Efficient Trajectory Cover Search for Moving Object Trajectories

Vikram Goyal, Ankita Likhyani, Neha Bansal
Information Management and Data Analytics Group,
IIIT-Delhi,
New Delhi, India
{*vikram, ankita1120, neha1125*} *@iiitd.ac.in*

Ling Liu
College of Computing,
Georgia Institute of Technology
Atlanta, Georgia
ling.liu@cc.gatech.edu

***Abstract*—Given a set of query locations and a set of query keywords, a Trajectory Cover (CT) query over a repository of mobile trajectories returns a minimal set of trajectories that maximally covers the query keywords and are also spatially close to the query locations. Processing CT queries over mobile trajectories requires substantially different algorithms than those for location range queries. The main contributions of this work are three-fold. First, we introduce a notion of Trajectory Cover that enables mobile users to get most relevant trajectories of their interest to plan their route of travel. Second, we show that CT search is an NP-hard problem and we present a greedy algorithm that combines spatial proximity and keyword proximity with an efficient filter. We use the *promising value* filter to select better candidate trajectories for inclusion in the CT and prune out non-promising trajectories in the Greedy search process. We also develop a lower bound and upper bound mechanism to efficiently compute the *promising value* of each trajectory, allowing our promising value based greedy algorithm to scale to large trajectory databases. Finally, we conduct a performance study to demonstrate that our greedy algorithm is efficient in evaluating trajectory cover queries.**

I. Introduction

Technical advances in location-acquisition devices have resulted into generation of huge volume of trajectory data. In such datasets, a trajectory is a sequence of places, each place having an associated text description. Popular examples are community based resources such as bikely.com[1], location-based social networks like Geolife[2], social networks like Flickr[3] and Foursquare[4], where people upload their travel routes, download and search for other user's travel routes. People also write blogs to share life experiences and build connections among each other using human location history. One of the most interesting types of queries is to find the routes that have all their travel locations in a given region and contain a given set of keywords. Such trajectory queries can be used for providing mobile services like helping people to plan their travel by knowing the travel experiences of others, recommending relevant routes to the mobile users on the basis of their preferences etc. However, retrieving routes that match both the spatial locations and textual descriptions from the trajectory dataset demands different query processing algorithms than location based range queries.

The existing research on querying spatial-textual data mainly focuses on spatial web object retrieval with or without any text description. For example, a location aware top-k text retrieval query returns a list of k spatial web objects ranked according to a ranking function that combines their distances to the query location and the relevance of their textual description to the query keywords [1]. Recent work on trajectory retrieval can be classified into two categories: spatial relevance without keyword coverage and spatial relevance with keyword relevance. For example, [2] studies a k-nearest neighbor trajectory query which returns the k-nearest mobile object trajectories to a given set of query points based on the minimum distance from the query points to a trajectory. [3] considers both a set of points and a set of keywords in a query, and returns the top-k trajectories based on a rank formula combining textual relevancy with spatial relevancy scores. We observe that existing work solves the spatial-textual matching of trajectories by treating each trajectory as independent entity and thus focusing on searching individual trajectory that maximizes the textual similarity and minimizes the spatial distance with respect to the query. This approach fails to provide good results when the number of query keywords and the set of query locations are not small since the top-k matching trajectories tend to be sub-optimal in terms of keyword coverage and spatial proximity. For example, if there is a query for Chinese restaurant, Starbuck coffee, dry clean store as activities and there are many trajectories with only Starbuck coffee activity near the set of query locations, earlier approaches would only return those trajectories with Starbuck coffee. We argue that finding a smallest subset of trajectories with maximal coverage of the query activities and the maximum spatial vicinity to the set of query locations will provide higher quality of spatial-textual trajectory query services by delivering what users really desire to find.

In this paper, we define a new query type called Trajectory Cover (CT) query. A CT query consists of a set of points and a set of keywords, and returns a minimal set of trajectories that are textually relevant to the query

[1]http://www.bikely.com/
[2]http://research.microsoft.com/en-us/projects/geolife/default.aspx
[3]http://www.flickr.com
[4]https://foursquare.com/

keywords with maximum coverage and are spatially close to the query location points with minimum spatial proximity. This type of queries will be most relevant in scenarios where a user wants to find the routes in which she can do all the query activities at/near to the query locations. In addition to introducing trajectory cover queries, we also develop an efficient approach to processing CT queries. First, we show that the trajectory cover problem is NP hard. We propose and implement a heuristic-based Greedy algorithm with a novel filter to efficiently retrieve a trajectory cover for each given query. The filter is based on the concept of promising value, which helps to identify candidate trajectories that are highly relevant to finding the CT. To speed up the computation of promising value for each trajectory, we develop the lower bound and upper bound mechanism to filter out irrelevant trajectories that do not contribute to the CT and minimize unnecessary computation of non-relevant trajectories. To highlight our greedy algorithm with dual goals of maximization of both textual and spatial relevance, we also implement two baseline approaches, one focuses on maximal keyword coverage coined as Keyword-based approach (KA) and the other focuses on maximal spatial relevance, coined as Location-based approach (LA). We demonstrate the effectiveness of the Greedy approach using real world trajectory datasets. To the best of our knowledge no existing work has studied trajectory cover queries.

