NetClus: A Scalable Framework for Locating Top-K Sites for Placement of Trajectory-Aware Services

Shubhadip Mitra*, Priya Saraf*, Richa Sharma*, Arnab Bhattacharya*, Sayan Ranu†, Harsh Bhandari*

*Dept. of Computer Science and Engineering, Indian Institute of Technology, Kanpur, India.
†Dept. of Computer Science and Engineering, Indian Institute of Technology, Delhi, India.

Abstract—Optimal location queries identify the best locations to set up new facilities for providing service to the users. For several businesses such as gas stations, cellphone base-stations, etc., placement queries require taking into account the mobility patterns (or trajectories) of the users. In this work, we formulate the TOPS (Trajectory-Aware Optimal Placement of Services) query, which locates the best \( k \) sites on a road network for the prevailing user trajectories. The problem is NP-hard. The greedy approach, which is the state-of-the-art technique for this problem, is not scalable and practical for real urban-scale scenarios, primarily due to its high memory footprint beyond the capabilities of commodity machines. To overcome these challenges, we develop an indexing framework called Netclus that derives its power through an unique combination of FM sketches with network clustering. We provide theoretical guarantees on both the quality and the computation cost of our algorithm. Empirical studies show that Netclus requires less than 100 s to answer the TOPS query on real datasets comprising of more than 250,000 sites and 120,000 trajectories. In addition, Netclus is the only technique that can absorb dynamic updates such as change in trajectories or candidate locations, and handle constraints such as site cost and capacity.

I. INTRODUCTION

Optimal location (OL) queries in a road network aim to identify the best locations to set up new facilities with respect to a given service \([1]–[6]\). Examples include setting up new retail stores, gas stations, cellphone base stations, etc. However, for several of these services such as gas stations, ATMs, bill boards, traffic monitoring systems, etc., it is not enough to just analyze the static locations of users. Rather, the mobility patterns of the users need to be incorporated since these services are accessed while commuting. We refer to such services as trajectory-aware services.

Formally, a trajectory is a sequence of location-time coordinates that lie on the path of a moving user. Such trajectory data are commonly available from GPS traces \([7]\) or CDR (Call Detail Records) data \([6]\), social network check-ins, etc. In this work, we formalize the problem of OL queries over trajectories of users in road networks. We refer to this problem as the TOPS (Trajectory-Aware Optimal Placement of Services) query. Given a set of user trajectories \( T \) and a set of candidate sites \( S \) over a road network that can host the services, TOPS query with input parameters \( k \) and \( \tau \) seeks to report the best \( k \) sites \( Q \subseteq S \) that maximize its utility \( U(Q) \), i.e., the number of trajectories that lie within the threshold distance \( \tau \) of any site in \( Q \).

OL queries with respect to trajectories has been studied by a number of previous works \([8]–[13]\). Although the formulations are not identical, the common eventual goal is to identify the best site(s). However, all these works remain limited to theoretical exercises and cannot be applied in a real-life scenario due to a number of issues as explained next.

- **Data-based mobility model:** Existing techniques are neither based on real trajectories nor on real road networks \([8]–[12]\). They base their solutions on simplistic assumptions such as traveling in shortest paths on synthetic road networks. It is well known that the shortest path assumption does not hold in real life \([14]\). Our framework is the first to study TOPS on real trajectories and real road networks.

- **Scalability:** The state-of-the-art technique for TOPS query \([10]\) requires prohibitively large memory. Consequently, it fails to scale on urban-scale datasets. Hence, a scalable framework for TOPS query is a basic necessity. In addition, all OL queries including TOPS are typically used in an interactive fashion by varying the various parameters such as \( k \) and the coverage threshold \( \tau \) \([4]\). Hence, practical response time is desirable. This factor has been completely ignored in the existing works. While the state-of-the-art technique crashes for many query settings, it takes more than an hour to answer some of the other settings (the details are described in Sec. \( \text{VIII} \)).

- **Practical framework:** Existing techniques do not model several practical factors such as capacity constraints, site costs, updates in the set of trajectories and/or candidate sites, etc. that we study.

- **Extensive benchmarking:** Since TOPS is NP-hard, heuristics have been proposed. How do their effectiveness vary across road-network topologies? Are these heuristics biased towards certain specific parameter settings? The existing techniques do not explore these questions. We, on the other hand, perform benchmarking that is grounded to reality by extensively studying the performance of TOPS across multiple major city topologies and other important parameters.

To summarize, the proposed framework is the first practical solution to address TOPS queries. The need to develop a data-focused, practical algorithm for OL queries has been expressed recently \([4]\). However, \([4]\) assumes the users to be static. Our work enriches this field further by proposing algorithms for mobile users.

Fig. 1 depicts the top-level flow diagram of our solution. Given the raw GPS traces of user movements, they are
map-matched [15] to the corresponding road network. Using the map-matched trajectories, a multi-resolution clustering of the road network is built to construct the index structure Netclus. Indexed views of both the candidate sites and trajectories are maintained in a compressed format at various granularities. This completes the offline phase.

In the online phase, given the query parameters, the optimum clustering resolution to answer the query is identified, and the corresponding views of the trajectories and the road network are analyzed to retrieve the best set of \( k \) sites.

The major contributions of our work are as follows:

1) We formulate the trajectory-aware optimal placement of services problem, TOPS, that can handle practical factors such as capacity constraints, and site costs. (Sec. II)

2) To scale up to real city-scale datasets, we design a multi-resolution clustering-based index structure called Netclus that generates a compressed representation of the road network and the set of trajectories. Netclus further expedites computation through FM sketches and allows incremental updates of trajectories and candidate sites. (Sec. VI and Sec. VII)

3) TOPS being NP-hard, we show that Netclus provides a polynomial time approximation scheme with bounded quality guarantees (Sec. VI-B)

4) Extensive experiments show that our framework answers TOPS queries in less than 100 s on urban scale datasets comprising of more than 250,000 sites and 120,000 trajectories (Sec. VIII).

II. TOPS PROBLEM

Consider a road network \( G = (V, E) \) over a geographical area where \( V \) denotes the set of road intersections, and \( E \) denotes the set of road segments between two adjacent road intersections. The direction of the underlying traffic that passes over a road segment is modeled by directed edges.

Assume a set of candidate sites \( S = \{s_1, \ldots, s_n\} \subseteq V \) where a certain service or facility can be set up. The choice of \( S \) is generally provided by the application itself by taking into account various factors such as availability, reputation of neighborhood, price of land, etc. Most of these factors are outside the purview of the main focus of this paper and are, therefore, not studied. We simply assume that the set \( S \) is provided as input to our problem. However, as described later, if all the latent factors of choosing a site can be encapsulated as its cost, we can handle it quite robustly.

The candidate sites can be located anywhere on the road network. If it is already on a road intersection, then it is part of the set of vertices \( V \). If not, i.e., if it is on the middle of a road connecting vertices \( u \) and \( v \), without loss of generality, we augment \( V \) to include this site as a new vertex \( w \). We augment the edge set \( E \) by two new edges \((u, w)\) and \((v, w)\) (with appropriate directions) and remove the old edge \((u, v)\). Thus, ultimately, \( S \subseteq V \).

The set of candidate sites \( S \) can be in addition to existing service locations \( E_S \).

The set of trajectories over the road network is denoted by \( T = \{T_1, \ldots, T_m\} \) where each trajectory is a sequence of nodes, \( T_j = \{v_{j1}, \ldots, v_{jn}\}, v_j \in V \). The trajectories are usually recorded as GPS traces and may contain arbitrary spatial points on the road network. For our purpose, each trajectory is map-matched [15] to form a sequence of road intersections through which it passes. We assume that each trajectory belongs to a separate user. However, the framework can easily generalize to multiple trajectories belonging to a single user.

Suppose \( d(v_i, v_j) \) denotes the shortest network distance along a directed path from node \( v_i \) to \( v_j \), and \( d_r(v_i, v_j) \) denotes the shortest distance of a round-trip starting at node \( v_i \), visiting \( v_j \), and finally returning to \( v_i \), i.e., \( d_r(v_i, v_j) = d(v_i, v_j) + d(v_j, v_i) \). In general, \( d(v_i, v_j) \neq d(v_j, v_i) \), but \( d_r(v_i, v_j) = d_r(v_j, v_i) \). With a slight abuse of notation, assume that \( d_r(T_j, s_i) \) denotes the extra distance traveled by the user on trajectory \( T_j \) to avail a service at site \( s_i \). Formally, \( d_r(T_j, s_i) = \min_{v_k, v_l \in T_j} \{d(v_k, s_i) + d(s_i, v_l) - d(v_k, v_l)\} \).

