Processing Spatial Keyword Query as a Top-k Aggregation Query

Dongxiang Zhang Chee-Yong Chan Kian-Lee Tan
Department of Computer Science
School of Computing, National University of Singapore
{zhangdo,chancy,tankl}@comp.nus.edu.sg

ABSTRACT

We examine the spatial keyword search problem to retrieve objects of interest that are ranked based on both their spatial proximity to the query location as well as the textual relevance of the object's keywords. Existing solutions for the problem are based on either using a combination of textual and spatial indexes or using specialized hybrid indexes that integrate the indexing of both textual and spatial attribute values. In this paper, we propose a new approach that is based on modeling the problem as a top-k aggregation problem which enables the design of a scalable and efficient solution that is based on the ubiquitous inverted list index. Our performance study demonstrates that our approach outperforms the state-of-the-art hybrid methods by a wide margin.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process

Keywords

Spatial keyword search; Top-k aggregation

1. INTRODUCTION

The prevalence of smartphones and social network systems has enabled users to create, disseminate and discover content on-the-move. According to a recent study [2], Twitter has 164 million monthly active users accessing its site from mobile devices while Facebook has 425 million mobile users doing so. Consequently, a tremendous amount of social content, including tweets, check-ins, POIs and reviews, are generated every day and notably, these data are attached with geo-coordinates and form a large-scale *geo-document* corpus. Spatial keyword search is an important functionality in exploring useful information from a corpus of geo-documents and has been extensively studied for years [5, 13, 14, 16, 18, 22, 29, 33–36]. The work on spatial keyword search can be broadly categorized into two classes: *distance-insensitive* and *distance-sensitive*.

In the traditional *distance-insensitive* keyword search, the documents are organized based on a geographical ontology [5, 13, 18, 36]. For example, "University Ave" → "Palo Alto" →" South Bay"→ "Bay Area" [5] is an example hierarchical path in a geographical ontology. Given a keyword query with a locational constraint, the constraint is used to restrict the document search within matching hierachies of the ontology. Thus, the query's locational constraint here is used as a filtering criterion. In *distance-sensitive* keyword search, each geo-document is associated with a precise reference point [1] so that its spatial distance from a query location can be calculated to evaluate its relevance in terms of spatial proximity. Thus, the query location here is used as a ranking criterion.

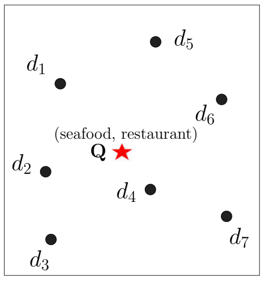
Distance-sensitive keyword search has many useful applications that facilitate a user to search for nearby locations/events/friends based on some keywords with the matching results ranked in increasing proximity to the user's location. For example, point-of-interest (POI) applications such as Foursquare and Yelp are very useful to search for nearby venues or restaurants. As another example, location-based social discovery applications such as Tinder are very useful to search for nearby users with mutual interests.

In this paper, we focus on the problem of top-*k* distance-sensitive spatial keyword query search: *given a spatial keyword query (with a location and a set of keywords), the objective is to return the best k documents based on a ranking function that takes into account both textual relevance as well as spatial proximity.* Figure 1 illustrates a simple example of distance-sensitive spatial keyword search for a corpus with seven geo-documents $d_1, \cdots, d_7$. Each document is associated with a location and a collection of keywords together with their relevance scores. Consider a spatial keyword query Q with keywords "seafood restaurant" issued at the location marked by the star. In this example, document d_2 is considered to match Q better than document d_4 because d_2 is close to the query location and it is highly relevant to the query keywords; in comparison, d_4 does not match the query keywords well, though it is slightly closer to Q then d_2.

The efficiency of evaluating top-*k* distance-sensitive spatial keyword queries becomes critical for a large-scale geo-document corpus. Existing methods for this problem [14, 16, 22, 29, 35] have shown that combining the spatial and textual attributes together can effectively prune the search space. These methods can be divided into two broad categories: *spatial-first* and *textual-first*. In the *spatial-first* strategy, a spatial data structure (such as an R-tree [10]) is first used to organize the geo-documents into spatial regions based on their locations. Next, each spatial region is augmented with an inverted index to index all the documents contained in that region. An example of a *spatial-first* scheme is the IR-tree [14]. On the other hand, in the *textual-first* strategy, the search space of geo-documents is first organized by keywords, and then for

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.
Copyright 2014 ACM 978-1-4503-2257-7/14/07 ...$15.00.
http://dx.doi.org/10.1145/2600428.2609562.

[1]The location information can be naturally derived from the GPS of smartphones.



d_1	(pizza 0.6), (restaurant,0.4)
d_2	(seafood 0.9), (restaurant 0.8)
d_3	(seafood 0.2), (pizza 0.5)
d_4	(noodle 0.7), (seafood, 0.2)
d_5	(spicy 0.8), (noodle 0.5), (restaurant 0.6)
d_6	(spicy 0.4), (restaurant 0.5)
d_7	(seafood 0.1), (restaurant 0.3)

Figure 1: An example of spatial keyword search scenario

each keyword, a spatial data structure is employed to index the documents associated with that keyword. S2I [29] and I^3 [35] are two schemes that follow this strategy. Recent extensive performance studies [29, 35] have demonstrated that both S2I and I^3 are significantly faster than IR-tree. As such, we consider S2I and I^3 to be the state-of-the-art solutions for distance-sensitive spatial keyword search.

However, there are two key limitations with the state-of-the-art solutions. First, these solutions are not sufficiently scalable to cope with the demands of today's applications. For example, Foursquare has over 60 millions of venues, and the number of geo-tweets published by mobile users per day is in the hundreds of million. None of existing solutions [29, 35] have been evaluated with such large-scale datasets; indeed, our experimental study reveals that the state-of-the-art solutions do not perform well for large datasets. Second, these solutions employ specialized "hybrid" indexes that integrate the indexing of both textual as well as spatial attributes which present additional complexity for their adoption in existing search engines in terms of implementation effort, robustness testing, and performance tuning.

In this paper, we propose a new approach to address the limitations of the state-of-the-art solutions. Our paper makes three key contributions. First, we present a simple but effective approach to solve the top-k distance-sensitive spatial keyword query by modeling it as the well-known top-k aggregation problem. Second, we propose a novel and efficient approach, named *Rank-aware CA (RCA) algorithm*, which extends the well-known CA algorithm with more effective pruning. In contrast to the specialized hybrid solutions, our RCA algorithm is more practical as it relies only on the ubiquitous and well-understood inverted index. Third, we demonstrate that RCA outperforms the state-of-the-art approaches by a wide margin (up to 5 times faster) on a real dataset with 100 million geo-tweets.

The rest of the paper is organized as follows. We present the problem statement in Section 2, and review related work in Section 3. In Section 4, we present our new approach to model the problem and two baseline algorithms. We present our new algorithm, RCA, in Section 5. In section 6, we evaluate the efficiency of RCA with an extensive performance study. Section 7 concludes the paper.

2. PROBLEM STATEMENT

We model a geo-document by a quintuple, $\mathcal{D} = \langle docID, x, y, terms\rangle$; where *docID* denote the document id, (x,y) denote the (longitude, latitude) location of the document, and *terms* denote a collection of weighted terms. Each weighted term is of the form $\langle term_i, \phi_t(\mathcal{D}, term_i)\rangle$, where $term_i$ denote a term in document $\mathcal{D}$ and $\phi_t(\mathcal{D}, term_i)$ denote the textual relevance score between $\mathcal{D}$ and $term_i$. Here, ϕ_t denote the tf-idf textual reference scoring function [9, 25].

