Find Another Me Across the World - Large-scale Semantic Trajectory Analysis Using Spark

Chaoquan Cai, Dan Lin, *Senior Member, IEEE*

Abstract—In today's society, location-based services are widely used which collect a huge amount of human trajectories. Analyzing semantic meanings of these trajectories can benefit numerous real-world applications, such as product advertisement, friend recommendation, and social behavior analysis. However, existing works on semantic trajectories are mostly centralized approaches that are not able to keep up with the rapidly growing trajectory collections. In this paper, we propose a novel large-scale semantic trajectory analysis algorithm in Apache Spark. We design a new hash function along with efficient distributed algorithms that can quickly compute semantic trajectory similarities and identify communities of people with similar behavior across the world. The experimental results show that our approach is more than 30 times faster than centralized approaches without sacrificing any accuracy like other parallel approaches.

1 Introduction

Today, mobile devices with the Internet and positioning technology are extremely common, and almost all users of these devices utilize some forms of location-based services, such as using Google Maps to find nearby restaurants, getting directions from current locations, and checking in on Yelp or other social network sites. As a result, a huge amount of trajectories has been collected by service providers.

Given the collected trajectories, people have looked into their space similarities and semantic similarities. Identifying geometrically similar trajectories is beneficial for applications such as traffic flow analysis and urban planning, while analyzing semantic meanings of visited places may lead to even more interesting findings regarding humans' daily activity patterns and preferences. For example, as shown in Figure 1, Alice's and Cindy's trajectories in Sydney are overlapping, i.e., geometrically similar. However, we will find that they might have quite different interests after considering the semantic meanings of their trajectories. Specifically, Alice is visiting a high school and a grocery store while along the same path Cindy stops by a company and a cinema which demonstrates a different social behavior compared to Alice. In contrast, Alice's and Bob's trajectories have nothing in common in terms of spatial similarity as they live in different cities, whereas the semantic meanings of their trajectories reveal that they are both related with schools and share common daily activity patterns since they visited places of similar types. Similarly, Carol and Dave demonstrate activity patterns of being frequent flyers although their trajectories are located in different cities and different countries.

The above examples illustrate the potential benefits of identifying semantically similar trajectories, such as for targeted and precise product and service recommendations, and online community formations. Especially with the emerging of the future metaverse – a social networking cyberspace where users have "second" life through virtual reality and augmented reality technology, the boundaries between the physical world and cyber world are diminishing. Communities can be formed beyond those who are only spatially closed. Semantically similar trajectories in the real world could become a good indicator for friend exploration and recommendation in the metaverse. However, it is indeed a very challenging task to analyze today's enormous volume of trajectories in a timely manner. The most representative and sophisticated techniques provide only centralized algorithms for calculating semantic similarity among trajectories [1, 2]. It could take days and months for a centralized algorithm to analyze tens of billions of collected trajectories considering Facebook already has 2.4 billion

- *Chaoquan Cai is with Computer Science Department at Vanderbilt University, TN 37235, USA.*
  E-mail: chaoquan.cai@vanderbilt.edu
- *Dan Lin is with Computer Science Department at Vanderbilt University, TN 37235, USA.*
  E-mail: dan.lin@vanderbilt.edu

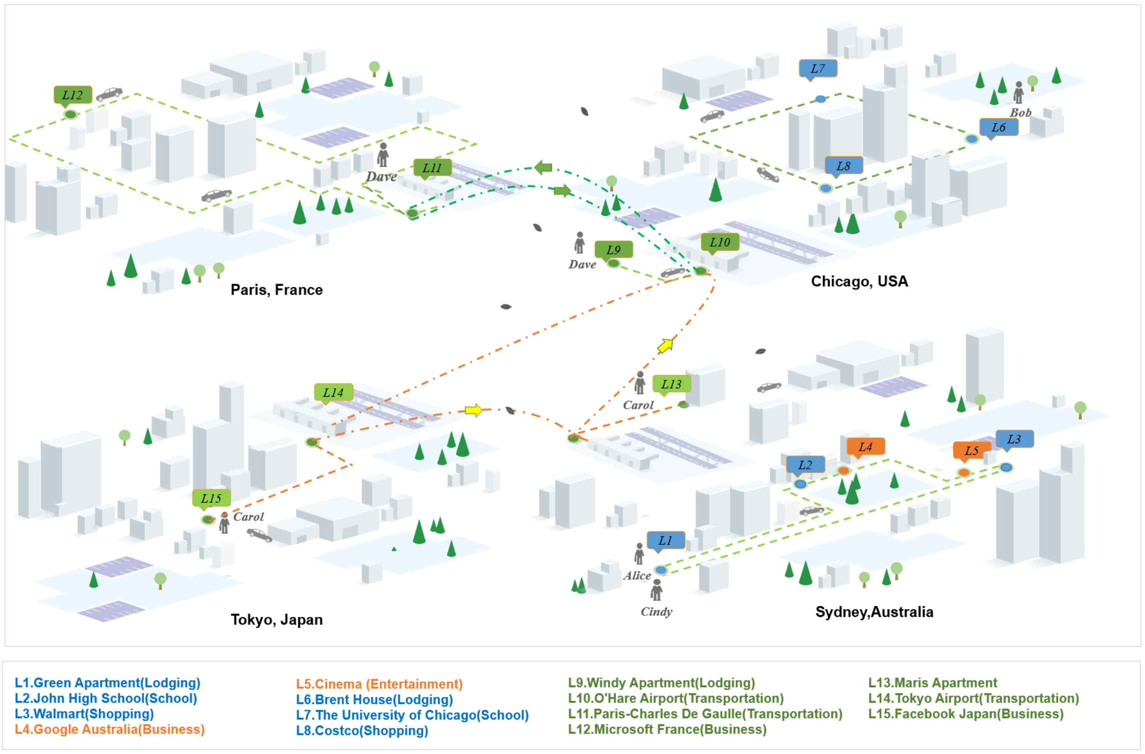

L12
L11
Dave
L9
Dave
L10
L7
Bob
L6
L8
Paris, France
Chicago, USA
L14
L13
Carol
L15
Carol
L4
L5
L3
L2
L1
Alice
Cindy
Tokyo, Japan
Sydney,Australia
L1.Green Apartment(Lodging)
L2.John High School(School)
L3.Walmart(Shopping)
L4.Google Australia(Business)
L5.Cinema (Entertainment)
L6.Brent House(Lodging)
L7.The University of Chicago(School)
L8.Costco(Shopping)
L9.Windy Apartment(Lodging)
L10.O'Hare Airport(Transportation)
L11.Paris-Charles De Gaulle(Transportation)
L12.Microsoft France(Business)
L13.Maris Apartment
L14.Tokyo Airport(Transportation)
L15.Facebook Japan(Business)

Fig. 1. Geometrically Similar Trajectories vs. Semantically Similar Trajectories: (1) Alice's and Cindy's trajectories are geometrically similar but semantically different; (2) Alice's and Bob's trajectories are geometrically different but semantically similar; (3) Dave and Carol also have geometrically different but semantically similar trajectories as they are both frequent flyers.

active users. Moreover, unlike calculating the spatial similarity of trajectories which can be easily scaled up by parallelly processing trajectory datasets in different regions (e.g., cities) [3], semantically similar trajectories are no longer confined by regions as they may be located in different countries as shown in the example. Thus, it is not a trivial task to design a divide-and-conquer strategy for semantic trajectory similarity calculation.

In order to address the aforementioned challenges, in this paper, we propose an efficient distributed computing algorithm on the Apache Spark platform [4] for analyzing large-scale semantic trajectories. Our algorithm is named *AnotherMe* with the goal of finding people with similar social behavior across the world, who may share common interests and new insights. Specifically, we design a novel hash function, namely *Sequence-Sensitive Hashing (SSH)*, to effectively produce initial partitions of trajectories that are likely to be semantically similar. Then, we calculate trajectory similarity scores at different granularity levels in each partition in parallel. It is worth noting that our similarity function is more sophisticated than those in the previous works. Specifically, we take into account the recurrence of the same places in a trajectory (e.g.,company→cafe→company) which is commonly seen in the real world scenarios, whereas existing works simply consider distinct places in a trajectory and ignore their repetitive patterns.

Our proposed work has made the following contributions:

- Unlike most existing works on trajectory analysis [3, 5, 6] which mainly calculate the spatial similarity of trajectories, our work not only analyzes the more challenging type of similarity – semantic similarity – in a large scale setting, but also provides a new way of identifying communities of common interests based on dynamic trajectory information rather than users' static profiles.
- We design a novel hash function, the Sequence-Sensitive Hashing (SSH) which captures the characteristics of semantic similarity among trajectories much better than the original hashing function in the Apache Spark. Specifically, the trajectory analysis results obtained from our distributed algorithm have achieved 100% accuracy compared to the ground truth obtained from a centralized approach, whereas the Spark's original hash function yields very low accuracy.

- We have an important finding regarding the implementation of distributed algorithms in Apache Spark. We found that although the common approach for implementing custom data analysis functions using Apache Spark's user-defined function feature is convenient, such an approach does not fully unleash Apache Spark's parallel computing capabilities. Significantly better performance is achieved after we strategically decompose and convert the tasks in the trajectory analysis process into atomic tasks that can be carried out directly by Apache Spark's built-in functions.
- We conduct extensive experiments on both real trajectory datasets and large-scale synthetic trajectory datasets. The results prove that both the time and memory consumption of our method are orders of magnitude less than the centralized version. Specifically, our Spark-version is more than 30 times faster than the centralized approach

The remainder of the paper is organized as follows. Section II provides a brief overview of the background knowledge about Apache Spark. Section III reviews the related work. Section IV presents our distributed trajectory analysis algorithms. Section V reports the experimental results. Finally, Section VI concludes the paper.

II Background of Apache Spark

We first briefly introduce the key features of Apache Spark [4] as our algorithms are run on this environment. Spark is an open-source tool designed for large-scale data processing. It aims to address the performance limitations in an earlier popular large-scale data processing paradigm – MapReduce by leveraging in-memory caching and processing. The fundamental data structure in Spark is the resilient distributed dataset (RDD) which helps parallelize distributed data processing over a cluster of machines. Spark also has an important component called Spark SQL that supports parallel SQL operations which are used in our work. Spark SQL utilizes a data abstraction called DataFrames which can store structured and semi-structured data. Spark SQL can perform efficient SQL functions such as "group by" and "join". Moreover, Spark also provides several built-in hash functions for general uses, such as Bucketed Random Projection and MinHash. Overall, Apache Spark is the leading platform for the large-scale data analytics.