It is convenient for a user to avail a service only if its location is not too far off from her trajectory. Thus, beyond a distance \( \tau \), we assume that the utility offered by a site \( s_i \) to a trajectory \( T_j \) is 0. We call this user-specific distance \( \tau \) the coverage threshold.

**Definition 1 (Coverage):** A candidate site \( s_i \) covers a trajectory \( T_j \) if the distance \( d_r(T_j, s_i) \) is at most \( \tau \), where \( \tau \geq 0 \) is the coverage threshold.

The goal of the TOPS query is to report a set of \( k \) sites \( Q \subseteq S \), \( |Q| = k \), that have the largest utility. The utility of a set \( Q \) is defined as the total number of trajectories covered by any site \( s_i \in Q \):

\[
U(Q) = \sum_{T_j \in \mathcal{T}} U_j
\]

where

\[
U_j = \begin{cases} 
1 & \text{if } \exists s_i \in Q \text{ such that } s_i \text{ covers } T_j \\
0 & \text{otherwise}
\end{cases}
\]

Formally, the TOPS problem is stated as follows.

**Problem 1 (TOPS):** Given a set of trajectories \( \mathcal{T} \), a set of candidate sites \( S \) over a road network \( G \), the TOPS problem with query parameters \((k, \tau)\) seeks to report the set of \( k \) sites \( Q \subseteq S \) \( |Q| = k \), that maximizes the utility \( U(Q) \) according to Eq. 1.

TOPS problem was introduced in [10], and was shown to be NP-hard due to reduction from the set cover problem.

Besides the basic variant of TOPS as stated above, in this work, we also study various practical extensions, as described next. For easy reference, the important symbols along with their meanings are listed in Table I.
to report a set \( Q \subseteq S \) of size \( k \) that maximizes the utility \( U(\mathcal{E}_S \cup Q) \) where \( U(\cdot) \) is defined according to Eq. 1.

Since TOPS reduces to TOPS-EXISTING by assigning \( \mathcal{E}_S = \emptyset \), TOPS-EXISTING is an NP-hard problem.

III. RELATED WORK AND MOTIVATION

The related work falls in two main classes, optimal location queries [1]–[4], [13], and flow based facility location problems [8]–[12], [16]. An optimal location (OL) query has three inputs: a set of facilities, a set of users, and a set of candidate sites. The objective is to determine a candidate site to set up a new facility that optimizes an objective function based on the distances between the facilities and the users. Comparing OL query with TOPS query, we note that: (a) While fixed user locations are considered for OL queries, TOPS uses trajectories of users. (b) OL queries report only a single optimal location, while TOPS reports \( k \) locations. (c) Unlike OL queries that are solvable in polynomial time, TOPS is NP-hard (it is polynomially solvable only for \( k = 1 \)).

Recently, [13] studied OL queries on trajectories over a road network. Two algorithms were proposed to compute the optimal road segment to host a new service. Their work has quite a few limitations and differences when compared with our work: (a) Since a single optimal road segment is reported, their problem is polynomially solvable. (b) Their work identifies the optimal road segment, rather than the optimal site. (c) There is no analysis on the quality of the reported optimal road segment, either theoretically or empirically. (d) It is not shown how does the reported road segment performs for other established metrics, such as number of new users covered, distance traveled by the users to avail the service, etc. Facility location problems [17], [18] typically consider a set of users, and a set of candidate sites. The goal is to identify a set of \( k \) candidate sites that optimize certain metrics such as covering maximum number of users, or minimizing the average distance between a user and its nearest facility, etc. Almost all of these problems are NP-hard. While most of the early works assumed that the users are static, few later works did consider mobile users [8]–[12], [16]. A fairly comprehensive literature survey is available in [19]. Most of these works assume a flow model to characterize human mobility, or assume users travelling on shortest paths, instead of using real trajectories. In addition, the proposed techniques are not practical on city-scale datasets.

A. Limitations of Existing Approaches for TOPS

Since TOPS is NP-hard, any optimal algorithm takes exponential time which is impractical for city-scale datasets. The state-of-the-art algorithm was proposed by [10], which we refer to as INC-REDY. This algorithm starts with an empty set of sites \( Q = \emptyset \), and in each of the successive \( k \) iterations, it greedily selects the site \( s_i \in S - Q \), such that the resulting utility \( U(Q \cup \{s_i\}) \) is maximized. Although this approach guarantees an approximation factor of \( 1 - 1/e \), it is not scalable to large real-life datasets. The reasons are:

| Symbol | Description |
|--------|-------------|
| \( G = (V, E), | Road network \( G \) with node set \( V \) and edge set \( E \) |
| \( T, | Set of trajectories |
| \( S, | Set of candidate sites |
| \( d(u,v) | Distance of shortest path from node \( u \) to \( v \) |
| \( d_i(u,v) | Round-trip distance between nodes \( u \) and \( v \) |
| \( d_i(T_j,s_j) | Round-trip distance from trajectory \( T_j \) to site \( s_j \) |
| \( k | Desired number of service locations |
| \( \tau | Coverage threshold |
| \( cost(s_i) | Cost of candidate site \( s_i \) |
| \( cap(s_i) | Capacity of candidate site \( s_i \) |
| \( Q \subseteq S, | Set of locations to set up service |
| \( Q \subseteq S, | Utility of trajectory \( T_j \) over the set of sites \( Q \) |
| \( U(Q) = \sum_{j \in Q} U_j | Total utility offered by \( Q \) |

TABLE I: Frequently used symbols.

A. Cost Constrained TOPS

In this extension, each site \( s_i \in S \) has a cost \( cost(s_i) \) associated with it, and the goal is to select a set of sites within a fixed budget \( B \) such that they cover the maximum number of trajectories.

**Problem 2 (TOPS-COST):** Given a set of trajectories \( T \), a set of candidate sites \( S \) where each site \( s_i \in S \) has a fixed cost \( cost(s_i) \), TOPS-COST problem with query parameters \( (B, \tau) \) seeks to report a set \( Q \subseteq S \), that maximizes the utility \( U(Q) \) such that \( cost(Q) = \sum_{s_i \in Q} cost(s_i) \leq B \).

In contrast to TOPS, TOPS-COST does not restrict the number of sites selected in the answer set. As stated earlier, TOPS-COST helps to model situations where the cost of a site encapsulates various latent factors such as location, price, etc.

TOPS reduces to TOPS-COST by assigning \( cost(s_i) = 1 \) to each site and \( B = k \). Therefore, TOPS-COST is also NP-hard.

B. Capacity Constrained TOPS

In this extension, each site \( s_i \in S \) has a fixed capacity \( cap(s_i) \) that denotes the largest number of trajectories it can serve. The goal is to select a set \( Q \subseteq S \) of size \( k \) such that they serve the maximum number of trajectories.

**Problem 3 (TOPS-CAPACITY):** Consider a set of candidate sites \( S \) where each site \( s_i \in S \) can serve at most \( cap(s_i) \) trajectories. For any set \( Q \subseteq S \), let \( x_{ji} \) be a boolean indicator variable such that \( x_{ji} = 1 \) if and only if the trajectory \( T_j \in T \) is served by the site \( s_i \in Q \). TOPS-CAPACITY problem with query parameters \( (k, \tau) \) seeks to report a set \( Q \subseteq S, |Q| = k \), that maximizes the utility \( U(Q) = \sum_{j \in T} U_j \) such that \( U_j = max_{s_i \in Q} U_{ji} \), and \( \forall s_i \in Q, \sum_{T_j \in T} x_{ji} \leq cap(s_i) \).

TOPS-CAPACITY is useful in situations such as setting up gas stations where the number of cars that can be served by a single gas station in a unit of time is limited.

TOPS reduces to TOPS-CAPACITY by assigning the capacity of each site to be larger than the total number of trajectories in the dataset. Hence, TOPS-CAPACITY is NP-hard as well.

C. TOPS with Existing Services

Optimal location queries usually factor in existing service locations before identifying new ones. For example, a company already having gas stations in a city may want to expand its business. The following extension of TOPS takes into account the location of the existing services.

**Problem 4 (TOPS-EXISTING):** Suppose \( \mathcal{E}_S \) is the set of existing service locations. The goal of TOPS-EXISTING is
- **High query cost:** The input parameters for TOPS query, \((k, \tau)\), are available only at query time. Based on the value of \(\tau\), for each site \(s_i\), we need to compute its covering set \(TC(s_i)\), i.e., the set of trajectories that are within a round-trip distance of \(\tau\) fro \(s_i\). These covering sets can be generated only at query time. Even if all pairwise site-to-trajectory distances are pre-computed, this step requires a high computation cost of \(O(mn)\) (both in terms of time and memory) where \(m, n\) denote the number of trajectories, and candidate sites, respectively. Due to this reason, for real city-scale datasets (such as the Beijing dataset [20] used in our experiments that has over 120,000 trajectories and 250,000 sites), INC-GREEDY is not scalable. Fig. 2a shows that INC-GREEDY takes about 2000 sec. to complete for \(\tau = 1.2\) Km. and \(k = 5\), and goes out of memory for \(\tau > 1.2\) Km.

