Reverse Keyword-Based Location Search

Xike Xie[†‡1], Xin Lin[♯2], Jianliang Xu[§3], Christian S. Jensen[♮4]

[†]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[‡]Suzhou Institute for Advanced Study, University of Science and Technology of China, Suzhou, China
[♯]Shanghai Key Laboratory of Multidimensional Information Processing, East China Normal University, Shanghai, China
[§]Department of Computer Science, Hong Kong Baptist University, Hong Kong, China
[♮]Department of Computer Science, Aalborg University, Aalborg, Denmark
[1]xkxie@ustc.edu.cn, [2]xlin@cs.ecnu.edu.cn, [3] xujl@comp.hkbu.edu.hk, [4]csj@cs.aau.dk

***Abstract*—The proliferation of geo-textual data gives prominence to spatial keyword search. The basic top-k spatial keyword query, returns k geo-textual objects that rank the highest according to their textual relevance and spatial proximity to query keywords and a query location. We define, study, and provide means of computing the reverse top-k keyword-based location query. This new type of query takes a set of keywords, a query object q, and a number k as arguments, and it returns a spatial region such that any top-k spatial keyword query with the query keywords and a location in this region would contain object q in its result. This query targets applications in market analysis, geographical planning, and location optimization, and it may support applications related to safe zones and influence zones that are used widely in location-based services.**

We show that computing an exact query result requires evaluating and merging a set of weighted Voronoi cells, which is expensive. We therefore devise effective algorithms that approximate result regions with quality guarantees. We develop novel pruning techniques on top of an index, and we offer a series of optimization techniques that aim to further accelerate query processing. Empirical studies suggest that the proposed query processing is efficient and scalable.

I. Introduction

So-called spatial web objects are objects accessible on the web that are both geo-tagged and textually annotated. Spatial web objects may represent different kinds of points of interest such as restaurants and tourist attractions. Given a set of keywords, a location, and a number k, a prototypical top-k spatial keyword query intuitively returns k spatial web objects that are most textually relevant to the query keywords and nearest to the query location. For example, this kind of query enables smartphone users to find nearby points of interest that are relevant to keywords [1]–[5].

We study a different kind of query that takes a set of keywords ψ, a spatial web object q, and a number k as arguments. It then returns a spatial region such that object q would belong to the result of a top-k spatial keyword query with query keywords ψ and any location in the region as arguments. We call this the reverse top-k keyword-based location (*RTkKL*) query. It targets applications in market analysis and decision support. For example, a business wants to identify the geographical region in which a spatial web object (e.g., a hotel) is highly ranked under some search keywords (e.g., "Free Wi-Fi" and "Fine food"). Follow-up marketing and promotion campaigns can then be launched in this region, e.g., budgeting for billboards or posters required.

The RTkKL query can also support applications that involve the concepts of influence zone [6]–[8] and safe zone [9]–[15], if the objects being considered are associated with textual descriptions and locations.

- Studies on influence zones [6]–[8] assume that a facility has high influence on a location if the facility is the nearest one to that location, or is among the k nearest ones. The influence zone of a facility contains all points that are closer to that facility than to other facilities. Studies involving influence zones target applications in spatial planning and location optimization.
- Studies on safe zones [9]–[15] often consider a moving query point that continuously requests k nearest facilities. A safe zone for a top-k result contains the query location and has the property that the top-k result is the same for all locations in the zone. Then, the top-k result needs to be recomputed only if the query point exits the safe zone.

Voronoi-based concepts are usually used for defining such kinds of spatial regions. Given a set S of spatial point objects, the Voronoi cell for object $q' \in S$ is the part of the space that contains all points in the underlying space that have q' as their nearest neighbor. Object q' is called the cell's site. This concept can be extended to k-nearest neighbors. If the site of a Voronoi cell is a set s of objects ($s \subseteq S$ and $|s| = k$), the cell is called order-k Voronoi cell. Every point of the underlying space in the cell takes s as their k nearest neighbors.

Existing techniques [6]–[15] that work for plain Voronoi cells cannot be used for computing RTkKL queries because they take only spatial proximity into account. The textual relevance is not addressed.

There have also been studies on Voronoi cells for spatial web objects where textual relevance is modeled as the weight of an object. In particular, Wu et al. [3] study the order-k weighted Voronoi cell for a single site (a set of k objects), and Xie et al. [4] investigate the imprecise version of order-1 weighted Voronoi cells.

There is a substantial difference between computing a single weighted Voronoi cell and computing the result region

(called a V-region) of a RTkKL query. Specifically, a V-region is more complex, as it is the union of a set of order-k weighted Voronoi cells. Even worse, it is difficult to compute the site of a high-order Voronoi cell. While we know that its site contains query object q and is of size k, for a given order, there could be many sites containing q for deriving the V-region. According to our experiments, when $k = 100$, there are more than $5,000$ such sites on average for a V-region, meaning that the methods based on a single site [3] [4] have no straightforward application here. Furthermore, even if we can successfully retrieve those cells, they are delineated by complex curves, making it difficult to merge them into a desired V-region.

In this paper, we define the RTkKL query for spatial web objects and offer a thorough coverage of the concepts, properties, and algorithms related to the computation of the V-region returned by the query. To the best of our knowledge, this is the first study of this problem.

Our contributions are summarized as follows:

- We offer detailed theoretical analysis of the features, properties, and performance of computing the RTkKL query.
- We propose an algorithm capable of computing approximate V-regions with quality guarantees.
- To further accelerate V-region computation, we use a quad-tree for indexing the solution space and an IR-tree for indexing the objects, and we show how to use the two indexes in combination to enable pruning.
- We report on extensive experiments to offer insight into the efficiency and scalability of our proposals.

The remainder of the paper is organized as follows. Section II defines the problem and presents relevant concepts and properties related to V-regions. Section III covers the basic method for computing V-regions. Section IV utilizes the indexing techniques to improve efficiency. Section V reports on the empirical study. Section VI covers related works, and Section VII concludes the paper.

II. Problem Setting and Preliminaries

We consider a set $\mathbb{O}$ of spatial web objects in two-dimensional Euclidean space Ω. We first define the top-k spatial keyword query. Based on that, we formally define reverse top-k keyword-based location query.

A. Problem Definition

Top-k spatial keyword query. Given a point $p \in \Omega$, a set ψ of keywords, and a number k, the top-k spatial keyword query $S_k(p, \psi)$ returns a set S_k of k spatial web objects with lowest ranking scores:

$$\forall o \in S_k, \forall o' \in \mathbb{O} - S_k, score(p, \psi, o) < score(p, \psi, o')$$

Function $score()$ [2], in Equation 1, combines spatial proximity and textual relevance. When ψ is clear from the context, we use $|p, o|$ to represent $score(p, \psi, o)$.

$$score(p, \psi, o) = w_s \cdot |p, o|_E + w_t \cdot (1 - tr(\psi, o.\psi)) \quad (1)$$

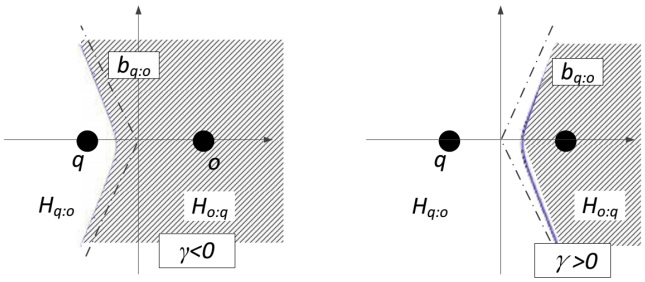


Figure 1: Bisectors and Half-spaces ($|\gamma| < |q, o|_E$)

Here, parameters w_s and w_t enable controlling the importance of two terms. Function $|p, o|_E$ denotes the Euclidean distance between p and $o.loc$, and function $tr()$, e.g., cosine similarity, computes the textual relevance between its two arguments. The smaller the score of an object is, the more relevant the object is to the query.

Definition 1: **Reverse top-k keyword-based location query (RTkKL).** Given a set ψ of query keywords, a query object $q \in \mathbb{O}$, and a number k, the RTkKL returns the maximum spatial region $\mathcal{V}_q$ such that q is contained in the result of any top-k spatial keyword query with ψ as the keywords and any location in $\mathcal{V}_q$ as arguments. Formally,

$$\mathcal{V}_q = \{p \in \Omega \big| q \in S_k(p, \psi) \wedge q \in \mathbb{O}\} \quad (2)$$

$S_k(p, \psi)$ is the result of a top-k spatial keyword search with p and ψ as arguments.