III Related Work

In this section, we review related works on trajectory analysis. We will first quickly discuss early efforts that developed centralized approaches for trajectory analysis. Then, we will analyze a few recent distributed approaches for large-scale trajectory datasets that are more related to our work.

Trajectory analysis is typically based on two kinds of similarities: coordinate-based similarity and semantic-based similarity. The coordinate-based similarity measures include Euclidean distance [7], Hausdorff and Frechet distance, Angular metric or shape similarity [8, 9, 10] and edit distance on movement pattern strings[11]. The most commonly used coordinate-based similarity metric is Euclidean distance which aggregates the Euclidean distances of nearest pairs of positions in the two trajectories. The semantics-based trajectory similarity is commonly measured using longest common subsequence or longest common sequence [12, 13]. The longer the common subsequence, the less semantic-distance between the two trajectories, which means the more similar the two trajectories will be. The following is a simple example of the calculation of the coordinate-based similarity and the semantic-based similarity of two trajectories. Assume Alice and Bob live in two different cities and their trajectories are represented as the place names and the coordinates as follows. Without loss of generality, we use integer coordinates for the ease of illustration.

T_{Alice} = [high_school(3,4) $\rightarrow$ shopping_mall(5, 8) $\rightarrow$ dinner(10,7) $\rightarrow$ home(10,8)].

T_{Bob} = [university(100,100) $\rightarrow$ grocery_store(105, 110) $\rightarrow$ cafe(110,120) $\rightarrow$ home(120,125)].

The coordinate-based similarity between these two trajectories can be simply calculated by finding the average distance between each closet pair of locations, which is

D=$\frac{1}{4}[\sqrt{(100-3)^2+(100-4)^2} + \sqrt{(105-5)^2+(110-8)^2} + \sqrt{(110-10)^2+(120-7)^2} + \sqrt{(120-10)^2+(125-8)^2}]$=148

This coordinate-based distance is very large due to the fact that the two people live in different cities. However, if we look at the semantic-based similarity between these two trajectories, we will notice that the places that Alice and Bob have visited belong to the same type and in the same sequence. Therefore, its semantic-based distance is close to 0. From here, we can observe that coordinate-based similarity is good at describing the spatial similarity of two trajectories while the semantic-based similarity is good at describing the social meaning of two trajectories.

Various centralized approaches have been proposed to identify similar trajectories based on coordinate-based distance [5, 6, 14, 15, 16]. Most of the early works focus on trajectory clustering [17] using Euclidean distances between coordinates of locations in trajectories. Later, Zheng et al. [18, 19, 20] started examining trajectory similarity in terms of semantic meanings of places visited. Zhang et al. [21] also pro-

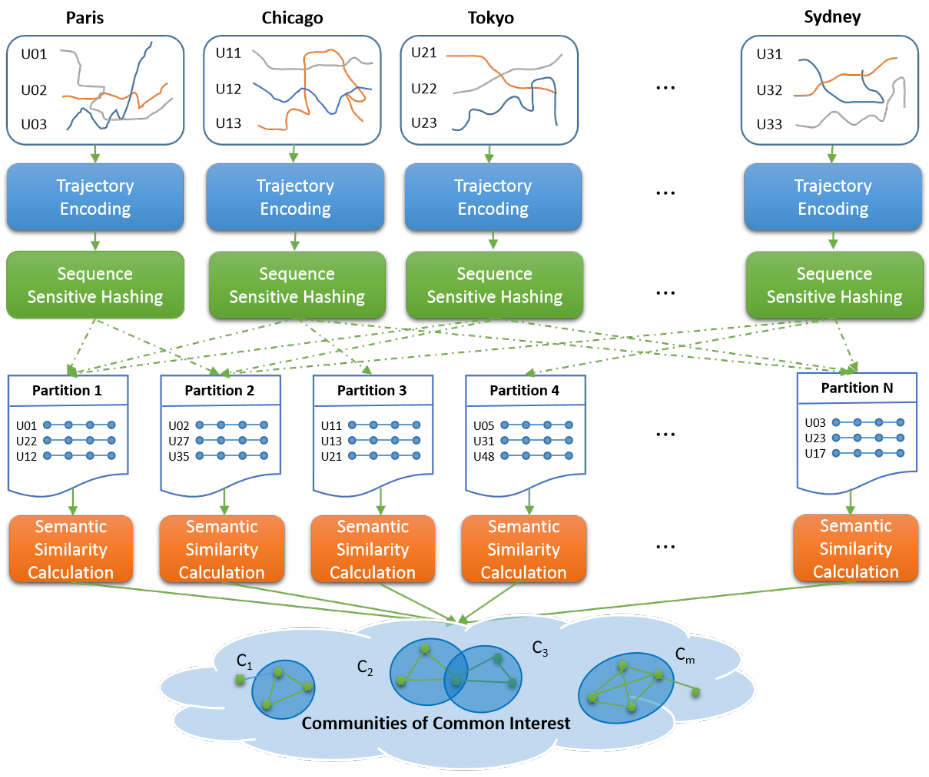


Fig. 2. An Overview of the AnotherMe System

pose an approach namely SPLITTER to detect sequential patterns in semantic trajectories. The SPLITTER groups similar places together to mine coarse patterns, and then progressively refine them using weighted snippet shifting to obtain fine-grained patterns. Choi et al. [22] devise a mining algorithm, called RegMiner, to find the local frequent movement patterns. Celik and Dokuz [23] further consider the temporal context of locations and only frequently visited or socially important locations when determining the similarity of trajectories. Wan et al. [24] take into account both coordinate-based distance and semantic-based distance, and introduce a new concept of semantic-geographic similar trajectories. Gao et al. [25] use clustering approaches to identify regions of interests and then represent trajectories in multi-resolutions. This multi-resolution idea is similar to our proposed hierarchical trajectory representation whereas we leverage the semantic hierarchy instead of the regions of interests. Most recently, semantic trajectory analysis has fostered a series of new applications, such as recommender systems in cultural spaces proposed by Angelis et al. [26], the itinerary recommender system by Cai et al. [27], similar pattern group identification by Cao et al. [28], and social-space keyword query by [29]. All of these works are centralized approaches and would not scale up when the huge amount of trajectories across the world need to be compared as that in our problem setting.

There have been some works on distributed trajectory analysis [3, 22, 30]. However, most of them [3, 31, 32, 33] use coordinate-based distance for trajectory comparison, which eases the development of the parallel computation since trajectories can be easily divided into subsets based on regions they are located. Among these coordinate-based trajectory analyses works, many [34, 35, 36] utilize MapReduce; very few [32, 33] takes advantage of Apache Spark which has been shown to be much more efficient than MapReduce. A representative work using Apache Spark is TrajSpark [32] which can answer k-nearest neighbor queries among trajectories based on the distance between coordinates of locations in the trajectories. Alvares et al. [37, 38] append the semantic information to trajectories as a post-processing of the coordinate-based trajectory classification. The most recent work on distributed semantic trajectory analysis is by Chen et al. [30]. Their approach first groups trajectories based on semantic meanings and then further partitions the trajectories based on spatial proximity. Unlike their work which leverages spatial partitioning to obtain a smaller dataset to process in parallel, the problem addressed in this paper aims to identify trajectories across the globe with similar semantic meanings which is much more challenging since we cannot employ spatial partitioning to reduce the number of trajectories that need an in-depth comparison.

To sum up, to the best of our knowledge, there has not been any distributed Spark-based algorithm for identifying semantic similarity among large-scale trajectories without spatial constraints.

In addition, it is worth noting that another line of research on spatial keyword search [39] also look into the semantic meanings of locations, but it is not sufficient to conduct semantic trajectory analysis because the spatial keyword search is still constrained by geographic regions and the keyword search does not consider the visiting order of places.

IV THE ANOTHERME ALGORITHM

We propose a distributed computing algorithm, called *AnotherMe*, which analyzes a set of semantic trajectories and returns a set of communities of interest. The set of semantic trajectories belongs to people who may be in different regions of the world. The communities of interest are formed by people who share common activity patterns as illustrated in the examples shown in Figure 1. The similarity between people's activity patterns is measured based on the similarity between their semantic trajectories. The formal definitions of these notions will be given in the following subsections.

Figure 2 presents an overview of our algorithm which consists of four main phases: (i) Semantic encoding which converts the original semantic trajectory into an encoding of place types at different levels of semantic meanings; (ii) Sequence-Sensitive Hashing (SSH) which creates initial partitions of trajectories based on semantic similarities; (iii) Semantic trajectory similarity computation which calculates semantic trajectory similarity in each partition in parallel; and (iv) Output communities of common interests, i.e., trajectories with same hash values and their pair-wise trajectory similarity. In what follows, we elaborate the detailed algorithm in each phase.

IV.1 Semantic Encoding

The goal of this phase is to be able to generate an encoding for each trajectory so that we can easily compare semantic trajectories (defined in Definition 1) that do not share any geographic overlap.

