Distributed Mining of Popular Paths in Road Networks

Panagiota Katsikouli
Univ.Edinburgh, Informatics &
Univ.Lyon, Inria, INSA-Lyon, CITI
Lyon, France
panagiota.katsikouli@inria.fr

Maria Sinziana Astefanoaei
Univ.Edinburgh, Informatics
Edinburgh, UK
m.s.astefanoaei@sms.ed.ac.uk

Rik Sarkar
Univ.Edinburgh, Informatics
Edinburgh, UK
rsarkar@inf.ed.ac.uk

***Abstract*—We consider the problem of finding large scale mobility patterns. A common challenge in mobility tracking systems is that large quantity of data is spread out spatially and temporally across many tracking sensors. We thus devise a spatial sampling and information exchange protocol that provides probabilistic guarantees on detecting prominent patterns.**

For this purpose, we define a general notion of *significant popular paths* that can capture many different types of motion. We design a *summary sketch* for the data at each tracking node, which can be updated efficiently, and then aggregated across devices to reconstruct the prominent paths in the global data. The algorithm is scalable, even with large number of mobile targets. It uses a hierarchic query system that automatically prioritizes important trajectories – those that are long and popular. We show further that this scheme can in fact give good results by sampling relatively few sensors and targets, and works for streaming spatial data. We prove differential privacy guarantees for the randomized algorithm. Extensive experiments on real GPS data show that the method is efficient and accurate, and is useful in predicting motion of travelers even with small samples.

***Index Terms*—Trajectories, road networks, sensor networks, distributed algorithm, clustering, spatial query processing, differential privacy**

I. Introduction

Modern sensing and localization technologies have given rise to large quantities of location and trajectory data. As a result, the analysis of such trajectories has become a fundamental research topic. For example, in urban scenarios, understanding of paths used by vehicles is essential in transport management and smart cities [22], [21], [17], [21], [9]. Digital maps, like OpenStreetMap, use trajectory information to update and refine map data. Knowledge of movements of the masses can be beneficial to adapting wireless infrastructures to changing traffic demands [19].

Past works related to tracking of moving targets with sensors have usually focused on identifying individual targets and localizing them continuously [4], [20], thus recording their trajectories. In this paper, we look beyond tracking individual targets and consider the problem of extracting common patterns in moving targets from data stored in distributed sensors over a large region.

A major challenge in this form of trajectory processing is that the dataset is spread out spatially over many sensors, making it hard to collect, process, and store. Transmitting all data to a central location incurs impractical costs when tracking a large population. We thus take the approach of efficiently filtering the data locally to obtain the aggregate global results.

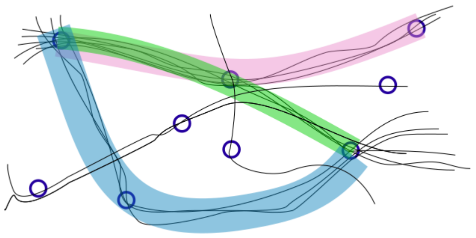


Fig. 1. Sensing devices detect users passing in their vicinity. Clusters consist of long sub-trajectories with sufficient support, and different clusters can overlap, increasing the difficulty of the problem.

In the example in Fig. 1 we have sensor nodes (circles) deployed in a domain and they can detect the users passing in their vicinity. There are three main clusters: a pink, a green and a blue cluster. The objective is to detect such clusters among the motion of large number of travelers, where data is recorded by spatially distributed sensors.

The problem is challenging because clusters of complete trajectories do not necessarily represent mobility patterns. Consider, for example, travelers that have very different start and destination locations, but nonetheless share a sequence of highway routes in the middle. We are interested in such sections of trajectories that are common to many users, which may be applied to infer road networks [1], and traffic management. However, finding clusters consisting of such sub-trajectories is a substantially different problem than clustering point sets. In fact, finding large sets of matching subtrajectories is NP-hard in the geometric version of the problem [5].

Our Contributions. We formulate the problem of significant trajectory patterns in a road network as finding paths of sufficient length ℓ that have been used by sufficiently many travelers k. The static sensors in the network are assumed to store the id of each mobile object passing through it. A popular path thus consists of a subset of k ids that appear in all sensors along the path. It is this subset of ids and the sequence of sensors that we must identify.

Solving the popular paths problem probabilistically in a distributed setup can be split into two fundamental challenges. The first is to determine sensors that need to be sampled and should exchange information to detect the common sets of size k. Note that the set of ids in each sensor can be large, and checking against all sensors in the system can be expensive. The second problem is to minimize the information exchanged between any two selected sensors to detect the intersection set efficiently, even when each sensor may store a large number of ids.

For the first question, we perform a hierarchic sampling of sensors. The higher levels of the sampling contain a sparse deployment of sensors, and detect the longer patterns of paths, while the lower levels with dense deployment of sensors detect shorter paths. We show in theoretical analysis that a small fraction (corresponding to $1/\ell$) of sampled sensors serves to detect paths of length ℓ or more with high probability. This implies that in resource constrained scenarios, a sparse deployment of sensors at only a few road junctions suffices.

For the second question, we devise a sketching based protocol to compare id sets at pairs of sensors. The sketching protocol incrementally exchanges larger and larger samples, until the sensors can infer with confidence that they have seen a common set of size at least k.

These mechanisms have the property that given a query, they identify the more prominent patterns first (i.e., paths of larger values of k and ℓ). As a result, they can be easily adapted to low time and resource availability, where the most important results are guaranteed to be delivered. Detail of results degrade gracefully with diminishing resources.