The rest of the paper is organized as follows. In section 2, we give problem overview and proof of NP-hardness for trajectory cover query. Section 3 presents our proposed algorithm. Experimental setup and results are described in section 4. Section 5 discusses related work. Finally we conclude the paper in section 6 with some directions for future work.

II. Overview

In this section we first introduce the basic concepts and definitions involved in the trajectory cover query model. Then we give a brief overview of our solution approach.

A. Basic Concepts and Definitions

Let D be a dataset in which each data object is an activity trajectory. Each activity trajectory $t \in D$ is defined as a sequence of places with keyword based annotations. Formally, $t = p_1, ..., p_i, ..., p_n$ and $p_i =< l_i, k_i >$ where l_i denotes the geometric coordinate position of the place p_i and k_i denotes the set of keywords that describe the place p_i in terms of activities done at that location. For presentation convenience, when p = (l, k), we use $p.l$ to denote the geo-spatial location of p and $p.k$ to denote the set of keywords associated to the place p.

Figure 1 shows some examples of trajectories, i.e. $D = \{t_1, t_2, t_3, t_4\}$, where trajectory places and query points are represented by circles and triangles, respectively. Consider place p_{11} of trajectory t_1. We have $p_{11}.l$ = $(15, 14)$ and $p_{11}.k$ = $\{r\}$.

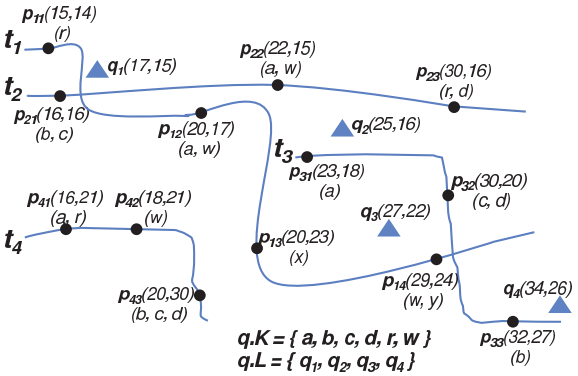

Figure 1: Example

A user query q over a repository of activity trajectories is defined in terms of two components: a set of keywords K and a set of query locations L, i.e. $q = (L, K)$. In our running example of Figure 1, the user query $q = (L, K)$ has $q.L = \{q_1, q_2, q_3, q_4\}$ and $q.K = \{a, b, c, d, r, w\}$. For query q, we say that a place p_j in trajectory t is a valid place with respect to the query q if $t.p_j.k$ contains at least one query keyword, i.e., $q.K \cap t.p_j.k \neq \phi$. For example, in Figure 1, $t_1.p_{11}$ is a valid place whereas $t_1.p_{13}$ is not. For presentation convenience, in the remaining of the paper, when places are used in our definitions or algorithms, we refer to only valid places unless mentioned explicitly otherwise.

Definition 1: (**Trajectory Query Distance (TQD)**) The trajectory query distance $TQD(q, t)$ of a trajectory t from a user query q is defined as follows:

$$TQD(q,t) = \sum_i \min_j Dist(q.l_i, t.p_j),$$
$$\exists k' \in q.K, k' \in t.p_j.k,$$

Dist(·, ·) is the euclidean distance between two locations, and each trajectory place $t.p_j$ considered in $Dist()$ function is at the minimum distance from some query point $q.l_i$.

In Figure 1, the distance of a query point q_1 from trajectory t_3 is about 6.70, because the place $p_{31} \in t_3$ is the valid nearest place out of all the valid places in t_3 with respect to q_1. Similarly for q_2, q_3 and q_4, we have p_{31}, p_{32} and p_{33} places at a distance of about 2.82, 3.61 and 2.24, respectively. Thus, $TQD(q, t_3)$ = 6.70 + 2.82 + 3.61 + 2.24 = 15.37.

Definition 2: (**Trajectory Set Query Distance (TSQD)**) The trajectory set query distance $TSQD(q, G)$ of a set of trajectories G from a user query q is defined as follows:

$$TSQD(q,G) = \sum_{t_i \in G} TQD(q,\ t_i).$$

Consider the trajectory set $G = \{t_1, t_4\}$ in our running example of Figure 1, $TSQD(q, G) = (TQD(q, t_1) + TQD(q, t_4)) = (15.54 + 38.3) = 53.84$.

Definition 3: (**Trajectory Query Keywords (TQK)**)

Trajectory Query Keywords of a trajectory t is the set of keywords obtained by taking the union of query keywords at the nearest places of the trajectory from the query points.

$$TQK(t,q) = \bigcup_i \ (t.p_j.k \cap q.K),$$
$$\forall m, j \neq m, Dist(q.l_i, t.p_j) \leq Dist(q.l_i, t.p_m).$$

For example, TQK of trajectory t_1 is $\{a, r, w\}$, as p_{11}, p_{12}, p_{14} locations are the nearest trajectory places from query points and have keywords $\{a, r, w\}$ associated to them. It may be noted that $TQK(t,q)$ would always be subset of trajectory keywords $t.K$.