Definition 1. *(Semantic Trajectory) A semantic trajectory of a user u is in the form of T_u=[$place_1 \rightarrow place_2 \rightarrow \ldots \rightarrow place_n$], where $place_i$ $(1 \leq i \leq n)$ denotes the name of the place where user u stayed for more than τ time (a time parameter used to distinguish passing points from stay points), and the arrows indicate the visiting order.*

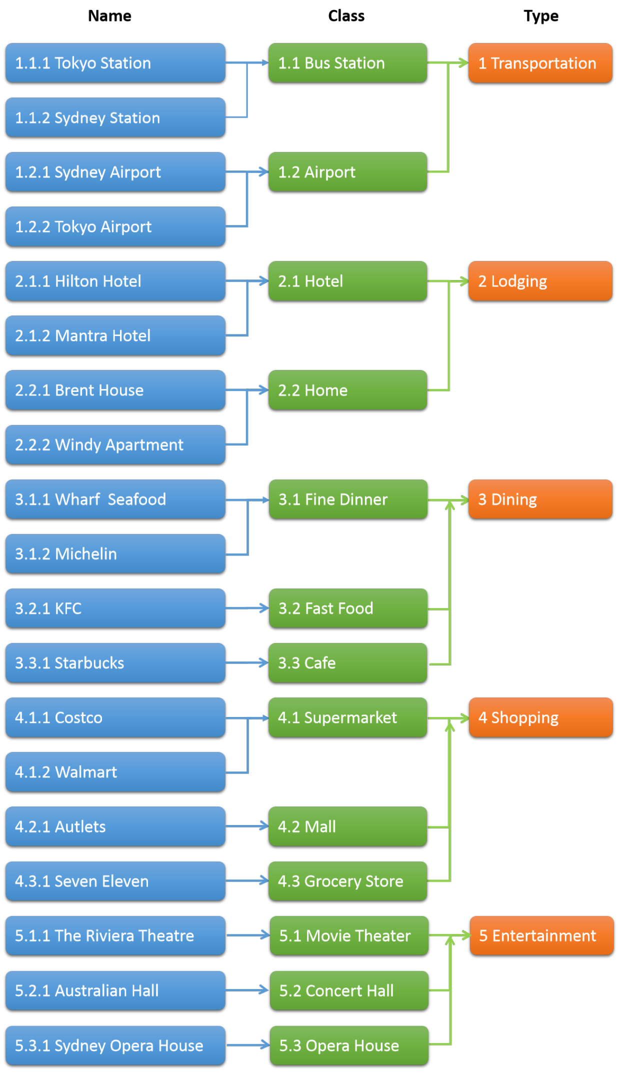


Fig. 3. An Example of Semantic Forest

The definition of the semantic trajectory considers mainly stay points, which helps the subsequent trajectory comparison to focus on meaningful places that a person has visited rather than just passed by briefly. Our representation capture the information of the length of stay by including the same place n times if the duration of the stay at the place is n times of the time interval τ. This trajectory representation integrates spatial and temporal information that makes our proposed hash function more effective. By extending the examples in Figure 1, the semantic trajectories of Carol and Dave can be represented as follows:

T_{Carol} = [Maris Apartment $\rightarrow$ Sydney Airport $\rightarrow$ O'Hare Airport $\rightarrow$ Tokyo Airport $\rightarrow$ Facebook Japan$\rightarrow$ KFC $\rightarrow$ Tokyo Airport $\rightarrow$ Sydney Airport $\rightarrow$ Maris Apartment].

T_{Dave} = [Windy Apartment $\rightarrow$ O'Hare Airport

$\rightarrow$ Paris-Charles De Gaulle $\rightarrow$ Microsoft France $\rightarrow$ Restaurant Goude $\rightarrow$ Paris Convention Center $\rightarrow$ Paris-Charles De Gaulle $\rightarrow$ O'Hare Airport $\rightarrow$ Windy Apartment].

From the above example, we can observe that Carol and Dave have a similar travel pattern between hotels and airports. However, such similarity cannot be identified if we simply try to match the place names in their trajectories. This is because they are visiting hotels and airports with different names that do not match directly. Therefore, besides the exact name matching, it is also important to consider the types of places. For this, we build a *semantic forest* to organize places into three levels from fine granularity to coarse granularity, i.e., place name, place class, and place type. Figure 3 shows an example of the semantic forest, which can be generated based on many existing ontology analysis of semantic meanings of words such as WordNet.

Each place in the semantic trajectory will be mapped to "class" and "type" levels in the semantic forest. For example, at the "class" level, Carol's trajectory will be represented as "(hotel, airport, airport, airport, company, fast food, company, hotel)" , while Dave's trajectory will be represented as "(home, airport, airport, company, fine dinner, convention, airport, airport, home)"; at "type" level, Carol's trajectory will be "(lodging, transportation, transportation, transportation, business, dining, business, lodging)" while Dave's trajectory will be "(lodging, transportation, transportation, business, dining, business, transportation, transportation, lodging)".

As a result, a single trajectory will have total three levels of representations: the original trajectory, the trajectory representation at "class" level and the trajectory representation at "type" level. The semantic similarity between a pair of trajectories is then defined as the integration of the similarity among three levels as shown in Definition 2.

Definition 2. *(Semantic Trajectory Similarity) Given a pair of semantic trajectories T_{u_1} and T_{u_2}, the semantic trajectory similarity between them is calculated as follows:* $SS(T_{u_1}, T_{u_2}) = \sum_{i=1}^{n}[w_i \cdot S(T_{u_1}^i, T_{u_2}^i)]$, *where $T_{u_1}^i$ and $T_{u_2}^i$ represents the trajectory mapping at level i among n levels in the semantic forest, w_i is the corresponding weight and $\sum_{i=1}^{n} w_i = 1$, and $S(T_{u_1}^i, T_{u_2}^i)$ denote the similarity of trajectories at level i.*

The above definition is generic that applies to any number of levels that a semantic forest may have. The weight w_i is a tunable parameter. By default, we set all the weights equal. If an application has special background knowledge or preference on fine or coarse grain of semantic levels, the weights can be adjusted to reflect the varied importance of different levels. In our discussion, we will use 3 levels for ease of illustration. The definitions of the similarity among each level will be elaborated in Section IV.3.

Next, we propose a novel approach to consolidate the three-level representation of a trajectory to make the subsequent trajectory comparison highly efficient. Specifically, we map each place to an encoding in the form of "$E_{type}.E_{class}.E_{name}$". For example, "Tokyo station" is encoded as "1.1.1", "Sydney station" is encoded as "1.1.2", and Sydney airport is encoded as "1.2.1". When looking at the entire encodings of the above three places, we can see that they are different since the place names are different. However, when only the first two components of the encodings are considered, we can observe the similarity between "Tokyo station" and "Sydney station" as their encodings of the first two components are the same, i.e., "1.1.x". Finally, since all the three places have the same first component "1.x.x" in their encodings, we also know that they belong to the same type. With the aid of the encoding, we will not need to store multiple representations for each trajectory. We only need to map the original trajectory to place encodings, and then the trajectory comparison at different levels can be conducted between the extracted corresponding level of encodings as shown in the above example. Note that this encoding strategy can be easily extended to n-level representations of a trajectory in the form of "$E_{l_1}.E_{l_2}...E_{l_n}$".

IV.2 Sequence-Sensitive Hashing (SSH)

Given a large amount of trajectory data, comparing each pair of trajectories to identify their similarity will be extremely time-consuming. For example, if there are 1 million (10^6) trajectories, there will be 1 trillion (10^{12}) pair-wise comparisons. To obtain a reasonable processing time, a natural idea is to adopt the divide-and-conquer method. This is easy when looking for the geometric similarity between trajectories whereby a large trajectory dataset can be split according to their regions (e.g., city or state), and then comparisons can be done within much smaller subsets of trajectories in parallel. As for semantic trajectories, spatial partitioning will not work since semantically similar trajectories could locate in different areas. The MinHashLSH function provided by Apache Spark does not work well either since MinHashLSH does not consider the frequency of the same type of places being visited. Thus, the partitioning of the semantic trajectories calls for a new strategy.

To conquer the aforementioned challenge, we propose a new hashing method, referred to as Sequence-Sensitive Hashing (SSH), which captures the semantic similarity among trajectories to help divide trajectories into semantic groups through only one scan of the trajectory dataset. Each semantic group contains trajectories that have similar types of places and are likely to have high similarity scores. Unlike geometric partitioning which does not have overlaps among partitions, our proposed semantic groups allow a trajectory to be classified as more than one semantic group. Such overlapping conforms with the real-world situation that a person's trajectory may demonstrate multiple characteristics, e.g., frequent flyer and businessman as shown in the previous example. Formally, our SSH method is built upon a new notion called *K-sequential Shingles* as defined in Definition 3.

***Definition* 3.** *(K-sequential Shingling) Given a trajectory T=[place$_1$ $\rightarrow$ place$_2$ $\rightarrow$ ... $\rightarrow$ place$_n$], its k-sequential Shingling is a set of m distinct shingles $G = \{g_1, g_2, ..., g_m\}$, where each shingle is formed by the types of k contiguous or non-contiguous places in T in the same order as they occur in the trajectory, i.e., g_i=[type$_{i_1}$ $\rightarrow$type$_{i_2}$...$\rightarrow$type$_{i_k}$], and $1 \leq i \leq m$, $m \leq \binom{n}{k}$.*

Figure 4 illustrates an example of 3-sequential shingles. The first two rows in the figure are the encoded trajectories of Carol and Dave, and the bottom two rows are their corresponding shingles formed by the types of places they have visited. Specifically, the types of places (i.e., the first encoding of a place) visited by Carol include: 2 $\rightarrow$1$\rightarrow$1$\rightarrow$1$\rightarrow$6$\rightarrow$3$\rightarrow$6$\rightarrow$2, as highlighted using different colors in T_{Carol}. The corresponding 3-shingles for the types of places are generated by selecting three place types from the trajectory while preserving their visiting sequence. For example, by selecting the 1st, 2nd and 3rd place types from the trajectory, we obtain the first 3-shingle "2 1 1" as denoted by orange and green colors, respectively; by selecting the 1st, 2nd and 5th place types, we obtain another 3-shingle "2 1 6". After collecting all the unique 3-shingles, we will obtain S_{carol} as shown in Figure 4.

