Efficient Processing of Spatial-textual Nearest Neighbors Queries Using Standard Indexes

Qu Qiang
—
—
–-

Anders Skovsgaard
—
—
–-

ABSTRACT

TODO

1. INTRODUCTION

TODO: Change the introduction below but preserve some of the motivations. The proliferation of Internet-enabled, geo-positioned mobile devices has created an increased demand for local information. Thus, we are also witnessing an increased proliferation and use of different map-based services that serve local information. Local information is typically organized around Points of Interest (PoIs) that may be of different types, e.g., hotels, restaurants, and parks, that include exact locations, names, and textual descriptions.

Map-based services allow users to retrieve such different types of PoIs near them. For example, users may look for hotels or restaurants in neighborhoods where they are currently present or are going to be at a later time. Typical map-based functionality supports the retrieval of relevant PoIs that belong to the part of space visible on the user's screen. This functionality supports users who wish to browse or explore their surroundings and is different from the retrieval of PoIs that are closest to an exact user location.

TODO: Change it to Nearest Neighbor

Several proposals exist that are capable of finding the most relevant places that are closest to an exact location. They are typically either R-tree based [3, 7, 8, 13, 17, 18, 20–22], grid based [15, 19], or space filling curve based [5, 6]. The proposals either support Boolean top-k or queries, or they support top-k queries with a ranking function. However, they all suffer from two limitations: (i) they involve the use of a special index structure, making it difficult to leverage existing data management systems, and (ii) they focus on finding the top-k most relevant objects, that are closest to an exact location instead of providing the top-k most relevant objects in a region.

TODO: Change to NN

We study a query that returns the top-k most relevant PoIs in a spatial region. An example is given in Figure 1, that shows the top-5 most relevant objects for a region and the result of a nearest-neighbor type query. The left example supports exploratory user behavior in the region, since the top-5 most relevant objects may

SIGSPATIAL'14, November 04 - 07 2014, Dallas/Fort Worth, TX, USA
Copyright 2014 ACM ACM 978-1-4503-3131-9/14/11 ...$15.00
http://dx.doi.org/10.1145/2666310.2666399.

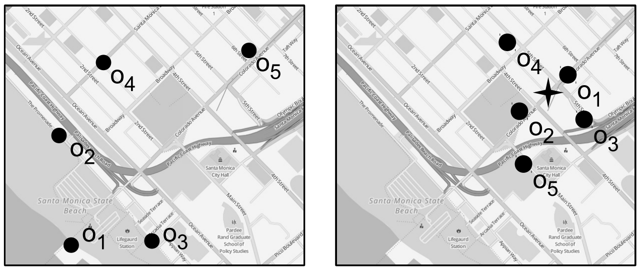

Figure 1: Top-5 results for a region query (left) and a nearest neighbor query (right)

be located anywhere in the region. In the example, the three best places are near the beach. This approach is used by all of the major providers of map-based services, including Google Maps, Bing Maps, and Yahoo! Maps. The example to the right retrieves objects close to the query location and is fundamentally different.

The paper makes four contributions. TODO:

The remainder of the paper is structured as follows. Section 2 covers related work. The problem definition is given in Section 3. Section 4 presents the proposed solution, encompassing the spatio-textual bit string encoding and indexing strategy followed by an exact query processing algorithm. The experimental evaluation is given in Section 5. Finally, we conclude and offer research directions in Section 6.

2. RELATED WORK

TODO: Mention IR-tree, S2I, I3, the GIS paper, all the exp-paper references.

Recently, substantial research efforts on geo-textual indexes have been reported. The proposed solutions often use a combination of the R-tree [12] or a variant [2] for the spatial indexing and inverted files for indexing the text [7, 13, 17, 18, 20, 22]. Others use bitmap files for text indexing [3, 8, 21]. Solutions using grids [15, 19] and space-filling-curves [5, 6] combined with inverted files have also been proposed. The spatio-textual parts are combined in three different ways: (i) either closely combined, with no clear separation [6–8, 13, 15, 17, 20, 21], (ii) with the spatial index followed by the text index [3, 5, 19, 22], (iii) or with the text index followed by the spatial index [18, 19, 22]. A comprehensive experimental evaluation of geo-textual indexes is also available [4].