This randomized framework is found to be orders of magnitude more efficient than a deterministic method in both theory and experiments, and it produces results with very low errors. It can be applied to streaming mobility data and it provably offers differential privacy.

Figure 2 shows that the results for applying the randomized and deterministic algorithms on a particular dataset are virtually almost identical. Our extensive experiments show several interesting results, which suggest that our technique can produce good results in a sparse deployment and with small number of samples. Figure 3(a) shows popular paths with a high support value of $k \geq 40$ in a set of 200 trajectories, thus identifying highly popular patterns. Figure 3(b) shows the patterns restricted to one particular junction, displaying the most popular destinations from that point.

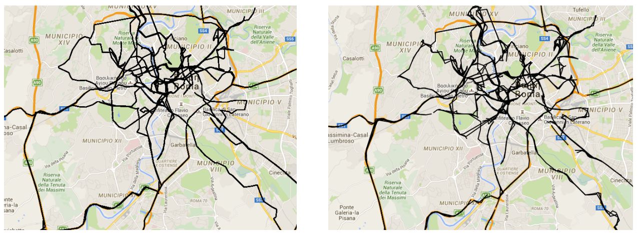
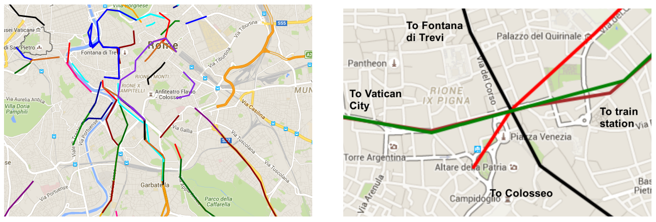

(a) (b)

Fig. 2. (a) Popular paths found by deterministic method. (b) Popular paths founds by randomized method. The results are visually almost identical in the two cases.



(a) (b)

Fig. 3. (a) Popular paths found from $\approx$ 200 taxi trajectories in the city of Rome. The paths shown have popularity at least 40. (b) All popular routes from a single intersection lead to historic landmarks. (better seen in color)

II. Related works

Finding clusters and medians of trajectories have been treated from the computational geometry and database points of view in [6], [15], [18]. In these approaches, the focus is usually on clustering a given set of complete trajectories with the same starting and ending points.

Tracking of groups or clusters of moving objects has been studied in several works – see for example [11], [14] and references therein. These methods aim to keep track of groups as they move at the same time, and thus do not apply to clustering of trajectories that have followed similar paths at different times. A recent work [8], employs MinHash signatures to record mobile entities in a sensor network and answer queries regarding persistent paths. The technique here is geared toward only very large sets of moving objects – constant fractions of the whole population – and thus is difficult to apply for smaller clusters, or where the population size may be unknown.

Mining of trajectory patterns has been addressed in [12]. In this work, regions of interest must be identified manually. A pattern is then defined as a sequence of regions of interest along with the travel times in between.

Methods based on randomized sampling have been applied to streaming data for more efficient analysis of long timeseries. However, in our work we consider a distributed scenario where we want to reduce the quantity of data stored and communicated between sensors and at the same time guarantee consistency across the sensors in the preserved data.

The problem of clustering geometric sub-trajectories has been addressed in [5]. Here, in addition to the parameters popularity k and length ℓ, a thickness parameter d is used in clusters. All sub-trajectories in a cluster are required to be within distance d of each-other, measured under the Fréchet distance [2]. The results in [5] show the problem of finding clusters with maximal ℓ to be NP-hard, and constant factor approximations to be $O(N^2)$ in computational cost where N is the total input size. Therefore, this method is impractical for distributed operation, large datasets and streaming data.

Unlike these methods, we take a more practical point of view, where we transform the problem to finding sub-trajectory clusters in a graph such as a road network.

III. Problem description

Let us consider the setup where mobile objects are tracked by static sensors or access points on a road network. We assume that each sensing device can identify mobile objects.

The sensors are assumed to be deployed densely, at unit intervals along any path. (We will show later that few sensors in fact suffice.) The sensors record when an object is within range. For any mobile object, the temporal sequence of sensors that have tracked it, represent the object's trajectory in space.

Let us now define the set of significant popular paths in a dataset, based on two properties: they should have sufficient length, and they should have sufficient *support*, meaning they should be used by enough number of travelers:

Definition 3.1. (k, ℓ) **popular paths:** *The set of maximal-length paths of span at least ℓ that have support of at least k (i.e. used at least k times).*

We define *span* to be the Euclidean distance between the start and end points of the path.

Both k and ℓ are lower bounds. Longer overlaps of trajectories and those used by more targets are likely to be more important and of interest to the user. Also note that we are only interested in the *maximal overlaps*, that is, if a path P is in the output, no subpath of P can be in the output.

The parameters k and ℓ are to be supplied by the user. This lets the user control the query with respect to the application and available resources. For example, in an urban region, only routes of interest are those with support in several thousands, while in rural regions, supports are expected to be lower, but distances larger. Also note that the definition allows the user to simply query for popular path by numbers – e.g. "Paths of length at least 3 km used by at least 500 people", without any accessory knowledge such as geographic range of operation, or size of population, as required by [8].

For a given trajectory set $\mathcal{T} = \{T_1, T_2, \ldots T_n\}$, let us assume that each trajectory is of length at most m. The essential challenge in computing popular subtrajectories comes from the fact that there are $\Omega(nm^2)$ subpaths of all possible lengths over the complete set of n trajectories.

