TOP-K QUERIES OVER DIGITAL TRACES

YIFAN LI

A THESIS SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

GRADUATE PROGRAM IN COMPUTER SCIENCE
YORK UNIVERSITY
TORONTO, ONTARIO
DECEMBER 2018

Abstract

Recent advances in social and mobile technology have enabled an abundance of digital traces (in the form of mobile check-ins, WiFi hotspots handshaking, etc.) revealing the physical presence history of diverse sets of entities. One challenging, yet important, task is to identify k entities that are most closely associated with a given query entity based on their digital traces [1]. We propose a suite of hierarchical indexing techniques and algorithms to enable fast query processing for this problem at scale. We theoretically analyze the pruning effectiveness of the proposed methods based on a human mobility model which we propose and validate in real life situations. Finally, we conduct extensive experiments on both synthetic and real datasets at scale, evaluating the performance of our techniques, confirming the effectiveness and superiority of our approach over other applicable approaches across a variety of parameter settings and datasets.

[1]This thesis focuses on Query-by-Example tasks.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor, Prof. Xiaohui Yu, for his guidance. In the past two years, Prof. Yu has led and encouraged me with great patience and experience to explore novel and meaningful research topics and helped me to understand how to be a researcher.

Besides my supervisor, I would like to thank Prof. Nick Koudas, for his detailed and excellent advice towards finishing the thesis. Prof. Koudas has influenced me greatly with his motivation and enthusiasm to research.

I would also like to thanks Prof. Parke Godfrey for reviewing the thesis word for word and providing lots of valuable comments. The rigorous attitude of Prof. Godfrey has always inspired me to do better works.

I offer my gratitude to my thesis examination committee members for providing me with insightful suggestions to improve the thesis.

Last but not least, I would like to thank my parents for taking care of and supporting me. This work would not have been possible without their endless love

and affection. I would like to dedicate this work with you.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

The prevalence of mobile devices, social media, ambient wireless connectivity, and associated positioning technologies have made it possible to record digital traces at an unprecedented rate. Such traces correspond to location sharing through social apps, handshaking with WiFi hot-spots (recording the wireless chip's MAC address or other characteristics of a device in proximity of a WiFi network [2]), connecting to cellular stations via a mobile device, and many other passive/active location capturing scenarios. This has given rise to an abundance of *digital traces.* Such traces reveal the presence history of diverse sets of *entities*—depending on the application—which includes humans, devices, etc. At a high level, any digital trace takes the form of a tuple, $\langle entity, location, timestamp \rangle$, recording that an entity (e.g., a person) was present at a physical location (e.g., a restaurant) for the

[2]The WiFi protocol reveals to the access points the MAC address of any WiFi enabled device in the vicinity of the network, even if the device is not connected to the network.

indicated timestamp. Typically location corresponds to physical locations which exhibit a hierarchical structure that is known *a priori* (e.g., city - district - street - building), and the timestamp is discretized to a tunable atomic unit such as an hour or a minute, depending on the application. For example, the tuple $\langle$Tom, W London, 10 $a.m.\rangle$ represents the fact that Tom was at the W London hotel during the hour starting at 10 $a.m.$

A challenging task is to identify the entities that are closely associated with a given query entity using their digital traces. Intuitively two entities are associated if there exists a large overlap between their digital traces. Numerous definitions for what constitutes an overlap are possible; for example, a large overlap in the locations followed by an overlap (proximity) of associated timestamps. Thus, if two entities were present at W London at 10 $a.m.$, they are associated. Similarly if two entities are present at W London, one at 10 $a.m.$ and the other at 11 $a.m.$, they are still associated but possibly less so than the previous two entities. Alternatively, one may take into account the spatial proximity of locations to define association in addition to the timestamps. Thus, if two entities are present in the same postal code at the same time, they are associated as well, but probably less so than two entities appearing at the same specific location, say a restaurant, at the same time. It is evident that there are numerous ways to define the degree of association between entities given their digital traces, which are probably application dependent. As such, we adopt a generic approach and define a class of functions that have generic properties to quantify association. All our subsequent developments in this work hold for this generic class of functions sharing such properties (see Section 3.2).

1.2 Application scenarios

Given a suitable function to quantify association, we are interested to identify the top-k associated entities to a given query entity. Supporting efficient processing of such queries over a large volume of digital traces enables a variety of applications. For example, this assists law enforcement authorities to identify individuals closely related to a person of interest. In most cases, the behavior patterns of the criminals before, during, and after the crime are highly different from those of ordinary people. Therefore, given a target person, we may be able to identify the criminal gang by finding those people having a large overlap in digital traces with this person. Also, the techniques developed herein can help to build location-based recommender systems using user associations. A large overlap in the digital traces of two people implies that they either know each other very well, or share very similar living patterns. Therefore, we can make recommendation and promotion to a certain user based on the choices of his/her associated users, which will highly improve advertising precision.

This research is motivated by our work with authorities enabling post-crime investigation using location data collected from mobile devices. Such information is crucial to prove the joint presence of suspects at crime scenes and also their association before and after the crime. For this specific reason, the main interest is to assess association across large sets of digital traces, corroborating the association before and after specific events. For example, in our ongoing work with a large national telecommunications provider on this problem, we work with a dataset involving 30M individual devices with an average of 650K detections by WiFi hotspots each; in addition, each device is present, on average, at 500 locations during the time ranges of interest for the queries being asked. In a different context, the techniques developed herein enable marketers to identify groups of individuals with related behavior in the physical world for more effective advertising. As an example, marketers may use associated behavior inside a shopping center (as reported by triangulated WiFi signals of mobile devices) to identify families or couples who are of prime interest for specific types of location-based marketing. Once again association across a large collection of traces enforces a closer bond between the entities involved as opposed to chance encounters.

1.3 Methodology

In the target applications we engage with, the number of entities is in the multiple millions while the number of digital traces is in the billions. As such, techniques that compare the query entity to all other entities are inefficient.

Aiming to provide fast query response times, we propose a suite of indexing structures and algorithms for this problem. At a high level, we consider entities as points in a high-dimensional space with $\langle location, timestamp \rangle$ pairs of each entity corresponding to a dimension. The basic idea of our approach consists of two parts: (1) transforming an entity's digital traces into a lower-dimensional space for more efficient computation; this lower-dimensional representation also allows the ordering of entities along each dimension, making it possible to build an index structure; (2) constructing an index that groups the entities in a hierarchical fashion using this lower-dimensional representation, so that associated entities tend to appear in the same group, enabling effective pruning.

We adopt MinHash [6], a widely-employed technique for duplication detection, to compute a signature for each entity (details of MinHash can be found in Section 2.3). When a spatial hierarchy over locations is present we do so at each level of the spatial hierarchy, resulting in a list of signatures for each entity. The size of each signature depends on the number of hash functions used and can be considered as the dimensionality of this lower-dimensional space. It may be tuned for a performance/cost trade-off. The list of signatures for each entity are subsequently indexed with a tree structure. The guiding principal of this index is to group the entities based on their signatures at each level such that for a given query entity, only a small portion of the branches in the tree have to be explored; the remaining branches are guaranteed not to contain the top-k results and can thus be safely discarded. This is made possible by assessing a signature for each group of entities at a tree node, which serves as the basis for estimating bounds on the association between the query entity and the entities in the subtree rooted at this node. The index naturally supports incremental updates. We then develop algorithms to process top-k queries using the index.

We also develop and present a model for mobility which we validate against real data at scale. using this model, we subsequently present a thorough analysis of the pruning effectiveness of the proposed method. Our results reveal that the proposed technique has strong pruning capabilities, limiting the scope of search to only a small portion of all available entities. We also validate our model for pruning effectiveness against real mobility traces and demonstrate its accuracy both analytically and experimentally.

Experiments are conducted on both synthetic and real datasets at scale to (1) compare the performance of the proposed method against that of baseline methods, and (2) conduct a sensitivity analysis of the proposed method with respect to varying parameters of interest (e.g., number of hash functions, data characteristics). Our results demonstrate orders of magnitude performance improvement over a designed baseline approach.

1.4 Contributions and outline

In summary we make the following contributions:

- Motivated by real-life applications with telecommunications providers, we formally define the problem of top-k query processing over digital traces. To the best of our knowledge, our work is the first to address this important problem.

- We develop a suite of novel data transformation and indexing techniques as well as the corresponding search methodologies, which demonstrate strong pruning capabilities, allowing us to focus the search only on a small portion of the space.

- We analytically and experimentally quantify the pruning effectiveness of our methods using models of human mobility patterns.

- We perform extensive experiments on both real and synthetic data at scale to study thoroughly the performance of the proposed method, confirming its effectiveness and superiority over other approaches across a variety of settings.

The rest of the paper is organized as follows. Chapter 2 provides an overview of related work, including querying trajectory, time series analysis, hashing techniques, frequent pattern mining, and research regarding digital traces. Chapter 3 formally defines the problem of top-k query over digital traces and other assist terms. Chapter 4 describes the approach, including the data transformation principle, the data organization technique, and the complexity of indexing. In Chapter 5, we prove the early termination condition of the proposed approach and give the search algorithm. In Chapter 6, we analytically quantify the pruning effectiveness of the approach. In Chapter 7, we present experiment results across a variety of settings, and Chapter 8 concludes this paper.

2 Literature Review

In this chapter, we give brief introduction of works in the communities of: trajectory query, which deals with real world trajectories produced by human, vehicle, etc.; time series analysis, which focuses on knowledge discovery from temporal data; MinHash, which is widely employed in duplication detection tasks; frequent pattern mining, which aims to find frequently co-occurred items; and digital traces analysis, which deals with digital traces produced by human as this thesis does.

2.1 Trajectory query

Querying trajectories, as querying digital traces, deals with the spatio-temporal data produced by human in daily activities. Both works focus on the movement and presence of real world objects and reveal potential relations among these objects from their trajectories/digital traces. There are two branches of existing research on querying trajectories, namely, nearest neighbor queries [1, 3, 4, 7, 8, 9, 13, 16, 20, 24, 26, 30, 32, 36, 39], and top-k queries [2, 12, 27, 31, 35, 37, 40, 43, 44, 46, 47].

We give detailed introduction of both branches together with some representative works in the following.

2.1.1 Nearest neighbor query

Similar to the problem proposed in this paper, nearest neighbor query finds trajectories and objects in proximity, and thus spatial distance is one of the dominant factors in determining the similarity/association between objects and trajectories in nearest neighbor search. In this section we introduce some influential works in the area of nearest neighbor query on trajectories, and differentiate our work from trajectory similarity query.

Guting et al. [16] propose 3D-R-Tree for efficient k-nearest neighbor search over moving objects, which is built to obtain coverage function and thus enable effective pruning. The authors then design a filter-and-refine algorithm to evaluate kNN query.

Tang et al. [39] investigate the problem of k Nearest Neighboring Trajectories (kNNT) search, which finds the k trajectories with the minimal aggregated distance to a set of query points. The authors use a structure called global heap to retrieve candidate trajectories and design a candidate-generation-and-verification strategy to facilitate efficient and scalable searching.

Fang et al. [13] study the problem of kNN join; i.e., given two sets of trajectories

$\mathcal{R}$ and $\mathcal{S}$, for each trajectory in $\mathcal{R}$, return its k nearest neighbors in $\mathcal{S}$. The authors propose a parallel solution framework and a novel bounding technique to accelerate query processing.

Sharifzadeh et al. [36] propose VoR-Tree, a combination of R-Tree and Voronoi diagram, which demonstrate high effectiveness in both locating and exploring search region. VoR-Tree improves the efficiency of solving various Nearest Neighbor queries by reducing the I/O complexity to a large extent.

Works on kNN search on trajectories mainly focus on spatial closeness, without considering the influence of spatial topology, such as hierarchy, on measuring the association among trajectories and the corresponding entities. Consequently they lack the ability of inferring the association degree between entities from their trajectories.

2.1.2 Top-k query

Different from nearest neighbor search which finds kNN for all objects, top-k queries on trajectories deal with a particular query object, which is more relevant to the task defined in this thesis. In addition to spatial distance, metrics used in Top-k queries also consider other factors such as duration, activity type, etc., which facilitate quantifying the association between entities from multiple aspects as is desired in our problem. Here we summarize several pieces of work most related to our problem.

Ma et al. [27] design p-distance to measure the dissimilarity between uncertain trajectories which are common in real world, and define top-k similarity query (KSQ) on uncertain trajectory. The authors design a grid-based data structure called UTgrid to index uncertain data and develop algorithm on UTgrid which facilitates effective pruning.

Frentzos et al. [14] consider both spatial distance and duration when measuring the similarity between trajectories and propose definite integral based metrics for k-Most Similar Trajectory (k-MST) search over moving object databases. The metrics used in the paper, as the authors argued, are computationally heavy, and thus they propose a R-Tree based approximation approach for efficient query processing.

Wang et al. [44] study Exemplar Trajectory Query (ETQ) which takes the input of a set of locations and an activity description, and return k trajectories with highest spatial and textual similarity to the query conditions. The authors introduce an incremental pruning baseline and design a two-level gap-based optimization algorithm to reduce computational cost.

Metrics used in top-k query on trajectories are mostly based on sequence distance (e.g., Longest Common SubSequence [42]), or Time Series distance (e.g., Dynamic Time Warping [34]), which either ignore time dimension or assume trajectories are aligned in time, which are not guaranteed to satisfy the constraints proposed for association degree measures in Section 3.2.

2.2 Time series analysis

Time series analysis focuses on knowledge discovery from time-oriented data, giving birth to a direction of studies which quantifies the similarity/association between entities from their temporal overlap patterns. Time series analysis benefits our research as the temporal co-occurrence is considered as one of the dominant factors in determining the association between entities. In this section we briefly discuss the metrics commonly used in time series analysis and how we can import such metrics to define association degree between entities given their digital traces.

Bozkaya et al. [5] modify traditional Longest Common SubSequence (LCSS) metric to support distance measure between sequences with different length and design an index which is built on the length the sequences and relative distances between sequences to accelerate query processing. The major drawback of the metric is that it depends on exact equality match on real values and is thus not suitable for similarity search in real world environment.

Chen et al. [7] propose a novel distance function, Edit Distance on Real Sequence (EDR), which is robust to noise such as sensor failures, disturbance signals, and different sampling rates. The main idea of the metric is quantizing the distance between a pair of elements to 0 and 1 and define distance on the minimal edit operations to transfer one sequence to another. The metric, as argued in the paper, demonstrate superior robustness and accuracy for similarity search on moving objects.

Dynamic Time Warping (DTW) is another popular distance metric in time series analysis. Vlachos et al. [41] use DTW as rotation invariant distance measures for trajectories, which is robust under the transformation operations desired in the paper and very efficient to compute. Little et al. [25] use path and speed curves to represent the motion trajectories and use DTW to measure the distance between trajectories extracted from video.

Time series distance metrics, although provide plenty of ways to quantify the similarity between the evolving patterns of trajectories, do not give enough consideration to the characteristics of moving objects in a hierarchical spatial environment, which limits the chance of direct employment of time series metrics to measure the association between entities in our task.

2.3 MinHash

We also investigate duplication detection problems over image [10] and web [28] datasets, which seem not closely related to the topic but do provide a means to address the problem. The basic detection strategy is to model an image/document as a set and pixels/words as tokens, and employ token-based hashing functions to map sets into a hashing space where similar sets are placed closer to each other. The technique is preferred in our problem for its efficiency to deal with large volume data. In this section, we introduce MinHash, a widely-adopted hashing technique for duplication detection under Jaccard Similarity.

The MinHash technique uses a family of m hash functions to assign each set a signature, with each signature consisting of m hash values. The similarity between two sets is estimated by comparing the corresponding signatures.

Suppose we use m hash functions $h_1, h_2, \cdots, h_m$ to do MinHash. For a set $S = \{e_1, e_2, \cdots, e_n\}$, the signature of S, SIG_S, is calculated in a following way.

