In-Network Approximate and Efficient Spatiotemporal Range Queries on Moving Objects

Guang Yang
Imperial College London

Ahbirup Ghosh
University of Cambridge

Liang Liang
Imperial College London

Thomas Heinis
Imperial College London

ABSTRACT

Data aggregations enable privacy-aware data analytics for moving objects. A spatiotemporal range count query is a fundamental query that aggregates the count of objects in a given spatial region and a time interval. Existing works are designed for centralized systems, which lead to issues with extensive communication and the potential for data leaks. Current in-network systems suffer from the distinct count problem (counting the same objects multiple times) and the dead space problem (excessive intra-communication from ill-suited spatial subdivisions).

We propose a novel framework based on a planar graph representation for efficient privacy-aware in-network aggregate queries. Unlike conventional spatial decomposition methods, our framework uses sensor placement techniques to select sensors to reduce dead space. A submodular maximization-based method is introduced when the query distribution is known and a host of sampling methods are used when the query distribution is unknown or dynamic. We avoid double counting by tracking movements along the graph edges using discrete differential forms. We support queries with arbitrary temporal intervals with a constant-sized regression model that accelerates the query performance and reduces the storage size.

We evaluate our method on real-world mobility data, which yields us a relative error of at most 13.8% with 25.6% of sensors while achieving a speedup of 3.5×, 69.81% reduction in sensors accessed, and a storage reduction of 99.96% compared to finding the exact count.

1 INTRODUCTION

Data privacy in the moving object data analytics context has become increasingly crucial as large-scale online location-based services continue to grow [36, 37]. In data analytics on moving objects, systems face significant challenges due to the amount of data, the real-time query processing requirements, and data privacy concerns. Current privacy-aware data analytics systems aggregate data to implicitly store relevant information about the query to improve query processing by reducing the data size while ensuring anonymity. One fundamental query in moving object data analytics is the spatiotemporal range query, ubiquitous in location-based services [15, 17, 19, 36, 37]. In the privacy-aware analytics context, the spatiotemporal range query aggregates the count of the total number of distinct moving objects in a spatial region over a time interval. This calls for a framework that efficiently aggregates data and allows aggregated counts to deal with persistent moving object updates without storing identifiers.

Existing systems [9] preserve privacy by continuously aggregating data at a central server before answering queries. They benefit from the extensive mature centralized algorithms [5, 15, 19, 36] and ease of management from having data in a single

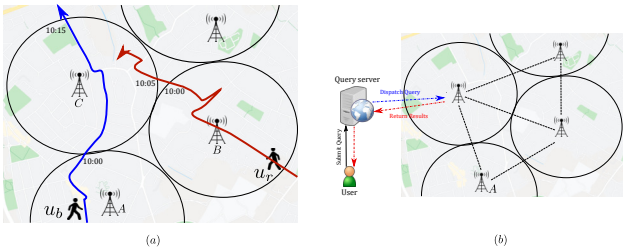


Figure 1: (a) A sensors network tracking two users u_b and u_r. (b) The associated communication infrastructure.

place. While centralized query processing systems are currently the norm, in-network systems are increasingly popular as they mitigate the problems of centralized systems, where they cannot efficiently deal with high rates of updates for real-time applications and often face communication bottlenecks. Extensive continuous communication is inevitable since updates must be sent to the central server for aggregation. Additionally, privacy concern arises with centralized systems since all data is stored in the same location before aggregation. In-network systems, on the other hand, naturally maintain data locality.

In-network systems aggregate updates on relevant (close-by) sensors and only communicate to sensors that are involved in the spatial query range. We depict an example query in Figure 1 for a cell tower load balancing application, where the number of users in each cell tower range needs to be monitored at different time intervals. An example spatiotemporal aggregate range query for sensing region C in Figure 1a between time intervals $10:05$ to $10:10$ returns the number of distinct objects in C to be 2. Privacy is preserved as all updates within C are local without needing to know the full history of the moving object. Hence, no single parties have access to the full mobility pattern at any time. Queries are dispatched to the relevant sensors (only to C without needing to communicate to B or A) shown in Figure 1b. Communication to a centralized query dispatcher is needed only when querying, which is much more efficient than the continuous sync required for centralized systems.

However, adapting in-network systems for moving object aggregation queries is challenging as these queries require handling frequent updates of moving objects, which they suffer due to the double counting and dead space problems. Specifically, the *double counting problem* occurs when we do not store moving objects' identifiers, resulting in distinct objects being counted multiple times during updates [15, 36]. This is challenging for privacy-aware queries as we cannot store the object identifiers. The *dead space problem* generates excessive communication between sensors when ill-suited spatial partitions generate excessive dead space in the partitions. Unlike centralized systems, in-network systems are unaware of dead spaces. Therefore, they must flood all sensors within the query region - a challenge for in-network query processing, increasing the communication cost (proportional to the area of the query region [34]).

In this paper, we, therefore, develop a novel framework for efficient privacy-aware in-network aggregate queries. Our main

design idea is to build a framework based on a non-axis-aligned spatial representation of the sensor network that can efficiently support moving object updates. We introduce the idea of storing both the incoming and outgoing counts at the edges of spatial partitions during updates, which solves the double counting problem without unique identifiers. A planar graph representation configures the spatial partitions (based on sensor and edge locations) to minimize the dead spaces while preserving query accuracy. To achieve efficient query processing (in terms of query efficiency and in-network sensor infrastructure cost), we open up an alternative way of thinking from in-network sensor placement research: *can we configure the sensor locations in such a way that will make the query processing more efficient?*

We propose a planar graph representation of the in-network system that decomposes the spatial domain based on sensor placement techniques. The planar graph minimizes dead space by selecting sensors based on a host of sampling methods when the query distribution is unknown and dynamic. When the query distribution is known, a modified submodular maximization algorithm [27] is used. The sensor configuration can aid sensor deployment (to achieve the best cost-saving and query accuracy) or used to select a subset of sensors to activate at particular times to save energy for a pre-existing set of sensors. An associated query algorithm based on differential 1-form only requires accessing the sensors on the perimeter of the query region to reduce communication. Specifically, counts are stored on the edges of the planar graph, and queries are answered by integrating the counts along the edges of the query region (defined as faces of the planar graph). To solve the double counting problem, we store two counts (incoming and outgoing) on each directed edge in the planar graph. When answering queries, the incoming and outgoing counts will ensure objects that exist and enter disjoint partitions (or faces in the graph) multiple times will cancel out. To preserve privacy in the temporal dimension, we take inspiration from learned indexing [41] to infer the count at different timestamps using constant-sized machine-learning models. This also increases query performance and reduces storage size.
Our contributions are summarized as follows:

- We propose an in-network framework with sensor placement techniques to efficiently answer queries and aggregate data within the sensor network. Data is processed and stored at each sensor, with no third parties accessing full mobility patterns at any time. Our method seeks to bridge the gap between privacy-aware moving object analytics and efficient in-network processing.
- We present a solution to the double counting problem in an in-network setting by modifying the query algorithm to account for incoming and outgoing counts.
- We support queries with arbitrary temporal intervals. To accelerate the query process, we propose a constant-size machine learning model that efficiently predicts the count using inference. This significantly reduces the storage size in each sensor and improves performance.
- We evaluate our framework and show it can achieve a high accuracy level while significantly reducing the storage cost, lookup time, and communication cost.

2 RELATED WORKS

To the best of our knowledge, no existing methods bridge the gap between privacy-aware data aggregation and sensor placement. Both have been independently studied in the literature. In the following section, we review all relevant works, including, spatiotemporal range queries, privacy-preserving queries, in-network query processing, and sensor placements.

2.1 Spatiotemporal Range Queries

We first discuss range queries for static objects before discussing moving objects queries. Range queries on static objects are efficiently answered with hierarchical data structures, such as R-trees [6], *kd*-trees [2] and QuadTrees [28, 40], using divide-and-conquer. They are extended to querying moving objects by supporting updates on objects, which includes lazy updates [1], incremental updates [7], and selective updates [39]. However, these methods are not privacy-preserving and require the entire history of the moving objects.

2.2 Privacy-Preserving Queries

Increasing concerns over data privacy in recent years [12, 19, 38] have led to growth in privacy-preserving querying methods. Data aggregation is one of the most prominent methods for desensitizing data and reducing storage requirements. Existing works on moving objects queries include distributed Euler-histograms [15, 19], tree structures [36], and holistic aggregate functions [5]. Recent works seek to perturb the moving object data to preserve privacy [11] or improve the accuracy of existing approximate range count queries, specifically, with a private BBD-Tree alongside a notion of fuzziness to the query range [23].

However, existing methods are designed for centralized servers where the full history of the moving objects must be kept before aggregation. We argue that in-network systems can better preserve privacy by keeping the data local to each sensor, which prevents any party from viewing the full history at any time.

Additionally, learned indexes [13, 16, 25, 41], which exploit patterns in the underlying data distribution, have recently emerged as a compact alternative to traditional algorithmic indexes such as the B+Tree. They model the Cumulative Distribution Function (CDF) of the indexed keys to reduce the search bound of the index. This inspired us to explore using a model to approximate the count across timestamps, which no existing methods have tried. Our approach takes inspiration from FLIRT [41] to further preserve privacy and reduce storage requirements.

