# Federated Trajectory Search via a Lightweight Similarity Computation Framework

Chen Wu[1], Quanqing Xu[2], Sheng Wang[1], Yuan Sun[3],
Zhifeng Yang[2], Yi Zhang[2], Chuanhui Yang[2], Zhiyong Peng[1]
[1]Wuhan University, [2]OceanBase, [3]University of Melbourne
{chenwu,swangcs,peng}@whu.edu.cn,{xuquanqing.xqq,zhuweng.yzf,yuanzhi.yz,rizhao.ych}@oceanbase.com,
yuan.sun@unimelb.edu.au

## ABSTRACT

This paper presents a federated trajectory search engine called Fetra. Fetra is able to efficiently process top-$k$ search over a data federation composed of numerous mobile devices. In Fetra, we first propose a new similarity measure LCTS to evaluate the spatio-temporal companion time as similarity , which can be widely applied in pandemic management, e.g., COVID-19. LCTS is more effective and efficient than existing trajectory similarity measures and well-suitable for mobile devices with limited resources. Indexing and LCTS-based search in Fetra work as follows: Firstly, each raw trajectory is transformed into a set of visits on POIs (Point of Interests) in mobile devices. Then, we build a federated grid index FGI composed of local indexes in mobile devices called STI and a global index called GTI. Given a query, a pruning strategy over FGI is applied to dynamically prune the candidate mobile devices. Meanwhile, we propose a local optimization to accelerate similarity computations using LCTS based on STI in mobile devices. Extensive experiments on real-world datasets verify the effectiveness and efficiency of Fetra.

**PVLDB Reference Format:**
Chen Wu, Quanqing Xu, Sheng Wang, Yuan Sun, Zhifeng Yang, Yi Zhang, Chuanhui Yang, Zhiyong Peng. Federated Trajectory Search via a Lightweight Similarity Computation Framework. PVLDB, 16(X): XXX-XXX, 2023.
doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**
The source code, data, and/or other artifacts have been made available at https://github.com/whuwuchen/Fetra.

## 1 INTRODUCTION

With the rapid development of mobile internet and locating technologies, human mobility data are collected into trajectory data by locating modules like Bluetooth and GPS in mobile devices. One direct application of the trajectory data is contact tracing [24] which aims to detect whether people have come into direct contact with the patients during COVID-19 pandemic [9]. By exploiting the spatio-temporal feature, trajectory management has been adopted to reduce the spread of the COVID-19 pandemic [10, 17].


Proceedings of the VLDB Endowment, Vol. 16, No. X ISSN 2150-8097.
doi:XX.XX/XXX.XX

However, existing trajectory management systems running on either a single machine [20, 48] or distributed clusters [21, 30, 38, 52] suffer from data isolation [12, 16, 43] since trajectory data are separately owned by multiple parties and it is prohibitive to share these data due to legal regulations or commercial reasons. A promising solution to data isolation is to perform federated search over a data federation consisting of multiple data silos [39]. For example, several mobile network operators can cooperate as a spatial data federation to support query services over historical location data [6]. In a data federation, data silos autonomously manage their own data and do not need to upload data to a third party, where a server acts as a coordinator, interacts with data silos, and resolves external query requests. Federated search has been adopted in spatial data management called federated spatial search which focuses on location point data query including range query/counting [39], distance join and $k$NN join [43]. However, there is still no work on federated search over trajectory data.

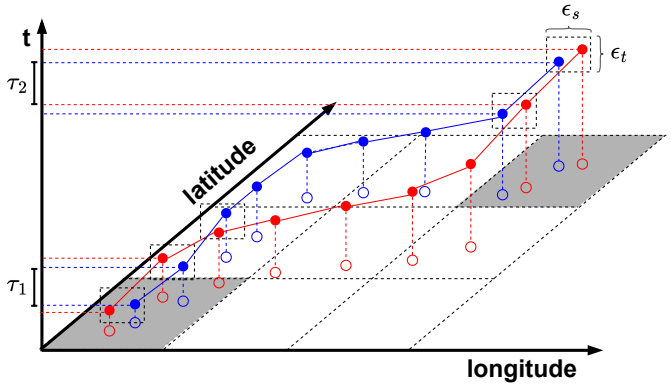


**Figure 1: An example of Spatial temporal longest common subsequence (STLCSS) and Spatio-temporal companion (STC).**

As hardware continues to upgrade, the storage capacity and computing power of mobile devices have fulfilled daily usage. Besides collecting trajectory data, mobile devices have been adopted in on-device works such as *stay-point detection* [34] and *online map-matching* [18]. In some privacy-critical cases, e.g., federated learning [35, 49], as participating clients, mobile devices compute gradient-based updates and communicate only these updates to a central aggregator. These works motivate us to adopt mobile devices into personal trajectory management without uploading data to an external server and processing centrally. In other words, we need a data federation over mobile devices where each member

holds a local trajectory dataset and evaluates personal activity risk in pandemics.

The main challenge is how to measure the risk by trajectory similarity and migrate the processing into mobile devices. Most existing trajectory similarity measures are based on point-wise comparison. An example is *spatial temporal longest common subsequence* (STLCSS) [44] shown in Figure 1. STLCSS treats two points to be matched only if their spatial and temporal distances are less than two thresholds $\epsilon_s$ and $\epsilon_t$, respectively, which requires point-wise distance computation and cannot fit into mobile devices due to limited resources. Another method is *spatio-temporal companion* (STC) [4] that splits the spatial range into grids and accumulates staying time in jointly visited ones. Once the time exceeds a threshold, people will be treated as potential contacts and restricted from spatial activity. In Figure 1, two trajectories share two common cells (gray ones) and the corresponding intersected time intervals i.e., $\tau_1$ and $\tau_2$, will be considered. One drawback is that this method covers the irrelevant trajectory segments on outdoor while infections mainly happen in indoor buildings.

In this paper, we design and implement Fetra, a federated trajectory search engine for data federation over mobile devices. Fetra has the following characteristics: (1) **Flexibility**. Fetra is specially designed for personal mobility data management. Users can collect and store their mobility data in mobile devices. Similarity computation works in mobile devices without uploading data to servers and leaking privacy; (2) **Lightweight**. We design a lightweight federated grid index (FGI) which supports effective pruning in federated search. Meanwhile, we propose the longest companion time similarity (LCTS) which is based on a lightweight trajectory representation and is well-suitable for mobile devices; (3) **Efficiency**. Due to the lightweight index framework and similarity measure, Fetra supports federated trajectory search efficiently. We also implemented a demo system for personal trajectory management in daily usage [51]. In our experiments, Fetra can answer a federated trajectory query in about 1 second over a data federation with 100,000 mobile devices.

In summary, we make the following contributions in this paper:

- We propose a federated trajectory search engine especially for the COVID-19 tracing applications (see Section 4).
- We propose a novel similarity measure based on the companion time, which is efficient to compute for mobile devices (see Section 5).
- We design a lightweight federated index framework which can work for global pruning in a federated manner and boost similarity computation locally (see Sections 6 and 7).
- We build a simulated federated environment and conduct experiments on it over real-world datasets (see Section 8).

# 2 RELATED WORK

## 2.1 Trajectory Similarity Measures

According to trajectory representations, existing trajectory similarity measures can be broadly divided into point-based similarity measures and road network-based similarity measures. Point-based similarity measures treat a trajectory as a set of points, such as Dynamic Time Warping (DTW) [55], Longest Common Subsequence (LCSS) [45], Edit Distance on Real sequence (EDR) [19], Fréchet distance [36] and Hausdorff distance [33]. These similarity measures focus on spatial dimension and can be extended with temporal information. STLCSS [44] extends LCSS by including the temporal condition, where a temporal threshold controls how far in time a point can go to match a given point in another trajectory. Another example is STLC [37] which linearly combines spatial distance and temporal distance between two compared trajectories. These measures suffer from high computation costs [46] and hardly adapt to mobile devices with limited memory and computing abilities.

Road network-based similarity measures can be used in traffic management such as transportation monitoring and planning [27, 50]. In contrast to raw trajectories, the vehicle trajectories will be mapped onto road networks and represented by a set of connected road segments, which is called map matching [32] and many open-source libraries are available for this purpose, such as GraphHopper [3]. Based on the map-matched trajectories, Wang et al. [48] propose LORS to measure their similarity. Further, a simplified LORS, called EBD [47], is employed for trajectory clustering with much lower complexity. Although these road network-based similarity measures are more efficient than point-based ones by avoiding point-wise distance computation, they highly depend on the road network and map matching for trajectory pre-processing, which is infeasible in mobile devices.

## 2.2 Top-$k$ Trajectory Search

Given a query trajectory, top-$k$ trajectory search aims to find the $k$ most similar trajectories from a set of trajectories, which is widely used in trajectory analysis tasks such as trajectory clustering or route planning [11]. Existing studies on top-$k$ trajectory search focus on either centralized solutions, e.g., TrajStore [20] and Torch [48], or distributed solutions including DFT [52], DITA [21], TrajMesa [30], and UITraMan [38]. Centralized solutions put all trajectories in a single machine, which is not suitable for personal mobility data management due to privacy. On the other hand, existing distributed solutions are mainly based on open-source computing frameworks, e.g., Spark [57], which horizontally partition trajectories over clusters. Further, they utilize distributed indexing mechanism to support large-scale trajectory search. However, these systems require high-performance clusters with large memory, which is beyond the scope of this paper.

## 2.3 Federated Search

Federated search works over a data federation [39] where a service provider interacts with multiple data owners and resolves external query requests. As a promising solution to data isolation caused by restricted data sharing, federated search has been adopted in many applications including data science [15, 31], file management [40], and secure query [12–14, 58].

As the volume of spatial data continues to grow and spatial datasets are privately owned by multiple parties, federated search is also adopted in spatial data management called *federated spatial search* [43]. Federated spatial search aims to tackle data isolation and support efficient secure queries over spatial data federation. Existing federated spatial search mainly adopts secure multi-party computation (SMC) [12, 16, 43] to enable secure query processing. It focuses on location point query such as range queries [39], distance join, and $k$NN join [42, 53].

Inspired by federated spatial query, Fetra creates a trajectory data federation on mobile devices. In the federation, each mobile device can be viewed as an autonomous data silo that holds a personal trajectory dataset. To support the contact tracing application, Fetra enables trajectory similarity search over the federation, which is called *federated trajectory search*. Distributed spatial systems improve query processing via data partition and indexing techniques [42, 53, 56]. However, data partitioning is inapplicable in a data federation since the entire data is held by autonomous data silos, which motivates us to develop an efficient indexing technique to improve federated trajectory search.

