop of the sweep line (v=4), 20 is added to nsq, i.e., the area of region r_1 in the stripe from x=2 to x=4. D is now updated to include intervals [5,6] with a counter 1 (just r_1), [6,8] with a counter 2 (r_1 and r_2), and [8,15] with a counter 1 (just r_1). After a few steps, when the line reaches v=11, three values are added to nsq: the area of $[3, 6] \times [10, 11]$ multiplied by 1^2, the area of $[6, 8] \times [10, 11]$ multiplied by 2^2, and the area of $[8, 10] \times [10, 11]$ multiplied by 1^2, in total 13, hence nsq increases from 200 (previous step) to 213. In addition, two intervals (due to r_2 are deleted from D) and the counter for interval [6,8] is decreased by 1.

Algorithm 2 Norm computation algorithm

Require: set of RoIs (footprint) $F(r)$;
1: Sort endpoints $\langle v, r_i.id, type\rangle$ of RoIs projections on the x-axis
2: $D \leftarrow [(0, 0)]$; $nsq \leftarrow 0$; $prev \leftarrow$ min x-value;
3: **for** each $\langle v, r_i.id, type\rangle$ in v-order **do**
4: **for** each entry e in D in $start$-order **do** ▷ update norm square
5: $nsq \leftarrow nsq + (e.next.start - e.start) \cdot (v - prev) \cdot e.count^2$
6: **end for**
7: **if** $type = Start$ **then** ▷ add entries to D
8: $e \leftarrow e \in D$ with largest $start$, s.t. $e.start \leq r_i.y_{low}$
9: $e \leftarrow (r_i.y_{low}, e.count + 1)$; $D.add(e)$
10: **while** $e.next.start < r_i.y_{up}$ **do**
11: $e.next.count \leftarrow e.next.count + 1$
12: $e \leftarrow e.next$
13: **end while**
14: $D.add((r_i.y_{up}, e.count - 1))$;
15: **else** ▷ $type = End$: remove entries from D
16: $e \leftarrow e \in D with start = r_i.y_{low}$
17: $D.remove(e)$
18: **while** $e.next.start < r_i.y_{up}$ **do**
19: $e.next.count \leftarrow e.next.count - 1$
20: $e \leftarrow e.next$
21: **end while**
22: $D.remove(e.next)$ ▷ entry for $r_i.y_{up}$
23: **end if**
24: $prev = v$
25: **end for**
26: **return** $\sqrt{nsq}$

Complexity Analysis If there are n RoIs in the set, the algorithm needs $2n$ steps. Each step scans D to update nsq and D contains at most $2n$ entries. In addition, each step, updates D to add or remove 2 entries and these changes require a scan of D in the worst case. Hence, the time complexity is $O(n^2)$. Regarding space, the algorithm has to maintain D, which is $O(n)$.

Extraction of Disjoint Regions Algorithm 2 can directly be used to extract a set of disjoint regions and their frequencies, which can be used as an alternative (to the set of RoIs) for representing a footprint. Specifically, when we update nsq (line 5), we can output each of the regions that contribute to nsq together with its frequency (equal to $e.count$). This will give us a set of disjoint rectangular regions that can be used to model the footprint.

Incremental Norm Updates **TODO: Incremental norm update: given a new RoI in a footprint, how can we efficiently update the footprint's norm by simply finding the overlap of the new RoI to existing ones? Can deletions be handled?**

5.2 Similarity computation

We now propose a variant of Algorithm 2, which can be used to compute the similarity between two footprints $F(r)$ and $F(s)$, which are in their original format (i.e., a set of RoIs). Besides the two footprints, Algorithm 3 takes as input their norms (assumed to have been pre-computed). Algorithm 3 sorts the endpoints of all RoI projections on a selected dimension (e.g., the x-axis) and accesses them in order, simulating a plane sweep line that stops at each endpoint. At each stop of the line it maintains in two data structures D_r and D_s the active intervals and their counts from $F(r)$ and $F(s)$. The main difference to Algorithm 2 is lines 5-19, where a routine *merges* the (ordered) contents of D_r and D_s to compute the contribution of a stripe (i.e., the area between the current line position v and the previous one $prev$) to the numerator $simn$ of the similarity function (Eq. 1). This routine computes the *weighted-intersection* between the disjoint regions for $F(r)$ and $F(s)$ in the stripe. After the merge-join routine, D_r or D_s is updated depending on the source of the current endpoint, indicated by flag src in the representation of endpoints. After completing all stops, the algorithm divides $simn$ by the product of the two norms and returns the similarity.

