2012 IEEE 28th International Conference on Data Engineering

DESKS: Direction-Aware Spatial Keyword Search

Guoliang Li, Jianhua Feng, Jing Xu

Department of Computer Science, Tsinghua University, Beijing 100084, China
liguoliang@tsinghua.edu.cn; fengjh@tsinghua.edu.cn; xmandbq@gmail.com

***Abstract*— Location-based services (LBS) have been widely accepted by mobile users. Many LBS users have direction-aware search requirement that answers must be in a search direction. However to the best of our knowledge there is not yet any research available that investigates direction-aware search. A straightforward method first finds candidates without considering the direction constraint, and then generates the answers by pruning those candidates which invalidate the direction constraint. However this method is rather expensive as it involves a lot of useless computation on many unnecessary directions. To address this problem, we propose a direction-aware spatial keyword search method which inherently supports direction-aware search. We devise novel direction-aware indexing structures to prune unnecessary directions. We develop effective pruning techniques and search algorithms to efficiently answer a direction-aware query. As users may dynamically change their search directions, we propose to incrementally answer a query. Experimental results on real datasets show that our method achieves high performance and outperforms existing methods significantly.**

I. INTRODUCTION

Location-based services (LBS) have been widely accepted by mobile users. Many online location-based services are available, such as AT&T (http://www.wireless.att.com/lbs) and go2 (http://www.go2.com/). Recently many LBS users have direction-aware search requirement that answers must be in a search direction. For example, a user on the highway wants to find nearest gas stations or restaurants. She has a search requirement that the answers should be in the right front of her driving direction, if in a right-hand traffic country (e.g., US and China). Consider another example that a user is walking to a supermarket. She wants to find an ATM around her walk direction so as to avoid a long walk. In this case she also has a direction-aware search requirement. There are many other direction-aware search requirements in LBS, e.g., multiple destination routing and virtual reality (to show local 3D streetscape). More importantly, many modern mobilephones (e.g., iPhone 4 and HTC) have GPS and compass. We can easily get user's location via the GPS and direction by the compass. Thus we can utilize user's location and search direction to improve user search experiences in LBS.

However to the best of our knowledge there is not yet any research available that investigates direction-aware search. A straightforward method to support direction-aware search first finds the candidates without considering the direction constraint (e.g, [6] and [5]) and then generates the answers by pruning those candidates that invalidate the direction constraint. However this method is rather expensive as it involves a lot of useless computation on many unnecessary directions.

To address this problem, we propose a direction-aware spatial keyword search method, called DESKS, which inherently supports direction-aware search. We first formulate the problem of direction-aware spatial keyword search as follows. Consider a set of Points of Interest (POIs) where each POI is associated with spatial information and textual description. Given a direction-aware spatial keyword query with a location, a direction, and a set of keywords, the direction-aware search finds k nearest neighbors of the query which are in the search direction and contain all input keywords.

To support direction-aware spatial keyword queries, we devise novel direction-aware index structures to prune unnecessary directions. We first group the POIs based on their distances to the bottom-left point of the Minimum Bounding Rectangle (MBR) that contains all POIs. Then for POIs in each group, we sort them based on their directions to the bottom-left point. Given a query, we can deduce a direction range with a lower direction bound and an upper direction bound. We can prove that for any POI if its direction to the bottom-left point is not in the direction range of the query, it will not be an answer, and we can prune the POI. Similarly we can also prune a group of POIs based on the direction range. Motivated by this observation, we develop novel direction-aware index structures, effective pruning techniques, and efficient search algorithms to facilitate direction-aware spatial keyword search. To summarize, we make the following contributions.

- We formulate the problem of direction-aware spatial keyword search and propose an efficient direction-aware search method to address this problem.
- We devise a novel direction-aware index structure which groups the POIs based on their distances and directions. The indexing structures can be used to effectively prune many unnecessary POIs.
- We develop effective pruning techniques and search algorithms to answer direction-aware spatial keyword queries. As mobilephone users may dynamically change search directions, we propose to incrementally answer a query based on the cached results of previously issued queries.
- We have implemented our method, and the experimental results show that our method achieves high performance and outperforms existing methods significantly.

The rest of this paper is organized as follows. We first formulate the problem of direction-aware spatial keyword search and devise a novel indexing structure in Section II. We develop effective pruning techniques in Section III. Section IV gives efficient algorithms to answer a direction-aware query. We discuss how to incrementally answer a query in Section V. Experiment results are provided in Section VI. We review related works in Section VII and conclude in Section VIII.

1084-4627/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDE.2012.93

II. Direction-Aware Spatial Keyword Search

A. Problem Formulation

Data: Consider a set of POIs, $\mathcal{P} = \{p_1, p_2, \cdots, p_{|\mathcal{P}|}\}$. Each POI p_i has a location $(p_i.x, p_i.y)$ where $p_i.x$ is the x-coordinate and $p_i.y$ is the y-coordinate of the POI. p_i is also associated with a set of keywords, denoted by $p_i.d$. Thus a POI is denoted by $p = \langle (p.x, p.y); p.d \rangle$.

Query: A query q contains a location $(q.x, q.y)$ with an x-coordinate $q.x$ and a y-coordinate $q.y$. Query q has a direction constraint $[\alpha, \beta]$, which denotes that the user is only interested in the POIs with directions to q in $[\alpha, \beta]$. Query q contains a set of user-input keywords $\mathcal{K} = \{k_1, k_2, \cdots, k_{|\mathcal{K}|}\}$. Users can specify an integer k to find top-k relevant answers. Thus query q is denoted by $q = \langle (q.x, q.y); [\alpha, \beta]; \mathcal{K}; k \rangle$.

Answer: Let $\mathcal{R}$ denote the Minimum Bounding Rectangle (MBR) that contains all POIs in $\mathcal{P}$. Given a query q with direction $[\alpha, \beta]$, let $\mathcal{S}_q$ denote the sector centered at q with a radius r and an angle from α to β, where r is the maximal distance from q to the boundary of region $\mathcal{R}$. Let $\mathcal{R}_q$ denote the intersection of $\mathcal{S}_q$ and $\mathcal{R}$, which is the search region satisfying the direction constraint. A POI p is an answer of query q, if p is in $\mathcal{R}_q$ and $p.d$ contains all keywords in $\mathcal{K}$. Let $\mathcal{P}_q$ denote the set of all answers of q. We find k nearest neighbors of q from $\mathcal{P}_q$. Next we formulate our problem.

***Definition* 1** (Direction-aware Spatial Keyword Search) Given a set of POIs $\mathcal{P}$ and a query $q = \langle (q.x, q.y); [\alpha, \beta]; \mathcal{K}; k \rangle$, let $\mathcal{P}_q$ denote the set of POIs in $\mathcal{R}_q$ that contain all keywords in $\mathcal{K}$. DESKS finds a subset $\mathcal{P}_q^k$ of $\mathcal{P}_q$ with k POIs such that $\forall p \in \mathcal{P}_q^k$ and $\forall p' \in \mathcal{P}_q - \mathcal{P}_q^k$, $\texttt{dist}(p, q) \leq \texttt{dist}(p', q)$, where $\texttt{dist}(\cdot)$ is a distance function and in the paper we use Euclidean distance*.

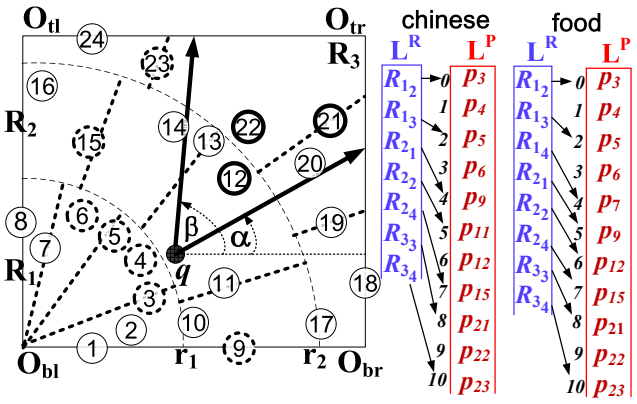
Consider an example in Figure 1. There are 24 POIs. Given a query q with keywords "`chinese food`", the ten highlighted POIs $p_3, p_4, p_5, p_6, p_9, p_{12}, p_{15}, p_{21}, p_{22}, p_{23}$ contain the two keywords. If we have no direction constraint, p_3 and p_4 are two nearest neighbors. If we have direction constraint as shown in Figure 1, p_{12} and p_{22} are two nearest neighbors.

We can extend existing spatial keyword search methods (e.g, [6] and [5]) to support our problem. The method contains two steps. (1) The filter step: It ignores the direction constraint and finds k nearest neighbors of query q which contain all keywords. (2) The verification step: For each found POI in the first step, it checks whether the POI is in the search direction. If yes, it is a k nearest neighbor of q. As most k nearest neighbors of q may invalidate the direction constraint, it needs to repeatedly execute the two steps until finding k answers. Although we can incorporate the verification step into the filter step, this method still needs to visit many unnecessary POIs. To address this problem, we propose a direction-aware spatial keyword search method to achieve a high performance.

B. Direction-aware Indexing Structures

Given a set of POIs, we first generate the MBR $\mathcal{R}$ that contains all POIs. Let $O_{bl}, O_{br}, O_{tr}, O_{tl}$ respectively denote the bottom-left point, the bottom-right point, the top-right point, and the top-left point of $\mathcal{R}$ as shown in Figure 1.

*We suppose $q \in \mathcal{R}$ and our method can be extended to support $q \notin \mathcal{R}$.