One possible approach to the (k, ℓ) queries problem is to consider all possible sub-trajectories, and sort them in $\Omega(nm^2 \log(nm^2))$ time. Then, matching sub-trajectories are merged, which gives us the k-values for each sub-trajectory. We can create a sorted index on k and use that to answer the query. This pre-processing is already impractical for datasets consisting of large trajectories.

Data structures such as generalized suffix trees can improve query time for centralized substring search [7], but how they can be utilized for our distributed problem is far from trivial, and cannot avoid the worst case $\Omega(nm)$ cost.

IV. Randomized distributed algorithms

The model we assume is a graph G representing a road network. The nodes correspond to locations where the static sensors are placed. Nodes are connected via edges which can correspond to streets or sequence of streets. Throughout the presentation, we therefore use the terms sensors and nodes interchangeably.

A trajectory of a mobile object $T = v_1, v_2, \ldots$ passes though a sequence of sensors. Let $v_1, v_2, \ldots$ be a sequence of sensors that detects and records the mobile object. We represent the whole set of ids of the objects detected by sensor u as $S(u)$. We also use $L(P)$ and $K(P)$ to represent the span (or length) and support of a path P.

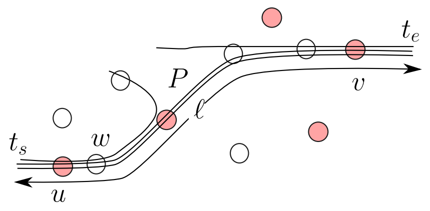


Fig. 4. A cluster P of sub-trajectories passes through sensors u and v.

Approach Overview. We consider a hierarchy of the sensors in G where at each level of the hierarchy only a subset of the sensors is active (i.e., is used). Consider the example of Figure 4, at a particular level of the hierarchy, where only the colored sensors are the active ones. We choose and compare (active) sensors at random at each level in order to detect candidate popular paths.

Suppose a path like P is sufficiently long, we will show that at least some of the sensors on it will be active at the considered level. First, sensors u and v estimate the number of trajectory ids they have both seen $|S(u) \cap S(v)|$ using an intersection estimation protocol. Second, sensors u and v estimate the span of P.

If the estimated size of intersection is at least k and the distance between the selected sensors is at least $\approx \ell$, the sensors can conclude that P is a candidate for a popular pattern. Then it can be checked that set $S(u) \cap S(v)$ passes through a path in G. We describe an algorithm that runs through all levels of the sensors hierarchy and detects all paths down to length ℓ.

We will start the detailed algorithm description with the intersection computation protocol to estimate the size of $|S(u) \cap S(v)|$.

A. Intersection estimation

The purpose of this protocol is to compute a fundamental quantity – the size of intersection of sets of mobile ids seen by sensors u and v. The inference of popular paths will depend on efficient estimation of this quantity. Consider the scenario of Fig. 4 where a path cluster P passes through nodes u and v, and $S(u)$ and $S(v)$ are sets of trajectory ids passing through them. Let $\mu = K(P) = |S(u) \cap S(v)|$ be the size of the intersection. Our goal is to efficiently find an estimate for μ.

Each of $S(u)$ and $S(v)$ can be individually large, thus a simple protocol such as transmitting $S(u)$ to v and computing the intersection at v is inefficient as it requires quadratic computation cost and communication cost proportional to the size of the set in consideration. We describe an iterative protocol that can estimate the size at considerably lower communication cost.

Our protocol will operate by comparing random samples of $S(u)$ and $S(v)$ for iteratively larger sizes. At every node u, we first use a hash function H_1 to save a random subset of $S(u)$. Let us say that the size of the saved subset $H_1(S(u))$

is $|S(u)|/2$. This can be realized by using the function H_1 to hash items to 2 buckets, from which we preserve elements in the first bucket and discard everything else. Using the same hash function at every node ensures that nodes u and v will save the same subset of $S(u) \cap S(v)$.

Next, each node u pre-computes a sequence of sub-samples using iterative hashes: $H_2(H_1(S(u))), H_3(H_2(H_1(S(u)))), \ldots$ etc of sizes $|S(u)|/4, |S(u)/|8, \ldots$ respectively. Let us refer to these sets in the reverse order as: $B_0^u, B_1^u, \ldots,$ from smallest to largest. That is, if hashes go upto H_β, then $B_0^u = H_\beta(H_{\beta-1}(\ldots S(u) \ldots))$. The sets B are the intersection sketches of nodes.

Observation 4.1. *If $x \in S(u)$ and $x \in H_i(\ldots S(u))$, then $x \in S(v) \Rightarrow x \in H_i(\ldots S(v))$.*

When computing intersection $S(u) \cap S(v)$, let us suppose without loss of generality, that $S(u)$ is the smaller of the two sets. Nodes u and v can decide this by simply exchanging the size of their contents. They also exchange the size of u's hash hierarchy given by β. Next, node u follows the sketch transmission procedure in rounds:

Algorithm 1 Intersection protocol for node u with smaller $S(u)$.

```
Input: Hierarchy size $\beta$, Recursive Hash buckets
    $H_2(H_1(S(u))), H_3(H_2(H_1(S(u)))), \ldots$
Output: size estimate $\rho$
1: for $i = 0$ to $\beta$ do
2:     Find $j$ such that $B_i^u = H_j(\ldots(S(u)\ldots))$
3:     Send $(B_i^u, j)$ to $v$
4:     receive message $m$ from $v$
5:     if $m = terminate$ then
6:         BREAK loop
7:     end if
8: end for
9: receive size estimate $\rho$ from $v$
```

On the other side, v follows the procedure of comparing u's sketches with its own:

Algorithm 2 Intersection protocol for node v with larger $S(v)$.

```
Input: Hierarchy size $\beta$, threshold $\eta$, Recursive Hash buckets
    $H_2(H_1(S(v))), H_3(H_2(H_1(S(v)))), \ldots$
Output: Size estimate $\rho$
1: $\rho = 0$
2: for $i = 0$ to $\beta$ do
3:     Receive $(B_i^u, j)$ from $u$
4:     Set $B_w^v = H_j(\ldots(S(v)\ldots))$
5:     if $|B_i^u \cap B_w^v| > \eta$ then
6:         Set $\rho = |B_i^u \cap B_w^v| \cdot 2^j$
7:         Send terminate to $u$. Break.
8:     end if
9: end for
10: send $\rho$ to $u$
```

At the end of computation, the variable ρ indicates the estimate for the actual size μ of the intersection. The threshold η determines the confidence of the intersection being large enough. The following theorem shows that $\eta \geq \frac{3}{\varepsilon^2} \ln \frac{1}{\delta}$ suffices to ensure that the estimate ρ is within an ε factor with probability at least $1 - \delta$. Note that if a *terminate* message is not generated and $\rho = 0$, it implies that the intersection size was not estimated with the required confidence $1 - \delta$.

Theorem 4.2. *The distributed intersection protocol with $\eta \geq \frac{3}{\varepsilon^2} \ln \frac{1}{\delta}$ ensures that:*

$$\Pr(|\rho - \mu| \geq \varepsilon\rho) \leq \delta.$$

Proof: Suppose H_j is the hash producing the final comparison sets. B_i^u and B_w^v are obtained in effect by selecting with probability $1/2^j$ from $S(u)$ and $S(v)$. The probability that a particular element of $S(u) \cap S(v)$ is in B_i^u is $1/2^j$, thus the expected number of elements of $S(u) \cap S(v)$ in B_i^u is $\mu/2^j$.

From Obs. 4.1, $S(u) \cap S(v) \cap B_i^u = S(u) \cap S(v) \cap B_w^v = B_i^u \cap B_w^v$.

Let $\eta = E[|B_i^u \cap B_w^v|]$. Thus, in expectation, $\eta = \rho/2^j = \mu/2^j$. Using Chernoff bound [16] and setting $\eta = \frac{3}{\varepsilon^2} ln \frac{1}{\delta}$:

$$\Pr(|\mu/2^j - \eta| \geq \varepsilon\eta) \leq e^{-\eta\varepsilon^2/3} \leq e^{-\frac{3}{\varepsilon^2} \ln \frac{1}{\delta} \varepsilon^2/3} \leq \delta.$$

Scaling all terms on the left hand side by 2^j gives the required probability $\leq \delta$. □

The theorem implies that a sample size of $\frac{3S(u)}{\mu\varepsilon^2} \ln \frac{1}{\delta}$ suffices to reliably estimate any μ. Based on this, we show that:

Theorem 4.3. *The communication and storage required at node u by the intersection size estimation protocol to detect intersections of size k or more, is bounded by*

$$O\left(\min\left(\frac{|S(u)|}{\mu}, \frac{|S(u)|}{k}\right)\right).$$

Proof: For the sample B_i^u to contain $\frac{3}{\varepsilon^2} \ln \frac{1}{\delta}$ elements out of μ, the sample size must be $\frac{3S(u)}{\mu\varepsilon^2} \ln \frac{1}{\delta}$. As we are only interested in intersections larger than k in size, we only need to consider samples up to: $|B_i^u| \leq \frac{3S(u)}{k\varepsilon^2} \ln \frac{1}{\delta}$. Thus, this is the largest set that node u needs to store or transmit. □

Therefore, a node only needs to store and use $O(S(u)/k)$ elements in order to estimate the intersection size. Observe that as data volume increases, the patterns of interest are those that satisfy larger k, for example, in thousands. Thus, in a busy urban scenario, the information sample needed by this protocol is only a small fraction (such as one thousandths) of the data.

Based on the above, we can add an additional termination condition to the protocol, where node u stops computation when $|B_i^u| \geq \frac{S(u)}{2^j} \frac{\mu}{k}$ to conclude that the intersection is smaller than k.

$B_i^u \cap B_w^v$, along with the information that it was computed at level j of hash hierarchy, can be seen as a sketch of the intersection $S(u) \cap S(v)$. Such a sketch has constant size at each level j and it can be used to verify the existence of the same subset of elements at other nodes – through which the trajectory cluster passes. Set intersection can also be estimated using MinHash sketches [13], [8], however, these are restricted to detecting intersections a constant fraction of the set sizes. In contrast, our sketch protocol can be applied to detect intersection with any user specified number of elements.

B. Path length (span) estimation.

For this analysis, we assume that edges in the graph have bounded length, and neighboring vertices are not far from each other. This is simply for ease of explanation. As we will show, a small number of sensors relative to the graph size in fact suffices to estimate path lengths with high probability. More precisely, we focus on the detection of trajectories with a large *span*. We sometimes use length and span interchangeably. Suppose we sample the sensors such that each node is selected to be active with probability p.