Figure 5 shows a snapshot of Algorithm 3 while computing $sim(\{r_1, r_2\}, \{s_1, s_2\})$ (i.e., for the footprints shown in Figure 2(b)). The sweep line is at $\langle 8, s_2.id, 1, End\rangle$. At this point, $D_r = \{(0,0),(3,1),(5,2),(10,1),(14,0)\}$ and $D_s = \{(0,0),(6,1),(7,2),(8,1),(9,0)\}$. The routine that "joins" D_r with D_s to compute their contribution to $simn$ (lines 5-19) initializes $e_r = (0,0)$ and $e_s = (6,1)$. In the concurrent scan of D_r and D_s by the routine, the (e_r, e_s) pairs that produce non-zero contribution to $simn$ are $((10,1),(6,1))$ (produces $1 \cdot 1 \cdot 1 \cdot 2 = 2$), $((10,1),(7,2))$ (produces $1 \cdot 1 \cdot 2 \cdot 2 = 4$), and $((10,1),(8,1))$ (produces $1 \cdot 1 \cdot 1 \cdot 2 = 2$). For all other pairs, at least one of the two multiplied counts is zero. After the routine, D_s is updated to $\{(0,0),(7,1),(9,0)\}$.

Algorithm 3 Similarity computation algorithm

Require: sets of RoIs (footprints) $F(r)$, $F(s)$; $normr$, $norms$
1: Sort endpoints $\langle v, r_i.id, src, type\rangle$ of RoIs projections on the x-axis
2: $D_r \leftarrow [(0,0)]$; $D_s \leftarrow [(0,0)]$
3: $prev \leftarrow$ min x-value; $simn \leftarrow 0$
4: **for** each $\langle v, r_i.id, src, type\rangle$ in v-order **do**
5: $e_r \leftarrow$first entry in D_r
6: $e_s \leftarrow$second entry in D_s
7: **while** $e_r \neq$ null and $e_s \neq$ null **do**
8: **if** $e_r.start < e_s.start$ **then**
9: $up \leftarrow \min\{e_r.next.start, e_s.start\}$
10: $simn \leftarrow simn +$
11: $(up - e_r.start) \cdot (v - prev) \cdot e_r.count \cdot e_s.prev.count$
12: $e_r \leftarrow e_r.next$
13: **else** ▷ $e_r.start \geq e_s.start$
14: $up \leftarrow \min\{e_s.next.start, e_r.start\}$
15: $simn \leftarrow simn +$
16: $(up - e_s.start) \cdot (v - prev) \cdot e_s.count \cdot e_r.prev.count$
17: $e_s \leftarrow e_s.next$
18: **end if**
19: **end while**
20: **if** $src = 0$ **then** ▷ endpoint from an $F(r)$ RoI
21: update D_r by running lines 7-23 of Alg. 2, for $D = D_r$
22: **else** ▷ endpoint from an $F(s)$ RoI
23: update D_s by running lines 7-23 of Alg. 2, for $D = D_s$
24: **end if**
25: $prev = v$
26: **end for**
27: **return** $simn/(normr \cdot norms)$

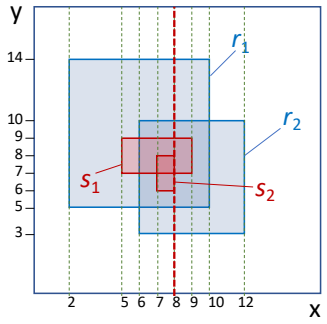


Figure 5: Snapshot of Algorithm 3

Complexity Analysis If there are n RoIs $F(r)$ and m RoIs $F(s)$, the algorithm needs $2(n+m)$ steps. Each step performs a concurrent scan to D_r and D_s to compute their contribution to $simn$; D_r and D_s may include up to $2(n+m)$ entries. The update of eiher D_r or D_s takes at most $O(n)$ and $O(m)$, respectively. Hence, the overall time complexity is $O((n+m)^2)$. The space complexity is $O(n+m)$ as we only have to maintain the current state of D_r and D_s at each step.