$\mathcal{R}_{1_1}=\{p_1,p_2\}, \mathcal{R}_{1_2}=\{p_3,p_4\}, \mathcal{R}_{1_3}=\{p_5,p_6\}, \mathcal{R}_{1_4}=\{p_7,p_8\}$
$\mathcal{R}_{2_1}=\{p_9,p_{10}\}, \mathcal{R}_{2_2}=\{p_{11},p_{12}\}, \mathcal{R}_{2_3}=\{p_{13},p_{14}\}, \mathcal{R}_{2_4}=\{p_{15},p_{16}\}$
$\mathcal{R}_{3_1}=\{p_{17},p_{18}\}, \mathcal{R}_{3_2}=\{p_{19},p_{20}\}, \mathcal{R}_{3_3}=\{p_{21},p_{22}\}, \mathcal{R}_{3_4}=\{p_{23},p_{24}\}$

Fig. 1. A running example

We sort the POIs based on their distances to the bottom-left point O_{bl}. Without loss of generality, assume the sorted POIs are $p_1, p_2, \cdots, p_{|\mathcal{P}|}$ where $\texttt{dist}(p_i, O_{bl}) \leq \texttt{dist}(p_j, O_{bl})$ for $i < j$. Then we evenly partition them into N disjoint buckets, $B_1, B_2, \cdots, B_N$. If every POI has a distinct distance to O_{bl}, we have $B_i = \{p_{(i-1)\times\lambda+1}, \cdots, p_{i\times\lambda}\}$ for $1 \leq i \leq N-1$ and $B_N = \{p_{(N-1)\times\lambda+1}, \cdots, p_{|\mathcal{P}|}\}$ where $\lambda = \lceil \frac{|\mathcal{P}|}{N} \rceil$. If multiple POIs have the same distance to O_{bl}, we partition the POIs into different buckets as follows. We first put the first λ POIs into the first bucket B_1. If $\texttt{dist}(p_{\lambda+1}, O_{bl}) = \texttt{dist}(p_\lambda, O_{bl})$, we add $p_{\lambda+1}$ into B_1; otherwise, we add λ POIs starting with $p_{\lambda+1}$ into B_2. Iteratively we can put each POI into a bucket. Let r_{i-1} denote the smallest distance of POIs in B_i for $1 \leq i \leq N$. We draw $N-1$ arcs centered at O_{bl} with radiuses $r_1, r_2, \cdots, r_{N-1}$. The $N-1$ arcs partition $\mathcal{R}$ into N regions (quarter concentric rings) $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_N$, where $\mathcal{R}_1$ is within r_1, $\mathcal{R}_N$ is outside r_{N-1}, and $\mathcal{R}_i$ is between r_{i-1} and r_i for $1 < i < N$. Obviously the POIs in B_i fall in $\mathcal{R}_i$. Especially a POI on the i-th arc belongs to region $\mathcal{R}_{i+1}$. Obviously the distance of any POI in $\mathcal{R}_i$ to O_{bl} is in $[r_{i-1}, r_i)$ for $1 \leq i < N (r_0 = \texttt{dist}(p_1, O_{bl}))$. For example, in Figure 1, we partition POIs into three regions $\mathcal{R}_1 = \{p_1, \cdots, p_8\}$, $\mathcal{R}_2 = \{p_9, \cdots, p_{16}\}$, and $\mathcal{R}_3 = \{p_{17}, \cdots, p_{24}\}$.

Each POI p in region $\mathcal{R}_i$ has a direction to the bottom-left point O_{bl}, denoted by $p_\theta = \arctan \frac{p.y - O_{bl}.y}{p.x - O_{bl}.x}$. For ease of presentation, suppose $O_{bl} = (0, 0)$. Thus $p_\theta = \arctan \frac{p.y}{p.x}$. We sort POIs in $\mathcal{R}_i$ based on their directions in ascending order. Similarly we evenly partition POIs in $\mathcal{R}_i$ into M buckets $B_{i_1}, B_{i_2}, \cdots, B_{i_M}$. Each bucket contains about $\frac{|\mathcal{P}|}{M \times N}$ POIs. Suppose the minimal direction of POIs in bucket B_{i_j} is $\theta_{i_{j-1}}$ for $1 \leq j \leq M$. We use $M-1$ lines from O_{bl} with directions $\theta_{i_1}, \theta_{i_2}, \cdots, \theta_{i_{M-1}}$ to partition $\mathcal{R}_i$ into M sub-regions (a part of concentric rings) $\mathcal{R}_{i_1}, \mathcal{R}_{i_2}, \cdots, \mathcal{R}_{i_M}$. Obviously the direction of any POI in $\mathcal{R}_{i_j}$ is in $[\theta_{i_{j-1}}, \theta_{i_j})$. For example, in Figure 1, we partition each $\mathcal{R}_i$ into four sub-regions. For instance, we partition $\mathcal{R}_2$ into $\mathcal{R}_{2_1} = \{p_9, p_{10}\}$, $\mathcal{R}_{2_2} = \{p_{11}, p_{12}\}$, $\mathcal{R}_{2_3} = \{p_{13}, p_{14}\}$, and $\mathcal{R}_{2_4} = \{p_{15}, p_{16}\}$.

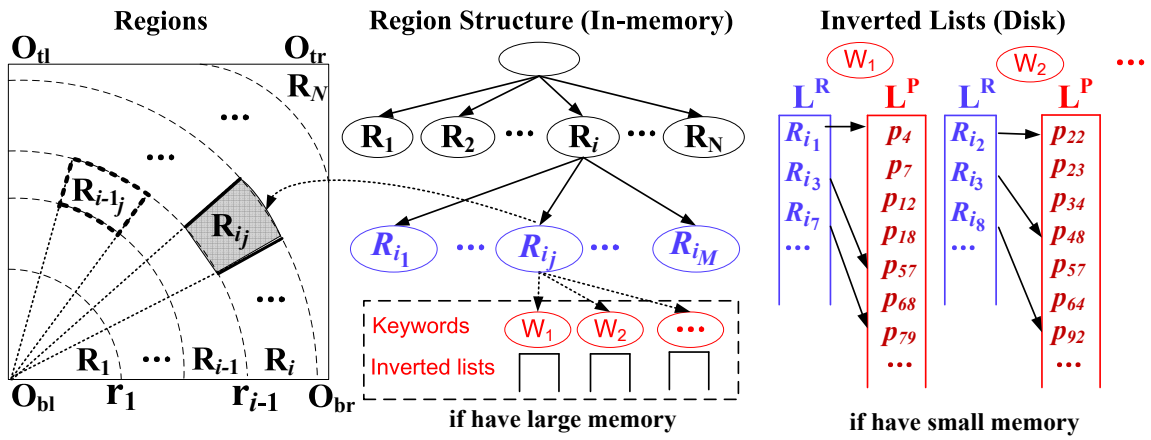


Fig. 2. Indexing structure

Our region structure is illustrated in Figure 2, which has two salient features. Firstly given two sub-regions $\mathcal{R}_{i_s}$ and $\mathcal{R}_{j_t}$, for any POI $p \in \mathcal{R}_{i_s}$ and $p' \in \mathcal{R}_{j_t}$, if $i < j$, we have $\texttt{dist}(p, O_{bl}) < \texttt{dist}(p', O_{bl})$. Secondly given two sub-regions $\mathcal{R}_{i_s}$ and $\mathcal{R}_{i_t}$, for any POI $p \in \mathcal{R}_{i_s}$ and $p' \in \mathcal{R}_{i_t}$, if $s < t$, we have $p_\theta < p'_\theta$. We will use these two features to do efficient pruning. Notice that traditional MBRs have no such features, thus we propose the new index structure to facilitate direction-aware search.

Although we can use the region structure to do spatial pruning, we cannot use it to do textual pruning. To address this issue, we build an inverted list for keywords in each sub-region $\mathcal{R}_{i_j}$. We give the space complexity of our index structure. For the region structure, its space complexity is $\mathcal{O}(M \times N)$. As $M \times N$ is not large (N=1000, M=600 for 16 million POIs, see Section VI), we can keep the region structure in memory. For the inverted lists, suppose each POI contains W distinct keywords in average. The total inverted-list size is $\mathcal{O}(|\mathcal{P}| \times W)$.

If the inverted-list size is very large, we use a disk-based structure. For each keyword k_x, we maintain two inverted lists: (1) The region list $\mathcal{L}^{\mathcal{R}}_{k_x}$ that keeps the sorted IDs of sub-regions that contain k_x. The sub-regions are sorted as follows. $\mathcal{R}_{i_s} < \mathcal{R}_{j_t}$ if $i < j$, and $\mathcal{R}_{i_s} < \mathcal{R}_{i_t}$ if $s < t$; (2) The POI list $\mathcal{L}^{\mathcal{P}}_{k_x}$ that keeps the sorted IDs of POIs that contain k_x: The POIs in different sub-regions are sorted by sub-region order and the POIs in the same sub-region are sorted by directions. In $\mathcal{L}^{\mathcal{R}}_{k_x}$, for each $\mathcal{R}_{i_j} \in \mathcal{L}^{\mathcal{R}}_{k_x}$, we also maintain a pointer to the POI list $\mathcal{L}^{\mathcal{P}}_{k_x}$ that keeps the position of the smallest POI ID in $\mathcal{R}_{i_j} \cap \mathcal{L}^{\mathcal{P}}_{k_x}$. Based on the sorted property, suppose $\mathcal{R}_{i_j}$'s pointer is l_{i_j} and the pointer of its next sub-region is $l_{i_{j+1}}$. We can efficiently find POIs in $\mathcal{R}_{i_j}$ that contain keyword k_x from $\mathcal{L}^{\mathcal{P}}_{k_x}$, e.g., the POIs in $\mathcal{L}^{\mathcal{P}}_{k_x}[l_{i_j}, l_{i_{j+1}})$. Suppose each sub-region $\mathcal{R}_{i_j}$ contains L distinct keywords in average. The space complexity of the disk-based inverted list is $\mathcal{O}(|\mathcal{P}| \times W + L \times M \times N)$.

The overall index structure is shown in Figure 2. Note that to efficiently answer a query, besides building an index structure for O_{bl}, we also maintain index structures for O_{br}, O_{tr}, O_{tl}. Thus the total index size is four times of that for O_{bl}.

For example, in Figure 1, there are 24 POIs. Suppose N=3 and M=4. We generate 12 sub-regions, $\mathcal{R}_{1_1}, \cdots, \mathcal{R}_{1_4}$, $\mathcal{R}_{2_1}, \cdots, \mathcal{R}_{2_4}$, $\mathcal{R}_{3_1}, \cdots, \mathcal{R}_{3_4}$. Each sub-region has two POIs. For example, in $\mathcal{R}_{2_2}$, there are two POIs p_{11} and p_{12}. For keyword "chinese", we maintain a region inverted list which has seven sub-regions and a POI inverted list that has eleven POIs as shown in Figure 1. The pointer of $\mathcal{R}_{1_3}$ is $\mathcal{L}^{\mathcal{P}}_{\texttt{chinese}}[2] = p_5$, that is p_5 is the smallest POI in $\mathcal{R}_{1_3}$ that contains "chinese". Thus we can easily get POIs in $\mathcal{R}_{1_3}$ that contain "chinese" using its pointer as the start position ($\mathcal{L}^{\mathcal{P}}_{\texttt{chinese}}[2]$) and using the pointer of its next sub-region as the end position ($\mathcal{L}^{\mathcal{P}}_{\texttt{chinese}}[4]$), i.e., $\mathcal{L}^{\mathcal{P}}_{\texttt{chinese}}[2, 4) = \{p_5, p_6\}$.

