MESA: A Map Service to Support Fuzzy Type-ahead Search over Geo-Textual Data

Yuxin Zheng [†], Zhifeng Bao [‡], Lidan Shou [#] and Anthony K. H. Tung [†]
[†] School of Computing, National University of Singapore, Singapore
[‡] School of Engineering&ICT, HITLab AU, University of Tasmainia, Australia
[#] College of Computer Science and Technology, Zhejiang University, China
[†]{yuxin, atung}@comp.nus.edu.sg, [‡]zhifeng.bao@utas.edu.au, [#]should@zju.edu.cn

ABSTRACT

Geo-textual data are ubiquitous these days. Recent study on spatial keyword search focused on the processing of queries which retrieve objects that match certain keywords within a spatial region. To ensure effective data retrieval, various extensions were done including the tolerance of errors in keyword matching and the search-as-you-type feature using prefix matching. We present MESA, a map application to support different variants of spatial keyword query. In this demonstration, we adopt the autocompletion paradigm that generates the initial query as a prefix matching query. If there are few matching results, other variants are performed as a form of relaxation that reuses the processing done in earlier phases. The types of relaxation allowed include spatial region expansion and exact/approximate prefix/substring matching. MESA adopts the client-server architecture. It provides fuzzy type-ahead search over geo-textual data. The core of MESA is to adopt a unifying search strategy, which incrementally applies the relaxation in an appropriate order to maximize the efficiency of query processing. In addition, MESA equips a user-friendly interface to interact with users and visualize results. MESA also provides customized search to meet the needs of different users.

1. INTRODUCTION

With the proliferation of geographical applications, such as Google Earth and Foursquare, geo-textual data are becoming ubiquitous these days. To retrieve such geo-textual data effectively, recent studies focused on the processing of spatial keyword queries which retrieve data objects that match certain keywords within a spatial region [4]. Further enhancements of spatial keyword search are conducted in two ways. One is to support autocompletion using prefix matching [2], which returns data objects that satisfy the prefix matching condition within the query region. We refer it as spatial prefix search. Another extension is to allow the approximate matching, which tolerates the fuzzy matching between query keywords and data objects [9]. However, the fuzzy search applied in [9] modeled the approximate matching at word level, which requires users to type at least one full word in a query. This approach violates the concept of autocompletion because real-time search engines should be error-tolerant for autocompletion [3].

Goal: error-tolerant autocompletion in spatial keyword search. Motivated by the error-tolerant autocompletion feature in string matching [3], the scope of autocompletion should also be extended to tolerate errors in the context of spatial keyword search. The combination of autocompletion and error tolerance improves the query efficiency and user experience. It can also act as a hint in guiding users' typing and reduce the probability of errors being produced. This leads to our goal to support error-tolerant autocompletion over geo-textual data.

Given a spatial prefix query, it is possible that no or few data objects are returned due to the mismatches between users' domain knowledge and the underlying data: (1) the query range is wrongly specified by the users, such as tourists, who are not familiar with the query region; (2) the query text does not match the prefix condition due to typos in the query text or data. As a result, query relaxation must be performed to ensure that useful answers are returned. Such relaxation can be conducted in two directions: (1) expand the spatial region to a larger region; (2) relax the prefix condition to more general textual conditions.

The relaxation of the prefix matching condition can be further divided into several degrees. The first natural choice is to relax it to the *substring query*, which requires that the query text be an *exact substring* of the textual content of a spatial object. To handle the typo issue, the approximate matching is adopted. This leads to the second type of relaxation by allowing mismatches in keywords but limiting the matching to start at the prefix. This is called *approximate prefix query*. If both of these types of relaxation fail to produce sufficient number of results, we would then allow approximate string matching to be applied to any part of the textual content. This third type of relaxation is named *approximate substring query*.

An ideal search engine should have the following features to fulfil the error-tolerant autocompletion function, which are also the motivations of our design.