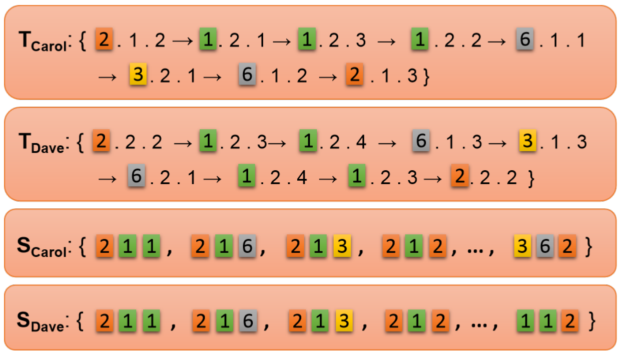

T_Carol: { 2 . 1 . 2 → 1 . 2 . 1 → 1 . 2 . 3 → 1 . 2 . 2 → 6 . 1 . 1 → 3 . 2 . 1 → 6 . 1 . 2 → 2 . 1 . 3 }
T_Dave: { 2 . 2 . 2 → 1 . 2 . 3 → 1 . 2 . 4 → 6 . 1 . 3 → 3 . 1 . 3 → 6 . 2 . 1 → 1 . 2 . 4 → 1 . 2 . 3 → 2 . 2 . 2 }
S_Carol: { 2 1 1 , 2 1 6 , 2 1 3 , 2 1 2 , ... , 3 6 2 }
S_Dave: { 2 1 1 , 2 1 6 , 2 1 3 , 2 1 2 , ... , 1 1 2 }

Fig. 4. An Example of 3-sequential Shingling

Observing the shingles in Figure 4, we can easily spot the similarities between the trajectories. In order to capture such similarities, we calculate the k-sequential shingles for each trajectory and then cluster

Algorithm 1: 3-sequential Shingling

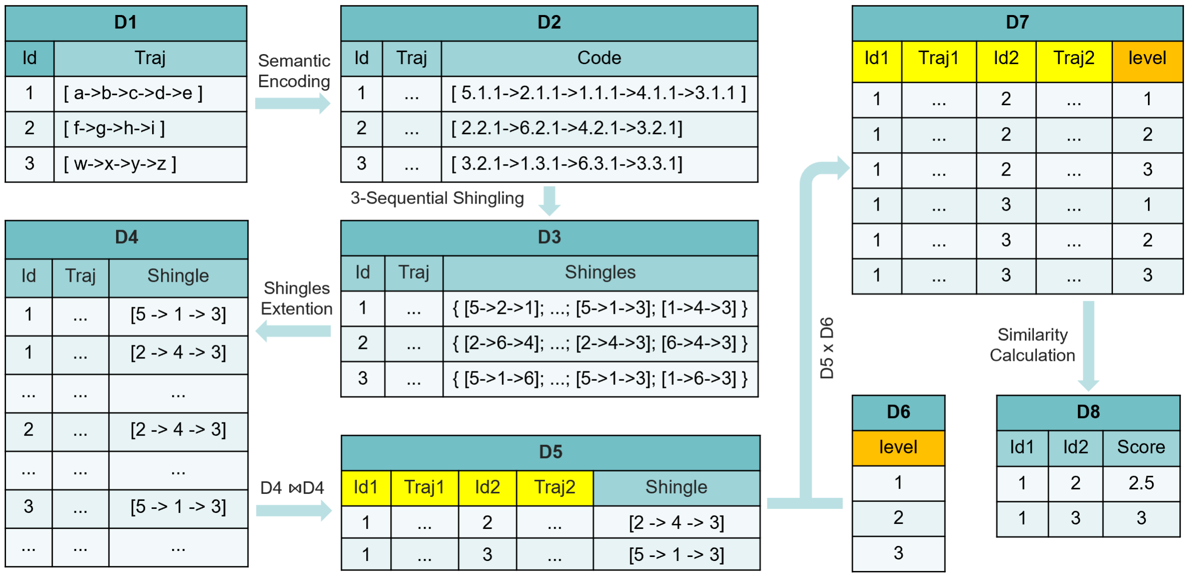
```
Input: A trajectory containing a sequence of
       location: $T = x_1, x_2, x_3, ..., x_l$
Output: A set of shingles: $S = \{s_1, s_2, ..., s_z\}$
1 for $i = 1; i \leq l; i++$ do
2     for $j = i+1; j \leq l; j++$ do
3         for $k = j+1; j \leq l; k++$ do
4             s=concat($x_i$.name||$x_j$.name||$x_k$.name);
              S.insert(s);
5 return $S$;
```

the trajectories which contain at least one identical shingle into the same partition (semantic group). Note that, if two trajectories have not even a single common k-sequential shingle, that means the two trajectories did not visit more than k same types of places in the same order, and hence they are not very similar and it is not necessary to further calculate their similarity score.

We proceed to present how to implement the Sequence-Sensitive Hashing (SSH) in Apache Spark. A straightforward method is to implement SSH as a user-defined function (UDF) in Apache Spark. However, we found that the use of UDF does not fully take advantage of Apache Spark's parallel computing capabilities due to the limitation of Apache Spark's abilities to understand arbitrary UDF codes. Also, it is hard for UDF to benefit from the Apache Spark query optimizer called Catalyst which keeps improving Apache Spark's built-in functions with each new version release. As such, we design the following algorithm that leverages Apache Spark's built-in SQL query functions to conduct the proposed trajectory partitioning task.

Figure 5 shows an example of hashing process. The original trajectories are stored in a data frame (denoted as D_1) along with its unique trajectory ID. For each trajectory, its semantic encoding is appended as a new column in D_2. Based on the semantic encoding, we calculate the k-sequential shingles for each trajectory and store the results in the data frame D_3. In D_3, each trajectory may contain various numbers of shingles. We further expand D_3 into a new data frame D_4 by listing each trajectory shingle as a separate row. Then, we can leverage the Apache Spark join function to join the D_4 with itself to identify trajectories containing identical shingles. The join results are stored in another data frame (D_5). Each row of D_5 contains a pair of trajectories that are likely to be similar to each other. After that, we will use D_6, D_7 and D_8 to calculate the final similarity scores which will be elaborated in the next subsection.

We now take a deep look at the SSH. Given Algorithm 1, let N be the number of trajectories and



D1

Id	Traj
1	[a->b->c->d->e]
2	[f->g->h->i]
3	[w->x->y->z]

D2

Id	Traj	Code
1	...	[5.1.1->2.1.1->1.1.1->4.1.1->3.1.1]
2	...	[2.2.1->6.2.1->4.2.1->3.2.1]
3	...	[3.2.1->1.3.1->6.3.1->3.3.1]

D3

Id	Traj	Shingles
1	...	{ [5->2->1]; ...; [5->1->3]; [1->4->3] }
2	...	{ [2->6->4]; ...; [2->4->3]; [6->4->3] }
3	...	{ [5->1->6]; ...; [5->1->3]; [1->6->3] }

D4

Id	Traj	Shingle
1	...	[5 -> 1 -> 3]
1	...	[2 -> 4 -> 3]
...	...	...
2	...	[2 -> 4 -> 3]
...	...	...
3	...	[5 -> 1 -> 3]
...	...	...

D5

Id1	Traj1	Id2	Traj2	Shingle
1	...	2	...	[2 -> 4 -> 3]
1	...	3	...	[5 -> 1 -> 3]

D6

level
1
2
3

D7

Id1	Traj1	Id2	Traj2	level
1	...	2	...	1
1	...	2	...	2
1	...	2	...	3
1	...	3	...	1
1	...	3	...	2
1	...	3	...	3

D8

Id1	Id2	Score
1	2	2.5
1	3	3

Fig. 5. An Overview of Hashing Process (D1: Dataframe containing original trajectories; D2: Datafame after semantic encoding; D3: Dataframe with calculated 3-Sequential Shingling; D4: Dataframe that associates each individual shingle with its trajectory to prepare for the subsequent join operation; D5: Join result that contains pairs of trajectories with matching shingles; D6: Dataframe containing the total number of semantic levels (which is 3 in the example) to prepare for the subsequent cross product operation; D7: Each pair of trajectory with matching shingles is assigned to all the semantic levels for final similarity calculation at different semantic levels; D8: Sum up the similarity scores of each pair of trajectories in D7 at all the semantic levels.)

L be the average number of locations in a trajectory. The computational complexity of Sequential-Sensitive Hashing is $O(N * L^3)$. Since the total number of trajectories is much larger than the average number of locations in a trajectory, i.e., $N >> L$, the actual computational complexity of SSH is $O(N)$ which is linear to the data size.

Next, we examine the collision rate of the SSH. In our problem, the SSH is used to help group similar trajectories. Therefore, we do not want the collision rate to be too small. For example, if the collision rate is 0, it means each trajectory is in its own group and we will not be able to find any similar trajectories. Assume there are Q items in total in the type level of semantic forest. We will obtain at most Q^K different hash values by applying K-sequential Shingling for all. Each trajectory will have $\binom{L}{K}$ shingles. On average, the collision rate is $\binom{L}{K}/Q^K$.

IV.3 Semantic Trajectory Similarity Computation

After the trajectory partitioning, we only need to compare similarities among trajectories within the same partition. In order to better capture the semantic similarity of trajectories that contain repetitive visits to the same type of places as discussed in the introduction, we introduce a new of semantic trajectory similarity as follows.

Definition 4. *(Multi-Level Semantic Trajectory Similarity) Let* $ET_u = \{E^{l_1}_{u_1}.E^{l_2}_{u_1}...E^{l_n}_{u_1}, ..., E^{l_1}_{u_k}.E^{l_2}_{u_k}...E^{l_n}_{u_k}\}$, $ET_v=\{E^{l_1}_{v_1}.E^{l_2}_{v_1}...E^{l_n}_{v_1}, ..., E^{l_1}_{v_m}.E^{l_2}_{v_m}...E^{l_n}_{v_m}\}$, *denote the semantic encodings of user u's and v's trajectories where k and m refer to the number of places visited by each user, respectively. The multi-level semantic trajectory similarity (MSS) is defined as:*

$$MSS(ET_u, ET_v) = \sum_{h=1}^{n} \beta_h \cdot |M_h|$$

where M_h contains the maximum matching encoding sequence in two trajectories at level h, β_h denotes the importance of the similarity at level l_i, and $\sum_{i=1}^{n} \beta_{l_i} = 1$.

Considering the three-level semantic hierarchy, two points that match at type level do not necessarily match at class or place level. However, two points matching at class level must match at type level and two points matching at place level must match at class level and type level. Therefore, $|M_{typ}| \geq |M_{cls}| \geq |M_p|$. In addition, $\beta_{typ} + \beta_{cls} + \beta p = 1$, so given a similarity threshold $n \leq \rho < n+1$, where n is a positive integer denoting the lower bound of ρ. Trajectories with a similarity score $MSS > \rho$ has $|M_{typ}|$ matching points at type level, and $n \leq MSS \leq |M_{typ}|$. In this way, our Sequential Sensitive Hashing applying k-sequential shingle function at type level with a $k \leq n+1$ is able to cluster all the possible similar pairs of trajectories satisfying the condition above into partitions, and it performs most efficiently when $k = n+1$.