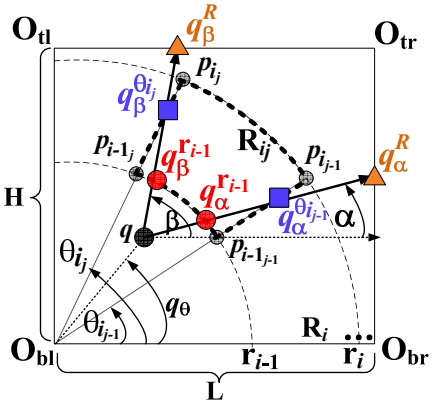


Fig. 3. Notations

In this paper we study how to use our index structures to answer a direction-aware spatial keyword query and leave data update as a future work.

C. Notations

For ease of presentation, we introduce some notations as shown in Figure 3. Let $q_\theta = \arctan\frac{q.y}{q.x}$ denote the direction of q to O_{bl} and $q_d = \texttt{dist}(q, O_{bl})$ denote the distance of q to O_{bl}. Given a region $\mathcal{R}_i$, let r_{i-1} and r_i respectively denote the radius of its inner arc and its outer arc. Given a sub-region $\mathcal{R}_{i_j}$, we use a quadruple to denote the region, $\langle r_{i-1}, r_i, \theta_{i_{j-1}}, \theta_{i_j} \rangle$, where $\theta_{i_{j-1}}$ is the minimum direction and θ_{i_j} is the maximal direction of POIs in $\mathcal{R}_{i_j}$ to O_{bl}. Let $p_{i-1_j}, p_{i-1_{j-1}}, p_{i_j}, p_{i_{j-1}}$ respectively denote the bottom-left point, bottom-right point, top-left point, and top-right point of $\mathcal{R}_{i_j}$ (Figure 3).

Let $q_\alpha^{r_{i-1}}(q_\beta^{r_{i-1}})$ denote the intersection of the line from q with $\alpha(\beta)$ direction and the inner arc of $\mathcal{R}_i$ (with radius r_{i-1}).

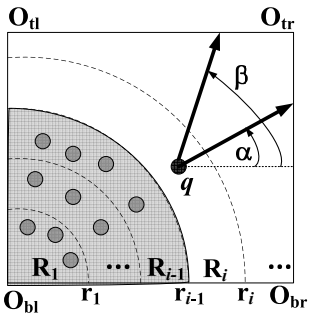
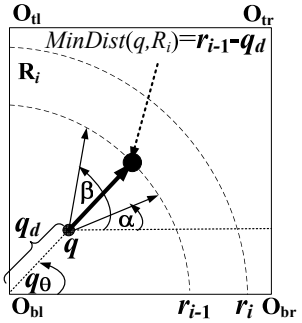
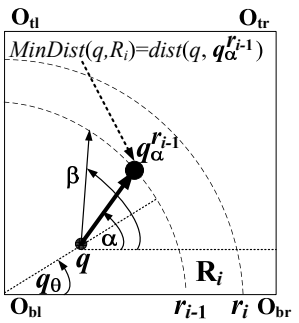


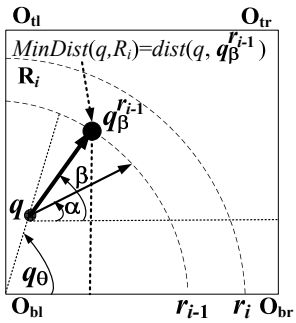
Fig. 4. Pruning $\mathcal{R}_1, \cdots, \mathcal{R}_{i-1}$



(a) $\alpha \le q_\theta \le \beta$



(b) $q_\theta < \alpha$



(c) $q_\theta > \beta$

Fig. 5. MINDIST$(q, \mathcal{R}_i)$

As $q_\alpha^{r_{i-1}}(q_\alpha^{r_{i-1}}.x, q_\alpha^{r_{i-1}}.y)$ is on the arc with radius r_{i-1}, we have $(q_\alpha^{r_{i-1}}.x)^2 + (q_\alpha^{r_{i-1}}.y)^2 = r_{i-1}^2$. In addition, as the point is on the line with direction α to q, $(q_\alpha^{r_{i-1}}.y - q.y)/(q_\alpha^{r_{i-1}}.x - q.x) = \tan\alpha$[†]. Thus we can compute the x-coordinate and y-coordinate of $q_\alpha^{r_{i-1}}$ using the following Equations

$$\begin{cases} (q_\alpha^{r_{i-1}}.y - q.y)/(q_\alpha^{r_{i-1}}.x - q.x) = \tan\alpha \\ (q_\alpha^{r_{i-1}}.x)^2 + (q_\alpha^{r_{i-1}}.y)^2 = r_{i-1}^2 \end{cases} \quad (1)$$

Similarly, we can compute the point $q_\beta^{r_{i-1}}$.

Let $q_\alpha^{\theta_{i_{j-1}}}(q_\alpha^{\theta_{i_j}})$ denote the intersection of the line from q with α direction and the line from O_{bl} with $\theta_{i_{j-1}}(\theta_{i_j})$ direction. Similarly we can define $q_\beta^{\theta_{i_{j-1}}}$ and $q_\beta^{\theta_{i_j}}$. As $q_\alpha^{\theta_{i_{j-1}}}(q_\alpha^{\theta_{i_{j-1}}}.x, q_\alpha^{\theta_{i_{j-1}}}.y)$ is on the line with direction $\theta_{i_{j-1}}$ to O_{bl}, $(q_\alpha^{\theta_{i_{j-1}}}.y)/(q_\alpha^{\theta_{i_{j-1}}}.x) = \tan\theta_{i_{j-1}}$. As the point is on the line with direction α to q, $(q_\alpha^{\theta_{i_{j-1}}}.y - q.y)/(q_\alpha^{\theta_{i_{j-1}}}.x - q.x) = \tan\alpha$. Thus we can compute the x-coordinate and y-coordinate of $q_\alpha^{\theta_{i_{j-1}}}$ using the following Equations

$$\begin{cases} (q_\alpha^{\theta_{i_{j-1}}}.y - q.y)/(q_\alpha^{\theta_{i_{j-1}}}.x - q.x) = \tan\alpha \\ (q_\alpha^{\theta_{i_{j-1}}}.y)/(q_\alpha^{\theta_{i_{j-1}}}.x) = \tan\theta_{i_{j-1}} \end{cases} \quad (2)$$

Similarly, we can compute the points $q_\alpha^{\theta_{i_j}}$, $q_\beta^{\theta_{i_{j-1}}}$, and $q_\beta^{\theta_{i_j}}$.

Suppose the intersection of the line from q with $\alpha(\beta)$ direction and the boundary of $\mathcal{R}$ is $q_\alpha^{\mathcal{R}}(q_\beta^{\mathcal{R}})$ as shown in Figure 3. Next we discuss how to compute $q_\alpha^{\mathcal{R}}$. Suppose q'_θ denote the direction from q to the top-right point O_{tr}. If $\alpha > q'_\theta$, $q_\alpha^{\mathcal{R}}$ will fall on the top line from O_{tl} to O_{tr}. In this case the y-coordinate of $q_\alpha^{\mathcal{R}}$, $q_\alpha^{\mathcal{R}}.y = H$, and x-coordinate of $q_\alpha^{\mathcal{R}}$, $q_\alpha^{\mathcal{R}}.x = q.x + (H - q.y)/\tan\alpha$, where H is the height of the MBR $\mathcal{R}$. If $\alpha = q'_\theta$, $q_\alpha^{\mathcal{R}}$ is exactly O_{tr}. If $\alpha < q'_\theta$, $q_\alpha^{\mathcal{R}}$ will fall on the right line from O_{br} to O_{tr}. In this case the x-coordinate $q_\alpha^{\mathcal{R}}.x = L$ and the y-coordinate $q_\alpha^{\mathcal{R}}.y = q.y + (L - q.x) \times \tan\alpha$, where L is the length of the MBR $\mathcal{R}$. Thus we can compute the point $q_\alpha^{\mathcal{R}}$ as follows.

$$(q_\alpha^{\mathcal{R}}.x, q_\alpha^{\mathcal{R}}.y) = \begin{cases} (q.x + \frac{H - q.y}{\tan\alpha},\ H) & \alpha > q'_\theta \\ (L,\ H) & \alpha = q'_\theta \\ (L,\ q.y + (L - q.x) \times \tan\alpha) & \alpha < q'_\theta \end{cases} \quad (3)$$

Similarly we can compute $q_\beta^{\mathcal{R}}$. We will use the above-mentioned points to do pruning in the following sections.

[†]In this section, we suppose $0 \le \alpha \le \beta \le \frac{\pi}{2}$ and our technique can be easily extended to support other directions (Section IV).

III. PRUNING UNNECESSARY REGIONS

In this section, we propose effective pruning techniques to prune unnecessary regions $\mathcal{R}_i$ (Section III-A) and $\mathcal{R}_{i_j}$ (Section III-B). We first consider the direction in $0 \le \alpha \le \beta \le \frac{\pi}{2}$ and discuss how to support any direction in Section IV.

A. Pruning Region $\mathcal{R}_i$

Consider regions $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_N$ with the radiuses of their outer circles respectively $r_1, r_2, \cdots, r_N$. Given a query q, we first locate in which region q appears. To this end, we first compute its distance to O_{bl}, q_d. Then we use a binary search on $r_1, r_2, \cdots, r_N$ to find the first radius which is larger than q_d. Suppose we find r_i such that $r_{i-1} \le q_d < r_i$ as shown in Figure 4. We can prove that any POI in regions $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_{i-1}$ will not be an answer of query q, as they are not in the search direction as formalized in Lemma 1.

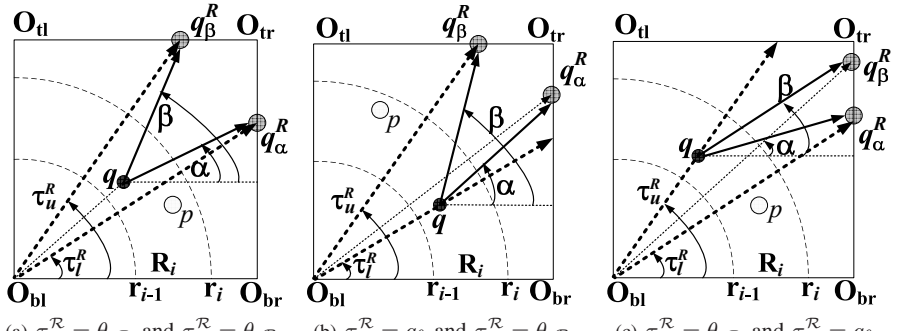
***Lemma* 1** Given a query point q with $0 \le \alpha \le \beta \le \frac{\pi}{2}$, suppose $r_{i-1} \le q_d < r_i$. Any POI in $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_{i-1}$ cannot be an answer of q[‡].