Definition 4: (**Trajectory Set Query Keywords (TSQK)**) Trajectory Set Query Keywords of a set of trajectories G with respect to query q is defined as the set of query keywords covered by trajectories in set G, i.e.,

$$TSQK(q,G) = \bigcup \ TQK(t,q), \forall t \in G$$

A set G of trajectories is said to cover a user query q with respect to keywords, if and only if $q.K \subseteq TSQK(q,G)$. In our running example of Figure 1, the set $G = \{t_1, t_3\}$ covers the query keywords of q.

Definition 5: (**Trajectory Cover (CT)**) Let D denote a database of trajectories and $q = (L, K)$ denote a query over D where L denotes the set of location points and K denotes a set of keywords being queried. A trajectory cover of q over D is a minimal subset of activity trajectories, say G, that covers the maximum number of query keywords in $q.K$ and have the smallest trajectory set query distance (TSQD) with respect to all the query points $q.L$, i.e., $CT(q,D) = \{G \mid G \subseteq D, \nexists G' \subset D, |G'| < |G| \wedge TSQD(G',q) < TSQD(G,q) \wedge |TSK(G',q)| > |TSK(G,q)|\}$.
In our running example of Figure 1, for query $q = (L, K)$, $q.L = \{q1, q2, q3, q4\}$ and $q.K = \{a, b, c, d, r, w\}$, we have $G = \{t_2\}$, since G covers all the query keywords $q.K$ and have $TSQD$ value as 22.12.

Lemma 1: Given a trajectory dataset D and a query q, finding the trajectory cover of q is an NP-hard problem.

Proof: This lemma says that the problem of finding a minimal subset $CT(q,D) \subseteq D$ for a given user query q over the trajectory dataset D is NP-hard. One way to prove the lemma is by reduction from the weighted set cover problem [4]: An instance of the weighted set cover problem consists of a universe $U = \{1, 2, ..., n\}$ of n elements and a family of sets $S = \{S_1, S_2, ..., S_m\}$, where $S_i \subseteq U$ and each S_i is associated with a positive cost C_{S_i}. The decision problem is to decide if we can determine a set F of subsets of S such that $\cup_{S_i \in F} S_i = U$ and such that its cost $\sum C_{S_i}, S_i \in F$ is minimized.

To reduce weighted set cover problem to the trajectory cover problem, we observe that $U = q.K$, each S_i corresponds to a set of keywords associated with a trajectory t_i, and the weight of S_i is $TQD(q, t_i)$. Now, it is easy to show that there exists a solution to the weighted set cover problem if and only if there exists a solution to query $CT(q,D)$. ■
Given that the trajectory cover problem is NP-hard, in this paper we solve this discrete optimization problem by developing a Greedy algorithm. Our goal is to develop an efficient Greedy algorithm that can find the best approximate trajectory cover(s) for each query. Note that by our trajectory cover definition, there may exist more than one trajectory covers for a given query.

B. Solution Overview

There are several ways to approach the problem of finding trajectory covers of a given query. Given that the trajectory cover is defined based on a minimal subset of D that have both maximal keyword coverage and the maximal spatial relevance, we can either approach this problem by sorting trajectories by one of the criteria, which give us two baseline approaches. However, we show that our greedy algorithm outperforms either of these baseline methods. Furthermore, in order to speed up the trajectory cover computation, we introduce a novel filter to prune irrelevant trajectories prior to the CT computation. We give a detailed description of our solution approach in the next section.

III. Algorithms

In this section, we first present the two baseline algorithms, namely Keyword-based approach (KA) and Location-based approach (LA). The keyword-based approach optimizes textual relevance score with respect to the query but does not take into account the minimum spatial proximity. On the other hand, the location-based approach decides the inclusion of trajectories in the CT of a query solely based on the spatial distance proximity and does not consider maximal textual relevance. In contrast, our greedy algorithm takes into account of both spatial and textual relevance by finding the smallest subset of trajectories, which maximizes the keyword coverage and the spatial proximity.

A. Keyword-based Algorithm (KA)

Input: D: Trajectory dataset
q: (K, L)
Output: G: Trajectory cover

$QK = q.K$;
$G = \phi$;
$\forall t \in D$, if $t.k \cap QK = \phi$, Remove t from D ;
repeat
  Find a trajectory $t \in D, t \notin G$ and $|t.K \cap QK|$ is maximum and $TQK(t,q) \cap QK \neq \phi$;
  $G = t \cup G$;
  $QK = QK - t.K$;
until $TSQK(G) \supseteq q.K$;
return G ;

Algorithm 1: Keyword-based Algorithm

First baseline algorithm uses inverted list index for its working. It filters out all trajectories in the database D that do not contain any keyword mentioned in the query $(q.K)$. Then for the remaining candidates, we sequentially process each trajectory in the order of maximum keyword relevance

with respect to the query keywords, i.e. a trajectory having largest number of query keywords is considered first for CT. We initialize the resultant group to empty (line 2) and add a most keyword relevant trajectory in the result set in each iteration (line 5-6). Once the result set G has trajectories that together cover the query keywords, it is returned at the output (line 8). Note that in cases when we obtain more than one trajectory covering the same number of query keywords then we choose one trajectory out of them randomly for inclusion in G. Consider the example given in figure 1. We will obtain either $G = \{t_4\}$ or $G = \{t_2\}$, since t_2 and t_4 both covers all the query keywords.

