Similarity Search based on Geo-footprints

Achilleas Michalopoulos
Department of Computer Science and Engineering
University of Ioannina, Greece
amichalopoulos@cse.uoi.gr

Konstantinos Lampropoulos
Department of Computer Science and Engineering
University of Ioannina, Greece
klampropoulos@cse.uoi.gr

George Kelantonakis
University of Crete and FORTH-ICS
Heraklion, Greece
kelantonag@ics.forth.gr

Chrysostomos Zeginis
University of Crete and FORTH-ICS
Heraklion, Greece
zegchris@ics.forth.gr

Kostas Magoutis
University of Crete and FORTH-ICS
Heraklion, Greece
magoutis@ics.forth.gr

Nikos Mamoulis
Department of Computer Science and Engineering
University of Ioannina, Greece
nikos@cse.uoi.gr

ABSTRACT

Many applications track the movements of mobile users, especially in controlled (e.g., indoor) environments. This information can be combined with other data (e.g., user transaction records) for effective promotion marketing 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, inspired by 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 or nearest-neighbor classifiers).

ACM Reference Format:
Achilleas Michalopoulos, Konstantinos Lampropoulos, George Kelantonakis, Chrysostomos Zeginis, Kostas Magoutis, and Nikos Mamoulis. 2017. Similarity Search based on Geo-footprints. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

There has been a lot of previous work on managing the locations and movement of mobile objects. Most of it focuses on handling (outdoor) user trajectories [19]. This includes trajectory data analytics systems [8, 15], trajectory cleaning and transformation [5], trajectory similarity measures [17], trajectory retrieval [14, 21], and trajectory clustering [20] and classification [2]. Targeted applications include traffic monitoring and analysis [22], navigation [9], trip recommendation [13]. Location-based services for indoor spaces are gaining popularity [16], 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.) [23], computer vision [10], or by using the smartphone's inertial sensors [11]. 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.

This work focuses on the capturing and use, for each mobile user, of the locations or regions have been of *interest* to the user, such as a visit to the TVs section of a department store. Instead of storing the entire history of user locations (i.e., the complete trajectories), we focus on location information, which concisely captures and 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). We call this information *geo-footprints*.

Example 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.

Motivation and Contribution By comparing the geo-footprints of two 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. 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, such as Foursquare. 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.

The contributions of this paper 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 the concept of geo-footprint, which finds application in market-analysis and recommendation.
- We propose a measure for the similarity between geo-footprints, which naturally extends document similarity measures from information retrieval.
- 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 evaluate our techniques using real and synthetic data.

2 RELATED WORK

There has been ample work on tracking, managing. and analyzing mobility data [2, 5, 8, 10, 11, 13–17, 19–23], as discussed in the introduction. Related to our work is the problem of identifying hot spots of moving vehicles [12], which finds application in traffic management and transportation analysis. 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. As opposed to our geo-footprints, these hot spots are not linked to individual users, so they cannot be used as personalized spatial profiles.

Related to our geo-footprints are the locations associated to documents or objects in location-based retrieval [6, 7, 18]. The locations mentioned in each document are extracted and indexed together with the text content, to facilitate location-based keyword search (i.e., find documents containing biword "Chinese restaurant" near my location). The differences between these regions/locations and our geo-footprints is that, for a single object (or document), the same location is not allowed to be included multiple times in the object's profile. Hence, our geo-footprint definition and similarity functions are essentially different.

3 GEO-FOOTPRINTS

This section presents definitions and 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.

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) that encloses 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 measured using Euclidean distance (L_2). 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 to indicate that the user's high interest to the region. 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.