Lemma 1 holds for any query with direction $0 \le \alpha \le \beta \le \frac{\pi}{2}$. For example, in Figure 1, we can directly prune region $\mathcal{R}_1$ and all POIs in $\mathcal{R}_1$ do not need to be accessed. Note that it may not hold if $\beta > \frac{\pi}{2}$. Consider a counter-example where a query q is on the bottom line from O_{bl} to O_{br}. If β is larger than $\frac{\pi}{2}$, the search direction may have overlap with $\mathcal{R}_{i-1}$. Similarly α should be no smaller than 0 and the counter-example is a query on the left line from O_{bl} to O_{tl}.

MINDIST function for $\mathcal{R}_i$: To facilitate nearest neighbor search, traditional methods use function MINDIST to estimate the distance between a query and an MBR [10]. Formally, given a query q and an MBR `mbr`, function MINDIST$(q, \texttt{mbr})$ returns the minimal distance of q to `mbr`. As $\mathcal{R}_i$ in our method is not an MBR and our query has direction constraint, we extend the function to support our problem as follows.

If q is outside the outer arc of $\mathcal{R}_i(q_d \ge r_i)$, we have MINDIST$(q, \mathcal{R}_i)$=$\infty$ based on Lemma 1. If q is in $\mathcal{R}_i(r_{i-1} \le q_d < r_i)$, we have MINDIST$(q, \mathcal{R}_i) = 0$. If q is inside the inner arc of $\mathcal{R}_i(q_d < r_{i-1})$, we give the function as follows. Consider the direction of q to O_{bl}, q_θ. If $\alpha \le q_\theta \le \beta$, the nearest neighbor of q in $\mathcal{R}_i$ is the intersection of the line with q_θ direction and the inner arc of $\mathcal{R}_i$ with radius r_{i-1}(Figure 5(a)). Thus MINDIST$(q, \mathcal{R}_i) = r_{i-1} - q_d$. If $q_\theta < \alpha$, the nearest

[‡]In this paper, we omit the proofs of Lemmas due to space constraints.



(a) $\tau_l^{\mathcal{R}} = \theta_{q_\alpha^{\mathcal{R}}}$ and $\tau_u^{\mathcal{R}} = \theta_{q_\beta^{\mathcal{R}}}$ (b) $\tau_l^{\mathcal{R}} = q_\theta$ and $\tau_u^{\mathcal{R}} = \theta_{q_\beta^{\mathcal{R}}}$ (c) $\tau_l^{\mathcal{R}} = \theta_{q_\alpha^{\mathcal{R}}}$ and $\tau_u^{\mathcal{R}} = q_\theta$

Fig. 6. Direction-based pruning for regions $\mathcal{R}_i, \cdots, \mathcal{R}_N$

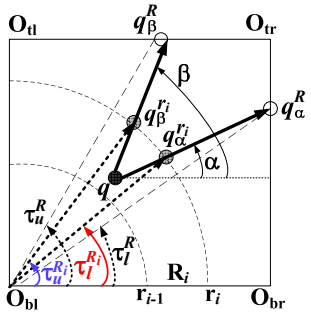


Fig. 7. Direction-based Pruning for $\mathcal{R}_i$

neighbor of q in $\mathcal{R}_i$ is $q_\alpha^{r_{i-1}}$ which is the intersection of the line from q with α direction and the inner arc of $\mathcal{R}_i$(Figure 5(b)). Thus $\text{MINDIST}(q, \mathcal{R}_i) = \texttt{dist}(q, q_\alpha^{r_{i-1}})$. Similarly if $q_\theta > \beta$, $\text{MINDIST}(q, \mathcal{R}_i) = \texttt{dist}(q, q_\beta^{r_{i-1}})$ (Figure 5(c)).

Thus we give the MINDIST function as follows.

$$\text{MINDIST}(q, \mathcal{R}_i) = \begin{cases} \infty & q_d \geq r_i \\ 0 & r_{i-1} \leq q_d < r_i \\ r_{i-1} - q_d & q_d < r_{i-1} \ \& \ \alpha \leq q_\theta \leq \beta \\ \texttt{dist}(q, q_\alpha^{r_{i-1}}) & q_d < r_{i-1} \ \& \ q_\theta < \alpha \\ \texttt{dist}(q, q_\beta^{r_{i-1}}) & q_d < r_{i-1} \ \& \ q_\theta > \beta \end{cases} \tag{4}$$

where $q_\alpha^{r_{i-1}}$ and $q_\beta^{r_{i-1}}$ can be computed using Equation 1.

Given a query q, we first find its located region $\mathcal{R}_i$ and access the POIs in $\mathcal{R}_i$. Then we verify whether the POIs satisfy the direction constraint and contain all keywords. Suppose the k-th smallest distance of the candidates that have been computed is d_k. Then for the next region $\mathcal{R}_{i+1}$, if $\text{MINDIST}(q, \mathcal{R}_{i+1}) \geq d_k$, we terminate and prune $\mathcal{R}_{i+1}, \cdots, \mathcal{R}_N$; otherwise we access POIs in $\mathcal{R}_{i+1}$. Iteratively we can find all answers. As we use the best-first search method, we only utilize MINDIST function and will not use MINMAXDIST function [10]. For example, in Figure 1, suppose $k = 1$. In $\mathcal{R}_2$, we find an answer p_{12}. As $\text{MINDIST}(q, \mathcal{R}_3) > \texttt{dist}(q, p_{12})$, we terminate and prune POIs in $\mathcal{R}_3$.

However this method neglects the fact that some sub-regions $\mathcal{R}_{i_j}$ in $\mathcal{R}_i$ may not satisfy the direction constraint. For example, in Figure 1, although $\mathcal{R}_{2_1}$ has a POI p_9 which contains all keywords, we can prune the region as it is not in the search direction. Similarly we can prune $\mathcal{R}_{2_4}$. To achieve our goal, we discuss how to effectively prune $\mathcal{R}_{i_j}$ in $\mathcal{R}_i$.

B. Pruning Regions $\mathcal{R}_{i_j}$

In this section, we first introduce how to prune some unnecessary sub-regions $\mathcal{R}_{i_j}$ which have no overlap with the search direction, and then give the function $\text{MINDIST}(q, \mathcal{R}_{i_j})$. In the rest of this paper, if the context is clear, the term "region" and "sub-region" are used interchangeably for $\mathcal{R}_{i_j}$.

Our indexing structure has a salient feature: If a POI p is an answer of q, its direction ($p_\theta = \arctan \frac{p.y}{p.x}$) to O_{bl} must be in a range $[\tau_l^{\mathcal{R}}, \tau_u^{\mathcal{R}}]$. In other words, we can prune the POIs with direction smaller than $\tau_l^{\mathcal{R}}$ or larger than $\tau_u^{\mathcal{R}}$. Next we discuss how to deduce the lower bound $\tau_l^{\mathcal{R}}$ and the upper bound $\tau_u^{\mathcal{R}}$.

Given query q with direction $[\alpha, \beta]$, consider the intersection $q_\alpha^{\mathcal{R}}(q_\beta^{\mathcal{R}})$ of the line from q with $\alpha(\beta)$ direction and the boundary of region $\mathcal{R}$ as shown in Figure 6. Let $\theta_{q_\alpha^{\mathcal{R}}}$ and $\theta_{q_\beta^{\mathcal{R}}}$ respectively denote the directions of points $q_\alpha^{\mathcal{R}}$ and $q_\beta^{\mathcal{R}}$ to O_{bl}. As $\alpha \leq \beta$, $\theta_{q_\alpha^{\mathcal{R}}} \leq \theta_{q_\beta^{\mathcal{R}}}$. Let $\tau_l^{\mathcal{R}} = \min(\theta_{q_\alpha^{\mathcal{R}}}, q_\theta)$ and $\tau_u^{\mathcal{R}} = \max(\theta_{q_\beta^{\mathcal{R}}}, q_\theta)$. For any point p, if $p_\theta > \tau_u^{\mathcal{R}}$, its direction to q must be larger than β, thus p cannot be an answer of q (Figure 6(b)). Similarly, if $p_\theta < \tau_l^{\mathcal{R}}$, its direction to q must be smaller than α, thus p cannot be an answer of q (Figure 6(c)). The correctness is formalized in Lemma 2.

***Lemma* 2** Given a query q with direction $[\alpha, \beta]$, let $\tau_l^{\mathcal{R}} = \min(\theta_{q_\alpha^{\mathcal{R}}}, q_\theta)$ and $\tau_u^{\mathcal{R}} = \max(\theta_{q_\beta^{\mathcal{R}}}, q_\theta)$. For any POI p, if $p_\theta > \tau_u^{\mathcal{R}}$ or $p_\theta < \tau_l^{\mathcal{R}}$, p cannot be an answer of q.