**Remarks.** Our work differs in three-fold. Firstly, existing similarity measures are infeasible for mobile devices with limited resources. Secondly, existing studies on trajectory search focus on scenarios on the server side rather than personal trajectory management on mobile devices. Thirdly, existing work has not considered federated trajectory search over mobile devices. We are the first study to work on it, which has promising applications.

# 3 PROBLEM DEFINITION

## 3.1 Data Model

**Definition 1** (GPS Point)**.** A GPS point $p = (p.lat, p.lng, p.t)$ contains the spatio-temporal information, which includes latitude $p.lat$, longitude $p.lng$ and timestamp $p.t$.

**Definition 2** (Raw Trajectory)**.** A raw trajectory $T$ consists of a sequence of GPS points $\{p_1, p_2, \cdots, p_n\}$. The time interval between two adjacent GPS points is constant decided by the sampling rate.

**Definition 3** (Stay Point)**.** A stay point $SP = (lat, lng, \tau)$ stands for a geographic location where the user stayed during the time interval $\tau = [t_s, t_e]$.

**Definition 4** (Stay Point Trajectory)**.** A stay point trajectory $\tilde{T} = \{SP_1, SP_2, \cdots, SP_n\}$ is a sequence of stay points in ascending order by time.

The process that extracts stay points from a raw trajectory is called *stay point detection* [22, 29, 60]. In this paper, we adopt a cluster-based algorithm [29] to extract stay points from the raw trajectories in mobile devices.

**Definition 5** (Point of Interests (POI))**.** A POI is a tuple $\rho = (id, lat, lng)$, where $id$ is a unique identifier, $lat$ and $lng$ represent the latitude and longitude of geography location, respectively.

**Definition 6** (Visit)**.** A visit is a tuple $\pi = (\rho, \tau)$ which means the user stayed in POI $\rho$ during the time interval $\tau = [t_s, t_e]$.

**Definition 7** (POI-Matched Trajectory)**.** A POI-matched trajectory $\bar{T} = \{\pi_1, \pi_2, \cdots, \pi_n\}$ is a set of visits in ascending order by time.

Each visit indicates a user stayed on a POI for a while, so we can detect the relevant visit from the stay point by searching nearby POIs within a certain distance where the visit contains the same time interval as the stay point. This process is called *POI visit extraction*. Another way is to directly record the checking-in and out time using location services like *Foursquare* [7]. Figure 2 shows three example trajectories and their interval list of visits, where $Q_0$ is a patient's trajectory, and $T_1$ and $T_2$ are the trajectories of two pedestrians.

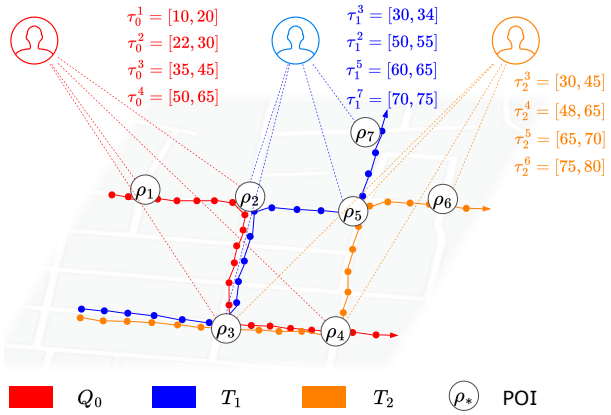


**Figure 2: Example POI-matched trajectories and the timetable of visits on POIs.**

**Definition 8** (Trajectory Data Federation)**.** The trajectory data federation $S = \{s_1, s_2, \cdots, s_n\}$ consists of $n$ mobile devices. Each device $s_i$ holds a trajectory dataset $D_i$.

Similar to previous studies on data federation [12, 13, 39], Fetra also utilizes a server, denoted $s_0$, as the coordinator that accepts external query requests and communicates with these mobile devices. In this paper, we assume the server $s_0$ can access the similarity values and other non-critical data such as *footprint data* (defined in Section 6) from mobile devices.

## 3.2 Query Model

**Definition 9** (Federated Top-$k$ Trajectory Search (FTS))**.** Given a query trajectory dataset $Q$, federation $S$, trajectory similarity measure $M$, and parameter $k$, the search returns the result set $S' \subseteq S$ with $k$ mobile devices such that: $\forall s_i \in S', \forall s_j \in S - S', M(Q, D_i) > M(Q, D_j)$, where $M(Q, D_i)$ measures the similarity of two trajectory datasets $Q$ and $D_i$.

To the best of our knowledge, there is still no work on the trajectory search over data federation. There are mainly two challenges to federated top-$k$ trajectory search: (1) Existing trajectory similarity measures, including point-based or road network-based ones, suffer from heavy point-wise distance computation cost or map-matching cost, which is an onerous burden for mobile devices to compute similarity locally; (2) There is no efficient pruning mechanism during the search, which cannot fulfill the low latency requirement for mobile users in daily usage.

In the following sections, we will mainly explain our solution to federated top-$k$ search using a lightweight trajectory similarity measure. Table 1 summarizes the major notations used throughout this paper.

# 4 FETRA FRAMEWORK

Fetra is a federated trajectory search engine over a data federation composed of mobile devices. Figure 3 presents the framework of Fetra, which consists of three main modules: *Local Processing*, *Federated Indexing*, and *Query Processing*.

**Local Processing.** This module works in mobile devices, takes raw trajectories as input, and performs three main tasks: (1) stay point

**Table 1: Summary of major notations**

| Notation | Description |
|---|---|
| $Q$, $\|\bar{Q}\|$ | query dataset and its visit number |
| $S$, $\|S\|$ | a federation $S$ and its size |
| $s_i$, $D_i$ | a device $s_i$ and its local dataset |
| $s_0$ | the server to support query over $S$ |
| $I_{s_i}$, $I^{\rho}_{s_i}$ | local index in device $s_i$ and the sub index in $\rho$ |
| $P_{D_i}$ | a set of POI visited in $D_i$ |
| $R$ | spatial range |
| $G$, $c_i$ | a global grid over $R$ and a cell numbered i |
| $I_S$ | global index in server $s_0$ |
| $I_S^{c_i}$ | temporal index in cell $c_i$ |
| $\delta_d$ | grid width of $G$ |
| $\theta_r$ | distance threshold in stay point detection |
| $\rho_i$ | a POI numbered i |
| $\Delta$ | communication bandwidth among $S$ |

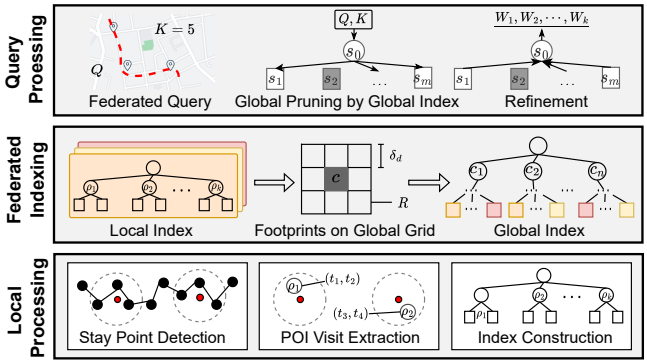


**Figure 3: Our proposed Fetra framework.**

detection, which identifies the locations where a mobile device stayed for a while within a certain time and distance threshold; (2) POI visit extraction, which extracts visits (defined in Section 3.1) around detected stay points; (3) index construction, which builds a local index called STI to accelerate local similarity computation.

**Federated Indexing.** This module builds a global index called GTI in the server by merging local indexes from mobile devices. Specifically, mobile devices convert their visits on POIs to footprints on cells of a global grid $G$, which hides location details for privacy. The global index GTI is built on the grid $G$, whose cells index the temporal ranges of these footprints inside it. Fetra employs a federated index framework called FGI to support efficient similarity query over the federation (detailed in Section 6).

**Query Processing.** With the help of the federated index framework FGI, Fetra supports efficient query over LCTS on the federation in two steps: (1) **global pruning**, which filters out unqualified mobile devices by GTI and sends pruned query data to candidate ones; (2) **refinement**, where each candidate mobile device accepts query data, computes local LCTS similarity with our boost method, and uploads similarity result to the server. The server receives these similarities and terminates the query process when the condition is fulfilled (detailed in Section 7).

# 5 LIGHTWEIGHT SIMILARITY MEASURE

## 5.1 Companion Time As Similarity

Most existing studies define spatio-temporal similarity as the weighted sum of spatial and temporal distances [37], which is not suitable for contact tracing limited by critical spatial and temporal conditions. For example, as shown in Figure 2, $Q_0$ and $T_1$ have an overlapped segment between $\rho_2$ and $\rho_3$, so they are similar in terms of spatial dimension. However, the common segments are not temporally relevant as they are not overlapped in the temporal range. So $Q_0$ and $T_1$ cannot fulfill the contact condition. On the other hand, we observe that $Q_0$ and $T_2$ have a shorter overlapped segment between $\rho_3$ and $\rho_4$, but more contact time in $\rho_3$ and $\rho_4$, which means $T_2$ has more infection risk. Such temporal contact relationship is not captured in existing point-based spatio-temporal similarity measures. Furthermore, the cost of computing spatial distance in point-based similarity is high due to point-wise distance computation.

Inspired by LCSS [46] and LORS [48], we define the Longest Companion Time Similarity (LCTS) to measure the similarity between two POI-matched trajectories.

**Definition 10** (LCTS). Let $\bar{T}_1$ and $\bar{T}_2$ be two POI-matched trajectories of $T_1$ and $T_2$, the Longest Companion Time Similarity is defined as:

$$\bar{M}(\bar{T}_1, \bar{T}_2) = \sum_{\rho \in P_{\bar{T}_1} \cap P_{\bar{T}_2}} |l^{\rho}_{\bar{T}_1} \cap l^{\rho}_{\bar{T}_2}| \tag{1}$$

where $P_{\bar{T}_1}$ is a set of POIs visited in $\bar{T}_1$, $l^{\rho}_{\bar{T}_1}$ is an interval list of visits at $\rho$ in $\bar{T}_1$ and $|l^{\rho}_{\bar{T}_1} \cap l^{\rho}_{\bar{T}_2}|$ represents the overlapped time of two interval lists in POI $\rho$. As the visits in a single POI are temporally disjoint, we can compute $|l^{\rho}_{\bar{T}_1} \cap l^{\rho}_{\bar{T}_2}|$ in a sequential scan with linear complexity. LCTS measures the overlapped time length between $\bar{T}_1$ and $\bar{T}_2$.