Feature 1: the support for different types of relaxation. Different queries need different types and degrees of relaxation. As a result, we need an integrated system that can support all the aforementioned types of relaxation.

Feature 2: efficient query processing to achieve

Proceedings of the VLDB Endowment, Vol. 7, No. 13

real-time human-computer interaction. Determining what relaxation to perform and performing each type of relaxation efficiently are the major challenges. While existing works handled some of these types of relaxation as a single query [2, 9], performing all these different types of relaxation means re-issuing a new query every time. This is inefficient especially when these queries are answered using different indexes proposed in different works. In addition, these extensions built their solutions by assuming that the relaxation stops at a certain type of relaxed query.

To achieve efficient query processing, we adopt a unifying search strategy to perform a query. We first process a query as a spatial prefix query. If few results are returned, we relax the query in two directions. The relaxation is applied incrementally: the relaxation on the spatial region is first processed, then the relaxation of the prefix matching condition is performed. When relaxing the string condition, the substring matching is first applied, followed by the approximate prefix matching, then the approximate substring matching.

Based on the above observations, we design our solution in a progressive way. We first design an index that serves two purposes: it can support the processing of each type of relaxation, and it can provide backup for optimizations to accelerate the query processing. Then we observe that different types of relaxation share much common query processing and their results can be reused to save computation cost. By ordering these types of relaxation in an appropriate order, we can use the dependencies between them to optimize our query processing via reuse of previous results.

Feature 3: a user-friendly interface to interact with users and visualize results. In addition, different types of relaxation are required for different queries. We need to have a proper way to visualize these results such as diversifications of results returned from different types of relaxation. Moreover, the interface should provide customized search components to meet the needs of different users. For example, advanced users can control which type of relaxation to perform for a particular query.

Integrating all the above features, we demonstrate a map-based search system, called MESA (Map sErvice for relaxation of SpAtial Prefix Query). The major advantages of MESA are summarized as follows:

1. MESA provides error-tolerant autocompletion in the context of spatial keyword search such as the approximate prefix matching and the approximate substring matching.
2. MESA builds a one-size-fits-all index to support the spatial prefix query and different types of relaxation.
3. MESA optimizes the query processing using a unifying incremental relaxation strategy to achieve real-time human-computer interaction.
4. MESA provides a user-friendly map-based interface to interact with users and visualize the results in a proper way.

2. CORE TECHNIQUES OF MESA

When building MESA, two major challenges are represented: (1) we need a one-size-fits-all index that can support the spatial prefix search and its different types of relaxation; and (2) we need an efficient way to process queries.

2.1 Challenge 1: A One-size-fits-all Index

Quadtree [5] is widely adopted to achieve fast retrieval of objects in geographical space. Each node of a two-dimensional Quadtree represents a bounding box covering some part of the space, while a Hilbert curve [1] is a space-filling curve, which maps multi-dimensional data to one dimension. The sequential order of a cell in a Hilbert curve is identified by its Hilbert code.

Definition 1 (Hilbert Code). *Given a Quadtree node n, its Hilbert code hc is a concatenation of the Hilbert code of its parent node and n's local order. The Hilbert code of the root node is set null. A Hilbert code hc_1 is defined to be ahead of hc_2, if hc_1 has a smaller value at the first different bit of hc_1 and hc_2.*

We combine the Hilbert curve with the Quadtree as in [1]. The cells of a Quadtree are sequentialized using the Hilbert curve to retrieve objects in a given region efficiently. We call this a *Hilbert-encoded Quadtree.* of which an example is shown in Figure 1.