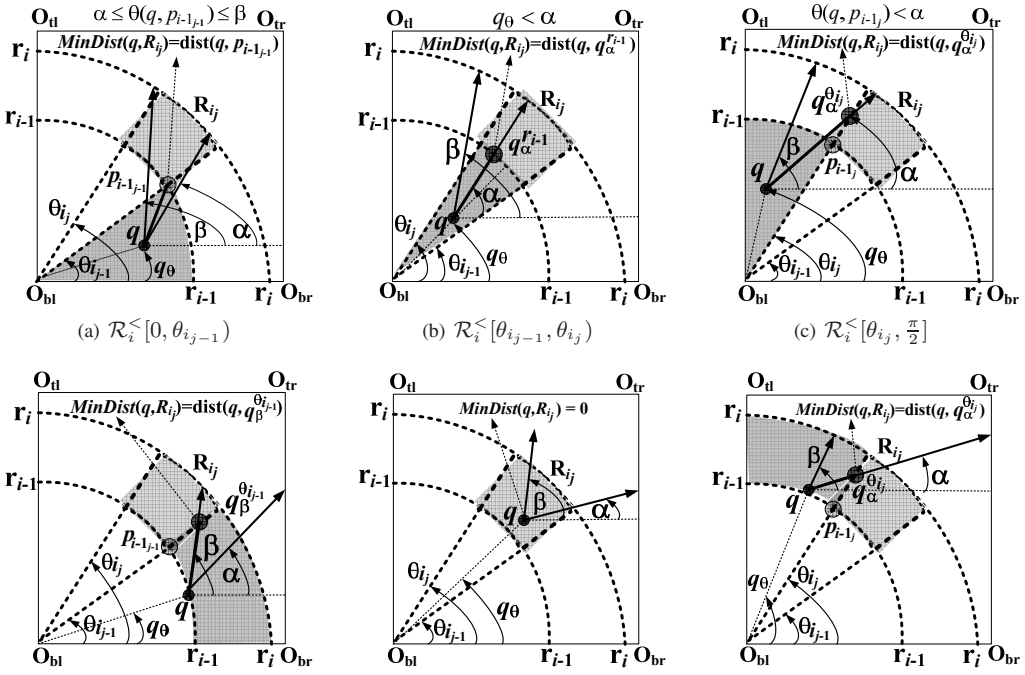
Based on Lemma 2 we only need to access the POIs with directions between $\tau_l^{\mathcal{R}}$ and $\tau_u^{\mathcal{R}}$. Moreover, a region $\mathcal{R}_{i_j}$ has a lower direction bound $\theta_{i_{j-1}}$ and an upper direction bound θ_{i_j}, which respectively denote the minimal direction and the maximal direction of POIs in $\mathcal{R}_{i_j}$. In other words, for any POI $p \in \mathcal{R}_{i_j}$ we have $\theta_{i_{j-1}} \leq p_\theta < \theta_{i_j}$. Based on Lemma 2, for region $\mathcal{R}_{i_j}$ with direction $[\theta_{i_{j-1}}, \theta_{i_j})$, if $\theta_{i_j} \leq \tau_l^{\mathcal{R}}$ or $\theta_{i_{j-1}} > \tau_u^{\mathcal{R}}$, we can prune the region $\mathcal{R}_{i_j}$ as formalized in Lemma 3.

***Lemma* 3** Given a query q with direction $[\alpha, \beta]$, let $\tau_l^{\mathcal{R}} = \min(\theta_{q_\alpha^{\mathcal{R}}}, q_\theta)$ and $\tau_u^{\mathcal{R}} = \max(\theta_{q_\beta^{\mathcal{R}}}, q_\theta)$. For any region $\mathcal{R}_{i_j}$ with direction $[\theta_{i_{j-1}}, \theta_{i_j})$, if $\theta_{i_j} \leq \tau_l^{\mathcal{R}}$ or $\theta_{i_{j-1}} > \tau_u^{\mathcal{R}}$, any POI in $\mathcal{R}_{i_j}$ cannot be an answer of q.

For example, in Figure 1, although $\mathcal{R}_{2_1}$ and $\mathcal{R}_{2_4}$ have POIs that contain all keywords, we can prune them as they are not in search direction based on the direction-based pruning technique in Lemma 3. Notice that this pruning technique is valid for all regions. Next we devise tighter direction bounds for region $\mathcal{R}_i$. Let $\tau_l^{\mathcal{R}_i}$ denote the tighter lower bound and $\tau_u^{\mathcal{R}_i}$ denote the tighter upper bound for $\mathcal{R}_i$. For any POI p in $\mathcal{R}_i$, if $p_\theta < \tau_l^{\mathcal{R}_i}$ or $p_\theta > \tau_u^{\mathcal{R}_i}$, we can prune the POI. Next we discuss how to deduce the two tighter bounds.

Consider the intersection of the line from q with $\alpha(\beta)$ direction and the outer arc of $\mathcal{R}_i$, denoted by $q_\alpha^{r_i}(q_\beta^{r_i})$. The two points can be computed by Equation 1. Let $\theta_{q_\alpha^{r_i}}, \theta_{q_\beta^{r_i}}$ respectively denote the directions of points $q_\alpha^{r_i}, q_\beta^{r_i}$ to O_{bl}. It is easy to figure out that if $q_\alpha^{r_i}$ is in region $\mathcal{R}$ (denoted by $q_\alpha^{r_i} \in \mathcal{R}$), $\theta_{q_\alpha^{r_i}} \geq \theta_{q_\alpha^{\mathcal{R}}}$; otherwise $\theta_{q_\alpha^{r_i}} < \theta_{q_\alpha^{\mathcal{R}}}$ (Figure 7). Similarly if $q_\beta^{r_i} \in \mathcal{R}$, $\theta_{q_\beta^{r_i}} \leq \theta_{q_\beta^{\mathcal{R}}}$; otherwise $\theta_{q_\beta^{r_i}} > \theta_{q_\beta^{\mathcal{R}}}$. Based on this observation, we give the tighter bounds $\tau_l^{\mathcal{R}_i}$ and $\tau_u^{\mathcal{R}_i}$.

$$\tau_l^{\mathcal{R}_i} = \begin{cases} q_\theta & q_\theta \leq \alpha \\ \theta_{q_\alpha^{r_i}} & q_\theta > \alpha \ \ \& \ \ q_\alpha^{r_i} \in \mathcal{R} \\ \theta_{q_\alpha^{\mathcal{R}}} & q_\theta > \alpha \ \ \& \ \ q_\alpha^{r_i} \notin \mathcal{R} \end{cases} \tag{5}$$



(a) $\mathcal{R}_i^<[0,\theta_{i_{j-1}})$ (b) $\mathcal{R}_i^<[\theta_{i_{j-1}},\theta_{i_j})$ (c) $\mathcal{R}_i^<[\theta_{i_j},\frac{\pi}{2}]$

(d) $\mathcal{R}_i[0,\theta_{i_{j-1}})$ (e) $\mathcal{R}_i[\theta_{i_{j-1}},\theta_{i_j})$ (f) $\mathcal{R}_i[\theta_{i_j},\frac{\pi}{2}]$

Fig. 8. MINDIST$(q,\mathcal{R}_{i_j})$

$$\tau_u^{\mathcal{R}_i}=\begin{cases} q_\theta & q_\theta \geq \beta \\ \theta_{q_\beta^{r_i}} & q_\theta < \beta \ \ \& \ \ q_\beta^{r_i} \in \mathcal{R} \\ \theta_{q_\beta^{\mathcal{R}}} & q_\theta < \beta \ \ \& \ \ q_\beta^{r_i} \notin \mathcal{R} \end{cases} \quad (6)$$

Then consider region $\mathcal{R}_{i_j}$ with the minimal direction $\theta_{i_{j-1}}$ and the maximal direction θ_{i_j}. If $\theta_{i_j} \leq \tau_l^{\mathcal{R}_i}$ or $\theta_{i_{j-1}} > \tau_u^{\mathcal{R}_i}$, region $\mathcal{R}_{i_j}$ has no overlap with the search direction, thus we can prune $\mathcal{R}_{i_j}$. In other words, for $\mathcal{R}_i$, we only need to access the regions $\mathcal{R}_{i_l},\cdots,\mathcal{R}_{i_u}$, such that $\theta_{i_{l-1}}\leq\tau_l^{\mathcal{R}_i}<\theta_{i_l}$ and $\theta_{i_{u-1}}\leq\tau_u^{\mathcal{R}_i}<\theta_{i_u}$. To efficiently identify such regions, we use $\tau_l^{\mathcal{R}_i}$ to do a binary search on the directions of regions in $\mathcal{R}_i$, $\{\theta_{i_1},\cdots,\theta_{i_M}\}$, and find the smallest one which is larger than $\tau_l^{\mathcal{R}_i}$, i.e., $\mathcal{R}_{i_l}$. Then we use $\tau_u^{\mathcal{R}_i}$ to do a binary search on the directions in $\{\theta_{i_{l+1}},\cdots,\theta_{i_M}\}$, and find the largest one which is smaller than $\tau_u^{\mathcal{R}_i}$, i.e., $\mathcal{R}_{i_u}$. Thus we only need to access $\mathcal{R}_{i_l},\cdots,\mathcal{R}_{i_u}$. Lemma 4 formalizes the pruning technique.

***Lemma* 4** Given a query q with direction $[\alpha,\beta]$ and a region $\mathcal{R}_i$, let $\tau_l^{\mathcal{R}_i}=\min(\theta_{q_\alpha^{r_i}},q_\theta)$ and $\tau_u^{\mathcal{R}_i}=\max(\theta_{q_\beta^{r_i}},q_\theta)$. For any POI $p\in\mathcal{R}_i$, if $p_\theta>\tau_u^{\mathcal{R}_i}$ or $p_\theta<\tau_l^{\mathcal{R}_i}$, p cannot be an answer of q; For any region $\mathcal{R}_{i_j}\in\mathcal{R}_i$ with direction $[\theta_{i_{j-1}},\theta_{i_j})$, if $\theta_{i_j}\leq\tau_l^{\mathcal{R}_i}$ or $\theta_{i_{j-1}}>\tau_u^{\mathcal{R}_i}$, any POI in $\mathcal{R}_{i_j}$ cannot be an answer of q.

Consider the example in Figure 1. We can prune regions $\mathcal{R}_{2_1}$ and $\mathcal{R}_{2_4}$ in $\mathcal{R}_2$, and regions $\mathcal{R}_{3_1}$ and $\mathcal{R}_{3_4}$ in $\mathcal{R}_3$.

MINDIST for $\mathcal{R}_{i_j}$: For each region $\mathcal{R}_{i_j}$ in $\{\mathcal{R}_{i_l},\cdots,\mathcal{R}_{i_u}\}$, we use MINDIST function to estimate the distance between q and $\mathcal{R}_{i_j}$, i.e., MINDIST$(q,\mathcal{R}_{i_j})$. To this end, we partition $\mathcal{R}$ into three regions by the inner arc (r_{i-1}) and the outer arc (r_i), i.e., the region inside the inner arc $\mathcal{R}_i^<$, the region $\mathcal{R}_i$, and the region outside $\mathcal{R}_i^>$. Obviously, if $q\in\mathcal{R}_i^>$, any POI in $\mathcal{R}_{i_j}$ will not be an answer of q based on Lemma 1, thus MINDIST$(q,\mathcal{R}_{i_j})=\infty$. For $\mathcal{R}_i^<$ and $\mathcal{R}_i$, we respectively partition them into three regions based on the two directions $\theta_{i_{j-1}}$ and θ_{i_j}, denoted by $\mathcal{R}_i^<[0,\theta_{i_{j-1}}),\mathcal{R}_i^<[\theta_{i_{j-1}},\theta_{i_j}),\mathcal{R}_i^<[\theta_{i_j},\frac{\pi}{2}]$, and $\mathcal{R}_i[0,\theta_{i_{j-1}}),\mathcal{R}_i[\theta_{i_{j-1}},\theta_{i_j}),\mathcal{R}_i[\theta_{i_j},\frac{\pi}{2}]$ (Figure 8).