Most of the related work targets Boolean queries and does not return ranked results. This paper focuses on providing a ranked result to provide a size-limited result. With the vast amounts of data that may exist in a region, it is often not helpful to return all objects in a region that contain a given term.

Some of the related work is able to efficiently return the top-k most relevant nearest neigbors [7, 17, 18, 20]. These works all use the R-tree and the query processing algorithms may be modified to find the top-k most relevant objects in a region. A baseline algorithm that uses a geo-textual index is given in Section 3.2. However, this related work uses custom index structures, while this paper proposes a technique that may be employed with existing standard indexing techniques. Despite the substantial research on answering spatio-textual queries, to the best of our knowledge, no existing work addresses encoding of spatio-textual objects in combination with existing indexing techniques.

Multi-dimensional spaces can be mapped to a one-dimensional space in order to support the indexing of multi-dimensional points in a standard DBMS. The mapping of multi-dimensional space into a one-dimensional space is often done using a space-filling curve such as a Z-curve or a Hilbert-curve [23]. The existing techniques consider only the spatial dimension, whereas we propose a new problem that considers both the spatial and the textual properties of the objects.

The Google S2 Geometry Library [11] provides an open-source implementation that maps two-dimensional objects on the Earth's surface to a one-dimensional representation. It encloses the Earth sphere in a cube and imposes a quad-tree [10] type partitioning to each of the 6 faces. The cells are encoded and decoded by means of a Hilbert curve [14]. This provides a bit string representation of a location; this paper leverages and extends this work.

Hash functions can map text strings of arbitrary lengths to fixed-length hash values [16]. When two different strings produce the same hash value, a collision occurs. As the length of the hash value decreases, the number of collisions increases. A perfect hash functions map the input to unique hash values and is thus collision-free. Hash functions are used in the proposed solution to limit the number of textual descriptions and their lengths.

3. PRELIMINARIES

We proceed with giving a formal definition of the problem addressed in the paper followed by a baseline algorithm that modifies state-of-the-art algorithms.

3.1 Problem Definition

Let D be a set of spatio-textual objects $o = (\lambda, doc)$, where λ is a point location, and doc is a text document. TODO: Add the S2I/IR-tree textual relevancy and the NN-definitions (should just be the same/similar since we want to challenge these.

A top-k spatio-textual nearest neighbor query (TkSTNN) query TkSTNN = $key\text{-}\ word, k)$ returns an ordered list of k objects that are the most relevant to $keyword$ according to the spatial distance and the textual relevance function computed by language models similar to the one presented in related work [7]. For ease of understanding, we use the term frequency function $tf(t, o.doc)$, that returns the number of occurrences of term t in the text document $o.doc$. However, other textual relevance functions may be applied.

3.2 Baseline Algorithm

TODO: Compare to IR-tree, S2I, I3.

4. PROPOSED SOLUTION

TODO: Remove the tre creations and mention it in related work. Instead focus should be on the new query processing with standard indexes.

We present a framework that encodes spatio-textual objects into compact bit strings that may be stored in standard data management systems and indexed using standard index structures that are generally available in such systems. The framework encompasses an exact query processing algorithm capable of computing TkSTR queries by using the bit strings and by combining partial results.

4.1 Spatio-Textual Bit String Encoding

We proceed to describe the encoding of the spatio-textual objects and then provide an overview of the framework.

Modern data management systems generally have a number of indexes that are optimized for processing queries efficiently. Standard indexes include B-trees [1] and hash tables! [16], and some systems also support R-trees [12]. However, many queries are not able to use these standard index structures directly, including the TkSTR query and the top-k most relevant nearest neighbor query.

We aim to develop an approach that can process the TkSTR query by utilizing standard indexes that are available in virtually any data management system such as a DBMS or a key-value store: B-trees or the hash tables. To achieve this, we first encode spatio-textual objects into bit strings that they may be readily indexed by standard indexes. A bit string has two parts: (i) a spatial part that encodes the location of the object being encoded and is used to determine whether the object is located in the query region, and (ii) a textual part that represents the text document of the object and helps determine whether the object indeed contains the query term.