- **High storage cost:** As discussed above, to facilitate faster computation of covering sets \(TC\), we need to pre-compute all pairwise site-to-trajectory distances. However, for any city-scale dataset, this storage requirement is prohibitively large. For example, the Beijing dataset [20] require close to 250 GB of storage. This is unlikely to fit in the main memory and, therefore, multiple expensive random disk seeks are required at run-time. Even with pre-computed distances up to 10 Km., INC-GREEDY crashes beyond \(\tau 1.2\) Km. (shown in Fig. 2b). An alternative approach would avoid storage of such pre-computed distances by computing them at run-time. The advantage is that since \(\tau\) is known at query time, the distances can be computed only up to \(\tau\). However, there are two important drawbacks. Firstly, this distance computation overall the candidate sites, would significantly raise the query time which is already too high. Secondly, since for any sufficiently large \(\tau\), and any site \(s_i \in S\), \(|TC(s_i)| = O(m)\), hence storing all the covering sets require \(O(mn)\) space, which is impractical.

### IV. USING FM SKETCH TO SPEED UP INC-GREEDY

A careful analysis of INC-GREEDY reveals that there are two main phases of the algorithm. The first phase involves computing the covering sets \(TC(s_i)\) for each \(s_i \in S\), i.e., the set of trajectories covered by the site \(s_i\). In the second phase, the algorithm proceeds in iterations, and updates these covering sets based on the sites added to the answer set.

The first phase is heavier in terms of time and space requirements. Thus, to make it efficient, we design an index structure, NETCLUS, which is described in Sec. V and Sec. VI. The use of indexing reduces the computational burden of the update (i.e., the second) phase as well. Further, Section VII shows how NETCLUS allows easier handling of additions and deletions of candidate sites and trajectories.

In addition, the update phase of INC-GREEDY can be performed quite efficiently using FM sketches [21], [22]. We next describe the details.

### A. FM Sketch for INC-Greedy

The main use of FM sketches is in counting the number of distinct elements in a set or union of sets [21]. Suppose, the maximum number of distinct elements is \(N\). An FM sketch is a bit vector, which is initially empty, and is of size at least \(O(\log_2 N)\). The probability of an element from the domain hashing into the \(i^{th}\) bit of the FM sketch is \(2^{-i}\). Thus, if a set has \(\Gamma\) elements, the probability of the last bit marked in the FM sketch is \(O(\log_2 \Gamma)\). Hence, after the elements of a set are hashed to the FM sketch, the last bit set can be used to estimate the number of distinct elements in the set. (The details of how the hashing function is chosen and the exact estimates are in [21].) Although the FM sketch does not count the number of distinct elements exactly, it provides a multiplicative guarantee on the error in counting. When more copies of the FM sketch is used, the error decreases.

The FM sketch can be used to speed up the update stage of INC-GREEDY, since selecting a site with the largest maximal utility is the same as selecting a site that covers the largest number of distinct trajectories not yet covered.

For each site \(s_i \in S\), the set of trajectories that it covers, i.e., \(TC(s_i)\) is maintained as an FM sketch. Thus, instead of maintaining \(O(m)\)-sized lists for each site where \(m\) is the total number of trajectories, we only need to maintain \(O(\log_2 m)\)-sized bit vectors per site.

Suppose the count of distinct trajectories covered by a site \(s_i\) is \(\chi_i\). The *marginal* utility of site \(s_j\) when site \(s_i\) has been chosen is the number of distinct trajectories that the two sites *together* cover over the number of trajectories that site \(s_j\) *alone* covers. The estimate of the number of trajectories covered by the union of \(s_j\) and \(s_i\) can be obtained by the bitwise OR of the FM sketches corresponding to \(s_j\) and \(s_i\). If this estimate is \(\chi_{ij}\), the marginal utility of site \(s_j\) over site \(s_i\) is \(\chi_{ij} - \chi_i\).

Therefore, when there are \(n\) candidate sites, to determine the site that provides the best marginal utility over site \(s_i\), \(n-1\) such bitwise OR operations are performed, and the maximum is chosen. At the end of the \(\theta\)th iteration, the combined number of trajectories covered by the sites in \(Q_\theta\) is stored by the *union* of the FM sketches obtained successively in the \(\theta\) iterations. The \((\theta + 1)^{th}\) site is chosen by using this combined FM sketch as the base.

The above brute-force algorithm can be improved in the following way. The upper bound of the marginal utility for any site \(s_j\) is its own utility. Thus, if the current best marginal utility of another site \(s_k\) is already greater than that, it is not required to do the union operation with \(s_j\). If the sites are *sorted* according to their utilities, the scan can stop as soon as the first such site \(s_j\) is encountered. All sites having a lower utility are guaranteed to be useless as well.

In our implementation, the FM sketches are stored as 32-bit words. This allows handling of roughly \(2^{32}\) (which is more
than 4 billion) number of trajectories. The length 32 is chosen since the bitwise OR operation of two such regular-sized words is extremely fast in modern operating systems. This variant of INC-GREEDY based on FM sketches, is henceforth, referred to as FM-GREEDY.

V. OFFLINE CONSTRUCTION OF NETCLUS

As discussed in Sec. [II-A] INC-GREEDY has three computationally expensive components. While FM sketch expedites the information update component, computation of TC, SC, etc. still remain a bottleneck with $O(m.n)$ time and storage complexity. To overcome this scalability issue, we develop an index structure.

One of the most natural ways to achieve the above objectives is to cluster the sites in the road network to reduce the number, and then apply INC-GREEDY on the cluster representatives. The clustering of sites can be done according to two broad strategies. The first is to consider the similarities based on the set of trajectories that are covered by the sites. Each site is, thus, represented as a set of trajectories and the Jaccard similarity between them quantifies the similarity. However, this approach is not practical due to two major limitations: (1) The representation of sites as sets requires computing the trajectories that lie within the coverage threshold $\tau$ which is available only at query time. (2) Alternatively, multi-resolution clustering may be performed based on few fixed values of $\tau$, and at query time, a clustering instance of a particular resolution is chosen based on the value of the query parameter $\tau$. However, this still requires computing the similarity between each pair of sites. Owing to large number of sites, and large size of the covering sets, such computation demands impractical memory overhead. Hence, we adopt the second clustering option, that of distance-based clustering.

We first state one basic observation. If two sites are close, the sets of trajectories they cover are likely to have a high overlap. Hence, when $k \ll n$, which is typically the case, the sites chosen in the answer set are likely to be distant from each other. The index structure, NETCLUS, is designed based on the above observation.

Our method follows two main phases: offline and online. In the offline phase, clusters are built at multiple resolutions. This forms the different index instances. A particular index instance is useful for a particular range of query coverage thresholds. In the online phase, when the query parameters are known, first the appropriate index instance is chosen. The INC-GREEDY algorithm is then run with the cluster representatives of that instance. We explain the offline phase in this section and the online phase in the next. The important notations used in the NETCLUS scheme are listed in Table II.

A. Distance-Based Clustering

The clustering method is parameterized by a distance threshold $R$, which is the maximum cluster radius. The round-trip distance from any node within the cluster to the cluster-center is constrained to be at most $2R$. The radius is varied to obtain clusters at multiple resolutions. We describe the significance of the choices of $R$ later.

The objective of the clustering algorithm is to partition the set of nodes in the road network, $V$, into disjoint clusters such that the number of such clusters is minimal. This leads not only to savings in index storage, but more importantly, it results in faster query time, as INC-GREEDY is run on a smaller number of cluster representatives. We next describe how to achieve this objective.

1) Generalized Dominating Set Problem (GDSP): Given an undirected graph $G = (V, E)$, the dominating set problem (DSP) [23] computes a set $D \subseteq V$ of minimal cardinality, such that for each vertex $v \in V - D$, there exists a vertex $u \in D$, such that $(u, v) \in E$. DSP is NP-hard [23]. In [24], it was generalized to the measured dominating set problem for weighted graphs. In this work, we propose a generalized dominating set problem (GDSP), that uses a different notion of dominance from [24].