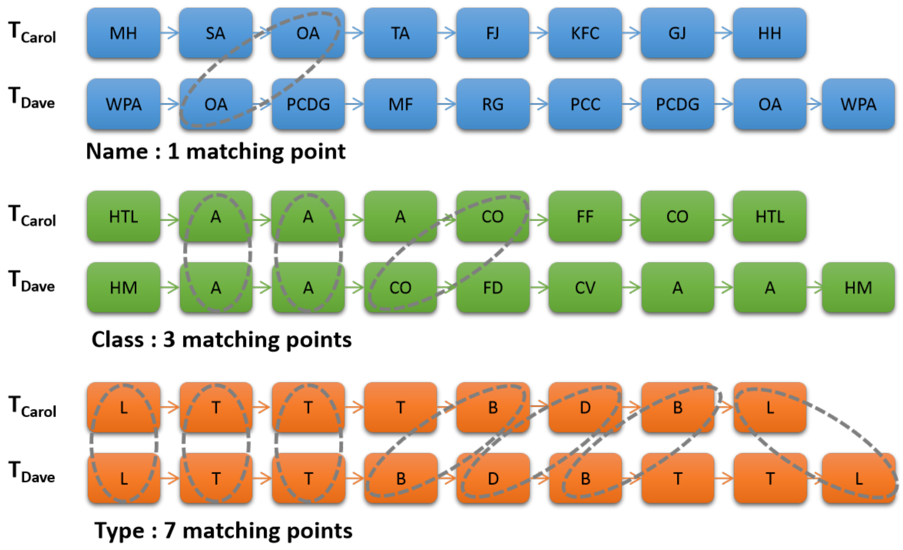


Fig. 6. An Example of Multi-level Semantic Similarity (After mapping Carol's and Dave's trajectory to different semantic levels, there are one pair of matching locations at the lowest semantic level (i.e., location name level), 3 pairs of matching locations at the 2nd semantic level (i.e., location class level), and 7 pairs of matching locations at the highest semantic level (i.e., location type level).)

$$M_h(ET_u^h, ET_v^h) = \begin{cases} \emptyset, & if \;\; i = 0 \;\; or \;\; j = 0; \\ M_h(ET_{u_{i-1}}^h, ET_{v_{j-1}}^h) \bigoplus h_{u_i}, & if \;\; h_{u_i} = h_{v_j}; \\ max\{M_h(ET_{u_{i-1}}^h, ET_{vj}^h), M_h(ET_{u_i}^h, ET_{v_{j-1}}^h)\}, & others. \end{cases} \quad (1)$$

```
Algorithm 2: Sequence Sensitive Hashing
Input: A Dataframe containing all semantic
       trajectories: $D_1 = \{T_1, T_2, ..., T_N\}$
Output: A Dataframe containing all pairs of
        candidate similar trajectories: $D_5$
1 Encode the place names of each trajectory in
   Dataframe $D_1$ into corresponding place types
   in semantic trees, get Dataframe $D_2$;
2 Use 3-Sequential Shingling function to
   calculate a set of shingles of each trajectory in
   Dataframe $D_2$ and add a new column named
   "shingles" to store it, get Dataframe $D_3$;
3 Extend the Dataframe $D_3$ into a new
   Dataframe $D_4$ by separating the elements of
   shingles set of every trajectory into different
   rows;
4 Join Dataframe $D_4$ with itself on condition: left
   $D_4$.id != right $D_4$.id and left $D_4$.shingles =
   right $D_4$.shingles, get Dataframe $D_5$;
5 return $d_5$;
```

Definition 5. *(Maximum Matching Encoding Sequence) Let ET_u^h = $\{h_{u_1}, h_{u_2}, ..., h_{u_k}\}$, ET_v^h=$\{h_{v_1}, h_{v_2}, ..., h_{v_m}\}$ denote the semantic encodings of user u's and v's trajectories at level h. Let $M_h(ET_{u_i}^h, ET_{v_j}^h)$ represents the longest common subsequence of prefixes till h_u^i and h_v^j in the two trajectories, where $0 \le i \le k$, $0 \le j \le m$, and k and m refer to the number of places visited by the two users. The maximum matching encoding sequence is defined in Equation 1, whereby "$\bigoplus$" denotes symmetric difference of two sets.*

Figure 6 shows an example of the multi-level semantic trajectory similarity calculation. For convenience, place names, classes, and types are represented using their abbreviations. In this example, the number of matching points increases as the semantic meaning becomes more general. Specifically, there is only one matching place in Carol's and Dave's trajectories, but three places with matching classes, and seven places with matching types. Thus, if given the importance of the similarity at different level as $\beta_{typ} = 0.2$, $\beta_{cls} = 0.3$, and $\beta_p = 0.5$, the similarity score MSS(ET_{Carol}, ET_{Dave}) = $\beta_{typ} \cdot 7 + \beta_{cls} \cdot 3 + \beta_p \cdot 1$ = 0.2×7+0.3×3+0.5×1=2.8. Here, the weight values β are chosen in a descending order of their importance whereby the highest weight value is given to the lowest semantic level, i.e., the place level.

It is worth noting that our algorithm is capable of capturing the repetition of visits to the same places. However, existing definitions of semantic trajectory similarity will consider the same place only once [1, 40] which will not be able to distinguish frequent visits from occasional visits, such as frequent flyers vs. occasional travelers, customers to a shopping mall vs. sale persons in a shopping mall.

As shown in Figure 5, the calculation of the similarity score is realized as follows in the Spark. We create a new data frame D_6 that has one column named "level", and three rows with values: 1, 2, and 3 which represent the "type", "class", and "name" level in the semantic forest respectively. We then multiple D_5 with D_6 to obtain a new data frame D_7. In D_7, each row is a concatenation of two trajectories and the semantic level ID (e.g., 1, 2, or 3) at which we want to calculate the similarity score for these two trajectories. This multiplication helps separate the similarity calculation at different semantic levels, so the parallel computing capability of Spark can be better utilized when we calculate the semantic similarity score for each row in D_7. We append a new column "score" to D_7 and obtain a new data frame D_8. The last step is to apply the Apache Spark's aggregation function to sum the scores for each pair of id_1 and id_2, which yields the final trajectory similarity score.

In addition, the same pairs of trajectories may occur in multiple partitions. For example, Carol's and Dave's trajectories may appear together in two partitions: one is for frequent flyers and the other is for businessmen. In this case, our approach will ensure that the similarity between the same pair of trajectories will be calculated only once. Specifically, each pair of the trajectories will be loaded to the main memory only once no matter how many partitions they belong to. This is achieved by joining two candidate trajectories that share any identical shingles, but no matter how many shingles they share. Our approach can save a lot of redundant calculations on duplicate pairs.

IV.4 Algorithm Outline

We now summarize the AnotherMe algorithm. Given a set of semantic trajectories, our algorithm starts from converting each trajectory into its corresponding semantic encoding. Based on the semantic encoding, we further calculate the k-sequential shingles for each trajectory. Then, we group trajectories which have at least one identical shingle into the same partition. Up to this point, we obtain multiple data partitions, and these partitions may contain overlapping trajectories since one person may belong to multiple social groups. Finally, inside each partition, we leverage the Spark built-in functions to calculate the similarity scores of each pair of trajectories. The obtained similarity scores can be further used to identify communities of interests, each of which contains a group of trajectories that are similar to one another.

V Experimental Studies

In this section, we first introduce experimental settings and then report the comparison results of our approach against centralized approaches and the default hashing function in Apache Spark.

V.1 Experimental Settings

We have implemented our AnotherMe algorithm in Spark. The testing environment consists of 1 master node and 1 to 20 worker nodes. Each node has 8 CPUs with Intel Xeon Cascade Lake 8255C (2.5 GHz), 32G of memory and 1T disk space available. The number of executor in Spark is the same as the number of worker nodes.

We compare our proposed AnotherMe algorithm with the following four algorithms.

- Centralized: This is a centralized approach that performs the calculation on the entire dataset. It calculates the semantic similarity of each pair of trajectories.
- MinHash: This is a distributed approach that employs the MinHashLSH function provided by Apache Spark to generate hash values for each trajectory and group trajectories with similar hash values for further similarity calculation using the function defined in Definition 4. Specifically, it first builds a vocabulary of place types and then encodes each trajectory at the type level into a single binary vector. In the binary vector, non-zero values are used to represent the presence of place type in the corresponding trajectory. The MinHashLSH is applied to the trajectory encodings to obtain hash values.
- BRP: This is also a distributed approach that employs another hash function, namely Bucketed Random Projection (BRP), provided by Apache Spark.
- User-defined: We implemented another version of our algorithm by enclosing all the tasks as a stand-alone user-defined function in Apache Spark. This algorithm has the same logic (i.e., the same four phases) as AnotherMe with the only difference in implementation. Recall that AnotherMe breaks down the tasks into subtasks that can leverage the built-in functions such as group-by in Apache Spark.

We test the performance of different approaches using both synthetic and real datasets. The synthetic dataset contains up to 1 million trajectories. The length of each trajectory is defined as the number of semantic locations visited, which varies from 5 to 10 to mimic the average person's daily travel patterns. Each location on the trajectories is randomly selected from 10,000 places. The number of synthetic place type is 30 and the number of classes in each type is 10. We set the importance of the similarity at different level as $\beta_{typ} = \beta_{cls} = \beta p = 1/3$. Besides synthetic datasets, there are not any real datasets of semantic trajectories. The largest real trajectory dataset we can find is GeoLife [19, 41, 42], which contains 17,621 trajectories collected from 182 users by GPS devices. The trajectories in GeoLife are a sequence of GPS locations. In order to carry out semantic queries, we first detect stay points in the given GPS trajectories [1, 40] and then map the stay points to semantic names. Finally, we assign each converted semantic trajectory a user ID.

By default, we apply 3-Sequential Shingling in our SSH algorithm which means that trajectories with less than three matching locations will not be considered for further analysis. The reason to use at least 3-sequential shingling is due to the consideration that most of the trajectories already contain at least two common types of places, i.e., home and workplace. Two similar trajectories are expected to visit at least one more additional type of place. In terms of the final query result, we only report to the query issuer those trajectories with a similarity score greater than threshold 2.