B. Problem Analysis and Geometric Preliminaries

Bisector. Given the query object $q \in \mathbb{O}$ and another object $o \in \mathbb{O}$, the bisector of q with respect to o is defined as:

$$b_{q:o} = \{p \in \Omega \big| |p, q| = |p, o| \text{ and } q, o \in \mathbb{O}\} \quad (3)$$

Substituting the score function (Equation 1) into the equation, we obtain the equation below.

$$\begin{aligned} |p, q| &= |p, o| \\ w_s \cdot |p, q|_E + w_t \cdot (1 - tr(\psi, q.\psi)) &= \\ w_s \cdot |p, o|_E + w_t \cdot (1 - tr(\psi, o.\psi)) & \\ |p, q|_E - |p, o|_E = \frac{w_t}{w_s} \cdot (tr(\psi, q.\psi) - tr(\psi, o.\psi)) & \quad (4) \end{aligned}$$

For a given set of query keywords, the textual relevance to an object is fixed. As exemplified in Figure 1, Equation 4 shows that the bisector has the shape of a hyperbola branch, where p and q's locations are the foci. Let γ denote the right-hand part of Equation 4.

The bisector is the locus of the points where the difference of the distances to the two foci equals γ. The bisector partitions the domain space into two halves. We call the half closer to q the half space from q to o and denote it by $H_{q:o}$. If a point p belongs to $H_{q:o}$, object q is more relevant p than o. According to Equation 4, when $\gamma < 0$, half space $H_{q:o}$ is convex, i.e., as illustrated by the dashed region in Fig. 1(a); when $\gamma > 0$, the half space is concave, as illustrated in Fig. 1(b); when $\gamma = 0$, the bisector degenerates into the perpendicular bisector between q and o.

The RTkKL query returns a region. Considering the region for a top-1 query, this is the additively weighted Voronoi cell of q. It can be formed by the intersection of the half spaces of q w.r.t. every object $o \in \mathbb{O}$. Formally,

$$V(q) = \cap_{q \neq o, o \in \mathbb{O}} H_{q:o}$$

Next, we extend the region to the top-k case. Given a set $S_k = \{s_1, s_2, ..., s_k\}$ of k objects, the region that takes S_k as the top-k result is an additively weighted order-k Voronoi cell with the generating site S_k. Formally,

$$\begin{aligned} V^{(k)}(S_k) &= \cap_{o \notin S_k} H_{s_1:o} \bigcap \cap_{o \notin S_k} H_{s_2:o} \bigcap ... \cap_{o \notin S_k} H_{s_k:o} \\ &= \cap_{q' \in S_k, o \notin S_k} H_{q':o} \end{aligned} \quad (5)$$

If q is an element of set S_k, object q is in the top-k results for any point in the region, $p \in V^{(k)}(S_k)$. Note that such S_k may not be unique for a given k. This means that one must consider multiple order-k Voronoi cells for a given k.

As described above, we can represent the region returned by the RTkKL query as a *V-region*, which is essentially the union of Voronoi cells from order 1 to k with sites containing q. For example, the order-1 Voronoi cell of q returns the points having q as the top-1 relevant object. Similarly, an order-2 Voronoi cell $V_q^{(2)}$ returns the points having q as one of its top-2 relevant objects, and so on. Therefore, the region returned by the RTkKL query should cover the region of the top-1, top-2, ..., top-k objects containing q, and it is thus the union of Voronoi cells of orders from 1 to k.

Definition 2: (**V-region**) Given an object $q \in \mathbb{O}$ and a number k, an order-k V-region of q is the union of the Voronoi cells from order 1 to order k whose sites contain q, denoted as $\mathcal{V}_q$ ($\mathcal{V}_q \subseteq \Omega$). Formally,

$$\mathcal{V}_q = \bigcup_{i=1,...,k} \cup_{q \in S_i, S_i \subseteq \mathbb{O}} V_q^{(i)}(S_i) \quad (6)$$

According to Okabe et al. [16], the union of an object's order-$(i+1)$ Voronoi cells covers that of the order-i Voronoi cells. So, Equation 6 can be rewritten as:

$$\mathcal{V}_q = \bigcup_{q \in S_k, S_k \subseteq \mathbb{O}} V_q^{(k)}(S_k) \quad (7)$$

C. Complexity

An intuitive way of building a V-region is to construct Voronoi cells incrementally, from the 1-st to the k-th order, and then take their union. However, this is computationally challenging.

First, the number of Voronoi cells composing a V-region can be very large. For high-order Voronoi cells, there can be more than one Voronoi cells whose sites contain q. Given an order, say k, the number of sites, or k-subsets, is combinatorial, $O(n^k)$. Specifically, the average number of k-subsets is $O(k(n-k))$ [16]. One may generate a large number of unnecessary k-subsets before getting those really contributing to Voronoi cells and then V-regions.

Second, the contour of a V-region consists of curves and concave shapes that are difficult to store and query in real applications.

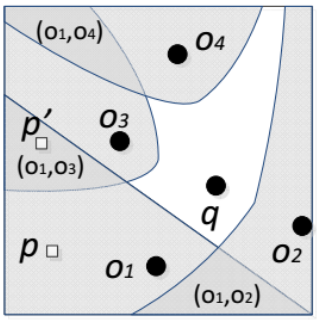


Figure 2: An example of V-region ($k = 1$)

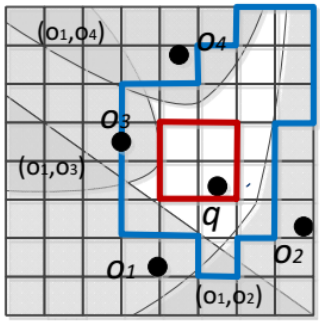


Figure 3: Grid Aproximation with Quality Guarantees

To further enable efficient query processing, it would be desirable to develop a solution that returns a well-approximated and error-bounded region composed by simple shapes, e.g., orthogonal rectangles. In the sequel, we present efficient algorithms that achieve such a solution.

Table I: A Summary of Notations

Notation	Meaning
Ω, $p \in \Omega$	the spatial domain space, and a point inside
$\mathbb{O}$	a set of spatial web objects $\{o_1, \ldots, o_n\}$
o	an spatial web object $\{o.loc, o.\psi\}$
q	the query web object $\{Q.loc, Q.\psi\}$
S, S_k	set, k-subset
$\|a,b\|_E$	Euclidean distance between a and b
$\|a,b\|_{minE}$, $\|a,b\|_{maxE}$	lower and upper bound for $\|a,b\|_E$
$\|a,b\|$	weighted distance between a and b
$\|a,b\|_{min}$, $\|a,b\|_{max}$	lower and upper bound for $\|a,b\|$
$\|q,o\|_{bd}$	min center-border distance between q and o
$b_{q:o}$	bisector of q with respect to o
$H_{q:o}$	half space from q to o
$\prec$, $\prec^{(k)}$	domination and k-domination
$V_q^{(k)}$	k-th order Voronoi cell of q
$\mathcal{V}_q$	V-region of q, $\mathcal{V}_q = \cup_{i=1...k} V^i(q)$

III. Grid Approximation: From Solution Space's Perspective

From this section on, we investigate how a V-region can be efficiently constructed. We start with order-1 case, then extend it to more general cases. An example ($k = 1$) of V-region $\mathcal{V}_q$ is shown in Fig. 2. We consider half spaces of q with respect to all other objects, i.e., o_1, o_2, o_3, and o_4. Their intersection is represented by the white area. For order-1 case, such an intersection is the V-region, $\mathcal{V}_q$, to be rendered.

It is costly to use the curve-shaped contours for implementing $\mathcal{V}_q$, as analyzed previously. To tackle that, we study how $\mathcal{V}_q$ can be approximated by space division techniques, particularly, by orthogonal grids.

We illustrate an approximated $\mathcal{V}_q$ in Fig. 3. The spatial domain Ω is discretized into a set of grids uniformly. The grids can be of three cases.

- *Rejected* grids are those outside the blue contour and are not part of $\mathcal{V}_q$;
- *Accepted* grids are those within the red contour and are part of $\mathcal{V}_q$;

- *Undetermined* grids are located between the red and blue contours and partially belong to $\mathcal{V}_q$.

Then, a V-region can be the union of all accepted and undetermined grids[1]. The precision of the approximation can be tuned by the granularity of grids. With sufficiently small sized grids, the red and blue contours of Fig. 3 will approach to each other closely. The two contours would converge using infinitely small sized grids.

