Interactive Trip Planning Using Activity Trajectories

Sheng Wang* Zhifeng Bao* J. Shane Culpepper* Timos Sellis†
Mark Sanderson* Munkh-Erdene Yadamjav*
*School of CSIT, RMIT University, Australia
†School of Software and Electrical Engineering, Swinburne University of Technology, Australia
*firstname.surname@rmit.edu.au +tsellis@swin.edu.au

ABSTRACT

We present an interactive trip planning system called @FINDER which uses an exemplar trajectory query to find the most related top-k spatial-textual trajectories. @FINDER is implemented to support various degrees of user information needs for trip planning. For users with zero knowledge about places to travel, @FINDER provides a heatmap of popular points of interest (POIs) as well as popular activities from a trajectory database. The system helps users quickly explore the places, and helps formulate an *exemplar trajectory query*, which specifies preferred places to go and activities of interest. Then @FINDER provides efficient query processing of the top-k related spatial-textual trajectories using a new approach to spatial-textual trajectory indexing recently developed at RMIT University. For each of the top-k results found in the form of a set of POIs and activities, @FINDER further computes the optimal route (in term of the travel time) covering all of the POIs, and returns an *album* to the user. Lastly, users can further interact with @FINDER by adding or deleting POIs/activities in the original exemplar query, and the system will update the results in a timely manner.

Keywords

Activity Trajectory Search; Trip Planning; Interactive Exploration

1. INTRODUCTION

Trip planning [4, 5] is an important tool for people who want to make the most of their travels. Most people find it difficult and time-consuming to plan a trip, and it can be even more difficult to reliably match personal interests to Points of Interest (POIs) in an unfamiliar location. In this work, we present a tool to help users plan more reliable trips using historical trajectories in a social network.

In recent years, a large number of trajectories composed of a set of points which record people's daily movements and activities have been created in social networking repositories. Consider the example in Figure 1. Jack visited Los Angeles and routinely posted on Facebook during his trip. The posts indicate that he went to three different places, and photos with each activity were shared.

ADCS '16, December 05 - 07, 2016, Caulfield, VIC, Australia
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4865-2/16/12...$15.00
DOI: http://dx.doi.org/10.1145/3015022.3015030

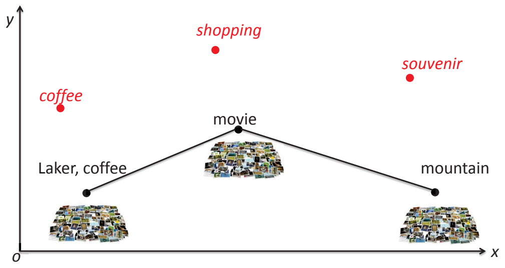


Figure 1: Example of a Trajectory (black) and a Query (red).

We refer to an album recording a user's movement as an album-annotated activity trajectory $\mathcal{T}$ (or, more simply, a "trajectory"), e.g., Jack's trajectory $\mathcal{T}$ is three ordered points and each point contains keywords. Such trajectories can play a significant role in guiding tourists for their trip planning.

EXAMPLE 1. *George will visit Los Angeles for the first time. There are three places he would like to visit initially – the hotel reserved in advance, the airport, and Hollywood. He also has a few favorite activities such as "drinking good coffee", "shopping" and "looking at souvenirs", which can be suggested for each place. If these preferences are defined as a query Q, related trajectories such as Jack's trip, which cover similar locations and activities can be used to suggest further refinements, and can help George plan a more rewarding trip.*

We refer to the query in Example 1 as a *Top-k* ***E****xemplar* ***T****rajectory* ***Q****uery* (*ETQ*), which is defined as:

DEFINITION 1. ***(Top-k Exemplar Trajectory Query)*** *Given a trajectory database $D = \{\mathcal{T}_1, \ldots, \mathcal{T}_{|D|}\}$ and query Q, a trajectory search retrieves a set $R \subseteq D$ with k trajectories such that: $\forall r \in R, \forall r' \in D - R, \hat{S}(Q, r) > \hat{S}(Q, r')$, where $\hat{S}(Q, r)$ is the relevance score between the query Q and a trajectory r that will be discussed in Section 2.2.*

The query Q consists of points with textual descriptions of activities. We observe that previous work [6, 8] only allows users to specify their preferred activities at the trajectory level, while users are not allowed to specify their preferred activities at the POI level. This may result in coarse-grained result matching. Moreover, we find that existing commercial systems such as Google Trips[1] and Triphobo[2] only provide trip recommendations based on a destination city, while (advanced) search as specified in the above example does not currently exist.