(1) $q\in\mathcal{R}_i^<[0,\theta_{i_{j-1}})$ **(Figure 8(a)).** If we have no direction constraint, the nearest neighbor of q is the bottom-right point $p_{i-1_{j-1}}$. Next, we consider the case with direction $[\alpha,\beta]$. Let $\theta(q,p_{i-1_{j-1}})$ denote the direction from q to $p_{i-1_{j-1}}$. If $\alpha\leq\theta(q,p_{i-1_{j-1}})\leq\beta$, the nearest neighbor of q is still $p_{i-1_{j-1}}$. If $\theta(q,p_{i-1_{j-1}})<\alpha$, the nearest neighbor of q is $q_\alpha^{r_{i-1}}$, which is the intersection of the line from q with α direction and the arc with radius r_{i-1} (computed by Equation 1). Similarly if $\theta(q,p_{i-1_{j-1}})>\beta$, the nearest neighbor of q is $q_\beta^{\theta_{i_{j-1}}}$, which is the intersection of the line from q with β direction and the line from O_{bl} with $\theta_{i_{j-1}}$ direction (computed by Equation 2).

(2) $q\in\mathcal{R}_i^<[\theta_{i_{j-1}},\theta_{i_j})$ **(Figure 8(b)).** If $\alpha\leq q_\theta\leq\beta$, the nearest neighbor of q is $q_\theta^{r_{i-1}}$ which is the intersection of the line from q with q_θ direction and the arc with radius r_{i-1}. The distance is $r_{i-1}-q_d$. If $q_\theta<\alpha$, the nearest neighbor of q is $q_\alpha^{r_{i-1}}$. If $q_\theta>\beta$, the nearest neighbor of q is $q_\beta^{r_{i-1}}$.

(3) $q\in\mathcal{R}_i^<[\theta_{i_j},\frac{\pi}{2}]$ **(Figure 8(c)).** Similar to case (1), consider the bottom-left point p_{i-1_j}. Let $\theta(q,p_{i-1_j})$ denote the direction from q to p_{i-1_j}. If $\alpha\leq\theta(q,p_{i-1_j})\leq\beta$, the nearest neighbor of q is p_{i-1_j}. If $\theta(q,p_{i-1_j})<\alpha$, the nearest neighbor of q is $q_\alpha^{\theta_{i_j}}$. If $\theta(q,p_{i-1_j})>\beta$, the nearest neighbor of q is $q_\beta^{r_{i-1}}$.

(4) $q\in\mathcal{R}_i[0,\theta_{i_{j-1}})$ **(Figure 8(d)).** As $\beta\leq\frac{\pi}{2}$, the nearest neighbor of q must be $q_\beta^{\theta_{i_{j-1}}}$ (computed by Equation 2).

(5) $q\in\mathcal{R}_i[\theta_{i_{j-1}},\theta_{i_j})$ **(Figure 8(e)).** As q is in $\mathcal{R}_{i_j}$, MINDIST$(q,\mathcal{R}_{i_j})$=0.

(6) $q\in\mathcal{R}_i[\theta_{i_j},\frac{\pi}{2}]$ **(Figure 8(f)).** As $\alpha\geq 0$, the nearest neighbor of q must be $q_\alpha^{\theta_{i_j}}$ (computed by Equation 2).

To summarize, we give function MINDIST$(q,\mathcal{R}_{i_j})$ in Table I.

TABLE I

MINDIST$(q, \mathcal{R}_{i_j})$

Regions	MINDIST$(q, \mathcal{R}_{i_j})$
$\mathcal{R}_i^{\geq}$	∞
$\mathcal{R}_i^{<}[0, \theta_{i_{j-1}})$	$\begin{cases} \texttt{dist}(q, q_\alpha^{r_{i-1}}) & \theta(q, p_{i-1_{j-1}}) < \alpha \\ \texttt{dist}(q, p_{i-1_{j-1}}) & \alpha \leq \theta(q, p_{i-1_{j-1}}) \leq \beta \\ \texttt{dist}(q, q_\beta^{\theta_{i_{j-1}}}) & \theta(q, p_{i-1_{j-1}}) > \beta \end{cases}$
$\mathcal{R}_i^{<}[\theta_{i_{j-1}}, \theta_{i_j})$	$\begin{cases} \texttt{dist}(q, q_\alpha^{r_{i-1}}) & q_\theta < \alpha \\ r_{i-1} - q_d & \alpha \leq q_\theta \leq \beta \\ \texttt{dist}(q, q_\beta^{r_{i-1}}) & q_\theta > \beta \end{cases}$
$\mathcal{R}_i^{<}[\theta_{i_j}, \frac{\pi}{2}]$	$\begin{cases} \texttt{dist}(q, q_\alpha^{\theta_{i_j}}) & \theta(q, p_{i-1_j}) < \alpha \\ \texttt{dist}(q, p_{i-1_j}) & \alpha \leq \theta(q, p_{i-1_j}) \leq \beta \\ \texttt{dist}(q, q_\beta^{r_{i-1}}) & \theta(q, p_{i-1_j}) > \beta \end{cases}$
$\mathcal{R}_i[0, \theta_{i_{j-1}})$	$\texttt{dist}(q, q_\beta^{\theta_{i_{j-1}}})$
$\mathcal{R}_i[\theta_{i_{j-1}}, \theta_{i_j})$	0
$\mathcal{R}_i[\theta_{i_j}, \frac{\pi}{2}]$	$\texttt{dist}(q, q_\alpha^{\theta_{i_j}})$

IV. SEARCH ALGORITHMS

In this section, we first give an algorithm to answer a query with direction $0 \leq \alpha \leq \beta \leq \frac{\pi}{2}$(Section IV-A), and then discuss how to answer a query with any direction (Section IV-B).

A. Answering Queries with $0 \leq \alpha \leq \beta \leq \frac{\pi}{2}$

We combine our pruning techniques and MINDIST functions to answer a query with direction $0 \leq \alpha \leq \beta \leq \frac{\pi}{2}$. Figure 9 gives the pseudo-code of our algorithm. To efficiently find k nearest neighbors of q, we maintain a priority queue $\mathcal{Q}$ (line 2) and keep the k-th smallest distance of POIs in $\mathcal{Q}$ to q (d_k) that have already been computed (line 3). Given a query q, we first locate which region query q appears using a binary search method on radiuses $r_1, r_2, \cdots, r_N$ (line 4). Suppose we find $\mathcal{R}_i$ such that $r_{i-1} \leq q_d < r_i$. If MINDIST$(q, \mathcal{R}_i) \geq d_k$, we terminate as there is no answer in $\mathcal{R}_i \cdots \mathcal{R}_N$ (line 6); otherwise for each region $\mathcal{R}_i$, we find the "*candidate regions*" which have overlap with the search direction and contain all keywords in $\mathcal{K}$, by calling function FINDCANDREGIONS(line 7). Next for each candidate region $\mathcal{R}_{i_j} \in \mathcal{C}_{\mathcal{R}_i}$, if MINDIST$(q, \mathcal{R}_{i_j}) \geq d_k$, we break as there is no answer in $\mathcal{R}_{i_j} \cdots \mathcal{R}_{i_M}$ (line 9); otherwise we find "*candidate POIs*" in $\mathcal{R}_{i_j}$ which are in the search direction and contain all keywords, by calling function FINDCANDPOIS(line 10). Finally we need to access region $\mathcal{R}_{i+1}$ if necessary (line 11). Iteratively we can find the k nearest neighbors of query q.

Then we discuss how to compute the candidate regions in $\mathcal{R}_i$. Function FINDCANDREGIONS gives the pseudo-code (Figure 9). We first compute the lower direction bound $\tau_l^{\mathcal{R}_i}$ and the upper direction bound $\tau_u^{\mathcal{R}_i}$ (line 2). Next we find the regions satisfying the direction constraint $\mathcal{R}_i[\alpha, \beta] = \{\mathcal{R}_{i_l}, \cdots, \mathcal{R}_{i_u}\}$ (in $[\tau_l^{\mathcal{R}_i}, \tau_u^{\mathcal{R}_i}]$) using a binary search method on the directions $\theta_{i_1}, \cdots, \theta_{i_M}$ (line 3). Then if the inverted lists are in memory, we check whether each region in $\mathcal{R}_i[\alpha, \beta]$ contains all keywords and add such regions into candidate-region set $\mathcal{C}_{\mathcal{R}_i}$. If we use a disk-based method, we load region inverted lists for each keyword $\mathcal{L}_{k_i}^{\mathcal{R}}$(line 4), compute their intersection $\mathcal{L}_{\mathcal{K}}^{\mathcal{R}}$ that satisfies keyword constraint (line 5), intersect the regions satisfying keyword constraint $\mathcal{L}_{\mathcal{K}}^{\mathcal{R}}$ with the regions satisfying region constraint $\mathcal{R}_i[\alpha, \beta]$, and get $\mathcal{R}_i^{\mathcal{K}}[\alpha, \beta]$ (line 6). For each region $\mathcal{R}_{i_j} \in \mathcal{R}_i^{\mathcal{K}}[\alpha, \beta]$, if MINDIST$(q, \mathcal{R}_{i_j}) < d_k$, we add $\mathcal{R}_{i_j}$ into the candidate-region set $\mathcal{C}_{\mathcal{R}_i}$ (line 8). Finally we sort the regions in $\mathcal{C}_{\mathcal{R}_i}$ based on the MINDIST function in ascending order (line 9).

