Location-sensitive Query Auto-completion

Chunbin Lin†, Jianguo Wang†, Jiaheng Lu‡
† University of California, San Diego ‡ University of Helsinki, Finland
† {chunbinlin, csjgwang}@cs.ucsd.edu ‡ jiaheng.lu@helsinki.fi

ABSTRACT

This paper studies the location-sensitive auto-completion problem. We propose an efficient algorithm *SQA* running on a native index combining both IR-tree and Trie index. The experiments on real-life datasets demonstrate that SQA outperforms baseline methods by one order of magnitude.

1. INTRODUCTION

The pervasiveness and importance of spatial keyword search has sparked a lot of research work on the topic. However, they ignore the fact that typing meaningful spatial keyword queries is a time-consuming and error-prone process, especially on mobile devices. Query auto-completion [4, 5] is a popular feature of web search engines that aims to assist users to formulate queries faster and avoid spelling mistakes by presenting them with possible completions as soon as they start typing.

Motivation. However, existing auto-completion techniques are not location sensitive (or spatial-aware). They only consider the text descriptions while ignoring the spatial information. For example, a user in San Diego types "Uni" to formulate a query. One of the expected queries is "University of California, San Diego". However, existing auto-completion algorithms may return "University of California, Los Angeles" as a suggestion, which is more than 100 miles far away from the user. A successful spatial keyword search system should provide efficient location sensitive auto-completion feature to help users formulate a meaningful query. More precisely, given a query prefix (a prefix that a user is typing) and a location, which can be obtained automatically by tracking the GPS signal of the mobile devices, the answer is a list of ranked spatial objects whose distances to the given location are bounded by a threshold value and the text descriptions have prefixes matching the query string. Then users can choose one from the list to complete the query. The challenges are (i) how to efficiently return such a suggestion list and (ii) how to rank the objects within the list.

Contribution. In this paper we study the location-sensitive (or spatial-aware) query auto-completion problem. We propose an efficient algorithm, called SQA, running on a native index structure, which combines a spatial index, e.g., R-tree [1] and Trie trees [2]. We also define a new ranking function for the auto-completion answers. We conduct experiments to evaluate the performance of proposed algorithms, which demonstrate that SQA outperform the baselines by one order of magnitude.

2. PROBLEM STATEMENT

Consider a geo-textural database D where each object $o \in D$ is defined as a tuple (*o.loc*, *o.doc*), where *o.loc* is a location descriptor in multidimensional space (For example, (longitude, latitude) in a 2-dimensional geographic space) and *o.doc* is a text that describes the object (e.g., "University of California, San Diego"). Given a query q=$\{\ell, s\}$ where ℓ refers to a spatial point and s refers to a string term typed by users. The **L**ocation **S**ensitive **A**uto-**C**ompletion problem (LSAC) is to find a set of ranked objects $\mathcal{O} = o_1, ..., o_k$, such that $dis(o_i.loc, q.\ell) < \tau$ for any $i \in [1, k]$, where dis(a,b) means the Euclidean distance of spatial points a and b, τ is a given threshold value, and the prefix of $o_i.doc$ matches $q.s$. The returned objects are ranked according to the ranking function defined below.

Ranking function. We derive a ranking function $\mathcal{F}$ as a linear interpolation of normalized factors for ranking an object o with regard to a query q:

$$\mathcal{F}(o, q) = \alpha\frac{dis(o.loc, q.\ell)}{maxD} + (1 - \alpha)\frac{o.Score}{maxS}$$

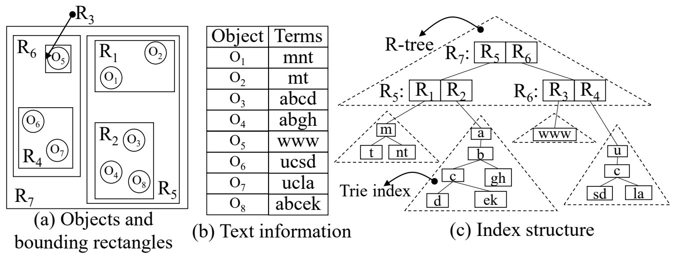
where $\alpha \in (0, 1)$ is a parameter used to balance spatial proximity and the relevancy of other features. $\frac{dis(o.loc,q.\ell)}{maxD}$ is the normalized distance between objects o and q. $\frac{o.Score}{maxS}$ is the normalized score aggregated from several features, e.g., user rating, user feedback, and price. $o.Score = 0$ if none of the above features are provided.

3. SQA ALGORITHM

In this section, we first present the index structure, then describe the SQA algorithm.

Index structure. Consider a geo-textural database D, we build an efficient index, called *RT-tree*, to index all the objects including the spatial information and text information. RT-tree consists of two components: (1) an R-tree in the top and (2) a trie tree for each leaf-node of the R-tree.