[1]https://www.google.com/trips/
[2]https://www.triphobo.com/

Therefore, we propose Exemplar Trajectory Queries (*ETQ*) to enable finer-grained specifications on places and activities. Moreover, we introduce a new similarity function, and an efficient processing algorithm to integrate textual and spatial similarity. Then we develop a system called @FINDER to help users plan a personalized trip by specifying locations and activities directly on top of a Google map. The prototype of @FINDER is available at http://115.146.90.65/finder/, whose main features are:

- An interactive tool to create queries. Even with no prior knowledge, users can quickly find popular activities and places in the city using @FINDER.
- *ETQ* is supported using a pointwise similarity function which incrementally computes the search results efficiently, and allows efficient updates.
- Final results are presented as the best matching route, which covers all points along with the travel distance and/or travel time.
- Query refinement by adding or deleting points in each search iteration is also supported.

2. OVERVIEW

2.1 Architecture

Figure 2 shows the overall architecture for @FINDER. The interface has two main functions. First, a user submits an initial query using an overlay on a Google map, and the query is sent to the server to compute the top-k trajectory results, which are sent back and displayed on the Google map in the user's browser. When the user chooses to view a trajectory, the corresponding *album* is fetched from the database and displayed. Second, an interactive exploration can then be conducted using the returned results to help the user refine their trip.

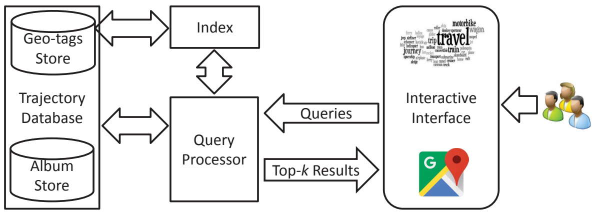


Figure 2: The system architecture of the @FINDER system.

2.2 Pointwise Similarity

Before describing our processing framework, we first give the definition of similarity $\hat{S}(Q, \mathcal{T})$ between a query and a trajectory. For each query point q_i in Q, the similarity $\hat{S}(q_i, p_j)$ is computed with every point p_j in the trajectory $\mathcal{T}$. The maximum value is represented by $\hat{S}(q_i, \mathcal{T})$, and the sum of the query points is the overall similarity $\hat{S}(Q, \mathcal{T})$.

DEFINITION 2. ***(Trajectory Similarity)*** *The trajectory similarity between $\mathcal{T}$ and Q is a sum of point-trajectory similarities between $\mathcal{T}$ and each point in Q, normalized by $|Q|$:*

$$\hat{S}(Q, \mathcal{T}) = \sum_{q_i \in Q} \hat{S}(q_i, \mathcal{T}) / |Q|. \tag{1}$$

Here $\hat{S}(q_i, \mathcal{T})$ is the similarity between a query point q_i, and a trajectory $\mathcal{T}$, and defined as:

$$\hat{S}(q_i, \mathcal{T}) = \max_{p_j \in \mathcal{T}} \left\{ \hat{S}(q_i, p_j) \right\} \tag{2}$$

Algorithm 1: Two-level Threshold Algorithm (ETQ-2TA)

```
Input: Trajectory database $D$, query $Q$, $MT$
Output: Top-$k$ result set $RS$
$it \leftarrow 0$; $RS \leftarrow \emptyset$;          ▷ $it$: initial number of iterations
while it $<$ it$_{\max}$ do
    foreach $q_i \in Q$ do
        $R_{it}(q_i) \leftarrow ExploreSptial(q_i, it, D.P)$;
        $R_{it}(q_i) \leftarrow ExploreTextual(q_i, it, D.P)$;
    $C_{tra} \leftarrow \bigcup_{i=1}^{|Q|} Covered(R_{it}(q_i), MT)$;   ▷ $MT$:mapping table
    if $|C_{tra}| > k$ then
        $unseen\_UB = UB_{unseen}(D - C_{\text{tra}})$;
        $seen\_LB[] = \bigcup_{T \in C_{tra}} LB_{seen}(T)$;
        Sort $seen\_LB[]$ in decreasing order;
        if $seen\_LB[k] > unseen\_UB$ then
            $seen\_UB[] = \bigcup_{T \in C_{tra}} UB_{seen}(T)$;
            Sort $C_{tra}$ by $seen\_UB[]$ in decreasing order;
            foreach $T_i \in C_{\text{tra}}$ do
                Compute $\hat{S}(Q, T_i)$;   ▷ Refer to Definition 2
                if $|RS| < k$ then
                    Insert $T_i$ in $RS$;
                else
                    if $\hat{S}(Q, T_i) > RS.min$ then
                        Replace $RS.min$ with $T_i$;
                    if $RS.min > seen\_UB[i+1]$ then
                        break;
    $it$++;
return $RS$;
```

where $\hat{S}(q_i, p_j)$ is the similarity between two points q_i, p_j, which in turn is defined as:

$$\hat{S}(q_i, p_j) = \begin{cases} 0, & q_i.act \cap p_j.act = \varnothing \\ \alpha \cdot \hat{S}_S + (1-\alpha)\,\hat{S}_T, & \text{otherwise.} \end{cases} \tag{3}$$