1. Initialize SIG_S as an m-length array, with each value set to positive infinity.

2. For each element $e \in S$, do the following.

    - For each hash function h_i $(1 \leq i \leq m)$, if $h_i(e) < \mathrm{SIG}_S[i]$, $\mathrm{SIG}_S[i] := h_i(e)$; else, do nothing.

We use an example to illustrate the process [21]. Assume four sets $S_1 = \{0, 3\}$, $S_2 = \{2\}$, $S_3 = \{1, 3, 4\}$, $S_4 = \{0, 2, 3\}$, and two hash functions $h_1 = x + 1 \mod 5$, $h_2 = 3x + 1 \mod 5$. Initially, we set the table as follows.

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

First, we consider the column of S_1. The first element in S_1 is 0, we have $h_1(0) = (0+1 \mod 5) = 1 < \infty$,and $h_2(0) = (3 \times 0 + 1 \mod 5) = 1 < \infty$, so we update the table to the following.

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	∞
h_2	1	∞	∞	∞

The next element in S_1 is 3, we have $h_1(3) = 4$, which is larger than the current h_1 value of S_1, so the value remains unchanged; Similarly, we have $h_2(3) = 0 < 1$, so we update h_2 value of S_1 to 0. The signature table is now

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	∞
h_2	0	∞	∞	∞

We repeat the process for each set, and finally obtain the signature table

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

We can now estimate the Jaccard similarity between sets using the signature table. Consider S_1 and S_4 as an example. Since $h_1(S_1) = h_1(S_4) = 1$, $h_2(S_1) = h_2(S_4) = 0$, the similarity between S_1 and S_4 is thus estimated as 1, while their true Jaccard Similarity is 2/3.

Although signature table helps to accelerate calculating the Jaccard Similarity between two sets, to find duplication sets to a given query set still requires estimating the Jaccard Similarity between the query set and each other set. To further reduce computation cost, the signature table is partitioned into *bands* by rows. Assume that we use m hash functions to build the signature table, and the table is partitioned into b bands, then each band contains m/b consecutive rows. Let $B_i(S_q)$ ($i \in [1, b]$) be the list of hash values of set S_q at rows contained by band i (values are ordered by row number), then set S_p becomes a candidate of duplication sets to S_q iff $\exists j \in [1, b]$, s.t. $B_j(S_q) = B_j(S_p)$.

In above example, assume that we partition the signature table into 2 bands, then each band contains one row. When finding duplication sets to S_1, we only retrieve sets S_3 and S_4 as candidates as S_2 equals to S_1 in neither bands.

Assume that the real Jaccard Similarity between set S_p and the query set is s, then the probability of S_p becoming a candidate is $1 - (1 - s^{m/b})^b$. Generally, the more hash functions, the lower the false positive, and thus the more efficient the algorithm.

MinHash estimation is guaranteed to be $(d_1, d_2, 1 - d_1, 1 - d_2) - \textit{sensitive}$ [21]; i.e. satisfies the following properties:

- If $1 - J(S_p, S_q) \leq d_1$, then the probability that S_p becomes a candidate is at least $1 - d_1$;

- If $1 - J(S_p, S_q) \geq d_2$, then the probability that S_p becomes a candidate is at most $1 - d_2$,

where S_q is the query set, $J(S_p, S_q)$ denotes the Jaccard Similarity between S_p and S_q, $d_1, d_2 \in [0, 1]$.

Multiple variants of MinHash have been proposed to improve the efficiency and accuracy [18, 19, 22, 23, 45]. However, as is clear from above discussion, MinHash techniques are mainly used in the approximation task and are restricted to Jaccard-based metrics. In our approach, we modify the strategy of building and comparing signatures to provide exact answers and also generalize the technique to other similarity measures. Details are given in Section 4.2.

2.4 Frequent pattern mining

Frequent pattern mining aims to discover frequently co-occurred items from transaction database. Researchers have also imported frequent pattern based techniques to trajectory analysis tasks; e.g., Giannotti et al. [15] define trajectory pattern to analyze the trajectory of moving objects, Monreale et al. [29] mine trajectory patterns for next location prediction. Although techniques like FP-growth can be borrowed as a grouping approach of entities, the digital traces among population, as will be discussed in Section 7.2, show low degree of frequent patterns, which limits the pruning effectiveness of frequent pattern based approaches.

2.5 Digital traces analysis

A few pieces of work in recent years also deal with digital traces of human beings [17, 33]. Digital traces in their context, however, mainly refer to the records produced by digital devices on the Internet, such as emails, twitter posts, which are not associated with spatial-temporal presences and thus share little semantic similarity with the digital traces proposed in this paper.

3 Preliminaries

In this chapter, we define the terms that are required for the subsequent discussion, and formally define the problem of top-k query processing over digital traces.

3.1 Terminology

The locations we consider are spatial and thus exhibit a hierarchical structure (e.g., city - district - street - building). We assume that a description of the hierarchical structure of locations is available via a tree structure (referred to as *sp-index*) that organizes locations from coarsest to finest. Nodes in this tree are referred to as *spatial units*.

To ease notation, we assume that the spatial units at the same level of the sp-index are non-overlapping. We label the levels of spatial units from 1 (for the root of the tree) to m (for the lowest level in the tree). For a spatial unit l, we use parent(l) to denote the parent unit of l on the sp-index.

At the lowest level of the tree are *base spatial units*, the atomic locations in digital traces at which entities can be present. Examples of a base spatial unit include supermarkets, restaurants. All base spatial units form a set $\mathcal{L}$.

We assume that timestamp is discretized in base temporal units (e.g., hour). The combination of a base temporal unit and a base spatial unit is referred to as a *spatial-temporal cell* (or ST-cell). We use the associated base temporal unit and base spatial unit to denote an ST-cell; e.g., $t_1 l_1$. An ST-cell is the atomic unit where entities can be present. All possible such combinations form an ST-cell set $\mathcal{S}$.

We formally define a digital trace associated with entity e to make it suitable for the environment of sp-index.

Definition 1 (Presence Instance). *A presence instance (PI) p of an entity is characterized by a four attribute tuple, $p = \langle e, tid, path, level, pd \rangle$, where*

- *e is the associated entity to p,*
- *tid is the id of the sp-index where p belongs (tid is necessary when multiple sp-index trees exist),*
- *$level$ is the level in the sp-index where p exists,*
- *$path = [node_1, node_2, \cdots, node_{level}]$ is the list of nodes in the sp-index on the path from the root to the node that reflects the location associated with p, and*

- *pd is a continuous time period associated with p; it is in the format* [*start time*, *end time*].

Evidently $|path|$ denotes the level where the presence instance exists in sp-index *tid*, which is denoted as *p.level* in sequel. Typically *start time* is the same as *timestamp*. In some applications, such as WiFi proximity sensing of MAC addresses, *end time* is obtained by the time of the last probe of the device MAC address to the WiFi network. In some other applications, such as social media check-ins, the end time is estimated based on the average time individuals spend in the venue (obtained from services such as Google Maps).

Definition 2 (Digital Trace). *The set of PIs associated with entity e forms the digital trace of e, $\mathcal{P}_e$.*

The overlap between the digital traces of two entities, *Adjoint Presence Instance*, is defined as follows:

Definition 3 (Adjoint Presence Instance). *Given two PIs, $p_a = \langle e_a, tid_a, path_a, level_a, pd_a\rangle$, $p_b = \langle e_b, tid_b, path_b, level_b, pd_b\rangle$, if $tid_a = tid_b$, $pd_a \cap pd_b \neq \emptyset$, then e_a and e_b form an adjoint presence instance (AjPI) $p_{ab} = \langle \{e_a, e_b\}, tid_{ab}, path_{ab}, pd_{ab}\rangle$, where:*

- $tid_{ab} = tid_a = tid_b$,

- $path_{ab} = path_a \cap path_b$, *which is the set of common ancestors of the two PIs, and*

- $pd_{ab} = pd_a \cap pd_b$, *the intersection of the two time periods.*

$path_{ab}$ denotes the set of common ancestors of the two PIs, and thus $|path_{ab}|$ equals to the level of AjPI p_{ab} in the sp-index tid_{ab}, which is denoted as $p_{ab}.level$ in sequel.

Each pair of entities, say e_a and e_b, form a set of AjPIs denoted as $\mathcal{P}_{ab}$, where $0 \leq |\mathcal{P}_{ab}| \leq |\mathcal{P}_a| \cdot |\mathcal{P}_b|$. The definition can be naturally extended to adjoint presence instances of multiple entities.

An AjPI specifies a spatio-temporal co-occurrence of two entities, and thus reveals a potential association between the entities. Such an association is defined as a function over the domain of the corresponding presence instances and adjoint presence instances of each entity pair, as outlined in the next section.

We give the following notation table to ease review.

Notation	Meaning
m	level of sp-index
e_a	an arbitrary entity
$\mathcal{E}$	all entities
l_x	an arbitrary base spatial unit
$\mathcal{L}$	all base spatial units
$l_x t_y$	an arbitrary ST-cell
$\mathcal{S}$	all ST-cells
p_a	a presence instance associated with entity e_a
$\mathcal{P}_a$	all presence instances associated with entity e_a
p_{ab}	an adjoint presence instance between entity e_a and entity e_b
$\mathcal{P}_{ab}$	all adjoint presence instances between entity e_a and entity e_b
$deg(e_a, e_b)$	the association degree between entity e_a and entity e_b
n_h	the number of hash functions
h_u	an arbitrary hash function
seq_a	the ST-cell set sequence of entity e_a
seq_a^l	the l-th set in seq_a
sig_a	the signature list of entity e_a
sig_a^i	the i-th signature in sig_a
N	an arbitrary node on the MinSigTree
SIG_N	the signature of node N
$\mathcal{PS}$	pruned set
$\mathcal{PPS}$	partial pruned set
PE	pruning effectiveness

Table 3.1: Notation table

3.2 Problem definition

One important but challenging task is to discover all entities that are closely associated with a given entity. Since there may exist many ways to quantify association, we define a generic class of scoring functions that share commonly desired properties. While the particular choice of the function varies depending on the application, our approach would apply as long as the function exhibits these generic properties.

For an arbitrary AjPI p_{ab}, the scoring function $f(p_{ab})$ has the following properties.

- Normalization; i.e., $f(p_{ab}) \in [0, 1]$.
- Total order; i.e., $f(p_{ab}) \geq f(p_{ac})$ if

  $I(I_1(p_{ab}.pd.length, p_{ac}.pd.length), I_2(p_{ab}.level, p_{ac}.level)) \geq \delta$,

  and $f(p_{ab}) < f(p_{ac})$ otherwise,

where $I \in [0, 1]$ is a function monotonically increasing on I_1 and monotonically decreasing on I_2, $I_1 \in [0, 1]$ is a function monotonically increasing on $p_{ab}.pd.length$ and monotonically decreasing on $p_{ac}.pd.length$, $I_2 \in [0, 1]$ is a function monotonically increasing on $p_{ab}.level$ and monotonically decreasing on $p_{ac}.level$, $\delta \in [0, 1]$ is a threshold. Basically, the second property gives AjPIs at finer spatial units and for longer durations a higher score. A direct conclusion from the second property is:

- $\forall e_c$, $f(p_{ab}) \geq f(p_{ac})$ if $p_{ab}.pd.length \geq p_{ac}.pd.length \wedge p_{ab}.level \geq p_{ac}.level$.

These properties capture the intuition that the association between two entities is higher when corresponding digital traces match closely at locations (i.e., appear at finer levels of the sp-index, say, at the same restaurant rather than just in the same city) or their temporal co-occurrence is longer.

Let $\mathcal{P}_{ab}$ be the set of AjPIs formed by e_a and e_b. The overall score for this set is defined as

$$F(\mathcal{P}_{ab}) = \sum_{p_{ab} \in \mathcal{P}_{ab}} f(p_{ab}), \tag{3.1}$$

which has to be further normalized to take into consideration the individual behaviors of e_a and e_b. It is evident that the AjPI involving an entity with many PIs is less interesting than that with an entity having few PIs. Therefore, we define a scoring function for individual PI p_a, which is considered as a special case of the AjPI score; i.e., $f(p_a) = f(p_{aa})$. The score for the set of PI $\mathcal{P}_a$ is:

$$F(\mathcal{P}_a) = \sum_{p_a \in \mathcal{P}_a} f(p_a) \tag{3.2}$$

Clearly, $\forall e_a, \forall e_b, F(\mathcal{P}_a) \geq F(\mathcal{P}_{ab}), F(\mathcal{P}_b) \geq F(\mathcal{P}_{ab})$.

Intuitively, closely associated entities are those who have a large presence instance overlap; i.e., more adjoint presence instances and fewer total presence instances for either entity. Thus we define the association degree between two entities

e_a and e_b as $deg(e_a, e_b)$, which satisfies the following constrains.

- Normalization; i.e., $deg \in [0, 1]$.
- Monotonicity; i.e., $\forall e_c$, $deg(e_a, e_b) \geq deg(e_a, e_c)$ if $\mathcal{P}_c \subseteq \mathcal{P}_b \subseteq \mathcal{P}_a$.
- Total order; i.e., $deg(e_a, e_b) \geq deg(e_a, e_c)$ if

  $J(J_1(F(\mathcal{P}_{ab}), F(\mathcal{P}_{ac})), J_2(F(\mathcal{P}_b), F(\mathcal{P}_c))) \geq \delta'$,

  and $deg(e_a, e_b) < deg(e_a, e_c)$ otherwise,

where $J \in [0, 1]$ is a function monotonically increasing on J_1 and monotonically decreasing on J_2, $J_1 \in [0, 1]$ is a function monotonically increasing on $F(\mathcal{P}_{ab})$ and monotonically decreasing on $F(\mathcal{P}_{ac})$, $J_2 \in [0, 1]$ is a function monotonically decreasing on $F(\mathcal{P}_b)$ and monotonically increasing on $F(\mathcal{P}_c)$, $\delta' \in [0, 1]$ is a threshold. A direct conclusion from the third constraint is:

- $\forall e_c$, $deg(e_a, e_b) \geq deg(e_a, e_c)$ if $F(\mathcal{P}_b) \leq F(\mathcal{P}_c) \wedge F(\mathcal{P}_{ab}) \geq F(\mathcal{P}_{ac})$.

The digital traces of an entity are modeled as a set of PIs in this context, and thus the association degree between entities can be considered as a type of set similarity. The association degree measure, deg, is proposed as a generalization of a large family of set similarity functions; e.g., Jaccard similarity, Dice similarity, F-score. By introducing function F which flexibly maps an AjPI into $[0, 1]$, we are able to simulate different scenarios where the association degree between entities grows linearly, super-linearly, or sub-lineraly with respect to their real set similarity (regarding digital traces). Therefore, *deg* is applicable in most applications which use the overlap of digital traces to measure the association between entities.

However, the measure proposed herein is not without limitation. First, the measure cannot be naturally extended to discover "following patterns" between entities; e.g., when entity e_a visits the same set of base spatial units as e_b, but with unknown and varying time lags. Second, the measure does not give full consideration to the characteristics of spatial units in deciding the association degree between entities. For example, AjPIs at a restaurant might be more important than at a shopping mall in measuring the association between entities. In this thesis, we focus on the general case of digital trace overlapping and will address the aforementioned limitation in our future work.

Let $\mathcal{E}$ be the set of all entities. The problem of identifying the k most associated entities (entities with the highest association degree) to a given query entity is defined as follows.

Definition 4 (Top-k Query over Digital Traces)**.** *Given a query entity e_p and association degree measure deg, the top-k query over digital traces is to return the set of entities $\mathcal{Q}_k$ such that $\mathcal{Q}_k \subseteq \mathcal{E} - \{e_p\}$, $|\mathcal{Q}_k| = k$ and $\forall e_q \in \mathcal{Q}_k$, $\forall e_t \in (\mathcal{E} - \{e_p\} - \mathcal{Q}_k)$, $deg(e_p, e_q) \geq deg(e_p, e_t)$, where $1 \leq k < |\mathcal{E}|$.*

Without loss of generality, we assume that the value of k is in a proper range such that for any entity e_q in the result set, $deg(e_p, e_q) > 0$; i.e., all returned entities must have formed AjPI(s) with the query entity. Otherwise we arbitrarily complement the result.