A top-k distance-sensitive spatial keyword query is denoted by $Q = \langle x_q, y_q, w_1, w_2, \ldots, w_m, k\rangle$; where (x_q, y_q) denote the (longitude, latitude) location of the query, $\{w_1, w_2, \ldots, w_m\}$ denote the set of query keywords, and k denote the number of top-matching documents to be returned for the query.

The relevance score between a document $\mathcal{D}$ and a query Q is evaluated by a monotonic aggregation of textual relevance and spatial proximity. The textual relevance is measured by a scoring function $\phi_t(\mathcal{D}, Q)$ that sums up the relevance score of each query keyword:

$$\phi_t(\mathcal{D}, Q) = \sum_{w_i \in Q} \phi_t(\mathcal{D}, w_i) \quad (1)$$

If a document $\mathcal{D}$ does not contain any query keyword, we consider it irrelevant and set $\phi_t(\mathcal{D}, Q)$ to be $-\infty$ to prevent it from appearing in the results. The spatial proximity is measured by $\phi_s(\mathcal{D}, Q)$ which assigns a score based on the distance between $\mathcal{D}$ and Q such that a smaller spatial distance yields a higher $\phi_s(\mathcal{D}, Q)$ value. In this paper, we use the same spatial ranking function as in [14, 29, 35].

$$\phi_s(\mathcal{D}, Q) = 1 - \frac{d(\mathcal{D}, Q)}{\gamma} \quad (2)$$

where $d(\mathcal{D}, Q)$ measures the Euclidean distance between the locations of the query Q and document $\mathcal{D}$, and γ is a parameter to normalize $\phi_s(\mathcal{D}, Q)$ between $[0, 1]$ which is set to be the maximum distance among all the pairs of documents. Finally, as in [14, 16, 29, 35], we define the overall ranking function $\phi(\mathcal{D}, Q)$ as a linear combination of textual relevance and spatial proximity:

$$\phi(\mathcal{D}, Q) = \alpha \cdot \phi_t(\mathcal{D}, Q) + (1 - \alpha) \cdot \phi_s(\mathcal{D}, Q) \quad (3)$$

Based on the ranking function ϕ, we can formally define the top-k spatial keyword query as follows:

DEFINITION 1. *Top-k spatial keyword query*
Given a document corpus $\mathbb{C}$, a top-k spatial keyword query Q retrieves a set $O \subseteq \mathbb{C}$ with k documents such that $\forall \mathcal{D} \in O$ and $\mathcal{D}' \in \mathbb{C} - O$, $\phi(\mathcal{D}, Q) \geq \phi(\mathcal{D}', Q)$.

3. RELATED WORK

Spatial keyword search is a well-studied problem and can be categorized into two classes: *distance-insensitive* and *distance-sensitive*.

3.1 Distance-insensitive Approaches

The traditional local search engines such as Google Local [1] and Yahoo! Local [3] belong to the distance-insensitive category. In these systems, location information is first extracted from web pages and organized using a hierarchical geographical ontology. Given a query with a locational constraint, the candidate documents are retrieved from the matching hierarchies in the ontology using existing information retrieval methods such as TAAT [6–8, 30] or DAAT [11, 31].

In [36], Zhou et al. built separate R-tree [10] and inverted index for the spatial and textual attributes respectively; and evaluated a query by first searching with one of the indexes (spatial or textual) followed by ranking the matching documents using information from the other index.

In [13], the locational regions are first extracted from the web documents which are then used to organize the documents using the grid file [27] or space-filling curve [19] such that spatially close documents are clustered near to one another on disk. Given a query, documents whose locational regions contain the query location are first retrieved and their textual relevance wrt the query are are then computed.

In [18], Fontoura et al. proposed query relaxation techniques to generate top-k results when there are not enough matching documents in the specified region.

In [22], Hariharan et al. proposed KR*-tree, the first hybrid index to handle spatial keyword query with AND-semantics, where a matching document must contain all the query keywords and its query location must intersect with the query region. In KR*-tree, an inverted index is maintained for each tree node, where both spatial filtering and textual filtering can be employed at the same time when accessing an index node. In [32], Zhang et al. recently proposed a more efficient hybrid index to process queries with AND-semantics. The index combines linear quadtree [20] and fixed-length bitmap for query processing.

3.2 Distance-sensitive Approaches

In the distance-sensitive category of work [12, 14, 16, 29, 35], matching documents are ranked based on both their textual relevance and spatial proximity to the query.

In [16], Felipe et al. proposed IR^2 which is a hybrid index of the R-tree and signature file; their work is based on the AND-semantics which sorts the matching documents by their distance to the query location.

Cong et al. [14] proposed IR-tree which is a spatial-first approach that integrates the R-tree with inverted index; each R-tree node is augmented with inverted lists to index documents located within that node.

More recently, two textual-first approaches, S2I index [29] and I^3 [35], have been proposed. S2I uses aggregated R-tree [28] for spatial relevance evaluation. The search algorithm expands from the query location in the spatial index associated with each query keyword until k best results are found. I^3 builds one Quadtree [17] for each keyword and adopts the best-first search strategy in query processing. Performance studies [29, 35] have shown that both S2I and I^3 are significantly faster than IR-tree.

4. OUR APPROACH

In this section, we present a new approach for the top-k distance-sensitive spatial keyword search problem. Our approach is based on the simple but effective idea of modeling the problem as a top-k aggregation problem [15].

DEFINITION 2. *Top-k aggregation problem*
Consider a database $\mathbb{D}$ where each object $o = (x_1, x_2, \ldots, x_n)$ has n scores, one for each of its n attributes. Given a monotonic aggregation function f, where $f(o)$ or $f(x_1, x_2, \ldots, x_n)$ denote the overall score of object o, the top-k aggregation problem is to find a set of top-k objects in $\mathbb{D}$ with the highest overall scores; i.e., find $O \subseteq \mathbb{D}$ with k objects such that $\forall o \in O$ and $o' \in \mathbb{D} - O$, $f(o) \geq f(o')$.

The reformulation of the top-k distance-sensitive spatial keyword search problem as a top-k aggregation problem is straightforward. Given a query Q with m keywords, each geo-document D in a database $\mathbb{D}$ can be modeled as a $m+1$-tuple $(x_1, x_2, \ldots, x_{m+1})$; where x_i $(1 \leq i \leq m)$ is the textual relevance score (defined by ϕ_t) between D and the i^{th} query keyword, and x_{m+1} is spatial proximity score (defined by ϕ_s) between D and the query location. By setting the aggregation function to be ϕ in Equation 3, the top-k distance-sensitive spatial keyword query problem is now reformulated as a top-k aggregation problem.