Example 1. *Considering trajectories $Q_0$, $T_1$ and $T_2$ in Figure 2, we can observe that $P_{\bar{Q}_0} = \{\rho_1, \rho_2, \rho_3, \rho_4\}$, $P_{\bar{T}_1} = \{\rho_2, \rho_3, \rho_5, \rho_7\}$ and $P_{\bar{T}_2} = \{\rho_3, \rho_4, \rho_5, \rho_6\}$. Hence, $P_{\bar{T}_1} \cap P_{\bar{T}_2} = \{\rho_3, \rho_5\}$. Note that $l^{\rho_3}_{\bar{T}_1} = \{[30, 34]\}$, $l^{\rho_5}_{\bar{T}_1} = \{[60, 65]\}$, $l^{\rho_3}_{\bar{T}_2} = \{[30, 45]\}$ and $l^{\rho_5}_{\bar{T}_2} = \{[65, 70]\}$, we can derive $|l^{\rho_3}_{\bar{T}_1} \cap l^{\rho_3}_{\bar{T}_2}| = 4$ and $|l^{\rho_5}_{\bar{T}_1} \cap l^{\rho_5}_{\bar{T}_2}| = 0$, therefore, LCTS between $\bar{T}_1$ and $\bar{T}_2$ is $\bar{M}(\bar{T}_1, \bar{T}_2) = 4$. In the same way, we have $\bar{M}(\bar{Q}_0, \bar{T}_1) = 0$ and $\bar{M}(\bar{Q}_0, \bar{T}_2) = 25$.*

Property 1. *LCTS is a non-metric similarity measure.*

This property can be validated by observing a counter-example in Figure 2. Note that $\bar{M}(Q_0, T_1) = 0$, $\bar{M}(Q_0, T_2) = 25$ and $\bar{M}(T_1, T_2) = 4$, then $|\bar{M}(Q_0, T_1) + \bar{M}(T_1, T_2)| < \bar{M}(Q_0, T_2)$. Hence LCTS does not obey the triangular inequality, and is non-metric. LCTS does not require extra parameters and avoids complex spatial distance computation between points.

## 5.2 Interval Set Intersection

When computing LCTS between two POI-matched trajectory datasets, we only need to sum up the overlapped time intervals of visits on the same POIs, which can be viewed as a set intersection problem [26] except that LCTS considers interval intersection lengths. We call this process as *interval set intersection*. Since a

trajectory or trajectory dataset can both be viewed as a list of visits, we can also compute LCTS between two trajectory datasets in the same way used in trajectories.

One baseline method for federated top-$k$ search using LCTS is that the server $s_0$ directly broadcasts the query $Q$ to all mobile devices and accepts similarities from them. We call this method as direct. The time cost is as follows:

$$\begin{aligned} C_{\text{direct}} &= C_{Comm} + C_{Comp} \\ &= O(|S| * \frac{|Q|}{\Delta}) + \max_{s_i \in S} O(\sum_{\rho \in P_Q \cap P_{D_i}} (|l_Q^\rho| + |l_{D_i}^\rho|) \\ &= O(|S| * \frac{|\bar{Q}|}{\Delta}) + \max_{s_i \in S} O(|\bar{Q}| + |\bar{D}_i|) \end{aligned} \tag{2}$$

where $\Delta$ is the communication bandwidth between the server and mobile devices. The cost can be divided into communication cost $C_{Comm}$ and local computation cost $C_{Comp}$. As all devices compute LCTS independently, the local computation cost greatly depends on the slowest one.

# 6 LIGHTWEIGHT FEDERATED INDEX FRAMEWORK

To support efficient search using LCTS, we propose a lightweight *federated grid index* framework (short as FGI) that consists of local indexes called *segmented timeline index* (short as STI) in mobile devices, and a global index called *grid temporal index* (short as GTI) in the server $s_0$. Figure 4 presents an example of FGI, which employs two levels of indexes: (1) GTI which helps the server find relevant devices that may contain trajectories similar to the query trajectory dataset $Q$; (2) STI which enables each mobile device to efficiently compute LCTS with $Q$.

## 6.1 Local Index on Devices

**Pre-processing.** When there is a POI dataset in mobile devices, we can build STI over visits in local POI-matched trajectories. In this case, we transform the raw trajectories into POI-matched trajectories. In this paper, we propose a two-phase trajectory mapping method shown in Figure 4. Firstly, we utilize a cluster-based algorithm [29] to detect stay points in raw trajectories. To detect stay points more fine-grained, we set smaller time and distance thresholds of 5 minutes and 50m than the default values of 30 minutes and 200m in [29] respectively. Then we extract the POIs around stay points within a distance threshold $\theta_r$ (default 50m same as the distance threshold in stay point detection) as visited.

**Local Indexing.** After pre-processing, we group these visits by their POIs. Then we build a temporal index, like timeline index [25], for each group. As all visits are temporally disjoint, the temporal index can be simplified as a sorted integer list. In this way, a local index in device $s_i$ can be represented as a set of sub-indexes: $I_{s_i} = \{I_{s_i}^\rho | \rho \in P_{D_i}\}$, where $I_{s_i}^\rho$ is the temporal index for the group in $\rho$ and $P_{D_i}$ is the POI set visited in $s_i$. As shown in Figure 4, there are six visits in $s_1$ grouped by three POIs $P_{D_1} = \{\rho_1, \rho_2, \rho_6\}$. So $s_1$ holds a local index $I_{s_1} = \{I_{s_1}^{\rho_1}, I_{s_1}^{\rho_2}, I_{s_1}^{\rho_6}\}$. When a device computes LCTS with $Q$ locally, it retrieves related visits by binary search using STI and accumulates overlapped time lengths.

**Serialization.** Each mobile device can serialize its STI into a string and upload it to the server. We store each sub-index as a sorted integer list. Then we concatenate these lists and convert POI ids to corresponding cell ids (introduced in Section 6.2). Finally, we can compress these integer lists using *delta encoding* [28]. For example, Figure 5 presents the serialization of above mentioned index $I_1$ in $s_1$. The sub-index $I_{s_1}^{\rho_6}$ contains two intervals $[46, 51]$ and $[52, 57]$, we store it as $\{46, 51, 52, 57\}$ and further compress it as $\{46, 5, 1, 5\}$ with delta encoding. As $\rho_6$ is located in cell $c_4$, we replace the POI id 6 with the cell id 4. We adopt the same operation in other sub-indexes $I_{s_1}^{\rho_1}$ and $I_{s_1}^{\rho_2}$, concatenate these strings, and get the serialized index.

## 6.2 Global Index on Server

As a prerequisite for federated search, the server constructs GTI to find candidate mobile devices with the query dataset $Q$.

**Definition 11** (Global Grid)**.** A global grid $G = \{c_1, c_2, \cdots, c_{m*n}\}$ partitions the spatial range $R$ into $m * n$ cells, where $R = \{lat_{min}, lat_{max}, lng_{min}, lng_{max}\}$ is a bounding rectangle of a city and each cell is a square with width $\delta_d$.

**Spatial Partitioning.** It is infeasible for a mobile device to upload STI to the server directly since it contains private personal activity information. In the federation, the server partitions the spatial range into multiple cells, each covering several POIs. The serialization of STI is based on the grid by converting POIs to corresponding cells. Considering the global grid in Figure 4, six POIs are grouped into four cells. As the range $R$ is fixed, the partition is only determined by the width $\delta_d$. The cell width $\delta_d$ affects the granularity of the grid and pruning effectiveness. On the one hand, a smaller $\delta_d$ means finer partitioning and better pruning for federated search. On the other hand, as $\delta_d$ decreases, it exposes more accurate personal locations, which causes a higher risk for privacy.

**Definition 12** (Footprint)**.** A footprint $\sigma = (c, i, \tau)$ records the time interval $\tau$ of device $s_i$ staying on cell $c$.

**Global Indexing.** The global index GTI maintains a two-layer tree structure as shown in Figure 4. In the first layer, it keeps a global grid in the root node, which partitions the spatial range into unified cells, called spatial partitioning. In the second layer, it utilizes a temporal index to index the time intervals of footprints in each cell, called *temporal indexing*. With the independence between spatial partitioning and temporal indexing, different temporal indexes, like interval tree or timeline index [25], can be employed in GTI. Thus, a global index can be represented as a set of temporal indexes over the spatial grid: $I_S = \{I_S^c | c \in G\}$, where $G$ is the spatial grid and $I_S^c$ is the temporal index over cell $c$. Algorithm 1 illustrates the construction of GTI. To unify the global grid in the federation, the server broadcasts parameters $\delta_d$ and $R$ to all devices (Line 1). Then each mobile device converts its visits to a footprint set and uploads it to the server (Line 2-4). Meanwhile, the server partitions the spatial range into cells (Line 7) and creates a temporal index for each cell (Line 8). Finally, the server traverses the footprint set, groups them by cell, and updates interval trees in each cell (Line 10-11).