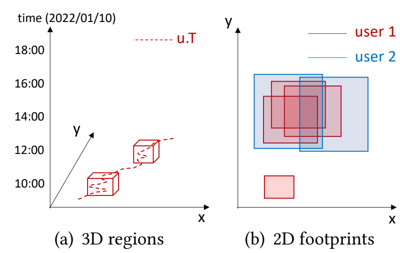


Figure 1: Extracted regions and user footprints

3.2 Geo-footprint Extraction

From the same trajectory, we could potentially extract a huge number of regions that satisfy Definition 2. This is because two regions of interest R_1 and R_2 in the same trajectory may temporally overlap. To minimize the number of extracted regions per trajectory and at the same time unify temporally consecutive, overlapping regions, we propose Algorithm 1 as an extraction algorithm for regions which are temporally maximal and temporally disjoint. Algorithm 1 is 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 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 ← null                                          ▷ Current region
 2: for each l_i ∈ T do
 3:     if R ∪ l_i.p does not violate ε then
 4:         add l_i to R
 5:     else
 6:         if |R| ≥ τ then                   ▷ Current region has enough points
 7:             add R to user profile
 8:             R ← {l_i}                          ▷ Initialize current region
 9:         else
10:             newR ← {l_i}                           ▷ Initialize new region
11:             while newR ∪ R.last does not violate ε do
12:                 newR ∪ R.last; delete R.last;
13:             end while
14:             R ← newR                           ▷ Initialize current region
15:         end if
16:     end if
17: end for
18: if |R| ≥ τ then                          ▷ Last region has enough points
19:     add R to user profile
20: end if
```

We run Algorithm 1 for all trajectories of a user u, to 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 GEO-FOOTPRINTS

To define the similarity between users based on their geo-footprints, we follow an information retrieval approach, and define the preference of a user u to a location l by the number of times l is in the RoIs of $F(u)$, i.e., u's geo-footprint. 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 spatial 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:

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

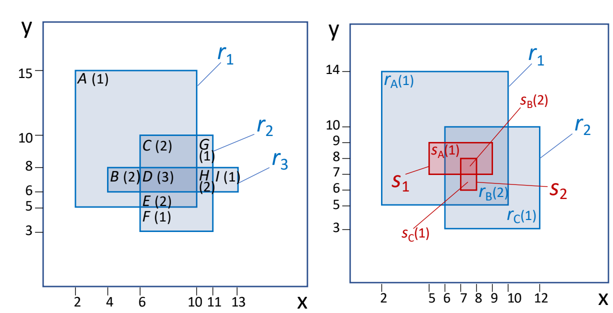
As in cosine similarity, the numerator in Eq. 1 aggregates the common locations in the footprints $F(r)$ and $F(s)$ and multiplies them with their frequencies in the footprints. That is, for each pair (X, Y) of regions that intersect and X is in $F(r)$, Y is in $F(s)$, their area $|X \cap Y|$ of their 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 SIMILARITY COMPUTATION

Given the footprints $F(r)$ and $F(s)$ of two users r and s, we investigate how we can efficiently compute their similarity. We first study the efficient computation of the norm $||F(r)||$ of a footprint $F(r)$ and then suggest methods for similarity computation.

5.1 Norm computation

Algorithm 2 is a plane-sweep algorithm that computes the norm $||F(r)||$ of a footprint $F(r)$. Alg. 2 can be used in a *preprocessing* phase, where the norms of all user footprints are computed and stored, so that they can readily be used in Eq. 1 whenever we need to compute the similarity between two footprints.

Algorithm 2 computes from the set of RoIs of $F(r)$ a set of disjoint regions and their frequencies, i.e., the (X, f_X) pairs discussed in Sec. 4. While doing so, it incrementally constructs the norm by summing the contribution of each such region X. Recall that each RoI is a rectangle. Initially, we pick a *sorting* dimension (e.g., 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 *ssq* the sum of squares computed as the line progresses. We also keep in variable *prev* the previous position of the sweep line (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 *ssq*, 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. In the end, Alg. 2 returns the $\sqrt{ssq}$ which is the norm of the input set of RoIs.

Complexity Analysis If there are n RoIs in the set, the algorithm needs $2n$ steps. Each step scans D to update *ssq* 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 $O(n)$.

Algorithm 2 Norm computation algorithm

```
Require: set of RoIs (footprint) F(r);
1: Sort endpoints ⟨v, r_i.id, type⟩ of RoIs projections on the x-axis
2: D ← [(0, 0)]; ssq ← 0; prev ← min x-value;
3: for each ⟨v, r_i.id, type⟩ in v-order do
4:     for each entry e in D in start-order do          ▷ update norm square
5:         ssq ← ssq + (e.next.start − e.start) · (v − prev) · e.count^2
6:     end for
7:     if type = Start then                              ▷ add entries to D
8:         e ← e ∈ D with largest start, s.t. e.start ≤ r_i.y_low
9:         e ← (r_i.y_low, e.count + 1); D.add(e)
10:        while e.next.start < r_i.y_up do
11:            e.next.count ← e.next.count + 1
12:            e ← 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 ← e ∈ Dwithstart = r_i.y_low
17:        D.remove(e)
18:        while e.next.start < r_i.y_up do
19:            e.next.count ← e.next.count − 1
20:            e ← 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 √ssq
```

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 *ssq* (line 5), we can output each of the regions that contribute to *ssq* 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.

5.2 Similarity computation

We 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 precomputed). 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 5-17, 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.

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 state of D_r and D_s at each step.

Algorithm 3 Similarity computation algorithm