B. Location-based Algorithm (LA)

Input: D: Trajectory Data
q: (K,L)
Output: G: Trajectory cover

1 $G = \phi$;
2 $QK = q.K$;
3 $\forall t \in D$, if $t.k \cap QK = \phi$, Remove t from D ;
4 **repeat**
5 Find a trajectory $t \in D, t \notin G$ with the shortest $TQD(t, q)$ value;
6 **if** $TQK(t, q) \cap QK \neq \phi$ **then**
7 $G = G \cup t$;
8 $QK = QK - t.K$;
end
until $TSQK(G) \supseteq q.K$;
9 return G

Algorithm 2: Location-based Algorithm

LA algorithm (Algorithm 2) as like KA algorithm first prunes out trajectories that do not contain any query keyword $q.K$ (line 3). Then it finds trajectories in the order of their TQD value (line 5). We use the algorithm proposed in [2] for this step. The algorithm returns top-k nearest trajectories from a given set of location points. We extend it to an incremental algorithm version to retrieve a trajectory at a time in the order of their TQD value. We add a trajectory t to the resultant trajectory cover G, if t covers any new query keyword not yet covered by G (line 6-7).

In our example (figure 1), this algorithm will first include t_3 in G as it has the smallest distance of 15.38 from query points. Since, $t_3.k = \{a, b, c, d\}$, therefore $QK = \{r, w\}$ (according to line 8). Now, the algorithm will select trajectory t_1 as it has next smallest distance of 15.54 from query points and also covers QK. Thus, $G = \{t_1, t_3\}$ covers all the query keywords and has its $TSQD(G, q)$ value as 30.92. Hence, the algorithm stops iterating over trajectories and returns the trajectory cover G as the result.

C. Greedy algorithm (GA)

1) Promising Value Filter and Its Efficient Implementation: Like the baseline KA and LA algorithms, GA also filters out trajectories not having any query keyword. However, unlike the baseline algorithms, which optimize only one dimension, either spatial or textual, for the remaining candidate trajectories, GA introduces two novel filters: the promising value filter and the lower-upper bound filter. We compute the *promising value* for each candidate trajectory using both spatial and textual similarity of the trajectory with respect to the user query q. Then we sort all the candidate trajectories by the increasing order of their promising values. For our running example (Figure 1), GA computes CT as $G = t_2$ with a $TSQD(G, q)$ value 22.12. It can be seen that CT computed by GA is compact and spatially close, compared to KA and LA. For example, KA algorithm computes compact CT but with high $TSQD$ value.

Definition 6: **Promising Value (PValue)** Given a user query q, the promising value of a trajectory t is defined as a ratio of its spatial distance from the query points $q.L$ to the number of new query keywords it covers with respect to q:

$$PValue(t, q) = \frac{TQD(t,\ q)}{|TQK(t,\ q)\ \cap\ QK|}$$

Here QK is the set of query keywords not yet covered by the trajectories in the partially computed trajectory cover G, i.e., $QK = q.K - TSQK(G)$.

The notion of $PValue$ allows search for a smallest set of trajectories covering more query keywords even if they might be at slightly farther distances from the query points. A trajectory with the minimum $PValue$ is considered as the most promising trajectory in the candidate set of trajectories to be included in the CT of q.

The naive approach for employing the promising value filter is to compute the $PValue$ for every trajectory in the trajectory database D and sort them according to the increasing order of their $PValue$. Then we can compute CT by examining the most promising trajectory t every time in a greedy manner to compute the maximum coverage of query keywords. When the size of the trajectory database D is large, this approach is not scalable due to its large computation cost. Therefore, we design an efficient approach by using a lower bound lb and upper bound ub of promising values. Conceptually lb defines the minimum expected $Pvalue$ of a trajectory and ub defines the maximum $Pvalue$ of any trajectory. A trajectory with its lb value greater than the ub value can be classified as non-relevant trajectory.