Observation 4.4. *A sampling rate of $p \geq \frac{1}{L}\ln\frac{1}{\delta}$ suffices to have at least one sampled sensor on a trajectory T of span L with probability at least $1-\delta$, where $L \geq \ell$.*

Proof: The probability that a particular sensor node is not selected is $1-p$, therefore: the probability that none of the L nodes along T is selected is $\leq (1-p)^L \leq \delta$ (Using $(1-p)^{1/p} \leq 1/e$ and setting $p = \frac{1}{\frac{1}{L}ln\frac{1}{\delta}}$). □

By using union bound, $p \geq \frac{1}{L}\ln\frac{2}{\delta}$ suffices to obtain two sensors on a trajectory with probability $1-\delta$. Notice that having two active sensors on a trajectory T suffices to test whether T belongs in a popular enough cluster of trajectories. Another consequence of this observation is that a corresponding sampling rate in fact suffices to estimate the length of a path with a small error. This follows from the fact that with high probability there is a sampled sensor close to both the start and end of a trajectory:

Corollary 4.5. *Given a trajectory T of length ℓ, let t_s and t_e be the segments of length $\frac{\varepsilon}{2}\ell$ at the start and end of T (see Fig. 4). If $p \geq \frac{1}{\varepsilon L}\ln\frac{2}{\delta}$ then the probability that both t_s and t_e contain a sampled sensor is at least $(1-\delta)$.*

The proof follows from Obs. 4.4 using union bound. It implies that sampling at this rate produces enough sensors to estimate the length or span of any trajectory to within a factor of $(1-\varepsilon)$.

C. Finding (k, ℓ) popular paths

We now describe the algorithm that efficiently finds all the (k, ℓ) popular paths, by combining the results above. We are looking for the longest, and more important paths first. In order to do that, our algorithm operates in rounds, where in each round a denser fraction of the deployed sensors is considered. This is achieved by building an hierarchy of the sensors in G, as follows:

Hierarchy building. The hierarchy consists of sets G_i denoting the selected (i.e., active) nodes at level i (G_0 contains all nodes of G). A sensor node u in G_i makes a random choice of probability q to promote itself to G_{i+1} and thus be selected at that level. Thus, each node promotes itself to a higher level until it makes a decision not to promote any more. The hierarchy of G_i nodes can be thus built through purely local computations. Each G_i consists of a set selected from G with probability q^i.

Additionally, each node u knows the sequence of hash functions $H_1, H_2, \ldots$ and maintains the corresponding subsets $B^0, B^1, \ldots$, as described earlier.

Query processing. Given a query with parameters k, ℓ, our algorithm starts at the topmost level of the hierarchy.

At any level i, a node $u \in G_i$ considers the set of nodes $\{v \in G_i : |u-v| \leq \ell\}$ to estimate $|S(u) \cap S(v)|$. We call these nodes the i-neighbors of u. For any cluster set $C = S(u) \cap S(v)$, the length of the cluster can be estimated as $|u-v|$, which denotes the distance (i.e., number of edges) between u and v. If $|u-v| \geq \ell$ and $|C| \geq k$, then the path between u and v is a candidate cluster.From Obs. 4.4, at any level i, we can expect to have two nodes on any trajectory of length $\ln(2/\delta)/q^i$ with probability at least $1-\delta$.

Next, we need to verify that the trajectories in C travel together from u to v. Suppose c is the constant sized sketch of C, computed according to our intersection protocol. c can be checked at all neighbours of u in G_0, and identify the node through which C passes. This node can further forward c to trace the path up to v, essentially doing a breadth first search. We avoid paying the high computational cost of breadth first by randomly picking a trajectory $T' \in C$ and check if c is found in the sensors constituting T' (or their 1-hop neighbours) between sensor nodes u and v. This technique can be used to search further and trace the complete path of C beyond u and v in t_s and t_e.

Once the path is traced, the set c can be removed from all nodes in the path to avoid duplicate results at lower level searches. Then we can proceed to level $i-1$ and repeat the process.

Theorem 4.6. *Any (k, ℓ) popular path is detected with probability $1-\delta$ in $G_{log_q(\ln(4/\delta)/\ell)}$ using a trajectory sample size $O(n/k)$.*

Proof: As discussed above, popular trajectories of length ℓ can be detected with probability $(1-\delta/2)$ in $G_{log_q(\ln(4/\delta)/\ell)}$. Analogously, a sample size $O(S(u)/k)$ suffices to detect intersections of size k with probability $(1-\delta/2)$. In the worst case, when $S(u) = n$, the sample size required is $O(n/k)$. By using union abound, the given (k, ℓ) trajectory is detected with probability $1-\delta$. □

D. A distributed in-network query system

We describe how popular path queries are handled efficiently in a distributed system.

Localized query at node u. Let us consider first a restricted version of the problem, where the user wants to know (k, ℓ) popular paths through a particular node u. This query is important on its own. It corresponds to asking the utility of a busy intersection or road (see for example Figure 3(b)).

Node u can operate top-down starting from the highest level of the hierarchy. Let $L_i = \ln(2/\delta)/q^i$ be the scale of level i.

At level i, u communicates with all nodes in G_i in a radius of L_i around itself and computes intersections of id-sets (note that in this case, u itself may not be in G_i) and searches for (k, ℓ) paths as described earlier. Since sampling rate for level i is $1/q^i (\approx (1/L)$, there are $O(L)$ nodes within radius L with which u computes intersections. Thus, in a domain of diameter D, u performs at most $O(D)$ comparisons.

Global query. For the version where (k, ℓ) paths through all nodes are required, the procedure starts at pairs of nodes at the topmost level and moves through lower levels as before.

In both versions, each successful intersection computation is followed up with a tracing of the cluster using the constant sized summary, to check continuity of the path. The expected cost of tracing the clusters is $O(\mathcal{L})$ where $\mathcal{L}$ is the total length of the (k, ℓ)-clusters. We find in our experiments that if a cluster of trajectories is common in two different nodes, all the trajectories of the cluster usually traverse the same path in between. Thus, this verification step can be omitted in practice.