EXAMPLE 1. *Consider again the example spatial keyword search problem in Figure 1. Since all the documents contain at least a query keyword, we have $\mathbb{D} = \{d_1, d_2, \ldots, d_7\}$. Each document D is now modelled as a three-dimensional vector (x_1, x_2, x_3); where x_1 is the D's textual relevance to the keyword "seafood", x_2 is D's textual relevance to the keyword "restaurant", and x_3 is the spatial proximity between D and the query. If we set the aggregation function to be ϕ, the top-k aggregation problem for this example would return the results with the highest scores ranked by ϕ which are exactly the results for the spatial keyword search problem.*

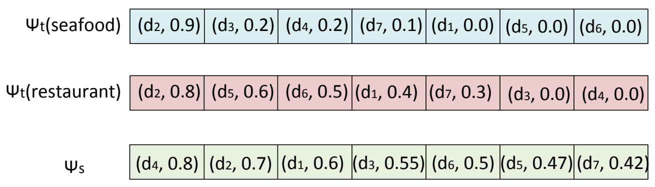

Ψt(seafood) (d2, 0.9) (d3, 0.2) (d4, 0.2) (d7, 0.1) (d1, 0.0) (d5, 0.0) (d6, 0.0)
Ψt(restaurant) (d2, 0.8) (d5, 0.6) (d6, 0.5) (d1, 0.4) (d7, 0.3) (d3, 0.0) (d4, 0.0)
Ψs (d4, 0.8) (d2, 0.7) (d1, 0.6) (d3, 0.55) (d6, 0.5) (d5, 0.47) (d7, 0.42)

Figure 2: An example of top-k aggregation for query "seafood restaurant"

By formulating the problem as a top-k aggregation problem, we are able to design an efficient solution that relies only on the simple and widely used inverted index in contrast to existing hybrid solutions. The rest of this section is organized as follows. We first review top-k aggregation algorithms in Section 4.1, and then explain how they are adapted as baseline algorithms for the top-k distance-sensitive spatial keyword search problem in Section 4.2.

4.1 Top-k Aggregation Algorithms

In this section, we present an overview of top-k aggregation algorithms. A comprehensive survey of these algorithms are given in [23].

For convenience, we present the algorithms in the context of spatial keyword search as follows. Given a query with m keywords and a geo-location, assume that we have constructed $m+1$ sorted lists, $L_1, L_2, \ldots, L_m$, wrt a database of geo-documents. Each L_i, $i \in [1, m]$, is sorted (in non-ascending order) by the documents' textual relevance scores wrt the i^{th} query keyword; and L_{m+1} is sorted (in non-ascending order) by the documents' spatial proximity to the query location. Thus, each sorted list entry is a pair of document identifier and a score (textual relevance or spatial proximity).

The first algorithm is the TA algorithm proposed by Fagin et al. [15], which consists of two main steps.

1. Perform a sorted access in parallel to each of the $m+1$ sorted lists L_i. For each document accessed, perform a random access to each of the other lists L_i to find its score in L_i. Then, compute the aggregated score of the document using the ranking function ϕ in Equation 3. If the computed aggregated score is one of the k highest we have seen so far, remember the document and its score.

2. For each list L_i, let $high[i]$ be the score of the last document seen under sorted access. Define the threshold value B_k to be the aggregated score of $high[i]$ by the ranking function ϕ. As soon as at least k documents have been seen whose score is at least equal to B_k, the algorithm terminates.

Although the TA algorithm has been proven to be instance optimal, the optimality depends on the cost of random access. The algorithm does not perform well if the cost of random access is too high. Subsequently, an improved variant of the TA algorithm, the CA algorithm [15], was proposed to achieve a good tradeoff between the number of sequential access and random access.

The CA algorithm uses a parameter h to control the depth of sequential access. In each iteration, h documents in each list are sequentially accessed. h is set to be the ratio of the cost of a random access to the cost a sequential access. For each accessed document doc, let $B(doc)$ denote an upper bound of the aggregated score of doc. A document doc is defined to be *viable* if $B(doc)$ is larger than the k^{th} best score that has been computed so far. At the end of each iteration, the viable document with the maximum $B(doc)$ value is selected for random access to determine its aggregated score. The algorithm terminates when at least k distinct documents have been seen and there are no viable documents.

Besides the TA and CA algorithms, there are a number of other variants that optimizes the approach for early termination. In [21], an improved TA variant, *Quick-Combine*, was proposed where instead of accessing the sorted lists in a round-robin manner, *Quick-Combine* first estimates the influence of each sorted list and selects the most influential list in each iteration for random access. In [26], Marian et al. proposed the *Upper and Pick* algorithm to conduct random access by minimizing the upper and lower bounds of all objects.

4.2 Baseline Algorithms

In this section, we explain how the standard top-k aggregation algorithms described in the previous section are adapted as baseline algorithms for the top-k distance-sensitive spatial keyword search problem.

First, note that while it is possible to precompute the sorted lists for the textual attributes (i.e., $L_1, L_2, \ldots, L_m$) since the tf-idf scores are independent of the query, this is not the case for the list L_{m+1} for the spatial attribute as the spatial proximity score is dependent on the query location. Thus, the sorted list L_{m+1} needs to be created at query time.

Second, since the documents matching a query's keywords are generally not clustered together (i.e., their docIDs are not related), performing random access to matching documents to retrieve the overall relevance score typically incurs a large number of random disk I/O. To further minimize the number of random disk I/O for such retrievals, we introduce a simple optimization to organize the single, large document list into multiple smaller ones by maintaining a document list of matching documents for each keyword. By clustering the document entries that match the same keyword, this optimization helps to reduce the number of random disk I/O incurred for document identifier lookups. However, this advantage is at the cost of additional storage for the document lists as a document's information is now replicated among several lists.

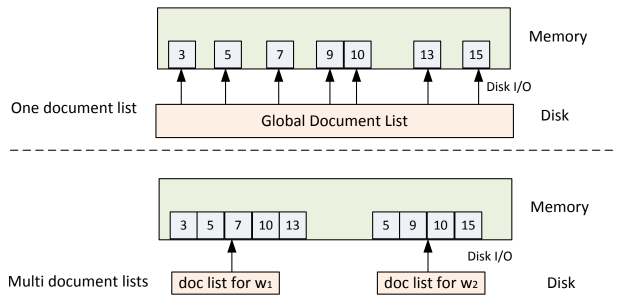


Figure 3: One global document list v.s. multiple small lists

EXAMPLE 2. *Figure 3 illustrates how the use of multiple document lists can help reduce disk I/O. In this example, keyword w_1 appears in documents $L_1 = \{3, 5, 7, 10, 13\}$ and keyword w_2 appears in documents $L_2 = \{5, 9, 10, 15\}$. If we need to perform random access on all the documents in L_1 and L_2, using a single, global document list could incur a total of 7 disk I/O (one for each distinct document); in contrast, with the use of multiple document lists, the number of disk I/O could be reduced to only 2 if each of the keyword-based document lists fits on one disk page.*

5. RANK-AWARE CA ALGORITHM

While the baseline algorithms presented in the previous section could be easily incorporated into existing search engines that support the ubiquitous inverted list index, the baseline algorithms have a performance drawback in that the spatial attribute list cannot be precomputed statically but need to be sorted at runtime using the query location to compute the spatial proximity values. In this section, we present an optimized variant of the CA algorithm, termed *Rank-aware CA (RCA)*, to address this limitation.

The key idea of our optimization is to sort the spatial attribute list offline based on an approximate spatial order preserving encoding such that the two-dimensional location attribute values are encoded into totally ordered values with the desirable property that a pair of encoded location values that are close together in the total order represents a pair of locations that are likely to be spatially close to each other. In this paper, we apply the well-known Z-order [4] to obtain such a mapping.