Here $\hat{S}_T(q_i, p_j)$ is the text similarity and $\hat{S}_S(q_i, p_j)$ is the spatial proximity between two points, $q_i.act$ is the keywords attached to point q_i. The value $\alpha \in (0, 1)$ is used to adjust the relative importance of the spatial proximity and the textual similarity. We use the sum of the textual relevance of each term [7] to measure the textual similarity and the Euclidean distance to measure the spatial similarity. This results in our final combined similarity measure:

$$\hat{S}_T(p_i, p_j) = \sum_{t \in p_i.act} \gamma(t) \tag{4}$$

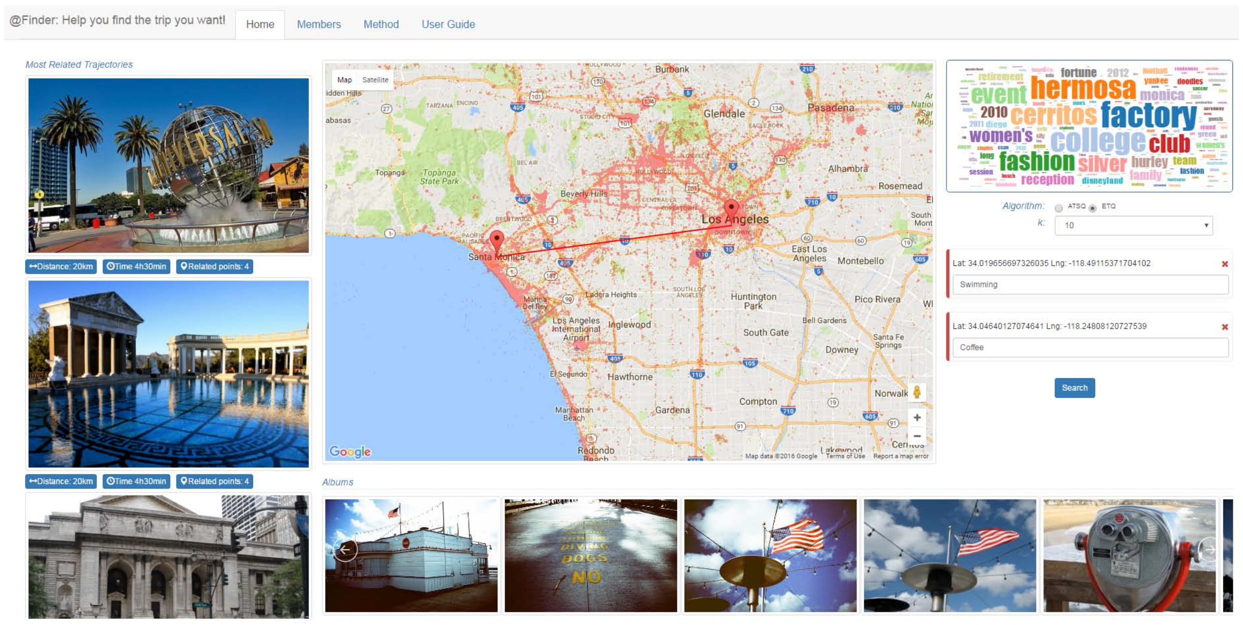
$$\hat{S}_S(p_i, p_j) = \frac{D_{max} - Euclidean(p_i, p_j)}{D_{max}} \tag{5}$$

where $\gamma(t)$ is the weight of term t, and we use a simple TF·IDF model [1] to compute $\gamma(t)$ in this paper. D_{max} is the maximum distance between any two unique points in geographical space.

2.3 Processing

Trajectory Database & Index. Storage is divided into two parts, the trajectory and the album. The main objective is to increase the throughput during query processing. Since the album is not taken into consideration when processing the query, the album will only be accessed and displayed when the top-k results are returned. Three basic data structures are maintained:

(1) An *inverted index*, where the key is a keyword, and the value



Algorithm: ATSQ [✓ ETQ]
k: 10
Lat: 34.019656697326035 Lng: -118.49115371704102
Swimming
Lat: 34.04640127074641 Lng: -118.24808120727539
Coffee

Figure 3: The @FINDER interface with a simple query scenario.

is a sorted posting list of points whose activity contains the keyword.