Computing Norms and Similarity Simultaneously With a minor modification, Algorithm 3 can also compute the

norms of $F(r)$ and $F(s)$, in case they have not been precomputed. For this, we have to apply lines 4-6 of Algorithm 2 for D_r or D_s before the set is updated and also keep track of the previous stops in D_r and D_s (in place of *prev* of Algorithm 2), to update *normr* or *norms* at each step. Overall, (modified) Algorithm 3 is a general method that can compute the similarity between two sets of RoIs, regardless of whether their norms have been precomputed or not.

5.3 Computation based on spatial join

Interestingly, we can apply any spatial intersection join algorithm [1, 3] (with a minor modification) to compute $sim(F(r), F(s))$, provided that the norms $||F(r)||$ and $||F(s)||$ are available. This approach is very intuitive. Each pair (r_i, s_j) of RoIs, $r_i \in F(r)$, $s_j \in F(s)$ that intersect contribute to the numerator *simn* of Eq. 1 as much as the area of their intersection. For example, consider the two footprints shown in Figure 2(b); region $s_B(2)$ will (correctly) contribute four times to the numerator *simn*, as it is included in the intersection area of all four spatial join pairs (r_1, s_1), (r_1, s_2), (r_2, s_1), and (r_2, s_2). Algorithm 4 describes a similarity computation algorithm based on this idea. In the example of Figure 5, Algorithm 3 "breaks" the contribution of intersecting pair (r_1, s_1) to 5 pieces of contributions counted at four stops of the sweep line, whereas Algorithm 4 computes and adds this contribution just once, i.e., when (r_1, s_1) is identified. Note that, unlike Algorithm 3, Algorithm 4 cannot be extended to compute the norms $||F(r)||$ and $||F(s)||$, so, it cannot compute similarity if the norms are not given.

Algorithm 4 Join-based similarity computation

Require: sets of RoIs (footprints) $F(r)$, $F(s)$; *normr*, *norms*
1: $simn \leftarrow 0$
2: **for** each pair (r_i, s_j), $r_i \in F(r)$, $s_j \in F(s)$ that intersect **do**
3: $simn \leftarrow simn + |r_i \cap s_j|$ ▷ add intersection area of pair
4: **end for**
5: **return** $simn/(normr \cdot norms)$

Complexity Analysis Algorithm 4 is expected to be faster than Algorithm 3, as plane-sweep based spatial intersection join has a cost of $O(n \log n) + O(m \log m) + O(n + m + K)$, where K is the size of the join output [3]. For each join pair, we only have to compute the intersection area which takes $O(1)$ per pair.

5.4 Discretization

Two RoIs in a user footprint may have large overlap, but they are unlikely to be identical. This is due to the fact that two identical positions are seldom sampled by GPS or indoor location methods. At the same time, the intent of users is not very sensitive to their exact position (besides, the definition and computation of RoIs are approximate by nature). Therefore, it could be reasonable to *discretize* the domain of RoIs to a set UR of *unit regions* and define each RoI as a subset of UR. Discretization could be done by a regular grid that divides the space into cells or by a domain expert, who takes into account the (groups of) points of interest in space. Figure 6 illustrates the discretization of the space where customers of a small supermarket move. The rectangles with dashed lines are the unit regions.

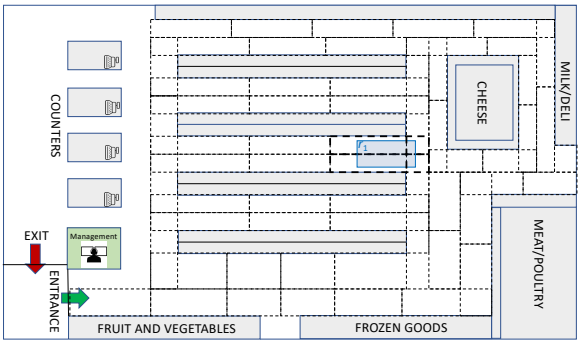


Figure 6: Customer space discretization in a super market

Given a domain discretization D, each RoI can be approximated by the set of units it intersects. For example, RoI r_1 in Figure 6 is modeled by the highlighted four units which enclose it. Then, the discretized footprint $\overline{F(r)}$ of a user r becomes a set of units and their weights $\overline{F(r)} = \{(u, w(r,u)) : u \in D\}$, where the weight $w(r,u)$ of each unit u is the number of RoIs in $F(r)$ that have u in their approximations (obviously, we do not have to store units of zero weight). The similarity $sim(\overline{F(r)}, \overline{F(s)})$ between two discretized footprints is now defined as:

$$sim(\overline{F(r)}, \overline{F(s)}) = \frac{\sum_{u \in D} area(u) \cdot w(r,u) \cdot w(s,u)}{||\overline{F(r)}|| \cdot ||\overline{F(s)}||}, \quad (3)$$

where

$$||\overline{F(r)}|| = \sqrt{\sum_{(u,w(u)) \in \overline{F(r)}} area(u) \cdot w(u)^2} \quad (4)$$