4 Our Approach

A brute-force approach to answer top-k queries involves computing the association degree between the query entity and all other entities. Clearly, the cost can be prohibitive, as the number of entities are often in the millions and the number of digital traces in billions in our target applications. As such, we introduce a data structure, called the *MinSigTree*, that indexes entities based on their presence instances, facilitating efficient pruning of entities to be examined during the search for top-k answers.

As a high-level overview, we first organize the PIs of each entity as a sequence of ST-cell sets. Then we construct a list of *signatures* for each entity which can be considered as summaries of the entity's PIs. Subsequently, we construct the MinSigTree that groups closely associated entities together based on their signatures.

4.1 Data representation

Digital traces have varying representations and formats. For example, raw WiFi logs representing device presence are highly different than social media check-ins. The first step is to organize the data by entity so that the presence instances of an entity at each sp-index level and the resulting association degree between entity pairs can be computed efficiently.

We build a sequence of ST-cell sets for each entity, where the length of the sequence equals the height of the sp-index, m. The sequence of sets for entity e_a is denoted as seq_a, and the i-th set in seq_a, $i \in [1, m]$, corresponding to the level i of the sp-index, is denoted as seq_a^i.

seq_a^m, for the lowest level of the sp-index, can be obtained directly from e_a's digital trace; i.e., for an ST-cell s, $s \in seq_a^m$ iff e_a is present at s. For other levels, ST-cell set seq_a^i, $i \in [1, m)$ is built from set seq_a^{i+1}. For an ST-cell $s = t_z l_x$, $s \in seq_a^i$ iff $\exists s' = t_z l_y$, s.t. $s' \in seq_a^{i+1}$ and $l_x =$parent(l_y).

Example 4.1.1. *Let L_1, L_2, L_3, and L_4 be four base spatial units, and parent(L_1)= parent(L_2)=L_5, parent(L_3)=parent(L_4)=L_6, $m = 2$. Assume that entity e_a has presence in base spatial unit L_3 at time T_1, and L_1 at time T_2, then $seq_a^2 = \{T_1L_3, T_2L_1\}$. Since $T_1L_3 \in seq_a^2$ and $L_6 =$parent(L_3), $T_1L_6 \in seq_a^1$, similarly, $T_2L_5 \in seq_a^1$. Finally we have $seq_a^1 = \{T_1L_6, T_2L_5\}$.*

The ST-cell set sequence not only records the PIs of a single entity at any level of the sp-index, but reflects the AjPI between entities as well. If entities e_a and e_b form AjPIs at level i, then $seq_a^i \cap seq_b^i \neq \emptyset$.

4.2 Data organization

Although ST-cell set sequences facilitate the direct retrieval of PIs of each entity at any level, a brute-force approach would have to explore the whole search space of all entities to identify the top-k answers, which is still too expensive. We thus propose to group entities based on their common ST-cells to allow efficient pruning of the search space. Note that the number of ST-cells in which an entity is present could vary vastly from entity to entity (e.g., one short occurrence vs. frequent and prolonged visits to multiple locations). If we consider each ST-cell as a dimension, conceptually all entities can be considered as bit vectors in a very high-dimensional space where each bit indicates whether that entity is present in the ST-cell. However, if they are physically treated as such, the storage and computation cost would be prohibitive when the number of ST-cells is large. To enable more effective indexing, we employ a family of hash functions to map ST-cell sets into a lower-dimensional space. This is achieved by assigning each entity a *signature* at each level, with each value in the signature acting as a summary of the entity's PIs, and then grouping entities by their signatures.

4.2.1 Signature

We use n_h hash functions to map an ST-cell set into a vector in a n_h-dimensional space, where each element of the vector is a hash value in range $[0, |\mathcal{S}| - 1]$. Since each entity is associated with an ST-cell set sequence of length m, for an arbitrary entity e_a ,we obtain m vectors, which form a list of signatures, sig_a, and we use sig_a^i to denote the i-th signature in sig_a (corresponding to level i in the sp-index), and $sig_a^i[u]$ to denote the u-th hash value in sig_a^i, where $u \in [1, n_h]$.

The way we compute signatures for each entity is similar to that for MinHash [6]. A hash function h_u maps each ST-cell to a value in the range $[0, |\mathcal{S}| - 1]$. The u-th value in the signature sig_a^i corresponds to the minimal hash value produced by h_u across all ST-cells in seq_a^i; i.e., $sig_a^i[u] = \perp_u^i = \min(\{h_u(s) | s \in seq_a^i\})$.

The hash functions employed above should satisfy that, for ST-cell $s = t_z l_x$ and $s' = t_z l_y$, if l_x =parent(l_y), then $h_u(s) \leq h_u(s')$. A simple implementation of this constraint is achieved by hashing ST-cells at level-m first and assign values to ST-cells at higher levels accordingly: let $\mathcal{C}_x$ be the child spatial unit set of l_x, the above constraint is satisfied by assigning $h_u(t_z l_x) = \min(\{h_u(t_z l_c) | l_c \in \mathcal{C}_x\})$. The constraint guarantees the following property which makes signatures at different levels comparable.

Theorem 1. *For any entity $e \in \mathcal{E}$, $sig_e^i[u] \leq sig_e^{i+1}[u]$ always holds.*

The proof follows from above constraint and is omitted for brevity.

Example 4.2.1. *Consider the following hash table:*

	T_1L_1	T_2L_1	T_1L_2	T_2L_2	T_1L_3	T_2L_3	T_1L_4	T_2L_4
h_1	2	8	5	1	4	6	7	3
h_2	8	3	6	5	4	1	2	7

Table 4.1: Hash table

Assume that the four base spatial units follow relations indicated in Example 4.1.1. Let e_a, e_b, e_c and e_d be four entities with the following ST-cell set sequence: We first build sig_a^2 given $seq_a^2 = \{T_1L_2, T_2L_1\}$. Since $h_1(T_1L_2) = 5$, $h_1(T_2L_1) = 8$,

e_a	$\langle\{T_1L_5, T_2L_5\}, \{T_1L_2, T_2L_1\}\rangle$
e_b	$\langle\{T_1L_5, T_2L_5\}, \{T_1L_1, T_2L_2\}\rangle$
e_c	$\langle\{T_1L_6, T_2L_5\}, \{T_1L_3, T_2L_1\}\rangle$
e_d	$\langle\{T_1L_6, T_2L_6\}, \{T_1L_4, T_2L_4\}\rangle$

Table 4.2: ST-cell set sequence

we have $sig_a^2[1] = 5$; similarly, since $h_2(T_1L_2) = 6$, $h_2(T_2L_1) = 3$, we have $sig_a^2[1] = 3$. Therefore, $sig_a^2 = \langle 5, 3\rangle$. Subsequently, we build sig_a^1 given $seq_a^1 = \{T_1L_5, T_2L_5\}$. Since L_5 = parent(L_1) =parent(L_2), $h_1(T_1L_5) = \min\{h_1(T_1L_1), h_1(T_1L_2)\} = 2$; similarly we have $h_1(T_2L_5) = 1$, $h_2(T_1L_5) = 6$, $h_2(T_2L_5) = 3$. Therefore, $sig_a^1 = \langle 1, 3\rangle$. We build signatures for all entities and finally obtain the following signature table.

e_a	$\langle\langle 1,3\rangle,\langle 5,3\rangle\rangle$
e_b	$\langle\langle 1,3\rangle,\langle 1,5\rangle\rangle$
e_c	$\langle\langle 1,2\rangle,\langle 4,3\rangle\rangle$
e_d	$\langle\langle 3,1\rangle,\langle 3,7\rangle\rangle$

Table 4.3: Signature table

As each value in a signature is obtained by hashing all ST-cells in the corresponding set to a certain domain, it can be considered as a summary of the ST-cell set. Hash values sig_a^i enables us to determine certain facts regarding the ST-cells contained in the set seq_a^i.

Theorem 2. *For signature sig_a^i ($i \in [1, m]$) and an ST-cell s, if $\exists u \in [1, n_h]$ s.t. $sig_a^i[u] > h_u(s)$, then $s \notin seq_a^m$.*

Proof. If $s \in seq_a^m$, then $sig_a^m[u] \leq h_u(s)$. From Theorem 1 we know that $sig_a^i[u] \leq sig_a^m[u]$, and thus $sig_a^i[u] \leq h_u(s)$, which contradicts the condition. $\square$

Via Theorem 2, for a given signature sig, we can obtain a *pruned set* of ST-cells such that entities bearing sig are guaranteed not to have presence in those ST-cells. We use $\mathcal{PS}_a^i$ to denote the pruned set based on signature sig_a^i. This property will be explored in pruning the search space while computing the top-k answers.

4.2.2 MinSigTree

We next build the MinSigTree, an m-level tree structure, which groups entities sharing similar signatures together. Each node in the MinSigTree has at most n_h child nodes (with n_h being the number of hash functions used while computing signatures), each leaf node contains a set of entities, and each entity is contained in a single leaf node. If node N contains entity e_a, we consider all ancestor nodes of N to contain conceptually e_a as well to ease notation (but no physical storage is involved). For node N containing entity set $\mathcal{E}_N$, we compute a group-level signature SIG_N summarizing the PIs of all entities in $\mathcal{E}_N$.

Assuming that there is a virtual root node (at level 0), we use Algorithm 1 to build the MinSigTree.

As Step 1, we fetch the level 1 signature of every entity, $(sig_1^1, sig_2^1, \cdots, sig_{|\mathcal{E}|}^1)$, and partition these signatures into n_h non-overlapping groups. This is done in a way such that entity e_a is routed to the u-th group ($u \in [1, n_h]$) if $\forall v \in [1, n_h](v \neq u), sig_a^1[u] \geq sig_a^1[v]$; i.e., u is the position of the maximal hash value in sig_a^1 (ties are broken arbitrarily). We call u the *routing index* of the u-th group (Line 3).

Step 2 involves computing a group-level signature for each node (Lines 5—7). Assume that node N_u contains entity set $\mathcal{E}_{N_u}$. Then the signature of N_u, SIG_{N_u}, can be computed by $\text{SIG}_{N_u}[v] = \min_{e \in \mathcal{E}_{N_u}}\{sig_e^1[v]\}$, where $v \in [1, n_h]$. The newly

Algorithm 1 Building MinSigTree

```
Input: Entity set $\mathcal{E}$, signatures of all entities
Output: MinSigTree
 1: Initialization: MinSigTree root to contain all entities; root enqueued to priority queue $Q$;
 2: for $N : Q$ do
 3:     $\mathcal{G}$ = sets of entities in $N$ grouped by routing index;
 4:     for $g : \mathcal{G}$ do
 5:         $u$ = routing index of $g$;
 6:         $\mathcal{E}_g$ = entities contained in $g$;
 7:         $\mathrm{SIG}_g$ = group-level signature of $\mathcal{E}_g$;
 8:         $N_u = \mathrm{node}(u, \mathrm{SIG}_g[u], \mathcal{E}_g)$;
 9:         $N$.addChild($N_u$);
10:         if $i \neq m$ then
11:             enqueue $N_u$to $Q$;
12:         else
13:             insert $\mathcal{E}_g$ to $N_u$;
14:         end if
15:     end for
16: end for
17: return MinSigTree;
```

created nodes are then inserted as the children of the root (Lines 8—9).

The second step computes a group-level signature for each node in a way that any hash value in SIG_{N_u} is no greater than the corresponding hash values in the signatures of entities in $\mathcal{E}_{N_u}$. With signatures computed this way, we can obtain a group-level pruned set,

$$\mathcal{PS}_{N_u} = \bigcap_{e \in \mathcal{E}_{N_u}} \mathcal{PS}_e^1 \tag{4.1}$$

All entities in $\mathcal{E}_{N_u}$ are guaranteed not to have presence in the ST-cells contained in $\mathcal{PS}_{N_u}$. Note that there is no need to store the pruned set of each node, as it can be inferred from the group-level signature.

In practice, however, storing the entire signature of a node imposes space overhead. It is evident from the grouping strategy that, given a group-level signature SIG_N with routing index u, $\forall v \in [1, n_h](v \neq u)$, $\mathrm{SIG}_N[u] \geq \mathrm{SIG}_N[v]$ ($\mathrm{SIG}_N[u] \gg \mathrm{SIG}_N[v]$ when the number of employed hash functions is high). From Theorem 2 it follows that the pruned set of a signature is mainly decided by the large hash values in the signature. Thus one can materialize $\mathrm{SIG}_N[u]$ only, instead of SIG_N. This greatly reduces storage costs, at the expense of pruning effectiveness. We explore this further in Section 5.1.

Consider the signature table in Example 4.2.1. We fetch all level 1 signatures and group entities accordingly. As a result, group $N_1 = \{e_d\}$ with routing index 1, $N_2 = \{e_a, e_b, e_c\}$ with routing index 2, and $\mathrm{SIG}_{N_1} = \langle 3, 1\rangle$, $\mathrm{SIG}_{N_2} = \langle 1, 2\rangle$.

The grouping principle of Step 1 is designed in a way to prevent the group-level signature from becoming too small. For example, if e_c and e_d were to be grouped together, the group-level signature would be $\langle 1, 1\rangle$, which would not be greater than any hash values and the pruned set would thus be empty.

Now we have grouped entities at the first level of the MinSigTree based on the level 1 signatures of all entities. However, the level 1 signatures reveal only the PI patterns at the highest/coarsest sp-index level. Intuitively, entities belonging to different groups at level 1 are guaranteed not to be strongly associated, but entities belonging to the same group may still have different PI patterns at a finer-level. For example, if two people both visited New York City, but one in Manhattan and the other in Brooklyn, their PIs are different in the district level. Therefore, we need to further partition the entities based on their finer-level signatures.

For node N_u at level i, if $i \neq m$ (i.e. N_u is not at the leaf level), we fetch the level $(i+1)$ signatures of entities in $\mathcal{E}_{N_u}$ (Lines 10—11), partition $\mathcal{E}_{N_u}$ by the routing indexes, compute a signature for each new group, and add these newly created nodes as children of N_u. We repeat this process until we reach the leaf level. If an entity belongs to node N_f at the leaf level, we insert this entity to N_f (Lines 12—13).

In the above example, group $N_1 = \{e_d\}$, $N_2 = \{e_a, e_b, e_c\}$. Since $sig_d^2[2] > sig_d^2[1]$, e_d belongs to the sub-group with routing index 2; i.e. $N_{12} = \{e_d\}$. Similarly, we have $N_{21} = \{e_a, e_c\}$, $N_{22} = \{e_b\}$. Group level signatures are $\text{SIG}_{N_{12}} = \langle 3, 7\rangle$, $\text{SIG}_{N_{21}} = \langle 4, 3\rangle$, and $\text{SIG}_{N_{22}} = \langle 1, 5\rangle$. The overall MinSigTree is given in Figure 4.1.

By partitioning entities recursively at each level, each group will end up containing entities that are similar at all sp-index levels, and thus very likely to result in high association degrees with each other. In addition, the partitioning strategy guarantees the following property.

Theorem 3. *If N_a is an ancestor node of N_d, then $\mathcal{PS}_{N_a} \subseteq \mathcal{PS}_{N_d}$.*

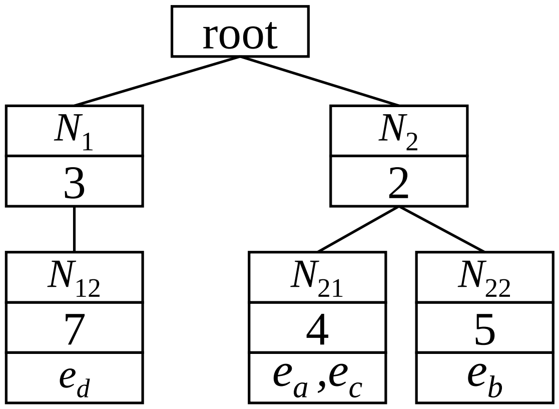


Figure 4.1: A sample MinSigTree

The proof follows from the building process and is omitted for brevity.