**Parallelism.** Note that the temporal indexes in GTI are spatially independent. When the devices upload their footprints to the server,

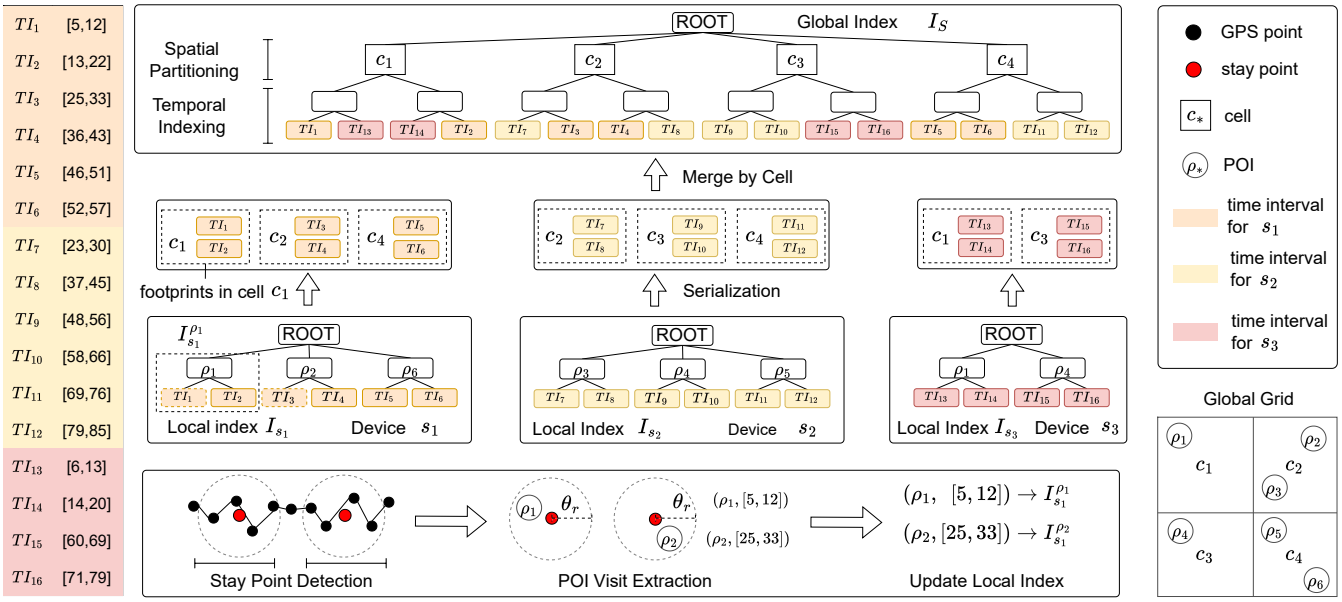


**Figure 4: Our federated grid index (FGI) framework.**

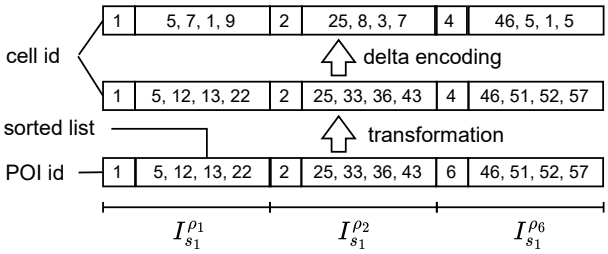


**Figure 5: Index Serialization of STI.**

we can update these temporal indexes in parallel. Moreover, we can also retrieve candidate devices using GTI in parallel.

## 6.3 Period Temporal Index

Note that GTI adopts a two-layer structure where temporal indexing is independent of spatial partitioning, we further design a period temporal index for efficient interval retrieval shown in Figure 6.

**Construction.** As the duration of human activity is rarely more than one day, we split the temporal range into several buckets with the length of 24 hours. Each bucket is further divided into $L$ levels, where the $i$-th level contains $2^{i-1}$ partitions. Each raw interval is aligned by linear scaling:

$$[t_s, t_e] \rightarrow [\lfloor \frac{t_s}{|B|} * (2^{L-1} - 1) \rfloor, \lceil \frac{t_e}{|B|} * (2^{L-1} - 1) \rceil] \quad (3)$$

where $|B|$ is the bucket length, i.e., 24h. To index time intervals, we first determine the corresponding bucket and align each interval using Equation 3. Then we insert the aligned interval to the bucket as presented in Algorithm 2. The resolution, that is, the partition length in the lowest level, is $\delta_t = \frac{|B|}{2^{L-1}}$.

Furthermore, the quantization error $\epsilon_e$ is defined as the length difference between the aligned interval and the raw interval, which can be bounded by the following equation.

$$\epsilon_e \leq 2 * \delta_t = \frac{|B|}{2^{L-2}} \quad (4)$$

When the interval length is greater than the bucket length, we split it into multiple segments and add each segment to the corresponding buckets.

**Comparison-free Search.** Given a query interval, we check the overlapped temporal partitions for each level and add all the candidate devices to the result. As we do not store the raw intervals in the period temporal index, there is no comparison during the search. Since we need to access all levels, the query cost, in the worst case, is $O(L)$. Smaller $L$ means lower query cost while introducing a worse pruning effect as the quantization error. In real usage, we set $L = 10$ and the quantization error is about 5.5 minutes (337 seconds) (using Equation 4).

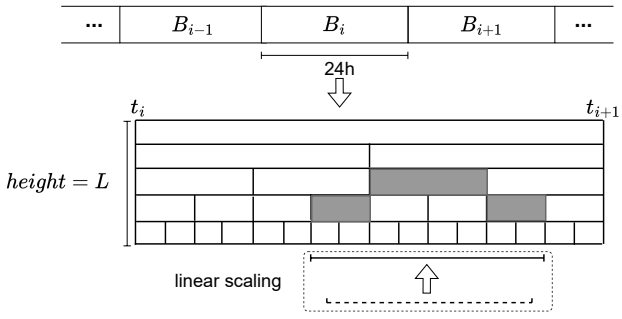


**Figure 6: Period Temporal Index (PTI).**

**Algorithm 1:** GlobalIndexConstruction($\delta_d$, $R$, $S$)

**Input:** $\delta_d$: cell width, $R$: range, $S$: federation
**Output:** $I_S$: GTI

Broadcast $\delta_d$ and $R$ to devices in $S$;
**foreach** *device* $s_i \in S$ **do**
  $F_i \leftarrow LocalConvert(\delta_d, R, I_i)$; ▷ convert visits to footprints
  $s_i$ uploads footprint set $F_i$ to server $s_0$ ;
$m \leftarrow \lceil \frac{R.lat_{max} - R.lat_{min}}{\delta_d} \rceil$ ;
$n \leftarrow \lceil \frac{R.lng_{max} - R.lng_{min}}{\delta_d} \rceil$ ;
Partition $R$ into $m * n$ cells with width $\delta_d$;
Create a temporal index $I_S^c$ for each cell $c$;
$F \leftarrow \bigcup_{i=1}^{|S|} F_i$ ;
**foreach** *footprint* $\sigma \in F$ **do**
  Add $\sigma$ to $I_S^{\sigma.c}$; ▷ merge by cell
**return** $I_S = \{I_S^{c_1}, I_S^{c_2}, \cdots, I_S^{c_{m*n}}\}$;

---

**Function** LocalConvert($\delta_d$, $R$, $I_{s_i}$)**:**
  **Input:** $\delta_d$: cell width, $R$: range, $I_{s_i}$: local index
  **Output:** $F_i$: footprint set
  $F_i \leftarrow \emptyset$, $rownum \leftarrow \lceil \frac{R.lng_{max} - R.lng_{min}}{\delta_d} \rceil$;
  **foreach** $I_{s_i}^{\rho_j} \in I_{s_i}$ **do**
    $y \leftarrow \lceil \frac{\rho_j.lat - R.lat_{min}}{\delta_d} \rceil$;
    $x \leftarrow \lceil \frac{\rho_j.lng - R.lng_{min}}{\delta_d} \rceil$;
    $c \leftarrow y * rownum + x$;
    **foreach** $\tau \in I_{s_i}^{\rho_j}$ **do**
      $\sigma \leftarrow (c, i, \tau)$ ;
      $F_i \leftarrow \{\sigma\} \bigcup F_i$;
  **return** $F_i$;

**Algorithm 2:** AddFootprint($B_i$, $\sigma$)

**Input:** $\sigma$: footprint, $B_i$: $i$-th bucket
**Output:** updated $B_i$ after insertion

$L \leftarrow$ height of $B_i$;
$[t_s, t_e] \leftarrow$ time interval of $\sigma$;
$t_s \leftarrow \lfloor \frac{t_s}{|B|} * (2^{L-1} - 1) \rfloor$;
$t_e \leftarrow \lceil \frac{t_e}{|B|} * (2^{L-1} - 1) \rceil$;
**while** $L \geq 0$ *and* $t_s \leq t_e$ **do**
  **if** *last bit of* $t_s$ *is 1* **then**
    add device id of $\sigma$ to $t_s$-th partition at level L;
    $t_s \leftarrow t_s + 1$;
  **if** *last bit of* $t_e$ *is 0* **then**
    add device id of $\sigma$ to $t_e$-th partition at level L;
    $t_e \leftarrow t_e - 1$;
  $t_s \leftarrow t_s/2$, $t_e \leftarrow t_e/2$;
  $L \leftarrow L - 1$;
**return** $B_i$;

# 7 FEDERATED TOP-K SEARCH

It is noteworthy that the baseline method in Section 5.2 requires high communication cost and local computation cost. Hence, effective pruning and similarity computation methods are essential for federated search. We present our method for federated top-$k$ search with FGI.

## 7.1 Algorithm Overview

Similar to the dynamic pruning strategy in [48], our algorithm is composed of *filtering* (Line 4-11) and *refinement* (Line 16-22) based on GTI $I_S$, as shown in Algorithm 3.

To filter out unqualified devices, the GTI index $I_S$ plays an important role in LCTS. The principle is that candidate devices must have commonly visited POIs with the visits in query dataset $Q$. So the filtering phase will traverse visits in $Q$, identify the cell of a visit, and search for the footprints overlapped with the target visit in the same cell (Line 5-10). The server maintains a similarity upper bound list $UB$ for candidates (Line 9). Termination occurs when newly accepted similarity is greater than the upper bound $UB_{max}$ for the remaining devices (Line 18), and the server terminates the query by sending termination signals to these devices and returns the top-$k$ result set.

## 7.2 Pruning for LCTS

Recall from Property 1 that LCTS is non-metric, for which a tree-structure index such as an R-tree cannot be used for pruning. To this end, the main idea of our pruning is to terminate the query process in advance without waiting for the responses of all candidates. We maintain an upper bound list $UB$ for the remaining candidates and update it with accepted similarities. The query can be terminated when the newly accepted similarity is greater than the current upper bound $UB_{max}$. We refer to this process as *early termination*.

**Upper Bounding Similarity.** For LCTS, after accessing temporal indexes in $I_S$ for all visits in $Q$, we get potential visit lists $Q^- = \{Q_1^-, Q_2^-, \cdots, Q_{|S|}^-\}$ for all devices. A device $s_i$ becomes candidate only when $Q_i^-$ is not empty, which means it could have relevant visits with $Q$. The upper similarity bound is the sum of overlapped time interval length (Line 9), where $|\pi.\tau \cap \sigma.\tau|$ is the overlapped time length.

**Query Data Pruning.** Besides filtering out unqualified mobile devices, we also accelerate the query process by dropping unrelated visits in $Q$. The server sends pruned query data $Q_i^- \subseteq Q$ to device $s_i$. We have the following lemma.

Lemma 1. $\forall s_i \in$ *candidate devices*, $\bar{M}(Q, D_i) = \bar{M}(Q_i^-, D_i)$.