2.3 In-Network Query Processing

In-network query processing seeks to answer queries within a physical sensor network (similar to edge computing), where the low-resource sensors track the movement of moving objects. Each sensor tracks the movement in spatial subdivisions. Traditionally, spatial subdivisions can be divided using axis-aligned spatial indexes, including, Grids [15, 19, 29], R-trees [6, 36], *kd*-trees [2, 7], and QuadTrees [28, 40]. However, these methods generate much dead space. Sensors are unaware of whether the area is a dead space. Hence, the in-network system must flood all nodes in the query range to answer the query, which results in a communication cost proportional to the area of the query region [18]. [34] improves the communication cost by using planar graphs and differential 1-forms to subdivide the space. They support spatial range queries but do not answer spatiotemporal queries. We take inspiration from [34]. While they focus on the network performance of planar graphs and differential forms, we extend this idea to spatiotemporal queries with a subset of sensors and analyze the performance based on query performance.

2.4 Sensor Placement

Sensor placement has been studied extensively as a standalone problem. For example, [3, 30] optimizes sensor placements for traffic flows. [26] proposed submodular maximization for selecting sensors for contamination detection.

Recently, sensor placements have begun to incorporate sampling. For example, [22] approximates a query by collecting samples in each sensor, and [8] reduces the number of sensors required to track movement between spatial subdivisions. However, no existing methods focus on the network performance (reducing sensors and communication costs), nor do they discuss where to select sensors or how to track and aggregate movements for queries within the sensor network.

3 BACKGROUND

In this section, we explain the problem setting and clarify the reasoning for our design. We then formulate the query and introduce technical details about concepts and mathematical terminologies used in the rest of the paper.

3.1 Problem Setting

In-network systems store counts locally to the sensor responsible for the detection. Sensors can communicate with each other (e.g., mesh network), and some sensors act as communication nodes to manage and aggregate counts from nearby sensors [43].

They improve privacy and reduce communication under continuous updates over centralized systems. Privacy is guaranteed as data is kept local to relevant sensors, with no parties obtaining the full mobility pattern of moving objects at any time. As for communication, centralized systems continuously sync with all the sensors for up-to-date counts, where they need to manage a large number of simultaneous connections from each sensor. This may be superior when the frequency of queries is higher than updates. However, we argue that in most practical cases, the update frequency is much higher than the query frequency, which in-networks systems have lower communication costs and apply to all network scenarios (e.g., increase bandwidth from 6G). Furthermore, substantial network bandwidth and power are needed for centralized systems if sensors are far from the servers (e.g., high-power radios for long-distance data transmission, which can quickly drain battery-powered sensors).

However, we must address the dead space and distinct count problems to support privacy-aware aggregate count query processing for in-network systems. To create an efficient in-network query framework, we explore the idea of a planar graph with discrete differential forms and sensor placement techniques to decompose the spatial domain. Aside from solving these problems, we must also ensure efficient query performance (accuracy & time) and low sensor infrastructure cost (setup & operating cost). We now explain the two problems and justify the reasoning and advantages of the design decisions.

3.1.1 Dead Space Problem. We first expand on the dead space problem described in the introduction. The *dead space problem* occurs when ill-suited spatial partitions generate unnecessary communication. The main rationale is that existing axis-aligned spatial partitioning techniques are designed for centralized systems. Therefore, the partition considers the spatial distribution of the entire data rather than the distribution of sensors.

For example, a sensor network is deployed for traffic monitoring in a city with curved roads (exemplary of real-world cities, except Manhattan). Suppose we apply axis-aligned partitioning based on the data distribution. In that case, we may place sensors or select sensors (if a pre-existing set of sensors exists) at regular axis-aligned intervals, using R-trees [6, 36], *kd*-trees [2, 7], or QuadTrees [40]. This means we have sensors on roads and in areas between roads (where no vehicle is present). Processing queries would require the system to collect data from all the sensors within the spatial and temporal range, leading to unnecessary communication and computation. Additionally, this leads to a waste of resources as the sensors at the dead spaces consume power without any contribution.

3.1.2 Double Counting Problem. The *double counting problem* occurs when we cannot distinguish unique objects without identifiers and count an object multiple times when it exits and re-enters the sensing region during updates. Let's consider an example of a sensor network that monitors traffic flow on a highway. The sensors are placed at the exit and entry ramps. A vehicle will be counted every time it enters the highway and exits at the next ramp before re-entering the highway, resulting in an overestimation of the number of distinct vehicles.

3.1.3 Our approach. We avoid ill-suited spatial partitions by introducing a planar graph representation of the sensor distribution. The nodes of the planar graph represent the sensors, while the faces represent the coverage of the sensors. The edges of the planar graph are communication paths between nodes and represent the border of the sensor coverage. The sensors at the nodes bordering an edge detect moving object updates when they move from one face to another. To prevent the double counting problem, we introduce two counts (incoming & outgoing) at the edges to record the direction of travel. The exact techniques for finding the direction of moving objects in a sensor network are not the interest of this paper; readers can refer to [21]. To answer queries, we represent the query region as faces of the planar graph and find the count of the nodes at the border of these faces. The planar graph allows us to route through the graph's edges to answer queries efficiently. An associated query algorithm is introduced to take into account the two counts when routing through the edges during aggregation.

To achieve efficient query processing, we need to consider the sensor configuration. We do not need every sensor on the graphs to communicate with the remote query dispatching server. Instead, fewer communication sensors can improve the overall efficiency at a loss of query accuracy by saving operating costs and communication costs (which also decrease query response time with less routing). The details of how communication sensors collaborate to track objects are not the focus of this paper. Readers can refer to [42]. The challenge here is minimizing the communication sensors while preserving query accuracy.

We seek to tackle the sensor configuration challenge from the perspective of an in-network sensor placement problem. We propose a sensor sampling algorithm based on submodular maximization, which accounts for the query distribution to maximize the sensor coverage for a given number of sensors. We introduce a host of sampling methods to configure the sensors to support applications when the query distribution is unknown and dynamic. Note that our main focus is on increasing communication and computation efficiency and does not focus on privacy requirements. Before presenting the algorithms, we provide technical background about the planar graph representation in the following sections.

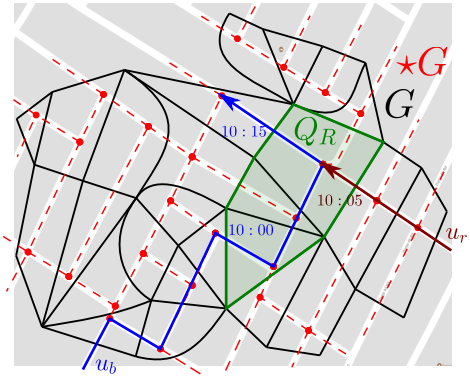


Figure 2: Example of road network as the mobility graph $\star G$ and dual graph G as the sensing graph. Two objects, u_b and u_r, are tracked within the region. Q_R is the query region shown in green. u_b enters the Q_R at 10:00 and leaves at 10:15. u_r enters the Q_R at 10:05 and does not leave.

3.2 Planar Graph Representation

We first introduce the mobility and sensor domains used for tracking moving objects.

3.2.1 Mobility Domain. The mobility domain is where moving objects are located and represents physical spatial regions. E.g., the roads of the city form the mobility domain, where vehicles move along the roads and movements are restricted by the layout of the road. We usually represent this as a network graph. E.g., in a road network, the roads represent edges, and junctions represent nodes.

3.2.2 Sensor Domain. The sensor domain represents the placement of sensors in a mobility domain used to track movements. In the road network example, we can place sensors at strategic points to track movement where the sensors' network connectivity graph represents the sensor domain.

3.2.3 Planar Graph Representation of Sensing and Mobility Domains. Our framework abstracts the mobility and sensor domains as planar graphs for their properties and advantages. This is widely accepted as we can easily convert mobility graphs through spatial decomposition and convert overlapping sensing regions to non-overlapping "virtual" sensing regions.

The planar graph representation is shown in Figure 2. We denote the planar mobility graph as $\star G = (\star V, \star E)$. Using the road network example, $\star V$ are junctions, and $\star E$ are roads. The sensor graph is denoted $G = (V, E)$, where V are the sensors and E are communication links between the sensors. G is constructed to be the dual graph of $\star G$, which means the V represents a face in $\star G$. This allows us to use properties of the dual graph, such as vertex-edge duality.

We represent the movements using discrete differential 1-forms on the edges of E. To answer a query, we convert the spatial range to be a union of faces in the sensor graph G, shown as Q_R in Figure 2. The count is found by routing the sensors around the perimeter of Q_R. Before explaining the exact algorithms, we formulate the query in respect of the planar graph representation and introduce differential forms.

3.3 Query Formulation

As mentioned above, we define spatiotemporal range count queries in terms of planar graphs. Spatiotemporal queries have both a spatial and time component. The spatial region is Q_R, and the time interval is $[t_1, t_2]$. Here, we focus on two types of queries:

(1) *Static object count query.* Counts the objects that enter Q_R before time t_1 and leave after time t_2 (and does not temporarily leave Q_R). This counts the number of objects within the region during the time interval. E.g., in Figure 2, the count in Q_R between [10:05,10:10] is 2. This query strictly generalizes the spatial range count query, as studied in [34]. To answer the spatial case, we can set the timestamp t_1 and t_2 to be very close.

(2) *Transient object count query.* Counts the objects that either enter Q_R before t_1 and leave during $[t_1, t_2]$; or enter Q_R between $[t_1, t_2]$ and leave after t_2. E.g., in Figure 2, the count in Q_R between [10:05,10:10] is one since u_b enters before 10:05 and leaves after 10:10. If we consider the time interval [10:00,10:03], u_r is not in the count. This query provides the net change of objects during the time interval, which is useful for applications such as traffic flow estimation [35], where the transient count is used to calculate the velocity of vehicles in a region (net count/time).