4.2.3 Incremental update

Similar to the building process, the MinSigTree also supports incremental update. More specifically, after building the MinSigTree, we can deal with new records of entity e by re-computing the signature on the path from the root to the leaf containing e (or the leaf to insert e if e is a new entity). The time complexity of this is linear w.r.t. the height of the MinSigTree. Updating MinSigTree is discussed in detail in Section 7.8.

4.3 Cost of index construction

We analyze the I/O and processor cost of the index construction, and provide a theoretical minimum memory usage to avoid repetitive data retrieval. Methods proposed in this chapter require digital traces to be organized by entities, but real world data have varying formats. In a system with adequate memory, we can load all records into memory and directly fetch the digital traces of a specific entity. However, when memory becomes the bottleneck, sorting the digital traces by entity becomes a necessity. We employ the well-known B-way external merge sort [11] algorithm to do the sorting. The I/O cost in the sorting process is $2N \times \lceil 1 + \lceil \log_B \lceil N/B \rceil \rceil \rceil$, where N is the total number of pages storing the digital traces and B is the number of buffer pages in memory. $\lceil 1 + \lceil \log_B \lceil n/B \rceil \rceil \rceil$ denotes the total number of passes, and in each pass, we need to read and write each page exactly once.

With access to the digital traces organized by entity, we can compute the signature list for each entity and build the MinSigTree. As described above, for each entity, we obtain its ST-cell sets at m levels, employ a family of n_h functions to map each set to a hash space, and then build an m-level MinSigTree based on the signatures of all entities. Therefore, the total processor cost in the indexing process is $O(|\mathcal{E}|Cm^3 n_h)$, where C is the average number of ST-cells that an entity has been present in.

Since the signatures of each entity are calculated independently, we can fetch one entity into memory at a time and update the MinSigTree incrementally. In order to avoid extra I/O cost, we need to keep the MinSigTree and hash functions in memory.

Theoretically, the size of the MinSigTree is $(n_h)^m$. However, since the total number of entities is $|\mathcal{E}|$, the number of leaves in the tree is bounded by $|\mathcal{E}|$. Since each node has one and only one parent node, the number of nodes at other level of the tree is also bounded by $|\mathcal{E}|$. Therefore, the size of the MinSigTree is $\min\{(n_h)^m, |\mathcal{E}| \times m\}$. The minimum memory required is thus $(\min\{(n_h)^m, |\mathcal{E}| \times m\} + n_h + C)$ to store the MinSigTree, n_h hash functions and the ST-cells of one entity.

5 Query Processing

The MinSigTree partitions entities to groups enabling an efficient search strategy for top-k query processing. We present an algorithm for top-k query evaluation using the proposed structure.

5.1 Early termination

Given a query entity e_q with ST-cell set sequence seq_q, the basic search strategy is to compute an upper bound on the association degree between e_q and each candidate node of the MinSigTree (which contains a group of candidate entities), and then progressively visit the node with the maximal upper bound until the top-k answers are identified. We outline how to compute and gradually tighten the upper bound of a node in order to prune more entities and terminate the search earlier.

We use $\mathcal{S}_q$ to denote seq_q^m, which contains all ST-cells in which e_q is present. For each node N in the search path, we determine an upper bound, UB_N, for the association degree between e_q and entities in node N, to decide whether to continue searching or to terminate.

Theorem 4. *Let $\mathcal{PS}_N$ be the pruned set of node N, and e_v be an artificial entity with ST-cell set $\mathcal{S}_v = \mathcal{S}_q - \mathcal{PS}_N$. Then $UB_N = deg(e_v, e_q)$.*

Proof. Because $\mathcal{S}_v \subseteq \mathcal{S}_q$, we have $\mathcal{P}_v \subseteq \mathcal{P}_q$, where $\mathcal{P}_v$ and $\mathcal{P}_q$ are the PI sets of e_v and e_q, respectively. Thus, $\mathcal{P}_{vq} = \mathcal{P}_v$, where $\mathcal{P}_{vq}$ is the set of AjPIs between e_v and e_q.

Let $\mathcal{E}_N$ denote the set of entities contained in N. Since $\forall e_p \in \mathcal{E}_N$, $\mathcal{S}_p \cap \mathcal{S}_q \subseteq \mathcal{S}_v$, we have $\mathcal{P}_{pq} \subseteq \mathcal{P}_v = \mathcal{P}_{vq}$. Therefore, $F(\mathcal{P}_{pq}) \leq F(\mathcal{P}_{vq})$.

$\forall e_p \in \mathcal{E}_N$, if $\mathcal{P}_v \subseteq \mathcal{P}_p$, then $F(\mathcal{P}_p) \geq F(\mathcal{P}_v)$, and thus $deg(e_v, e_q) \geq deg(e_p, e_q)$; otherwise,we have $(\mathcal{P}_v - \mathcal{P}_p) \neq \emptyset$ and $(\mathcal{P}_v - \mathcal{P}_p) \subseteq \mathcal{P}_q$, and thus $deg(e_v, e_q) \geq deg(e_p, e_q)$ according to the definition of deg given in Section 3.2. □

In practice, it is not required to compute the entire pruned set of a node; instead, it can be conducted in a more efficient way. Let u be the routing index of the node N. For an ST-cell $s \in \mathcal{S}_q$, if $h_u(s) < \text{SIG}_N[u]$, it is guaranteed that $s \in \mathcal{PS}_N$. All such ST-cell s form the partial pruned set $\mathcal{PPS}_N$, which can be used to create the artificial entity e'_v with ST-cell set $\mathcal{S}'_v = \mathcal{S}_q - \mathcal{PPS}_N$. Since we only use one hash value to build the partial pruned set, $\mathcal{S}'_v$ may be slightly larger than $\mathcal{S}_v$, which leads to a larger upper bound UB'_N. However, as the hash value at the routing index is, in general, far larger than the other hash values in the group-level signature,

UB'_N should be very close to UB_N. Introducing the partial pruned set thus leads to significant savings in storage and computation cost. In the experiment we use partial pruned sets to evaluate performance; details are given in Chapter 7.

As discussed in Theorem 3, the pruned set of a descendant node contains that of its ancestor nodes. Therefore, in a specific branch of the MinSigTree, the upper bound can be gradually tightened before we reach the leaf nodes and check the contained entities.

5.2 Search algorithm

A search algorithm based on the early termination condition is given in Algorithm 2. We initialize the result as a priority queue sorted by association degree (from high to low), and start the search from the root of MinSigTree (Line 1) whose upper bound is set to 1. We fetch the node with maximal UB in the candidate list (Line 3) and insert all its child nodes into the candidate list (Line 8). Once we reach a leaf node, we calculate the exact association degree between the query node and all entities in this node, and update the result accordingly (Lines 10—13). The process terminates when either (1) we have identified k entities, and the association degree between any of these k entities and the query entity is no less than the maximal UB of the remaining candidates (Lines 4—5), or (2) all leaves have been explored (Line 16). It is worth noting that the algorithm is applicable to all association degree measures as long as they satisfy the constraints of d specified in Section 3.2.

Algorithm 2 Top-k query processing

```
Input: MinSigTree $T$, $k$, query entity $e$, measure $deg$
Output: $k$ most associated entities to $e$
1: Initialization: Result = { }, Candidate = {root of $T$};
2: while Candidate$\neq \emptyset$ do
3:     $N$ = node with maximal $UB$ in Candidate;
4:     if Result.minKey$\geq$N.UB and Result.size==$k$ then
5:         return Result;
6:     end if
7:     if $N$ is not leaf then
8:         Candidate = Candidate $\cup${all child nodes of $N$};
9:     else
10:        $\mathcal{E}_N$ = entities contained in $N$;
11:        for $e'$ : $\mathcal{E}_N$ do
12:            $s = deg(e, e')$;
13:            Result.update($e'$, $s$)
14:        end for
15:    end if
16: end while
17: return Result;
```

Example 5.2.1. *Let us again consider the MinSigTree in Figure 4.1 as an example. We use a Dice similarity-based function as the measure of association degree:* $deg(e_i, e_j) = 0.1 \times \frac{|seq_i^1 \cap seq_j^1|}{|seq_i^1|+|seq_j^1|} + 0.9 \times \frac{|seq_i^2 \cap seq_j^2|}{|seq_i^2|+|seq_j^2|}$. *Let e_c be the query entity, and the Top-1 result is desired. As indicated in Example 4.2.1,* $seq_c^2 = \{T_1L_3, T_2L_1\}$, $h_1(T_1L_3) = 4$, $h_2(T_1L_3) = 4$, $h_1(T_2L_1) = 8$, $h_2(T_2L_1) = 3$. *We start the search from the root. For node N_1, as $3 < h_1(T_1L_3)$ and $3 < h_1(T_2L_1)$, we have $\mathcal{PPS}_{N_1} = \emptyset$, and thus we know the upper bound of N_1, $UB_{N_1} = 1$. Similarly $UB_{N_2} = 1$. The* *candidate queue is* $(1 : \{N_1, N_2\})$*. Since there is no remaining node at level 1, we dequeue* N_1*, the only child of which is* N_{12}*. As* $7 > h_2(T_1L_3)$ *and* $7 > h_2(T_2L_1)$*, we have* $\mathcal{PPS}_{N_{12}} = \{T_1L_2, T_2L_1\}$*,* $UB_{N_{12}} = 0.1 \times 1 + 0.9 \times 0 = 0.1$*, where* 1 *is the UB of the parent node of* N_{12}*, and* 0 *corresponds to the fact that both query ST-cells are contained in* $\mathcal{PPS}_{N_{12}}$*. We then dequeue* N_2*. The first child of* N_2 *is* N_{21}*. As* $4 < h_1(T_1L_3)$ *and* $4 < h_1(T_2L_1)$*,* $UB_{N_{21}} = 1$*. For node* N_{22}*, as* $5 > h_2(T_1L_3)$ *and* $5 > h_2(T_2L_1)$*,* $UB_{N_{22}} = 0.1 \times 1 + 0.9 \times 0 = 0.1$*. The candidate queue becomes* $(1 : \{N_{21}\}, 0.1 : \{N_{12}, N_{22}\})$*. We then dequeue* N_{21}*. Since* N_{21} *is a leaf node, we calculate the actual association degree between* e_a *and entities contained in* N_{21}*, and obtain* $deg(e_a, e_c) = 0.15$*. Since* $deg(e_a, e_c) > 0.1$*, the algorithm returns* e_a*.*

6 Pruning Effectiveness Analysis

In this chapter, we introduce a hierarchical mobility model significantly extending the well-established single-level individual mobility (IM) model [38]. In addition, we theoretically analyze the pruning effectiveness of our algorithms using the proposed model.

6.1 Individual mobility model

In this section we give a brief introduction of the IM model [38] which simulates human mobility in real world. In the ensuing discussion, β, ρ, γ, α, ζ, μ, and ν are all model parameters. For an entity e, the duration Δt of each PI follows

$$P(\Delta t) \sim |\Delta t|^{-1-\beta} \ (0 < \beta \leq 1), \tag{6.1}$$

which indicates that the duration of each PI follows a power law distribution; i.e., entities tend to stay for a short duration at each base spatial unit than for a long period.

When e leaves the current base spatial unit, it will either take an exploratory jump to a new base spatial unit, or return to somewhere it has previously visited. The probability of taking an exploratory jump is

$$P_{new} = \rho S^{-\gamma} \ (\gamma \geq 0, 0 < \rho \leq 1), \tag{6.2}$$

where S is the number of base spatial units visited. As e visits more base spatial units; i.e., when S increases, the probability of e taking an exploratory jump decreases.

The direction of an exploratory jump is selected randomly, and its displacement follows

$$P(\Delta r) \sim |\Delta r|^{-1-\alpha} \ (0 < \alpha \leq 2), \tag{6.3}$$

which stipulates that an entity tends to jump to some base spatial unit near its current position.

When taking a returning jump, the probability of returning to l is proportional to the number of e's previous visits to l. The visit frequency of e to its y-th most visited base spatial unit follows

$$f_y \sim y^{-\zeta} \ (\zeta \geq 0), \tag{6.4}$$

which indicates that most visits of an entity are to the few top-ranked base spatial units.

Given a duration t, the total number of distinct base spatial units visited by e is

$$S(t) \sim t^{\mu} \ (\mu \geq 0), \tag{6.5}$$

and the mean squared displacement follows

$$\langle \Delta x^2(t) \rangle \sim t^{\nu} \ (\nu \geq 0), \tag{6.6}$$

which indicates that the longer the duration, the further e will drift away from its starting position.

6.2 Hierarchical individual mobility model

The IM model in Section 6.1 describes human mobility patterns at the finest spatial level. However, AjPIs may occur at multiple levels. In this section, we give the general spatial units distribution patterns and aggregate the mobility pattern at the finest level into patterns at higher levels.

To ease analysis, we assume that the area of interest is a square with side length L, and that it is equally divided into a grid of non-overlapping cells where each cell is a square with side length L_{bsu}. Each base spatial unit corresponds to a cell in this grid. Therefore, there are $(\frac{L}{L_{bsu}})^2$ base spatial units in total. For the sp-index, the size of each spatial unit (i.e., the number of base spatial units contained therein) and the structure of the tree depend on two parameters:

- *width*; i.e., the number of nodes at each level; and
- *relative density*; i.e., the relative sizes of nodes at the same level.

Intuitively, there are more spatial units at a finer level in the tree. Therefore, we assume that the width parameter follows a power law distribution w.r.t. level; i.e.,

$$W_l = Q \cdot l^a, \tag{6.7}$$

where $l \in [1, m]$ is the level, a is a tunable parameter, and $Q = (\frac{L}{L_{bsu}})^2/m^a$ serves as a normalization factor.

In most cases, the nodes at the same level have varying sizes; e.g., business districts usually have more buildings than rural areas. Therefore, we use the following power law distribution to model the relative sizes of nodes at level l:

$$D_l^i = W_l \cdot R \cdot i^b, \tag{6.8}$$

where $i \in [1, W_l]$ is the index of nodes at level l, b is a tunable parameter, and $R = 1/\sum_{i=1}^{W_l} i^b$ is a normalization factor.

We validate the applicability of distribution proposed in Equation (6.7) and (6.8) with real world Point-of-Interest data. Details are given in Section 7.1.

With parameters L, L_{bsu}, a and b, we can obtain the number of spatial units and also the size of each spatial unit at any level. Next we demonstrate how distributions introduced in Section 6.1 modeling mobility at the finest level can be extended and supplemented with other necessary distributions to derive a hierarchical mobility model.

Let U be a spatial unit at level l which contains a set of base spatial units $\mathcal{S}_U$. An exploratory jump of an entity takes place when (1) the entity jumps to a new base spatial unit; and (2) the new base spatial unit is contained in a spatial unit previously not visited, at level l. The probability of the first condition is given in Equation (6.2); the probability of the second condition, referred to as P_{out}, can be computed by

$$P_{out}(U) = \frac{n^U_{visited}}{n^U_{reachable}} \sum_{s \in \mathcal{S}_U} \frac{1}{|\mathcal{S}_U|} H(s), \tag{6.9}$$

where $n^U_{reachable}$ denotes the number of spatial units within one jump's distance from U, $n^U_{visited}$ denotes the number of spatial units visited among these reachable ones, s is a base spatial unit in $\mathcal{S}_U$, and $H(s)$ denotes the probability of jumping outside U from s. It is evident that $H(s)$ is a function of the distance from s to the boundary of U as well as the jump distance distribution given in Equation (6.3). Therefore, the probability of taking an exploratory jump to a new spatial unit, P'_{new}, is

$$P'_{new}(U) = P_{new} \times P_{out}(U) \tag{6.10}$$

Since spatial units at higher levels have varying sizes and ranges, it is essential to derive the probability of an entity having visited unit U (the size of which is $|\mathcal{S}_U|$) within time t:

$$P_U(t) = \frac{|\mathcal{S}_U|}{|\mathcal{S}|} + \sum_{U'} M(U, U', t), \tag{6.11}$$

where $\mathcal{S}$ denotes the set of base spatial units, U' denotes some spatial unit at level l s.t. $U \neq U'$. To derive this probability we have to consider two cases: the starting position of the entity is within U or it is not. The probability of the former case is $\frac{|\mathcal{S}_U|}{|\mathcal{S}|}$. For the latter case, the starting position can be within any other spatial unit, U'. $M(U, U', t)$ describes the probability of an entity starting from unit U' having visited U within time t, which can be inferred by the mean square displacement distribution over time t given in Equation (6.6).