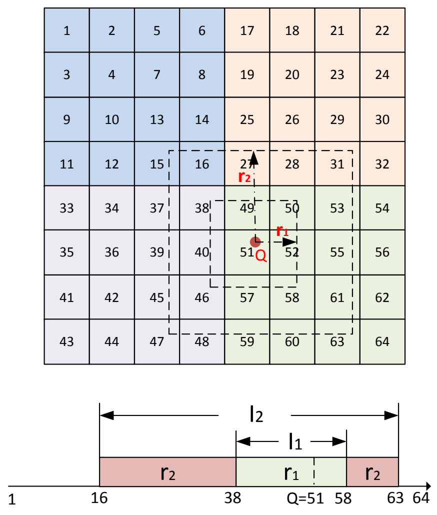


Figure 4: An example of Z-order encoding

EXAMPLE 3. *Figure 4 illustrates an application of Z-order to encode location values for a two-dimensional space that is partitioned into 8×8 cells. Applying the Z-order encoding, each cell is assigned a unique Z-order value from 1 to 64. Since each document is located within some cell, the document's location is represented by the Z-order value of its cell location. Note that the Z-order encoding provides an approximate preservation of spatial proximity.*

There are two useful properties of Z-order encoding that we exploit in our RCA algorithm. First, given any rectangular region R in the location space, the top-left corner cell of R has the smallest Z-order value (denoted by R_{min}) and the bottom-right corner cell of R has the largest Z-order value (denoted by R_{max}) among all the cells in R. Thus, all the cell locations in a region R are contained in the range of Z-order values $[R_{min}, R_{max}]$. As an example, consider a query Q that is located in cell 51 and a region R that is centered at Q with a radius of r_1 as shown in Figure 4. We have $[R_{min}, R_{max}]$ = $[38, 58]$ which contains the Z-order values of all the cells in R. Second, for any cell with a Z-order value c that is outside of a region R centered at Q with radius r_1 (i.e., $c < R_{min}$ or $c > R_{max}$), the distance of this cell from Q must be larger than r_1.

Based on the properties of the Z-order encoding, the RCA algorithm progressively accesses the documents in the spatial attribute list in iterations in a *score-bounded* manner. In each iteratoin, unlike conventional CA algorithm which explores a fixed number of items, our RCA algorithm accesses all the documents with spatial proximity score within a fixed-length score interval.

5.1 Score-bounded Access of Spatial Attribute Lists

In this section, we elaborate on the score-bounded access of the spatial attribute lists.

Similar to the rationale for organizing a document list into multiple shorter lists as explained in Section 4.2, the spatial attribute list is also organized as multiple shorter lists with one spatial inverted list L_w for each keyword w; i.e., the entries in L_w represent all the documents that contain the keyword w. The entries in L_w are sorted in ascending order of their Z-order encodings of the document locations, and these spatial inverted lists are created offline without incurring any runtime sorting overhead.

Assume that the spatial relevance scores are normalized to the range $(0, 1]$ by the scoring function ϕ_s. Let η_s denote the maximum number of iterations to be used to access all the documents in the spatial attribute lists. Therefore, spatial score range $(0, 1]$ is partitioned into η_s disjoint intervals (each of length $\frac{1}{\eta_s}$): $T_1 = (1 - \frac{1}{\eta_s}, 1]$, $T_2 = (1 - \frac{2}{\eta_s}, 1 - \frac{1}{\eta_s}]$, …, $T_{\eta_s} = (0, \frac{1}{\eta_s}]$; where T_i denotes the range of spatial proximity score values for the documents accessed in the i^{th} iteration. By Equation 2, it follows that for the i^{th} iteration, we have $\phi_s = 1 - \frac{d(\mathcal{D}, Q)}{\gamma} > 1 - \frac{i}{\eta_s}$. Thus, $d(\mathcal{D}, Q) < \frac{i\gamma}{\eta_s}$ for the i^{th} iteration. In other words, in the i^{th} iteration of the score-bounded access, the search radius for that iteration is set to $r_i = \frac{i\gamma}{\eta_s}$; and the radius progressively increases over the iterations.

At each iteration of the score-bounded access, the search on a spatial inverted list is actually performed in terms of a forward-scan and a backward-scan. Consider again the example in Figure 4 where the Z-order encoding of the query location (denoted by z_q) is in cell 51 (i.e., $z_q = 51$). With an initial radius of r_1, we access the documents located in the Z-order range $I_1 = [38, 58]$ as shown in Figure 4. At the next iteration when the radius is increased to r_2, we have the Z-order range $I_2 = [15, 63]$ which contains I_1. To access only the documents contained in I_2 but not in I_1, the spatial search is split into a backward-scan of Z-order range $[15, 37]$ and a forward-scan of Z-order range $[58, 63]$.

Based on the properties of Z-order, it is possible that some of the documents accessed in the searched spatial region (specified by some range of Z-order values) could be false positives; i.e, the actual distance between the accessed document and query could be larger than the current search radius r_i at the i^{th} iteration. To avoid processing these false positives too early, we maintain η_s buffers to temporarily store these false positive documents: a false positive document that should have been processed later in the j^{th} iteration (i.e., $j > i$) will be temporarily stored in the j^{th} buffer. Thus, the documents in the j^{th} buffer will be considered later during the j^{th} iteration.

EXAMPLE 4. *Figure 5 illustrates an example of scored-based access for the spatial attribute. The algorithm starts from the region with radius $\lambda_s = \frac{\gamma}{\eta_s}$ which is then progressively increased to $2\lambda_s$,*

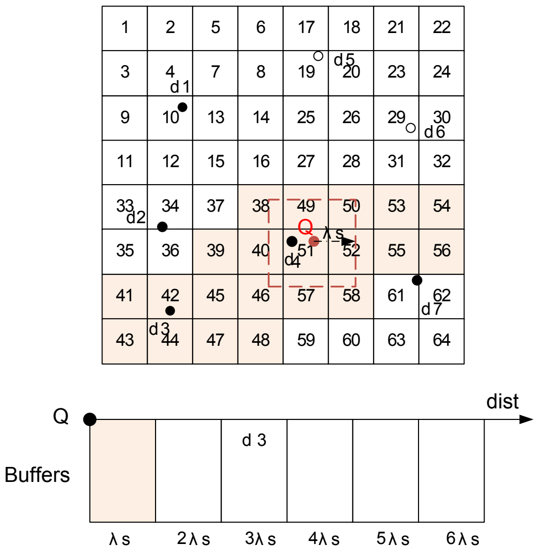


Figure 5: Buffering false positives in spatial search

$3\lambda_s$, etc. until it has found k documents with the highest scores. For the iteration with radius = λ_s, the Z-order range in this iteration is $[38, 58]$ and two documents $\{d_3, d_4\}$ are accessed during the scan of the spatial list. Among them, only d_4 is a true candidate. d_3 is a false positive and it is temporarily stored in the appropriate buffer to be considered in a subsequent iteration.

In contrast to the CA algorithm, which accesses a fixed number of documents in each iteration, the documents accessed by RCA is determined by a score interval corresponding to the iteration. This difference between CA and RCA is motivated by two reasons. First, the upper bound score for the ranking function ϕ_s (Equation 2) relies on the minimum distance of all the unseen objects to the query location and this distance computation is complex as each Z-order range in general corresponds to an irregular polygon. Second, the upper bound score for ϕ_s could decrease very slowly if a fixed number of documents is accessed per iteration. This is because the Z-order encoding only preserves spatial proximity approximately and spatially close objects could have Z-order values that are far apart in the linear order. For example, in Figure 4, although the minimum distance between cells 16 and 49 is zero, the difference in their Z-order values is 33. As another example, if RCA were to adopt CA's approach of accessing a fixed number of objects per iteration, then there is actually no change in the upper bound spatial relevance score from the objects accessed in cell 49 to cell 17.