4.1.1 Spatial Encoding

We have the following requirements to the spatial part of the bit string:

1. Compact representation.
2. Fast lookup of regions of arbitrary size and location.
3. Sufficiently high resolution.
4. Uniform query performance.

It is desirable to have a compact bit string to reduce the storage requirements and access cost. We aim at efficiently processing queries independently of the size and location of the query region. Also, we aim at providing a result efficiently for even very small query regions. Finally, the performance should not vary greatly depending on the size or location of the query region.

The Google S2 Geometry Library [11] utilizes a quad-tree style partitioning that yields cells of uniform area at multiple granularities. This satisfies the requirements of supporting arbitrary region sizes and locations. A bit string is created by enclosing the Earth by a cube and by imposing a quad-tree style partitioning on each of the 6 faces of the cube, as illustrated in the example in Figure 2. The quad-tree type partitioning has multiple levels, and the cells are enumerated using a Hilbert curve [14] that makes it possible to efficiently find any cell at any level.

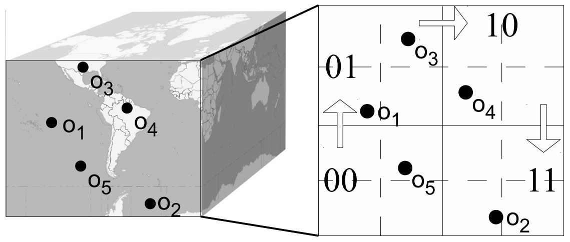

Figure 2: Earth Enclosed in a Cube (left) and a Grid Index for Each of the 6 Faces (right)

The faces of the cube can be represented with 3 bits, and the 4 cells at each level in the partitioning can be encoded using 2 bits.

With a total surface area of the Earth of 510,072,000 km^2, each of the faces covers an area of 85,012,000 km^2. To provide an accuracy of less than 1 cm^2, 30 levels are required, which can be captured by using 63-bits. We may not need this level of accuracy, and we may reduce the number of levels, thus making the bit string more compact while still providing sufficient resolution. The experimental evaluation studies the impact of different numbers of levels. This fulfils our requirement of achieving a compact bit string.

Example 1. Consider the object o_1 in Figure 2. It is on the first face of the cube which is represented by 3 bits: 000. By following the Hilbert curve in the grid tree, we find that object o_1 is in the cell with position 1 at the first level, which is represented by the bits 01. At the second level, it is in the cell with position 4, which us represented by 00. The complete encoding of the level 2 bit string of o_1 is then 0000100. □

With this approach, we are able to encode the spatial locations of objects as compact bit strings. The resulting bit strings can be indexed using standard indexes. We proceed to extend this approach with a textual bit string representation.

4.1.2 Textual Encoding

Large amounts of PoIs that cover all countries in the world are maintained by providers of map-based services. Thus, the textual descriptions may be in any language, resulting in a large vocabulary. We have two requirements to the textual part of the bit string:

1. Compact representation.
2. Limited number of bit strings.

Having a compact bit string representation instead of string with characters in any language reduces the storage requirements. Since standard indexing techniques like B-trees and hash tables are affected by the number of indexed objects, it is important to have a known and limited number of bit strings. Therefore, we aim at providing a limited number of compact bit strings that may be used for direct lookup in a standard index.

The TkSTR query takes the argument $keyword$, which is a term, e.g., "pizza" or "sushi." By encoding each term in the object document, $o.doc$, we may use the bit strings to produce a result for the TkSTR queries. To encode a term into a bit string of a fixed length, we propose to use the hash value of each term in the textual description. By using a one-way hash function, we achieve both of the above requirements. First, the length of the bit string may be set to be sufficiently short by the choice of the hash function. Second, a hash function produces a fixed number of hash values. Next, it is possible to use any hash function and to use truncation of the hash values to obtain a limited number of compact bit strings. In the experimental evaluation, we study the use of different lengths of the hash values.