Proof. This lemma can be derived from Equation 1. Any visit in $Q$ not located in the same cells with $D_i$, denoted as $Q - Q_i^-$, cannot affect the similarity as two visits have companion time only when they stay in the same POI. □

Based on the above Lemma, device $s_i$ can compute the similarity using pruned query dataset $Q_i^-$. This can reduce the communication cost and computation workload for mobile devices.

**Algorithm 3:** FederatedTopKSearch($Q$, $k$, $S$)

**Input:** $Q$: query dataset, $k$: parameter, $S$: federation
**Output:** $W$: Top-k result set

$can \leftarrow \emptyset, W \leftarrow \emptyset$ ;
Initialize a query data list $Q^- = \{Q_1^-, Q_2^-, \cdots, Q_{|S|}^-\}$ ;
Initialize upper bound list $UB = \{UB_1, UB_2, \cdots, UB_{|S|}\}$;
**foreach** $\pi \in Q$ **do**
    Identify the cell $c$ of $\pi.\rho$; ▷ `using lines 17-19 in Algorithm 1`
    $F \leftarrow$ temporally overlapped footprints in $I_S^c$; ;
    **foreach** $\sigma \in F$ **do**
        $i \leftarrow$ device id of $\sigma$;
        $UB_i \leftarrow UB_i + |\pi.\tau \cap \sigma.\tau|$;
        $Q_i^- \leftarrow Q_i^- \bigcup \{\pi\}$ ;
$can \leftarrow \bigcup_{|Q_i^-| \neq 0} \{s_i\}$;
Send query data $Q_i^-$ to $s_i \in can$ ;
**foreach** *device* $s_i \in can$ **do**
    $\bar{M}(Q_i^-, D_i) \leftarrow LCTSComputation(Q_i^-, D_i)$ ;
    Send $\bar{M}(Q_i^-, D_i)$ to the server $s_0$;
**while** $can \neq \emptyset$ **do**
    Accept $\bar{M}(Q_i^-, D_i)$ from $s_i$;
    **if** $\bar{M}(Q_i^-, D_i) \geq UB_{max}$ **then**
        Send termination signals to remaining devices ;
        break;
    $UB_i \leftarrow \bar{M}(Q_i^-, D_i), W \leftarrow W \bigcup \{s_i\}$;
    Remove $s_i$ from $can$;
Sort $W$ by similarity;
**return** $W_{1-k}$;

---

**Function** `LCTSComputation`$(Q, D)$:

**Input:** $Q$: query dataset, $D$: local dataset
**Output:** $\bar{M}(Q, D)$: LCTS similarity
$\bar{M}(Q, D) \leftarrow 0$;
Group visits in $Q$ by POI;
**for** *each group* $Q_{\rho_i}$ **do**
    calculate $\bar{M}(Q_{\rho_i}, D)$ using $I_s^{\rho_i}$ by binary search;
    $\bar{M}(Q, D) \leftarrow \bar{M}(Q, D) + \bar{M}(Q_{\rho_i}, D)$;
**return** $\bar{M}(Q, D)$;

## 7.3 Local Boost for LCTS

A direct way to compute LCTS between $Q$ and $D_i$ in device $s_i$ is to traverse these two lists order by time. Note that there exist spatial-temporal companions only when two visits happened in the same POI. Inspired by this, we boost local similarity computation on LCTS using STI.

Figure 7 presents the process of local computation, where $Q^-$ is the pruned query dataset by GTI. We first divide $Q^-$ into multiple groups based on POIs. Then, we calculate overlapped time length for each group by its corresponding sub-index. In the end, the results in all groups are accumulated as the final result. As all groups are independent, the process can be implemented in parallel.

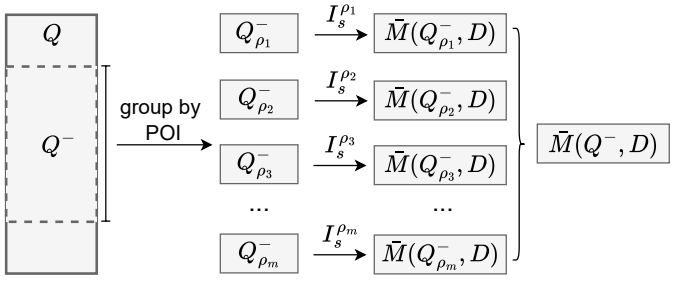


**Figure 7: Local LCTS Computation in Parallel.**

## 7.4 Search with $d$-LCTS

A POI dataset is critical for *POI visit extraction* in Section 3.1. POI dataset could be unavailable, e.g., a user travels across regions. In this case, we cannot calculate LCTS on POI-matched trajectories. We transform raw trajectories into stay point trajectories and calculate LCTS on stay point trajectories. To this end, we extend LCTS to stay point trajectories called $d$-LCTS as below:

$$\bar{M}'(\tilde{T}_1, \tilde{T}_2) = \sum_{SP_1 \in \tilde{T}_1 \wedge SP_2 \in \tilde{T}_2 \wedge d(SP_1, SP_2) \leq d} |SP_1 \cap SP_2| \quad (5)$$

where $|SP_1 \cap SP_2|$ is the overlapped time of two stay points, $d$ is the distance threshold and kept the same with the spatial parameter in stay point detection i.e., 50m.

**Spatio-temporal Pruning.** Similar to visits in POI-matched trajectories, we can also transform a stay point into a footprint in the same way. We employ GTI to filter out unqualified devices and prune unrelated stay points in the query dataset like Algorithm 3. All cells whose distances with the target stay point is less than $d$ will be checked. Then, the temporal indexes in these cells will be used for temporally filtering.

## 7.5 Query Performance

The time cost of federated top-$k$ search with FGI can be divided into three parts: filtering, communication, and local computation. The time cost complexity is analysed as follows:

$$\begin{aligned} C_{fts-fgi} &= C_{filtering} + C_{Comm} + C_{Comp} \\ &= O(L * |\tilde{Q}|) + O(\frac{\sum_{s_i \in can} |Q_i^-|}{\Delta}) + \max_{s_i \in can} (|Q_i^-| + |D_i|). \end{aligned} \quad (6)$$

where $can$ represents candidate devices. Compared with the baseline method in Section 5.2, Algorithm 3 improves query performance in two aspects: firstly, it filters out unqualified devices and reduces the communication cost; secondly, it prunes query data for each candidate devices and reduces the local computation cost.

# 8 EXPERIMENTS

## 8.1 Experimental Setup

**Datasets.** It is hard to access trajectory data during a pandemic and we adopt three trajectory datasets close to the real scenarios. We use the Beijing GPS trajectory dataset of the Geolife project [59, 60]

**Table 2: Statistics of trajectory datasets.**

| Characteristics | Geolife | NYC | Tokyo |
|---|---|---|---|
| #trajectories | 58,725 | 93,971 | 199,245 |
| #POIs | 625,910 | 400 | 385 |
| #visits | 385,639 | 228,511 | 575,996 |
| trajectory length $\|\bar{T}\|$ | 6.56 | 2.43 | 2.89 |
| sampling rate (s) | 5 | - | - |
| Space (MB) of $\mathcal{D}$ | 2,190 | 29.25 | 73.07 |

and the New York City and Tokyo Check-ins data of FourSquare [1]. These datasets contain real long-term mobility data of multiple users and fulfill our requirements. Since Tokyo and NYC datasets just contain the check-in time, we add the check-out time by assigning the stay time which ranges from 10 to 60 minutes randomly. Table 2 describes the statistics of the trajectory datasets. The sampling rate means the average time $t$ between two neighbor points in a trajectory (sampling a point every $t$ seconds). NYC and Tokyo are check-in data so they do not contain the sampling rate and have smaller space sizes. These check-in data denoted by the same user in one day are aggregated into a single POI-matched trajectory.

**Implementation.** Considering the difficulty of conducting such an experiment in real life, we implement a simulation program that contains a server and multiple clients in Java. We use FastPFor [2] to compress the indexes in Section 6. All raw trajectories are transformed into POI-matched trajectories before indexing. All experiments ran on a server with an Intel Xeon Platinum 8269CY CPU(8 cores) and 64GB RAM running CentOS 7.6.

**Experiment Goals.** The effectiveness and efficiency of Fetra are evaluated in the next three subsections from the following aspects:

(1) *Index Performance* - whether the compression techniques can significantly reduce the transmission and storage cost of STI in mobile devices.
(2) *Efficiency of Search* - whether FGI and our search algorithm answer federated top-$k$ search using LCTS efficiently.
(3) *Effectiveness of LCTS* - whether LCTS similarity is robust to the sampling rate, GPS error, and point shifting.

**Baseline Methods.** There are five methods in our experiments:

(1) **direct** - The baseline method in Section 5.2.
(2) **copt-PTI** - It optimizes federated search with local boost in Section 7.3.
(3) **popt-PTI** - It optimizes federated search with global pruning in Section 7.2 using FGI based on the period temporal index.
(4) **copt+popt-PTI** - It optimizes federated search with local boost and global pruning using FGI based on the period temporal index.
(5) **copt+popt-IT** - It optimizes federated search with local boost and global pruning using FGI based on the interval tree index.

We also compare performance of federated trajectory search using the existing two spatio-temporal similarity measures STLCSS [45] and STLC [41]. For STLCSS, we build the hierarchical index in [45] for each mobile device as the local index and merge them as the global index in the server. We also utilize *dynamic programming* in [45] to compute STLCSS locally. For STLC, we build the grid index in [37] for each mobile device as the local index and merge them as the global index in the server. As there is no optimization for STLC computation, we compute STLC by two loops locally. For these two similarity measures, we filter out unqualified mobile devices using the global index and just send relevant raw trajectories to candidates.

**Table 3: Parameters (Default value is highlighted).**

| Parameter | Value |
|---|---|
| Federation Size $\|S\|$ | $1\times\{10^1, 10^2, 10^3, 10^4, \mathbf{10^5}\}$ |
| Local Dataset Size $\|D\|$ | {100, 200, **300**, 400, 500} |
| Query Dataset Size $\|Q\|$ | {0.001, 0.002, **0.003**, 0.004, 0.005}×$\|\boldsymbol{Q}\|$ |
| $k$ | {10, 20, **30**, 40, 50} |
| Grid Width $\delta_d$ | $\{1, 3, \mathbf{5}, 7, 9\} \times 0.001°$ |

**Parameters.** Table 3 shows the parameters used in our methods. When a parameter varies, other parameters are set to default values (highlighted in bold). We select 0.001° as the unit of grid width, where 0.001° is roughly 111 meters and $\delta_d = 9$ means 0.009°, i.e., about 1000m. The query dataset size $|Q|$ is set as the ratio of the trajectory dataset $\boldsymbol{Q}$. For each parameter condition, we execute the experiments 10 times and report the average values as the results.