E. Properties of the randomized algorithm

Our system has several desirable properties:

1) Adaptive resolution, as it outputs the longest and most important paths first. It can thus be resource efficient in constrained situations.
2) Adaptation to streaming data, as new data can be incrementally incorporated in the system in constant time.
3) Storage and communication efficiency, as each node needs only a sample size $S(u)/k$.
4) Adjustement for directed paths, by recording the timestamp when each mobile object is seen by a node.
5) Differential privacy, on which we will elaborate next.

Differential privacy [10] was introduced as a measure of privacy for aggregate query mechanisms. Let x be the record of a particular user, which is included in the database. By making the same query on a dataset D and then on $D \setminus \{x\}$, the adversary can infer the value of the attribute for the user, without explicitly asking for this value. An ε-differentially private randomized algorithm intends to guard against such attacks, where smaller values of ε provide better privacy.

Theorem 4.7. *The popular paths algorithm is* $\ln \frac{k}{k-1}$ *differentially private.*

Proof: Suppose an adversary makes a (k, ℓ)-popular paths query on two trajectory datasets D_1 and D_2 where $D_1 = D_2 \setminus \{x\}$. Suppose $Y = \mathcal{M}(D_1)$ and $Z = \mathcal{M}(D_2)$ are the results returned by the algorithm, respectively. Although the trajectory x does not affect Y, it affects Z with probability $\frac{1}{k}$, since the algorithm works with a random sample of proportion $\frac{1}{k}$.

Our algorithm uses the same independent sampling on all queries, therefore $\Pr[Y = Z] \geq \frac{k-1}{k}$, since this is simply determined by the probability of x being in the sample. Now consider any subset $S \subseteq \text{Range}(\mathcal{M})$ such that $Y \in S$. Then $\Pr[Z \in S] \geq \frac{k-1}{k}$. In this case, since $\Pr[Y \in S] = 1$, we have that $\frac{\Pr[Y \in S]}{\Pr[Z \in S]} \leq \frac{k}{k-1} \leq e^{\ln \frac{k}{k-1}}$. □

Observe that by virtue of the small $1/k$-fraction sample requirement, the privacy is strong. For example, for $k = 100$, $\ln \frac{k}{k-1} \approx 0.01$, which implies strong privacy.

V. Experimental results

We implemented our algorithms and emulated the system on various datasets of GPS traces. The results discussed here correspond to a dataset of taxicab traces [3]. The main observations are:

- The lengths of subpaths can be estimated accurately even with small samples of sensors.
- The support of subpaths can be estimated accurately using small samples of the trajectory set and the summary based intersection computation method.
- Given values k, ℓ, our technique detects almost all popular paths, even with small sampling rates of sensor nodes, at significantly lower costs from the deterministic approach.
- The detected popular clusters let us predict the motion of collections of users along the popular routes.
- Paths that pass through two different sensor neighborhoods, usually traverse a similar trajectory in between. Thus, a sparse deployment of sensing devices can suffice to obtain accurate estimate of clusters at low cost.

The dataset consists of taxi daily trips with sampling intervals of $\approx$ 7 secs. In the experiments, we use trajectories extracted from 150 trips of randomly selected taxi drivers.

We assume a deployment of wireless devices in a square grid graph G of spacing 400 meters. That is, we assume there is a wireless device (a sensing node) covering each square in the grid. This separation corresponds approximately to cellular base station densities in urban regions. Each GPS trajectory is converted to a sequence of cell locations on the grid based on the nearest base station location. We assume one hop on this graph to be the unit of distance. We present our results in terms of hop distance (i.e., 1 unit distance = 1 hop = $400m$).

In Fig. 3(a) we show a set of popular paths computed from 200 trajectories, with support at least 40 users and average length 7 hops (i.e., $\approx 3km$). Salient features of traffic, such as popular streets in city center, can be detected from this small example.

Accuracy of length detection. We consider approximately 400 subtrajectories of lengths from a few to many kilometres (maximum, $100km$).

Each level i of the node hierarchy corresponds to a sampling frequency q. For each trajectory t in the grid G, we consider the set of q-sampled sensing nodes that belong to level i, and find the sampled nodes u and v that are closest to the ends of the subtrajectory. If $L(u, v)$ is the distance from u to v in G the length of the sub trajectory is estimated as $L(u, v) + 2/q$.

Fig. 5(a) shows that, even at very high levels of hierarchy, where nodes are sampled sparsely, the error in estimates of length of subtrajectories is small. For subpaths of smaller lengths (e.g, $5km$) the error is higher, as our sensing nodes have a spacing of $400m$. However, the error is not very large

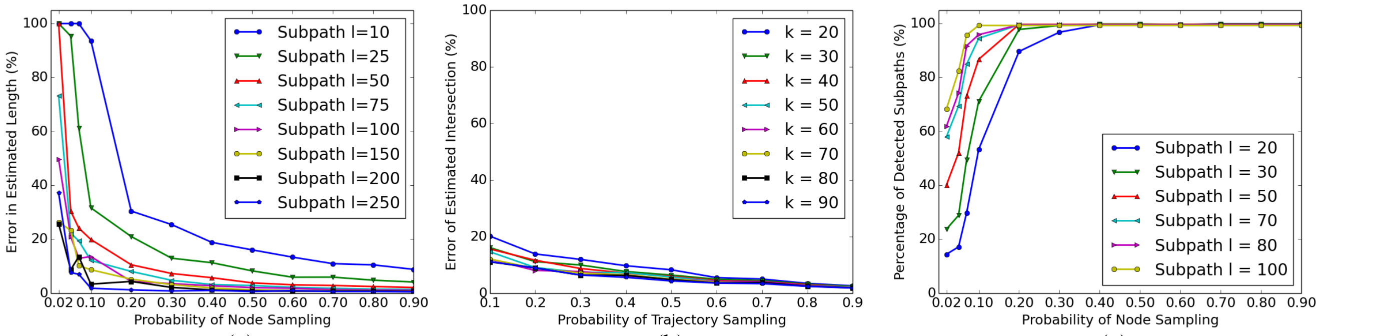


Fig. 5. (a) Even at small sampling rates of nodes, the error in estimate of length of a long subpath is on average less than 5%. (b) Even at small sampling rates of trajectories, the estimated support of a subpath has small errors. (c) Almost all of the popular paths are detected, even at small sampling rates.