```
Require: sets of RoIs (footprints) F(r), F(s); normr, norms
Sort endpoints ⟨v, r_i.id, src, type⟩ of RoIs projections on the x-axis
D_r ← [(0, 0)]; D_s ← [(0, 0)]
prev ← min x-value; simn ← 0
for each ⟨v, r_i.id, src, type⟩ in v-order do
    e_r ←first entry in D_r
    e_s ←second entry in D_s
    while e_r ≠ null and e_s ≠ null do
        if e_r.start < e_s.start then
            up ← min{e_r.next.start, e_s.start}
            simn ← simn + (up − e_r.start) · (v − prev) · e_r.count · e_s.prev.count
            e_r ← e_r.next
        else                                          ▷ e_r.start ≥ e_s.start
            up ← min{e_s.next.start, e_r.start}
            simn ← simn + (up − e_s.start) · (v − prev) · e_s.count · e_r.prev.count
            e_s ← e_s.next
        end if
    end while
    if src = 0 then                                   ▷ endpoint from an F(r) RoI
        update D_r by running lines 7-23 of Alg. 2, for D = D_r
    else                                              ▷ endpoint from an F(s) RoI
        update D_s by running lines 7-23 of Alg. 2, for D = D_s
    end if
    prev = v
end for
return simn/(normr · norms)
```

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. 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
simn ← 0
for each pair (r_i, s_j), r_i ∈ F(r), s_j ∈ F(s) that intersect do
    simn ← simn + |r_i ∩ s_j|                ▷ add intersection area of pair
end for
return simn/(normr · 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.

6 INDEXING AND SIMILARITY SEARCH

In this section, we investigate indexing schemes for geo-footprints and show how they 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 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.

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.

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.

6.2 User-centric R-tree

Another possible way to organize the data is to index in an R-tree, for each user u, the MBR of $F(u)$ and the ID of u, which can be used to access $F(u)$. That is, the regions in a user footprint $F(u)$ are not stored/indexed independently, but as a single entry. We denote this *user-centric* R-tree 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 5.3 to compute $sim(F(u), F(q))$.

R^U indexes the regions of each user together and computes the similarity of each accessed user in one step, instead of accumulating it during search, as the approach of Sec. 6.1; hence, it has an advantage if the RoIs in each geo-footprint are relatively close to each other. However, if the RoIs in user footprints are dispersed, the footprint MBRs will be very large and R^U is not expected to be effective.

7 EXPERIMENTS

In this section we present our experimental analysis. We first describe our setup and then present our experiments, which evaluate the efficiency of the algorithms we proposed for footprint extraction, norm computation, similarity computation, and similarity search.

Setup We compiled all codes in g++ 9.4.0 with flag -O3 and ran experiments on a 32GB Ubuntu 20.04.3 LTS machine with Intel Core i9-10900K CPU @3.70GHz.

Datasets. We experimented with a publicly available ATC shopping center dataset [4]. ATC is a dataset consisting of user trajectories which are exported from raw sensor measurements. The data is divided in different parts, each of which corresponds to a 8-10 day period of continuous recordings of user movements. The spatial coordinates of the trajectory points were normalized to take values inside [0,1]. In our experiments, we used four of the available parts as different datasets, denoted by Part A, B, C, D, respectively. Table 1 summarizes statistics about the data.

Table 1: Statistics of data and extracted RoIs

ATC Shopping Center	# users	avg. #regions	avg. relative	
			x-extent	y-extent
Part A	349K	68	0.034567	0.034247
Part B	292K	73	0.033608	0.033273
Part C	450K	70	0.040079	0.039694
Part D	377K	77	0.039513	0.039162

Footprint extraction. We implemented the footprint extraction algorithm we proposed in Section 3. Based on the data characteristics and in order to define realistic RoIs for the users based on their movements, we tuned the value of ϵ to 0.002 and the value of τ to 5. To decide of the best ϵ and τ, we experimented with different values and used the ones that lead to a resonable number of RoIs for each user, which are not excessively large. Table 1 summarizes, for each dataset, the average number of RoIs per user footprint and the average extent of the RoIs in each dimension. Footprint extraction was fast; it took about 70 sec for all 349K users of Part A.