5.2 Score-bounded Access of Textual Attribute Lists

Recall that our ranking function ϕ is a linear weighted combination of spatial proximity and textual relevance with the weight parameter α. Intuitively, when the value of α is small, the spatial relevance is more important than textual relevance; and it is therefore desirable to examine more documents in the spatial inverted lists than the textual inverted lists so that the upper bound textual relevance score for the unseen documents can decrease more significantly to enable an earlier termination of the algorithm.

To achieve the above property, the RCA algorithm also adopts the score-bounded access method for the textual attribute lists. Specifically, let η_t denote the maximum number of iterations to be used to access the textual attribute lists. Then, the textual relevance score domain $(0,1]$ is partitioned into η_t intervals where documents whose textual relevance score is within $(1-\frac{i}{\eta_t}, 1-\frac{i-1}{\eta_t}]$ are accessed in the i^{th} iteration. In this way, our RCA algorithm enables more relevant documents to be accessed before less relevant ones. At the end of the i^{th} iteration, the upper bound textual relevance score for the unseen documents is given by

$$B_t(i) = 1 - \frac{i}{\eta_t} \tag{4}$$

It follows that the parameters η_s and η_t are related as follows:

$$\eta_t = \frac{(1-\alpha)}{\alpha} \cdot \eta_s \tag{5}$$

EXAMPLE 5. *Consider once more the spatial keyword query with keywords "seafood restaurant", and assume that $\eta_t = 4$ (i.e., the length of the score interval is $\lambda_t = 0.25$). Figure 6 shows the documents that are score-bounded accessed for each of the four iterations w.r.t. the two inverted lists for the query keywords. In the first iteration, we access documents whose textual relevance score is within $(0.75, 1]$ and d_2 is accessed in both inverted lists. The upper bound textual relevance score for the unseen documents in each list is updated to be 0.75. In the second iteration, no documents are accessed for keyword "seafood" and only d_5 is accessed. In this way, our score-bounded approach prioritises the document retrievals to access documents that are more likely to be in the top-k results earlier. In contrast, the conventional CA algorithm will access the documents d_4, d_3 and d_7 with low relevance scores in the inverted list of "seafood" very early.*

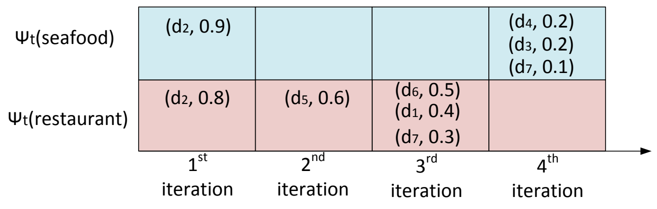


Figure 6: Score-bounded sequential access for keywords "seafood restaurant"

5.3 Overall Approach

In this section, we discuss the overall approach for our RCA algorithm. Our approach adopts a different random access strategy and termination criteria from the traditional CA algorithm. Recall that the CA algorithm selects the *viable* document with the maximum upper bound score for random access and the algorithm terminates if this upper bound score is no greater than the score of the k^{th} best document seen so far (denoted by W_k). In contrast, our RCA algorithm does not maintain $B(doc)$ to store the upper bound score for each viable document. Moreover, our random access is applied to the viable documents in the min-heap storing top-k results so that W_k can be increased as much as possible towards an early termination. Based on our score-bounded sequential access approach, we can also compute a tighter upper bound for a document's aggregated score (denoted by B_k). Specifically, after the i^{th} iteration, B_k is given by is calculated by

$$B_k = \alpha \cdot m \cdot B_t(i) + (1-\alpha) \cdot B_s(i)^2 \tag{6}$$

If $B_k \leq W_k$, we stop the sequential access on the sorted lists as it is guaranteed that that no unseen document could have an aggregated score higher than W_k. However, the algorithm cannot be terminated at this point because there could be some viable documents not in the top-k heap but with a upper bound score larger than W_k. For these documents, we need to conduct random access to get their full aggregated score and update the top-k heap if we find a better result. In this way, we can guarantee no correct result is missed.

Our rank-aware CA algorithm to process top-k spatial keyword queries is shown in Algorithm 1. The input parameter L_t is the collection of textual lists sorted by ϕ_t values and L_s is the collection of spatial lists sorted by Z-order encoding values. In lines 1 and 2, a top-k heap and η_s buffers are initialized, where $buf[i]$ is used to temporarily store any false positive documents to be processed during the i^{th} iteration. Three pointers p_t, p_f and p_b are used for sequential access: p_t is used for the textual lists and p_s (resp. p_b) is used for the forward (resp. backward) scan in the spatial lists (lines 6-8). In each iteration, we perform sequential access in the textual lists by calling exploreTextList (line 10) and forward/backward scans in the spatial lists by calling exploreForwardSpatialList and exploreBackwardSpatialList (lines 11-13). The function exploreTextList accepts the parameter $B_t(i)$ defined in Equation 4 so as to scan the documents whose relevance is in the range $(B_t(i), B_t(i-1)]$. Similarly, we calculate the Z-order range from the current search radius and perform forward and backward sequential access of documents within the computed Z-order range in the spatial lists. Any false positive documents are stored in the appropriate buffers in buf and examined in subsequent iterations (lines 14-15 in Algorithm 1). After the sequential access, we perform random access on the viable documents in the top-k heap (lines 16-17 in Algorithm 1). Finally, we update B_k according to Equation 6. If $B_k \leq W_k$, we stop the sequential access and for each viable document not in $topk$, we perform random access to obtain its complete score and update the top-k results if it is a better result.

6. EXPERIMENT EVALUATION

In this section, we compare the methods derived from top-k aggregation with state-of-the-art indexes that combine spatial partitioning and textual partitioning. More specifically, we compare the performance of TA, CA and RCA proposed in this paper with S2I [24] and I^3 [35] in processing top-k spatial keyword queries. We set $h = 8$ in the CA Algorithm and $\eta_t = 20$ in the RCA algorithm. All the methods are disk-based and implemented in Java. We conduct the experiments on a server with 48GB memory and Quad-Core AMD Opteron(tm) Processor 8356, running Centos 5.8.

6.1 Twitter Datasets

We use a Twitter dataset for the experiments. Our collection contains 100 million real geo-tweets that take up to 7.7GB in storage in the raw data format. For scalability evaluation, we sample four subsets whose sizes vary from 20 million to 80 million. The statistics of these datasets are shown in Table 1, where we report the dataset size, number of distinct keywords, average number of keywords in a document and amount of disk storage for each dataset.

In Table 2, we report the disk size of our inverted index and the comparison indexes. To support Rank-aware CA (RCA), we build three inverted lists for each keyword and use MapDB [3] to store all

[2]More precisely, the upper bound of textual relevance can be further reduced to $\alpha \cdot \sum_{1 \leq j \leq m} s_j$, where s_j is the score last seen in each textual list. Since the i^{th} iteration in the j^{th} textual list terminates when we access a document with a score $s_j \notin T_i$, we can be assured that $s_j \leq B_t(i)$.