## 8.2 Index Performance

In a federation with $|S|$ mobile devices, we assign $|D|$ trajectories randomly from the dataset to them and create GTI in the server using Algorithm 1.

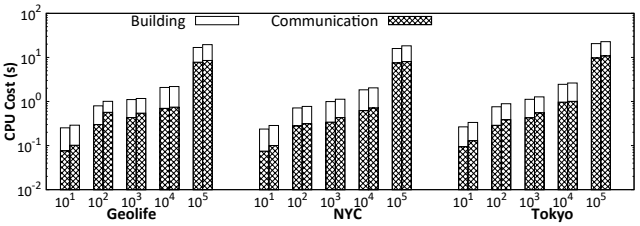

**Figure 8: GTI Construction with Increasing $|S|$.**

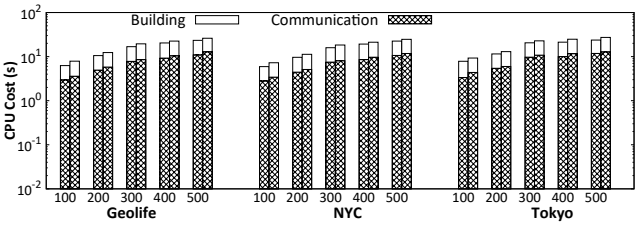

**Figure 9: GTI Construction with Increasing $|D|$.**

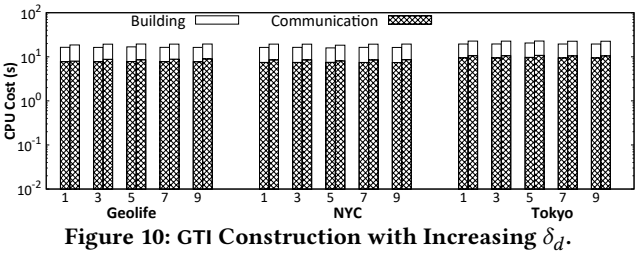

**Figure 10: GTI Construction with Increasing $\delta_d$.**

**Construction.** Figures 8, 9, 10 show the time spent on GTI construction over three datasets, including the communication time

when mobile devices upload STI to the server and the building time when the server constructs GTI. Each mobile device compresses STI using the serialization in Section 6 to reduce communication cost in GTI construction (left ones). We have the following observations from the results:

(1) With the increase of $|S|$ and $|D|$, it takes more communication and building time to construct GTI, but not for $\delta_d$. This is because increased $S$ and $D$ means more devices upload larger STI to the server, which enlarges the communication time. Moreover, it takes more time for the server to process more data thus the building time increases. As for $\delta_d$, it has little effect on the size of GTI thus the construction time does not change with $\delta_d$ apparently.
(2) The compression of STI reduces the communication time by $10-20\%$ with negligible increased building time. This is because the compression reduces serialization size of STI.

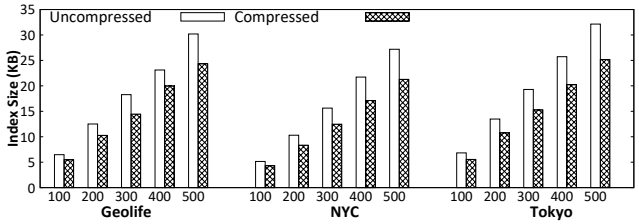

**Figure 11: Storage size of STI with increasing $|D|$.**

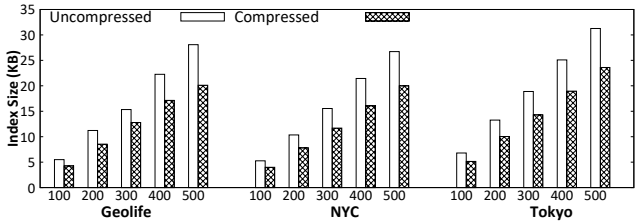

**Figure 12: Serialization size of STI with increasing $|D|$.**

**Compressibility.** Each mobile device constructs its STI and serializes it into a local file. We report the file size as the storage size and the serialized string length as the serialization size, respectively. We employ the serialization method in Section 6 with a local dataset size of $|D| = 100, 200, 300, 400, 500$. Figures 11 and 12 compare the results between STI and compressed STI on three datasets. We observe that:

(1) With the increase of $|D|$, the storage size and serialization size both increase proportionally. This is because there are more visits in each mobile device to be indexed in STI.
(2) Our compression method reduces STI by $10-20\%$, where smaller STI means lower storage cost for mobile devices. Meanwhile, compression also reduces construction cost for GTI because of lower communication cost presented in Figures 8, 9 and 10.
(3) In our experiments, the storage and serialization size of STI is no more than 35KB, which is lightweight for mobile devices. Moreover, the lightweight STI can be loaded in memory for local similarity computation during the federated search.

## 8.3 Efficiency Evaluation of Search Algorithms

In a federation with $|S|$ mobile devices, we randomly assign $|D|$ trajectories to each device and execute federated top-$k$ search 10 times on three datasets. We also randomly sample $|Q|$ trajectories from these datasets as the query dataset and report the average running time in Figures 13, 14, and 15. The other parameters keep default values when varying a parameter condition in each experiment. We compare five methods described in Section 8.1.

**Effect of Federation Size.** The first columns in Figures 13, 14, 15 show the performance when varying the federation size. We have the following observations:

(1) With the increase of $|S|$, it degrades the performance of all methods as a larger $|S|$ increases the communication workload on the server.
(2) copt+popt-PTI significantly outperforms the other methods. For example, when $|S| = 100,000$ on Geolife, direct takes 17.109s, copt-PTI takes 1.73s, popt-PTI takes 4.509s while copt+popt-PTI takes 0.566s. The reasons are two-fold: *i)* copt+popt-PTI employs an efficient pruning strategy in Section 7.2 to reduce unnecessary computations while direct and copt-PTI do not; *ii)* Compared with direct and popt-PTI, copt+popt-PTI utilizes the local boost method in Section 7.3 to accelerate the local similarity computations in mobile devices.
(3) copt+popt-PTI is less sensitive to $|S|$ compared with the other methods as copt+popt-PTI utilizes the pruning strategy based on PTI and the local boost method at the same time, which greatly improves the search efficiency.
(4) Our PTI improves the performance of FGI about 2 times compared with interval tree. For example, when $|S| = 100,000$ on Geolife, copt+popt-PTI takes 0.566s while copt+popt-IT takes 1.203s. This is because PTI employs a comparison-free search style and reduces the filtering time in the server.

Notice that (2), (3) and (4) also hold for the same reasons when varying the other four parameters.

**Effect of Local Dataset Size.** The second columns in Figures 13, 14, 15 show the performance when varying the local dataset size in mobile devices. We have the following observations:

(1) With the increase of $|D|$, all methods take more time for the larger computation workload in local devices.
(2) The local boost method usually outperforms the pruning strategy in federated top-$k$ search. For example, when $|D| = 300$ on Tokyo, copt-PTI and popt-PTI take 2.771s, 4.345s respectively. The reason is that all the mobile devices calculate their similarities in parallel thus the query time greatly depends on the slowest one. The local boost method can reduce the computation cost of LCTS.

**Effect of Query Dataset Size.** We vary the query dataset size $|Q|$ and have the following observation from the third columns in Figures 13, 14, 15. With the increase of $|Q|$, the query performance declines for all methods since a larger $|Q|$ increases the filtering workload for the server and the computation cost for mobile devices.
**Effect of *k*.** We vary the parameter $k$ and observe the query performance shown in the fourth columns in Figures 13, 14, 15. All methods are not sensitive to the increase of $k$, which is slightly different from the traditional top-$k$ search on a single machine or distributed clusters. The reason is that the server needs to wait for candidate mobile devices computing LCTS locally. Meanwhile, our