**Problem 5 (GDSP):** Given a weighted directed graph $G = (V, E, W)$ where $W : E \to \mathbb{N}$ assigns a positive weight for each edge in $E$, and a constant $R > 0$, a vertex $u \in V$ is said to dominate another vertex $v \in V$ if $d(u, v) + d(v, u) \leq 2R$, where $d$ denotes the directed path weight. The GDSP problem computes a set $D \subseteq V$ of minimal cardinality such that for any $v \in V - D$, there exists a vertex $u \in D$ such that $u$ dominates $v$.

GDSP is NP-hard due to a direct reduction from DSP where all edge weights are assumed to be 1, and $R = 2$.

2) Greedy Algorithm for GDSP: To solve GDSP, we adapt the greedy algorithm proposed in [24]. We refer to our algorithm as Greedy-GDSP. The only input parameter to the clustering process is the cluster radius $R$.

First, the dominating sets of every vertex $v$, denoted by $\Lambda(v)$, are computed. This is achieved by running the shortest path algorithm from a source vertex till distance $2R$. The dominance relationship is symmetric, i.e., $u \in \Lambda(v) \iff v \in \Lambda(u)$.

The main part of the Greedy-GDSP algorithm is iterative. In the first iteration, the vertex $v$ that dominates the largest number of vertices is chosen. The set of dominated vertices, $\Lambda(v)$, forms a new cluster with $v$ as the cluster center. The vertices $v$ and $\Lambda(v)$ are not considered for further comparisons. In addition, the vertices in $\Lambda(v)$ are removed from the dominating sets of other non-clustered vertices. In other words, for each $u \in \Lambda(v)$, if $u \in \Lambda(w)$ for some non-clustered vertex $w$ then $\Lambda(w) = \Lambda(w) - \{u\}$. In the subsequent iterations, the vertex with the largest incremental dominating set as produced from previous iterations is chosen. The dominated vertices that are not part of other clusters form a new cluster. The algorithm terminates when all the vertices are clustered.

Similar to the INC-GREEDY algorithm, we use FM sketches to efficiently update the dominating sets and choose the vertex with the largest incremental dominating set in each iteration. The details are same as those described in Sec. [IV] with trajectory covering sets for each candidate site in INC-GREEDY replaced by dominating sets for each vertex in Greedy-GDSP.

3) Analysis of Greedy-GDSP: The next two theorems characterize the approximation bound and time complexity of Greedy-GDSP.

**Theorem 1:** The cardinality of the dominating set computed using the proposed algorithm is within an approximation
bound of \((1 + \epsilon).(1 + \ln n)\) of the optimal where \(\epsilon\) is the approximation error of FM sketches.

**Proof:** Following the analysis in [24], the Greedy-GDSP algorithm offers an approximation bound of \(H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \leq 1 + \ln n\) which was shown to be tight unless \(P = NP\). This, however, does not consider the approximation due to the use of FM sketches. Incorporating that, the bound becomes \((1 + \epsilon).(1 + \ln n)\) where \(\epsilon\) is the approximation error of FM sketches.

**Theorem 2:** Greedy-GDSP runs in \(O(|V|(\nu \log \nu + \eta))\) time where \(\nu\) is the maximum number of vertices that are reachable within the largest round-trip distance \(R_{max}\) from any vertex \(v\), and \(\eta\) is the number of clusters returned by the algorithm.

**Proof:** As the underlying graph models a road network which is roughly planar, we assume that the number of edges (i.e., road segments) incident on any set of \(\nu\) vertices is \(O(\nu)\). Since the cost of running the shortest path algorithm from a given node in a graph \(G = (V, E)\) is \(O(|E| \log |V|)\) [25], therefore, the initial distance computation for a given source vertex requires \(O(\nu \log \nu)\) time. Thus, the distance computation across all the vertices in \(V\) takes \(O(|V| \nu \log \nu)\). The neighbors of each node \(v\) can be maintained as a list sorted by the round-trip distance from \(v\). Therefore, computing the dominating set for a particular \(R\) requires at most \(\nu\) time for each vertex. The total time for the construction of dominating sets, hence, is \(O(|V|, \nu)\).

Choosing the vertex with the largest incremental dominating set requires bitwise OR operations of FM sketches. For \(|V|\) vertices, there are at most \(|V| - 1\) such operations, each requiring \(O(f)\) time (since there are \(f\) copies of FM sketches each with a constant size of 32 bits). As the number of clusters produced is \(\eta\), the running time is \(O(|V|, \nu, \eta)\).

The total running time is, thus, \(O(|V|, (\nu \log \nu + \eta))\).

### B. Selection of Cluster Representatives

In order to run INC-GRREEDY on the clusters, each cluster needs to choose a representative candidate site. This may be different from the cluster center that was used to construct the cluster. The flexibility is needed since the cluster representative should necessarily be a candidate site, although the cluster center may be any vertex in \(V\). Taking into account the fact that the cluster representative should summarize the information about the cluster and the trajectories that pass through it, and use this information to compete against the other cluster representatives in the online phase, we study two alternatives of choosing the cluster representative:

1) The most frequently accessed candidate site, i.e., the one through which the largest number of trajectories pass through.

2) The candidate site that is closest to the cluster center.

While the first option guarantees that the utility of the cluster is at least that of its best site, the second summarizes the distribution of trajectories better. Empirical studies show that the utilities returned by the two alternatives are quite similar, but the second alternative is marginally better. Consequently, we adopt the second option.

### Table II: Important notations used in the algorithms.

| Symbol | Description                                      |
|--------|--------------------------------------------------|
| \(TC(s)\) | Set of trajectories covered by site \(s\)          |
| \(CL(g)\) | Neighbors of cluster \(g\)                       |
| \(TC(g)\) | Set of trajectories passing through cluster \(g\) |
| \(d_r(T, s)\) | Estimate of \(d_r(T, s)\) in the clustered space |
| \(TC(s)\) | Estimate of \(TC(s)\) in the clustered space     |
| \(\gamma\) | Resolution at which index instances change       |
| \(t\)   | Number of index instances                        |
| \(X_p\) | Index instance                                   |
| \(R_p\) | Cluster radius for \(T_p\)                       |
| \(\eta_p\) | Number of clusters for \(T_p\)                   |
| \(N(v)\) | Dominating set for node \(v\)                    |
| \(f\)   | Number of bit vectors for FM sketch              |
| \(\epsilon\) | Error parameter for FM sketch                   |

### C. Cluster Information

Suppose the above clustering algorithm produces clusters of radius \(R\), i.e., the maximum round-trip distance of any node within a cluster to its cluster center is at most \(2R\). Then, a pair of clusters are considered as neighbors of each other if their centers are within a round-trip distance of \(4R(1 + \gamma)\), where \(\gamma \in (0, 1)\) is the index resolution parameter to be described in Sec. VI-D. This choice of neighborhood is explained in Sec. VI-A.

As part of the index structure, every cluster \(g_i\) stores the following information:

1) Cluster center, \(c_i\).
2) Cluster representative, \(r_i\).
3) Trajectory set, i.e., list of trajectories passing through at least one site in \(g_i\) along with their round-trip distance to \(c_i\), \(TC(g_i) = \{T_j, d_r(T_j, c_i)\}\).
4) Cluster neighbors along with the round-trip distance between their centers and \(c_i\), \(CL(g_i) = \{\{g_j, d_r(c_i, c_j)\}\}\), sorted by \(d_r(c_i, c_j)\).
5) Set of nodes in the cluster and their round-trip distance to \(c_i\), \{\(v_i, d_r(v_i, c_i)\}\).

The set \(TC(g_i)\) is computed by scanning the sequence of nodes in each trajectory \(T_j\). If \(v_i \in T_j\), then \(T_j\) is added to \(TC(g(v_i))\), where \(g(v_i)\) denotes the cluster in which \(v_i\) resides. Thus, a trajectory is represented as a sequence of clusters. As neighboring nodes in any trajectory are likely to fall into the same cluster, this allows a compressed representation of the trajectory by collapsing consecutive copies of the same cluster into one. This compression contributes towards the efficiency of NETCLUS in terms of both space and time.