Two issues remain to be studied. First, how to determine the relationship between a grid $g \in \Omega$ and V-region $\mathcal{V}_q$, i.e., whether g is accepted, rejected, or undetermined? Second, grids of small granularities yield high precisions but also high construction overheads. How to design an adaptive method to achieve a good efficiency yet without compromising a guaranteed quality?

The first problem is solved by Section III-A. The second problem is addressed by Section III-B.

A. Dominance Relationship: from 1 to k

1) Starting with order-1 case:

Definition 3: **Dominance ($\prec$).** Given a grid region g and two objects q and o, if g is totally covered by $H_{o:q}$, we say that o dominates q w.r.t. g, meaning that o is more relevant than q for every point of g. Formally,

$$g \subseteq H_{o:q} \Rightarrow o \prec_g q \tag{8}$$

Perfectly but unrealistically, to determine whether grid g belongs to half space $H_{o:q}$ is to check whether every point of g belongs to $H_{o:q}$. Next, we introduce a set of practical alternatives by utilizing the convex or concave geometries of the half space[2].

Convex Half Space. If grid region g and half space $H_{o:q}$ are both convex, we can determine that g is contained by $H_{o:q}$, if each of g's four vertices is contained by $H_{o:q}$. Then, we reduce the problem of determining the relationship between two regions into a problem of determining the relationship between a point and a region. The correctness is guaranteed.

Concave Half Space. However, it is not easy to determine whether a region is contained by a concave half space.

Method 1: A loose (or sufficient but not necessary) condition is to use min/max distance comparison that is formalized as Lemma 1.

Lemma 1: For a region g, we have:

$$|q,g|_{min} > |o,g|_{max} \Rightarrow g \subseteq H_{o:q} \Leftrightarrow o \prec_g q$$

Proof:

$$\begin{aligned} |q,g|_{min} > |o,g|_{max} &\Rightarrow \forall p \in g, |q,p| > |o,p| \\ &\Rightarrow \forall p \in g, p \in H_{o:q} \Rightarrow g \subseteq H_{o:q} \end{aligned}$$

[1]A conservative version of a V-region might be the union of all accepted grids. Which version to be used is application dependent, e.g., whether false positive results are allowed. Note that our algorithms presented in this work can support both versions.

[2]Grid g is of square shapes and thus is convex. The determination of the dominance relationship between g and $H_{q:o}$ is the same as between g and $H_{o:q}$ and is omitted.

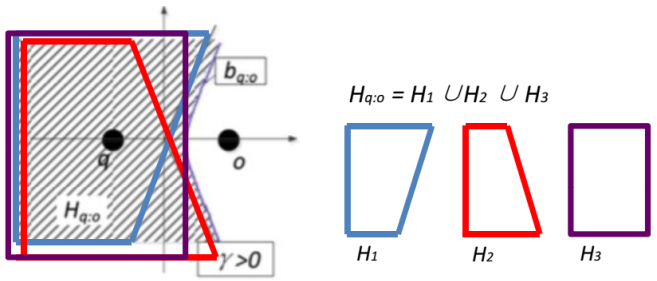


Figure 4: Concave Half Space Decomposition

■

Method 2: A tighter condition is to approximate the half space with three convex alternative polygons, as shown in Fig. 4. The three polygons are composed by cutting the domain space Ω with the two asymptotic lines, i.e., H_1 and H_2, and with the line perpendicular to line segment $\overline{qo}$ and internally tangent to $H_{o:q}$, i.e., H_3. The three polygons satisfy: 1) $H_1, H_2, H_3 \subseteq H_{o:q}$; 2) $H_1 \cup H_2 \cup H_3 \subseteq H_{o:q}$. The union of H_1, H_2, and H_3 is a conservative approximation of half space $H_{o:q}$. If region g is within any of the three polygons, g is within half space $H_{o:q}$.

We do not have to explicitly materialize the three polygons. Instead, we check if a grid is on the "o" side of the three bisector lines, i.e., two asymptotic lines and one perpendicular line. In this way, Method 2 takes three distance comparisons and Method 1 takes one distance comparison.

Note that there is a gap between the approximated polygons in Method 2 and the concave half space. If a grid falls in such a gap, the grid will be reported as undetermined. Moreover, further splitting the grid does not help in determining their overlapping relationships, as a descendant grid also falls in the gap. Method 1 has no such limitations. When the grid is sufficiently small, the dominance relationship can be simply determined by Method 1. The process of region splitting is detailed in Section III-B.

In summary, Method 1 is more general and efficient, whereas Method 2 is more accurate and thus more effective. In our implementation, we first use Method 1 and then Method 2 for examining the dominance relationship in order to reduce construction overheads. We stop using Method 2 if the size of a grid is smaller than the maximum distance between the half space and its asymptotic lines[3].

So far, we have investigated how an order-1 V-region (or a Voronoi cell equivalently) can be approximated by examining the dominance relationships between grids and half spaces. Next, we study how to extend the above to the order-k case.

2) Extension to order-k case:

Definition 4: **k-dominance Relationship ($\prec^{(k)}$).** Given region g, object q, and a set S of k other objects, $S = \{o_1, ..., o_k\}$, if g is totally covered by half spaces

[3]Although a better approximated half space can be achieved by a larger number of polygons, it costs more than applying Method 1 for the decomposed grid, which contradicts our intention. For simplicity, we adopt the heuristic described above.

$H_{o_1:q}, ..., H_{o_k:q}$, we say that q is dominated by S w.r.t. g, meaning that there exist k other objects that are more relevant than q for any point in region g. In other words, g is not a part of $V_q^{(k)}$ and thus not a part of $\mathcal{V}_q$. Formally,

$$\forall i_{1\leq i\leq k}, g \subseteq H_{o_i,q} \Rightarrow S \prec_g^{(k)} q \Leftrightarrow g \not\subseteq V_q^{(k)}(S) \Leftrightarrow g \not\subseteq \mathcal{V}_q \quad (9)$$

We first extend Lemma 1 to the order-k case and obtain Lemma 2.

Lemma 2: Consider a region g and a set S of objects. Let the objects in S be sorted according to their maximum distances to g in the ascending order and thus $|g, o_k|_{max}$ is the k-th maximum distance. We have:

$$|q, g|_{min} > |o_k, g|_{max} \Rightarrow S \prec_g^{(k)} q \quad (10)$$

See the example shown in Figure 2, where p is a point in the plane. If it is within half-space $H_{o_1:q}$, it is dominated by o_1 meaning that q cannot be p's first nearest neighbor. For another point p', if it is within half-spaces $H_{o_1:q}$ and $H_{o_3:q}$. Thus, object q cannot be p''s first two nearest neighbors.

In this way, we can render a V-region by excluding those regions that are dominated by k objects other than q. In Figure 2, the white region is the V-region of q when $k = 1$ and the union of the white region and light gray regions is the V-region of q when $k = 2$ (the area excluding those regions marked with (o_1, o_2), (o_1, o_3), and (o_1, o_4)).

The idea of counting for each region the number of objects dominating q or equivalently the number of half-spaces covered also appears in [17], but in a different problem setting. If the count of a region is greater than k, it means that there exist more than k objects that are more relevant than q. So, the region cannot be part of $V_q^{(k)}$. The correctness is guaranteed by Lemma 3.

Lemma 3: (**Dominance Monotonicity 1.**) If g is $k+1$-dominated, g is also k-dominated.

Lemma 4: (**Dominance Monotonicity 2.**) If g is dominated by a set S of objects w.r.t. q and g' is a subregion of g, then g' must be dominated by S w.r.t. q.

We have discussed how the dominance relationship between grid regions and objects can be determined. The process is summarized by Algorithm 1 in Section III-A3.

3) Basic Module: Unqualified grids will be rejected if they are dominated by k (or more) objects other than q.

For an accepted grid, there might exist some dominating objects, but such objects must be no more than $k-1$. So, for each grid, a counter, $count.min$, is set to record how many objects currently dominate the grid so far. Another counter, $count.max$, is set to record the number of dominating objects plus the number of undetermined objects. Therefore, the sum indicates the largest possible number of objects that dominate q w.r.t. the grid. If this number is smaller than k, we can quickly tell that the grid is accepted.

The undetermined grids are further explored and the corresponding V-region will be gradually expanded and approximated based on Lemma 4. The precision of the approximation depends on the grids' granularity. The indeterminacy can be alleviated if presented with a finer level of grids. In the sequel, we show how this is achieved in an adaptive manner.