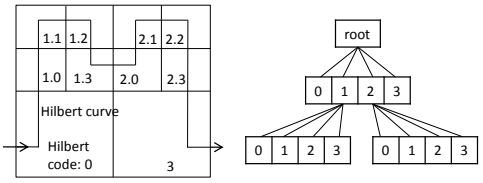

Figure 1: A Hilbert-encoded Quadtree

Based on the *Hilbert-encoded Quadtree*, we build our general index. In text search, building an inverted index on q-grams is a common choice [6]. To support the answering of a spatial prefix query and its different types of relaxation, we propose a two-level inverted index, which is an adaption of the inverted index from the granularity of q-gram to a finer level of granularity called *spatial q-gram.*

Definition 2 (Spatial q-gram). *A spatial q-gram ζ = (hc, tk) contains a Hilbert code hc and a q-gram tk.*

The essence of spatial q-grams is to combine a q-gram with a node of a Quadtree. It is a projection of a q-gram over a spatial region, which bridges the gap between the spatial information and the textual information. Based on the spatial q-gram, we propose a two-level inverted index that can support all types of relaxation that we have mentioned. The proposed structure has an object-level index to support functionalities and a node-level index to accelerate query processing.

On the one hand, to obtain the geo-textual information of an object, we build an inverted index at the object level by setting a spatial q-gram as the key and the objects containing the spatial q-gram as the value. We denote such an index as an object-level inverted index.

On the other hand, to quickly filter the nodes that cannot satisfy the string condition, we build a coarse-grained index, called node-level inverted index. We attach a count to each leaf node of the Quadtree to record the number of objects containing a particular q-gram in a node. Similar to the inverted index built to compute the tf-idf score [8], we build the node-level inverted index by setting a q-gram as the key and a list of (Hilbert-code, count) pairs as the value.

Such index design serves two main purposes: it can support the processing of any type of relaxation, and it can

provide backup for query optimization. Based on the proposed index, we then present the second challenge, which is efficient query processing.

2.2 Challenge 2: Efficient Query Processing

When a user issues a spatial prefix query, he/she usually prefers to have exact matching results before accepting approximate results. This serves as an intuitive guideline to process a query. Correspondingly, we propose a unifying search strategy in accordance with this intuitive guideline.

Based on the above one-size-fits-all index, our search strategy can be implemented in an efficient yet seamless way. We observe that different types of relaxation share much common query processing and their results can be reused to save cost. In our system, we adopt the autocompletion paradigm that generates the initial query as a prefix matching query. If few results are returned, other variants are performed as a form of relaxation that reuses the processing done in earlier phases. We order them carefully to maximize query reusability.

In addition, users usually input a query character by character, and newly typed characters are appended to the previous query forming a new query at any moment [7]. We call this type of query as *appending query*. Processing of appending queries is important to achieve the search-as-you-type paradigm for real-time search. It is obvious that an appending query can be handled by issuing it as a new query, but doing so will be inefficient. In contrast, we consider how query processing done for the initial query and relaxation can be reused in such instances.

Our system processes the spatial prefix query in the following way, as shown in Figure 2. Given an incoming query, we first check whether the incoming query Q' is an appending query of a previously-issued query Q.

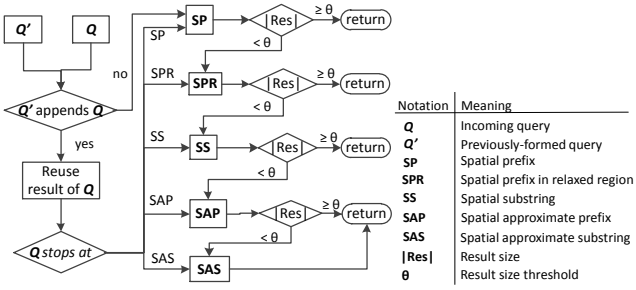

Figure 2: Search Strategy

Case 1: If the incoming query Q' is not an appending query, we process it as a new spatial prefix query. If the result size is not less than a certain threshold, the results are returned. Otherwise, the relaxation is applied incrementally: the relaxation on the spatial region is first processed, then the relaxation of the prefix matching condition is performed. When relaxing along the string dimension, the substring matching is first applied, followed by the approximate prefix matching, then the approximate substring matching.