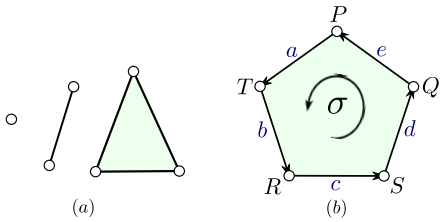


Figure 3: (a) 0,1,2 cells (b) An oriented face σ

3.4 Discrete Differential Forms

Before discussing how range queries are answered, we introduce relevant concepts about cell complexes and discrete differential one-forms. In algebraic topology, points/vertices/nodes are considered 0-dimensional cells (0-cells), lines/edges are 1-cells, and faces are 2-cells (example in Figure 3a). A cell complex is a collection of cells where it's dimension is determined by the highest-dimension cell. E.g., the planar graphs $\star G$ & G in Figure 2 are considered 2-cell complexes.

The orientation of a k-cell is inherited from its $(k+1)$-cell. In Figure 3b, the face σ, orientated counter-clockwise, imposes the orientation of directed edge e to be $[Q, P]$. The same edge e in the opposite direction is $-e$, which in our example will be $[P, Q]$. We orient the faces counter-clockwise for our convention [24].

A k dimensional chain C is a linear combination of k-cells c_i: $C = \sum \lambda_i c_i$, where λ_i are binary weights for the i-th cell in the chain. ∂ is the boundary operator for finding the linear combination along the perimeter of a face or the 1-dimensional chain of a 2-cell. In Figure 3b, the perimeter is formed of edges a, b, c, d, e. The 1-dimensional chain of the face σ assuming $\lambda_i = 1$ is: $\partial\sigma = a + b + c + d + e$.

Discrete differential 1-forms (denoted as differential forms) are functions on 1-cells or edges. We use differential forms to keep track of the number of moving objects on each edge. Formally, a differential form is a function: $\xi : E \rightarrow \mathbb{R}$. It has a property where $\xi(-e) = -\xi(e)$, and we denote the differential form at time t as ξ_t. We find the number of moving objects contained in each face by integrating the differential forms along the 1-chain C: $\xi(C) = \sum_{e \in C} \xi(e)$.

Figure 4: Sampling communication sensor location using road network of Beijing. (a) uniform sampling. (b) systematic sampling. (c) stratified sampling. (d) kd-tree sampling. (e) QuadTree sampling. (f) Regions (red) selected by submodular maximization given a set of queries (queries are not shown).

4 ALGORITHMS

In this section, we provide an overview, introduce how the sensing graph is generated and describe the query answer process before breaking down each step in detail.

4.1 Framework Overview

We show how to construct a planar graph representation when the sensor and mobility graphs are unknown for completeness in Section 4.2. This is useful for planning sensor development or using the sensor graph as a spatial decomposition technique.

The first step in our framework is to generate a sensor configuration from an existing planar graph representation of the sensing and mobility domains (generated from Section 4.2, or from pre-existing networks). We propose two approaches for selecting the sensors based on the amount of prior knowledge about the queries. If the query distribution is unknown or dynamic, we elect a subset of sensors using sampling algorithms. In Section 4.3, we showcase multiple sampling algorithms for different use cases. When the query distribution is known, we elect the sensors based on a submodular maximization, where we maximize the sensor coverage given a budget. We present the algorithm in Section 4.4. The selected sensors will be the communication sensors that communicate with the query server. They also manage and aggregate counts from local sensors within their sensing region. This is often done in practice to limit long-distance communication (sensor-to-server) to a few sensors. An example can be found in [42].

After generating a sensor configuration, we construct a new "sampled" planar graph representation of the sensing domain and generate the edges between the nodes in Section 4.5. For the query oblivious case, we connect the nodes either with a triangulation-based or k-NN-based algorithm. The edges are then materialized in the network by routing through the shortest paths between selected nodes in the original sensor graph.

We answer queries on the sampled sensor graph with the following steps: Section 4.6 explains how queries are dispatched to relevant node/s and the steps to find all nodes that boundary the spatial query region. We then filter for the temporal range at boundary nodes before computing the count considering incoming and outgoing objects in Section 4.7-4.8. These sections also explain how the sensor graph handles moving object updates. As mentioned in Section 3.1.3, our goal is to improve query efficiency. Therefore, we refrain from detailed privacy analysis, but one can extend our method using methods from [20] to include privacy guarantees.

4.2 Constructing Planar Graphs

We describe the process of generating planar mobility graphs from a spatial map or spatial network graph. The senor graph is the dual graph of the mobility graph.

As an example, we generate a road network graph from a standard map obtained from mapping services. The first step is to filter our non-relevant nodes and edges. E.g., all non-vehicle nodes and edges such as walking paths and train tracks. We then generate the planarized graph by removing intersections from underpasses and flyovers by inserting nodes at the intersections.

In the example above, moving objects travel along defined paths, which constrains their movement to predefined paths. However, there are many cases where moving objects can freely roam around the spatial region (e.g. air and sea transportation). In this case, we may want to generate "virtual" paths from historical

mobility traces. E.g., we generate paths from historical data where there is considerable traffic. However, estimating which paths to generate is not trivial; we will leave this for future work.

4.3 Query Oblivious Sensor Selection

In this section, we describe the sensor selection algorithms when we have no prior knowledge about the queries or when the query distribution is unknown. Formally, there are $|V|$ possible locations to place sensors, and the system has a budget for placing m sensors that optimizes query performance. We propose to use sampling to select a subset of nodes $\tilde{V} \subset V$ from the sensing graph G to generate a sampled graph $\tilde{G}$. We now present sampling approaches aimed at various applications.

Uniform random sampling. we selects m nodes from V with equal probability. Figure 4a shows sensors selected using uniform sampling. This is the simplest type of sampling and has a bias towards denser regions. Applications that query denser areas are suitable for this method. E.g., traffic monitoring applications may prefer more sensors in densely populated areas.

Systematic sampling. We impose a virtual grid on the mobility domain and select a node from each grid cell. The chosen node is either closest to the cell center or is randomly selected in each cell. An example is shown in Figure 4b. Systematic sampling ensures a uniform distribution of nodes across the mobility graph. This is suitable for applications where queries are uniformly distributed.

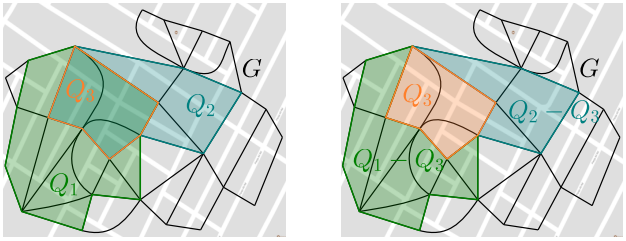
Stratified sampling. We partition the mobility graph using strata (e.g., each district or zip-code region). We uniformly sample a set of nodes from each stratum. Figure 4c shows an example where strata are districts in Beijing. A function defines the number of samples in each stratum. For simplicity, we use a function that defines the number of samples based on the area of each stratum. This function can be easily extended to incorporate other factors, such as the importance of each stratum. This type of sampling is useful when queries are defined based on strata.

Hierarchical space partition-based sampling. We build a QuadTree or a kd-tree for the nodes V in graph G by recursively partitioning the space until the leaf level has m nodes. Like systematic sampling, we select nodes closest to the center or randomly select nodes in each leaf node to form the sampled nodes. Figure 4d,e show random kd-tree sampling and random QuadTree sampling, respectively. This method takes benefits from both uniform and systematic sampling.

We assume the cost of selecting any sensor to be uniform for the sampling algorithms described above. However, we can include non-uniformity by using different weights for each node. For example, if we were to make our sampling methods query adaptive. We can use the number of times each node appeared in previous queries as the weight.

4.4 Query Adaptive Sensor Selection

This section considers the case when the expected query regions are known a priori (e.g., from domain experts or predictions based on historical data). Similar to Section 4.3, we select m sensors from $|V|$ from a subset of nodes $\tilde{V} \subset V \in G$ for generating the sampled graph $\tilde{G}$. We formulated the problem of sensor placement with combinatorial optimization and used submodular maximization [27] to approximate the solution. We start with how to use submodular maximization for sensor placement and adapt it to our problem.



(a) (b)

Figure 5: Example of two overlapping query regions. (a) Q_1 and Q_2 are two overlapping query regions to form $Q_3 = Q_1 \cap Q_2$. (b) Disjoint query regions $Q_1 - Q_3$, $Q_2 - Q_3$ and Q_3.

4.4.1 Submodular maximization for selecting sensors. We first define the sensor placement problem given expected query regions. Consider a classical setting for studying sensor placements: there are n possible locations ($\mathcal{V}$) to place the sensors where each sensor has a defined region it can sense. Given that we have the budget to deploy m sensors, how do we choose the locations to maximize the total coverage area? This is an NP-hard [26] problem because one needs to test all combinations of $\mathcal{V}$ to choose the best m locations, which needs exponential time.

Several approximation methods have been proposed in the literature to tackle this general line of problems. We consider submodular maximization as it matches our requirements. Let us denote the set of m locations to be L. For simplicity, suppose each sensor covers a unit circle on a plane. Here, the utility (area being covered by the sensors) is submodular or has diminishing return property. This means that when sensors are added sequentially, the marginal area covered by each new sensor will be smaller with more sensors. This is because the coverage area of the new sensor may overlap with previous sensors.