Working with discretized footprints can greatly decrease the cost of similarity computation, as the overhead of computing a possibly large number of intersection regions between arbitrary RoIs is avoided. On the other hand, there may be an accuracy loss in similarity computation especially if the RoIs are small compared to the discretized units. Therefore, the sizes of units should be defined to (i) fit well the application domain and (ii) be smaller (in general) compared to the expected RoIs. The total number of units in the discretized footprints affect performance, so we should not define extremely small units.

TODO: discuss tf.idf approach together with discretized representations

6 INDEXING AND SIMILARITY SEARCH

The next question to answer is how to store and index the geo-footprints of users in order to support fast similarity

search between users. In this section, we propose an indexing scheme for geo-footprints and show how it can be used to search for similar users to a query user q. Given the footprint of a *query user* q, the search objective is to find the k users with the highest footprint-based similarity to q.

6.1 Using an off-the-shelf R-tree

Recall, from the previous section, that we model the geo-footprint of each user in its original form, i.e., as a set of RoIs in the form of MBRs. Hence, a natural approach for indexing the footprints is to use a data structure for MBRs, such as the R-tree. The difference from a classic R-tree is that different indexed objects (i.e., different RoI-MBR) may have the same ID, if they belong to the same user, which is the ID of the user, whereas in a clasic R-tree, each indexed MBR corresponds to a different object. Still this change not affect the functionality of the index and its use in similarity search, as we will explain shortly.

6.1.1 Iterative search. The first (baseline) algorithm that we are going to discuss, each RoI $q.r$ of the query user q, searches the R-tree to find users u who have regions in their footprints $F(u)$ that overlap with $q.r$. For each such region $u.r$, the algorithm updates the similarity of u w.r.t. q by adding to the numerator $sim(F(u), F(q))$ the area of the spatial intersection $u.r \cap q.r$. The denominator of $sim(F(u), F(q))$ can be computed once, given that we have access to $||F(u)||$ and given that $||F(q)||$ can be computed once in the beginning of query processing, using Algorithm 2.

While finding users whose footprints overlap with $q.r$, we can keep track of the set of k most similar users to q so far. After finishing this iterative search (i.e., one spatial search for each $q.r \in F(q)$), the k most similar users found so far become the query result.

TODO: show example

6.1.2 Batch search. An improved approach is to perform search for all $q.r \in F(q)$ simultaneously. Specifically, we access the R-tree in search for the R-tree leaf nodes that have non-zero spatial overlap with $F(q)$. For this, we use the MBR of $F(q)$ to guide search. Whenever we visit a leaf node L of the tree, we conduct a *spatial join* between $F(q)$ and the contents of L. For each pair $(u.r, q.r)$ of intersecting RoIs, where $u.r \in L$ and $q.r \in F(q)$, we update $sim(F(u), F(q))$.

The $L \bowtie F(q)$ join can be done using the optimizations of [3]. Specifically, before performing the join, we access all contents of L and remove from consideration all $u.r \in L$ which do not intersect with $MBR(F(q))$. In addition, we ignore all $q.r \in F(q)$ which do not intersect $MBR(L)$. Then, we do a plane-sweep join between the non-eliminated RoIs in L and $F(q)$.

We expect this batch search approach to be significantly faster than the iterative baseline method because it avoids accessing the same R-tree nodes more than once per query.

TODO: show example

6.1.3 Early termination. The previously discussed methods compute $sim(F(u), F(q))$ for *all* users u with a non-zero similarity with q. In other words, they do not have a mechanism for pruning parts of the search space that do not affect the query result. We can do better, if we prioritize search on the R-tree, based on the overlap of its node MBRs with $F(q)$. Specifically, the highest contribution that a R-subtree rooted at node n can give to the similarity $sim(F(u), F(q))$ of any user equals $\sum_{r \in F(q)} |r \cap M(n)| \cdot m$, where $|r \cap M(n)|$ is the area of the overlap between region r and the MBR $M(n)$ of node n, and m is the maximum number of regions that any user has in her footprint. For example, **TODO: add example** By adding the highest contribution of a node n to the numerator of the already accumulated similarity of a user u, we can derive an upper bound of $sim(F(u), F(q))$.