The visit frequency of an entity to its y-th most visited base spatial unit is given in Equation (6.4). At higher levels, the visit frequency rank, y, reflects not only personal preference, but also unit characteristics: spatial units containing more base spatial units are likely to be top-ranked. Therefore, we can safely assume that the visit frequency follows the same distribution at higher level, where y now describes the rank of the visit frequency to a particular spatial unit.

6.3 Analysis of pruning effectiveness

The model proposed in Section 6.2 enables us to simulate the movements of entities, estimate the overlap between the digital traces of any entities at all levels, with which we can calculate the expected association degree between an entity and its k most associated entities, d_e. Thus we can discard all branches whose upper bound is smaller than d_e. With more branches discarded, answering the query will be more efficient. Here we formally define pruning effectiveness (PE):

Definition 5 (Pruning Effectiveness)**.** *Given a set of entities $\mathcal{E}$, a query entity e, an association degree measure deg, and a searching strategy S, if S accurately answers a top-k query w.r.t. $\mathcal{E}$, e, and deg by checking entities in set $\mathcal{E}'$ ($\mathcal{E}' \in \mathcal{E}$) only, then the pruning effectiveness of S is $\frac{|\mathcal{E}'|-k}{|\mathcal{E}|}$.*

The average PE of is obtained averaging the PE of the top-k query answers over multiple entities.

Evidently, the UB of a child node on the MinSigTree cannot be larger than that of its parent nodes. Therefore, we can estimate PE by computing the percentage of leaf nodes on MinSigTree whose UBs are larger than d_e.

Suppose that the total number of base spatial units is n and the duration is

t, the range of hash functions is thus $[0, n \times t - 1]$. For entity e_a with ST-cell set sequence seq_a and signatures sig_a, the probability of $sig_a^m[u] = i$ is

$$p(sig_a^m[u] = i) = \sum_{x=1}^{|seq_a^m|} C_{|seq_a^m|}^x (\frac{1}{n \times t})^x (\frac{n \times t - i}{n \times t})^{|seq_a^m| - x} \tag{6.12}$$

The condition of $sig_a^m[u] = i$ is that, $\exists \mathcal{S}_a \subset seq_a^m$, $\mathcal{S}_a \neq \emptyset$, s.t. $\forall s \in \mathcal{S}_a$, $h_u(s) = i$, and $\forall s' \in seq_a^m - \mathcal{S}_a$, $h_u(s') > i$. We assume that $|\mathcal{S}_a| = x$, the probability of which is $C_{|seq_a^m|}^x [1/(n \times t)]^x$, then all remaining ST-cells take hash values larger than i, the probability of which is $[(n \times t - i)/(n \times t)]^{|seq_a^m| - x}$. By grouping entities with the MinSigTree, the signature of a node N, SIG_N, satisfies $p(\text{SIG}_N[u] = i) \approx p(sig_a^m[u] = i)$ (equal when N only contains e_a).

Let r be the routing index of N, then the probability of $\text{SIG}_N[r] = i$ is

$$p(\text{SIG}_N[r] = i) = \sum_{x=1}^{n_h} C_{n_h}^x p(\text{SIG}_N[u] = i)^x p(\text{SIG}_N[u] < i)^{n_h - x},$$
$$p(\text{SIG}_N[u] < i) = \sum_{x=0}^{i-1} p(\text{SIG}_N[u] = x) \tag{6.13}$$

With the knowledge of $p(\text{SIG}_N[r] = i)$ we can estimate the value distribution of all leaves. Assume that range $[0, n \times t - 1]$ is divided into n_r consecutive equal-sized sub-ranges R, then we use $V[j]$ to denote the percentage of leaves whose value on the routing index is bounded by $R[j]$, $0 \leq j < n_r$.

Let n_c be the minimal number of ST-cells shared by entities with association degree larger than d_e. For node N, if $\exists \mathcal{S}_{ap} \in seq_a^m$, $|\mathcal{S}_{ap}| \geq n_c$, s.t. $\forall s \in \mathcal{S}_{ap}$, $s \notin \mathcal{PS}_N$, then N cannot be discarded.

Since hash functions are selected randomly, the hash values of all ST-cells are independent. Suppose that $\text{SIG}_N[r]$ is bounded by $R[j]$, then the probability that N cannot be discarded is

$$q(R[j]) = \sum_{x=n_c}^{|seq_a^m|} C_{|seq_a^m|}^{x} (\frac{n \times t - 1 - R[j]}{n \times t - 1})^x (\frac{R[j]}{n \times t - 1})^{|seq_a^m| - x} \tag{6.14}$$

PE can thus be calculated with the following equation:

$$PE = \sum_{j=0}^{n_r} V[j] q(R[j]), \tag{6.15}$$

where $V[j]$ denotes the percentage of leaf nodes with signatures bounded by $R[j]$; and $q(R[j])$ is the probability that at least n_c ST-cells in set seq_a^m take hash values greater than $R[j]$, which is also the probability that a node with signature bounded by $R[j]$ cannot be discarded, as discussed before Equation (6.14). Therefore, $V[j]q(R[j])$ corresponds to the percentage of leaf nodes with signatures bounded by $R[j]$ in the MinSigTree that cannot be discarded; if we sum up all $V[j]q(R[j])$ values for $j \in [0, n_r]$, we obtain the overall pruning effectiveness.

With Equation (6.15) derived from above discussion, we can estimate the pruning effectiveness of the proposed approach given a set of mobility parameters and spatial distribution parameters. We compare the theoretical pruning effectiveness with the measured pruning effectiveness over two datasets in Section 7.7.

6.4 Scalability

In this section, we briefly discuss the scalability of the approach in terms of indexing time, pruning effectiveness, and query processing time.

The total cost of the indexing step consists of IO cost (scanning the data once to build ST-cell set sequences) and processor cost (computing signatures and building the MinSigTree). The time of the scanning evidently grows linearly with the data volume. The processor cost, as discussed in Section 4.3, is $O(|\mathcal{E}|Cm^3n_h)$, where $\mathcal{E}$ is the entire set of entities, C is the average number of ST-cells an entity has presence in, m is the height of the sp-index, and n_h is the number of hash functions. The values of $|\mathcal{E}|$, C, and m are data dependent. It is clear that the processor cost grows linearly with $|\mathcal{E}|$ and C, but super-linearly with m. However, in practice, depending on the location distribution of the explored area, we can expect that m, the height of sp-index, always takes small values; i.e., values ranging from 3 to 5.

We have proven in Section 6.3 (Equation (6.12) to (6.15)) that the pruning effectiveness depends the number of hash functions only and is independent of data volume related factors; i.e., $|\mathcal{E}|$ and C mentioned in the last paragraph. Therefore, the size of the data has no effect on the pruning effectiveness; i.e., the approach is pretty scalable in terms of pruning effectiveness.

The query processing time cost depends on both the pruning effectiveness and the data volume. Since the pruning effectiveness is indifferent to data volume, it is straightforward to conclude that the query processing time grows linearly w.r.t. data volume.

We experimentally demonstrate the scalability of the approach in Chapter 7.

7 Experiments

In this chapter, we present a thorough experimental evaluation of our approach using synthetic and real datasets, varying parameters of interest to explore the sensitivity of our proposal as well as PE trends.

7.1 Settings

Environment. The experiments are conducted on an Amazon Web Service EC2 instance, with a 30 core 2.3GHz Xeon CPU, 120GB of RAM, and ITB EBS Throughput Optimized HDD (maximal throughput 1,750MiB/s). The programming language is Java (version 1.8.1).

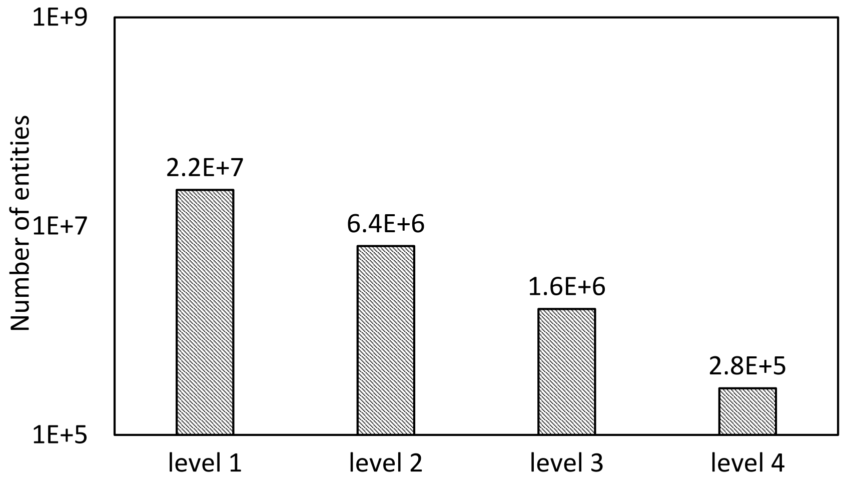
Datasets. We employ both real and synthetic datasets in our evaluation. Synthetic data are used as it is easy to vary parameters for sensitivity analysis. The synthetic dataset (referred to as SYN in the sequel) is generated by the hierarchical IM model in Chapter 6 with varying values of the parameters α, β, γ, ζ, ρ, a, b and m. Unless otherwise specified,we set $\alpha = 0.6$, $\beta = 0.8$, $\gamma = 0.2$, $\zeta = 1.2$,

$\rho = 0.6$, which correspond to the normal mobility pattern (as per [38]), and $a = 2$, $b = 2$, $m = 4$ (a and b usually take values in the range $[1, 2]$ in real datasets[3], and 4 is the typical hierarchical level in a city). The sensitivity to these parameters governing data characteristics is evaluated in Chapter 7.4. The locations in the data are drawn from a set of 9 equal-sized sp-indexes with 250K locations in total. The data consists of the digital traces of 100M entities for a period of 30 days.

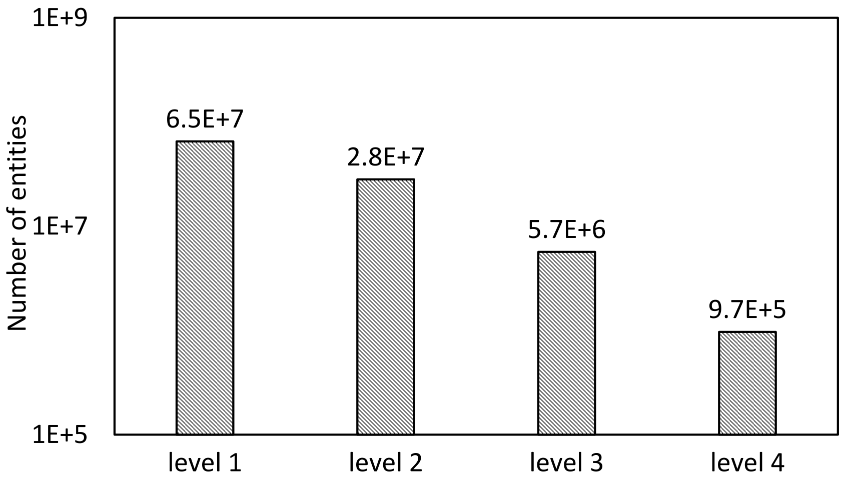
The real dataset (referred to as REAL) is a WiFi hotspot handshaking data set provided to us by a large telecommunications provider and includes 30 million mobile devices and 76,739 WiFi hotspots. The hotspots are organized into a 4-level sp-index.

The data distribution is depicted in Figure 7.1, demonstrating both data distribution across levels as well as distribution of AjPI duration. Note that the vertical axes in all these plots are in log scale. Figure 7.1(a) depicts the number of entities forming AjPIs with a particular entity at each level on REAL. Given an entity e, as shown in Figure 7.1(a), roughly 22 million entities form AjPIs with e at level 1 (two entities forming an AjPI at a finer level also form an AjPI at the coarser levels), etc. Figure 7.1(b) illustrates the same distribution on SYN. Figure 7.1(c) provides the duration distribution of AjPI at each level: roughly 20 million entities form AjPI with e at level 1 for durations shorter than 100 hours, etc.

[3]https://data.cityofnewyork.us/City-Government/Points-Of-Interest/rxuy-2muj

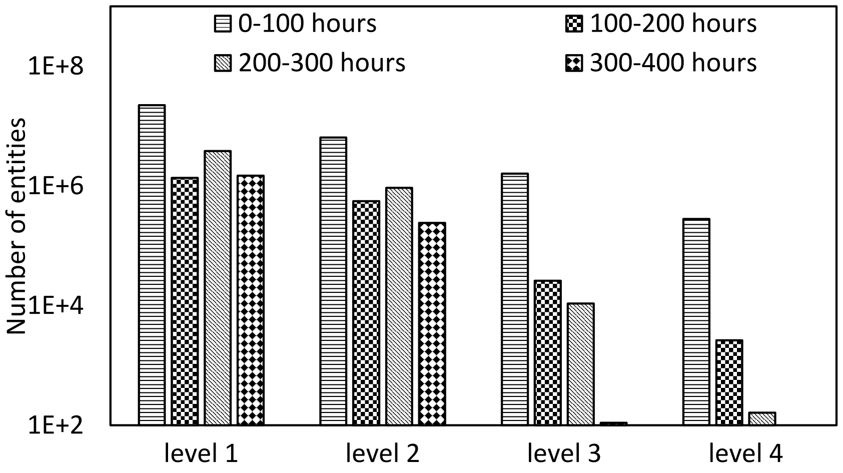


(a) REAL data

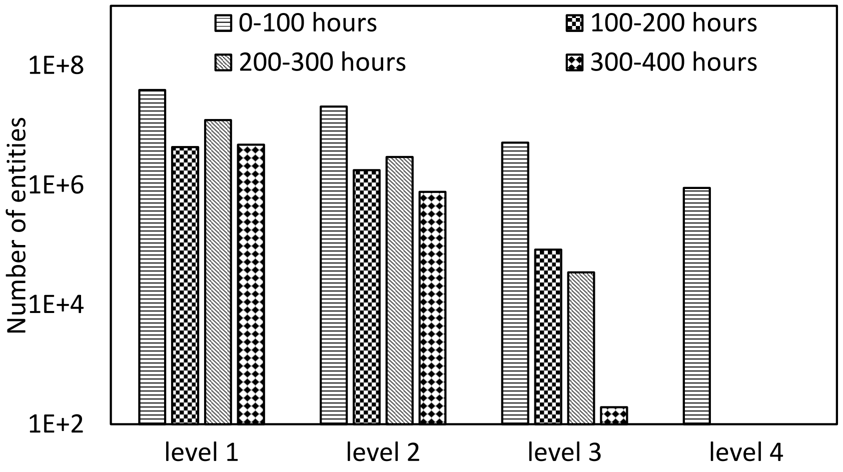


(b) SYN data

Figure 7.1: Data distribution



(c) REAL data



(d) SYN data

Figure 7.1: Data distribution

Association degree measure. There are two properties any association degree measure (ADM in sequel) must possess (as discussed in Section 3.2), namely monotonicity with respect to AjPI level and duration. We use the following extensible function as the ADM:

$$deg(e_a, e_b) = \frac{\sum_{l=1}^{m} l^u (\frac{|\mathcal{P}_{ab}^l|}{|\mathcal{P}_a^l|+|\mathcal{P}_b^l|})^v}{max}, \tag{7.1}$$

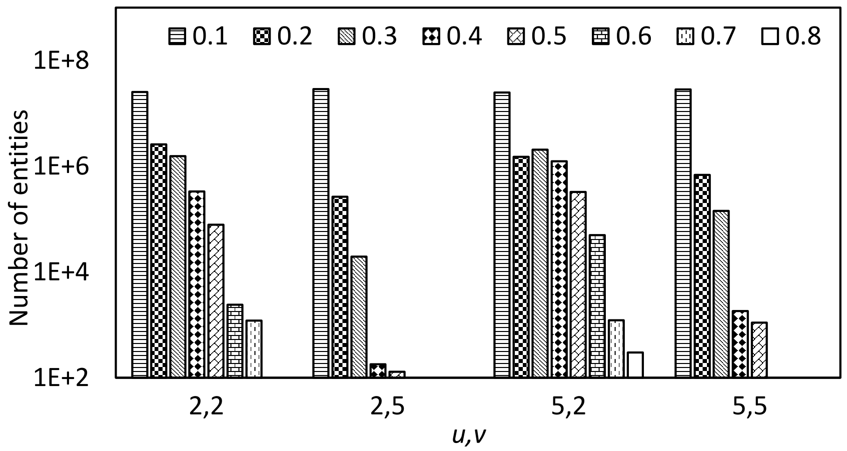
where max is a normalization factor guaranteeing the score falls into the range $[0, 1]$, $|\mathcal{P}_{ab}^l|$ denotes the total duration of all level l AjPIs in set $\mathcal{P}_{ab}$, and $u > 1$ and $v > 1$ are parameters that can be tuned. This ADM favors entities forming AjPIs at finer levels for longer durations. Naturally the ADM may also take other forms as long as they share the properties of Section 3.2. We focus our discussion on the ADM proposed here, as our experiments with several other ADMs reveal very similar trends and patterns.