We now define sensor placement in terms of a submodular optimization problem. Consider a set of sensors S, and the coverage of the set of sensors is $f(S)$. The marginal coverage of a new sensor x, where $(x \notin S)$, is $\Delta_S f(x) = f(S \cup \{x\}) - f(S)$ with respect to S. For two sets of sensors A and B, where $A \subseteq B$, the marginal coverage is:

$$\Delta_A(x) \geq \Delta_B(x), or$$
$$f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$$

The function is monotone if $f(A) \leq f(B)$. In order to maximize the sensor coverage, we can maximize the following objective function. Here we assume the cost of selecting any sensor to be uniform.

$$L = \text{argmax}_{L \subseteq \mathcal{V}} f(L) \quad (1)$$
$$\text{where, } |L| \leq m$$

This maximization admits an iterative greedy approximate solution – start with an empty set and at iteration i choose the sensor with the maximum marginal utility until m sensors are selected. We can break the ties with random selection. We can simplify the optimization with a $(1 - \frac{1}{e})$ approximation [31].

$$S_{i+1} = S_i \cup \text{argmax}_{v \in \mathcal{V} - S_i \text{ and } |S_i| < m} \Delta_{S_i}(v) \quad (2)$$

If the cost of the elements is not uniform, we need to add constraints to this problem.

$$L = \text{argmax}_{L \subseteq \mathcal{L}} f(L) \quad (3)$$
$$\text{where, } \sum_{v \in L} c(v) \leq B$$

Where u is a sensor, $c(u)$ is the cost function of selecting sensor u, and B is the optimization budget. In this case, the greedy algorithm starts with an empty set, and at each step i, it selects the item as follows.

$$S_{i+1} = S_i \cup \text{argmax}_{v \in \mathcal{V} - S_i \text{ and } c(v) \leq B - c(S_i)} \frac{\Delta_{S_i}(v)}{c(v)} \quad (4)$$

Furthermore, Equation 4 and 2 can produce a $\frac{1}{2}(1 - 1/e)$ approximation according to the findings in [27]. We will adapt the approximation for choosing sensors given historical query data.

4.4.2 Submodular utility function given query regions. To select query regions to monitor, let us consider a set of historical query regions, $\bar{\mathcal{Q}}_R = \{\bar{Q}_R\}$. Now our objective is to select the query regions to monitor by placing the sensors at the boundaries of these regions.

This problem differs from the ones described above as the sensors monitor the boundaries of the regions. Following is our insight. Let us consider two regions in Figure 5a, Q_1 and Q_2, that overlap to form an overlapping region $Q_3 = Q_1 \cap Q_2$. Suppose we choose to monitor the edges in ∂Q_1. Then our system cannot answer a query on Q_2. However, if we also monitor the edges in ∂Q_3, we could answer the query on both Q_1 and Q_2 (with a lower bound). This adds a small additional cost since $|\partial Q_3 \cap \partial Q_1| > 0$.

Thus our strategy is to first maximally partition $\bar{\mathcal{Q}}_R$ so that none of the resulting regions, $\mathcal{Q}_R = \{Q_R\}$, overlap and consider that to be the set of regions to be selected. For our example, Figure 5b shows the resulting query regions. Suppose σ denotes a maximal cell complex fully contained by at least one query region in $\mathcal{Q}_R$. Then the tracking sensors need to be placed on $\partial\sigma$. Assuming the cost of each edge is uniform, the cost function $c(\sigma)$ and utility function $f(\sigma)$ for selecting the cell complex σ is:

$$c(\sigma) = |\partial\sigma| \quad (5)$$

$$f(\sigma) = \sum_{\forall Q_R \text{ containing } \sigma} \frac{\omega(\sigma)}{\omega(Q_R)} \quad (6)$$

Where, $\omega(\sigma)$ denotes the number of cells within σ. We observe that the utility of the region σ increases as the region covers a larger area w.r.t a query region in $\mathcal{Q}_R$. We can replace $\omega(\sigma)$ with a more general notion representing cells with different weights. $c(\sigma)$ is the number of boundary edges for simplicity. We can use a more complex function to incorporate the actual cost as long as it monotonically increases with edges. E.g., increasing the size of the area should increase the cost.

The utility function $f(\sigma)$ is always submodular and monotone because (1) the ratio $\frac{\omega(S)}{\omega(Q_R)}$ never decreases with more sensors added to S. (2) $\omega(.)$ is submodular since we are dealing with the area. To select the subset of nodes $\tilde{V}$, we first select the sensing region σ that maximizes the utility function $f(\sigma)$ until the budget m (Figure 4f).

Equation 6 is the utility function used to select sensing regions for lower bound approximations (e.g., the resulting count will be less or equal to the actual count.) For upper bound approximations, we can modify the σ to denote a cell complex that contains one query region in $\mathcal{Q}_R$ (details in Section 4.6). An alternative formulation for this problem considers partially covering $\mathcal{V}$ regions. However, the naive solution to this formulation has high computational complexity (exponential number of possible sensing regions) and is not considered in our current scope.

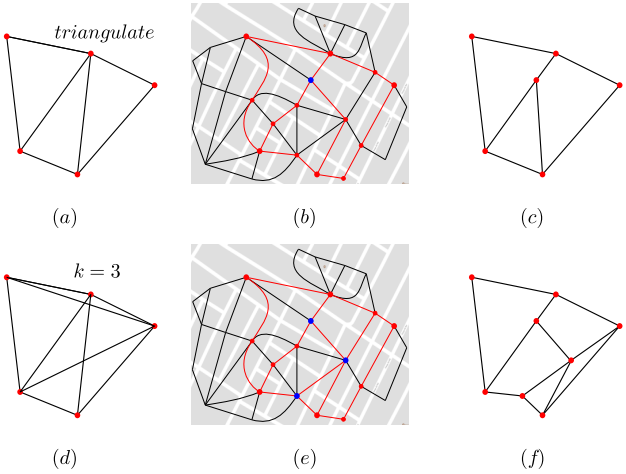


Figure 6: Generating edges from sampled nodes using triangulation and k-NN (k=3). (a) Edges after triangulating $\tilde{V}$. (d) Edges after finding k-nearest neighbor for each node v in $\tilde{V}$. (b,e) Replacing edges within the shortest path in G. The blue nodes are intersection nodes added to ensure $\tilde{G}$ is planar. (c,f) $\tilde{G}$ after simplification.

4.5 Generating Sampled Graph

Once the subset of nodes $\tilde{V}$ is found, we generate the sampled graph by connecting the nodes with edges. For the query oblivious selection method, we first need to generate the edges between the nodes, either using triangulation or k-NN.

For the same set of $\tilde{V}$, Figure 6a shows how triangulated edges are formed from $\tilde{V}$ (red), and Figure 6d shows edges formed by k-NN ($k = 3$). For k-NN , we find the k-nearest neighbor nodes $\tilde{u} \in \mathcal{N}_k(\tilde{v})$ from $\tilde{V}$ for each node $\tilde{v} \in \tilde{V}$. k-NN typically generates more faces, each with a smaller area. This benefit queries with smaller spatial regions as they are more likely to contain one of these faces. Note that $\tilde{G}$ becomes maximal when $k = m$.

Once the edges are formed between nodes, we must map the edges to network paths to avoid any intersections (which makes the graph non-planar). This is formalized by routing the shortest paths between nodes in the original sensing graph G. Figure 6b,e shows the shortest paths, where intersections are labeled as blue nodes. We add the intersections to the sampled graph for virtual representation proposes (they do not have to be communication sensors). The resultant sampled graph $\tilde{G}$ is shown in Figure 6c,f.

For the query adaptive selection method, the edges are formed from the boundary edges of selected regions. We only have to map the edges to network paths for the query adaptive selection method. For example, based on submodular maximization, we can use a trivial case where all regions are considered triangles. The first region contains three nodes which will form a triangle. We then add another node to form another triangle (that maximizes the area) with two nodes from the existing triangle. The process continues until we have m nodes. Therefore, the edges are generated during this process, and we only need to map the edges to the network paths to avoid intersections.

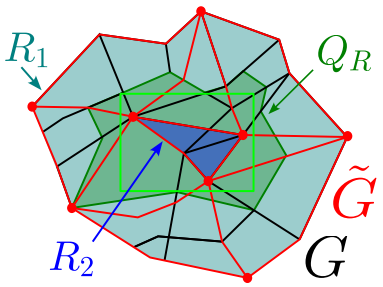


Figure 7: The query region Q_R (green) defines the rectangular query region as faces in the sensing graph G. R_1 and R_2 are the upper and lower approximation of Q_R.

4.6 Dispatching Queries

The query dispatches from a remote query server to the communication nodes in the sampled graph $\tilde{G}$. Before explaining which nodes to dispatch the queries to, we first introduce how the graphs define the query regions. The remote query server knows the structure of sensing graph G and sampled graph $\tilde{G}$ but does not know the exact counts. The query region is defined as the union faces of the sensing graph G, which supports the query region of any arbitrary shape. E.g., we enclose Q_R to a rectangular query region in Figure 7.

Since $\tilde{G}$ is a subgraph of G, the queries answered in $\tilde{G}$ will be approximate as we cannot guarantee Q_R to be present in $\tilde{G}$. There are two approximations: lower-bound approximation, which is the maximal region enclosed by the query region Q_R (shown as R_2), and upper-bound approximation, which is the minimal region containing Q_R (shown as R_1). Depending on the type of approximation (which can be specified by the user), we define the query region as the union of faces in $\tilde{G}$ that are bounded by (lower-bound) or bounds (upper-bound) the query region.

Once the query region is defined in $\tilde{G}$, there are two ways to communicate with the sensors depending on communication cost. The first method involves communicating with all nodes on the perimeter of the query region and aggregating the final count in the query server. The second method involves communicating with one node on the perimeter of the query region. The node then finds all other nodes on the perimeter via node traversal (e.g., flooding, routing) and aggregates the count back to the node before returning the result to the query server. The choice of method depends on the actual cost in the network and is not the focus of this paper.