We can take advantage of the highest contributions to design a best-first search algorithm as follows. First, we add all entries of the R-tree's root to a max-heap H. The entries e in H are organized based on their contributions $\sum_{r \in F(q)} |r \cap e.MBR| \cdot m$. Hence, each time the top element of H is the entry with the highest contribution to any user similarity. At the same time, we keep track of the *sum* of all contributions in H and the (partial) similarities of all users to $F(q)$ computed so far. We visit the R-tree nodes guided by H; at each step, we de-heap the top entry e of H. If e points to a non-leaf, we visit the node pointed by e and en-heap its entries to H. If e points to a leaf n, we visit the node and update the similarities of the users in n using the plane-sweep approach presented in Section **??**. The current similarities of all users are also organized in a max-heap U.

If, for a user u, the similarity $sim(F(u), F(q))$ computed so far exceeds the upper bound of the next best user then we can report u as the next most similar user to q. The upper bound of the next best user can be computed by taking the second-best user in U and adding to its similarity the sum of all contributions in H. As soon as the desired number k of most similar users has been computed, the algorithm can terminate.

[nikos: I do not expect this to work well because the bounds are too loose. This method could be more expensive than the simple batch search method. One possible optimization is to keep track of the number of regions for each user that have been seen so far, in order to derive more accurate upper bounds, where m is replaced by the number of unseen regions for each user and we do not compute the sum of contributions in H, but the maximum contribution per user, considering the overlaps in decreasing order in H. This can lead to a more effective bound, but it requires more expensive bound computations and updates.]

6.2 Using an augmented R-tree

The basic data structure of the R-tree can be changed to facilitate the computation of effective bounds that can reduce the cost of search and terminate early.

A simple approach is to augment each R-tree entry e with a number $e.max$, which is the maximum number of regions

that belong to the same user in all regions indexed in the subtree pointed by e. We denote this index by R^A_{simple}. If we have this information, we can set the bound for e (for the heap H) to $\sum_{r \in F(q)} |r \cap e.MBR| \cdot e.max$, which is expected to be more accurate than using the global maximum m.

A more "aggressive" approach is to keep, for each entry e of an R-tree node, together with its MBR, a vector of the form $\{(u_1, n_1), (u_2, n_2), \ldots, (u_m, n_m)$, where u_i is a user for which there is at least one region of $F(u_i)$ in the subtree pointed by e and n_1 is the number of regions of $F(u_i)$ in the subtree. We denote this index by $R^A_{detailed}$. Given this information, for each query region $q.r$ in the query footprint $F(q)$, we can derive an upper bound of how much $sim(F(u_i), F(q))$ can be increased by the subtree pointed by e. Specifically, this upper bound is $\sum_{q.r \in F(q)} |r \cap e.MBR| \cdot u_i$, as in the best case, the overlap between each region $u_i.r \in F(u_i) \cap e$ is equal to the intersection $|e.MBR \cap q.r|$. Given this, for each user which is indexed under entry e, we can derive an upper bound for the contribution of the subtree rooted at e to $sim(F(u_i), F(q))$. During search, we give higher priority to entries e that have large overlap with the query region(s). For each user, we can maintain a lower/upper bound and keep track of the top-k users by lower bound. As soon as the upper bounds of users not currently in the top-k are all not larger than the k-th lower bound, we can terminate search and report the top-k users.

6.3 User-centric R-tree

Another possible way to index the data is to index, for each user u, the MBR of $F(u)$ and the number of regions $|F(u)|$ in $F(u)$. That is, the regions in a user footprint $F(u)$ are not stored/indexed independently, but a single entry is stored per user, which entails the MBR of $F(u)$. We denote this *user-centric* index by R^U. We use the tree to find the user footprint MBRs that intersect the query footprint $F(q)$. For each such footprint $F(u)$, in a refinement step, we use Algorithm **??** to compute $sim(F(u), F(q))$.