R-tree. An R-tree is efficient to find close spatial objects to a query location. During building an R-tree, spatial objects



Object	Terms
O_1	mnt
O_2	mt
O_3	abcd
O_4	abgh
O_5	www
O_6	ucsd
O_7	ucla
O_8	abcek

Figure 1: RT index structure.

are first abstracted as minimum bounding boxes (MBBs). Spatial objects whose MBBs are closely located are clustered in leaf nodes. Then leaf nodes with closely located MBBs are grouped to form non-leaf nodes. This grouping process propagates until the root node is formed.

Trie-tree. A trie-tree is efficient in obtaining strings matching a query prefix. In particular, in a trie-tree, each string corresponds to a unique path from the root of the trie to a leaf node.

Dislike the existing R-tree based spatial indexes, e.g., IR-tree [3] and WIBR-tree [6], RT-tree maintains a trie tree to index the textual part for each leaf node of the R-tree while the existing R-tree based indexes maintain inverted lists for the texts in the leaf nodes. Figure 1 (c) shows an example RT-tree for the spatial objects (Figure 1(a)) and the texts (Figure 1(b)).

SQA algorithm. We design an algorithm SQA to efficiently answer spatial-aware query auto-completion queries. SQA operates on the following three steps: (Step 1) find the trie-tree T by traveling the R-tree at $\mathcal{R}$ based on $q.\ell$ and τ; (Step 2) find strings in the subtree located at the node at T that matches the prefix s and store them into $\mathcal{O}$; and (Step 3) sort $\mathcal{O}$ based on the ranking function.

Note that, during query formulation, SQA only needs to execute step 1 once for the first character. For the following characters, SQA skips step 1 as the location of the user is almost stable during the query formulation.

4. EXPERIMENTS

Experimental platform. All experiments were done on a computer with a 4th generation Intel i7-4770 processor, 16 GB RAM, running Ubuntu 14.04.1. All the algorithms were implemented in C++ using -O3 optimization.

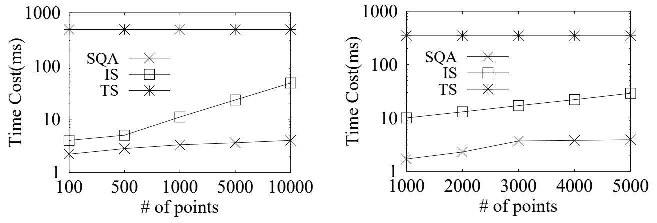
Dataset. Our experiments are conducted on two real datasets: WW and USA[1]. WW contains 60,962 worldwide POIs (points of interests) with both location and text, while USA has 35,989 tweets with geo-locations.

Baseline methods. We implement two baseline methods: (1) *IS*: Use the IR-tree index to get a list of objects close to the query location, then check the prefix for each of the object. (2) *TS*: Employ the Trie-tree index to get a list of objects with the prefix matching the query term, then check the distance to the query location for each object.

Performance evaluation. We first evaluate the performance against the number of points within the bounded distance. As shown in Figure 2, Seen from Figure 2, we have the following observations:

[1]http://www.ntu.edu.sg/home/gaocong/datacode.htm

1. SQA outperforms IS and TS. In particular, it is about 5 times faster than IS and 100 times faster than TS.
2. TS stays the same with different distance while fixing number of characters, since it always need to scan all the points whose prefixes are matched.
3. The running time of IS increases with the increasing of points as IS needs to do more prefix matchings with more points.



(a) WW (b) USA

Figure 2: Time cost (ms) (log-scale). Varying # of points within the distance while fixing # of characters to be 2.

5. RELATED WORKS

Auto-completion has been widely used in many applications such as XML search [4], Graph Search [7] and web search [5]. However, none of them is location sensitive. The only relevant previous work providing location sensitive auto-completion is MESA [8]. The goal of MESA is to support error-tolerant auto-completion over geo-textual data, which is different from this paper. In this paper we focus on providing efficient location sensitive auto-completion with exact prefix matching. In addition, the index structures are different as well as the ranking functions.

6. CONCLUSION

In this paper we proposed an efficient algorithm to solve the location-sensitive query auto-completion problem. Experiments demonstrate the high performance of the algorithm.

7. REFERENCES

[1] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[2] S. Ji, G. Li, C. Li, and J. Feng. Efficient interactive fuzzy keyword search. In *WWW*, pages 371–380, 2009.

[3] Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.

[4] C. Lin, J. Lu, T. W. Ling, and B. Cautis. Lotusx: a position-aware xml graphical search system with auto-completion. In *ICDE*, pages 1265–1268, 2012.

[5] B. Mitra, M. Shokouhi, F. Radlinski, and K. Hofmann. On user interactions with query auto-completion. In *SIGIR*, pages 1055–1058, 2014.

[6] D. Wu, M. L. Yiu, G. Cong, and C. S. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 24(10):1889–1903, 2012.

[7] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu. Autog: A visual query autocompletion framework for graph databases. *PVLDB*, 9(13):1505–1508, 2016.

[8] Y. Zheng, Z. Bao, L. Shou, and A. K. H. Tung. MESA: A map service to support fuzzy type-ahead search over geo-textual data. *PVLDB*, 7(13):1545–1548, 2014.