Algorithm 1: GridCheck

Data: query object q, a grid g, a set of objects g.list, k
Each grid g has a count which is an interval representing the minimum and maximum number of objects dominating g;
Use Methods 1 and 2 to detect if $o \prec_g^{(k)} q$ is true;
if $o \prec_g^{(k)} q$ *is detected to be true or* $g.count.min \geq k$ **then**
    return "$rejected$";
if $o \prec_g^{(k)} q$ *is detected to be false or* $g.count.max < k$ **then**
    return "$accepted$";
if $o \prec_g^{(k)} q$ *cannot be determined* **then**
    Add o to $g.list$;
    Update $g.count.min$, and $g.count.max$;
    return "$undetermined$";

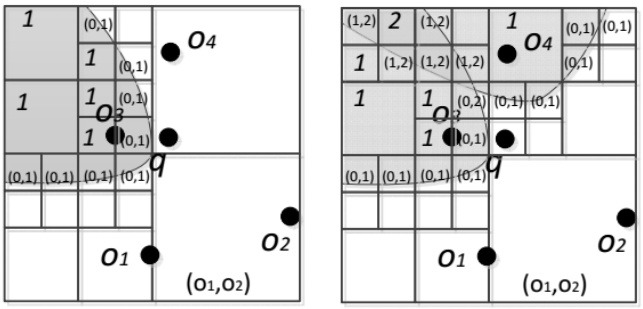
B. An Adaptive Approach

The idea is to use a quad-tree to approximate a V-region by initially setting the root grid as the entire spatial domain and the object list as the entire database.

The procedure is described in Algorithm 2. The root is initially split into 4 child grids and the splitting process is repeated recursively. At each iteration, we check if a grid can be rejected or accepted for partially being $\mathcal{V}_q$ and if an object can be rejected or accepted for the dominance determination with the grid. The splitting of the branch stops if the a grid is accepted or rejected. Otherwise, for a grid, all elements of its object list are examined. Once an object is detected to deterministically dominate a grid, e.g., the top and leftmost quadrant in Fig. 5(a) is contained by $H_{o_3:q}$, the object is removed from the object list and the counter is marked by "1" indicating that currently the grid is dominated by one object, i.e., o_3. If the dominance cannot be determined, we keep the object in the list and split the grid into four child grids. In this case, the counter is an interval with a minimum and a maximum number of objects dominating q w.r.t. the grid. For example in Fig. 5, when $k = 1$, grids with minimum counter no less than 1 can be rejected. Because there is already one object dominating q w.r.t. these grids, meaning that they have no chance for being a part of $\mathcal{V}_q$. The process continues with enumerating the remaining objects of the list until the finest granularity specified is reached.

The breadth-first process is implemented by a queue. The elements, i.e., grids, of the queue are stored in the descending order of grid areas. In this way, the algorithm tends to explore a coarser level of grids. In case of equal sized grids, we break ties by their minimum count of dominating objects. This is useful, because, intuitively, a grid with higher minimum count tends to be rejected at an earlier stage.

There remain two issues to clarify: 1) the stopping condition of the quad-tree splitting at each iteration; 2) the order of



(a) (b)

Figure 5: Quad-tree Approximation

Algorithm 2: GridDecomposition

Data: query object q, an object database D
A grid list $GridList = \emptyset$;
Initialize a quad-tree whose root node covers the domain Ω;
Start with the root level of quad-tree and add its 4 child grids into $GridList$;
while $GridList$ *is not empty* **do**
  Pop the grid g with the largest area; ▷ breaking tie by $g.count.min$;
  Invoke $GridCheck$ (Algorithm 1) for grid g;
  if *the result is "undetermined"* **then**
    Split g in to 4 child grids $\{g_i\}_{i\leq 4}$;
    for *each* g_i **do**
      Inherit $g_i.list$, $g_i.count.min$, and $g_i.count.max$ from g;
    Add $\{g_i\}$ into $GridList$;
  else
    Evaluate the current precision lower bound, i.e., ϵ, and break if the accuracy requirement is satisfied.

objects being examined. Issue 1) is detailed below. Issue 2) is addressed in Section IV.

The stopping condition of Algorithm 2 can be set for different purposes. For example, the condition can be the maximum splitting depth, or equivalently the minimum size of a grid. We can also control the quality of the approximation by considering the portion of the undetermined grids over all non-rejected cells. In Fig. 3, the grids within the red contour definitely belong to $\mathcal{V}_q$. The grids between the blue and the red contours are undetermined ones. If the portion of undetermined cells is low, it implies the similarity between the exact and approximated V-region is high. Thus, the precision can be lower bounded by ϵ, defined in Equation 11.

$$\epsilon = \frac{\sum_{g\in L_{acc}} area(g)}{\sum_{g\in L_{ud}} area(g) + \sum_{g\in L_{acc}} area(g)} \tag{11}$$

In the equation, L_{ud} represents the list of grids that are undetermined, and L_{acc} represents the list of grids that are accepted as part of $\mathcal{V}_q$. The value range of ϵ is between 0 and 1. A higher valued precision implies a finer approximated $\mathcal{V}_q$. Then, a quad-tree splitting can stop if the lower bound of the precision, i.e., ϵ, exceeds a user specified precision threshold, e.g., 0.95. The precision approaches 1 as more and more undetermined grids are explored.

C. Optimization

1) Reusing Vertices Checking: Considering a parent quad-tree node g_P, for determining the relationship with n_P half spaces, it requires at least $4\times n_P$ comparisons in Algorithm 2. After decomposition, four child nodes are generated that further require $4\times 4\times n_P$ distance comparisons. Comparing the four child nodes with g_P, only five vertices are new, meaning only $5 \times n_P$ comparisons are necessary if the computation of vertices checking of g_P level can be inherited.

2) Quad-tree Root Setting: The recursive grid decomposition is effective yet incurs considerable overheads, especially in the first few levels. Because a higher-level grid has a bigger size and thus has a lower chance to get rejected. A better efficiency can be achieved if the decomposition starts from a lower level. In an ideal case, the quad-tree root is set to the minimum grid covering $\mathcal{V}_q$. The procedure is summarized in Algorithm 3, followed by the details of estimating the area of a V-region.

Algorithm 3: Flooding Algorithm

Data:
Calculate the expected area of a V-region, α;
Start with level $l = \lceil log_4(\frac{|\Omega|}{\alpha})\rceil$;
Let g_q be the grid of level l that contains q;
Gradually detect g_q's neighboring grids from the inside out until all grids on the outer layer are k-dominated;
Revoke Algorithm 2 to evaluate $\mathcal{V}_q$;

We develop a model for estimating the size of a V-region, following the assumption that objects are uniformly distributed in the domain Ω. Let

$$\begin{cases} w_q = \frac{w_t}{w_s} \cdot tr(\psi, q.\psi) \\ w_o = \frac{w_t}{w_s} \cdot tr(\psi, o.\psi) \\ \widehat{w_o} = \frac{1}{k}\sum_{o\in S_k} w_o \approx \frac{1}{n}\sum_{o\in D} w_o, \end{cases}$$

where $\widehat{w_o}$ is the average textual relevance of all objects to ψ. With respect to an arbitrary point $p \in \Omega$, an object o with distance $|p,o|_E < |p,q|_E + w_q - \widehat{w_o}$ can be regarded as a dominator of q. If q is a top-k result, there must be less than k objects satisfying the above distance inequality. In other words, there are less than k objects located inside the circular region centered at p with radius $|p,q|_E + w_q - \widehat{w_o}$. Since objects are uniformly distributed, we can get a bound for $|p,q|_E$, i.e., $\frac{\pi\cdot|p,q|^2}{|\Omega|} < \frac{k}{n}$. After transformations, we get:

$$|p,q|_E < \sqrt{\frac{k\cdot|\Omega|}{\pi n}} + w_q - \widehat{w_o} \tag{12}$$

V_q is composed of all points $p \in \Omega$ such that Equation 12 holds. Then, we can obtain:

$$E(Area(\mathcal{V}_q)) = \pi \cdot |p,q|_E^2 = \pi \cdot [\sqrt{\frac{k\cdot|\Omega|}{\pi n}} + w_q - \widehat{w_o}]^2 \tag{13}$$

D. Discussion

The grid approximation approach renders a V-region from the solution space's perspective and bypasses the difficulties incurred by bisector curves. Also, the approximation approach rejects unqualified grids, which enhances the algorithm's adaptivity; it works regardless of whether or not the V-region is continuous.

In the sequel, we study how the properties of object space can be utilized to accelerate the construction process.

IV. Searching Enhancement with an Index

We assume that an IR-tree [2] is built on the set of objects. The textual relevance of a node can be calculated at runtime and is guaranteed to be higher than that of any descendant entries. The correctness follows the monotonicity property [2].