We create three data structures, namely R-tree, a global heap H and hash table C in our algorithm. An R-tree indexes trajectories in database D to enable efficient search of nearest trajectory places from each of the query points $q.l_i$. The heap H controls the schedule of trajectory places retrieval for given query points. Each entry in the heap is a quadruple $(key, trajectories, q.l_i, d)$; where $q.l_i$ is a query point for which next nearest place say p_j at distance d is retrieved from the R-tree, and $trajectories$ is a list of trajectories passing through p_j. The key value key is computed as the ratio of spatial distance d to the maximum number of query keywords a trajectory in the trajectory

list covers. As an example, if one of the places p_j gets returned from the R-tree for a query point q_l with a distance value d, and t is a trajectory passing through p_j has x number of query keywords not yet covered by the current trajectory cover G, then the key value for t for query point q_l will be d/x. The hash table C stores the computed trajectories distances. Each time a entry with the minimum key is removed from the global heap H, it is stored in C and H is replenished with next nearest location distance of the corresponding query point. Therefore, C will have some trajectories with their partially computed TQD value, and some trajectories with their fully computed TQD value called as partially-explored and fully-explored trajectories, respectively. The upper bound ub is the minimum $PValue$ of fully-explored trajectories. The lower bound lb is defined as the minimum estimated $Pvalue$ for any trajectory in partially-explored or unexplored categories. Any partially-explored trajectory which have its lb more than the upper bound (ub) can not be a promising trajectory.

Property 1: For a given user query q, and a global heap H, the following equation for lower bound lb defines the actual lower bound for any trajectory t.

$$lb(t) = \sum_i \frac{Dist(q.l_i, t)}{|q.K \cap t.K|}$$

$Dist(q.l_i, t) = C(t)[q.l_i]$ if the hash table C has an entry of t for query point $q.l_i$, and $Dist(q.l_i, t) = H(q.l_i)$ otherwise. $H(q.l_i)$ denotes the current distance value for query point $q.l_i$ in heap H.

Proof: Let $t.p_j$ be a place in trajectory t at the minimum distance from a query point $q.l_i$. Now, if $t.q_j$ is not present in C then $Dist(q.l_i, t.p_j) \geq H(q.l_i)$, since elements from the global heap H are extracted out in non-decreasing order of distance. Similarly, if C has a entry for query location $t(q.l_i)$, then $C(t)[q.l_i] = Dist(q.l_i, t.p_j)$. As $q.K \cap t.K$ is the maximum number of query keywords that a trajectory can cover, i.e. $t.K \supseteq TQK(t, q)$, the formula given above defines the minimum $PValue$ for t. ■

2) Promising Filter-based Greedy Algorithm: Algorithm 3 presents our proposed GA approach. We first initialize our variables G (trajectory cover), H (global heap), and C (candidate set) with their initial values (lines 1-3). For our example figure 1, after line 3 of the algorithm we will have $H = [\,]$, $G = \{\}$, and $C = [\,]$. The lb and ub initial values will be ∞ as none of the trajectory has been accessed yet. Lines 4-6 in the algorithm show how an entry for heap is prepared and inserted. In our example, after completion of retrieval of the nearest places for each of the query points from R-tree (lines 4-6), the state of H will change to $H = [(0.235, t_2, q_1, 1.41), (0.705, t_3, q_2, 2.82), (0.56, t_3, q_4, 2.24), (0.94, t_1, q_3, 2.82)]$.

The trajectory cover is computed in lines 7-17 of the

Input: D: Trajectory Data
q: (K,L)
Output: G: Trajectory cover

1 $G = \phi$;
2 $QK = q.K$;
3 Initialize global heap H and a candidate trajectory set C ;
4 **for** *($q.l_i \in q.L$)* **do**
5 T_L = A list of trajectories passing through p_j at the shortest distance d from $q.l_i$;
6 Insert tuple $(key, T_L, q.l_i, d)$ in heap H, where $key = \min_i(\frac{d}{|t_i.k \cap QK|}), t_i \in T_L$;
end
7 **repeat**
8 Extract minimum key value tuple $< key, T_L, q.l_j, d >$ from H;
9 **foreach** $t \in T_L$ **do**
10 **if** *($d < C(t, q.l_j)$)* **then**
11 $C(t, q.l_j) = d$;
end
end
12 T_L = A list of trajectories passing through p_j at the shortest distance d from $q.l_j$;
13 Insert tuple $(key, T_L, q.l_j, d)$ in heap H;
14 **while** *($q.K \supset TSQK(G)$ and $\nexists t', t'.lb < ub$)* **do**
15 $G = G \cup t$, where $PValue(t, q) = ub$;
16 Remove t from C;
17 $QK = QK - t.K$;
end
until *($q.K \subset TSQK(G)$)*;
18 return G

Algorithm 3: Greedy Algorithm

algorithm. As mentioned earlier, C is a hash table data structure. It uses trajectory-id as a key value and an array of size equal to the number of query points as its element value. $C(t, q.l_i)$ denotes the spatial distance of the nearest valid place of t. Whenever we remove an entry from the global heap H (line 8), we update the corresponding entry in C with the distance value d if it is not yet present there in C (lines 9-11). We also replenish the global heap with the next nearest distance value of the corresponding query point (lines 12-13). For our example, in the first iteration of the repeat loop of line 7 when we reach to line 13, we will have $C = (t2 : [(q1, 1.41)])$, as the place p_{21} returned by q_1 has the minimum key value of 0.235. Now, the value of H becomes $H = [(0.56, t_3, q_4, 2.24), (0.705, t_3, q_2, 2.82), (0.94, t_1, q_3, 2.82), (0.743, t_1, q_1, 2.23)]$. The ub remains unchanged as there is still no fully-explored trajectory in C, whereas $lb(t_2)$ becomes $lb(t_2) = (1.41 + 2.82 + 2.82 + 2.24)/5 = 1.858$. In this iteration the lines 15-17 will not be executed as $ub > lb(t_2)$.