Norm computation. After extracting the footprints for each of the four parts, we applied the norm computation algorithm (Algorithm algo:norm) on each part, to compute the norms of all footprints in it, which are to be used in similarity computations. The computation of norms for all parts is relatively fast. Indicatively, for Part A, it took us 38.29 sec. for computing the norms of all 349K user footprints (i.e., about 100 μsec per norm). The times are similar for the other parts.

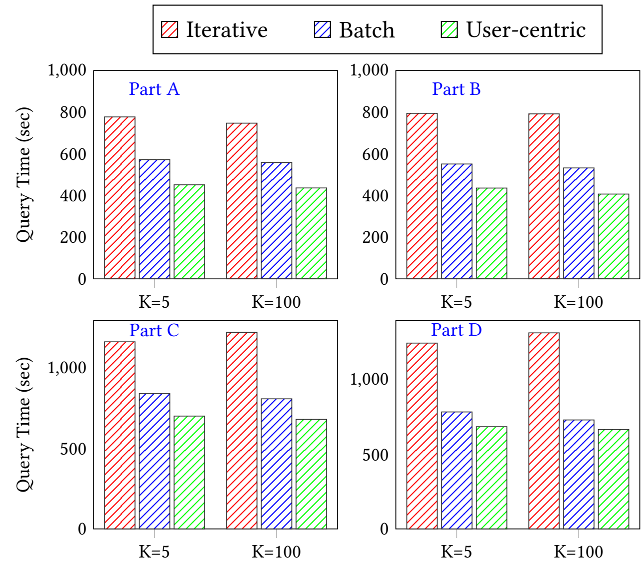


Figure 3: Similarity search performance

Similarity computation. With the footprints extracted we next investigate the performance of the user similarity algorithms we proposed in Sections 5.2 and 5.3. For this, we used Part A and selected 200 random user footprints in it. We measured the total cost of $200 \cdot 349K \approx 70M$ similarity computations between every footprint in the random sample and all data footprints, using Alg. 3 and Alg. 4. On average, Alg. 3 spends 9.3 μsec per similarity computation, while Alg. 4 only needs 1 μsec. Hence, Alg. 4 is the best option, provided that the norms have been pre-computed and they are available, which is consistent to our complexity analysis.

Similarity search. In the last set of experiments, we compare the three indexing and search approaches suggested in Sec. 6; namely, (i) using an R-tree and performing iterative search for each region in the query user footprint (Sec. 6.1.1) or (ii) performing batch search (Sec. 6.1.2), or (iii) using a user-centric R-tree, which indexes footprint MBRs (Sec. 6.2). Figure 3 shows the total runtime for 1000 random top-K similarity queries (sampled from the data) on each part. Observe that the user-centric R-tree approach consistently outperforms the other methods, because it constructs smaller trees and efficiently performs similarity search for each user footprint MBR that intersects the query footprint using Alg. 4.

8 CONCLUSIONS

In this work, we defined the concept of geo-footprints, which comprise the spatial regions in indoor spaces whereabout users stay for long time periods and model the user interests to nearby items. In addition, we defined the similarity between geo-footprints of different users which can be used in location-based market analysis and recommendations (e.g., in e-commerce). We presented efficient algorithms for similarity computation and for similarity-based search, based on geo-footprints. In the future, we plan to investigate the enrichment of footprints with temporal information (i.e., duration) and the adaptation of similarity definition and computation, accordingly.

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] Drazen Brscic, Takayuki Kanda, Tetsushi Ikeda, and Takahiro Miyashita. 2013. Person Tracking in Large Public Spaces Using 3-D Range Sensors. *IEEE Trans. Hum. Mach. Syst.* 43, 6 (2013), 522–534.

[5] 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

[6] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. 2006. Efficient query processing in geographic web search engines. In *ACM SIGMOD*. ACM, 277–288.

[7] Gao Cong, Christian S. Jensen, and Dingming Wu. 2009. Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects. *Proc. VLDB Endow.* 2, 1 (2009), 337–348.

[8] 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

[9] 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

[10] 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

[11] 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

[12] 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.

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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

[18] Artur Titkov and Panagiotis Bouros. 2022. Spatially Combined Keyword Searches. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT 2022, Edinburgh, UK, March 29 - April 1, 2022*. OpenProceedings.org, 2:403–2:407.

[19] 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

[20] 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

[21] 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

[22] 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

[23] 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