**Example 1:** Fig. 3 illustrates an example of NETCLUS clustering with cluster radius \(R\) and \(\gamma = 0.5\). The clusters \(g_i\), \(g_j\), \(g_k\), and \(g_l\) have centers \(c_i\), \(c_j\), \(c_k\) and \(c_l\) respectively. The cluster \(g_i\) has two candidate sites \(r_i\) and \(s_i\). Since \(r_i\) is located at \(c_i\), it is chosen as the cluster representative. While cluster \(g_j\) has no candidate site, each of the clusters \(g_k\) and \(g_l\) have one
candidate site, namely, \( r_k \) and \( r_k' \), respectively, each of which is a cluster-representative. The distance between the cluster centers are as follows: \( d_r(c_i, c_j) = 5.5R \), \( d_r(c_i, c_k) > 2R \), and \( d_r(c_i, c_l) = 4R \) where \( > \) means just greater than. The distance between any other pair of cluster centers is greater than or equal to \( 6R \). Given that \( \gamma = 0.5 \), the distance between any two neighboring cluster centers lies in the range \([2R, 6R]\). Based on this, the cluster neighbors, \( C.L \), are shown in the figure. The trajectories \( T_1, T_2 \) and \( T_3 \) pass through nodes \( c_j, r_i \) and \( s_i \), respectively. The figure lists the trajectory sets \( T.C \) for each cluster. Note that when \( \tau \geq 4R \), it is guaranteed that any site covers any trajectory that passes through the same cluster. For example, \( r_k \) covers \( T_2 \) as it passes through the cluster \( g_k \) that contains \( r_k \). \( \square \)

D. Multi-Resolution Index Structure

We next explain how the multi-resolution index structure, \( N.E.T.C.U.L.S. \), is built by using the clustering algorithm outlined above. Assume that the normal range of query coverage threshold \( \tau \) is \([\tau_{\min}, \tau_{\max}]\). (We discuss the two extreme cases later.) \( N.E.T.C.U.L.S. \) maintains \( t \) instances of index structures \( I_0, \ldots, I_{t-1} \) of varying cluster radii. From one instance to the next, the radius increases by a factor of \((1 + \gamma)\) for some \( \gamma \geq 0 \). Thus, the total number of index instances is \( t = \left\lfloor \log_{(1+\gamma)}(\tau_{\max}/\tau_{\min}) \right\rfloor + 1 \). For each instance, all the clusters and their associated information are stored.

Consider a particular index instance \( I_p \) with cluster radius \( R_p \). As discussed above, the maximum round-trip distance from a site \( s_i \) belonging to the cluster \( g_j \) to a trajectory \( T_j \) that passes through \( g_j \) is at most \( d_r(T_j, s_i) \leq 4R_p \). Thus, if the coverage threshold \( \tau \leq 4R_p \), then it is not guaranteed if \( s_i \) covers \( T_j \) or not. Hence, the index instance \( I_p \) is not useful for any \( \tau < 4R_p \), and a finer instance with a lesser cluster radius should be used.

On the other hand, if \( \tau \) is too large, too many neighboring clusters may cover a trajectory. Therefore, intuitively, it makes sense to switch to a higher index instance with a larger cluster radius so that less number of clusters need to be processed. The parameter \((1 + \gamma)\) captures the ratio of \( \tau \) to \( 4R_p \), beyond which the switch is made. Thus, if \( \tau > 4R_p(1+\gamma) \), a higher index instance is used.

Therefore, the range of useful \( \tau \) for the index instance \( I_p \) is \([4R_p, 4R_p((1+\gamma))\)\. Hence, the lowest cluster radius is \( R_0 = (\tau_{\min}/4) \), and the successive cluster radii for instances \( I_p \), \( p = 1, \ldots, t-1 \) are \( R_p = (1+\gamma)^pR_0 \). From one index instance to the next, as the cluster radius \( R_p \) grows, the number of clusters, \( |I_p| \), falls exponentially.

Choice of \( \gamma \): The number of index instances \( t \) depends on \( \gamma \). A smaller value of \( \gamma \) creates more number of instances, thereby requiring larger storage and offline running time. The approximation error is also affected by \( \gamma \). When \( \gamma \) is smaller, the range of \( \tau \) handled by a particular index instance is tighter. Therefore, the distance approximations are better. Experimental results showing the empirical impact of \( \gamma \) are discussed in Sec. VII-B.

Extreme cases of \( \tau \): The extreme values of the range of \( \tau \), namely, \( \tau_{\min} \) and \( \tau_{\max} \), are assigned respectively as the minimum and maximum round-trip distance between any two sites in \( S \). This particular choice is guided by the following analysis. If there is a query with \( \tau < \tau_{\min} \), then the method degenerates to normal INC-GREEDY as each site becomes a cluster by itself. If, on the other hand, \( \tau \geq \tau_{\max} \), then \( N.E.T.C.U.L.S. \) reports any \( k \) sites, as each site covers every other site, and consequently, all the trajectories. Hence, the multi-resolution \( N.E.T.C.U.L.S. \) is applicable to all query coverage thresholds.

VI. QUERYING USING N.E.T.C.U.L.S.

We next explain the online phase of querying that starts after the query parameters \((k, \tau)\) are available.

The first important consideration is choosing the index instance \( I_p \) that supports the given query threshold \( \tau \). The index \( \gamma \) is computed as \( \gamma = \lfloor \log_{(1+\gamma)}(\tau_{\min}/\tau_{\max}) \rfloor \). This ensures that \( 4.4R_p \leq \tau < 4R_p((1+\gamma)) \), where \( R_p \) is the cluster radius for \( I_p \).

We next discuss how to apply TOPS on the clustered space.

A. TOPS-Cluster Problem

Consider a cluster \( g_i \) with its representative \( r_i \), and a trajectory \( T_j \) passing through a cluster \( g_j \), where \( g_j \) may or may not be equal to \( g_i \). Then \( T_j \in T.C(g_i) \) if and only if \( d_r(T_j, r_i) \leq \tau \). In the clustered space, however, we only store the distances of each trajectory from the centers of the clusters that it passes through. Hence, it is not possible to compute \( d_r(T_j, r_i) \) without extensive online computation. Hence, an approximate distance \( \hat{d}_r(T_j, r_i) \) is computed and used. The round-trip distance estimate from \( T_j \) to \( r_i \) is

\[
\hat{d}_r(T_j, r_i) = d_r(T_j, c_j) + d_r(c_j, c_i) + d_r(c_i, r_i) \tag{3}
\]

It is important to note that the distance can be estimated using only the information computed in the offline phase. Since the distances are approximate, the approximate trajectory cover of \( r_i \) is

\[
\hat{T.C}(r_i) = \{ T_j \in T.C(g_i) | \hat{d}_r(T_j, r_i) \leq \tau \} \tag{4}
\]

where \( T.C(g_i) = T.L(g_i) \cup \{ T_j \in T.L(g_j) | g_j \in C.L(g_i) \} \) consists of the trajectories passing through \( g_i \) and its neighbors \( C.L(g_i) \).

For any \( T_j \in \hat{T.C}(r_i) \), \( T_j \in T.C(g_i) \), if and only if there exists a cluster \( g_j \) such that \( T_j \in T.L(g_j) \) and \( d_r(c_i, c_j) \leq \tau \). This follows from the fact that \( \hat{d}_r(T_j, r_i) \leq \tau \). For the index instance \( I_p \), since \( \tau \leq 4R_p(1+\gamma) \), therefore, this condition reduces to \( d_r(c_i, c_j) \leq 4R_p(1+\gamma) \). This is the reason why the neighborhood of a cluster is defined as those whose centers are within a round-trip distance of \( 4R_p(1+\gamma) \) in Sec. VII-C.

Consequently, to compute the set \( \hat{T.C}(r_i) \), it is sufficient to examine only the trajectory sets of the neighbors of the cluster \( g_i \). For each trajectory \( T_j \in T.L(g_j) \), where \( g_j \) is a neighbor of \( g_i \), the approximate distance \( \hat{d}_r(T_j, r_i) \) is computed. The trajectory \( T_j \) is included in \( \hat{T.C}(r_i) \) if \( \hat{d}_r(T_j, r_i) \leq \tau \).

Example 2: The \( T.C \) sets for the three clusters in Fig. 3a are shown in Fig. 5a. Using Eq. (3), the distance estimates between each pair of cluster representative and trajectory is
shown in Table III. Since $\gamma = 0.5$, therefore, the supported range of $\tau$ is $[4R, 6R]$. Thus, if $\tau = 4R$, $\hat{T}C(r_i) = \{T_2, T_3\}$, $\hat{T}C(r_i) = \{T_2\}$, and $\hat{T}C(r_i) = \emptyset$. Similarly, if $\tau = 5.75R$, $\hat{T}C(r_i) = \{T_1, T_2, T_3\}$, $\hat{T}C(r_k) = \{T_2\}$, and $\hat{T}C(r_i) = \{T_2\}$.

If a trajectory $T_j \in \hat{T}C(r_i)$, then it also lies in the set $TC(r_i)$ since $d_r(T_j, r_i) \leq \hat{d}_r(T_j, r_i) \leq \tau$. However, the reverse is not true, since there may be a trajectory $T_j$ such that $d_r(T_j, r_i) \leq \tau$, but $d_r(T_j, r_i) > \tau$. Therefore, $\hat{T}C(r_i) \subseteq TC(r_i)$. For example, in Fig. 3a, $d_r(T_3, r_k) \leq 4R$, but $d_r(T_3, r_k) \geq 6R$ which exceeds any supposed value of $\tau$. Thus, $T_3 \notin TC(r_k)$.