Finally, when there is no trajectory in partially-explored or unexplored category with its lb value less than the ub value, a trajectory with the current ub value is declared as the most promising trajectory and is included in the trajectory cover G (lines 14-15). After a trajectory t is included in G (line 16), the query keywords that remain yet to be covered get

reduced (line 17). Set G is returned at the output, whenever $TSQK(G)$ covers all the query keywords (line 18). In our example case (Figure 1), trajectory cover G includes trajectory t_2 first even though trajectory t_3 becomes fully-explored earlier than t_2. This is due to the reason that $lb(t_2)$ has value 2.79 which is smaller than the ub value 3.845 as currently set by t_3. Trajectories t_1 and t_4 get their lb more than the ub as set by t_3 and hence not analyzed further and ruled out for inclusion in G. Trajectory t_2 covers all the query keywords, hence algorithm returns $G = \{t_2\}$ as a trajectory cover with $TSQD$ value of 22.12.

IV. Experimental Evaluation

Datasets: We have conducted extensive experiments to evaluate the proposed methods. The synthetic trajectory datasets $D1$ and $D2$ used for experimental purpose are derived from trajectory data and geo-spatial locations respectively, by annotating the keywords with them. The dataset $D1$ has been generated from the TDrive project data. The trajectories in $D2$ are generated by distributing geo-spatial locations (of Flickr images) in each trajectory such that the number of location vs trajectories follows Gaussian distribution. Further, keywords are assigned to locations using Gaussian Distribution whereas locations to keywords assignment follows Power Law Distribution. Table I describes some useful parameters ($\mid K \mid$ = number of keywords, $\mid L \mid$ = number of locations, $\mid D \mid$ = number of trajectories in dataset, L/K = number of locations per keyword, K/L = number of keywords per location and ATL = Average Trajectory Length) for datasets $D1$ and $D2$.

DataSet	$\mid K \mid$	$\mid L \mid$	$\mid D \mid$	L/K	K/L	ATL
$D1$	500	5500	1165	406	37	250
$D2$	250	39075	200000	535	3	20

Table I: Description of Datasets

Experimental Setup: We have defined four different classes of queries on the basis of popularity of keywords and locations i.e., *Popular Locations Popular Keywords (PP), Unpopular Locations Popular Keywords (UP), and Popular Locations Unpopular Keywords (PU), and Unpopular Locations Unpopular Keywords (UU)*. A keyword is defined as popular if frequency of number of locations a keyword is associated with, is more than the average number of locations associated with a keyword in the data set. Similarly, a location is popular if frequency of keywords associated with that location is more than the average number of keywords associated with a location in the data set. We have generated different sizes of queries for each class.

Evaluation: We have defined three evaluation metrics:

- *group distance:* describes the quality of the query result in terms of spatial distance from query locations.
- *average execution time:* measures the actual clock time an algorithm takes to evaluate a query.
- *group size:* defines the compactness of a trajectory cover i.e., a CT with small size is preferable over a CT with large size.

We have created 100 different query instances for each query type and have reported the average value for each performance metric. The experiments are conducted on a PC with Intel Core-i3 CPU 2.10G Hz and 4.00 GB RAM. The operating system is Ubuntu11.04. All the algorithms are implemented in Java on Eclipse 3.5.2 platform with JDK 1.6.0_24.

A. Results

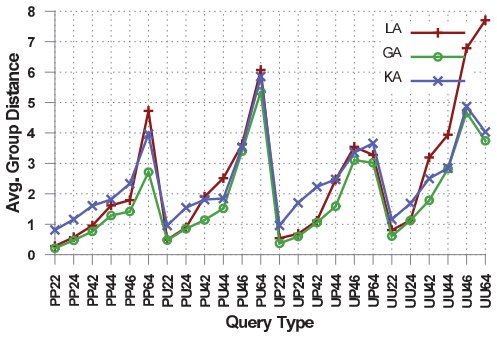

Figure 2: Effect of Query Type on Group Distance (Dataset TDrive)

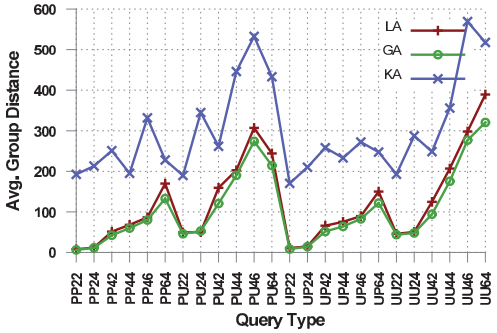

Figure 3: Effect of Query Type on Group Distance (Dataset Flickr)

1) Effect of Query Type on Group Distance ($TSQD$): Our experimentation results (Figures 2 and 3) show that the performance of Greedy algorithm (GA) is always better than both location algorithm (LA) and keyword algorithm (KA) in both the datasets. However, in case of dataset $D1$, we observe that the difference between group distance returned by GA is not much as compared to group distance returned by other two naive approaches. This is because of the reason that dataset $D1$ is very compact and ATL is also large. For dataset $D2$, KA clearly performs worst and LA returns same group G as GA quite often. This is due to the number of keywords per location is small in case of $D2$. However, our

next graphs for metric group size show GA always better than LA.