and it is less than 5% even at sparse sampling rates of 0.3 for longer and therefore more significant paths.

Accuracy of support detection: Fig 5(b) shows the accuracy of estimating the size of the intersection between two sensing nodes, for various sampling rates of the trajectory set, using our intersection protocol. We observe that, even for small intersection sizes and sampling rates, the error of the estimation is at most 20%, while it becomes more accurate as the size of intersection increases.

Comparison with deterministic method: We would like to ensure that any popular subpath is detected with high probability. We ran the deterministic method described in Section III on the dataset of 400 subtrajectories, for queries of various support values ($k \geq 15$) and various lengths (from 20 to 100 hops). This method finds all popular paths. For each popular path found we check if there are at least two sampled sensing nodes on them. In Fig. 5(c) we plot the percentage of popular paths detected at different sampling rates of the sensing nodes. Even at very small sampling rates, our technique detects more than 80% of the popular subpaths.

Computation times: One of our main motivations for designing the randomized algorithm was greater efficiency. We compare the running times of the deterministic and the randomized methods on an Intel(R) i5 PC at $3.20GHz$ and $7.7GB$ RAM, running scientific linux 6. For (k, l) queries on combinations of $k \in \{5, 10, 15, 20, 25, 20\}$ and $l \in \{10, 20, 25, 30\}$ hops, with $\delta = 0.15$, we run 10 queries per (k, l) combination and averaged the running times for both algorithms, computed as follows.

For the deterministic algorithm, for every (k, l) query we computed the time to extract all subpaths of length at least l, the time to count the supports of the subpaths and the time to remove redundant subpaths. For the randomized algorithm, for every (k, l) query we time the sampling of nodes for every level of the hierarchy and the detection of subpaths, starting from the top-most level of the hierarchy to the lowest (i.e, $p = 0.9$). As shown in Fig. 6(a) the deterministic approach requires orders of magnitude more computation time compared to the randomized approach. For datasets of 200 trajectories, our method answers a query in the order of seconds, whereas the deterministic approach requires several hours.

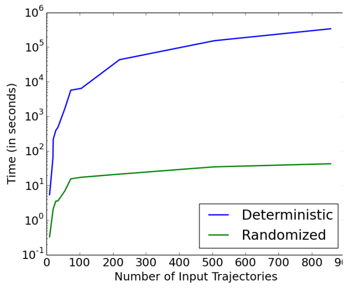
Fig. 2 shows a qualitative comparison of the popular paths found by (a) the deterministic approach and (b) our randomized algorithm on the dataset of the 150 day trips for popular paths of length at least 10 hops, support at least 40 and $\varepsilon = 0.1, \delta = 0.1$. The results of the randomized technique are almost indistinguishable from the deterministic method.

Communication cost in sensor networks: We considered communication and sensing at varying radii for each level of the hierarchy, starting from $600m$ radius at the top-most level of the hierarchy and decreasing by $50m$ at every other level, with the lowest level having a radius of $200m$. Using approximately 200 trajectories, we injected (k, l) queries to our sensor network, with varius combinations of $k = \{10, 20, 30, 40\}$ and $l = \{10, 20, 30, 40, 50, 75, 100\}$ hops (i.e., from 2 to 20 km).

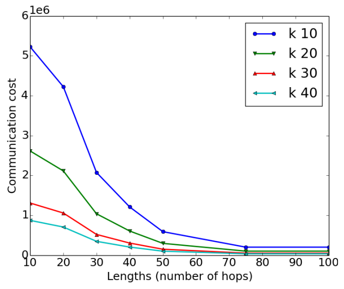
For every query, we computed the total communication cost required per level of the hierarchy, assuming greedy geographic routing for message exchange, and including the size of the sketches. In particular,for a single path, we estimated the communication cost assuming that the size of the message exchanged along the path is $s(u)/k$, which is the sufficient sample size for estimating instersections w.h.p, as shown in our analysis.Figure 6(b) shows that the cumulative communication cost drops fast for longer and popular paths.

Sparse sensor deployments. We conducted tests on trajectories that have at least two sensors in common to see how often they use the same intermediate path. Fig. 6(c) shows that the median accuracy is more than 90% at all scales including very long paths, meaning that these trajectories more or less stay together. Fig. 6(d) shows that the median Hausdorff distance of two such paths is also usually small compared to their lengths. Therefore, a sparse deployment of sensors, where we do not have reliable information about intermediate trajectories, can still produce useful clusters.