Case 2: If the incoming query Q' is an appending query, we process it from where the previously-processed query Q stops. If sufficient results have been found, processing of the incoming query Q' stops; otherwise, we further relax it in an incremental way as what we have done in Case 1.

In Case 1, the incremental relaxation for a non-appending query may contain several phases. We identify the commonality among relaxation at different levels. This motivates us to reuse the results of the less relaxed queries (computed in earlier phases) to process the query in a new phase. In Case 2, the reuse brings another optimization opportunity: we do not process the appending query from scratch; instead, we process it from where the previously-processed query stops. Compared to existing works which processed different variants of spatial keyword queries as new queries over different indexes, our approach offers a more compelling solution to efficient and effective spatial keyword search.

3. SYSTEM ARCHITECTURE

MESA adopts the client-server architecture to build the system. The system architecture is shown in Figure 3. On the client side, when a user issues a query, he/she would first zoom in a particular spatial region that he/she is interested in. Together with the text typed by the user, the query is sent to the server side for processing. After the query is processed, the results are sent back and displayed.

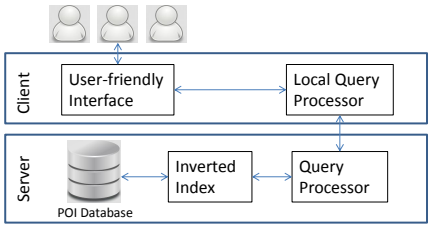

Figure 3: System Architecture

On the server side, MESA contains two major components: (1) the proposed one-size-fits-all index that can answer the spatial prefix query and its different types of relaxation; (2) a query processor to handle queries issued from the client side. The query processor adopts the proposed search strategy, which is presented in Section 2.2, to process a query. The server is built using JSP and Apache Tomcat.

On the client side, it includes two main components: (1) a user-friendly map interface to provide interactive search and result display, which is implemented using Google Maps API; (2) a local query processor to answer appending queries if the appending queries can be answered using the local cached results to save computational cost. The client side is implemented in JavaScript.

3.1 User-friendly Interface

The user-friendly Interface is used to provide interactive search and result display. For the interaction part, we provide a control panel to let users specify the input parameters. As a result, customized search can be achieved. Advanced users can issue their queries on demand, while common users can just use the default setting to issue queries. For the result display part, we use different colors to visualize the results returned from different types of relaxation. Therefore, results can be differentiated. This is motivated by the fact that users usually prefer the exact matching results to the approximate matching results. Besides, users can click the spatial objects for further information.

3.2 Local Query Processor

As described previously, the autocompletion paradigm allows additional characters to be appended after the initial query. To save the communication cost between the client and the server, we cache the results of the previous-issued query. When an appending query comes, the local query processor first checks whether the local cached results can answer the appending query. If yes, the answer set can be reused and the appending query is not necessary to be sent to the server; otherwise, the server side processes the appending query and returns the new results to the client.

All in all, MESA groups these key components into a system to provide the spatial prefix search and its different types of relaxation. The techniques, like spatial q-gram, two-level inverted index and the unifying search strategy, are proposed to make the spatial prefix query and its various types of relaxation more efficient and effective. Each of those techniques is an indispensable component of MESA, whose ultimate goal is to process fuzzy type-ahead queries in the context of spatial keyword search efficiently and effectively.

4. DEMONSTRATION

In this section, we would like to demonstrate how MESA handles spatial prefix search and its different types of relaxation effectively and efficiently, and how it enhances user experience by providing the error-tolerant autocompletion feature. Users will be able to experience how MESA facilitates the spatial keyword search by supporting the fuzzy type-ahead search.

Data sets: Three real data sets are used. The first one is extracted from the Geographic Names Information System[1] in the USA and contains the geographic name usage in the U.S. government, including about two million records. The second one is obtained from the SimpleGeo's Places[2] and has thirteen million records. The third one is obtained from OpenStreetMap's points of interest data in China[3].