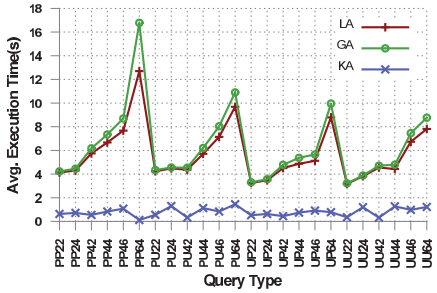

Figure 4: Effect of Query Type on Computation Time (Dataset TDrive)

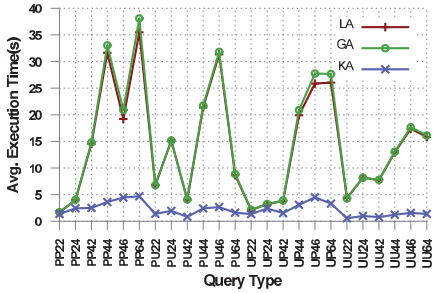

Figure 5: Effect of Query Type on Computation Time (Dataset Flicker)

2) Effect of Query Type on Computation Time: Experimental results (Figures 4 and 5) show that the computation time overhead for query evaluation using GA is minimal as compared to LA and KA algorithm. The worst case overhead for $D1$ dataset of GA is 4 seconds as compared to LA. On the other hand the worst case overhead for dataset $D2$ is around 2.0 seconds for query type $UP64$. This is due to the reason that GA analyzes more candidate trajectories before including a trajectory in G. The number of candidate trajectories is more in case of dataset $D2$, because average number of keywords per location is 3 (From table I). The KA computation time is always very small but its performance in terms of group distance is very poor as also seen in our previous graph for metric group distance (Figure 3).

3) Effect of Query Type on Group Size: (Figures 6 and 7) shows experimental results for group size metric. GA algorithm always performs better than LA algorithm. On the other hand KA out performs both LA and GA in group size. It can be seen in most of the cases KA returns single trajectory group in TDrive dataset. This is due to the reason that KA algorithm prefers trajectories on the basis of activity covered by a trajectory only. Since, TDrive dataset is dense (K/L is 37 and ATL is 250) therefore we obtain trajectory cover of size one in most cases for KA. On the other hand KA performs very poorly for metric group distance.

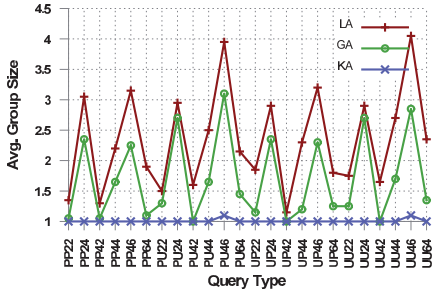

Figure 6: Effect of Query Type on Group Size (Dataset TDrive)

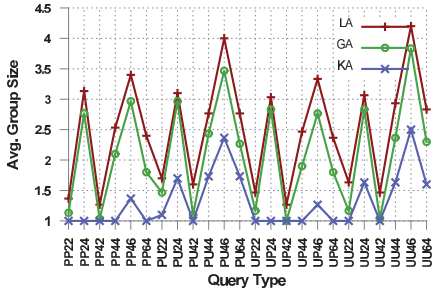

Figure 7: Effect of Query Type on Group Size (Dataset Flicker)

B. Remarks

We have shown that GA algorithm consistently performs better as compared to the baseline algorithms KA and LA, although LA appears to perform equally well in case of sparse trajectory data for group distance metric but the resulting set of trajectories produced by LA always have many more trajectories in CT than that of GA, showing that GA is a much better approximation algorithm for the trajectory cover problem. Similarly, KA appears better in terms of group size and computation time, but the trajectories produced by KA are often at far distances from the query points compared to that of GA, showing the lack of maximizing spatial relevance in KA.

V. Related Work

The most relevant research to this work is trajectory retrieval based on spatial proximity or based on both spatial and textual relevance. The work in [2] finds the top-k trajectories with the minimum aggregated distance from the set of query points. However, their work focuses on trajectory queries without any textual data. [3] is one of the first work on searching trajectories for trip recommendation. Their algorithm finds the trajectory having both spatial closeness and higher textual relevance with respect to a query (i.e., set of locations and keywords). However, their algorithm often returns many more trajectories since they treat trajectories as independent and they only optimize the spatial and textual relevance on per trajectory basis.