Finally, using these approximate covering sets, we run the following instance of TOPS problem, called TOPS-Cluster.

Problem 6 (TOPS-Cluster): Given an index instance $\hat{I}_p$ defined over the road network $G = (V, E)$, suppose $\hat{S} \subseteq S$ denote the set of cluster representatives in $I_p$. TOPS-Cluster problem seeks to report a set of $k$ cluster representatives, $Q \subseteq \hat{S}$, such that $U(Q)$ is maximal.

To solve TOPS-Cluster, we employ INC-GREEDY on the set of cluster representatives $\hat{S}$ using the above covering sets $\hat{T}C(r_i)$. For faster updating of marginal utilities during the execution of INC-GREEDY, the approximate covering set $\hat{T}C$ (of each cluster representatives) is represented as FM sketch, in the same manner as described in Sec. IV.

### B. Quality Analysis of NetClus

Assuming each node in $V$ to be a candidate site, the next result bounds the quality of NetClus.

**Theorem 3:** Given the value of $\tau$, suppose NetClus uses the index instance $\hat{I}_p$ that has $\eta_p$ clusters. Then the approximation bound of NetClus is at least $k/\eta_p$.

**Proof:** Suppose the answer set chosen by NetClus after iteration $\theta = 1, \ldots, k$, is denoted by $Q_\theta = \{r_1, \ldots, r_\theta\}$. The marginal utility gained due to the addition of any site $r_i$ to the set $Q_{\theta-1}$ in iteration $\theta = 1, \ldots, k$ is $U_\theta(r_i) = U(Q_{\theta-1} \cup \{r_i\}) - U(Q_{\theta-1})$. Thus, $U(Q_\theta) = \sum_{i=1}^\theta U_\theta(r_i)$.

From sub-modularity of TOPS [10], it follows that successive marginal utilities are non-increasing, i.e., $U_\theta(r_\theta) \geq U_{\theta+1}(r_{\theta+1})$. Thus,

$$U_\theta(r_\theta) = \sum_{i=\theta+2}^k U_\theta(r_i) = \frac{U(Q_{\theta})}{\eta_p - \theta + 1}$$

Next, we claim that $\forall \theta = 1, 2, \ldots, \eta_p$, $U(Q_\theta) \geq (\theta/\eta_p)U(Q_{\eta_p})$. We will prove this by induction on $\theta$.

Consider the base case $\theta = 1$. Using Ineq. 8, $U(Q_1) = U_1(r_1) \geq U(Q_{\eta_p})/\eta_p$ since $U(Q_0) = 0$. Thus, the induction hypothesis is true for the base case.

Next, we assume it to be true for iteration $\theta$. For iteration $\theta + 1$,

$$U(Q_{\theta+1}) = U(Q_{\theta}) + U_{\theta+1}(r_{\theta+1}) \geq U(Q_{\theta}) + U(Q_{\theta})/\eta_p$$

$$\geq \left(\frac{\eta_p - \theta + 1}{\eta_p - \theta + 1}\right) U(Q_{\theta}) + \left(\frac{1}{\eta_p - \theta + 1}\right) U(Q_{\eta_p})$$

Using the induction hypothesis for iteration $\theta$,

$$U(Q_{\theta+1}) \geq \left(\frac{\eta_p - \theta + 1}{\eta_p - \theta + 1}\right) U(Q_{\eta_p}) + \left(\frac{1}{\eta_p - \theta + 1}\right) U(Q_{\eta_p})$$

Thus, the induction hypothesis holds true for any $\theta = 1, \ldots, \eta_p$.

Since $Q_{\eta_p} = \hat{S}$, after $k$ iterations, $U(Q_k) \geq (k/\eta_p) \times U(\hat{S})$. Since each node in $V$ is assumed to be a candidate site, i.e., $S = V$, and each trajectory in $T$ is covered by the representative of each cluster that it passes through, therefore, $U(\hat{S}) = |T|$. Since the utility of any optimal set is at most $|T|$, the approximation bound is $k/\eta_p$.

Using FM sketches over NetClus yields the following bound.

**Theorem 4:** The approximation bound of using FM sketches over NetClus is $(k/\eta_p) \times (1 + \epsilon)^k$, where $\epsilon$ is the error parameter provided by the FM sketch.

**Proof:** If the error parameter for FM sketch is $\epsilon$, running it for $k$ iterations produces an error bound of at most $(1 + \epsilon)^k$.

In conjunction with Th. 3 the required bound is obtained.

### C. Computational Complexity of NetClus

For a given value of $\tau$, suppose the index instance $\hat{I}_p$ with $|\hat{I}_p| = \eta_p$ clusters is used. Assume that the largest number of trajectories passing through a cluster is $\xi_p = \max\{|TCL(g_i)|\}$, and $\lambda_p$ is the largest number of vertices in any cluster in $\hat{I}_p$.

**Theorem 5:** The time and space complexities of NetClus are $O(k\eta_p\xi_p)$ and $O(\sum_{p=1}^t (\eta_p\xi_p + \lambda_p))$ respectively.

The proof is provided in the appendix.

The average values of $\eta_p, \lambda_p$ and $|TCL|$ are shown in Table IV for different values of cluster radius $R_p$.

### VII. Dynamic Updates and Extensions of TOPS

We next, discuss how the NetClus framework efficiently handles dynamic updates of trajectories and candidate sites. We assume that the underlying road network does not change. In each of the following cases, the updates are processed for all the index instances (of varying cluster radii).

- **Addition of a site:** Suppose a location $s_{add}$ is identified as a new candidate site, i.e., $s_{add}$ gets added to $S$. If $s_{add}$ is already in $V$, its cluster $g_{add}$ is identified. Otherwise, it is added to the cluster $g_{add}$ whose cluster center $c_{add}$ is the closest. To determine the closest cluster center, the neighbors $N(s_{add})$ of $s_{add}$ in $G$ are used. The round-trip distance to a cluster center $c_i$ is estimated using $min_{s_i \in N(s_{add})} \{d_r(s_{add}, s_i) + d_r(s_i, c_i)\}$ if $d_r(s_i, c_i)$ is available. Suppose $c_{new}$ is the nearest cluster...
center to \( s_{\text{add}} \). If the distance \( d_c(s_{\text{add}}, g_{\text{nearest}}) > 2R_p \), then we create a new cluster \( g_{\text{add}} \) with \( s_{\text{add}} \) as its center. If the identified cluster \( g_{\text{add}} \) does not have a cluster-representative, then \( s_{\text{add}} \) is marked as its new cluster representative. Else, it is determined if \( s_{\text{add}} \) can be a better representative for \( g_{\text{add}} \) as discussed in Sec. VI-B. Finally, the exact round-trip distance to the cluster center, \( d_c(s_{\text{add}}, g_{\text{center}}) \), is computed.

- **Deletion of a site**: Suppose a particular site \( s_{\text{del}} \) is no longer viable for a service and, therefore, needs to be deleted from \( S \). Suppose, \( s_{\text{del}} \) lies in the cluster \( g_{\text{del}} \). First, it is untagged as a candidate site in \( g_{\text{del}} \). If \( s_{\text{del}} \) is not the cluster representative of \( g_{\text{del}} \), nothing more needs to be done. Otherwise, another candidate site, if available, is chosen as the new cluster representative using the methodology described in Sec. VI-B.

- **Addition of a trajectory**: Suppose a new trajectory \( T_{\text{add}} \) is added. It is first mapped into a sequence of clusters, \( g_1, \ldots, g_t \). For each such cluster \( g_i \), \( T_{\text{add}} \) is added to the set \( \mathcal{T}(g_i) \) and its round-trip distance to the cluster center \( c_i \) of \( g_i \), \( d_c(T, c_i) \), is computed and stored. In addition, \( g_i \) is added to the set \( \mathcal{C}(T_{\text{add}}) \). The procedure is essentially the same one discussed in Sec. VI.

- **Deletion of a trajectory**: Suppose a trajectory \( T_{\text{del}} \) is deleted. Assume that the coverage set of \( T_{\text{del}} \) is \( \mathcal{C}(T_{\text{del}}) = \{g_1, \ldots, g_t\} \). For each such cluster \( g_i \), \( T_{\text{del}} \) is removed from its coverage set \( \mathcal{T}(g_i) \). Finally, the set \( \mathcal{C}(T_{\text{del}}) \) is deleted.

While multiple updates can be applied one after another, batch processing is more efficient. Sec. VIII-G shows that the updates are handled quite efficiently.