Next we discuss how to compute the candidate POIs in $\mathcal{R}_{i_j}$. Function FINDCANDPOIS gives the pseudo-code (Figure 9). If the POI inverted lists are in memory, we directly compute

Algorithm 1: DESKS-BAISC $(\mathcal{P}, q)$

Input: $\mathcal{P}$: A collection of POIs
$q = \langle (q.x, q.y); [\alpha, \beta]; \mathcal{K}, k \rangle$: A query
Output: $\mathcal{P}_q^k = \{p | p \in \mathcal{P}_q$ and p is a knn of $q\}$, where $\mathcal{P}_q$ is the set of POIs in the search direction that contain all the keywords in $\mathcal{K}$.

```
1  begin
2      Initialize an empty priority queue Q;
3      Let d_k denote the k-th smallest distance in Q ;
4      Locate the region R_i where q appears using a binary
       search on r_1, ..., r_N ;
5      while i ≤ N do
6          if MINDIST(q, R_i) ≥ d_k then return;
7          else C_{R_i} = FINDCANDREGIONS(q, R_i, d_k) ;
8          for R_{i_j} ∈ C_{R_i} (C_{R_i} are sorted) do
9              if MINDIST(q, R_{i_j}) ≥ d_k then break;
10             else FINDCANDPOIS(q, R_{i_j}, d_k, Q) ;
11         i = i + 1 ;
12 end
```

Function FINDCANDREGIONS$(q, \mathcal{R}_i, d_k)$

Input: $q = \langle (q.x, q.y); [\alpha, \beta]; \mathcal{K}, k \rangle$: A query
d_k: The k-th smallest distance in $\mathcal{Q}$
$\mathcal{R}_i$: Region $\mathcal{R}_i$
Output: $\mathcal{C}_{\mathcal{R}_i}$: A sorted candidate-region set

```
1  begin
2      Compute direction bounds τ_l^{R_i} and τ_u^{R_i} ;
3      Find regions R_i[α, β] = {R_{i_l}, ..., R_{i_u}} in
       [τ_l^{R_i}, τ_u^{R_i}] using a binary search on θ_{i_1} ... θ_{i_M} ;
4      Load region inverted lists L_{k_i}^R for k_i ∈ K ;
5      Compute L_K^R = ∩_{k_i ∈ K} L_{k_i}^R ;
6      Compute R_i^K[α, β] = R_i[α, β] ∩ L_K^R ;
7      for R_{i_j} ∈ R_i^K[α, β] do
8          if MINDIST(q, R_{i_j}) < d_k then C_{R_i} ← R_{i_j} ;
9      Sort C_{R_i} based on the MINDIST function ;
10 end
```

Function FINDCANDPOIS$(q, \mathcal{R}_{i_j}, d_k, \mathcal{Q})$

Input: $q = \langle (q.x, q.y); [\alpha, \beta]; \mathcal{K}, k \rangle$: A query
d_k: The k-th smallest distance in $\mathcal{Q}$
$\mathcal{R}_{i_j}$: Region $\mathcal{R}_{i_j}$; $\mathcal{Q}$: Queue

```
1  begin
2      Load POI inverted lists L_{k_i}^P(R_{i_j}) for k_i ∈ K ;
3      Compute intersection L_K^P = ∩_{k_i ∈ K} L_{k_i}^P(R_{i_j}) ;
4      for p ∈ L_K^P do
5          if α ≤ θ(q, p) ≤ β & dist(q, p) < d_k then
6              add p into Q, and update Q and d_k ;
7  end
```

Fig. 9. DESKS-BAISC algorithm (using disk-based inverted lists)

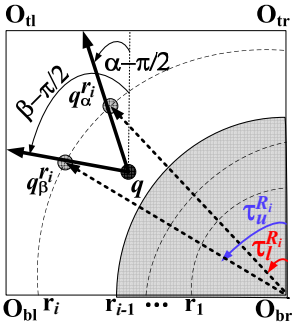


Fig. 10. Pruning for $[\frac{\pi}{2} \le \alpha < \beta < \pi]$

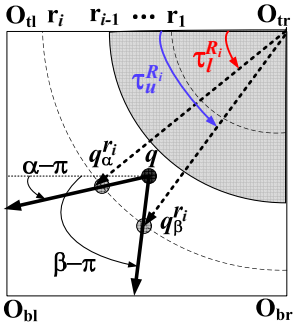


Fig. 11. Pruning for $[\pi \le \alpha < \beta < \frac{3\pi}{2}]$

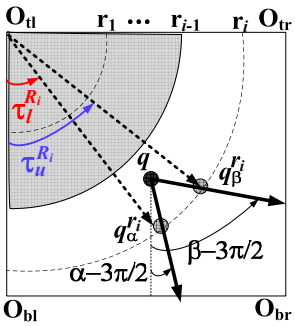


Fig. 12. Pruning for $[\frac{3\pi}{2} \le \alpha < \beta < 2\pi]$

their intersection. If the POI inverted lists are on disk, we load the POI inverted lists for each keyword. Note that for k_i we only load POIs that are in $\mathcal{R}_{i_j}$, $\mathcal{L}^{\mathcal{P}}_{k_i}(\mathcal{R}_{i_j})$, based on the pointers in region lists as shown in Figure 1 (line 2). Then we compute the intersection of POI lists $\mathcal{L}^{\mathcal{P}}_{\mathcal{K}} = \cap_{k_i \in \mathcal{K}} \mathcal{L}^{\mathcal{P}}_{k_i}(\mathcal{R}_{i_j})$(line 3). For $p \in \mathcal{L}^{\mathcal{P}}_{\mathcal{K}}$, if $\alpha \le \theta(q,p) \le \beta$ and $\texttt{dist}(q,p) < d_k$, p is a candidate. We add p into the priority queue and update d_k(line 6).

B. Answering Queries with Any Direction

In this section, we discuss how to answer a query with arbitrary directions. We first classify queries into basic queries and complex queries as follows.

- **Case 1 – Basic Queries:**
  - $0 \le \alpha \le \beta \le \frac{\pi}{2}$. We answer it using the index structures on O_{bl} as discussed in the above sections.
  - $\frac{\pi}{2} \le \alpha \le \beta \le \pi$. We answer it using the index structures on O_{br}, which is similar to answer a query with $0 \le \alpha \le \beta \le \frac{\pi}{2}$ as shown in Figure 10.
  - $\pi \le \alpha \le \beta \le \frac{3\pi}{2}$. We answer it using the index structures on O_{tr}, which is similar to answer a query with $0 \le \alpha \le \beta \le \frac{\pi}{2}$ as shown in Figure 11.
  - $\frac{3\pi}{2} \le \alpha \le \beta \le 2\pi$. We answer it using the index structures on O_{tl}, which is similar to answer a query with $0 \le \alpha \le \beta \le \frac{\pi}{2}$ as shown in Figure 12.
- **Case 2 – Complex Queries:** All other queries are called complex queries. For a complex query q with direction $[\alpha, \beta]$, we decompose q into at most four basic queries: (1) q_1 with direction $[0, \frac{\pi}{2}) \cap [\alpha, \beta]$; (2) q_2 with direction $[\frac{\pi}{2}, \pi) \cap [\alpha, \beta]$; (3) q_3 with direction $[\pi, \frac{3\pi}{2}) \cap [\alpha, \beta]$; and (4) q_4 with direction $[\frac{3\pi}{2}, 2\pi) \cap [\alpha, \beta]$. Thus we can first answer the sub-queries and then combine the results to generate the final answers of query q. [§]

A straightforward method to answer a complex query first decomposes it into basic sub-queries and then computes k nearest neighbors for each basic query. Finally it finds the real k nearest neighbors by combing the results of each basic query. However this method is very expensive as some sub-queries may have no real answers and we do not need to answer such sub-queries. To this end, we propose an efficient algorithm by pruning many unnecessary POIs. For each basic query, we first compute their candidate regions. Then we sort the candidate regions based on their MINDIST values. Next we access the candidate regions in order and prune unnecessary regions. The pseudo-code of the algorithm is shown in Figure 13.

[§]We use $\alpha \in [0, 2\pi)$ and $\beta \le \alpha + 2\pi$ to denote any direction. If $\beta > 2\pi$, we decompose the direction to $[\alpha, 2\pi)$ and $[2\pi, \beta] = [0, \beta - 2\pi]$. Then we decompose them to basic queries and generate at most five sub-queries.

Algorithm 2: DESKS $(\mathcal{P}, q)$

Input: $\mathcal{P}$: A collection of POIs
$q = \langle (q.x, q.y); [\alpha, \beta]; \mathcal{K}, k \rangle$: A query
Output: $\mathcal{P}_q^k = \{p | p \in \mathcal{P}_q$ and p is a knn of $q\}$, where $\mathcal{P}_q$ is the set of POIs in the search direction that contain all the keywords in $\mathcal{K}$.

```
1  begin
2      Initialize an empty priority queue Q_P for POIs;
3      Let d_k denote the k-th small distance in Q_P;
4      Initialize an empty priority queue Q_R for regions;
5      Decompose q into q_1, q_2, ··· , q_4; /*some may be empty*/
6      for 1 ≤ s ≤ 4 do
7          Locate region R_{i^s} for q_s where q_s appears;
8          add R_{i^s} into Q_R;
9      while Q_R ≠ φ do
10         Get region R_{i^m} with minimal MINDIST(q, R_{i^m});
11         if MINDIST(q, R_{i^m}) ≥ d_k then return;
12         else C_{R_{i^m}} =FINDCANDREGIONS(q, R_{i^m}, d_k);
13         for R_{i^m_j} ∈ C_{R_{i^m}} do
14             if MINDIST(q, R_{i^m_j}) ≥ d_k then break;
15             else FINDCANDPOIS(q, R_{i^m}, d_k, Q_P) ;
16         Pop R_{i^m} from Q_R;
17         if MINDIST(q, R_{i^m+1}) < d_k then
18             add R_{i^m+1} into Q_R;
19 end
```

Fig. 13. DESKS algorithm (using disk-based inverted lists)

We maintain two priority queues: $\mathcal{Q}_{\mathcal{P}}$ for candidate POIs (line 2) and $\mathcal{Q}_{\mathcal{R}}$ for regions (line 4). We first decompose the query into at most four sub-queries (line 5). Then for each sub-query q_s, we locate which region q_s appears (line 7) and add the region $\mathcal{R}_{i^s}$ into region queue $\mathcal{Q}_{\mathcal{R}}$ (line 8). Then we find region $\mathcal{R}_{i^m}$ with the minimal MINDIST value in $\mathcal{Q}_{\mathcal{R}}$ (line 10). If $\text{MINDIST}(q, \mathcal{R}_{i^m}) \ge d_k$, we terminate as we have found k nearest neighbors (line 11); otherwise we find candidate regions in $\mathcal{R}_{i^m}$, $\mathcal{C}_{\mathcal{R}_{i^m}}$ (line 12). For each candidate region $\mathcal{R}_{i^m_j} \in \mathcal{C}_{\mathcal{R}_{i^m}}$, if $\text{MINDIST}(q, \mathcal{R}_{i^m_j}) \ge d_k$, we break as there is no answer in $\mathcal{R}_{i^m_j} \cdots \mathcal{R}_{i^m_M}$ (line 14); otherwise, we compute candidate POIs in $\mathcal{R}_{i^m_j}$ (line 15). Next we pop $\mathcal{R}_{i^m}$ from $\mathcal{Q}_{\mathcal{R}}$. For the next region $\mathcal{R}_{i^m+1}$ after $\mathcal{R}_{i^m}$, if $\text{MINDIST}(q, \mathcal{R}_{i^m+1}) < d_k$, we add it into region queue $\mathcal{Q}_{\mathcal{R}}$ (line 18). Iteratively we can find all answers of query q.