The performance is evaluated using two major criteria: (i) CPU time; (ii) correctness. The correctness is defined by comparing the query output obtained from the distributed approaches with that obtained from the centralized approach. The centralized approach calculates the similarity scores among all pairs of trajectories, based on which it constructs maximal cliques that connect users whose trajectories' similarity scores are above a threshold to form communities of interests. The communities of interests identified by the centralized approach is used as the ground truth to evaluate the correctness of the results produced by distributed approaches. The distributed approaches will split the dataset into subsets and then calculate the similarity among trajectories in each subset. If the splitting strategy is not well designed, it is likely that some similar trajectories are classified into different subsets and missed in the query result. Formally, the query correctness is evaluated using QA_1, QA_2, QP_1, QP_2 as follows:

$$QA_1 = \frac{|N_{dis} \bigcap N_{cen}|}{|N_{cen}|} \tag{2}$$

In the equation, QA_1 is defined as the number of matching communities of common interests obtained from the distributed and centralized approach divided by the number of communities of common interests found by the centralized approach (N_{cen}). The use of QA_1 aims to match how the real-world applications could use the semantically similar trajectories for friends or social circle recommendations. Besides this QA_1 metric, we also adopt another direct metric QA_2 that simply compares the total number of similar trajectory pairs identified by the centralized approach and distributed approaches. Equation 3 provides the formal definition of QA_2, whereby ST_{dis} and ST_{cen} denote the number of similar trajectory pairs output by the distributed and centralized approaches, respectively.

$$QA_2 = \frac{|ST_{dis} \bigcap ST_{cen}|}{|ST_{cen}|} \tag{3}$$

QA_1 and QA_2 aim to check the percentage of answers found by the centralized algorithm that can also be identified by the distributed algorithm. In other words, this aims to evaluate whether the distributed algorithm would miss answers. In addition to this, we introduce two more metrics QP_1 and QP_2 that aim to check whether the distributed algorithm introduces additional wrong answers that are not included in the results generated by the centralized algorithm. These four metrics together provide a full examination of the correctness of answers produced by distributed algorithms.

$$QP_1 = \frac{|N_{dis} \bigcap N_{cen}|}{|N_{dis}|} \tag{4}$$

$$QP_2 = \frac{|ST_{dis} \bigcap ST_{cen}|}{|ST_{dis}|} \tag{5}$$

V.2 Comparison with the Centralized Approach

In the first round of experiments, we compare the four approaches by varying the number of trajectories from 10,000 to 60,000. Note that 60,000 is the maximum number of trajectories that the centralized approach can be tested in a single computing node without encountering the problem of memory explosion and requiring hours for a single run.

Figure 7 shows the overall processing time of the five algorithms in a single node: Centralized, MinHash, BRP, User-defined, and AnotherMe. The processing time taken by Centralized and User-defined approaches increases exponentially with the data size. They are extremely slower than MinHash, BRP, and AnotherMe. It is not surprising to see that the centralized approach requires extensive time to calculate the similarity scores between each pair of trajectories in the dataset whereas other approaches only need to compare smaller sets of candidate trajectories that are likely to be similar. At a closer look at these approaches, Figure 8 shows the actual number of trajectories being compared by each approach, and Figure 9 shows the portion of time taken by hash functions in each distributed approach. We can observe that the AnotherMe approach is slightly slower than MinHash and BRP when the dataset size increases. This is not because the computation efficiency of AnotherMe decreases. This is because MinHash and BRP calculate smaller number of similar trajectories since these two

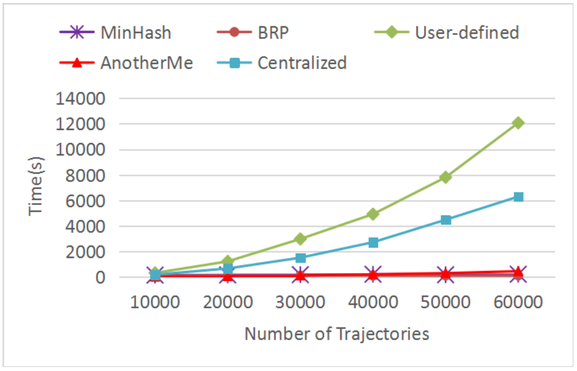


Fig. 7. Overall Processing Time when Varying the Number of Trajectories

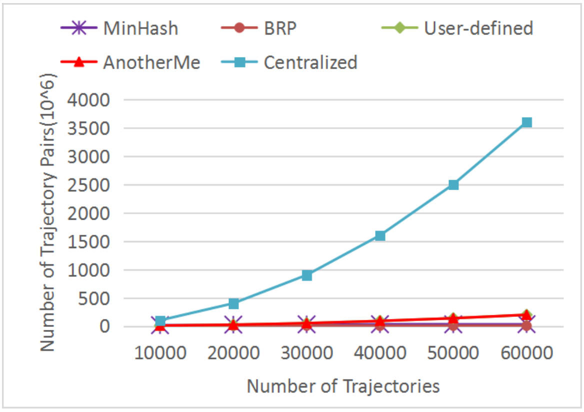


Fig. 8. Number of Trajectory Pairs Being Compared when Varying the Number of Trajectories

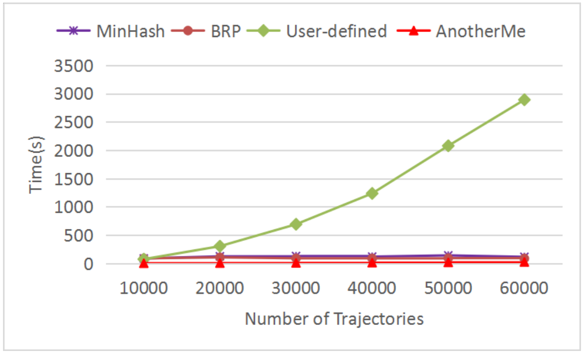


Fig. 9. Time Overhead Incurred by Hash Functions

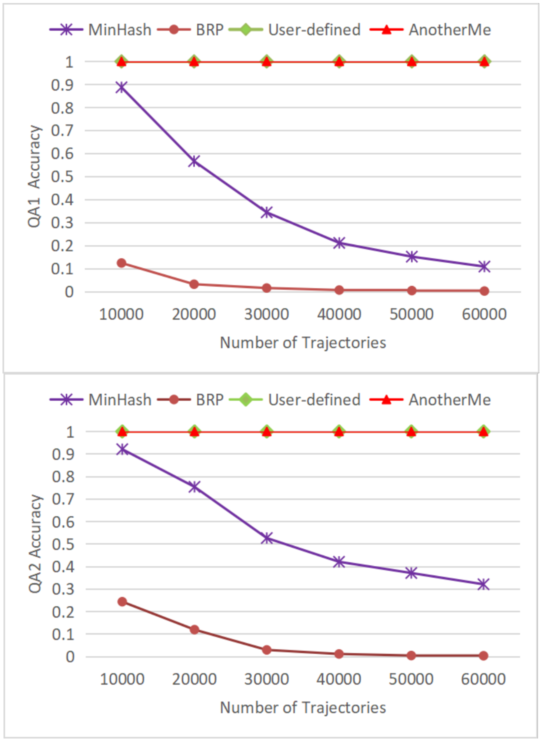


Fig. 10. QA_1 and QA_2 When Varying the Number of Trajectories

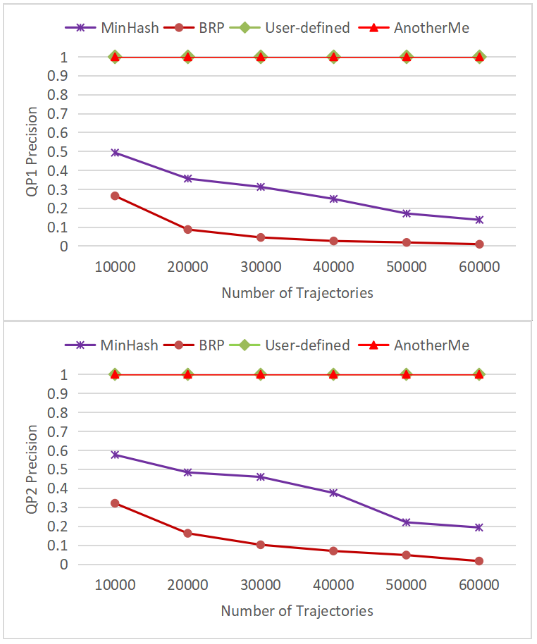


Fig. 11. QP_1 and QP_2 When Varying the Number of Trajectories

hash functions cannot accurately find out all similar trajectory pairs, and hence seem to be faster.

An interesting phenomenon in this experiment is that the User-defined approach in Apache Spark becomes even slower than the centralized approach when the dataset size increases. Note that each test dataset fits in the main memory during the comparison, which rules out the potential cause of the performance degradation of the User-defined approach due to the delay of data being swapped between memory and disk. We may have expected that a distributed approach would be generally faster than a centralized approach. However, our new finding is that Apache Spark simply treats user-defined functions as a black box and is not able to optimize and fully parallelize the processes in the user-defined functions. Also, due to the extra environmental setting overhead in Spark, the user-defined function approach becomes less efficient than the lightweight centralized approach.

From the experiments, we also observe that our proposed AnotherMe approach is not only very fast but also preserves the correctness of the answers. As shown in Figure 10 and Figure 11, the AnotherMe approach identifies exactly same number of communities of interests measured by QA_1 and QP_1, and exactly same number of similar trajectories measured by QA_2 and QP_2 when compared to the centralized algorithm. The results demonstrate that the distributed algorithm–AnotherMe–does not sacrifice any data quality while achieving high efficiency and scalability. Such performance is attributed to the novel design of AnotherMe's hash function which does not split similar trajectories into different data partitions. Also, our shingles cover all possible cases during the hashing. The user-defined approach also has high correctness as it is simply another implementation of the same logic as AnotherMe. The results clearly indicate that our proposed Sequence-Sensitive Hashing (SSH) very well captures the similarity among trajectories. In contrast, the percentage of correct answers produced by the default Spark hash functions (MinHash and BRP) decreases dramatically with the number of trajectories increases. This is mainly because that the default Spark hash functions are not tailored for the proposed trajectory similarity comparison. The default hash functions do not capture the visiting order of different places, and hence it is very likely that the default hash functions generate similar hash values for trajectories which are indeed different because of the order of places visited. As a result, the correctness of the answers yielded by MinHash and BRP is very low. The BRP misses almost all the correct communities and similar trajectory pairs.