Next, we discuss how the NETCLUS framework can be used to solve various extensions of TOPS described in Sec. III-

- **Cost Constrained TOPS**: The NETCLUS algorithm can be adapted based on the greedy heuristic for the budgeted maximum coverage problem \([26]\) to solve ‘TOPS-COST’. The algorithm starts with an empty set of sites \( Q = \emptyset \) and proceeds in iterations. In each iteration, it selects a site \( s_i \in S - Q \) such that \( (U(Q \cup \{s_i\}) - U(Q))/\text{cost}(s_i) \) is maximal. If \( \text{cost}(s_i) \) is within the remaining budget \( B - \text{cost}(Q) \), it is added to \( Q \); otherwise, it is pruned from \( S \). This process continues until \( S = Q \).

It was shown in \([26]\) that the above approach can perform arbitrarily bad. Thus, in order to bound the approximation guarantee, the algorithm is augmented with the following step. Assume \( s_{\text{max}} \) to be a candidate site such that \( \text{cost}(s_{\text{max}}) \leq B \) and \( U(\{s_{\text{max}}\}) \) is maximal. The algorithm returns either the site \( s_{\text{max}} \) or the set \( Q \) whichever offers the maximum utility.

- **Capacity Constrained TOPS**: NETCLUS can be adapted to solve ‘TOPS-CAPACITY’ in the following manner. Since site \( s_i \) can serve at most \( \alpha_i = \text{min}\{\|TC(s_i), \text{cap}(s_i)\}\) trajectories, its marginal utility is initialized to \( \alpha_i \). In each of the successive \( k \) iterations, after selecting the site \( s_i \) wit maximal marginal utility, we set the trajectory utility \( U_j = 1 \) for any of the \( \alpha_i \) trajectories in \( TC(s_i) \). Since these trajectories are served by \( s_i \), they are pruned from the covering sets \( TC(s_j) \) for each \( s_j \notin Q \) that cover them. Consequently, the marginal utilities of all such sites get updated. The process stops after \( k \) iterations.

- **TOPS with Existing Services**: Assume \( \mathcal{E}_S \) is the set of existing service locations. On receiving the query parameter \( \tau \), the covering sets, \( TC \), over the set of sites \( S \cup \mathcal{E}_S \) are computed. NETCLUS starts with \( Q_0 = \mathcal{E}_S \) and updates the marginal utilities of the sites in \( S \). The remaining algorithm stays unaltered. The algorithm terminates after selecting \( k \) sites from the set \( S \), in the same manner as TOPS. An advantage and important feature of NETCLUS is that the site chosen in a given iteration depends solely on what the existing service locations are, and not on how they were chosen.

### VIII. Experimental Evaluation

In this section, we perform extensive experiments to establish (1) Efficiency: that NETCLUS is efficient, practical and scales well to real-life datasets, and (2) Quality: that NETCLUS produces solutions that are close to the best possible polynomial time solution.

The experiments were conducted using Java (version 1.7.0) platform on an Intel(R) Core i7-4770 CPU @3.40GHz machine with 32 GB RAM running Ubuntu 14.04.2 LTS OS.

#### A. Evaluation Methodology

**Algorithms**: We evaluate the performance of three different algorithms to address TOPS: INC-GREEDY, FM-GREEDY and NETCLUS, which are henceforth referred to as INC-G, FM-G and NETCLUS respectively in text and in the figures. TOPS, being NP-hard, there is no polynomial time optimal algorithm to solve it. Since INC-G is the best known polynomial time approximation algorithm for TOPS \([10]\), therefore, we compare FM-G and NETCLUS against it, which acts as the baseline. To answer queries with arbitrary \( \tau \), INC-G requires computing distances between each pair of site and trajectory. Referring to the discussion in Sec. III-A this leads to infeasible memory overhead. Thus, for the sake of comparison, for all the experiments, INC-G uses pre-computed distances up to 10 Km.

**Metrics of evaluation**: The main metrics of evaluation were (a) total utility measured as a percentage of the total number of trajectories \( m \), and (b) query running time. The two basic parameters studied were (i) desired number of service locations \( k \), and (ii) coverage threshold \( \tau \). The default values of \( k \) and \( \tau \) were 5 and 0.8 Km. respectively.

We conducted experiments on both real and synthetic datasets, whose details are shown in Table IV. For simplicity, we assume that the set of candidate sites is the same as the set of nodes in the graph, unless otherwise stated.

#### Real datasets: We used GPS traces of taxis from Beijing consisting of user trajectories generated by tracking taxis for a week \([20, 27]\). This is the most widely used and one of the largest publicly available trajectory datasets. To generate trajectories as sequences of road intersections, the raw GPS-traces were map-matched \([15]\) to the Beijing road network extracted from OpenStreetMap (http://www.openstreetmap.org/).

#### Synthetic datasets: To study the impact of city geographies,
TABLE V: Variation across resolution of index instances, $\gamma$.

| $\gamma$ | Time (s)     | Space (GB)  | Error % |
|--------|-------------|-------------|---------|
| 0.25   | 108427      | 14,095      | 3.54    |
| 0.50   | 3216        | 4,315       | 3.97    |
| 0.75   | 1652        | 2,374       | 4.53    |
| 1.00   | 520         | 1,053       | 5.21    |

TABLE VI: Variation across the number of FM sketches, $f$.

| $f$   | Utility | Error % | Time (ms) | Speed-up |
|------|---------|---------|-----------|----------|
| 1    | 47.23   | 26.60   | 43.67     | 84.65    | 16.32   | 51.88   |
| 2    | 47.23   | 34.27   | 27.45     | 84.65    | 21.66   | 39.09   |
| 4    | 47.23   | 39.33   | 16.73     | 84.65    | 32.65   | 25.93   |
| 10   | 47.23   | 41.27   | 12.62     | 84.65    | 65.32   | 12.96   |
| 20   | 47.23   | 43.29   | 8.34      | 84.65    | 116.32  | 7.28    |
| 30   | 47.23   | 44.96   | 4.81      | 84.65    | 161.53  | 5.24    |
| 40   | 47.23   | 45.52   | 3.66      | 84.65    | 216.62  | 3.91    |
| 50   | 47.23   | 45.89   | 2.84      | 84.65    | 272.18  | 3.11    |
| 100  | 47.23   | 46.43   | 1.69      | 84.65    | 984.17  | 0.86    |

we generated three synthetic datasets that emulate trajectories in the patterns followed in New York, Atlanta and Bangalore. We used an online traffic generator tool, MNTG (http://mntg.cs.umn.edu/tg/index.php) to generate the traffic traces, that were later map-matched to generate the trajectories in the desired format.

B. Choice of Parameters

We first run experiments to determine the choice of two important parameters: (a) the resolution of the index instances, $\gamma$, and (b) the number of FM bit vectors, $f$.

As discussed earlier in Sec. V-D, the choice of $\gamma$ affects both the storage and offline run-time costs as well as quality. Table [V] lists the values for the Beijing dataset when $\gamma$ is changed. The error is measured as a percentage loss over INC-GREEDY when NETCLUS is used. When $\gamma$ is too small, there is almost no compression of the trajectories. As a result, the index structure size is large. On the other hand, with a very large $\gamma$, the error may be unacceptable. We fix $\gamma = 0.75$ for our experiments since it offers a nice balance of a medium sized index structure that can fit in most modern systems with an error of within 5%.

Table [VI] shows how the utility and running time varies when $f$ number of FM sketches are used as compared to the original NETCLUS without using FM sketches. The value of $\tau$ and $k$ are kept at their default values (0.8 Km and 5 respectively). When $f$ is very small, the error is too large. As $f$ increases, expectedly the error decreases while the speed-up decreases as well. When $f$ is extremely high, the number of operations may overshadow the gains and using FM sketches may be actually slower. We fixed $f = 30$ since it produces less than 5% error and produces a speed-up factor of more than 5.

C. Quality Results

Fig. [I] shows the utility yields for different values of $k$ and $\tau$. We note that the utilities of FMG and NETCLUS are close to that of INC-G. For reasons to be explained in the next section, INC-G and FMG do not scale beyond $\tau = 1.2$ Km. Although the utilities of FMG are closer to INC-G, but due to high memory overhead, it fails to scale beyond $\tau = 1.2$ Km. On the other hand, NETCLUS scales well for all values of $\tau$ and its utilities are always within 93% of INC-G on an average for $\tau \leq 1.2$ Km.