We observe that the accessing sequence of objects has non-negligible impact over the V-region construction performance. For example, $\mathcal{V}_q$ tends to be contributed by the objects whose half spaces w.r.t. q are close to q. On the other hand, a half space rendered by an object close to q tends to cover more coarser level grids. In such cases, it is preferable to first handle the objects with bigger impact on building $\mathcal{V}_q$. Moreover, an index structure enables the dominance relationship checking against a cluster of objects (vs. a single object). Thus, disqualifying a "rejected" grid can be executed in an earlier phase by pruning irrelevant index entries.

We present an enhanced Gridcheck method over Algorithm 1 in Section IV-A. We cover pruning techniques in Section IV-B.

A. Enhanced GridCheck

Sketch. The enhanced grid checking procedure is shown in Algorithm 4. We utilize a queue to browse the objects in a best-first search fashion [18] and access the index entries according to how close their half spaces to object q. Initially, the index root is inserted into the queue. We expand the index entries recursively and apply pruning techniques (detailed in Sections IV-B) to filter unqualified ones. An unvisited entry can be filtered if each object in the subtree is proved not to refine the current V-region.

The enhanced module of GridCheck can then be replanted into the GridDecomposition framework (Algorithm 2). A grid can be rejected if it is k-dominated. Otherwise, we follow the splitting strategy developed in Section III until the precision threshold is met.

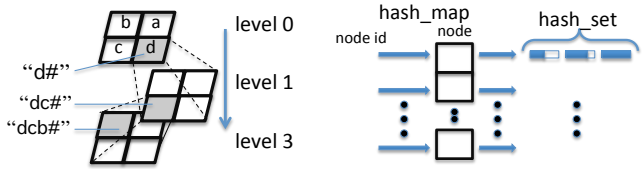
At each iteration of examining a grid g, the index is traversed. An index entry e can be pruned or a grid g can be rejected if the pruning rules (in Section IV-B) are triggered. If e is dominated by q w.r.t. g, for g's sub-grids, e is also dominated by q, according to the dominance monotonicity (Lemma 4). With unqualified entries being pruned, only a partial index (without pruned branches) is passed to g's sub-grids, to avoid unnecessary node accessing.

However, it would be costly to explicitly deliver such partial indexes between two subsequent levels of grids, because of the extra storage and the computation overheads that are non-shared between a grid and its slibings. Alternatively, we use a global data structure called *pruned set* to record index entries that have been pruned. By doing so, only one index copy is maintained. Each time visiting an index entry, we look up the pruned set to see if the current grid or its ancestors have a determined dominance relationship with it. With the pruned set, the redundant index entry accessing can be saved.

Algorithm 4: Enhanced GridCheck

Data: query object q, an IR-tree, a pruned set, a grid g ($list$,$count.min$, $count.max$), parameter k

```
1  Add index root to Queue_cnd, an initially empty queue;
2  while Queue_cnd is not empty do
3      Pop up an entry e from Queue_cnd;
4      if e has been pruned, i.e., in the pruned set then
5          continue;
6      else
7          Test the dominance relationship between e and q;
8          if e ≺_g q is true then
9              Add e to the pruned set; ▷ e dominates q Update
               g.count.min;
10         if q ≺_g e is true then
11             Add e to the pruned set;      ▷ q dominates e;
12             Update q.count.max by subtracting |e|;
13                    ▷ |e| indicates the number of objects in e's
                   subtree
14         if e is an object then
15             Add e to g.list;          ▷ e is undetermined;
16             Update g.count.min and g.count.max;
17         else
18             Add e's children into Queue_cnd;
19         Break, if q is k-dominated;
```



(a) Quadrant String (b) Pruned Set Hashing

Figure 6: Pruned Set Compression

For illustration, we use Fig. 6 to show the usage of the pruned set. First, we use *quadrant string* to identify a grid by a character string. The four quadrants (or child grids) of a grid are marked by letters a-d. For example in Fig. 6(a), the grid labelled "d#" is the fourth grid of the root node (level 0). The grid labelled "dc#" is the third quadrant of grid "d#" and so on[4]. When checking if a grid is a descendant of another grid, we can simply check if the prefix of the grid's string matches that of another grid. Second, for pruned index entries, we record them by a hash map, as shown in Fig. 6(b). For a pruned index entry, it is possibly associated with multiple quad-tree branches. We index these branches represented by corresponding quadrant strings with a hash set and link it to the pruned index entry. While inserting an element into the

[4]Using bit string would be more efficient for string matching. We choose character string because it is retains scalability for lower levels of the quad-tree. The whole process is compatible with bit strings by simple modifications.

hash set, they are transformed into equal-length hash codes and thus enable efficient string prefix matching for determining whether a grid is a descendant of another. With the two-level hashing structure, the searching of a pruned entry can be done in constant time.

B. Pruning Techniques

In this section, we investigate pruning techniques that help in achieving better efficiency. For a grid, we expect to retrieve only a small subset of objects which render an overall picture about its dominance relationship with all objects in the database. It is realized by: 1) accessing qualified objects first (Section IV-B1); 2) rejecting unqualified objects early (Section IV-B2). The purpose is to reduce the iterations of both grid and object accessing and to yield a fine-gained V-region at a rapid pace.

1) Minimum Border Distances: We define *minimum border distance*, which is the minimum Euclidean distance between query object q and half space $H_{o:q}$. Intuitively, half spaces closer to q would contribute more to $\mathcal{V}_q$. Hence, their corresponding objects are preferable to be accessed first.

Lemma 5: **Minimum Border Distance ($|q, o|_{bd}$).** The minimum distance between q to a half-space $H_{q:o}$ is $\frac{|q,o|_E+\gamma}{2}$.

Take Fig. 1 as an example, the intersection between $b_{o:q}$ and x-axis yields the minimum distance between q and $H_{o:q}$. The minimum border distance equation can be derived with simple transformation from Equation 4 and is omitted due to space limits. Note that both cases of $\gamma > 0$ and $\gamma < 0$ have been incorporated in the equation.

The minimum border distance between an index entry e and q can be defined by considering the entry as a virtual object. The virtual object's coordination is the closest point on e to q with the relevance as the highest one between e's descendant objects to keyword set ψ.

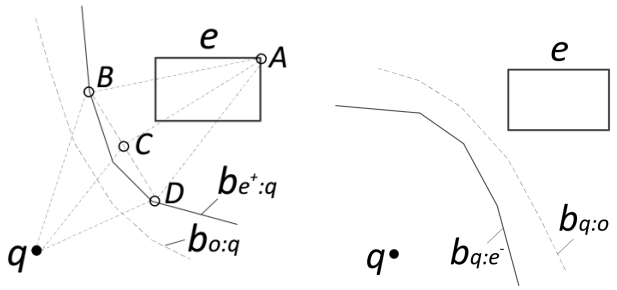
Lemma 6: For query object q, its minimum border distance with respect to an index entry is a conservative estimation of the minimum border distance to any objects in e's subtree.

$$\forall o \in e, |q, e|_{bd} \leq |q, o|_{bd} \tag{14}$$

Proof: The way we define the virtual object for e underestimates the Euclidean distance between q and e and overestimates the relevance between $e.\psi$ and ψ. Therefore, the weighted distance from q to $H_{e:q}$ is underestimated, meaning that border distance $|q, e|_{bd}$ is smaller than $|q, o|_{bd}(o \in e)$. ■

The minimum border distances are of two purposes: 1) for ordering the elements in the queue in an ascending order according to their border distances. It is proved by Lemma 6 and used in Algorithm 4; 2) for enabling a quick stopping condition for accessing index entries.

Given a partially constructed V-region, $\mathcal{V'}_q$, if the maximum distance between q and $\mathcal{V'}_q$ is smaller than the border distance $|q, e|_{bd}$, then it is also smaller than the border distance from q to any object in e. Then, it is not possible for half spaces $H_{o:q}(o \in e)$ to overlap with $\mathcal{V'}_q$ and hence the entry can be rejected. Moreover, the queue can be emptied because the rest entries have longer maximum distances.



(a) e^+ (b) e^-

Figure 7: Dominance Relationship with an Index Entry

2) Dominance Relationship with an Index Entry: Now we investigate how to determine the dominance relationship between an object and an index entry, which enables pruning on an index level.

We summarize the pruning techniques into three categories in Table II, according to the value of the textual relevance of a given entry.