Hash functions may introduce collisions, which occur when two or more input text strings produce the same output hash value. An example is shown in Figure 3 where the terms "pizza" and "sushi" both hash to the same value "00".

When the hash function produces a small number of output hash values or when the hash values are truncated, it is more likely to see more collisions. However, by increasing the amount and length of the output hash values, the storage requirements and the time to perform indexing and lookups also increase.

By encoding the terms as compact bit strings, we may concatenate these with the spatial bit strings to achieve a single, compact bit string. The resulting compact bit strings hold information about the locations and the textual descriptions of the objects they represent.

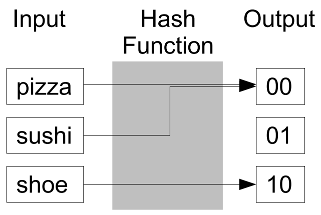

Figure 3: Two Input Strings Produce the Same Output Hash Value

Example 2. Consider the object o_1 in Figure 2 and the term frequencies in Table 1. With the hash function in Figure 3, the combined bit string with level 1 spatial resolution for the terms "pizza" and "sushi" is 0000100, while "shoe" encodes to the string 0000110. □

4.1.3 Handling Collisions

The text documents of objects that are located in the same spatial grid cell may contain the same terms, resulting in identical bit strings. We call this a bit string collision. Also, objects located in the same cell, but with different terms, may be encoded to the same bit string since we use one-way hash functions that produce collisions. We call this a term collision. To be able to provide exact result for the queries, we need to be able to distinguish between co-located objects that share the same terms. We proceed to present techniques for handling objects with the two types of collisions: bit string collisions and term collisions.

Bit String Collision Multiple objects may exists in the same grid cell, resulting in the same bit string. This is more likely at the first levels of the grid since they cover larger areas. With the large amounts of data, any grid cell may contain numerous objects. These objects are also contained in grid cells at the lower levels. We want to avoid the duplicate storage of these large amounts of objects.

According to Definition **??**, the TkSTR query returns $Q.k$ objects with the highest term frequency. Therefore, in order to provide an efficient and exact result for any given grid cell, it is only necessary to store the $Q.k$ objects with highest term frequency. The remaining objects will not be used to answer queries with a region that exactly matches the size of a grid cell. We propose to set a maximum value, $targetedK \geq Q.k$, for the number of objects to store in each cell. Thus, we can reduce the storage requirements and process all queries with $Q.k \leq targetedK$ without using any cells from the lower levels. Naturally, the value of $targetedK$ should be set sufficiently high in order to provide a useful number of objects in the result set.

The query region may not have the same size of the grid cell and thereby contain the $Q.k$ objects. The query region may be smaller than the grid cells and even smaller than the grid cells at the lowest level. To support queries with such regions, we store all objects at the lowest level. Thus, the lowest-level grid cells do not only store $targetedK$ objects, but store all objects that are located in the regions of the cells. Since we store all objects at the lowest level, no information is lost, and an exact result may be provided for any query region.

Term Collision By using hashing, we get a limited number of compact bit strings that may reduce storage requirements and the amount of identifiers to index. However, this may introduce collisions as shown in the example in Figure 3. We propose a method that handles the term collisions such that it may be employed using any key-value store.

The grid cells may describe up to $targetedK$ objects for any given term. For the cells at the lowest level, the number may exceed $targetedK$. The mapping of a term to the actual objects can easily be done when there are no term collisions. The hash value simply serves as a unique identifier for the specific term, and it may point directly to a bucket with the objects. Consider Figure 4 where the same input and output as in Figure 3 are used and $targetedK$ is set to 2. The input term "shoe" encodes to "10" and points directly to objects o_5 and o_2 from Figure 2 ordered by the term frequencies from Table 1.

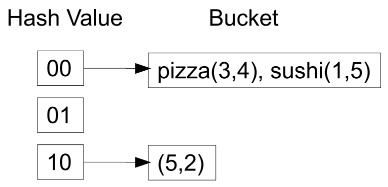

Figure 4: Buckets with Collisions

However, when term collisions occur, we propose to store a separate chain of terms in a bucket since the hash value no longer uniquely identifies the terms. Each of these stored terms identifies the corresponding objects. In Figure 4 both of the terms "pizza" and "sushi" encode to "00." Therefore, the bucket contains both terms along with the corresponding objects. With this approach, we can distinguish between objects that are located in the same grid cell and have the same terms.