V.3 Evaluating Real Dataset

In this round of experiments, we examine the performance in the real dataset, the GeoLife as described in the experimental settings. We compare all the approaches except the BRP which is not able to correctly detect most of the communities of interests. Figure 12 shows the overall processing time. As we can see from the figure, our proposed AnotherMe approach is the most efficient, faster than MinHash, and 10 times faster than the centralized approach. This indicates the practicality of our approach in the real world. In addition, we also observe that the user-defined approach is faster than the centralized approach in this case. This is mainly because the size of the real dataset is not very large, and hence not many computing nodes are needed for the user-defined approach, i.e., less communication overhead.

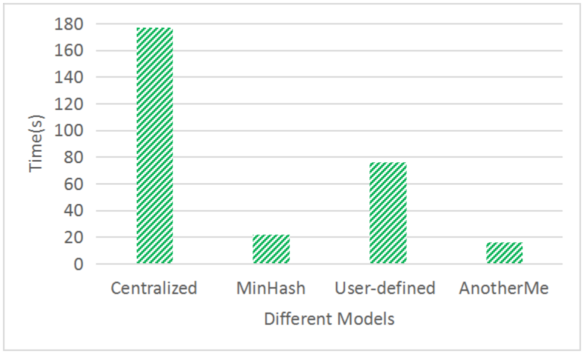


Fig. 12. Overall Processing Time of Different Approaches in Real Datasets

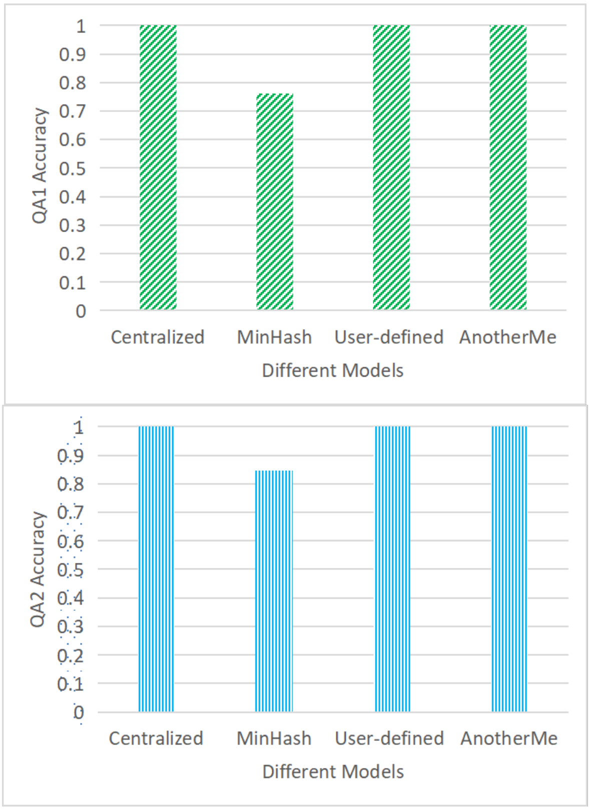


Fig. 13. QA_1 and QA_2 of Different Approaches in Real Datasets

Next, we examine the correctness of answers produced by all the approaches. Using the results generated by the centralized algorithm as the ground truth, we adopt the metrics QA_1, QA_2 to check if a distributed approach misses any correct answers (Figure 13), and adopt the metrics QP_1 and QP_2 to check if a distributed approach added wrong answers (Figure 14). From the figures, we can observe that our

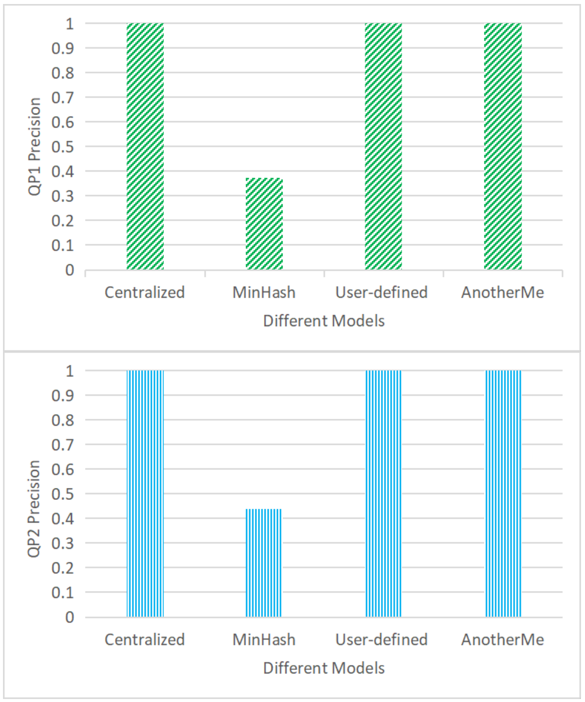


Fig. 14. QP_1 and QP_2 of Different Approaches in Real Datasets

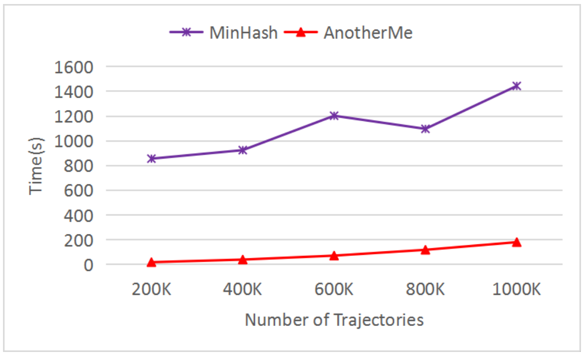


Fig. 15. Processing Time when Varying the Data Size

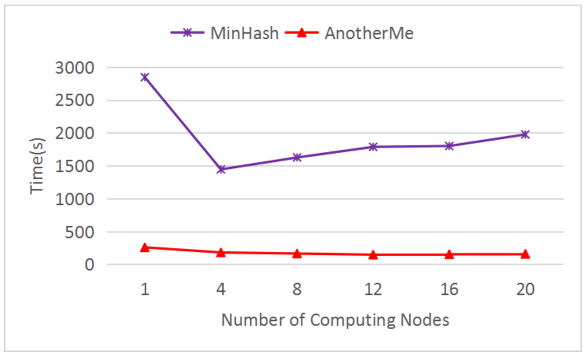


Fig. 16. Processing Time when Varying the Number of Computing Nodes

AnotherMe algorithm and its variant implementation that uses the user-defined function again achieve 100% correctness in all metrics. However, the default Apache Spark hash functions, the MinHash and BRP, not only missed a large percentage of correct answers as indicated by low QA_1 and QA_2 values in Figure 13, but also added a large number of wrong answers as indicated by low QP_1 and QP_2 values in Figure 14. That means these generic hash functions provided by Spark are not a good fit for the semantic trajectory analysis.

V.4 Evaluating Scalability

In this round of experiments, we aim to examine the scalability of the distributed approaches. We only compare the MinHash and our proposed AnotherMe approaches, since the centralized approach and user-defined function approach are no longer able to process large-scale datasets within hours, and the BRP approach has extremely low accuracy. The number of trajectories being tested has been increased up to 1 million and the number of different place types is set to 300.

Figure 15 reports the overall processing time of the MinHash and our AnotherMe algorithm when 4 worker nodes are used. We can clearly see that our approach is orders of magnitude faster than MinHash. The performance gain achieved by our AnotherMe approach should be attributed to the essential design spirit of AnotherMe which carefully maps the subtasks to highly-optimized Apache Spark's built-in operations on data frames. Moreover, since the number of place types is larger than that tested in smaller dataset, the benefits of our hashing algorithm in terms of capturing similar trajectory pairs and balancing the workload among nodes become more prominent.

In Figure 16, we vary the number of worker nodes from 1 to 20 and test the dataset of 1 million trajectories. Both approaches speed up with the increase of the number of worker nodes at first and then the performance does not further improve much. This is because of the effect of two factors. The more worker nodes, the fewer number of calculations to be conducted in individual nodes. However, more worker nodes also introduce more communication overhead such as data shuffling among nodes, which in turn introduces additional processing time. In our experiments, MinHash reaches its best performance when 4 worker nodes are employed, while AnotherMe reaches its best performance when 12 worker nodes are employed.

V.5 Evaluating Effect of Semantic Levels

In the previous experiments, we have adopted the 3-level semantic hierarchy. In this last round of experiments, we are interested in investigating the generality of our algorithm in future applications which may use

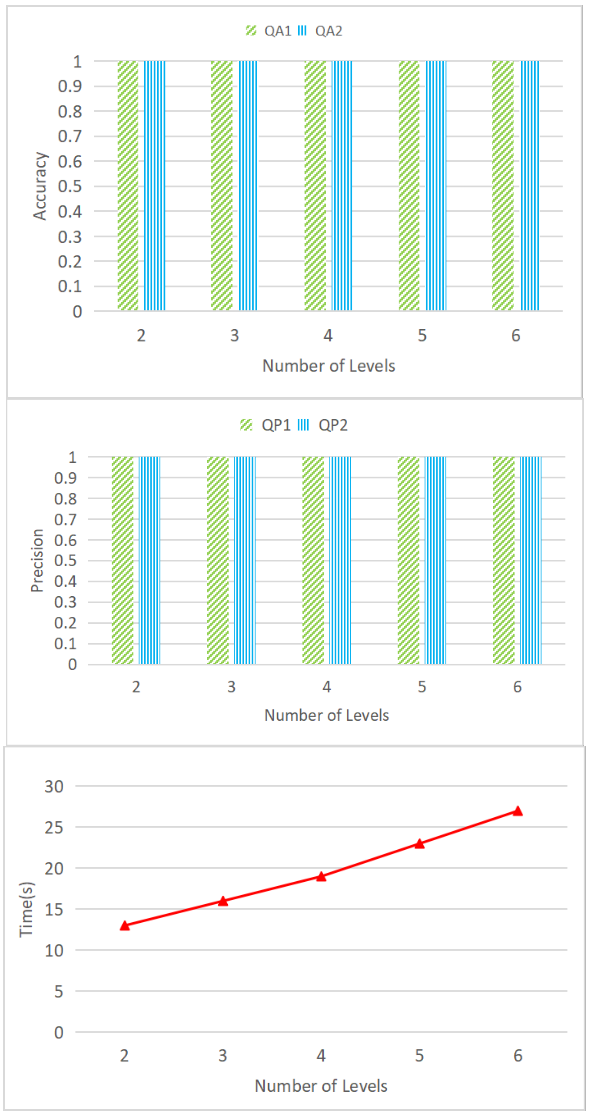


Fig. 17. Effect of the Number of Semantic Levels

different numbers of semantic levels. The experiments are conducted on the real trajectory dataset (GeoLife), whereby we adopt semantic hierarchy with levels ranging from 2 to 6. The accuracy is measured by comparing the number of communities and the number of similar trajectory pairs found by our algorithm with the centralized approach. Figure 17 shows the overall performance of our approach in terms of both correctness and time efficiency. As we can see that our algorithm performs consistently regardless the number of semantic levels. Specifically, when similarity measures are defined over different number of semantic levels, our algorithm still identifies the same similar trajectory pairs and communities as the centralized algorithm. This indicates certain robustness and generality of our algorithm. In addition, we also evaluate the processing time of our algorithm under these different settings. It is not surprising to see an increase of the time when the number of semantic levels grows. This is because our hashing needs to be performed at each level of the semantic hierarchy, and hence more levels will need more time to process.