Table II: Category

Category	Condition	Determination
e^+	$tr(\psi, q.\psi) > min_{o \in e} tr(\psi, o.\psi)$	$e \prec_g q$, tighter bound
e^-	$tr(\psi, q.\psi) < max_{o \in e} tr(\psi, o.\psi)$	$q \prec_g e$, tighter bound
Other	general conditions	$e \prec_g q$ or $q \prec_g e$

"e^+"case. We define distance $|p, e^+| = \frac{|p,e|_{maxE}}{min_{o \in e} tr(\psi, o.\psi)}$ which overestimates the distance between point $p \in \Omega$ to entry e. Then, we can draw a bisector $b_{e^+:q} = \{p \in \Omega \big| |p, q| = |p, e^+|\}$. The e part of the bisector is denoted as $H_{e^+:q}$, such that for any point inside, e is more relevant than q.

Lemma 7: $\forall o \in e$, $H_{e^+:q} \subseteq H_{o:q}$.

Proof: It is equivalent to prove that $p \in H_{e^+q}$ implies $p \in H_{o:q}(\forall o \in e)$, where $p \in \Omega$ is a point.

$$\left.\begin{array}{r} p \in H_{e^+:q} \Leftrightarrow |p, q| > |p, e^+| \\ \left.\begin{array}{l} |p, e^+|_{maxE} \geq |p, o|_{E(o \in e)} \\ tr(e^+.\psi, \psi) \leq tr(o.\psi, \psi)_{(o \in e)} \end{array}\right\} \Rightarrow |p, e^+| \geq |p, o|_{(o \in e)} \end{array}\right\}$$
$$\Rightarrow |p, q| > |p, o|(o \in e) \Rightarrow p \in H_{o:q}(o \in e)$$

■

Lemma 7 can be used for k-dominance checking. If a grid is in $H_{e^+:q}$, it must also be in $H_{o:q}(\forall o \in e)$, meaning that $o \prec_g e(\forall o \in e)$ if $e \prec_g q$. Then, if e is an entry with no less than k objects and it is detected that $e \prec_g q$, then grid g can be rejected, since g is dominated by no less than k objects.

Lemma 8: $H_{e^+:q}$ is convex.

Proof: We first choose two points B and C on the bisector $b_{e^+:q}$ as shown in Fig. 7(a). Let D be the middle point of line segment $\overline{BC}$. If we can prove D is in $H_{e^+:q}$, we can prove $H_{e^+:q}$ is convex. Let A be the point of e yielding the maximum

distance from D to e. We have:

$$|B, A|_E \leq |B, e|_{maxE} \text{ and } |C, A|_E \leq |C, e|_{maxE}$$
$$\Rightarrow |B, A|_E \leq |B, q|_E \text{ and } |C, A|_E \leq |C, q|_E \quad (B, C \in b_{e^+:q})$$
$$\Rightarrow |D, A|_E \leq |D, q|_E \text{ (Lemma 10 in [19])} \Rightarrow D \in H_{e^+:q}$$

Hence, the lemma is proved. ■

A grid is convex. If $H_{e^+:q}$ is also convex, the determination of $e \prec_g q$ can be reduced to checking if every vertex of g is inside $H_{e^+:q}$.

"e^-" case. We define distance $|p, e^-| = \frac{|p,e|_{minE}}{max_{o \in e} tr(\psi, o.\psi)}$ which underestimates the distance between point $p \in \Omega$ to entry e. A bisector $b_{q:e^-} = \{p \in \Omega \big| |p, q| = |p, e^-|\}$ splits the domain space into two halves. The part closer to q is denoted as $H_{q:e^-}$, such that for any point inside, q is definitely more relevant than e. It is exemplified in Fig. 7(b).

Lemma 9: $\forall o \in e, H_{q:e^-} \subseteq H_{q:o}$.

Lemma 10: $H_{q:e^-}$ is convex.

With Lemma 9, we can prove that if $q \prec_g e$ then $q \prec_g o (\forall o \in e)$. Lemma 10 enables determining the dominance relationship by checking if the four vertices of a grid are within a convex half space $H_{q:e^-}$. They can be proved in a similar way as Lemmas 7 and 8 and are omitted due to space limits.

Other cases. In more general cases, we can determine the dominance relationship by comparing the minimum and maximum distances as detailed by Lemmas 11 and 12.

Lemma 11: $max_{p \in g}|p, q| < min_{p' \in g}|p', e| \Rightarrow q \prec_g o_{(o \in e)}$.

Proof:

$$max_{p \in g}|p, q| < min_{p' \in g}|p', e| \Rightarrow \forall p \in g, |p, q| < |p, e|$$
$$\Rightarrow \forall p \in g, p \in H_{q:o} \Rightarrow g \subseteq H_{q:o} \Rightarrow q \prec_g e \Leftrightarrow \forall o \in e, q \prec_g o$$

■

Lemma 12: $min_{p \in g}|p, q| > max_{p' \in g}|p', e| \Rightarrow o \prec_g q_{(o \in e)}$.

The lemma can be proved similarly to Lemma 11.

V. Results

We cover the experimental setup in Section V-A, report the performance results of our algorithms in Section V-B, and provide a detailed analysis in Section V-C.

A. Experiment Setup

Datasets. We use two real geo-location datasets, EURO and CAR. EURO is a dataset of POIs, such as hotels, ATMs, and stores in Europe[5]. CAR is a dataset which combines the spatial data in California[6] and a real collection of short abstracts from DBpedia[7]. Table III summarizes the statistics of each dataset. We formulate the query keywords by randomly selecting keywords from each dataset. For all queries examined, 8 to 14% (10% on average) of all objects have non-zero textual relevance. To evaluate the scalability of our proposed algorithms, we also generate 4 synthetic datasets with cardinalities of 2, 4, 6, and 8 millions, respectively. In synthetic datasets, the locations are randomly generated and the keywords are randomly extracted from the extended abstracts of DBpedia.

[5]http://www.allstay.com
[6]http://www.usgs.gov
[7]http://wiki.dbpedia.org

Table III: Dataset Statistics

	EURO	CAR
Total # of objects	162,033	2,249,727
Total unique words in dataset	35,315	688,087

Competitors. To comprehensively evaluate our proposals, we consider three competitors: *Basic*, *Quad-tree*, and *Dual-tree*. The Basic solution is given by Algorithm 2, GridDecompose, but without using any optimization techniques, meaning that only Method 1, i.e., Lemma 1, is used in dominance checking. The Quad-tree solution improves Basic by: 1) using Method 2 for dominance checking; 2) using the flooding algorithm, i.e., Algorithm 3, for the quad-tree root setting; 3) using vertex caching for determination acceleration. The Dual-tree based solution improves Quad-tree by further speeding up the processing by means of an IR-tree. Each reported value is the average of 50 runs. We also offer a detailed analysis of the effect of each specific optimization technique.

Parameters. We test the performance of our proposals by varying different parameters, including the expected maximal ranking of the query object (k), the preference weight in scoring function (ω_s and ω_t), the precision setting of an approximate solution (ϵ), and the textual relevance between query keywords and query object q's textual part ($tr(\psi, q.\psi)$), as summarized in Table IV.

Table IV: Parameter Settings

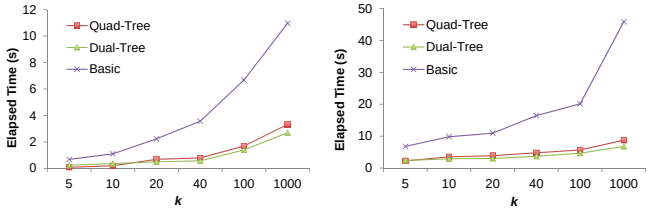
	Value Range	Default Setting
k	5 – 1,000	40
ω_s	0 – 1	0.5
ϵ	0.8 – 0.99	0.9
$tr(\psi, q.\psi)$	0 – 1	0.5

The experiments were executed on a laptop (Intel Core i5 2.5GHz CPU and 8GB RAM) running Mac OS X 10.8.5 operating system. The codes were written in Java (JDK 1.6).

B. Performance Results

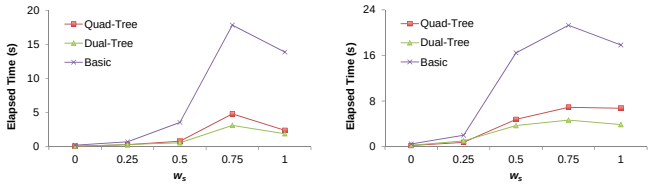
The query performance is evaluated in terms of both efficiency and scalability. The efficiency is measured by counting the clock time elapsed. In particular, we consider the effect of different parameter settings. The scalability is measured using datasets of different sizes.