Figure 7.2 provides the association degree distribution under different ADM parameters, where the horizontal axes are u, v parameter combinations, bars of different colors denote varying ranges of association degree, and the height of a bar denotes the number of entities falling in the corresponding ADM range with the query entity. From Figure 7.2, it is evident that most entities bear low association degrees with a particular entity. In the following experiments both parameters are set to 2 by default, and the sensitivity of the approach to ADM parameters is discussed in Section 7.5.

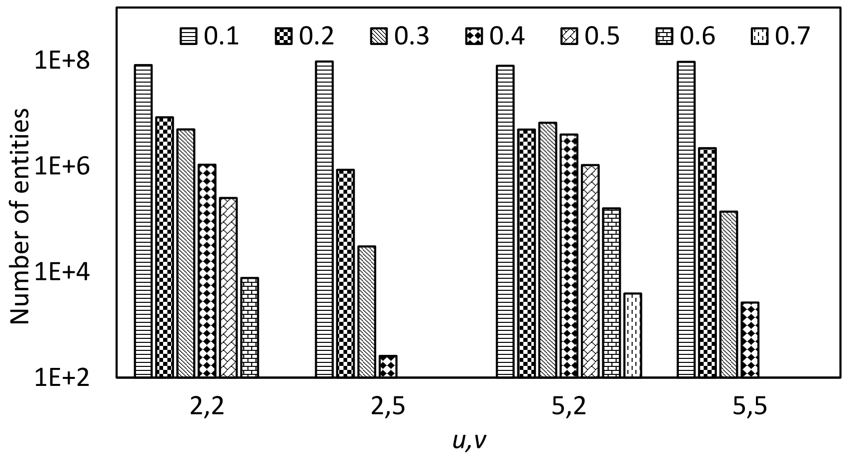
7.2 Baseline approach

We consider the following approach based on locality as the baseline approach for comparison purposes. At each level, we treat an ST-cell set of an entity as a transaction, each ST-cell as an item, and use frequent pattern mining techniques to find those frequently co-occurring ST-cells. As a result, ST-cells are partitioned into clusters, where each cluster is expected to contain ST-cells that are close to one another temporally and spatially. If there are n clusters in total, then we can assign each entity an n-bit vector, where the i-th bit in the vector of e equals 1 if e has presence in at least one ST-cell contained in cluster i, and 0 otherwise. We can thus use a bit-map to organize all entities. Given a query entity e_q, we compute an ADM upper bound between e_q and all bit-vectors. We start searching from the entities indexed by the vector with the highest UB, and continue until k entities are found where the minimal ADM of the k entities is already greater than the UBs between e_q and all remaining vectors.

The major drawback of such an approach is that in practice ST-cells show low degrees of locality; e.g., people living in the same neighborhood may work in different companies spread across the city, which makes it very difficult to iden-



(a) REAL data



(b) SYN data

Figure 7.2: Association degree distribution

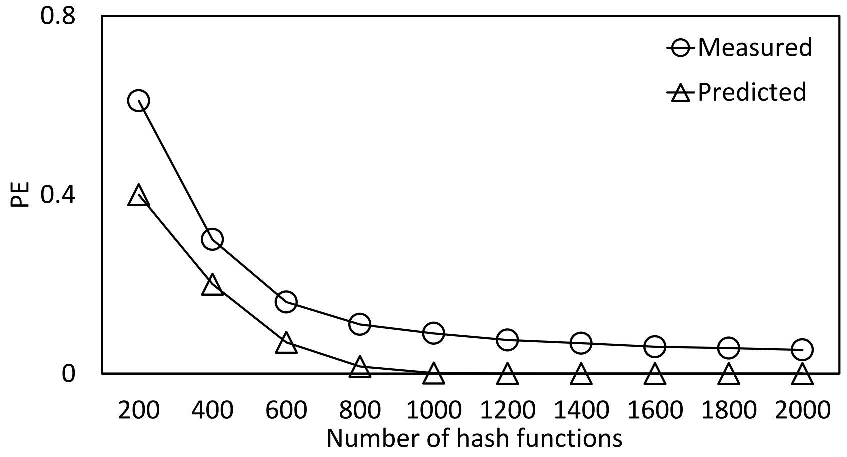
tify frequently co-occurring ST-cells. The direct consequence is that the clusters obtained demonstrate strong coupling and the bit vectors cannot capture the PI patterns of entities well. Therefore, the upper bound is loose as will be discussed later in Section 7.7.

7.3 Sensitivity to the number of hash functions

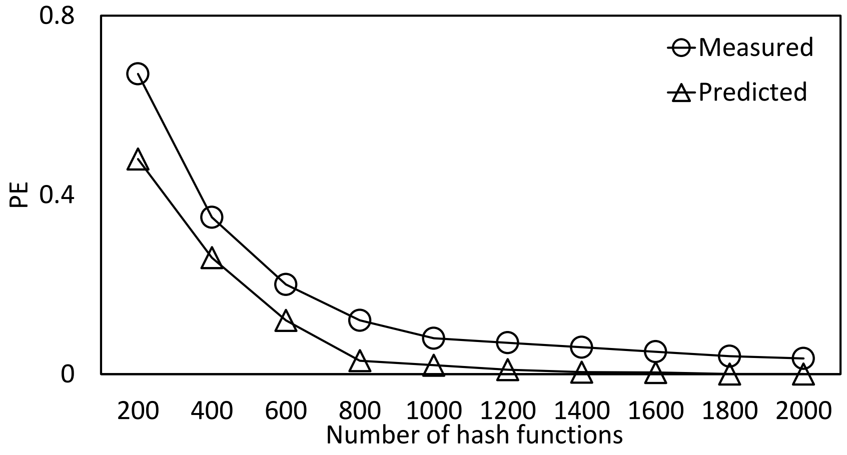
PE is closely related to the number of hash functions used to compute the signatures, n_h. We thus evaluate the PE of the proposed approach by varying n_h, and compare the measured PE in the experiment with the one predicted for the model of Section 6.3. The results are presented in Figure 7.3.

From the result one can observe that the MinSigTree provides high PE with more hash functions. The reason is that, compressing the large number of ST-cells into a low-dimensional space makes entities less unique, or even indistinguishable. With more hash functions employed, signatures can better summarize the PIs of entities and thus only closely associated entities will be placed in the same group. Diminishing returns occur when the number of hash functions reaches 1,000, as each entity has become unique enough that further employment of hash functions does not change the grouping.

As Figure 7.3 shows, the predicted PE is slightly better than measured, primarily for the following reasons:



(a) REAL data



(b) SYN data

Figure 7.3: PE vs. the number of hash functions

- Spatial units in the hierarchical IM model are assumed to be rectangles for analysis purposes, while in practice units can be in any shapes. As a result, the mobility patterns at higher levels diverge from the model;

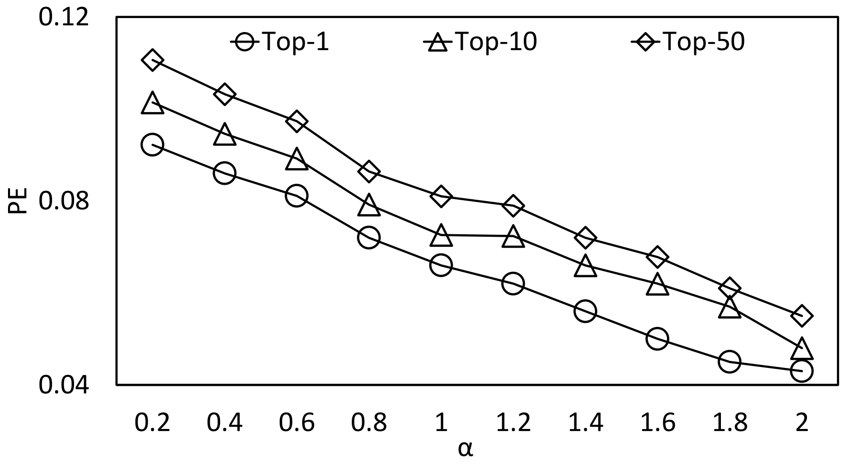
- It is assumed that the hash values are uniformly distributed on the range, which is not always the case in practice.

7.4 Sensitivity to data characteristics

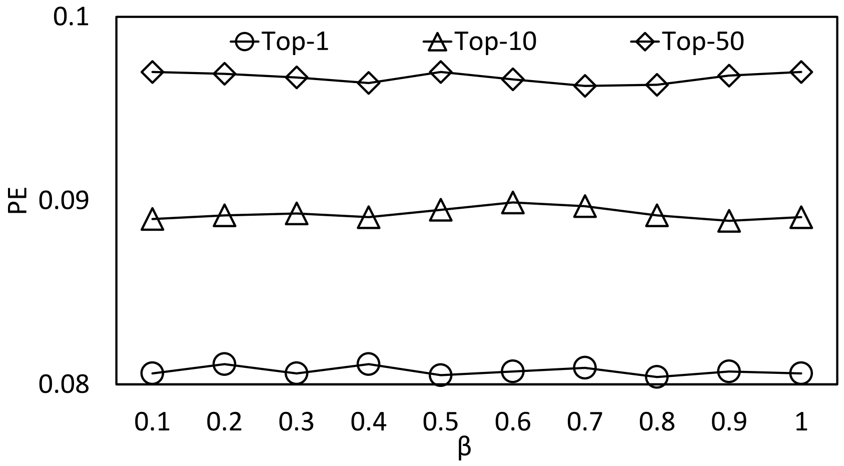
We evaluate the PE under different mobility patterns and location distributions by varying all parameters in the hierarchical IM model. Since these parameters independently control different aspects of mobility patterns and location distribution, in each experiment we only vary one parameter and fix other parameters. The results of answering Top-1, Top-10, and Top-50 query with 2,000 hash functions under different data characteristics are presented in Figure 7.4.

One can observe that curves in Figure 7.4(a) show a descending trend, as α controls the movement locality in the following way: as α increases, an entity is more likely to jump to locations in proximity when it leaves the current position. A higher level of locality will produce more closely associated entities, and thus lead to better performance.

Curves in Figure 7.4(b) demonstrate little variation, which indicates that the

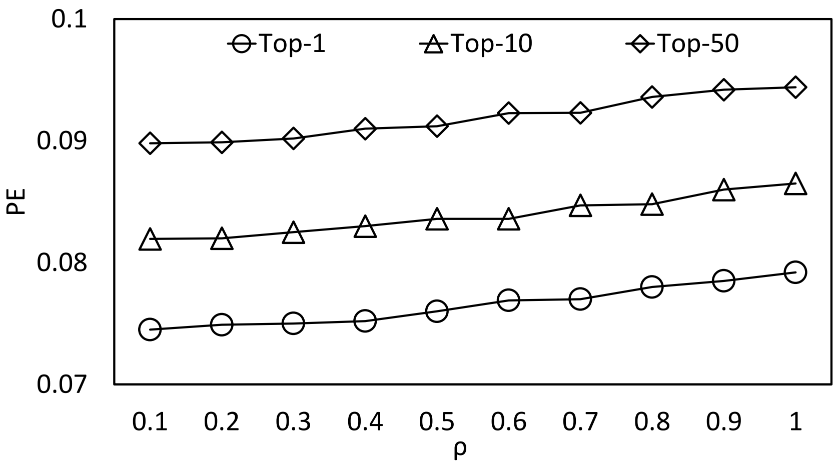


(a)

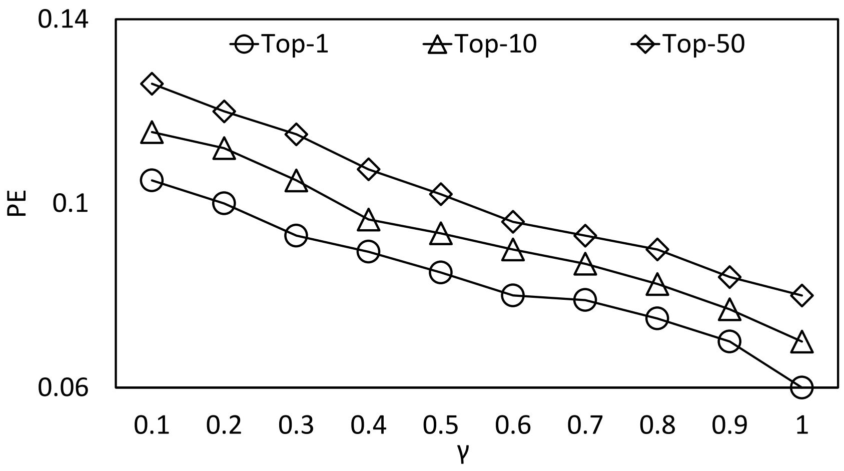


(b)

Figure 7.4: PE vs. data characteristics

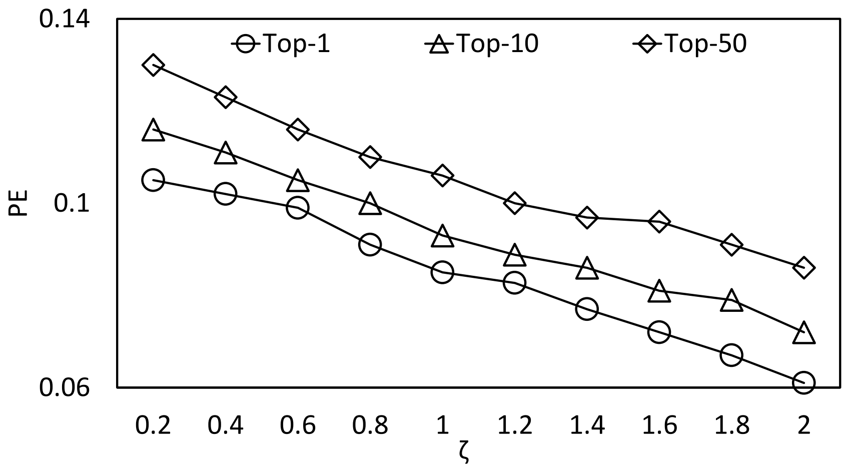


(c)