4.7 Querying the Sampled Graph

4.7.1 Snapshot. We first present how to update the count of moving objects with differential forms without timestamps (or a snapshot in time), before extending it to include timestamps. The idea is to have a differential form along each edge $\tilde{e}$ in $\tilde{G}$ to keep track of the count (physically, the count is stored at the nodes, and the edge counts act as an abstraction). Here we can take advantage of the vertex-edge duality, where the object moves along the edge $\star\tilde{e}$ in $\star\tilde{G}$ will cross an edge $\tilde{e}$ in $\tilde{G}$. Therefore the corresponding differential form $\xi(\tilde{e})$ will be updated.

We define two tracking forms for each directed edge: $\xi_t^+(.)$ and $\xi_t^-(.)$. An example is shown in Figure 8b where object T moves from source (σ) to target (τ). The shared edge crossed by T is seen as c for $\partial\sigma$ and $-c$ for $\partial\tau$. Therefore, T leaves σ via directed edge c in the perspective of $\partial\sigma$. We use $\xi_t^-(c)$ to track this type

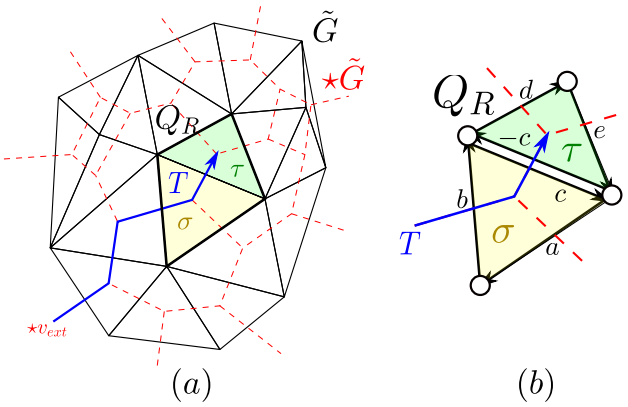


Figure 8: Differential form example. (a) The road network is the dual graph (dotted line), with the sensor network as the primal graph (solid line). $\star v_{ext}$ is the infinity node that acts as a source and sinks for objects entering and leaving the mobility graph. (b) Target T moving from face σ to τ. The query region Q_R is the union of faces σ and τ ($Q_R = \sigma \cup \tau$).

of movement. Similarly, we use $\xi_t^+(-c)$ to track T as it enters τ via $-c$ in the perspective of $\partial\tau$. We use + to denote the movement from one face to another:

$$\xi_t^+(-c) = \xi_t^+(-c) + 1 \text{ and } \xi_t^-(c) = \xi_t^-(c) + 1 \tag{7}$$

Theorem 4.1. *If the differential forms are updated as per Equation 7, then at the current time t, the number of objects inside a cell complex Q_R is given by $\sum_{e\in\partial Q} \xi_t^+(e) - \xi_t^-(e)$.*

Proof sketch. In Figure 8, τ is initially empty ($\partial(\tau) = 0$). Target T enters τ at time t from σ and crosses at edge c. According to Equation 7, $\xi_t^-(c) = 1$ since T is moving out of σ. Similarly, $\xi_t^+(-c) = 1$ as T moves into τ. We find the count by integrating the differential forms along the perimeter of the face. The count at time t is $\xi_t(\partial(\tau)) = (\xi_t^+(d) - \xi_t^-(d)) + (\xi_t^+(e) - \xi_t^-(e)) + (\xi_t^+(-c) - \xi_t^-(-c)) = \xi_t^+(-c) = 1$. Similarly, the count in σ is $\xi_t(\partial(\sigma)) = \xi_t^+(b) - \xi_t^-(c) = 1 - 1 = 0$.

4.7.2 Arbitrary Time Intervals. We extend the tracking form to contain a sequence of timestamps for each crossing event to answer spatiotemporal range queries. We use $\oplus$ to represent appending to a sequence (E.g., $\vec{x} \oplus y$ means appending y to a sequence $\vec{x}$). Thus when an object crosses an edge c in Figure 8, we update the tracking forms as follows.

$$\gamma_t^+(-c) = \gamma_{t-1}^+(-c) \oplus t \text{ and } \gamma_t^-(c) = \gamma_{t-1}^-(c) \oplus t \tag{8}$$

4.7.3 Static object count. We use a function $C(\gamma_t(e), t)$ to count the number of crossing events from $-\infty$ to t at edge e. We answer the static object count query by combining Equation 8, $C(\gamma_t(e), t)$ and Theorem 4.1.

Theorem 4.2. *If the tracking forms are updated as per Equation 8, the number of objects inside a cell complex σ up until t_q, is given by: $\gamma_t(\partial\sigma) = \sum_{e\in\partial\sigma} C(\gamma_t^+(e), t_q) - C(\gamma_t^-(e), t_q)$.*

Proof sketch. In Figure 10, a blue trajectory enters through edge b at t_0 and exits from edge c at t_3. The green trajectory enters through edge b at t_2, and the red trajectory enters through edge a at t_1. According to Equation 8, $\gamma_{t_3}^+(a) = \{t_1\}$ since the

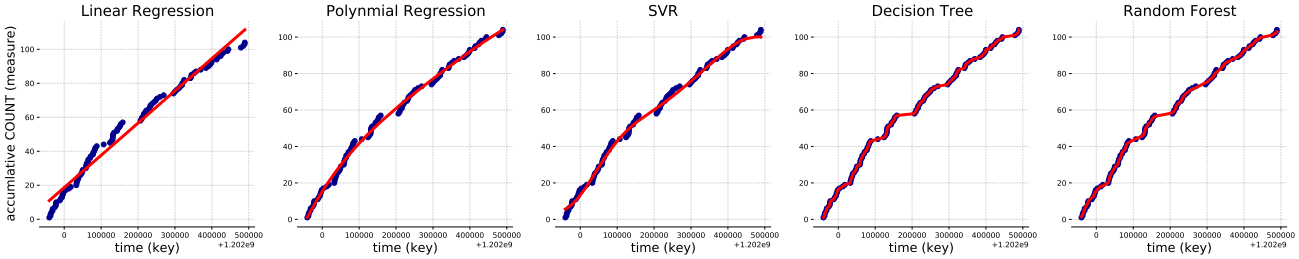


Figure 9: Storing the range count and timestamp(s) in an example tracking form using popular regressors.

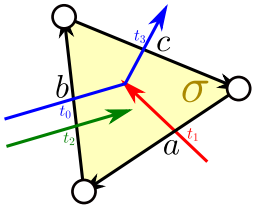


Figure 10: Example of two trajectories moving in and out of σ at different timestamps

red trajectories enters through edge a at t_1. Similarly, $\gamma^+_{t_3}(a) = \{t_1\}$, $\gamma^+_{t_3}(b) = \{t_0, t_2\}$ and $\gamma^-_{t_3}(c) = \{t_3\}$. We find the count in σ by applying Theorem 4.2. Therefore, the count is $\gamma_t(\partial\sigma) = C(\gamma^+_t(a), t_3) + C(\gamma^+_t(b), t_3) - C(\gamma^-_t(c), t_3) = 1 + 2 - 1 = 2$.

4.7.4 Transient object count. We modify the count function to find the number of objects between an arbitrary time range for the transient object count case. The count function takes the form $C(\gamma_t(e), t_1, t_2) = C(\gamma_t(e), t_2) - C(\gamma_t(e), t_1)$, where it returns the number of events from t_1 until t_2 at an edge e.

Theorem 4.3. *The number of objects that entered and left a cell complex σ from t_0 to t_1 is given by: $\gamma_t(\partial\sigma) = \sum_{e\in\partial\sigma} C(\gamma^+_t(e), t_0, t_1) - C(\gamma^-_t(e), t_0, t_1)$, where negative results mean more moving objects left during the time interval and vice versa.*

Proof sketch. In Figure 10, we want to find the transient object count in σ between $[t_1, t_3]$. The count at edge b is $C(\gamma^+_t(b), t_1, t_3) = C(\gamma^+_t(b), t_3) - C(\gamma^+_t(b), t_1) = 2 - 1 = 1$. The same logic is applied to edges a and c. According to Theorem 4.3, the count is $\gamma_t(\partial\sigma) = C(\gamma^+_t(a), t_1, t_3) + C(\gamma^+_t(b), t_1, t_3) - C(\gamma^-_t(c), t_1, t_3) = 0 + 1 - 1 = 0$. Intuitively, this makes sense as there are always two objects within σ during $[t_1, t_3]$.

4.8 Computing the count function using regression models

Lastly, we introduce the machine-learning models used to store and process timestamp sequences efficiently. The naive method to compute $C(\gamma_t(e), t)$ is to search the count at time t in the sequence $\gamma_t(e)$ at edge e. We can use indexing methods such as the B-tree to efficiently search and store the sequence. However, such indexing methods require the sequence to be stored explicitly, thus, making the storage size grow with increasing data size. We take inspiration from learned indexes to compact the sequence of timestamps which saves space. The tracking form's monotonic property (the temporal dimension continuously increases and γ^+ and γ^- are always increasing) allows us to use a similar approach to store tracking forms at each edge. Unlike learned indexing, we only store the model parameter and infer the results to reduce time complexity. This also grants us another level of approximation in the temporal dimension, increasing privacy.