Effect of k. We first test the efficiency of the three solutions by varying the parameter k. As shown in Fig. 8, both Quad-tree and Dual-tree outperform Basic. Dual-tree performs the best. It improves on Basic by up to 80% and on Quad-tree by up to 15%. This demonstrates the effectiveness of our proposed techniques. As expected, the performance of all algorithms degrades as k grows. A larger k makes it more difficult to reject a higher level grid, because a rejected grid needs to be dominated no less than k times. Thus, it incurs more overhead during grid splitting.



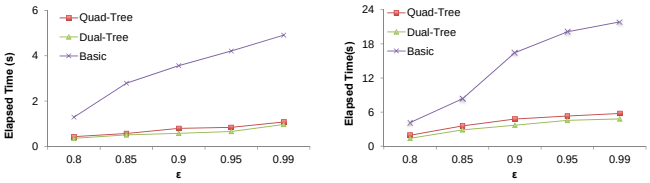
(a) EURO (b) CAR

Figure 8: The Effect of k



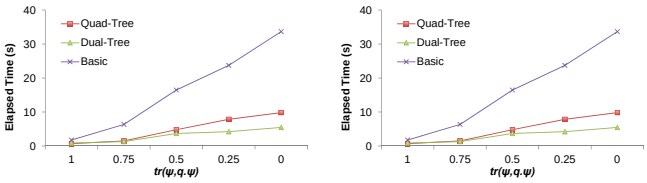
(a) EURO (b) CAR

Figure 9: The Effect of w_s and w_t



(a) EURO (b) CAR

Figure 10: Effect of Precision Threshold ϵ



(a) EURO (b) CAR

Figure 11: Effect of Relevance

When k is small, e.g., $k = 5$ in Fig. 8(a), Dual-tree costs slightly more than Quad-Tree. This is because Dual-tree uses an IR-tree whose fanout is much larger than k, meaning that the benefits gained in Dual-tree's pruning phase are small when compared with Quad-tree. However, as k increases, the advantages of IR-tree based pruning become more significant.

The advantage of Dual-tree over Quad-tree is more significant for the CAR dataset, as shown in Fig. 8(b). In particular, for a larger dataset, more objects can be rejected in the index level in Dual-tree to avoid excessive dominance checking. Therefore, these techniques have more significant effects. It also implies Dual-tree scales well.

Effect of weights (ω_s and ω_t). We vary parameter ω_s and therefore the value of ω_t to see how the algorithms perform with different settings of the weighted distance function. As shown in Fig. 9, when ω_s varies from 0 to 0.75, the elapsed time of all algorithms increases as ω_s grows. In particular, if ω_s is set to 0, the ranking of an object is decided only by the textual relevance. In other words, the ranking is independent of spatial attributes. Therefore, there is no need to decompose the grid in order to determine the dominance relationship.

As ω_s grows, the weight of the textual dimension decreases, and grids of finer granularities are needed to determine the dominance relationship. If ω_s is set to 1, the elapsed time drops compared to the setting of 0.75. The reason is two-fold: 1) if $\omega_s = 1$, all half spaces are convex, which helps to efficiently check dominance; 2) as it ignores the effect of textual dimension, the textual relevance computation can be avoided.

Effect of ϵ. In this section, we vary the value of ϵ to evaluate the performance of each algorithm, as shown in Fig. 10. If ϵ is less than 0.9, the increase of ϵ leads to a significant degradation of the performance. The reason is that a higher precision threshold leads to more grid checks in the query processing. However, when ϵ is over 0.9, the performance degradation is slower. This occurs because although the grid is decomposed into lower levels, there are very few remaining undetermined objects here. So the extra time cost in a low-level grid is low.

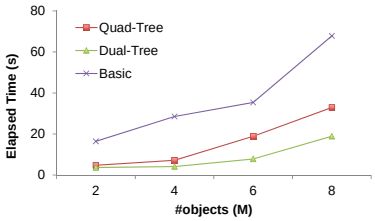


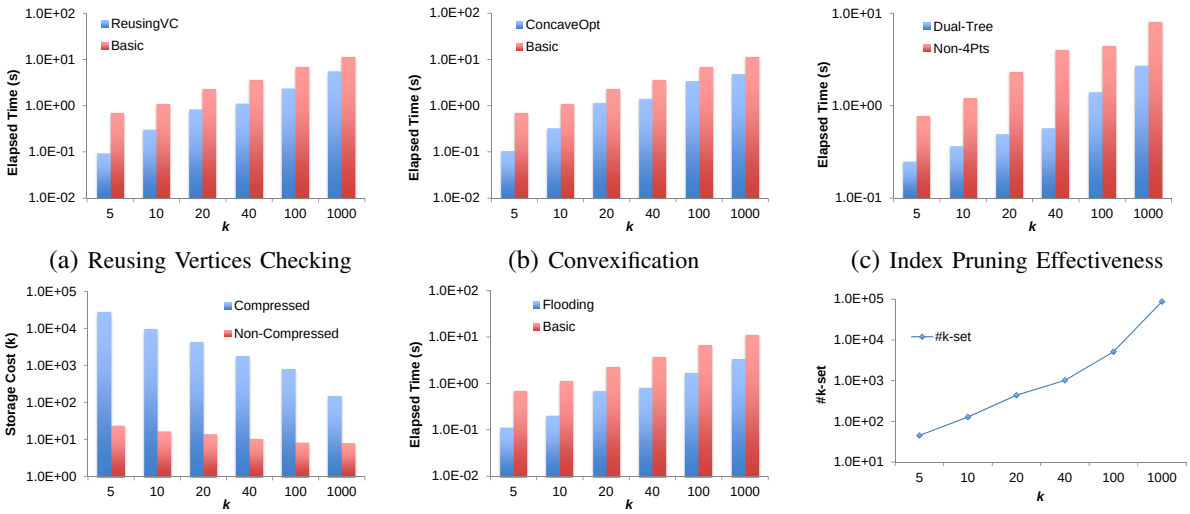
Figure 12: The Scalability Experiment

Effect of textual relevance. Next, we adjust the textual relevance between query keywords and query objects, i.e., $tr(\psi, q.\psi)$. The effect is shown in Fig. 11. Dual-Tree dominates the other two competitors in all settings. Further, the performance of all algorithms degrades when $tr(\psi, q.\psi)$ decreases. The reason is that when $tr(\psi, q.\psi)$ is large, the query object has a better chance of being a top-k object, which results in a bigger V-region. Then, more grids must be examined, which causes more overhead. Note that the cost of Dual-tree increases moderately w.r.t. the relevance, meaning that our algorithms are of good adaptivity to different valued relevance.

Scalability. We observe the scalability of the three solutions with different sized synthetic datasets. As shown in Fig. 12, when the dataset size increases, all three solutions cost more. Compared with Basic and Quad-Tree, which degrade significantly, Dual-tree increases only moderately. When the data cardinality is 6 million objects, Dual-tree costs less than half of that of Quad-tree, thus offering good scalability. The good performance is achieved by using an index that enables pruning at the index entry level.

C. Analysis

We have reported the performance of our proposals. Next, we conduct a detailed analysis to understand how the efficiency is achieved. We report results on EURO; similar observations are found with CAR.



(a) Reusing Vertices Checking (b) Convexification (c) Index Pruning Effectiveness

(d) Pruned Set Compression (e) Quad-tree Root Setting (f) Number of k-subsets

Figure 13: Analysis

Fig. 13(a) shows the benefits gained by reusing vertex checking (Section III-C1). To exclude the effect of other factors, we compare the performance with and without reusing the computation of Basic. We can see that the technique (labeled "ReusingVC") saves up to 60% elapsed time compared to Basic. Note that the y-axis has a logarithmic scale.

Fig. 13(b) demonstrates the effect of the convexification (Section III-A1) that decomposes a concave half space into a set of convex polygons, thus facilitating the dominance determination. Again, we compare the results with and without convexification using Basic. We see that the efficiency gained from our technique is significant. It saves up to 97% elapsed time compared to Basic.

Fig. 13(c) reports the effect of the pruning techniques based on the IR-tree (Section IV-B). The pruning techniques exploit the convex properties of half spaces defined between an object and an index entry. Without them, only distance comparisons can be used (Lemmas 11 and 12), yielding a loose pruning bound. We compare the results using Dual-tree while including or excluding the derived pruning rules, labelled by Dual-tree and Non-4PIs, respectively. The result shows that the pruning techniques improves the performance significantly. When $k = 40$, they save up to an order of magnitude.