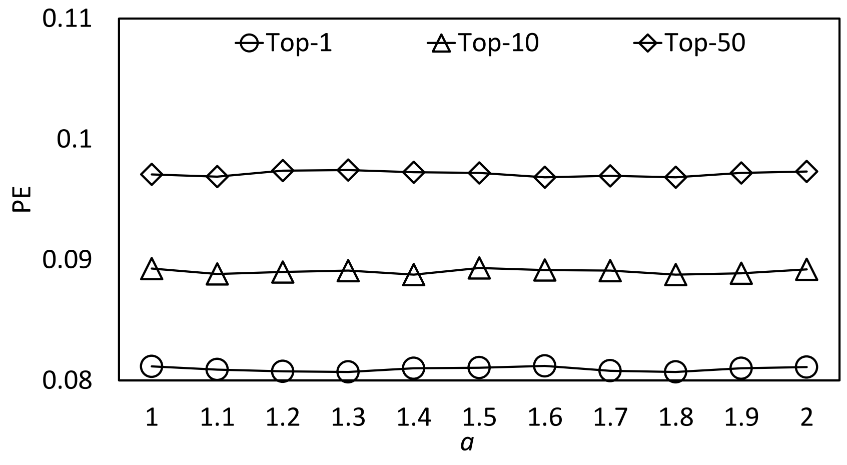


(d)

Figure 7.4: PE vs. data characteristics

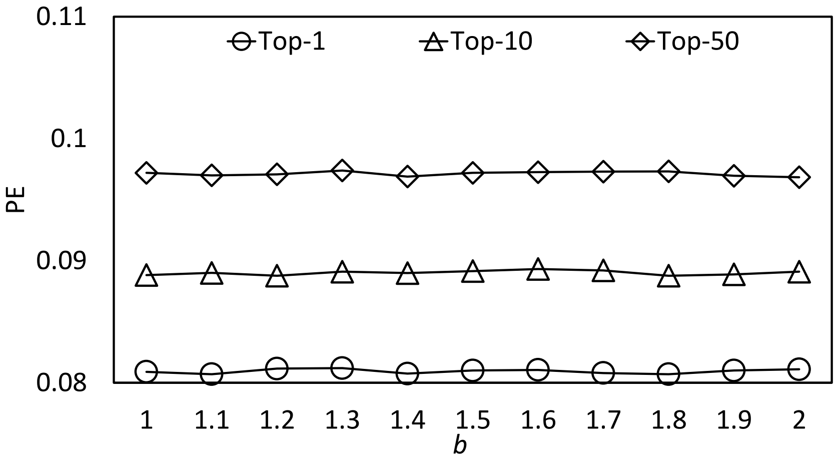


(e)

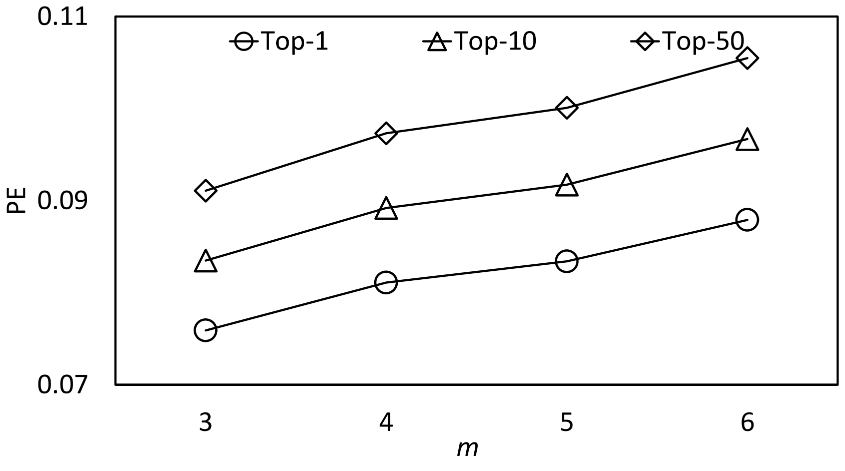


(f)

Figure 7.4: PE vs. data characteristics



(g)



(h)

Figure 7.4: PE vs. data characteristics

approach is not sensitive to the expected duration of each presence instance. This is because we partition PI into ST-cells, and consider the digital traces of an entity as a set of ST-cells. As a result, whether these ST-cells are consecutive in time or not has no influence on PE.

Parameters ρ and γ together control the tendency of an entity to return to some previously visited location. With smaller ρ and larger γ, entities visit fewer locations in total, which increases the locality. Therefore, Figure 7.4(c) depicts an ascending trend and Figure 7.4(d) a descending trend. ρ acts as a linear parameter, while γ is on the exponent; therefore curves in Figure 7.4(d) appear steeper than in Figure 7.4(c).

Similarly, Figure 7.4(e) demonstrates a descending trend, as ζ influences the locality by controlling the visit frequency distribution of an entity to locations. With higher ζ, most visits are to a few most frequently visited locations, while with lower ζ, visits are more uniformly distributed.

Curves in Figure 7.4(f) and (g) depict little variation, indicating that good PE can be achieved under any spatial distribution patterns. As is clear from the search algorithm, we touch the records of entity e only if the PI patterns of e resembles the PI patterns of the query entity at all sp-index levels. Although the values of a and b influence spatial units distribution at higher levels, base spatial unit numbers and distributions in the explored area are always constant, which means that the

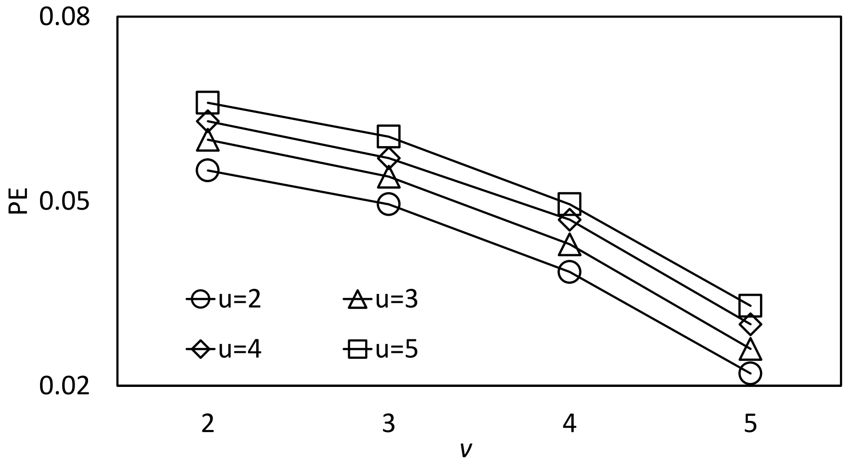
PI patterns of entities at the finest level do not change. As a result, groupings at level m of the MinSigTree remain unchanged under different values of a and b.

From Figure 7.4(h) we observe that the approach performs better with smaller m; i.e., fewer levels in the hierarchy. The reason is that with more spatial levels, more entities form AjPIs with each other, and thus the search space grows. As an example, if we assume that the spatial hierarchy is city-street-district-building, then if $m = 1$, we only consider AjPIs at the building level, while with $m = 2$ we consider AjPIs at both the building level and the street level, etc.

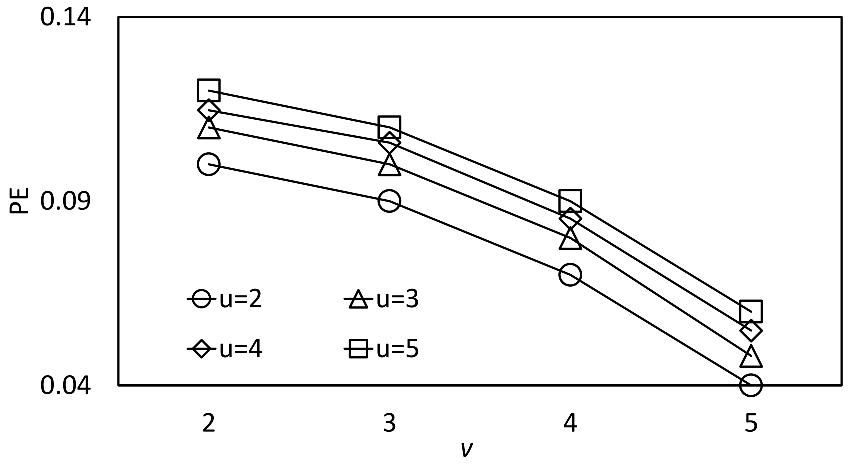
7.5 Sensitivity to ADM parameters

Values of u and v defined in the ADM of Section 7.1 provide different weights to AjPI level and duration when selecting associated entities. The PE under different ADM parameter values are presented in Figure 7.5.

As is clear from Figure 7.5, smaller u (level parameter) and larger v (duration parameter) yield higher PE in both data sets. The reason is that, while ST-cells contain timestamps, they do not contain level information. Since signatures are computed based on ST-cells, the AjPI level is not encoded into the signature. As a result, entities sharing AjPIs for longer duration are more likely to have similar signatures than entities sharing AjPIs at finer levels. In Figure 7.2, fewer entities are assigned high ADM for $u = 2, v = 5$ than other parameter combinations.



(a) REAL data



(b) SYN data

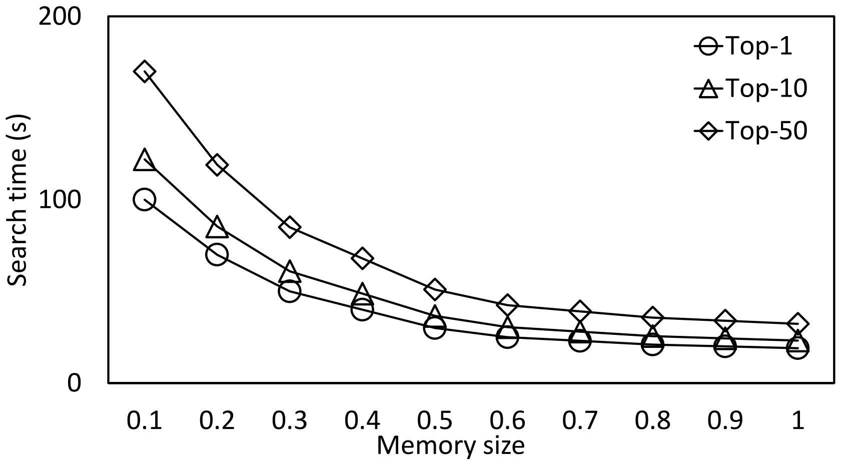
Figure 7.5: PE vs. ADM parameters

Consequently, under this combination we only need to check a small portion of entities to identify query answers. The results reveal that the approach performs better in cases where duration is the dominant factor of the association degree between entities.

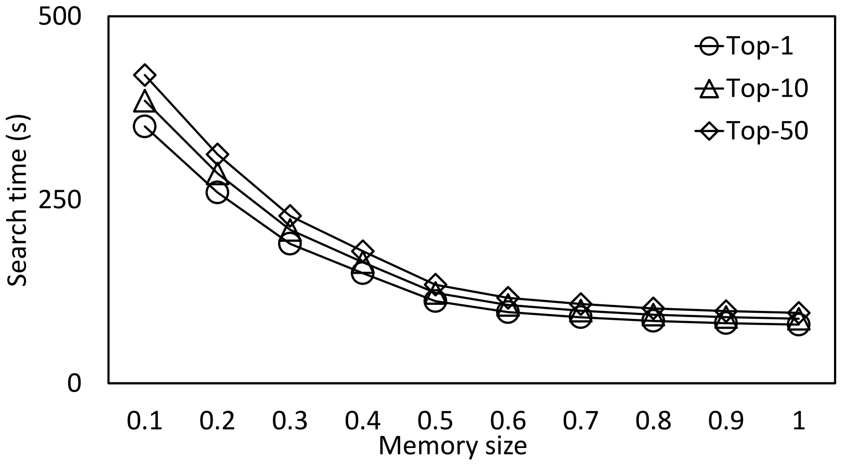
7.6 Sensitivity to memory size

If more data can be stored in memory, the time spent to fetch records from disk is reduced. Therefore, the allocated memory size has an impact on query time. Figure 7.6 depicts the time required to answer Top-1, Top-10, and Top-50 queries with 2,000 hash functions under different memory sizes.

The horizontal axis in Figure 7.6 denotes the allocated memory size (relative size compared to raw data). It is evident that the curves in Figure 7.6 depict a descending trend as expected. The curve drops super-linearly with respect to the allocated memory size. The reason is that, the relative position of entities in the MinSigTree is not always guaranteed to be correlated to their association degrees, especially when the number of hash functions is small (as discussed in Section 7.3). As a result, although we organize records by their relative position in the MinSigTree, closely associated entities are not always placed in adjacent disk blocks. However, as the memory size reaches $40\% - 50\%$ of the dataset size, the curves exhibit only small variation.



(a) REAL data



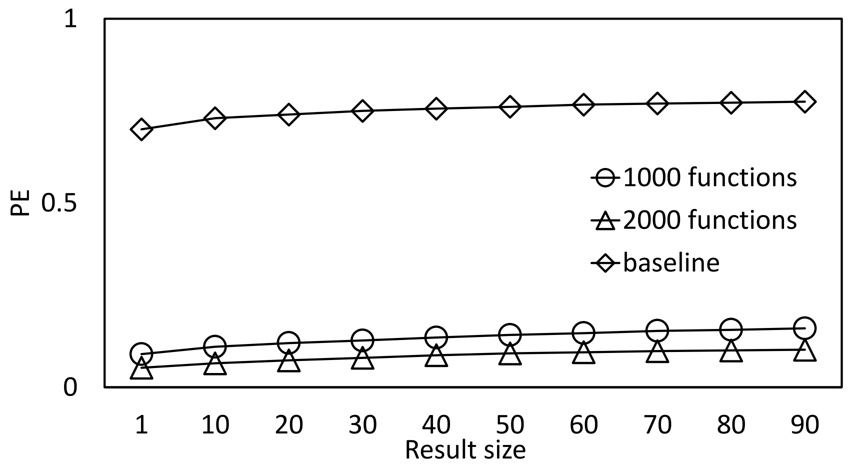
(b) SYN data

Figure 7.6: Search time vs. memory size

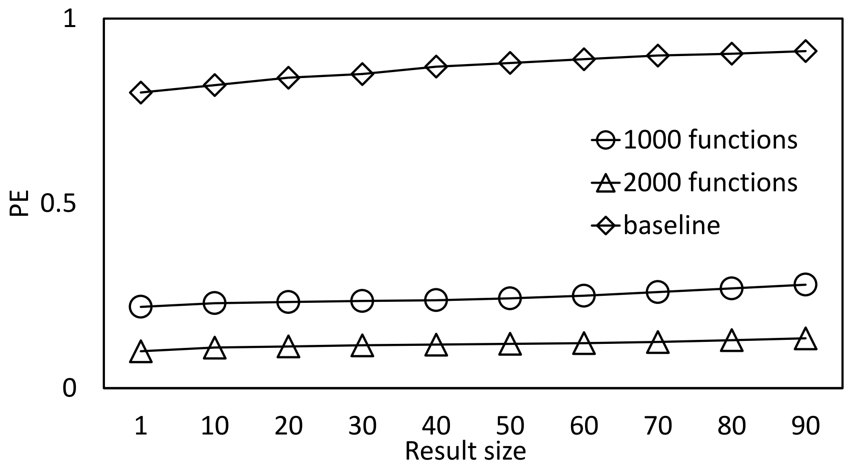
7.7 Sensitivity to result size

We also evaluate our approach as k (number of results desired in top-k) increases compared to the baseline method in Figure 7.7. PE on both SYN and REAL decreases slightly with increased result size, which is the consequence of both the ADM distribution and the nature of the branch-bound technique. Let e_q be the query entity, e_a be the i-th most associated entity to e_q, and e_b be the $(i+1)$-th most associated one. Let $df(i) = deg(e_q, e_a) - deg(e_q, e_b)$ denote the ADM difference between e_a and e_b. As Figure 7.2 indicates, the association degree distribution ranges for entities are denser when the association degree is small; i.e., $df(i) > df(j)$ if $i < j$ and $df(i) \to 0$ as i increases. Since the number of hash functions used to compute the signature is far less than the number of ST-cells, the UB of a node is not always guaranteed to be very tight. Let UB_b be the upper bound of the node containing e_b, then $deg(e_q, e_a) < UB_b$ may occur, especially when $df(i) \approx 0$; i.e., i is large, which means we always need to check e_b before returning e_a. As a result, more entities are checked when the value of k; i.e., result size, is large, which implies the trend of the curves in Figure 7.7.