Specifically, we store the timestamps as a CDF (cumulative distribution function) and predict the cumulative count of events that happened till a given timestamp t. The count is approximated by using the predicted count as the final answer. The lookup time is reduced to $O(1)$ (considering the model does not increase its parameters as we receive more events). Simple regression models are used to reduce storage and training costs. Figure 9 shows popular regressors modeling the function $C(\gamma_t(e), t)$ at an edge e. Assuming we are using linear regression (Figure 9a), we can compute the count $C(\gamma_t(e), t)$ by applying $C(\gamma_t(e), t) = \alpha + \beta t$, where β and α are the weights and biases of the linear model. The range count $C(\gamma_t(e), t_0, t_1)$ is the difference between $C(\gamma_t(e), t_1) = \beta + \alpha t_1$ and $C(\gamma_t(e), t_0) = \beta + \alpha t_0$.

We use a limited-size buffer that can accommodate n events to store new crossing events for updates. When the buffer is full, we build a new model and flush the buffer to make space for updates. When combing the model and the buffer, we can answer range queries on at most $2n$ events in the past. We can further reduce the storage space by learning the regressors incrementally. For example, we learn a model at time t that combines the buffer $[t - n, t]$ and the trained model at $t - n$. Alternatively, without storing the keys, we can use a similar queue-type learned index structure of FLIRT [41]. We leave this to future work and only study the feasibility of using a regression model to store the tracking form in this paper.

4.9 Theoretical Cost

The communication cost dominates the querying cost. Hence, the number of nodes along the perimeter of the query region serves as a proxy estimation for the querying cost.

We first formulate the cost for the non-sampled sensor graph G to serve as a benchmark. Suppose the distribution of the nodes is approximately uniform. In that case, the expected number of nodes within the query region is given by $|N_A| = \frac{A(Q_R)}{A(T_R)}|N|$, where $A(Q_R)$ is the area of the query region, $A(T_R)$ is the total area of the spatial region, and $|N|$ is the total number of nodes in G. $\frac{A(Q_R)}{A(T_R)}$ here is the proportion of the query area with respect to the total area.

We are interested in the number of points on the perimeter of the query region Q_R. We expect those points to be located mostly in a narrow band that runs close to the borders of Q_R. Say the total area of that band is $\alpha A(Q_R)$. Assuming that the nodes of G have an approximately uniform spatial distribution, the density of nodes everywhere is similar. Hence, if Q_R increases, we expect the perimeter band to increase proportionally to $A(Q_R)$. Hence,

we hypothesize that the number of nodes on the query perimeter is $|N_P| = \alpha \frac{A(Q_R)}{A(T_R)}|N|$.

Now, we compare it with our sampling strategy. For simplicity, consider the kNN-based scheme. The total number of points in the sample graph $\tilde{G}$ can be approximated by $mk\ell_G$, where m is the number of nodes sampled, k is the number of neighbors selected for each of the m nodes (see Section 4.5) and ℓ_G is the average shortest path length in G. For the triangulation-based connections, k would be the average degree for each node, which can be estimated with $k = \frac{|\tilde{E}|}{|\tilde{N}|} = \frac{3|\tilde{N}|-6}{|\tilde{N}|}$ (Euler's Formula for planar graphs). Hence, considering an approximately uniform distribution, there will be $|\tilde{N_P}| = \frac{A(Q_R)}{A(T_R)} mk\ell_G$ nodes from $\tilde{G}$ inside the query region. This considers the cost of aggregating counts from sensors to communication nodes.

Let's now consider the order of ℓ_G. It is well-known that many real-life networks exhibit the small world phenomenon, by which the average path length is $O(\log |N|)$ [32], where N is the number of nodes. The exact characterization of graphs that have this property is complicated [10], but in general, we expect $\ell_G = g(|N|)$ to hold, where g is a sub-linear function, logarithmic in the best case.

With this, $|\tilde{N_P}|$ simplifies to $|\tilde{N_P}| = \frac{A(Q_R)}{A(T_R)} mkg(|N|)$. The definitive comparison between the non-sampled cost $|N_P|$ and the sampled cost $|\tilde{N_P}|$ will depend on the exact order of the function g, but in general, we expect the small-world phenomenon to manifest and the sampled cost $|\tilde{N_P}|$ to be lower.

For the previous discussion, we assumed the $\frac{A(Q_R)}{A(T_R)}$ bounds our query region. However, this may not be the case for the upper bound approximation (E.g., $R_1 > Q_R$ in Figure 7). Determining the minimal area that contains the query region is non-trivial. It depends on many aspects, such as the distribution of nodes, shape, size, and alignment of the query region. To simplify the analysis, we can increase $\frac{A(Q_R)}{A(T_R)}$ by a factor to account for the extra area, which does not change the overall complexity. In summary, the querying cost for our framework is $O\left(\frac{A(Q_R)}{A(T_R)} mkg(|N|)\right)$, with g a sub-linear function which in the best case is logarithmic.

5 EXPERIMENTS

We simulate an in-network system with abstractions in our experiments. The main focus is evaluating the algorithmic improvement independent of the real distributed implementation (e.g., the actual communication protocol). The evaluation examines our framework's performance against the baseline for static and transient count queries under lower bound & upper bound approximation, edge generation methods (k-NN vs. triangulation), query misses, and systems gains. Highlights of the results are summarized as the following:

- We show significant performance gains in terms of processing speed, the number of nodes accessed, and storage size compared to the baseline while reducing relative error.
- The submodular maximization method achieves the lowest relative error.
- The kd-tree and the QuadTree have the lowest relative error for sampling methods.
- k-NN based connectivity has lower relative error for smaller query regions.
- The regression models provide speedup in query execution time and reduction in storage requirement with minimal impact on relative error.

5.1 Experimental setup

5.1.1 Dataset & Setup. The mobility graph uses a real-world road network of Beijing [33] shown in Figure 4. The moving objects data uses two GPS trajectory datasets, T-drive [44] and Geolife [45]. The combined dataset contains 29,027 trajectories collected from February 2007 to August 2012.

We conduct the experiments on a machine with Intel E5-2680 v3@2.50GHz running Centos 8. The core count is 48, and the memory size is 128GB. Each experiment is repeated 50 times. We report the median of the runs in solid lines, and the shaded regions are between the 25th and 75th percentiles.

5.1.2 Baseline. The obvious baseline compares our method against an unsampled sensing graph G based on [34] (Does not support arbitrary time intervals). However, we also want to compare with a baseline considering existing works on privacy-aware aggregate range queries and sampling. The baseline by combining these methods, as no single baseline meets all our goals.

Specifically, the baseline uses Euler-histograms [15, 19] to count the number of objects within each face of the graph G. We assume all counts are aggregated and stored in the nodes before querying. A random index sampling algorithm [14, 29] then uniformly sample faces in the graph.

5.1.3 Pre-processing. The road network is discretized using [4]. We insert nodes at any edge intersections and remove contour nodes (nodes that are not junctions added by maps to indicate the geometry of roads). We then map-match the trajectories to the road network by mapping each trajectory location to the nearest node and connecting them via the shortest path in the graph.

5.1.4 Performance measures. We use relative error to evaluate the estimation error from approximating the query. The relative error is $\frac{|\eta-\hat{\eta}|}{\eta}$ where η denotes the actual range count (count from the unsampled graph G) and $\hat{\eta}$ denotes the approximated range counts. The closer the relative error is to zero, the more accurate the approximation.

5.1.5 Query generation. As mentioned in Section 4.6, the queries are defined in terms of the sensor network to communicate with all relevant sensors for the query. Therefore, we define query region as a union of faces of the non-sampled sensing graph G.

We simulate this by selecting a rectangular spatial region of area A before finding the faces contained by the rectangular region in the sensing graph G. The union of the faces forms a subgraph which becomes our query region Q_R. The resulting query regions will have irregular shapes and an upper bound area of A. The temporal range is randomly sampled 7-day periods. Additionally, we choose 100 query regions uniformly as the historical data for generating sensor placements with the submodular maximization algorithm.

5.2 Size of the Sampled Graph

We evaluate the frameworks under varying sizes of the sampled graph. Figure 12a for the static object count and Figure 11a for the transient object count. The queries executed are lower bound approximations, and the query regions are fixed at 1.08% ($10km^2$ in km), defined as the percentage of the total sensing area. The sizes are shown as a percentage of the size of the original sensing graph G. Our sampling methods outperform the baselines, with kd-tree and QuadTree having the lowest error. The submodular

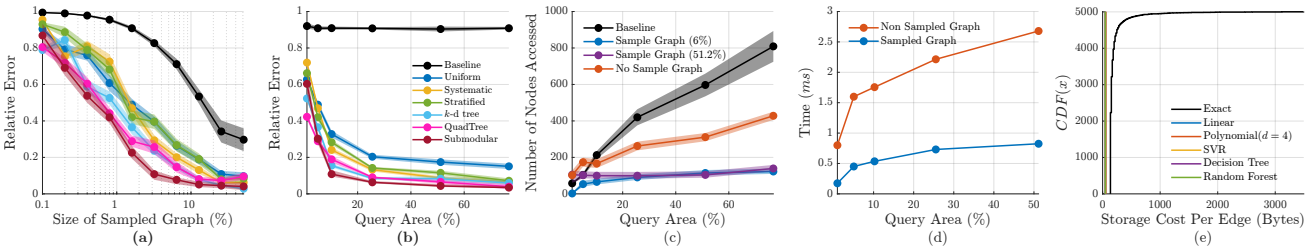


Figure 11: Lower bound relative error of transient range queries *(a)* w.r.t graph size. *(b)* w.r.t query sizes. *(c)* The number of nodes accessed w.r.t query sizes. *(d,e)* Execution time and storage cost.