[3]`http://github.com/jankotek/mapdb`

Table 1: The Twitter Datasets

DataSets	Number of geo-tweets	Number of distinct keywords	Average number of keywords	Disk storage
Twitter20M	20,000,000	2,719,087	6.92	1.6GB
Twitter40M	40,000,000	4,261,582	6.94	3.1GB
Twitter60M	60,000,000	5,530,216	6.95	4.6GB
Twitter80M	80,000,000	6,652,879	6.94	6.2GB
Twitter100M	100,000,000	7,672,170	6.94	7.7GB

the lists. Since TA and CA do not need the lists sorted by z-order, we let them share the inverted lists of RCA that are sorted by textual relevance and document id. As shown in Table 2, inverted index consumes only slightly more disk space than S2I. Although we maintain three inverted lists for one keyword while S2I builds one R-tree for one keyword, the inverted lists have higher disk utilization than R-tree and can be easily compressed to save disk space. I^3 allocates at least one disk page for an infrequent keyword but S2I merges them in one file. Thus, I^3 consumes the most disk space.

6.2 Query Set

We use real keyword queries from AOL search engine [4] to generate spatial keyword queries. First, we select hotel and restaurant as two typical location-based queries. All the keyword queries in the log containing "hotel" or "restaurant" are extracted. Among them, we keep the queries whose number of keywords are from 2 to 6. After removing duplicate queries, we note that, as shown in Table 3, queries with 3 keywords are the most common and hotel queries are more frequently submitted by users than restaurant queries. Next, we attach to each keyword query a spatial location. The location is randomly sampled and follows the same distribution as the tweet location.

6.3 Parameter Setup

In the following experiments, we will evaluate the performance of query processing in terms of increasing dataset size (from 20 million to 100 million), varying number of query keywords m (from 2 to 6), the number of query results k (from 10 to 200) and textual relevance weight α in the ranking function (from 0.1 to 0.9). The values in *bold* in Table 4 represent the default settings. In each experiment, we vary one parameter and fix the remaining parameters at their default values. The performance is measured by the average latency of query processing. Given a query set with thousands of hotel or restaurant queries, we start the timing when the first query arrives and stop when the last query finishes. Thus, the query processing time includes the I/O cost to load the index into memory. We do not set a cache limit for query processing. The part of index that is loaded in memory will stay as cached until all the keyword queries are processed.

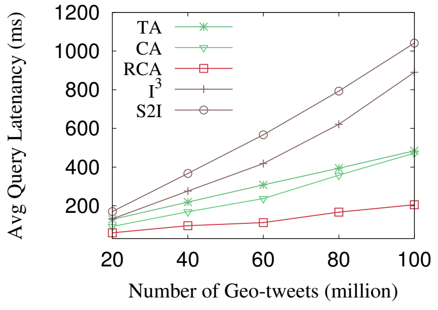
6.4 Query Results

The query results are top-k geo-tweets sorted by spatial proximity and textual relevance. Since tweets are normally short text, we merge the tweets with the same location into one geo-document. For example, a restaurant may be checked in multiple times and the term frequency can reflect the popularity of the location. Table 5 illustrates an example of top-10 geo-tweets for a query "seafood restaurant" submitted at location $(40.7312, -73.9977)$, corresponding to Washington Square Park in Manhattan, New York City. In this example, α is set to 0.3 in favor of geo-tweets closer to the user location. If there are multiple tweets at the same location, we only select a representative one for presentation purpose.

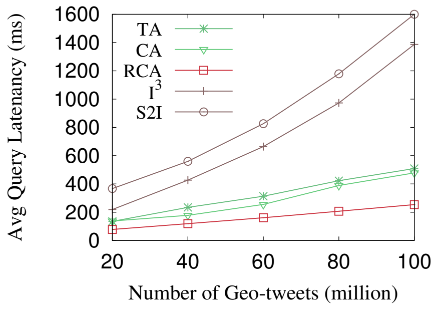
[4] `http://www.gregsadetsky.com/aol-data/`

6.5 Increasing Dataset Size

In this experiment, we examine the scalability in terms of increasing dataset size. We increase the number of geo-tweets from 20 million to 100 million and run the sets of 3-keyword hotel and restaurant queries (the number is 7,831 and 3,844 respectively). We report the average query processing latency of different methods in Figure 7. As can be seen, the inverted-index-based solutions, including TA, CA and RCA, achieve much better performance than the hybrid indexes that combine spatial partitioning and textual partitioning. S2I and I^3 maintain a spatial index for each keyword. Given a set of keywords, they start from the query location and expand the search region from the spatial attribute only. The documents closest to the query location will be accessed first, regardless of the textual relevance. In addition, the spatial index is disk-based and the tree nodes are accessed with a large number of random I/Os. Therefore, their performance do not scale well. When the dataset size increases to 100 million, the query latency is 5 times worse than that of RCA.



(a) Hotel Queries



(b) Restaurant Queries

Figure 7: Query latency of increasing dataset size

TA and CA demonstrate comparable performance. Although they need to sort the lists by the distance to the query location, the performance is still around 2 times better than state-of-the-art solutions. TA terminates earlier than CA because most of the doc-

Algorithm 1: RCA(Q, L_t, L_s, m, k, η_t, η_s)

```
1.  initialize a min-heap $topk$ with $k$ dummy documents with score 0
2.  initialize $\eta_s$ buffers $buf[1..\eta_s]$ for false positive documents
3.  $W_k \leftarrow 0$ // $W_k$ is the minimum score in $topk$
4.  $B_k \leftarrow 1$ // $B_k$ is the upper bound score for all the unseen documents
5.  for $i = 1; i \le m; i++$ do
6.      $p_t[i] \leftarrow 0$
7.      $p_f[i] \leftarrow binary\_search(slists[i], z_q)$ // $z_q$ is the Z-order of $Q$
8.      $p_b[i] \leftarrow p_f[i] - 1$
9.  for $i = 1; i \le \max(\eta_t, \eta_s); i++$ do
10.     exploreTextList($B_t(i)$)
11.     obtain the minimum Z-order $z_{min}$ and maximum Z-order $z_{max}$ from
        the rectangle $(Q.lat - i\lambda_s$, $Q.lat + i\lambda_s$, $Q.lng - i\lambda_s$, $Q.lng + i\lambda_s)$
12.     exploreForwardSpatialList($z_{min}$, $i \cdot \lambda_s$)
13.     exploreBackwardSpatialList($z_{max}$, $i \cdot \lambda_s$)
14.     for $doc \in buf[i]$ do
15.         seqAccess($doc$, $m+1$, $(1-\alpha) \cdot \phi_s(doc, Q)$)
16.     for each viable document $doc$ in $topk$ do
17.         randomAccess($doc$)
18.     $B_k \leftarrow \alpha \cdot m \cdot B_t(i) + (1-\alpha) \cdot B_s(i)$
19.     if $W_k \ge B_k$ then
20.         for $doc \in W$ do
21.             if $doc \notin topk$ and $doc$ is viable then
22.                 randomAccess($doc$)
23.         break
24. return $topk$
```

```
exploreTextList($minT$)
1.  for $i = 1; i \le m; i++$ do
2.      while $L_t[i][p_t[i]].score > minT$ do
3.          $doc \leftarrow L_t[i][p_t[i]]$
4.          seqAccess($doc, i, \alpha \cdot \phi_t(doc, Q.w_i)$)
5.          $p_t[i] \leftarrow p_t[i] + 1$
```

```
exploreForwardSpatialList($maxZ$, $radius$)
1.  for $i = 1; i \le m; i++$ do
2.      while $L_s[i][p_f[i]].order \le maxZ$ do
3.          $doc \leftarrow L_s[i][p_f[i]]$
4.          if $d(doc, Q) \le radius$ then
5.              seqAccess($doc, m+1, (1-\alpha) \cdot \phi_s(doc, Q)$)
6.          else
7.              put $doc$ in $buf[\lfloor \frac{d(doc,Q)}{\lambda_s} \rfloor]$
8.          $p_f[i] \leftarrow p_f[i] + 1$
```

```
exploreBackwardSpatialList($minZ$, , $radius$)
1.  for $i = 1; i \le m; i++$ do
2.      while $L_s[i][p_b[i]].order \ge minZ$ do
3.          $doc \leftarrow L_s[i][p_b[i]]$
4.          if $d(doc, Q) \le radius$ then
5.              seqAccess($doc, m+1, (1-\alpha) \cdot \phi_s(doc, Q)$)
6.          else
7.              put $doc$ in $buf[\lfloor \frac{d(doc,Q)}{\lambda_s} \rfloor]$
8.          $p_b[i] \leftarrow p_b[i] - 1$
```

```
seqAccess($doc$, $i$, $s$)
1.  $W(doc) \leftarrow W(doc) + s$
2.  $E(doc) \leftarrow E(doc) \cup i$
3.  updateTopk($doc$)
```

```
randomAccess($doc$)
1.  if $doc$ is not accessed then
2.      for $i = 1; i \le m; i++$ do
3.          $W(doc) \leftarrow W(doc) + \alpha \cdot \phi_t(doc, Q.w_i)$
4.      $W(doc) \leftarrow W(doc) + (1-\alpha) \cdot \phi_s(doc, Q)$
5.      updateTopk($doc$)
```

```
updateTopk($doc$)
1.  if $W(doc) > W_k$ then
2.      update $doc$ in $topk$
3.      $W_k \leftarrow \min_{doc \in topk} W(doc)$
```