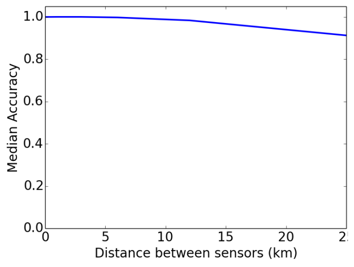
Predictability of trajectories. We hypothesize that along popular paths, users tend to move in predictable ways, and would like to evaluate the predictability at different levels of the sensing hierarchy. Given any two sensor nodes at a particular level of the hierarchy, if there are popular paths going through them, we compute the fraction of those continuing to their different neighbour (see Fig.7(a) for an example). We would like to know how many of these trajectories continue in the



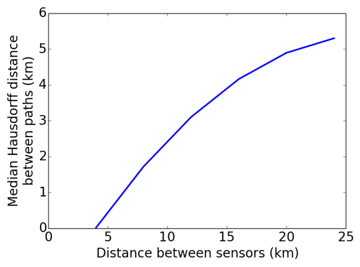
(a)



(b)



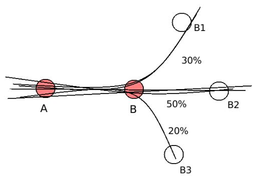
(c)



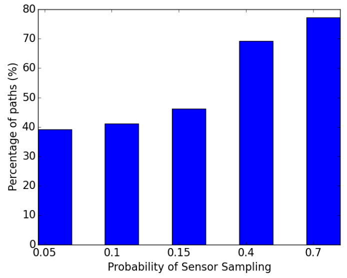
(d)

Fig. 6. (a) Given a dataset of hundreds of trajectories, the randomized approach can answer a popular path query in a few seconds. The deterministic approach can take in the order of hours. (b) Communication cost for different lengths of subpaths. (c) Trajectories that have two sensors in common usually follow the same path between these two sensors. (d) Median Hausdorff distance between two paths that have two sensors in common.

same direction as the most popular set of paths.



(a)



(b)

Fig. 7. (a) Of paths going through nodes A and B, different fractions go to $B1$, $B2$ and $B3$. (b) In clusters containing at least 10% of trajectories, the average fraction of trajectories that continue to the same next node in the given sampling.

Fig. 7(b) shows that in any popular cluster (with over 10% of trajectories in the system), a substantial fraction continues in the same direction. Even for sparse sampling rates of 0.05, corresponding to lengths of at least $1.5km$, 40% of trajectories continue along the popular direction. At denser samples predictability rises to 70%.

VI. Conclusion

We described a distributed scheme to find popular paths. The results show that it is possible to obtain fairly accurate results even with a small deployment of sensors and a small sample of users. Our randomized mechanism is naturally differentially private and adaptable to streaming data. The experiments show that users in popular paths tend to stay together even over large distances, and the clusters are good predictors of short term motion.

References

[1] https://www.openstreetmap.org.

[2] H. Alt and M. Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

[3] L. Bracciale, M. Bonola, P. Loreti, G. Bianchi, R. Amici, and A. Rabuffi. CRAWDAD data set roma/taxi (v. 2014-07-17). Downloaded from http://crawdad.org/roma/taxi/, July 2014.

[4] R. R. Brooks, P. Ramanathan, and A. M. Sayeed. Distributed target classification and tracking in sensor networks. *Proceedings of the IEEE*, 91(8):1163–1171, 2003.

[5] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, 2011.

[6] K. Buchin, M. Buchin, M. Van Kreveld, M. Löffler, R. I. Silveira, C. Wenk, and L. Wiratma. Median trajectories. *Algorithmica*, 66(3):595–614, 2013.

[7] L. Chi and K. Hui. Color set size problem with applications to string matching. In *Combinatorial Pattern Matching*, pages 230–243. Springer, 1992.

[8] J. Ding, C.-C. Ni, M. Zhou, and J. Gao. Minhash hierarchy for privacy preserving trajectory sensing and query. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, IPSN '17, pages 17–28, New York, NY, USA, 2017. ACM.

[9] J. Dong, C. Liu, and Z. Lin. Charging infrastructure planning for promoting battery electric vehicles: An activity-based approach using multiday travel data. *Transportation Research Part C: Emerging Technologies*, 38:44–55, 2014.

[10] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3&#8211;4):211–407, Aug. 2014.

[11] J. Gao, L. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Discrete mobile centers. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 188–196. ACM, 2001.

[12] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory pattern mining. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 330–339, New York, NY, USA, 2007. ACM.

[13] J. Leskovec, A. Rajaraman, and J. D. Ullman. *Mining of massive datasets*. Cambridge university press, 2014.

[14] Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 617–622. ACM, 2004.

[15] W. Luo, H. Tan, L. Chen, and L. 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. ACM, 2013.

[16] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[17] S. Nawaz and C. Mascolo. Mining users' significant driving routes with low-power sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 236–250. ACM, 2014.

[18] D. Pfoser, C. S. Jensen, Y. Theodoridis, et al. Novel approaches to the indexing of moving object trajectories. In *Proceedings of VLDB*, pages 395–406, 2000.

[19] R. Razavi and H. Claussen. Urban small cell deployments: Impact on the network energy consumption. In *Wireless Communications and Networking Conference Workshops (WCNCW), 2012 IEEE*, pages 47–52. IEEE, 2012.

[20] N. Shrivastava, R. Mudumbai, U. Madhow, and S. Suri. Target tracking with binary proximity sensors. *ACM Transactions on Sensor Networks (TOSN)*, 5(4):30, 2009.

[21] H. Tang, M. Kerber, Q. Huang, and L. Guibas. Locating lucrative passengers for taxicab drivers. In *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 504–507. ACM, 2013.

[22] F. Zhao. Large-scale transit network optimization by minimizing user cost and transfers. *Journal of Public Transportation*, 9(2):6, 2006.