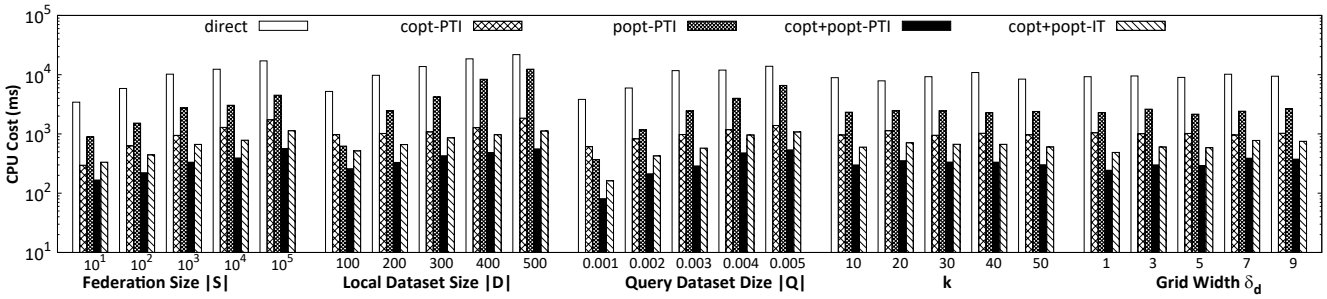


**Figure 13: Comparing Query Performance on Geolife.**

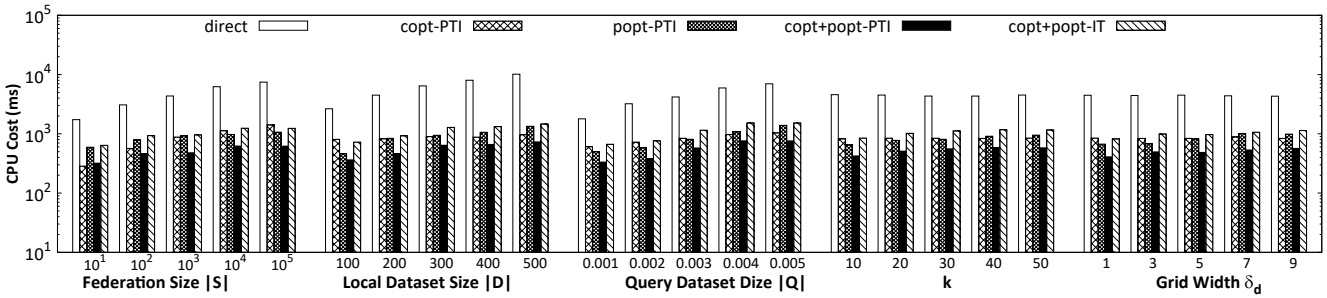


**Figure 14: Comparing Query Performance on NYC.**

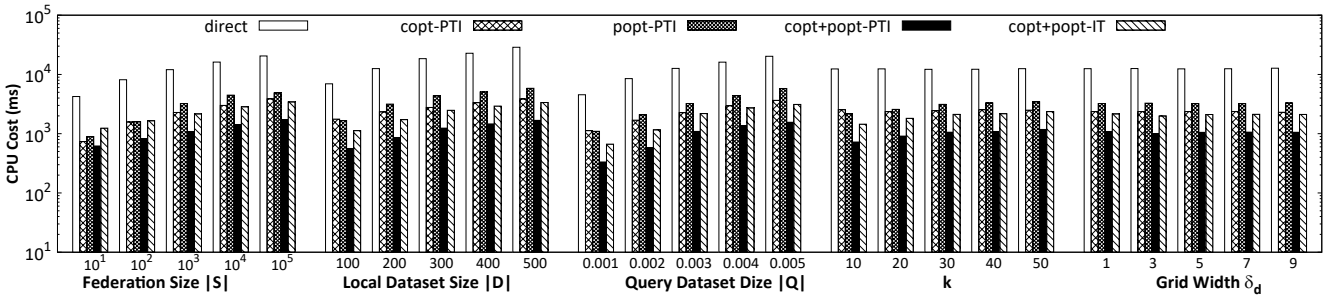


**Figure 15: Comparing Query Performance on Tokyo.**

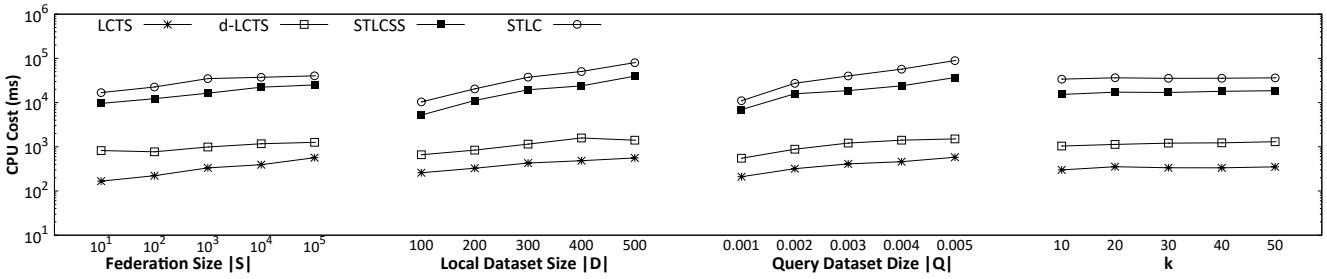


**Figure 16: Federated Search Using LCTS, *d*-LCTS, STLCSS and STLC on Geolife.**

local boost method and pruning strategy greatly speed up the federated top-$k$ search. For example, when $k = 50$ on Geolife, the five methods take 8.427s, 0.967s, 2.395s, 0.302s, and 0.612s, respectively.

**Effect of Grid Width.** We also vary the grid width $\delta_d$ and the result is shown in the fifth columns in Figures 13, 14, and 15. When $\delta_d$ increases from 0.001° to 0.009°, the query time of popt-PTI and copt+popt-PTI on Geolife increase from 2.283s, 0.244s to 2.643s, and 0.376s, respectively. The reason is that smaller $\delta_d$ partitions the spatial range $R$ more precisely and makes the spatial pruning more accurate, which accelerates the query process. However, it does not work notably on NYC and Tokyo because there are fewer POIs in these datasets thus the activity ranges of mobile devices are closer, which impairs the spatial pruning. Note that this has no impact on temporal pruning so our pruning strategy still works.

## 8.4 Efficiency Evaluation of LCTS

We also evaluate the efficiency of LCTS by conducting federated search experiments on Geolife using LCTS, *d*-LCTS, STLCSS and STLC. We utilize our filtering strategy for global pruning. As the indexes for STLCSS and STLC are not based on the spatial grid index, we

**Table 4: The robustness evaluation to sampling rate, GPS error, and time shifting.**

| | | Sampling Rate | | | | GPS Error | | | | Point Shifting | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | LCTS | $d$-LCTS | STLCSS | STLC | LCTS | $d$-LCTS | STLCSS | STLC | LCTS | $d$-LCTS | STLCSS | STLC |
| Geolife | P@5 | 0.971 | 0.922 | 0.905 | 0.903 | 0.975 | 0.935 | 0.925 | 0.918 | 0.955 | 0.931 | 0.915 | 0.909 |
| | P@10 | 0.957 | 0.913 | 0.897 | 0.890 | 0.959 | 0.928 | 0.917 | 0.912 | 0.947 | 0.924 | 0.907 | 0.905 |
| | P@15 | 0.948 | 0.908 | 0.889 | 0.876 | 0.953 | 0.923 | 0914 | 0.906 | 0.943 | 0.915 | 0.898 | 0.886 |
| | NDCG@5 | 0.982 | 0.941 | 0.923 | 0.919 | 0.982 | 0.946 | 0.936 | 0.931 | 0.967 | 0.942 | 0.923 | 0.917 |
| | NDCG@10 | 0.968 | 0.920 | 0.902 | 0.894 | 0.968 | 0.936 | 0.928 | 0.926 | 0.964 | 0.936 | 0.917 | 0.913 |
| | NDCG@15 | 0.956 | 0.914 | 0.897 | 0.884 | 0.964 | 0.932 | 0.926 | 0.919 | 0.959 | 0.926 | 0.908 | 0.897 |

mainly consider the other four parameters. The result is shown in Figure 16. We have the following observations.

(1) LCTS and $d$-LCTS outperform the other two similarity measures since they are computed over the transformed trajectories with shorter lengths than corresponding raw trajectories and have smaller computation costs. Note that LCTS measures similarity between POI-matched trajectories by comparing visits on the same POIs, thus it is more efficient than $d$-LCTS without extra spatial distance computation. STLC performs worse than STLCSS since it is a linear combination of spatial and temporal similarities, which can not be optimized with dynamic programming.
(2) The time cost of all similarity measures increases as $|S|$, $|D|$ and $|Q|$ increase. The query size $|Q|$ mainly affects the filtering cost while the other two parameters affects the local computation cost in the refinement phase.
(3) The parameter $k$ has little impact on the search performance. The reason is that the server waits for candidate mobile devices computing similarities locally, thus the search time highly depends on the slowest device.

## 8.5 Effectiveness Evaluation

We employ a relevance judgement method by creating trajectories on a given POI sequence in the city, and checking whether they can be still retrieved as a top-$k$ result. As STLCSS and STLC are defined on raw trajectories, we compare these four similarity measures on Geolife.

**Ground Truth Simulation.** We choose 1,600 trajectories from the dataset and transform them into POI-matched trajectories. For each POI sequence, we generate $k$ trajectories from the shortest paths planned by the navigation service [5] in walk mode and insert them into the trajectory dataset. To evaluate the effectiveness of spatio-temporal similarity measures, we mainly consider three features: sampling rate, GPS error and point shifting.

For the sampling rate, we take each navigation path as base, sample a point every 1,2,..., $k$ seconds along the path between two adjacent POIs, and add $\frac{\tau}{k}$ points around each POI to generate $k$ trajectories, where $\tau$ is the staying time in the POI. For the GPS error, we re-sample each point and move it by a distance of 2-7.8m as the GPS has global average error of $\leq 7.8m$ [8]. For point shifting, we shift all points of the query trajectory along the segments every two POIs by 1m, 2m, ..., $\frac{k}{2}$ in two directions. For all the search results of a query, we evaluate them with three grades: $\underline{\mathbf{2}}$ for those trajectories in the generated ground truth of the query, $\underline{\mathbf{1}}$ for those trajectories not in the generated ground truth of but temporally or spatially overlap with the query, and $\underline{\mathbf{0}}$ for all the remaining trajectories in the dataset.

**Comparison.** The comparison is conducted by changing $k = 5, 10, 15$ using three simulated ground-truth sets. We measure the precision with top-$k$ result precision (**P**@$k$) and *normalized discounted cumulative gain* (**NDCG**@$k$) [23]. We have the following observations.

(1) LCTS has the highest search precision because LCTS contains no extra parameters than the other three similarity measures and irrelevant trajectories are not matched to the original POIs.
(2) As $k$ increases, the precision of all measures degrades since the results contain more trajectories which are not in the ground truth set.
(3) LCTS and $d$-LCTS are more robust to GPS error and point shifting, this is because they are based on detected stay points which are mean points of clusters and less affected by spatial disturbance. The other two point-based measures are more sensitive to spatial disturbance.

## 8.6 Further Discussion

By aggregating observations from the above experiments, we have the following conclusions:

(1) LCTS is a lightweight similarity measure which is robust to GPS error, sampling rate, and point shifting; and LCTS is more efficient than STLCSS and STLC. Meanwhile, our local boost method efficiently accelerates the computation in mobile devices about **8** times.
(2) FGI improves the efficiency of federated top-$k$ search using LCTS by the pruning algorithm over about **4** times. Furthermore, the period temporal index improves the performance of FGI about **2** times compared with traditional temporal index, i.e., interval tree.
(3) Fetra effectively answers the federated search over the federation composed of mobile devices and achieves high performance in different parameter conditions.

# 9 CONCLUSIONS

This paper proposes a federated trajectory search engine called Fetra to efficiently answer the similarity search over a federation composed of mobile devices based on LCTS. Fetra employs a two-level federated index FGI to prune unqualified mobile devices. Furthermore, Fetra optimizes the refinement and LCTS computation locally to improve the search efficiency. In future, we plan to support more similarity measures (such as LORS), extend Fetra to the data federation with millions of mobile devices, and implement our algorithm with modern database system [54].