4.1.4 Framework Overview

We proceed to give an overview of the framework. First, the data is preprocessed such that each term of the object documents is handled with the proposed spatial and textual encoding techniques as illustrated in Figure 5. The result is a bit string for each term of the object documents. The bit string is then stored and indexed in a standard data management system using a standard indexing technique. A detailed description of the storing of objects is provided in the following section.

The query processing is shown in the lower part of Figure 5 and is covered in detail in Section 4.3. First, the query region and keyword are preprocessed such that the spatial and textual parts of the query are encoded into a bit string. This bit string is then used to perform a lookup in data management system used, which yields a number of objects. Lower levels of the grid structure may have to be searched in order to produce an exact result. Therefore, the bit string is refined and used to perform additional lookups. When enough objects necessary to answer the query have been retrieved, the result is returned.

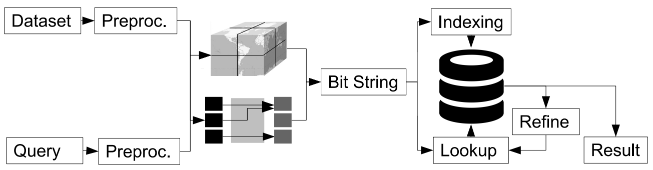

Figure 5: Framework Overview with the Spatial and Textual Encoding (two left figures) and a Standard Data Management System (right figure)

4.2 Storing Objects in a Key-Value Store

Existing DBMS and cloud database technologies all support key-value stores with standard indexing techniques such as B-trees and hash tables. We propose a method that works with all key-value store indexing techniques. The proposed method encodes an object to a bit string that may be represented by data types such as integers or strings, depending on the types available in the system used. When storing the objects, each term of the object documents is encoded into a bit string. To apply the proposed techniques to a key-value store, we store the bit strings as the key. By having an index on the key, we are able to perform lookups of the bit strings efficiently.

Since the query region may be of arbitrary size, we store a bit string for each level of the grid, as seen from the *key* column in Table 2 that describes all objects from Figure 2. If we only stored objects at the finest levels, a large number of cells would have to be fetched to provide a result for queries with large regions.

Without term collisions, any key will at most map to $targetedK$ object references. A standard page of size either 4 or 8 KB can hold 32 bit integer references to 1,000 or 2,000 objects, respectively. To avoid fetching each object when performing query processing, we propose to store the term frequency in a 16 bit integer along with a 32 bit integer for both the latitude and longitude for each object. Thus, the object representation will have the following structure:

$$id(INT32)tf(INT16)lat(INT32)lng(INT32).$$

In total, 14 bytes are required to store this object reference. A page of size 4 or 8 KB can hold approximately 300 or 600 objects, respectively. We assume that $Q.k$ will be much smaller than this number, and we thus also assume that $targetedK$ will be similarly small.

With these storage requirements, we propose to store all object references directly as the values, as illustrated in Table 2. The table describes the objects from Figure 2 ordered by the term frequencies from Table 1. To simplify, we only give the identifier of the object and not the full object representation.

key	$value$
00000	pizza(3,4), sushi(1,5)
00010	shoe(5,2)
0000000	pizza(5), sushi(5)
0000010	shoe(5)
0000100	pizza(3,1), sushi(1,3)
0000110	shoe(1)
0001000	pizza(4), sushi(4)
0001010	shoe(4)
0001100	pizza(2), sushi(2)
0001110	shoe(2)

Table 2: Key-Value Database with $targetedK$ set to 2

When objects produce term collisions, we lose the direct mapping from bit string to a term. The term of a stored object is required in order to detect when a term collision occurs. Therefore, we also store the terms along with the object representations to avoid fetching the complete textual descriptions of the objects. Term collisions may eventually occur, which is why we store the terms pro-actively for all values. The storing of terms requires more space depending on the number of term collisions. The experimental evaluation provides insight into the storage requirements.