Table 2: Disk space occupation of different indexes

DataSets	**Inverted Index**	**S2I**	I^3
Twitter20M	12GB	11GB	24GB
Twitter40M	21GB	21GB	43GB
Twitter60M	35GB	31GB	62GB
Twitter80M	47GB	41GB	79GB
Twitter100M	58GB	51GB	97GB

Table 3: Statistics of hotel and restaurant queries

Number of keywords	Hotel Queries	Restaurant Queries
2	3,471	3,319
3	8,436	5,314
4	7,831	3,844
5	4,116	1,758
6	1,619	587

uments only contain part of the query keywords. For these documents, even if all the contained keywords have been accessed, their upper bound score is still higher than the real score. Therefore, B_k decreases slowly in CA. TA does not need to maintain the upper bound score for each document and terminates earlier. However, it incurs the overhead of a larger number of random accesses. Each random access includes at most m times of binary search on the sorted lists that have been loaded in memory and the cost of random access is moderate. Consequently, the query processing time in TA and CA is close.

RCA achieves the best performance among all the methods. Even in the dataset with 100 million tweets, it takes around 200ms to answer a query which is 2 times better than TA and CA. The main reason is that it does not need to sort the spatial lists online and the rank-aware expansion is effective in saving the cost of sequential access. Table 6 shows the average fraction of documents sequentially traversed by the different top-k aggregation algorithms. TA uses much smaller number of sequential access than CA but it requires a random access for each document retrieved from the sequential traversal. RCA can be considered as a compromise between TA and CA with a moderate number of sequential access and random access.

Finally, restaurant queries take a relatively longer time than hotel queries to answer. This is not because restaurant is a more frequent keyword in the Twitter dataset. In fact, the average length of an inverted list for keyword that appears in restaurant queries is $53,967$ but this number is $73,045$ for hotel queries. Instead, our investigation shows that this performance difference is due to the memory cache. In this experiment, we run $7,831$ hotel queries, which is much higher than the $3,844$ restaurant queries. This means more inverted lists about hotels are cached in memory and the disk I/Os can be saved when the cached keyword appears in subsequent queries. Note that the comparison among different methods is fair enough because all of them adopt the same caching mechanism.

6.6 Increasing k

Table 4: Parameter Setting

Dataset size	20M, 40M, **60M**, 80M, 100M
Number of query keywords	2, **3**, 4, 5, 6
k	10, **50**, 100, 150, 200
α	0.1, **0.3**, 0.5, 0.7, 0.9

Table 5: Example tweet results for query "seafood restaurant"

1	40.717	-73.997	I'm at Dun Huang Seafood Restaurant (New York, NY) http://t.co/jxx7FRg56i
2	40.715	-73.996	APALA holiday dinner! #union #labor #aapi @ Sunshine 27 Seafood Restaurant
3	40.714	-73.996	Moms picking out a fish for dinner #chinatown @ Fuleen Seafood Restaurant
4	40.714	-73.996	I'm at East Seafood Restaurant (New York, NY) http://t.co/IGA8Am8ph9
5	40.707	-74.018	Mediterranean (@ Miramar Seafood Restaurant) https://t.co/W9ZhS0ZKD0
6	40.759	-73.982	Dinner. (@ Oceana Seafood Restaurant Bar)
7	40.813	-73.955	I'm at Seafood Boca Chica Restaurant (New York, NY) http://t.co/oAEjm15drA
8	40.645	-73.995	I'm at New Fulin Kwok Seafood Restaurant (Brooklyn, NY) http://t.co/qr7rdhpbk0
9	40.768	-73.911	Seafood Egyptian restaurant @ Sabrys
10	40.814	-73.956	good 1 (@ Seafood Restaurant) http://t.co/INvKPKNNHF

Table 6: Fraction of sequential access for hotel queries

	TA	CA	RCA
Twitter20M	0.085	0.449	0.230
Twitter40M	0.073	0.369	0.227
Twitter60M	0.065	0.320	0.223
Twitter80M	0.060	0.294	0.219
Twitter100M	0.057	0.277	0.217

The average query latency of the various algorithms as k increases from 10 to 200 is shown in Figure 8(a) and 8(b). The performance of S2I and I^3 degrade more significantly than the solutions based on top-k aggregation. This is because they access the documents in the order of the distance to the query location. Hence, their pruning power only relies on the spatial attribute and the textual relevance is not taken into account. The remaining methods consider the spatial and textual relevance as a whole and demonstrate better scalability to k. The results also shed insight on the effectiveness of search space pruning among TA, CA and RCA. In TA and CA, the query processing time include the cost to sort by spatial proximity and the cost of sequential and random access. As k increases, the overhead of sorting is fixed and the running time increases because more documents are accessed when k becomes larger. Hence, we can compare the pruning effectiveness of TA, CA and RCA from the experiment figures. As k increases from 10 to 200, the running time of RCA increases much slower than TA and CA, which means our rank-aware expansion is effective in reducing the access cost in the sorted lists.

6.7 Increasing Number of Query Keywords

In this experiment, we increase the number of query keywords m from 2 to 6 and evaluate the performance in Twitter60M dataset. Since a document is considered relevant if it contains at least one query keyword, the number of candidates grows dramatically as m increases. The average query latency is reported in Figure 8(c) and 8(d). As can be seen, the performance of spatial keyword query processing degrades dramatically as m increases. When the number of keywords increases from 5 to 6, the running time doubles for S2I and I^3. This is because as m increases, the number of documents (containing at least one of the query keywords) whose locations are near the query location also increases. Though many of these documents' relevance scores are too low to be among the top-k results, they still need to be accessed. In comparison, TA, CA and RCA scale smoothly with m; the pruning mechanism takes into account both spatial and textual relevance which is clearly more effective.