# REFERENCES


[1] 2014. Foursquare dataset. https://www.kaggle.com/datasets/chetanism/foursquare-nyc-and-tokyo-checkin-dataset.
[2] 2014. JavaFastPFOR. https://github.com/lemire/JavaFastPFOR.
[3] 2021. Map Matching based on GraphHopper. https://github.com/graphhopper/map-matching.
[4] 2021. Spatio-temporal companion. https://www.bloomberg.com/news/articles/2021-11-08/people-you-don-t-know-and-can-t-see-are-close-contacts-in-china.
[5] 2022. Amap. https://www.amap.com/.
[6] 2022. Communication travel card. https://xc.caict.ac.cn.
[7] 2022. FourSquare. https://foursquare.com/.
[8] 2022. GPS Accuracy. https://www.gps.gov/systems/gps/performance/accuracy/.
[9] 2022. Trace together. https://www.tracetogether.gov.sg/.
[10] Israel Edem Agbehadji, Bankole Osita Awuzie, Alfred Beati Ngowi, and Richard C. Millham. 2020. Review of big data analytics, artificial intelligence and nature-inspired computing models towards accurate detection of COVID-19 pandemic cases and contact tracing. *International Journal of Environmental Research and Public Health* 17 (2020), 1–16.
[11] Jie Bao, Tianfu He, Sijie Ruan, and Yu Zheng. 2017. Planning bike lanes based on sharing-bikes' trajectories. In *KDD*. 1377–1386.
[12] Johes Bater, Satyender Goel, Gregory Elliott, Abel Kho, Craig Eggen, and Jennie Rogers. 2016. SMCQL: Secure querying for federated databases. *PVLDB* 10, 6 (2016), 673–684.
[13] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: Differentially-Private Query Processing in Private Data Federations. *PVLDB* 12, 3 (2018), 307–320.
[14] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, Jennie Rogers, Jennie Rogers Saqe, and : Prac. 2020. SAQE: Practical Privacy-Preserving Approximate Query Processing for Data Federations. *PVLDB* 13, 11 (2020), 2150–8097.
[15] Sebastian Baunsgaard, Matthias Boehm, Ankit Chaudhary, Behrouz Derakhshan, Stefan Geißelsöder, Philipp M. Grulich, Michael Hildebrand, Kevin Innerebner, Volker Markl, Claus Neubauer, Sarah Osterburg, Olga Ovcharenko, Sergey Redyuk, Tobias Rieger, Alireza Rezaei Mahdiraji, Sebastian Benjamin Wrede, and Steffen Zeuch. 2021. ExDRa: Exploratory Data Science on Federated Raw Data. In *SIGMOD*. 2450–2463.
[16] Fattaneh Bayatbabolghani and Marina Blanton. 2018. Secure multi-party computation. In *CCS*. 2157–2159.
[17] Edward Buckland, Egemen Tanin, Nicholas Geard, Cameron Zachreson, Hairuo Xie, and Hanan Samet. 2021. Managing Trajectories and Interactions During a Pandemic. In *SIGSPATIAL*. 423–426.
[18] Chao Chen, Yan Ding, Zhu Wang, Junfeng Zhao, Bin Guo, and Daqing Zhang. 2020. VTracer: When Online Vehicle Trajectory Compression Meets Mobile Edge Computing. *IEEE Systems Journal* 14, 2 (2020), 1635–1646.
[19] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *SIGMOD*. 491–502.
[20] Philippe Cudre-Mauroux, Eugene Wu, and Samuel Madden. 2010. TrajStore: an adaptive storage system for very large trajectory data sets. In *ICDE*. 109–120.
[21] Xin Ding, Lu Chen, Yunjun Gao, Christian S Jensen, and Hujun Bao. 2018. UlTraMan : A unified platform for big trajectory data management and analytics. *PVLDB* 11, 7 (2018), 787–799.
[22] Yue Hu, Sijie Ruan, Yuting Ni, Huajun He, Jie Bao, Ruiyuan Li, and Yu Zheng. 2021. SALON: A Universal Stay Point-Based Location AnalysisPlatform. In *SIGSPATIAL*. 407–410.
[23] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems* 20, 4 (2002), 422–446.
[24] Fumiyuki Kato, Yang Cao, and Mastoshi Yoshikawa. 2022. PCT-TEE: Trajectory-based Private Contact Tracing System with Trusted Execution Environment. *ACM Transactions on Spatial Algorithms and Systems* 8, 2 (2022), 1–35.
[25] Martin Kaufmann, Amin A. Manjili, Panagiotis Vagenas, Peter M. Fischer, Donald Kossmann, Franz Farber, and Norman May. 2013. Timeline Index: A Unified Data Sturcture for Processing Queries on Temporal Data in SAP HANA. *SIGMOD*, 1173–1184.
[26] Sunghwan Kim, Taesung Lee, Seung Won Hwang, and Sameh Elnikety. 2018. List intersection for web search: Algorithms, cost models, and optimizations. *PVLDB* 12, 1–13.
[27] Benjamin Krogh and Christian S Jensen. 2016. Efficient in-memory indexing of network-constrained trajectories. In *SIGSPATIAL*. 1–17.
[28] Daniel Lemire and Leonid Boytsov. 2015. Decoding billions of integers per second through vectorization. *Software: Practice and Experience* 45, 1 (2015), 1–29.
[29] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. 2008. Mining User Similarity Based on Location History. In *SIGSPATIAL*. 1–10.
[30] Ruiyuan Li, Huajun He, Rubin Wang, Sijie Ruan, Tianfu He, Jie Bao, Junbo Zhang, Liang Hong, and Yu Zheng. 2021. TrajMesa: A Distributed NoSQL-Based Trajectory Data Management System. *IEEE Transactions on Knowledge and Data Engineering* 14, 8 (2021).
[31] Essam Mansour, Kavitha Srinivas, and Katja Hose. 2021. Federated Data Science to Break Down Silos [Vision]. *SIGMOD Record* 50, 4 (2021), 16–22.
[32] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *SIGSPATIAL*. 336–343.
[33] Sarana Nutanong, Edwin H. Jacox, and Hanan Samet. 2011. An incremental Hausdorff distance calculation algorithm. *PVLDB* 4, 8 (2011), 506–517.
[34] Rafael Pérez-Torres, César Torres-Huitzil, and Hiram Galeana-Zapién. 2016. Full on-device stay points detection in smartphones for location-based mobile applications. *Sensors* 16, 10 (2016).
[35] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. 2020. FetchSGD: Communication-Efficient Federated Learning with Sketching. In *ICML*.
[36] Markus Schmuck and Martin Z Bazant. 2015. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.* 75, 3 (2015), 1369–1401.
[37] Shuo Shang, Lisi Chen, Zhewei Wei, Christian S Jensen, Kai Zheng, and Panos Kalnis. 2017. Trajectory similarity join in spatial networks. *PVLDB* 10, 11 (2017), 1178–1189.
[38] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. Dita: Distributed in-memory trajectory analytics. In *SIGMOD*. 725–740.
[39] Yexuan Shi, Yongxin Tong, Yuxiang Zeng, Zimu Zhou, Bolin Ding, and Lei Chen. 2021. Efficient Approximate Range Aggregation over Large-scale Spatial Data Federation. *IEEE Transactions on Knowledge and Data Engineering* (2021).
[40] Josh Stoddard, Adam Mustafa, and Naveen Goela. 2021. Tanium reveal: A federated search engine for queryinunstructured file data on large enterprise networks. *PVLDB* 14, 12 (2021), 3096–3109.
[41] Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. 2020. A survey of trajectory distance measures and performance evaluation. *VLDB Journal* 29 (2020), 3–32.
[42] Mingjie Tang, Yongyang Yu, Qutaibah M Malluhi, Mourad Ouzzani, and Walid G Aref. 2016. LocationSpark: A Distributed In-Memory Data Management System for Big Spatial Data . *PVLDB* 9, 13 (2016), 1565–1568.
[43] Yongxin Tong, Xuchen Pan, Yuxiang Zeng, Yexuan Shi, Chunbo Xue, Zimu Zhou, Xiaofei Zhang, Lei Chen, Yi Xu, Ke Xu, and Weifeng Lv. 2022. Hu-Fu : Efficient and Secure Spatial Queries over Data Federation. *PVLDB* 15, 6 (2022), 1159–1172.
[44] Michail Vlachos, Dimitrios Gunopulos, and George Kollios. 2002. Robust similarity measures for mobile object trajectories. In *ICDE*. 721–726.
[45] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In *ICDE*. 673–684.
[46] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, and Gao Cong. 2021. A survey on trajectory data management analytics and learning. *Comput. Surveys* 54, 2 (2021), 1–39.
[47] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast large-scale trajectory clustering. *PVLDB* 13, 1 (2019), 29–42.
[48] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A search engine for trajectory data. In *SIGIR*. 535–544.
[49] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. 2019. Adaptive Federated Learning in Resource Constrained Edge Computing Systems. *IEEE Journal on Selected Areas in Communications* 37, 6 (2019), 1205–1221.
[50] Jung-im Won, Sang-wook Kim, Ji-haeng Baek, and Junghoon Lee. 2009. Trajectory clustering in road network environment. In *CIDM*. 299–305.
[51] Chen Wu, Sheng Wang, and Zhiyong Peng. 2022. Motorch: An On-Device Trajectory Data Management System During a Pandemic. In *SIGSPATIAL*. http://shengwang.site/papers/22SIGSPATIAL.pdf
[52] Dong Xie, Feifei Li, and Jeff M Phillips. 2017. Distributed trajectory similarity search. *PVLDB* 10, 11 (2017), 1478–1489.
[53] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. 2016. Simba : Efficient In-Memory Spatial Analytics. In *SIGMOD*. 1071–1085.
[54] Zhenkun Yang, Chuanhui Yang, Fusheng Han, Mingqiang Zhuang, Bing Yang, Zhifeng Yang, Xiaojun Cheng, Yuzhong Zhao, Wenhui Shi, Huafeng Xi, Huang Yu, Bin Liu, Yi Pan, Boxue Yin, Junquan Chen, and Quanqing Xu. 2022. OceanBase: A 707 Million tpmC Distributed Relational Database System. *PVLDB* 15, 12 (2022), 3385–3397.
[55] Byoung-Kee Yi, H V Jagadish, and Christos Faloutsos. 1998. Efficient retrieval of similar time sequences under time warping. In *ICDE*. 201–208.
[56] Jia Yu, Jinxuan Wu, and Sarwat Mohamed. 2015. Geospark: A cluster computing framework for processing large-scale spatial data. In *SIGSPATIAL*. 4–7.
[57] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, Michael J Franklin, and Scott Shenker. 2012. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI*. 15–28.
[58] Yanping Zhang, Chenghong Wang, David Pujol, Johes Bater, Matthew Lentz, Ashwin MacHanavajjhala, Kartik Nayak, Lavanya Vasudevan, and Jun Yang. 2020. Poirot: Private contact summary aggregation. In *PPML@NeurIPS*. 774–775.
[59] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2008. Understanding Mobility Based on GPS Data. In *Ubicomp*. 312–321.
[60] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *WWW*. 791–800.