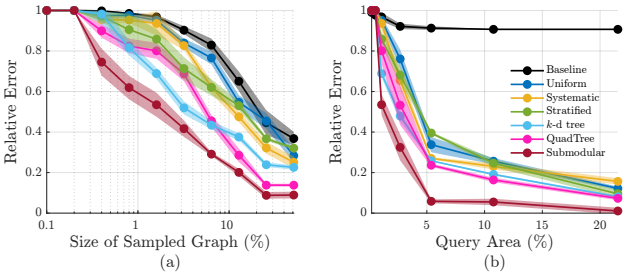


Figure 12: Lower bound relative error of static queries. *(a)* w.r.t graph size *(b)* w.r.t query sizes.

maximization method further decreases the error by considering historical query distribution.

Our method can cover more area than the baseline, which scatters sampled faces across the region. We consider the sensor distribution and track moving objects based on edge movements rather than keeping counts based on faces. Therefore, we can achieve a lower error for the same size (if we have horizontal lines on the graph). There is a lower limit to the relative error we can achieve due to the inherent nature of approximations, shown as the plateau at larger sizes. The submodular maximization method has the lowest limit, while the baseline requires more samples to reach the limit.

Interestingly, all methods were shown to have reasonably low errors at the sampled size of 25%, suggesting that sampling is a viable approach for aggregate queries without losing too much error. However, we need sufficient samples in the 0.4% – 3.2% range; all methods show poor performance. The highlight of our method is that we can reduce the number of samples to achieve a low relative error.

5.3 Query Region Size

Next, we show the relative error against varying query region sizes. The area of the query region is shown as the percentage of the total sensing area. Figure 12a shows the static object count, and Figure 11a for the transient object count over an extended query range. The size of the sampled graph is kept constant at the median graph size of 6%; other sample sizes show a similar trend.

The relative error decreases with increasing query regions as the likelihood of the query region containing a sampled face increases, with the submodular maximization method scaling exceptionally well with query size. This likelihood heavily impacts the effectiveness of the sampling methods, as smaller query regions generate substantial errors. On the other hand, the baseline only depends on the number of samples, as the area of the sampled faces predetermines the maximum coverage.

5.4 Communication Cost & Speed up

We evaluate the communication cost with the number of nodes accessed for the sampled graph, unsampled graph, and baseline in Figure 11c. We show results with a sampled graph size of 6% and 51.2%. Note that the nodes accessed differ with query sizes, but larger query sizes do not mean more nodes are accessed. E.g., in Figure 6c, the number of nodes accessed is five if the query region covers the entire perimeter or only the left triangle.

We show that sampled graphs achieve near-constant node access when averaged over graph sizes. The actual shape follows more of a logarithmic shape, which agrees with our theoretical cost in Section 4.9. The number of node access is very low for small areas, which also explains the high relative error. On the other hand, the number of access for the baseline and the unsampled graph linearly increases with the query area as we need to access all nodes within the query region.

Even though the node access is near constant for the sampled graphs, we show that the average querying time increases with query size in Figure 11d. This is because the planar graph represents the underlying sensor network. Data has to be aggregated to the sampled nodes. Larger query regions mean longer distances between the nodes, and the aggregation may need to route through more nodes. Additionally, the sampled graph reduces the processing time of queries, with a shallower slope for larger query regions.

5.5 Queries missed

We investigate the cause of the high relative error for smaller query regions and graph sizes by evaluating the number of queries missed. Query misses occur when the query region does not intersect with the sampled graph. Figure 13a & b shows the number of queries missed against the graph size and the query area. The results show that query misses rarely occur for our method, which suggests the high relative error is mainly caused by the sampled graph not covering enough area.

5.6 Upper bound approximation

We then look at the relative error for upper bound queries in Figure 13c & d. The upper bound count is given by the minimal region $\tilde{G}$ that contains the query region Q_R. The count is larger or equal to the actual count since it contains objects not in the query region (hence y-axis is greater than 1). The results show a similar trend as the lower bound count – with increasing query region and size of $\tilde{G}$, the error reduces.

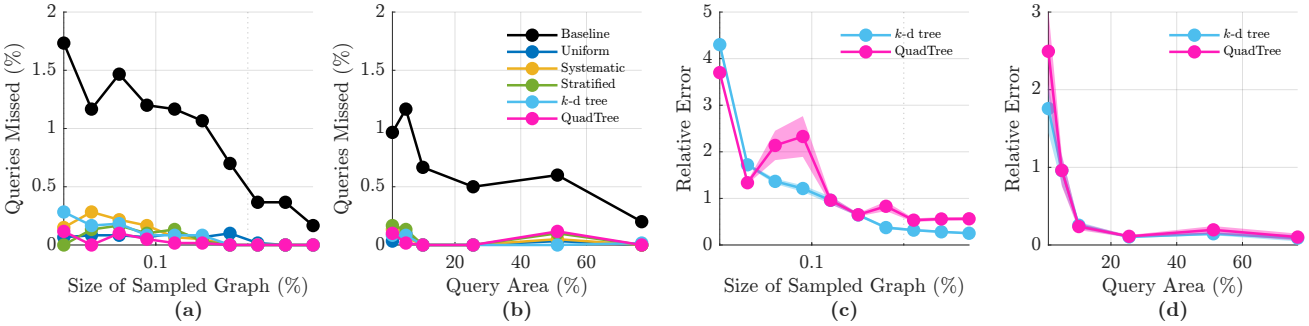


Figure 13: *(a,b)* The number of queries missed over the total number of queries. *(c,d)* The upper bound relative error.

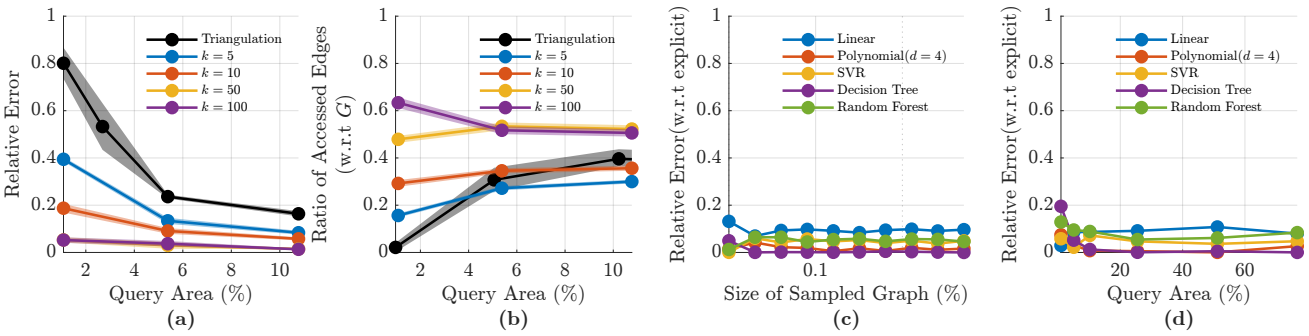


Figure 14: *(a)* The lower bound relative error of using k-NN-based connectivity. *(b)* The number of edges accessed in the sampled graph w.r.t to G. *(c,d)* The additional error generated by the regression models.

5.7 k-NN based connectivity

We also investigate if generating edges using k-NN after sampling would improve the error for smaller query areas since k-NN typically generates many smaller faces. Figure 14a shows the relative error of using k-NN compared to triangulation for sampling with a QuadTree. The error decreases with increasing k for the same query region. However, if we look at Figure 14b, we see that the number of edges accessed increases, which means more nodes need to be communicated. $k = 5$ is shown to be able to decrease the error and have less edge access compared to the triangulation method. This suggests that having more faces, each with smaller areas, effectively reduces error for smaller query areas but may be wasteful for larger query areas.

5.8 Performance of regression model

Lastly, we evaluate the error generated from the regression model and its space-saving. Figure 14c & d shows the error relative to storing counts explicitly (not to be confused with the count from unsampled graph G) for different regression models. The results suggest that simple regressors generate low error overhead. On average, the regression models add a 2.5% error penalty, greatly outweighed by the query speedup and storage reduction from these models.

Specifically, Figure 11e shows the storage-saving from the models. The y-axis shows the CDF of the number of timestamps stored in each edge with storage size in the x-axis. The CDF of the exact method shows there are more edges with smaller sizes and fewer edges with larger sizes. At around 500, there is a dramatic decrease in edges with storage costs > 500. On the other hand, the size of regression models is independent of the number of items on each edge. The maximum size of the leaned sampled graph can be calculated by $n_{edges} \times size_{model} \times 2$ regardless of the number of targets.

6 CONCLUSION

This paper presents a framework to efficiently answer privacy-aware aggregate spatiotemporal range queries within the sensor network. A planar graph representation of the sensor network allows our framework to be aware of the sensor distribution. We propose sensor placement techniques to select a subset of sensors to reduce communication based on query distribution. Additionally, we support queries with arbitrary temporal ranges with regression models to accelerate the query performance and reduce storage. Experimental results show the efficiency and accuracy of our system on real-world traffic data.

Notably, this work led to two important findings: 1) considering the sensor distribution greatly increases the effectiveness of sampling approaches compared to directly breaking down the problem with space partitioning methods. 2) Sampling (including submodular maximization) can significantly reduce communication without losing too much accuracy over communicating with the entire sensor network. However, the performance significantly depends on the number of samples, distribution of samples, and query region size, and they struggle when the sampled region does not cover enough of the query region. Based on these two findings, we believe that our framework bridges the gap between privacy-aware data aggregation with sensor placements. In future work, we can potentially address the data compression problem with data privacy guarantees and sensor placements with guaranteed query accuracy bounds.

REFERENCES

[1] Ittai Abraham, Danny Dolev, and Dahlia Malkhi. 2004. LLS: a locality aware location service for mobile ad hoc networks. In *Proceedings of the 2004 joint workshop on Foundations of mobile computing*. 75–84.