We prioritize accesses to R-tree nodes based on the area of overlap with the query region(s), in order to get to the most similar users faster. We can have an augmented version of R^U, denoted by R^{UA}, where, for each entry e, we also keep track of $e.max$ the maximum $|F(u)|$ under e. Best-first search is applied and the priority key is overlap_area$(q, e.MBR)$*$e.max$. If the top heap element cannot lead to a user that can enter the top-k set of most similar users, we terminate.

//OLD STUFF

Three options:

a) Store footprints in their original format (as MBR sets), build a spatial index for them (e.g. R-tree, our grid). Then a sim query can be solved as a *batch* range query evaluation, where we prune non-leaf nodes if they do not overlap with the footprint and for leaf nodes, we perform a spatial join. We need a data structure (hash table) which keeps track of the users with non-zero similarity so far, in order to update their similarities after each join.

b) Same as above, but we use the decomposed representation of RoIs to weighted disjoint regions. Not sure if there is any advantage compared to (a).

c) Use discretized representations and use an inverted file to index the footprints which are relevant to each discretized region/cell. Then an IR-style approach can be used for similarity search

7 EXPERIMENTS

In this section we present our experimental analysis. We first describe our setup and then present our experiments, which evaluate the effectiveness of the algorithms we proposed for user similarity

Setup

Datasets. We experimented with publicly available Gowalla dataset [5]. Gowalla is a location-based social networking site where users share their locations by checking-in. The dataset consists of 200k check-in locations from 93024 distinct users. A location where a user has check-in is considered a region of interest. The objects in the dataset were normalized so that the coordinates of each region take values inside [0,1].

TODO: Add description and experiments for GeoLife dataset

Methodology. We implemented the footprint extraction algorithm we proposed in section 3 with ϵ being 0.2 and τ 5. With the footprints extracted we next investigate the performance of the user similarity algorithms we proposed in section 5.2. Specifically, we compare three different approaches. The first one pre-computes the norms for each user and then computes the similarity. The second computes the norms while computing similarity between users. Our last approach employs the spatial join based similarity algorithm. All methods perform 90k similarity computations. Figure 7 breaks down the performance of these methods. We observe that pre-computing the norms for each user adds a significant overhead to the performance of algorithm 3. Finally, spatial similarity outperforms the other methods, confirming our complexity analysis.

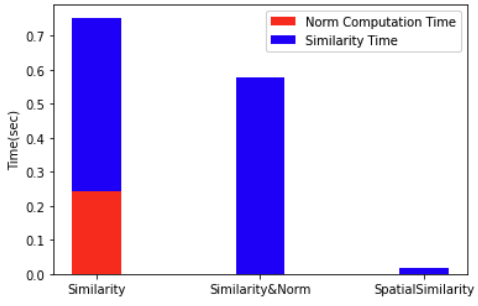


Figure 7: Performance of different similarity computation approaches

8 CONCLUSIONS

In this work, we +++

REFERENCES

[1] Lars Arge, Octavian Procopiuc, Sridhar Ramaswamy, Torsten Suel, and Jeffrey Scott Vitter. 1998. Scalable Sweeping-Based Spatial Join. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*. Morgan Kaufmann, 570–581.

[2] Jiang Bian, Dayong Tian, Yuanyan Tang, and Dacheng Tao. 2019. Trajectory Data Classification: A Review. *ACM Trans. Intell. Syst. Technol.* 10, 4 (2019), 33:1–33:34. https://doi.org/10.1145/3330138

[3] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. 1993. Efficient Processing of Spatial Joins Using R-Trees. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, May 26-28, 1993*, Peter Buneman and Sushil Jajodia (Eds.). ACM Press, 237–246.

[4] Maike Buchin, Anne Driemel, Marc J. van Kreveld, and Vera Sacristán. 2011. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *J. Spatial Inf. Sci.* 3, 1 (2011), 33–63. https://doi.org/10.5311/JOSIS.2011.3.66

[5] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, Chid Apte, Joydeep Ghosh, and Padhraic Smyth (Eds.). ACM, ACM, 1082–1090. http://doi.acm.org/10.1145/2020408.2020579

[6] Xin Ding, Lu Chen, Yunjun Gao, Christian S. Jensen, and Hujun Bao. 2018. UlTraMan: A Unified Platform for Big Trajectory Data Management and Analytics. *Proc. VLDB Endow.* 11, 7 (2018), 787–799. https://doi.org/10.14778/3192965.3192970