In contrast, our promising value based Greedy algorithm returns the trajectory cover with respect to a query, i.e., the smallest subset of trajectories, which maximizes textual relevance and minimizes spatial proximity on per trajectory group.

A lot of work has been done for indexing spatial temporal data [5], [6], [7], [8], [9]. These indexes help to retrieve k nearest neighbors or a set of objects within a given area efficiently. They also support retrieving trajectories which follows certain complex pattern [10], [11]. A lot of work has also been done on trajectory clustering[12], [13], [14], [15]. These work deal with only spatial data and do not consider textual dimension at all.

There have also been work on indexing of trajectory data e.g., a signature file-based index structure ($IR^2 - tree$) [16] that supports spatial-keyword queries retrieving a set of spatial objects covering the query keywords; a hybrid index structure integrating $R^* - tree$ and bitmap index [17] that also supports retrieval of spatial objects having a set of keywords; a hybrid index structure integrating $R - tree$ and inverted index ($IR - tree$) [18] that supports retrieval of top-k rank trajectories for a given location and a set of keywords; a Spatial Inverted Index structure [19] organizing spatial objects related to each keyword separately using $R-tree$ [20] and supports collective spatial keyword query [1] to retrieve spatially closest objects covering the query keywords; and a hybrid index structure integrating B^+-tree with spatial location codes derived from location coding mechanism ($B^{ck} - tree$) [21], which supports retrieval of top-k trajectories covering all the query keywords that are closest to the given query location. These indexes can be used to search CT faster and is complementary to our work.

VI. Acknowledgement

This work is supported by Department of Science and Technology India, and Department of Information Technology India.

VII. Conclusions and Future Work

We have introduced the notion of trajectory cover and showed that even though the trajectory cover problem is NP hard, we can have an efficient Greedy algorithm for processing trajectory cover queries. We introduce a novel promising value filter and a fast implementation of the filter using the expected lower bound and upper bound of promising values. This optimization allows us to prune out irrelevant trajectories that do not contribute to the trajectory cover of the query prior to the CT computation. We conduct an extensive experimental study on different datasets and show that Greedy approach performs better than the two baseline approaches in terms of quality and computation time. We are interested in exploring trajectory index based optimization in the future work and we are also interested in developing scaling techniques to allow our GA algorithm to run over much larger set of trajectories.

References

[1] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD*, 2011, pp. 373–384.

[2] L.-A. Tang, Y. Zheng, X. Xie, J. Yuan, X. Yu, and J. Han, "Retrieving k-nearest neighboring trajectories by a set of point locations," in *SSTD*, 2011, pp. 223–241.

[3] S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis, "User oriented trajectory search for trip recommendation," in *EDBT*, 2012, pp. 156–167.

[4] U. Feige, "A threshold of ln n for approximating set cover," *J. ACM*, vol. 45, no. 4, pp. 634–652, Jul. 1998.

[5] V. Chakka, A. Everspaugh, and J. Patel, "Indexing large trajectory data sets with seti," in *CIDR*, 2003.

[6] P. Cudré-Mauroux, E. Wu, and S. Madden, "Trajstore: An adaptive storage system for very large trajectory data sets," in *ICDE*, 2010, pp. 109–120.

[7] D. Pfoser, C. S. Jensen, and Y. Theodoridis, "Novel approaches in query processing for moving object trajectories," in *VLDB*, 2000, pp. 395–406.

[8] Y. Tao and D. Papadias, "Mv3r-tree: A spatio-temporal access method for timestamp and interval queries," in *VLDB*, 2001, pp. 431–440.

[9] Y. Theodoridis, T. K. Sellis, A. Papadopoulos, and Y. Manolopoulos, "Specifications for efficient indexing in spatiotemporal databases," in *SSDBM*, 1998, pp. 123–132.

[10] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations: an efficiency study," in *SIGMOD*, 2010, pp. 255–266.

[11] M. R. Vieira, P. Bakalov, and V. J. Tsotras, "Querying trajectories using flexible patterns," in *EDBT*, 2010, pp. 406–417.

[12] R. Agrawal, C. Faloutsos, and A. N. Swami, "Efficient similarity search in sequence databases," in *FODO*, 1993, pp. 69–84.

[13] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*, 2005, pp. 491–502.

[14] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *SIGMOD*, 2007, pp. 593–604.

[15] M. Vlachos, D. Gunopoulos, and G. Kollios, "Discovering similar multidimensional trajectories," in *ICDE*, 2002, pp. 673–684.

[16] I. De Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008, pp. 656–665.

[17] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *ICDE*, 2009, pp. 688–699.

[18] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, vol. 2, no. 1, pp. 337–348, 2009.

[19] J. a. B. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørvåg, "Efficient processing of top-k spatial keyword queries," in *SSTD*, 2011, pp. 205–222.

[20] D. Papadias, P. Kalnis, J. Zhang, and Y. Tao, "Efficient olap operations in spatial data warehouses," in *SSTD*, 2001, pp. 443–459.

[21] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang, "Efficient spatial keyword search in trajectory databases," *CoRR*, vol. abs/1205.2880, 2012.