All objects are stored in sorted order with regard to the term frequencies. The insertion algorithm is given in Algorithm 1, which takes a dataset of spatio-textual objects and a grid level as arguments. With no bit string collision, an object can be inserted directly, as shown in Line 10 since no other object with the term exists. When a term is creating a term collision for the first time, the same applies. If the cell contains less than $targetedK$ objects for a given term, the object is inserted, since there is room for it;

see Line 13. All objects are inserted in cells at the lowest level.

Algorithm 1: Insert(Dataset D, GridLevel gl, Boolean $lowest$)

```
   /* Global variables: */
1  $db \leftarrow$ Key-Value Store ;
2  $targetedK \leftarrow$ max obj. ref. stored in $DB$ ;          // conf. param.
   /* Local variables: */
3  $bs \leftarrow$ empty bit string ;
4  foreach $o \in D$ do
5      foreach $t \in o.doc$ do
6          $bs \leftarrow$ spatialEncode($gl$, $o.\lambda$) ;
7          $bs \leftarrow bs \cup$ termEncode($t$) ;
8          $value \leftarrow db$.lookup($bs$) ;
9          if $value$ is empty $\vee$ $value$ does not contain $t$ then
10             $db$.insert($o$) ;                    // no objects with $t$ exists
11         else if $value$ contains $t$ then
12             if $lowest \vee |value| < targetedK$ then
13                 $db$.insert($o$) ;                // there is room
14             else if $tf(t,o.doc) > value.mintf$ then
15                 $db$.sortAndUpdate($o$) ;         // more relevant
```

Because we store the term frequency as part of the object representation, dynamic updates are supported without fetching the actual objects. This is seen from Line 14 where the term frequency of the new object is compared with the lowest term frequency of already existing object representations in the cell. Therefore, we do not need to fetch already inserted objects in order to maintain the $targetedK$ most relevant objects of a cell.

4.3 Query Processing

TODO: New search algorithms that uses only standard indexes.

Alg. First fetch a cell that enclosed Q.l. Find first the NN of that cell, NN1. Next, calculate how far away the most relevant object, R, can be away from Q.l if it has the maximum textual relevance. Then, fetch the cell that encloses R. If no better match exists, return NN1. Else return R. Continue with NN2, and use the caches.

5. EXPERIMENTAL EVALUATION

TODO

6. CONCLUSIONS AND RESEARCH DIRECTIONS

TODO

Acknowledgments

This research was supported in part by the MADALGO research center, in part by the Geocrowd Initial Training Network, funded by the European Commission as an FP7 Peoples Marie Curie Action under grant agreement number 264994, and by a grant from the Obel Family Foundation.

7. REFERENCES

[1] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD*, pages 322–331, 1990.

[3] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial Boolean queries. In *SSDBM*, pages 87–95, 2010.

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

[5] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *SIGMOD*, pages 277–288, 2006.

[6] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: Efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

[7] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[8] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *In PODS*, 102–113, 2001.

[10] R. Finkel and J. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[11] Google Inc. Google S2 Geometry Library. `http://code.google.com/p/s2-geometry-library/`, 2011. [Online; accessed September 2014].

[12] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD*, pages 47–57, 1984.

[13] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *SSBDM*, pages 16–16, 2007.

[14] D. Hilbert. Uber die stetige abbildung einer linie auf ein flachenstuck. In *Mathematische Annalen, 38*: 459-460, 1891.

[15] A. Khodaei, C. Shahabi, and C. Li. Hybrid indexing and seamless ranking of spatial and textual features of web documents. In *DEXA*, pages 450–466, 2010.

[16] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching*, 2nd ed., Addison-Wesley. 1998.

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

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

[19] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *SSTD*, pages 218–235, 2005.

[20] D. Wu, G. Cong, and C. S. Jensen. A framework for efficient spatial web object retrieval. *VLDBJ*, 21(6):797–822, 2012.

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

[22] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162, 2005.

[23] V. Gaede, and O. Günther. Multidimensional Access Methods. In *CSUR*, pages 170-231, 1998