[7] Chenjuan Guo, Bin Yang, Jilin Hu, and Christian S. Jensen. 2018. Learning to Route with Sparse Trajectory Sets. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*. IEEE Computer Society, 1073–1084. https://doi.org/10.1109/ICDE.2018.00100

[8] Gísli R. Hjaltason and Hanan Samet. 1999. Distance Browsing in Spatial Databases. *ACM Trans. Database Syst.* 24, 2 (1999), 265–318. https://doi.org/10.1145/320248.320255

[9] Niklas Karlsson, Enrico Di Bernardo, James P. Ostrowski, Luis Goncalves, Paolo Pirjanian, and Mario E. Munich. 2005. The vSLAM Algorithm for Robust Localization and Mapping. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, ICRA 2005, April 18-22, 2005, Barcelona, Spain*. IEEE, 24–29. https://doi.org/10.1109/ROBOT.2005.1570091

[10] Fan Li, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. 2012. A reliable and accurate indoor localization method using phone inertial sensors. In *The 2012 ACM Conference on Ubiquitous Computing, Ubicomp '12, Pittsburgh, PA, USA, September 5-8, 2012*, Anind K. Dey, Hao-Hua Chu, and Gillian R. Hayes (Eds.). ACM, 421–430. https://doi.org/10.1145/2370216.2370280

[11] Siyuan Liu, Yunhuai Liu, Lionel M. Ni, Jianping Fan, and Minglu Li. 2010. Towards mobility-based clustering. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*. ACM, 919–928.

[12] Xin Lu, Changhu Wang, Jiang-Ming Yang, Yanwei Pang, and Lei Zhang. 2010. Photo2Trip: generating travel routes from geo-tagged photos for trip planning. In *Proceedings of the 18th International Conference on Multimedia 2010, Firenze, Italy, October 25-29, 2010*. ACM, 143–152. https://doi.org/10.1145/1873951.1873972

[13] Shuyao Qi, Panagiotis Bouros, Dimitris Sacharidis, and Nikos Mamoulis. 2015. Efficient Point-Based Trajectory Search. In *Advances in Spatial and Temporal Databases - 14th International Symposium, SSTD 2015, Hong Kong, China, August 26-28, 2015. Proceedings (Lecture Notes in Computer Science, Vol. 9239)*. Springer, 179–196. https://doi.org/10.1007/978-3-319-22363-6_10

[14] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed In-Memory Trajectory Analytics. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, Gautam Das, Christopher M. Jermaine, and Philip A. Bernstein (Eds.). ACM, 725–740. https://doi.org/10.1145/3183713.3183743

[15] Zhou Shao, Muhammad Aamir Cheema, David Taniar, and Hua Lu. 2016. VIP-Tree: An Effective Index for Indoor Spatial Queries. *Proc. VLDB Endow.* 10, 4 (2016), 325–336. https://doi.org/10.14778/3025111.3025115

[16] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. 2020. A survey of trajectory distance measures and performance evaluation. *VLDB J.* 29, 1 (2020), 3–32. https://doi.org/10.1007/s00778-019-00574-9

[17] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, and Gao Cong. 2021. A Survey on Trajectory Data Management, Analytics, and Learning. *ACM Comput. Surv.* 54, 2 (2021), 39:1–39:36. https://doi.org/10.1145/3440207

[18] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast Large-Scale Trajectory Clustering. *Proc. VLDB Endow.* 13, 1 (2019), 29–42. https://doi.org/10.14778/3357377.3357380

[19] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A Search Engine for Trajectory Data. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, SIGIR 2018, Ann Arbor, MI, USA, July 08-12, 2018*. ACM, 535–544. https://doi.org/10.1145/3209978.3209989

[20] Zuchao Wang, Min Lu, Xiaoru Yuan, Junping Zhang, and Huub van de Wetering. 2013. Visual Traffic Jam Analysis Based on Trajectory Data. *IEEE Trans. Vis. Comput. Graph.* 19, 12 (2013), 2159–2168. https://doi.org/10.1109/TVCG.2013.228

[21] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. 2019. A Survey of Indoor Localization Systems and Technologies. *IEEE Commun. Surv. Tutorials* 21, 3 (2019), 2568–2599. https://doi.org/10.1109/COMST.2019.2911558