Fig. 13(d) covers the compression techniques for the pruned set, which plays as a key role in reducing the space costs. We compare the compressed and non-compressed effects, i.e., with ids of all pruned nodes being recorded for each grid. The figure shows that our compressed solution saves up to 99% space compared to the non-compressed one.

Fig. 13(e) evaluates the effectiveness of the flooding algorithm (Algorithm 3). Instead of decomposing the quad-tree from the root, we start from an appropriate level and retrieve a super region of the V-region where the decomposition starts. By doing so, we skip the first few quad-tree levels where the object rejection is prone to be affected by coarse-size grids. The technique significantly improves the computation time. Specifically, when $k = 40$, up to 60% of the cost can be saved.

The flooding algorithm is supported by estimating the size of a V-region. We have tested the accuracy of the estimation model (Equation 13). When $k = 100$, the estimation accuracy is up to 60%, and the accuracy is even higher if k increases. For example, when $k = 1000$, the accuracy is up to 90%. Hence, the assessment of the model is very positive: 1) the effectiveness of the model is reflected by the efficient results of the flooding algorithm; 2) the correctness can be guaranteed even if the accuracy is extremely high; 3) the flooding algorithm has a bigger effects for large k when the need for efficiency improvement is higher.

Fig. 13(f) reports the number of k-subsets per V-region, which increases sharply as k increases. The result is consistent with our complexity analysis. It also shows that it is not impractical to invoke single Voronoi cell computations multiple times for computing a V-region.

VI. Related Work

A spatial keyword query retrieves the most relevant objects with respect to a given location and a set of keywords. Efficient evaluation of such queries with varying query semantics has been studied, for example, top-k queries [2], reverse top-k queries [5], [20], spatio-textual joins [21], continuous queries [1], [3], [4], why-not queries [22], etc. This paper considers a new problem of reverse keyword-based location search which returns a region where an object always ranks as one of the top-k most relevant objects to the query keywords. It differs from existing works on spatial keyword queries, e.g., reverse top-k queries [5], in the sense that it returns a spatial region instead of a set of objects. The solution of [5] cannot be adapted to our problem since a spatial region has an infinite number of points, which cannot be enumerated. Another study [20] targets a different problem, that is of finding the query keywords that rank a target object as one of the top-k results w.r.t. a given query location.

The reverse location search of non-textual version has been addressed in [7], [17]. However, with different problem settings, the geometric properties and corresponding techniques

are fundamentally different. In [7], [17], a half space is enclosed by straight lines yielding a convex region which is easy for distance comparison and topological operations, e.g., intersection. In our problem, the boundary of a half space is often curved which results in concave regions and thus makes the problem computationally challenging. We avoid the disadvantage by: 1) decomposing Ω space into convex grids; 2) replacing a concave half space with multiple convex polygons; 3) enabling the dominance relationship as an alternative of complex topological relationship between curve shaped regions.

The technique of space decomposition is also mentioned in [4], [23]. However, the problems considered are essentially different from our case. In particular, [23] handles retrieving an uncertain Voronoi cell and [4] handles a weighted uncertain Voronoi cell. They both target on a case of a single Voronoi cell of order-1, whereas a V-region refers to a union of many order-k weighted Voronoi cells. As we have analyzed, the V-region studied in our work is of novel geometric properties. Further, we investigate how such properties, e.g, convexification, can in turn facilitate the space decomposition techniques.

The work relates to Voronoi-based techniques that are widely used in location-based services [8], [11], [24]. However, these works only consider plain Voronoi cells. As we have covered, the features considered in our problem are only partially addressed by them. It is thus not clear on how to extend their work to our scenarios. We summarize the related works in Table V.

Table V: Related Works

Work	weights	order-k Voronoi cell	result as a region	spatial keyword
V-region	✓	✓	✓	✓
[7]	✗	✓	✓	✗
[9]	✗	✗	✓	✗
[3], [25]	✓	✗	✗	✓
[1]	✓	✗	✗	✓
[11]	✗	✗	✗	✗
[8]	✗	✗	✓	✗
[24]	✗	✓	✗	✗
[5]	✗	✗	✗	✓
[4]	✓	✗	✓	✓
[17]	✗	✓	✓	✗

VII. Conclusions

We study the problem of reverse keyword-based location search, covering the concepts, geometric properties, and algorithms needed for deriving the result region, V-region. It is computationally expensive to render such a region. To overcome it, we propose an error-bounded approximate solution using space decomposition techniques, i.e., by a quad-tree. We further utilize an IR-tree based method and thus come up with a dual-tree based solution which evaluates the result region with both efficiency and quality. Empirical performance on real datasets offer insights into the efficiency and scalability of our proposals.

In future research, it is of interest to study how to utilize the techniques for advanced applications, such as modeling safe zones in a mobile setting, influence analysis and progagation in geo-social scenarios, and how to extend the proposal in a constrained space, e.g., road networks.

Acknowledgment

Xike Xie is supported by the CAS Pioneer Hundred Talents Program. In addition, the research was supported in part by National Natural Science Foundation of China (No. 61572193), High Technology Research and Development Program of China (No. 2015AA015801), Hong Kong RGC grants 12201615 and 12244916, and a grant from the Obel Family Foundation. Xin Lin is the corresponding author.

References

[1] W. Huang, G. Li, K.-L. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," *CIKM*, 2012.

[2] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *PVLDB*, 2009.

[3] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," *ICDE*, 2011.

[4] X. Xie, P. Jin, M. Yiu, J. Du, C. Jensen, and M. Yuan, "Enabling scalable geographic service sharing with weighted imprecise voronoi cells," *TKDE*, 2016.

[5] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *SIGMOD*, 2011.

[6] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries," in *SIGMOD*, 2000.

[7] M. A. Cheema, X. Lin, W. Zhang, and Y. Zhang, "Influence zone: Efficiently processing reverse k nearest neighbors queries," in *ICDE*, 2011, pp. 577–588.

[8] M. Yiu, N. Mamoulis, and P. Karras, "Common influence join: A natural join operation for spatial pointsets." *ICDE*, 2008.

[9] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," *SIGMOD*, 2003.

[10] B. Zheng, J. Xu, W.-C. Lee, and L. Lee, "Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services," *VLDBJ*, 2006.

[11] M. Sharifzadeh and C. Shahabi, "Vor-tree: R-trees with voronoi diagrams for efficient processing of spatial nearest neighbor queries," *PVLDB*, 2010.

[12] J. Xu and B. Zheng, "Energy efficient index for querying location-dependent data in mobile broadcast environments," *ICDE*, 2003.

[13] C. Li, Y. Gu, J. Qi, G. Yu, R. Zhang, and Y. Wang, "Processing moving knn queries using influential neighbor sets," *PVLDB*, 2015.

[14] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The V*-Diagram: a query-dependent approach to moving knn queries," *VLDB*, 2008.

[15] G. Albers et al, "Voronoi diagrams of moving points," *Intl. Journal on Computational Geometry and Applications*, 1998.

[16] A. Okabe, B. Boots, K. Sugihara, and S. Chiu, *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2nd ed. Wiley, 2000.

[17] W. Wu, F. Yang, C.-Y. Chan, and K.-L. Tan, "Finch: Evaluating reverse k-nearest-neighbor queries on location data," in *VLDB*, 2008.

[18] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *TODS*, 1999.

[19] X. Xie, M. L. Yiu, R. Cheng, and H. Lu, "Trajectory possible nearest neighbor queries over imprecise location data (technical report)," 2012. [Online]. Available: http://dbtr.cs.aau.dk/DBPublications/DBTR-33.pdf

[20] X. Lin, J. Xu, and H. Hu, "Reverse keyword search for spatio-textual top-k queries in location-based services," *TKDE*, 2015.

[21] H. Hu, G. Li, Z. Bao, J. Feng, Y. Wu, Z. Gong, and Y. Xu, "Top-k spatio-textual similarity join," *TKDE*, 2016.

[22] L. Chen, J. Xu, X. Lin, and C. S. Jensen, "Answering why-not spatial keyword top-k queries via keyword adaption," in *ICDE*, 2016.

[23] T. Emrich, K. A. Schmid, A. Zfle, M. Renz, and R. Cheng, "Uncertain voronoi cell computation based on space decomposition," in *SSTD*, 2015.

[24] Y. Tao, D. Papadias, and X. Lian, "Reverse knn search in arbitrary dimensionality," in *VLDB*, 2004, pp. 744–755.

[25] D. Wu, M. L. Yiu, and C. S. Jensen, "Moving spatial keyword queries: Formulation, methods, and analysis," *TODS*, 2013.