The baseline method, as argued in Section 7.2, is based on the existence of clusters among ST-cells, which is not typical in real-life digital traces. Consequently, the PE of the approach is greatly limited, which explains the results in Figure 7.7



(a) REAL data



(b) SYN data

Figure 7.7: PE vs. result size (k)

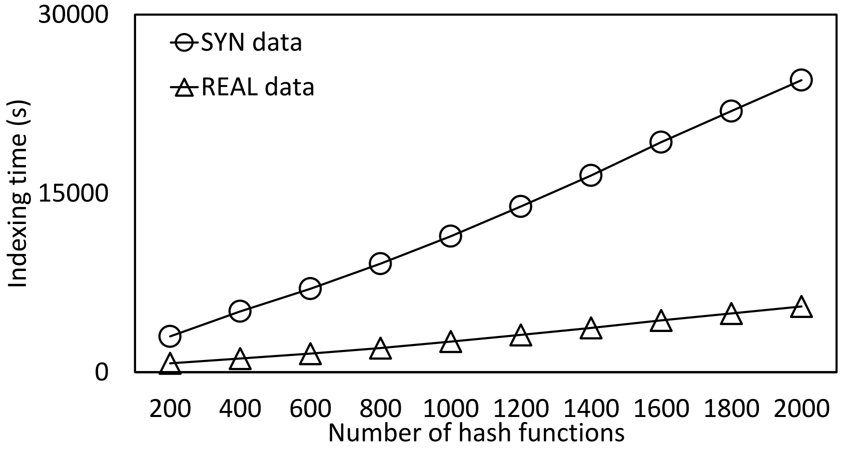
showing that MinSigTree outperforms the baseline approach by large factors.

7.8 Indexing and update cost

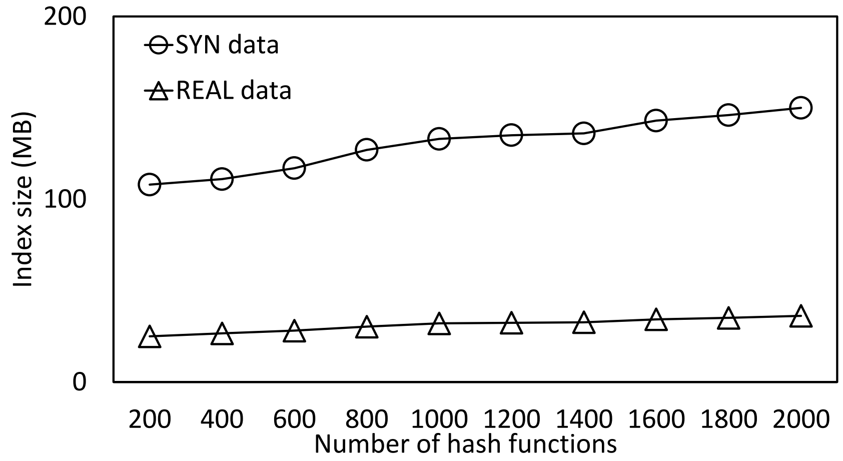
The pre-processing cost to build the MinSigTree is depicted in Figure 7.8. Pre-processing time grows almost linearly with the number of hash functions (n_h), as the most expensive step in the index construction process is the computation of signatures for each entity, which requires n_h hash operations for each ST-cell where the entity has a presence.

The size of the MinSigTree is provided in Figure 7.8(b). Generally, each node in the MinSigTree contains two integers, one indicating its routing index, and the other recording the hash value of the routing index. A leaf node also includes a pointer to the entities contained in this node. With more hash functions, each entity becomes more unique and thus a node with entity set $\mathcal{E}_N$ may split into several new nodes, each containing a subset of $\mathcal{E}_N$. Therefore, the size of the MinSigTree increases with the number of hash functions. However, the overhead is quite small compared to the data size.

Figure 7.9 illustrates the time required for dynamic index updates. In particular, the figure depicts the time to update records for 1 million entities in an already built MinSigTree. Since the update process is independent from the data distribution, we present experiments on SYN data. We report the time required under different



(a)



(b)

Figure 7.8: Indexing cost

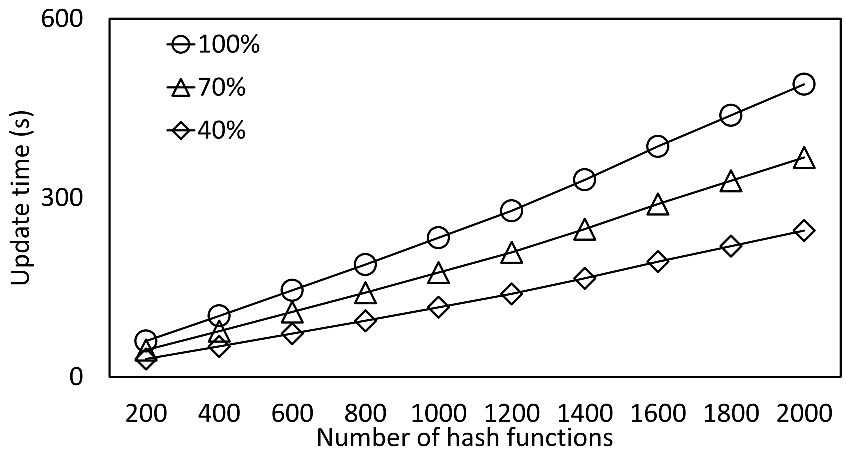


Figure 7.9: Update cost

conditions: when 100%, 70%, and 40% of the entities updated are existing dataset entities, respectively. The time to update grows linearly with the number of hash functions as in the case of building the index. In addition, one can observe that inserting new entities requires less time than modifying the records for existing entities. The reason is that, when updating an existing entity we have to perform the following steps: (1) locate the entity's position in the MinSigTree, (2) remove it from the corresponding leaf node in the index, (3) compute its new signature, and (4) insert it to the proper node. In contrast, for a new entity only steps (3) and (4) are required.

8 Conclusions and Future Work

8.1 Contributions

In this thesis, we have developed a generic association degree measure (ADM), which is based on the presence instances of a single entity, and adjoint presence instances between entities, to quantify the association, and formally defined the problem of top-k query over digital traces. We have also developed a suite of techniques to process such queries with high efficiency. The approach we propose consists of three parts: (1) we transform the digital traces into ST-cell set sequences, which facilitates efficient computation of the association degree between entities; (2) we design a MinHash-based approach that computes a signature list for each entity, so that entities with large overlaps on digital traces have similar signatures, and we design the MigSigTree that groups entities with similar signatures together; (3) we propose a search algorithm which takes advantage of the monotonic property of MinSigTree and exploits an early termination condition to process top-k queries with high efficiency. We have generalized a well-established individual mobility model to the hierarchical spatial setting and analytically quantified the pruning effectiveness of the proposed method based on the normal movement patterns of entities in such an environment. We have also conducted extensive experiments on both synthetic and real datasets to study the sensitivity of the proposed approach to data characteristics and tunable parameters, and compare it against a baseline method. The proposed approach shows strong and stable pruning effectiveness across a variety of settings, and significantly outperforms the baseline method.

8.2 Future work

This study represents the first step towards a new research direction that investigates the relationship between entities based on their digital traces, and many challenging questions remain. We envision three topics of follow up work that can be immediately investigated: (1) answering approximate top-k queries: many applications require the results be returned with very short delay and approximate answers would suffice. It is therefore interesting to develop new data structures and algorithms to support the fast yet approximate processing of top-k queries with certain quality guarantees; (2) embedding techniques: initially designed for Natural Language Processing tasks, embedding techniques have also found applications in spatio-temporal data management. Transforming entities and ST-cells into vectors in an embedding space will facilitate the use of machine learning methods in processing similarity queries involving digital traces; (3) kNN join: similarity join problems over digital traces, combining the kNN queries issued separately for multiple entities together, are important yet challenging. Efficient processing of kNN joins is the basis of a large family of data analysis and data mining tasks.

In summary, we initiated the study and formally defined the problem of top-k query over digital traces, and developed a suite of techniques to efficiently process such queries. We analytically quantified the pruning effectiveness of the proposed method, and presented the results of extensive experiments on both synthetic and real data sets demonstrating the practical utility of our proposal. Natural extensions of the study, such as approximate top-k queries and kNN joins, are worthy of further investigation.

Bibliography

[1] Tenindra Abeywickrama, Muhammad Aamir Cheema, and David Taniar. "K-nearest neighbors on road networks: a journey in experimentation and in-memory implementation". In: *Proceedings of the VLDB Endowment* 9.6 (2016), pp. 492–503.

[2] Pritom Ahmed et al. "Efficient Computation of Top-k Frequent Terms over Spatio-temporal Ranges". In: *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*. ACM. 2017, pp. 1227–1241.

[3] Ahmed M Aly, Walid G Aref, and Mourad Ouzzani. "Spatial queries with two kNN predicates". In: *Proceedings of the VLDB Endowment* 5.11 (2012), pp. 1100–1111.

[4] Senjuti Basu Roy and Kaushik Chakrabarti. "Location-aware type ahead search on spatial databases: semantics and efficiency". In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM. 2011, pp. 361–372.

[5] Tolga Bozkaya, Nasser Yazdani, and Meral Özsoyoğlu. "Matching and indexing sequences of different lengths". In: *Proceedings of the sixth international conference on Information and knowledge management*. ACM. 1997, pp. 128–135.

[6] Andrei Z Broder. "On the resemblance and containment of documents". In: *Compression and complexity of sequences 1997. proceedings*. IEEE. 1997, pp. 21–29.

[7] Lei Chen, M Tamer Özsu, and Vincent Oria. "Robust and fast similarity search for moving object trajectories". In: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. ACM. 2005, pp. 491–502.

[8] Zaiben Chen et al. "Searching trajectories by locations: an efficiency study". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 255–266.

[9] Farhana M Choudhury et al. "Maximizing bichromatic reverse spatial and textual k nearest neighbor queries". In: *Proceedings of the VLDB Endowment* 9.6 (2016), pp. 456–467.

[10] Ondrej Chum, James Philbin, Andrew Zisserman, et al. "Near Duplicate Image Detection: min-Hash and tf-idf Weighting." In: *BMVC*. Vol. 810. 2008, pp. 812–815.

[11] Richard Cole. "Parallel merge sort". In: *SIAM Journal on Computing* 17.4 (1988), pp. 770–785.

[12] Tobias Emrich et al. "An extendable framework for managing uncertain spatio-temporal data". In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*. ACM. 2014, pp. 1087–1090.

[13] Yixiang Fang et al. "Scalable algorithms for nearest-neighbor joins on big trajectory data". In: *IEEE Transactions on Knowledge and Data Engineering* 28.3 (2016), pp. 785–800.

[14] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. "Index-based most similar trajectory search". In: *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE. 2007, pp. 816–825.

[15] Fosca Giannotti et al. "Trajectory pattern mining". In: *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2007, pp. 330–339.

[16] Ralf Hartmut Güting, Thomas Behr, and Jianqiu Xu. "Efficient k-nearest neighbor search on moving object trajectories". In: *The VLDB Journal—The International Journal on Very Large Data Bases* 19.5 (2010), pp. 687–714.

[17] Cheng-Kang Hsieh et al. "Immersive recommendation: News and event recommendations using personal digital traces". In: *Proceedings of the 25th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2016, pp. 51–62.

[18] Sergey Ioffe. "Improved consistent sampling, weighted minhash and l1 sketching". In: *Data Mining (ICDM), 2010 IEEE 10th International Conference on.* IEEE. 2010, pp. 246–255.

[19] Jianqiu Ji et al. "Min-max hash for jaccard similarity". In: *Data Mining (ICDM), 2013 IEEE 13th International Conference on.* IEEE. 2013, pp. 301–309.

[20] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. "Trajectory clustering: a partition-and-group framework". In: *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. ACM. 2007, pp. 593–604.

[21] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[22] Ping Li and Kenneth W Church. “A sketch algorithm for estimating two-way and multi-way associations”. In: *Computational Linguistics* 33.3 (2007), pp. 305–354.

[23] Ping Li, Arnd Konig, and Wenhao Gui. “b-Bit minwise hashing for estimating three-way similarities”. In: *Advances in Neural Information Processing Systems*. 2010, pp. 1387–1395.

[24] Zhenhui Li et al. “MoveMine: mining moving object databases”. In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 1203–1206.

[25] James J Little and Zhe Gu. “Video retrieval by spatial and temporal structure of trajectories”. In: *Storage and Retrieval for Media Databases 2001*. Vol. 4315. International Society for Optics and Photonics. 2001, pp. 545–553.

[26] Jiaheng Lu, Ying Lu, and Gao Cong. “Reverse spatial and textual k nearest neighbor search”. In: *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM. 2011, pp. 349–360.

[27] Chunyang Ma et al. “KSQ: Top-k similarity query on uncertain trajectories”. In: *IEEE Transactions on Knowledge and Data Engineering* 25.9 (2013), pp. 2049–2062.

[28] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. "Detecting near-duplicates for web crawling". In: *Proceedings of the 16th international conference on World Wide Web*. ACM. 2007, pp. 141–150.

[29] Anna Monreale et al. "Wherenext: a location predictor on trajectory pattern mining". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 637–646.

[30] Johannes Niedermayer et al. "Probabilistic nearest neighbor queries on uncertain moving object trajectories". In: *Proceedings of the VLDB Endowment* 7.3 (2013), pp. 205–216.

[31] Julien Pilourdault, Vincent Leroy, and Sihem Amer-Yahia. "Distributed evaluation of top-k temporal joins". In: *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. ACM. 2016, pp. 1027–1039.

[32] Michalis Potamias et al. "K-nearest neighbors in uncertain graphs". In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 997–1008.

[33] Tobias Preis et al. "Quantifying the digital traces of Hurricane Sandy on Flickr". In: *Scientific reports* 3 (2013), p. 3141.

[34] Hiroaki Sakoe and Seibi Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978), pp. 43–49.

[35] Zhou Shao et al. “Vip-tree: an effective index for indoor spatial queries”. In: *Proceedings of the VLDB Endowment* 10.4 (2016), pp. 325–336.

[36] Mehdi Sharifzadeh and Cyrus Shahabi. “Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries”. In: *Proceedings of the VLDB Endowment* 3.1-2 (2010), pp. 1231–1242.

[37] Jieming Shi, Dingming Wu, and Nikos Mamoulis. “Top-k relevant semantic place retrieval on spatial RDF data”. In: *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data*. ACM. 2016, pp. 1977–1990.

[38] Chaoming Song et al. “Modelling the scaling properties of human mobility”. In: *Nature Physics* 6.10 (2010), p. 818.

[39] Lu-An Tang et al. “Retrieving k-nearest neighboring trajectories by a set of point locations”. In: *International Symposium on Spatial and Temporal Databases*. Springer. 2011, pp. 223–241.

[40] Bo Tang et al. “Extracting top-k insights from multi-dimensional data”. In: *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data*. ACM. 2017, pp. 1509–1524.

[41] Michail Vlachos, Dimitrios Gunopulos, and Gautam Das. “Rotation invariant distance measures for trajectories”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 707–712.

[42] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. “Discovering similar multidimensional trajectories”. In: *Data Engineering, 2002. Proceedings. 18th International Conference on*. IEEE. 2002, pp. 673–684.

[43] Haozhou Wang et al. “Sharkdb: An in-memory storage system for massive trajectory data”. In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pp. 1099–1104.

[44] Sheng Wang et al. “Answering top-k exemplar trajectory queries”. In: *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*. IEEE. 2017, pp. 597–608.

[45] Wan-Lei Zhao, Hervé Jégou, and Guillaume Gravier. “Sim-Min-Hash: An efficient matching technique for linking large image collections”. In: *Proceed-*

ings of the 21st ACM international conference on Multimedia. ACM. 2013, pp. 577–580.

[46] Bolong Zheng et al. “Approximate keyword search in semantic trajectory database”. In: *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE. 2015, pp. 975–986.

[47] Kai Zheng et al. “Towards efficient search for activity trajectories”. In: *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE. 2013, pp. 230–241.