D. Memory Footprint

Table [VII] shows that the memory footprints of NETCLUS are significantly lesser than those of INC-G and FMG. As the coverage threshold $\tau$ increases, the size of the covering sets, $TC$, used in INC-G and FMG, increase sharply. Consequently, these algorithms could not scale beyond $\tau = 1.2$ Km. On the other hand, with higher $\tau$, NETCLUS uses lower resolution clustering instances leading to higher data compression, thereby resulting in lower memory footprints. FMG requires slightly more memory than INC-G, due to storage of multiple bit vectors for each site.

E. Performance Results

We next measure the performance of the algorithms for different values of $k$ and $\tau$. Fig. [I] shows that for $\tau \leq 1.2$ Km., NETCLUS is up to 36 times faster than INC-G. For $\tau > 1.2$ Km., as stated in the previous section, INC-G and FMG fail to run due to high memory overheads. When $\tau$ increases, NETCLUS uses a higher index instance having lesser number of clusters, leading to its efficiency. On the other hand, INC-G and FMG use covering sets $TC$ of increasingly larger size, resulting in poor performance.

Although FMG offers a speed up of about 5 times in the algorithm running time when compared to INC-G, its effect is negated by a relatively large initial pre-processing time required for computing the covering sets. Due to this fact, we only compare the results of NETCLUS with that of INC-G in the subsequent sections.

F. Extensions of TOPS

We next show results of NETCLUS on different TOPS extensions (discussed in Sec. VII) over the Beijing dataset.

- **TOPS-COST**: We consider a budget of $B = 5.0$ and $\tau = 0.8$ Km. The cost of each site was assigned using a normal distribution with mean $\mu = 1.0$ and standard deviation varied between $\sigma \in [0, 1]$ (the least cost of a site was constrained to be 0.1). Fig. [I] shows that the utility increases with standard deviation (note that $\sigma = 0$ degenerates to basic TOPS). This is due to the fact that with higher standard deviation, more number of sites can be chosen with lower costs which ultimately leads to larger number of trajectories being covered.
- **TOPS-CAPACITY**: We consider \( k = 5 \) and \( \tau = 0.8 \) Km. The sites were assigned varying capacities drawn from a normal distribution where the mean was varied in the range \([0.1%, 100\%]\) of the total number of trajectories, and the standard deviation fixed at 10% of the mean. (note that mean capacity of 100% corresponds to basic unconstrained TOPS). Fig. 6 shows that, as expected, utility increases with mean capacity. \textsc{NetClus} has almost the same utility as that of \textsc{INCG}. We do not show the running time plots, as the algorithms for \textsc{TOPS-CAPACITY} are almost the same as those for \textsc{TOPS} and, hence, exhibit similar performance.

**G. Updates of Sites and Trajectories**

Table VIII shows that \textsc{NetClus} can efficiently process dynamic updates of sites and trajectories. Adding a trajectory requires more time than that for a candidate site since a trajectory usually passes through multiple clusters and the cluster-information of all those clusters need to be updated. Adding a site, on the other hand, requires simply finding the cluster it is in and updating the cluster representative, if needed.

**H. Robustness with Parameters**

**Number of Sites and Trajectories**: Fig. 7 shows scalability of \textsc{NetClus} w.r.t. number of sites and trajectories for default values of \( k \) and \( \tau \). Since \textsc{NetClus} clusters the sites and represents the trajectories in a compressed format, it scales much better than \textsc{INCG} with both number of sites and trajectories.

**City Geometries**: We experimented with three typical city geometries, Atlanta, New York, and Bangalore (Fig. 8). New York has a star topology while Bangalore is poly-centric. Consequently, Bangalore offers higher utility. Since Atlanta has a mesh structure with trajectories distributed all over the city, its utility is lowest. The running times for Atlanta and New York are higher than that of Bangalore because of much larger road network.

**Length of Trajectories**: To determine the effect of length of trajectories, the trajectories were divided into four classes based on their lengths and from each of the classes, 5,000 trajectories were sampled (Fig. 9). Longer trajectories are easier to cover since they pass through more number of candidate sites over a larger area and, therefore, exhibit higher utility than the shorter ones. The running time also increases with trajectory length due to more number of update operations of the marginal utilities.

**I. Index Construction**

Table IX shows different parameters associated with construction of \textsc{NetClus} index. As the cluster radius, \( R_p \), increases, the number of clusters, \( \eta_p \), decreases as the average dominating set sizes, \( |\Lambda| \), increases. Therefore, the average number of trajectories passing through a cluster, \( |TC| \), also increases. The average number of neighbors of a cluster, \( |CL| \), initially increases but finally decreases. The running time are practical and show the same trend.

**IX. Conclusions**

In this paper, we study the TOPS problem of finding facility locations for trajectory-aware services. The existing state-of-the-art algorithm \textsc{INC-Greedy} does not scale on large city-scale datasets. Thus, to overcome this, we develop an index structure, \textsc{NetClus}, based on multi-resolution clustering of the road network and FM sketches. Extensive experiments over urban scale datasets show that \textsc{NetClus} is practical and scalable. We also show that \textsc{NetClus} can efficiently handle dynamic updates to sites and trajectories, and moreover, can be easily adapted to answer various extensions such as those involving cost or capacity constraints.

In future, we would like to study a more generic formulation of the TOPS problem.
Table IX: Details of indexing for Beijing road network

| Facility location: a survey of applications and methods |
---|---|---|---|---|---|
| 19-21 | 15775 | 17.10 | 1525.09 | 42.73 | 281.62 |
| 28510 | 9.46 | 1162.93 | 53.74 | 205.74 |
| 233729 | 1.38 | 592.45 | 12.62 | 255.60 |
| 258340 | 1.04 | 561.88 | 4.29 | 269.35 |
| 76836 | 3.51 | 757.03 | 45.38 | 192.12 |
| 195910 | 1.15 | 571.41 | 6.43 | 239.58 |

References

[1] Y. Du, D. Zhang, and T. Xia, “The optimal-location query,” in Advances in Spatial and Temporal Databases, 2005, pp. 163–180.

[2] P. Ghaemi, K. Shahabi, J. P. Wilson, and F. Banaei-Kashani, “Optimal network location queries,” in SIGSPATIAL GIS, 2010, pp. 478–481.

[3] X. Xiao, B. Yao, and P. Li, “Optimal location queries in road network databases,” in ICDE, 2011, pp. 804–815.

[4] Z. Chen, Y. Liu, R. C.-W. Wong, J. Xiong, G. Mai, and C. Long, “Effective algorithms for optimal location queries in road networks,” in SIGMOD, 2014, pp. 123–134.

[5] S. Mitra, S. Ranu, V. Kolar, A. Telang, A. Bhattacharya, R. Kokku, and S. Raghavan, “Trajectory aware macro-cell planning for mobile users,” in INFOCOM, 2015.

[6] Y. Kolar, S. Ranu, A. P. Subramanian, Y. Shriniwasan, A. Telang, R. Kokku, and S. Raghavan, “People in motion: Spatio-temporal analytics on call detail records,” in COMSNETS, 2014, pp. 1–4.

[7] W. He, D. Li, T. Zhang, L. An, M. Guo, and G. Chen, “Mining regular routes from gps data for ridesharing recommendations,” in SIGKDD, 2012, pp. 79–86.

[8] O. Berman, R. C. Larson, and N. Fouska, “Optimal location of discretionary service facilities,” Transportation Science, vol. 26, no. 3, pp. 201–211, 1992.

[9] O. Berman, D. Krass, and C. W. Xu, “Locating discretionary service facilities based on probabilistic customer flows,” Transportation Science, vol. 29, no. 3, pp. 276–290, 1995.

[10] O. Berman, D. Bertsimas, and R. C. Larson, “Locating discretionary service facilities, ii: maximizing market size, minimizing inconvenience,” Operations Research, vol. 43, no. 4, pp. 623–632, 1995.

[11] O. Berman and D. Krass, “The generalized maximal covering location problem,” Computers & Operations Research, vol. 29, no. 6, pp. 563–581, 2002.

[12] O. Berman, “Flow intercepting spatial interaction model: a new approach to optimal location of competitive facilities,” Location Science, vol. 6, no. 1, pp. 41–65, 1998.

[13] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan, “Trajectory based optimal segment computation in road network databases,” in SIGSPATIAL GIS, 2013, pp. 396–399.

[14] P. Banerjee, S. Ranu, and S. Raghavan, “Inferring uncertain trajectories from partial observations,” in JCDM, 2014, pp. 30–39.

[15] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, “Map-matching for low-sampling-rate gsp trajectories,” in GIS, 2009, pp. 352–361.

[16] O. Berman, D. Krass, and C. W. Xu, “Locating flow-intercepting facilities: New approaches and results,” Annals of Operations research, vol. 60, no. 1, pp. 121–143, 1995.

[17] Z. Drezner, Facility location: a survey of applications and methods. Springer, 1995.