VI Conclusion

In this paper, we propose an efficient large-scale semantic trajectory analysis algorithm (dubbed AnotherMe) on the Apache Spark platform. Specifically, we design a novel hash function to effectively produce initial partitions of trajectories that are likely to be semantically similar. Then, we calculate trajectory similarity scores at different semantic granularity levels in each partition in parallel, which significantly improves the comparison speed. The experimental results show that our approach is orders of magnitude faster than centralized approaches while preserving the same accuracy. The obtained similarity scores between trajectories are beneficial for identifying communities of common interests across the physical boundaries. In addition, our proposed multi-level similarity metric and sequence-sensitive hashing technique may also be useful for other applications that need sequence analysis. For example, our approach may be leveraged to analyze the multiple steps that malware is taking to perform infection and identify similar malware behavior. The potential of the proposed technique is worth exploring in various application domains.

References

[1] Xiangye Xiao, Yu Zheng, Qiong Luo, and Xing Xie. Finding similar users using category-based location history. In *Proceedings of the 18th SIGSPATIAL international conference on advances in geographic information systems*, pages 442–445, 2010.

[2] Xiangye Xiao, Yu Zheng, Qiong Luo, and Xing Xie. Inferring social ties between users with human location history. *Journal of Ambient Intelligence and Humanized Computing*, 5(1):3–19, 2014.

[3] Haitao Yuan and Guoliang Li. Distributed in-memory trajectory similarity search and join on road network. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1262–1273. IEEE, 2019.

[4] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[5] Huiping Cao, Nikos Mamoulis, and David W Cheung. Mining frequent spatio-temporal sequential patterns. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[6] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339, 2007.

[7] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *Acm Sigmod Record*, 23(2):419–429, 1994.

[8] Yueguo Chen, Mario A Nascimento, Beng Chin Ooi, and Anthony KH Tung. Spade: On shape-based pattern detection in streaming time series. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 786–795. IEEE, 2007.

[9] Helmut Alt. The computational geometry of comparing shapes. In *Efficient Algorithms*, pages 235–248. Springer, 2009.

[10] Tetsuya Nakamura, Keishi Taki, Hiroki Nomiya, Kazuhiro Seki, and Kuniaki Uehara. A shape-based similarity measure for time series data with ensemble learning. *Pattern Analysis and Applications*, 16(4):535–548, 2013.

[11] Lei Chen, M Tamer Özsu, and Vincent Oria. Symbolic representation and retrieval of moving object trajectories. In *Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval*, pages 227–234, 2004.

[12] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings 18th international conference on data engineering*, pages 673–684. IEEE, 2002.

[13] Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S Tseng. Mining user similarity from semantic trajectories. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, pages 19–26, 2010.

[14] Wuman Luo, Haoyu Tan, Lei Chen, and Lionel M Ni. Finding time period-based most frequent path in big trajectory data. In *Proceedings of the 2013 ACM SIGMOD international conference on management of data*, pages 713–724, 2013.

[15] Sheng Wang, Zhifeng Bao, J Shane Culpepper, Timos Sellis, Mark Sanderson, and Xiaolin Qin. Answering top-k exemplar trajectory queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 597–608. IEEE, 2017.

[16] Albino Altomare, Eugenio Cesario, Carmela Comito, Fabrizio Marozzo, and Domenico Talia. Trajectory pattern mining for urban computing in the cloud. *IEEE Transaction on Parallel and Distributed Systems (TPDS)*, 28(2):586–599, 2017.

[17] Christian S Jensen, Dan Lin, and Beng Chin Ooi. Continuous clustering of moving objects. *IEEE transactions on knowledge and data engineering*, 19(9):1161–1174, 2007.

[18] Yu Zheng, Like Liu, Longhao Wang, and Xing Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *Proceedings of the 17th international conference on World Wide Web*, pages 247–256. ACM, 2008.

[19] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. Understanding mobility based on gps data. In *Proceedings of the 10th international conference on Ubiquitous computing*, pages 312–321. ACM, 2008.

[20] Yu Zheng, Yukun Chen, Quannan Li, Xing Xie, and Wei-Ying Ma. Understanding transportation modes based on gps data for web applications. *ACM Transactions on the Web (TWEB)*, 4(1):1, 2010.

[21] Chao Zhang, Jiawei Han, Lidan Shou, Jiajun Lu, and Thomas La Porta. Splitter: Mining fine-grained sequential patterns in semantic trajectories. *Proceedings of the VLDB*, 7(9):769–780, 2014.

[22] Dong-Wan Choi, Jian Pei, and Thomas Heinis. Efficient mining of regional movement patterns in semantic trajectories. *Proceedings of the VLDB Endowment*, 10(13):2073–2084, 2017.

[23] Mete Celik and Ahmet Sakir Dokuz. Discovering socially similar users in social media datasets based on their socially important locations. *Information Processing & Management*, 54(6):1154–1168, 2018.

[24] You Wan, Chenghu Zhou, and Tao Pei. Semantic-geographic trajectory pattern mining based on a new similarity measurement. *ISPRS International Journal of Geo-Information*, 6(7):212, 2017.

[25] Chongming Gao, Zhong Zhang, Chen Huang, Hongzhi Yin, Qinli Yang, and Junming Shao. Semantic trajectory representation and retrieval via hierarchical embedding. *Information Sciences*, 538:176–192, 2020.

[26] Sotiris Angelis, Konstantinos Kotis, and Dimitris Spiliotopoulos. Semantic trajectory analytics and recommender systems in cultural spaces. *Big Data and Cognitive Computing*, 5(4), 2021.

[27] Guochen Cai, Kyungmi Lee, and Ickjai Lee. Itinerary recommender system with semantic trajectory pattern mining from geo-tagged photos. *Expert Systems with Applications*, 94:32–40, 2018.

[28] Yang Cao, Fei Xue, Yuanying Chi, Zhiming Ding, Limin Guo, Zhi Cai, and Hengliang Tang. Effective spatio-temporal semantic trajectory generation for similar pattern group identification. *International Journal of Machine Learning and Cybernetics*, 11(2):287–300, 2020.

[29] Keyan Cao, Qimeng Sun, Haoli Liu, Yefan Liu, Gongjie Meng, and Jingjing Guo. Social space keyword query based on semantic trajectory. *Neurocomputing*, 428:340–351, 2021.

[30] Lisi Chen, Shuo Shang, Christian S Jensen, Bin

Yao, and Panos Kalnis. Parallel semantic trajectory similarity join. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 997–1008. IEEE, 2020.

[31] Jiwei Qin, Liangli Ma, and Qing Liu. Dfthr: A distributed framework for trajectory similarity query based on hbase and redis. *Information*, 10(2):77, 2019.

[32] Zhigang Zhang, Cheqing Jin, Jiali Mao, Xiaolin Yang, and Aoying Zhou. Trajspark: A scalable and efficient in-memory management system for big trajectory data. In *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*, pages 11–26. Springer, 2017.

[33] Xin Ding, Lu Chen, Yunjun Gao, Christian S Jensen, and Hujun Bao. Ultraman: a unified platform for big trajectory data management and analytics. *Proceedings of the VLDB Endowment*, 11(7):787–799, 2018.

[34] Yu Zhang, Youzhong Ma, and Xiaofeng Meng. Efficient spatio-textual similarity join using mapreduce. In *2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, volume 1, pages 52–59. IEEE, 2014.

[35] Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *2015 IEEE 31st international conference on Data Engineering*, pages 1352–1363. IEEE, 2015.

[36] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient in-memory spatial analytics. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1071–1085, 2016.

[37] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Jose Antonio Fernandes de Macedo, Bart Moelans, and Alejandro Vaisman. A model for enriching trajectories with semantic geographical information. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, pages 1–8, 2007.

[38] Luis Otavio Alvares, Vania Bogorny, Bart Kuijpers, Bart Moelans, Jose Antonio Fern, ED Macedo, and Andrey Tietbohl Palma. Towards semantic trajectory knowledge discovery. *Data Mining and Knowledge Discovery*, 12, 2007.

[39] Lisi Chen, Shuo Shang, Chengcheng Yang, and Jing Li. Spatial keyword search: a survey. *Geoinformatica*, 24:85–106, 2020.

[40] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–10, 2008.

[41] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.

[42] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800, 2009.

Chaoquan Cai received a BS degree in Software Engineering and an MS degree in Computer Science from Fuzhou University, Fuzhou, China. He is currently working towards a Computer Science PhD degree at Vanderbilt University. His research interests include statistical machine learning, large-scale semantic data management and analysis.

Dan Lin is currently a Professor of Computer Science and Director of I-Privacy Lab at Vanderbilt University. She received a PhD degree in Computer Science from National University of Singapore in 2007, and was a postdoctoral research associate at Purdue University for two years. Her research interests cover many areas in the fields of information security and database systems.