(2) A *Z-curve*, which is used to expand the spatial dimension based on an incremental range query [7].

(3) A *mapping table MT*, which maps an activity point to the parent trajectory.

Query Processor. We use a filter-refine processing framework. We filter the impossible trajectories from consideration, and refine based on the remaining candidates. We first extended the framework of Chen et al. [3] as a baseline method (**ILA**) with a gap-bounded optimization (**ILA-Gap**) to eliminate some of the performance bottlenecks we encountered. Further, we propose a two level algorithm which removes repetitive scanning over the trajectory points (**ETQ-2TA**) as shown in Algorithm 1, which we now summarize.

An *ETQ* finds the k closest trajectories for an exemplar query, where each trajectory is composed of a tuple of points. Essentially, the trajectory problem can be decomposed into finding the closest points for every point in the query, which is also reflected in our similarity computations. A pointwise processing algorithm is derived to solve the *ETQ* problem by incrementally expanding the search range until the top-k results are found. The key idea of our approach can be summarized into three steps:

*Step 1 (**Expansion**):* In lines 2-6, we fetch the posting lists for all keywords in the query, and load the points by setting a cursor in each posting list (*ExploreTextual()*) from the point sets $D.P$. Meanwhile, for each location in a query, we use a z-curve to do an incremental range query, and load potential candidates (*ExploreSpatial()*). All of the loaded points are stored in $R_{it}(q_i)$. Next, we find the candidate set C_{tra} using the mapping table *MT*. The trajectory containing the candidate is found through mapping table.

*Step 2 (**Bounded Computation**):* In lines 8-11, based on the cursors in the posting lists and z-curves, the upper bound for all unseen trajectories UB_{unseen}, and the lower bound of the k-th trajectory $seen\_LB[k]$ in the candidate set can be computed. When the lower bound is not less than the upper bound, the candidate set expansion stops, and *Step 3* is initialized. Otherwise *Step 1* is repeated, and expanding continues with new results based on posting list and the z-curve. To get the lower bound of trajectory T, we sum up all of the available similarities of T in all of the ranked lists $R_{it}(q_i)$, and the upper bound of the unseen trajectories can be computed by summing up the minimum similarity from each of the ranked lists, which is similar in spirit to ideas proposed by Zhang et al. [7].

*Step 3 (**Refinement**):* In lines 12-22, the top-k results are selected from the candidate set by computing the upper bound for each candidate trajectory in candidate set, and ranking the results in descending order (line 13). Then the final similarity is computed between each candidate trajectory and Q. If the similarity of the k-th result $RS.min$ is greater than the upper bound of the next-to-compute trajectory $seen\_UB[i+1]$, then the algorithm can terminate and return the top-k results.

2.4 Interactive Exploration

@FINDER also supports interactive exploration. When the top-k results are returned to user, the user can refine the query to improve the final search result. A user can add points from previously returned top-k trajectories to create a new exemplar query using new POIs and activities, and can also remove uninteresting points. This iteration can continue until the user finds the best combination of POIs and activities.

Case 1: A user can add points from previous results to update the query, and get a route which covers the query, or remove points that are not appropriate.

Case 2: A user can edit the returned trajectories directly using travel distance and time budget.

3. THE @FINDER SYSTEM

First, the interface of the system is shown. Then, two different query scenarios are shown to sketch the processing flow. Finally, we test the response efficiency using realistic dataset.

3.1 Interfaces

Figure 3 shows the interface for @FINDER. The interface is divided into four distinct modules.

Central Map. Central map (A) uses the Google Map API to show the results, including the route covering the trajectories on the road map. The user can also label desired locations directly using the heat map. So the module can be summarized as follows: (1) location labeling; (2) result presentation; and (3) query refinement.

Query Input. The right side (B) shows the keyword input interface including the location and activity, algorithm choice, and number of results k. *ETQ* and *ATSQ* [8] queries are currently supported. The word cloud suggests new keywords to users when a location is selected on the map. The word cloud suggestions include the most popular activities surrounding a POI. Users can also manually enter activity keywords directly.

Result List. The left side (C) displays the top-k results and summarizes the travel distance and time. The number of points related to the query is also displayed. Only trajectories highlighted in the result list are displayed in the map to improve presentation clarity.

Album Views. The bottom part (D) shows returned trajectory's details, including photos and textual descriptions for the trajectory.