Demonstration: The interface is shown in Figures 4. It consists of two parts: a control panel (upper part) and a display panel (lower part). The control panel is composed of four components: (1) a query text field to type query text; (2) a drop-down list to input the number of required results; (3) anther drop-down list to specify the type of relaxation; and (4) two slider bars to indicate the degrees of relaxation. If the type of relaxation is specified, MESA directly performs the particular type of relaxation. The slider bars are used to indicate what degrees of relaxation a user prefers. For example, users can decide how much to expand in the spatial region and what edit distance threshold to use in the approximate prefix/substring queries.

Besides, the display panel is composed of two parts: a result list and a map view. In the result list, the results of a query are shown in different colors. The color indicates that the result is returned from which type of relaxation. In addition, the results returned from the relaxation in earlier phases are ranked higher than those results returned from the relaxation in later phases because users usually prefer exact matching results to approximate matching results. In the map view, the returned objects are shown. When users click results on the map, further information pops up.

[1] http://geonames.usgs.gov/domestic/download_data.htm

[2] http://s3.amazonaws.com/simplegeo-public/places_dump_20110628.zip

[3] http://download.geofabrik.de/asia/china.html

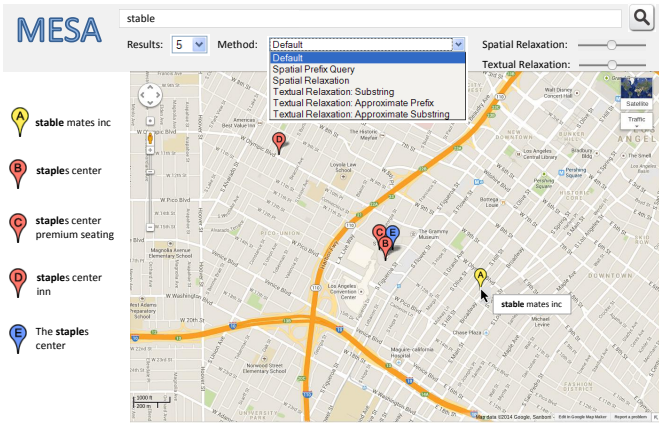

Figure 4: Interface of MESA

We plan to demonstrate the three features of MESA, as stated in the introduction. (1) MESA can answer each type or degree of relaxation on its own. If a user specifies a particular type of relaxation for a query, MESA can directly perform the type of relaxation. This feature can be shown by choosing different types of relaxation to be performed on the control panel. By default, MESA uses the unifying search strategy discussed in Section 2.2 to perform a query. (2) MESA provides efficient error-tolerant autocompletion. We will show that the results can be returned in real time to fulfill user-computer interaction requirement. (3) MESA equips a user-friendly interface to interact with users and visualize the results. Using this interface, customized search can be supported to meet the needs of different users. We will demonstrate these features by showing the results of queries issued in different places. In conclusion, this demonstration will show that MESA is able to process fuzzy type-ahead search efficiently and effectively over geo-textual data.

5. ACKNOWLEDGEMENT

This research was carried out at the SeSaMe Centre. It is supported by the Singapore NRF under its IRC@SG Funding Initiative and administered by the IDMPO.

6. REFERENCES

[1] M. Bader. *Space-Filling Curves: An Introduction With Applications in Scientific Computing*, volume 9. 2012.

[2] S. Basu Roy and K. Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In *SIGMOD*, 2011.

[3] S. Chaudhuri and R. Kaushik. Extending autocompletion to tolerate errors. In *SIGMOD*, 2009.

[4] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *VLDB*, 2013.

[5] R. Finkel and J. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1), 1974.

[6] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB*, 2001.

[7] G. Li, S. Ji, C. Li, and J. Feng. Efficient fuzzy full-text type-ahead search. *The VLDB Journal*, 20(4), 2011.

[8] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5), 1988.

[9] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou. Approximate string search in spatial databases. In *ICDE*, 2010.