[2] Pankaj K Agarwal, Jie Gao, and Leonidas J Guibas. 2002. Kinetic medians and kd-trees. In *European Symposium on Algorithms*. Springer, 5–17.

[3] Xu Bao, Haijian Li, Lingqiao Qin, Dongwei Xu, Bin Ran, and Jian Rong. 2016. Sensor Location Problem Optimization for Traffic Network with Different Spatial Distributions of Traffic Information. *Sensors* 16, 11 (2016). https://doi.org/10.3390/s16111790

[4] Geoff Boeing. 2017. OSMnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems* 65 (2017), 126–139. https://doi.org/10.1016/j.compenvurbsys.2017.05.004

[5] Fernando Braz, Salvatore Orlando, Renzo Orsini, Alessandra Raffaeta, Alessandro Roncato, and Claudio Silvestri. 2007. Approximate aggregations in trajectory data warehouses. In *2007 IEEE 23rd international conference on data engineering workshop*. IEEE, 536–545.

[6] Mengchu Cai and Peter Revesz. 2000. Parametric R-tree: An index structure for moving objects. In *In Proc. 10th COMAD International Conference on Management of Data*. Tata McGraw-Hill, 57–64.

[7] Yixi Cai, Wei Xu, and Fu Zhang. 2021. ikd-Tree: An Incremental K-D Tree for Robotic Applications. arXiv:cs.RO/2102.10808

[8] Chih-Yung Chang, Yao-Wen Kuo, Pei Xu, and Haibao Chen. 2018. Monitoring quality guaranteed barrier coverage mechanism for traffic counting in wireless sensor networks. *IEEE Access* 6 (2018), 30778–30792.

[9] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. 2004. Querying Imprecise Data in Moving Object Environments. *IEEE Trans. on Knowl. and Data Eng.* 16, 9 (sep 2004), 1112–1127. https://doi.org/10.1109/TKDE.2004.46

[10] Rama Cont and Emily Tanimura. 2008. Small-world graphs: characterization and alternative constructions. *Advances in Applied Probability* 40, 4 (2008), 939–965.

[11] Teddy Cunningham, Graham Cormode, Hakan Ferhatosmanoglu, and Divesh Srivastava. 2021. Real-World Trajectory Sharing with Local Differential Privacy. *Proc. VLDB Endow.* 14, 11 (jul 2021), 2283–2295. https://doi.org/10.14778/3476249.3476280

[12] Yves-Alexandre De Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. 2013. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports* 3, 1 (2013), 1–5.

[13] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 969–984.

[14] Yichen Ding, Yanhua Li, Xun Zhou, Zhuojie Huang, Simin You, and Jun Luo. 2019. Sampling Big Trajectory Data for Traversal Trajectory Aggregate Query. *IEEE Transactions on Big Data* 5, 4 (December 2019), 550–563. https://doi.org/10.1109/TBDATA.2018.2830780

[15] Maryam Fanaeepour, Lars Kulik, Egemen Tanin, and Benjamin I P Rubinstein. 2015. The CASE histogram: privacy-aware processing of trajectory data using aggregates. *GeoInformatica* 19, 4 (2015), 747–798.

[16] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds. *Proceedings of the VLDB Endowment (PVLDB)* 13, 8 (2020).

[17] Sorelle A Friedler and David M Mount. 2010. Spatio-temporal range searching over compressed kinetic sensor data. In *European Symposium on Algorithms*. Springer, 386–397.

[18] J. Gao and L. Guibas. 2012. Geometric algorithms for sensor networks. *Philosophical transactions.Series A, Mathematical, physical, and engineering sciences* 370, 1958 (Jan 13 2012), 27–51. https://doi.org/10.1098/rsta.2011.0215[doi] LR: 20130424; JID: 101133385; 2011/11/30 06:00 [entrez]; 2011/11/30 06:00 [pubmed]; 2011/11/30 06:01 [medline]; ppublish.

[19] Soheila Ghane, Lars Kulik, and Kotagiri Ramamohanarao. 2018. Publishing spatial histograms under differential privacy. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*. 1–12.

[20] Abhirup Ghosh, Jiaxin Ding, Rik Sarkar, and Jie Gao. 2020. Differentially Private Range Counting in Planar Graphs for Spatial Sensing. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*. 2233–2242. https://doi.org/10.1109/INFOCOM41043.2020.9155480

[21] David Lee Hall and Sonya AH McMullen. 2004. *Mathematical techniques in multisensor data fusion*. Artech House.

[22] Zaobo He, Zhipeng Cai, Siyao Cheng, and Xiaoming Wang. 2015. Approximate aggregation for tracking quantiles and range countings in wireless sensor networks. *Theoretical Computer Science* 607 (2015), 381–390.

[23] Ziyue Huang and Ke Yi. 2021. Approximate Range Counting Under Differential Privacy. In *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[24] L Christine Kinsey. 1997. *Topology of surfaces*. Springer Science & Business Media.

[25] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 489–504. https://doi.org/10.1145/3183713.3196909

[26] Andreas Krause and Daniel Golovin. 2014. Submodular function maximization. *Tractability* 3 (2014), 71–104.

[27] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-Effective Outbreak Detection in Networks. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*. New York, NY, USA, 420–429. https://doi.org/10.1145/1281192.1281239

[28] Francesco Lettich, Salvatore Orlando, Claudio Silvestri, and Christian S Jensen. 2017. Manycore GPU processing of repeated range queries over streams of moving objects observations. *Concurrency and Computation: Practice and Experience* 29, 4 (2017), e3881.

[29] Yanhua Li, Chi-Yin Chow, Ke Deng, Mingxuan Yuan, Jia Zeng, Jia-Dong Zhang, Qiang Yang, and Zhi-Li Zhang. 2015. Sampling Big Trajectory Data. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15)*. Association for Computing Machinery, New York, NY, USA, 941–950. https://doi.org/10.1145/2806416.2806422

[30] Negar Mehr and Roberto Horowitz. 2018. A Submodular Approach for Optimal Sensor Placement in Traffic Networks. In *2018 Annual American Control Conference (ACC)*. 6353–6358. https://doi.org/10.23919/ACC.2018.8431678

[31] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14, 1 (1978), 265–294.

[32] Mark EJ Newman and Duncan J Watts. 1999. Renormalization group analysis of the small-world network model. *Physics Letters A* 263, 4-6 (1999), 341–346.

[33] OpenStreetMap contributors. 2017. Planet dump retrieved from https://planet.osm.org . https://www.openstreetmap.org.

[34] Rik Sarkar and Jie Gao. 2010. Differential Forms for Target Tracking and Aggregate Queries in Distributed Networks. In *Proceedings of the Sixteenth Annual International Conference on Mobile Computing and Networking (MobiCom '10)*. Association for Computing Machinery, New York, NY, USA, 377–388. https://doi.org/10.1145/1859995.1860038

[35] Wenhuan Shi, Qing-Jie Kong, and Yuncai Liu. 2008. A GPS/GIS Integrated System for Urban Traffic Flow Analysis. In *2008 11th International IEEE Conference on Intelligent Transportation Systems*. 844–849. https://doi.org/10.1109/ITSC.2008.4732569

[36] Yufei Tao, George Kollios, Jeffrey Considine, Feifei Li, and Dimitris Papadias. 2004. Spatio-temporal aggregation using sketches. In *Proceedings. 20th International Conference on Data Engineering*. IEEE, 214–225.

[37] Yufei Tao and Dimitris Papadias. 2005. Historical spatio-temporal aggregation. *ACM Transactions on Information Systems (TOIS)* 23, 1 (2005), 61–102.

[38] Manolis Terrovitis, Giorgos Poulis, Nikos Mamoulis, and Spiros Skiadopoulos. 2017. Local suppression and splitting techniques for privacy preserving publication of trajectories. *IEEE Transactions on Knowledge and Data Engineering* 29, 7 (2017), 1466–1479.

[39] Haojun Wang and Roger Zimmermann. 2011. Processing of Continuous Location-Based Range Queries on Moving Objects in Road Networks. *IEEE Transactions on Knowledge and Data Engineering* 23, 7 (2011), 1065–1078. https://doi.org/10.1109/TKDE.2010.171

[40] Yuni Xia and Sunil Prabhakar. 2003. Q+ Rtree: Efficient indexing for moving object databases. In *Eighth International Conference on Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings*. IEEE, 175–182.

[41] Guang Yang, Liang Liang, Ali Hadian, and Thomas Heinis. 2023. FLIRT: A Fast Learned Index for Rolling Time frames. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 234–246.

[42] Frank Yeong-Sung, Cheng-Ta, and Yen-Yi Hsu. 2010. An energy-efficient algorithm for object tracking in Wireless Sensor Networks. In *2010 IEEE International Conference on Wireless Communications, Networking and Information Security*. 424–430. https://doi.org/10.1109/WCINS.2010.5544123

[43] Frank Yeong-Sung, Yen-Yi Hsu, et al. 2010. An energy-efficient algorithm for object tracking in Wireless Sensor Networks. In *2010 IEEE international conference on wireless communications, networking and information security*. IEEE, 424–430.

[44] Jing Yuan, Yu Zheng, Chengyang Zhang, Wenlei Xie, Xing Xie, Guangzhong Sun, and Yan Huang. 2010. T-Drive: Driving Directions Based on Taxi Trajectories. ACM SIGSPATIAL GIS 2010.

[45] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A Collaborative Social Networking Service among User, location and trajectory. *IEEE Data(base) Engineering Bulletin* (June 2010).