3.2 Query Scenarios

Zero-knowledge Query. A user selects POIs in the map overlay. The selected points result in an updated word cloud of activities, which can be used to refine the query. Alternately, the user can choose activities from the word cloud first and use the updated heatmap of POIs in the map overlay to refine the query.

Some knowledge. Consider George's search in Example 1. First, George labels three destinations (red) in the map, and adds preferred activities for each point using the keyword input form. He sets the number of returned results to 10, and uses the ETQ algorithm. After clicking the search button, the left side will show the top-k results, including travel distances and times for the related points. Then George clicks the first result, and the trajectory (blue) with 4 points is displayed in the map overlay. Each point has a corresponding album in the bottom pane containing more details.

3.3 Response Efficiency

Here we briefly report preliminary experimental results for our proposed algorithms.

Dataset. An activity trajectory dataset from Foursquare is used in the experiments: Los Angeles (LA) [2, 8]. The dataset includes 31,557 trajectories, each trajectory has an average length of 6.83 locations, and the average number of activities per trajectory is 14.67.

Queries. Trajectory queries are randomly sampled from the datasets. To generate queries with a different number of points and activities, we randomly choose sub-trajectories from the query set and sample partial activities from dataset. All experimental results are averaged by running 100 queries. All algorithms are implemented in Java and ran on a PC with a 3.30GHz CPU and 8GB RAM under Ubuntu 14.04.

Algorithms Tested. We compare the methods described in Section 2.3 for **ETQ** . (1) **ILA** is a baseline extended from the spatial-only case Chen et al. [3]. (2) **ILA-Gap** is an improvement that uses the gap-based dynamic expansion method introduced in Section 2.3. (3) **ETQ-2TA** is our new solution with uses the two-level threshold algorithm to further enhance performance.

Performance Evaluation. A variety of different parameter sweeps have been performed to compare the algorithms, and will be exhaustively analyzed in future work. One example of the performance improvements achievable using our new approach is shown in Figure 4. For the two parameters $|Q|$ (number of locations) and $|q.act|$ (number of activities), **ETQ-2TA** has the best performance. **ETQ-2TA** is up to 2x faster than the baseline method **ILA**, and can be 5x faster in many cases. The number of activities ($|q.act|$) has the greatest effect on performance. As more keywords are added, the number of possible candidates in the intermediate pruning steps also increases. In summary, our approach shows great promise for spatial-textual trajectory search.

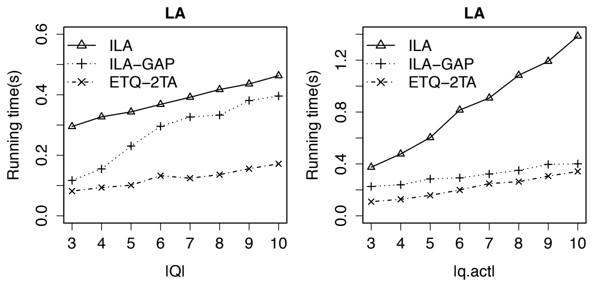


Figure 4: The effect of the number of locations in the query ($|Q|$), and the number of activities ($|q.act|$) on the total running time. Our two-level thresholding algorithm is more robust to longer trajectories, and algorithmic improvements improve performance with respect to the number of activites.

4. CONCLUSION

In this work we have presented our interactive trip planning system – @FINDER, which is an efficient implementation of the two-level threshold algorithm. Our long term goal is to devise more efficient and scalable solutions for the *ETQ* problem, and more carefully explore the current trade-offs in our current approaches.

Acknowledgment. This work was supported by the Australian Research Council's *Discovery Projects* Scheme (DP140103256). Shane Culpepper is the recipient of an Australian Research Council DECRA Research Fellowship (DE140100275).

References

[1] R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[2] J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL*, pages 199–208, 2012.

[3] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: An efficiency study. In *SIGMOD*, pages 255–266, 2010.

[4] T. Kurashima, T. Iwata, G. Irie, and K. Fujimura. Travel route recommendation using geotags in photo sharing sites. *CIKM*, pages 579–588, 2010.

[5] X. Lu, C. Wang, J. M. Yang, Y. Pang, and L. Zhang. Photo2trip: generating travel routes from geo-tagged photos for trip planning. In *ACM MM*, pages 143–152, 2010.

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

[7] D. Zhang, C.-Y. Chan, and K.-L. Tan. Processing spatial keyword query as a top-k aggregation query. In *SIGIR*, pages 355–364, 2014.

[8] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *ICDE*, pages 230–241, 2013.