6.8 Increasing α

In the last experiment, we evaluate the effect of the weight α in the ranking function ϕ on the performance. As α decreases, the spatial relevance plays a more important role in determining the final score. We can see from Figure 8(e) and 8(f) that the performance of S2I and I^3 improves significantly as α decreases. This is because S2I and I^3 examine documents near the query location first while the textual relevance is ignored. When α is 0.9, the textual relevance dominates spatial relevance and the spatial proximity is no longer important. However, S2I and I^3 access documents based on their distance to the query location. Even if the nearby documents are not textually relevant, they need to examine all of them. When α decreases, the spatial relevance becomes more important and the top-k results are more likely to be located around the query location. This is advantageous for the expansion strategy of S2I and I^3. That's why the running time of $\alpha = 0.1$ is nearly two times faster than that of $\alpha = 0.9$.

TA and CA are not sensitive to α. They build inverted lists sorted by textual relevance and spatial proximity. Hence, when the ranking function is biased on textual relevance, say $\alpha = 0.9$, the most relevant documents are likely to appear in the front of the inverted lists sorted by textual relevance. Similarly, when α is small, the spatial proximity is more important and the documents close to the query location will be accessed first by TA and CA algorithms. RCA, however, is affected by α. When α decreases, its performance improves because its inverted lists on the spatial attribute are not strictly sorted by the distance to the query location. When α is small, the top-k documents are close and the spatial expansion on the z-order list can be terminated earlier.

7. CONCLUSION

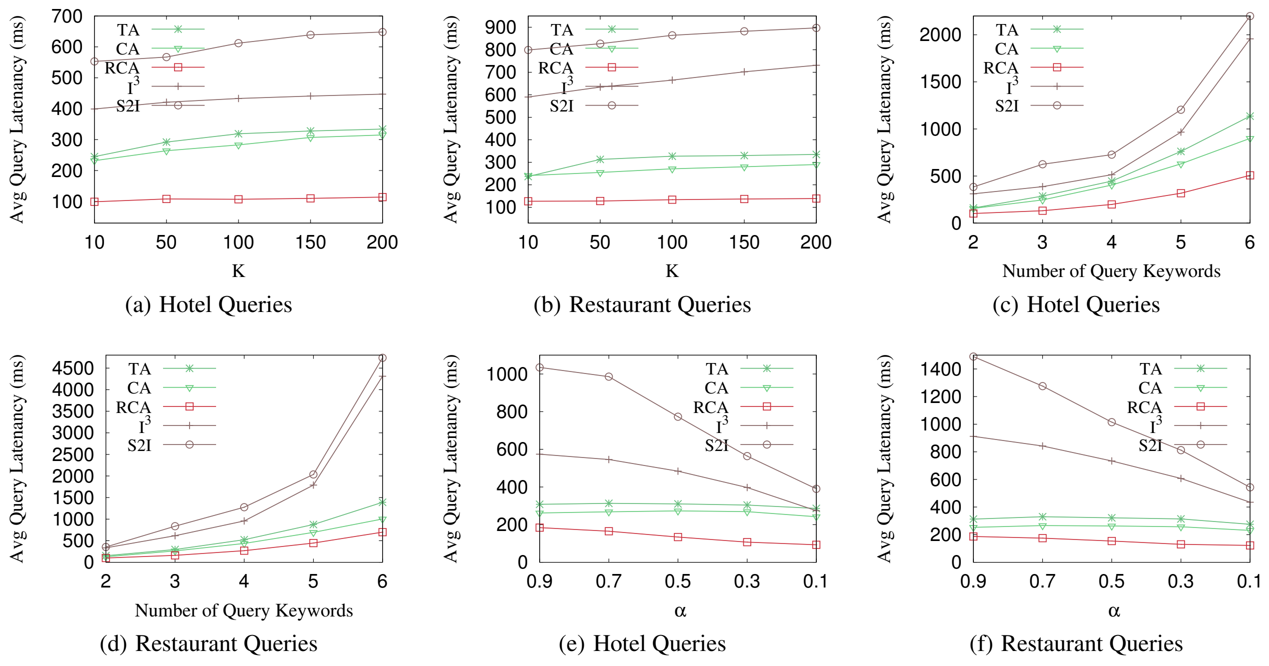
In this paper, we process *distance-sensitive* spatial keyword query as a top-k aggregation query and present the revised TA and CA algorithm for query processing. Furthermore, we propose a rank-aware CA algorithm that works well on inverted lists sorted by textual relevance and spatial curving order. We conduct experiments on Twitter dataset with up to 100 million geo-tweets. Our experimental results show that our proposed rank-aware CA scheme is superior over state-of-the-art solutions.

8. ACKNOWLEDGEMENT

This work is funded by the NExT Search Centre (grant R-252-300-001-490), supported by the Singapore National Research Foundation under its International Research Centre @ Singapore Funding Initiative and administered by the IDM Programme Office.

9. REFERENCES

[1] Google Local https://local.google.com/.
[2] http://techcrunch.com/2013/10/03/mobile-twitter-161m-access-from-handheld-devices-each-month-65-of-ad-revenues-coming-from-mobile/.
[3] Yahoo Local http://local.yahoo.com/.
[4] *Computer Graphics: Principles and Practice, second edition.* Addison-Wesley Professional, 1990.
[5] E. Amitay, N. Har'El, R. Sivan, and A. Soffer. Web-a-where: geotagging web content. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 273–280, New York, NY, USA, 2004. ACM.



(a) Hotel Queries (b) Restaurant Queries (c) Hotel Queries

(d) Restaurant Queries (e) Hotel Queries (f) Restaurant Queries

Figure 8: Query latency of increasing *k*, *m* and α

[6] V. N. Anh, O. de Kretser, and A. Moffat. Vector-space ranking with effective early termination. In *SIGIR*, pages 35–42, 2001.

[7] V. N. Anh and A. Moffat. Simplified similarity scoring using term ranks. In *SIGIR*, pages 226–233, 2005.

[8] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *SIGIR*, pages 372–379, 2006.

[9] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[10] 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 Conference*, pages 322–331, 1990.

[11] C. Buckley and A. F. Lewit. Optimization of inverted vector searches. In *SIGIR*, pages 97–110, 1985.

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

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

[14] 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.

[15] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

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

[17] R. A. Finkel and J. L. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 1974.

[18] M. Fontoura, V. Josifovski, R. Kumar, C. Olston, A. Tomkins, and S. Vassilvitskii. Relaxation in text search using taxonomies. *PVLDB*, 1(1):672–683, 2008.

[19] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.

[20] I. Gargantini. An effective way to represent quadtrees. *Commun. ACM*, 25(12):905–910, 1982.

[21] U. Güntzer, W.-T. Balke, and W. Kießling. Optimizing multi-feature queries for image databases. In *VLDB*, pages 419–428. Morgan Kaufmann, 2000.

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

[23] I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

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

[25] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[26] A. Marian, N. Bruno, and L. Gravano. Evaluating top-*k* queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[27] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

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

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

[30] T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In *SIGIR*, pages 175–182, 2007.

[31] T. Strohman, H. R. Turtle, and W. B. Croft. Optimization strategies for complex queries. In *SIGIR*, pages 219–225, 2005.

[32] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top k spatial keyword search. In *ICDE*, pages 901–912, 2013.

[33] 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*, pages 688–699, 2009.

[34] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in web 2.0. In *ICDE*, pages 521–532, 2010.

[35] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013.

[36] 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.