V. Incremental Search Algorithms

Mobilephone users will dynamically change directions if they cannot find expected answers in the current direction. A naive method is to answer a new query from scratch. However this method is very expensive. To address this issue, we propose to incrementally answer a query based on the cached results of previously issued queries. To avoid involving huge space, we only cache k nearest neighbors for a query. We consider the following two cases to update a direction¶.

Case 1: The user increases a direction from $[\alpha, \beta]$ to $[\alpha' < \alpha, \beta' > \beta]$. This corresponds to the case that the user increases the direction using two fingers on the mobilephone screen. Section V-A discusses how to answer such a query efficiently.

Case 2: The user moves the direction from $[\alpha, \beta]$ to $[\alpha + \delta_\theta, \beta + \delta_\theta]$. This corresponds to the case that the user changes the direction by moving the mobilephone direction. Section V-B discusses how to answer such a query efficiently.

Note that our method can support any direction-update queries using these two operations.

A. Increasing The Direction

Suppose a user has issued a query q with direction $[\alpha, \beta]$ and then the user issues a new query q' by increasing the direction to $[\alpha' < \alpha, \beta' > \beta]$. We use the cached results of q to answer this new query q' as follows. Obviously, an answer of q must be an answer of q'. Let $d'_k(d_k)$ denote the k-th smallest distance of nearest neighbors to query $q'(q)$. We have $d'_k \leq d_k$. Thus we can use d_k as an upper bound.

We insert k nearest neighbors of q into the priority queue of q'. Then we decompose q' into three queries, $q_1[\alpha', \alpha]$, $q_2[\beta, \beta']$, and $q[\alpha, \beta]$. We only need to answer q_1 and q_2 with bound d_k. We answer the two queries simultaneously as answering sub-queries in Section IV-B. Note that in the two new directions, if there is a POI p (or region $\mathcal{R}_{i_j}$) with distance to q larger than d_k, we prune p (or region $\mathcal{R}_{i_j}$); otherwise we insert it into the priority queue (or access the region). Thus we can incrementally and efficiently answer query q'.

B. Moving The Direction

Suppose a user has issued a query q with direction $[\alpha, \beta]$ and then the user issues a new query q' by moving the direction to $[\alpha + \delta_\theta, \beta + \delta_\theta]$. Firstly consider $\delta_\theta > 0$. If $\alpha + \delta_\theta > \beta$, q and q' have no overlapped direction and we answer the new query from scratch. On the contrary, q and q' have an overlapped direction $[\alpha + \delta_\theta, \beta]$. We examine each k nearest neighbors of q, and if it is in $[\alpha + \delta_\theta, \beta]$, we insert it into the priority queue of query q' and update the k-th smallest threshold d'_k. Then we answer the new query with direction $[\beta, \beta + \delta_\theta]$ using threshold d'_k. If we find k answers in the priority queue or in direction $[\beta, \beta + \delta_\theta]$ within distance d_k, we do not need to access regions in direction $[\alpha + \delta_\theta, \beta]$; otherwise, we need to access those regions in direction $[\alpha + \delta_\theta, \beta]$ with MinDist values no smaller than d_k. Thus we can use the bound d'_k to do effective pruning. Similarly if $\delta_\theta < 0$ and $\beta + \delta_\theta > \alpha$, we can use the above method to answer query q' with direction $[\alpha + \delta_\theta, \alpha]$. Thus we can incrementally and efficiently answer query q'.

¶In this paper we do not consider moving queries (changing locations).

VI. Experimental Study

We have implemented our proposed methods. We compared with two state-of-the-art methods MIR^2-tree [6] and LkT [5]‖. We extended their methods to support direction-aware search by examining whether each accessed MBR (or POI) is in search direction. For LkT, we got the codes from the authors [5] which were implemented in Java. For MIR^2-tree, we implemented it in C++. Our algorithms were also implemented in C++. All the C++ codes were compiled using GCC 4.2.3 with -O3 flag. As the baseline algorithms used disk-based indexes, we also used disk-based index structure. All the experiments were run on a Ubuntu machine with an Intel Core E5450 3.0GHz CPU and 4 GB memory.

We used three real datasets, POIs in California(CA), POIs in Virginia(VA), and POIs in China(CN). The statistics of the datasets was summarized in Table II. We generated five query sets with keyword numbers from 1 to 5 and each query set had 1000 queries.

TABLE II
Datasets.

	CA	VA	CN
Total number of POIs (millions)	0.91	0.96	16.5
Total number of terms (millions)	9.7	4.6	63.6
Total number of unique terms (thousands)	35	26	753
Average number of unique terms per POI	8.57	4.5	3.85

A. Varying M and N

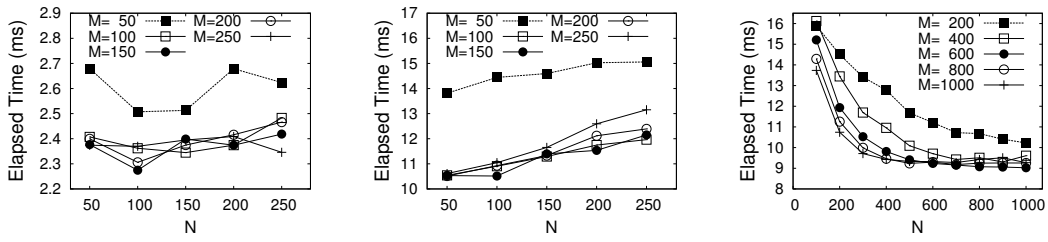
In this section, we evaluate the effect on varying region number N and sub-region number M. Figure 14 shows the results. We see that different values of N and M had no significant effect on the performance for $M > 50$. On the VA dataset, the running time was about 2.3-2.7 ms on every combinations of M and N, and we got the highest performance at N=100 and M=150. On the CA dataset, the running time was 11-15 ms for different M and N values, and we got the highest performance at N=100 and M=150. On the CN dataset, the time was about 9-16 ms. The highest performance was achieved at N=1000 and $M = 600$. Based on the results, we had a conclusion that each region $\mathcal{R}_i$ was better to contain 10, 000 POIs and each sub-region $\mathcal{R}_{i_j}$ was better to contain 100 POIs. In the reminder experiments, we used N=100 and M=150 on the CA and VA datasets, and N=1000 and M=600 on the CN dataset.

B. Evaluation on Pruning Techniques

In this section, we evaluate our pruning techniques. We implemented three methods. (1) Desks+R: We used the region-pruning techniques and function MinDist$(q, \mathcal{R}_i)$ to prune $\mathcal{R}_i$. (2) Desks+D: We used the direction-pruning techniques and function MinDist$(q, \mathcal{R}_{i_j})$ to prune $\mathcal{R}_{i_j}$. (3) Desks+RD: We used both region pruning and direction pruning.

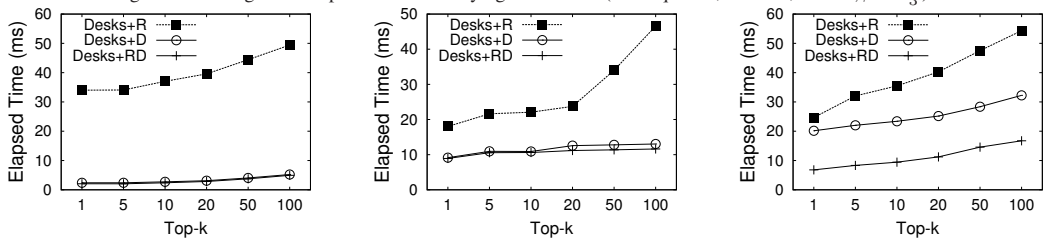
Varying k: We first evaluated the pruning techniques by varying k on the 5000 queries and α=0, β=$\frac{\pi}{3}$. Figure 15 shows

‖As MIR^2-tree generally achieves much higher performance than IR^2-tree, we do not report results for IR^2-tree.



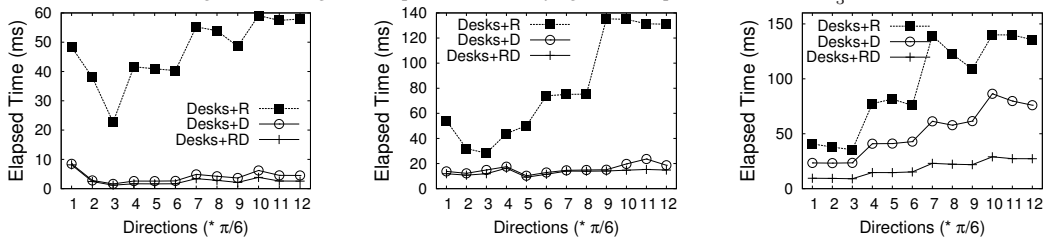
(a) VA (b) CA (c) CN

Fig. 14. Average search performance: Varying M and N (5000 queries, $k = 10$, $\alpha = 0, \beta = \frac{\pi}{3}$)



(a) VA (b) CA (c) CN

Fig. 15. Average search performance: Varying k (5000 queries, $\alpha = 0, \beta = \frac{\pi}{3}$)



(a) VA (b) CA (c) CN

Fig. 16. Average search performance: Varying directions $\beta - \alpha$ from $\frac{\pi}{6}$ to 2π (5000 queries, $k = 10$)

the results. We can see that DESKS+D and DESKS+RD significantly outperformed DESKS+R. This is because DESKS+R needed to access many unnecessary regions and the direction-based pruning can prune large numbers of unnecessary regions. DESKS+RD was also better than DESKS+D, especially on the CN dataset. This is because DESKS+RD can prune many regions $\mathcal{R}_i$. For example, on the CN dataset, for k=100, DESKS+R took 55 ms, DESKS+D improved it to 32 ms, and DESKS+RD further improved it to 16 ms. There are two reasons that the improvement of DESKS+RD over DESKS+D was not significant on the CA and VA datasets. Firstly, there were small numbers of POIs that contain all keywords. Both DESKS+D and DESKS+RD needed to access many regions. Secondly, there were small numbers of regions ($\mathcal{R}_i$). As N=100, DESKS+RD cannot prune large numbers of regions.

Varying directions: We evaluated the pruning techniques by varying directions on 5000 queries and $k = 10$. Figure 16 shows the results. Similarly DESKS+D and DESKS+RD significantly outperformed DESKS+R. On the VA dataset, DESKS+R took more than 20 ms to answer a query, and DESKS+D and DESKS+RD only took about 2 ms. This is because DESKS+R needed to enumerate many regions while DESKS+D and DESKS+RD can prune large numbers of regions based on the direction-aware indexes.

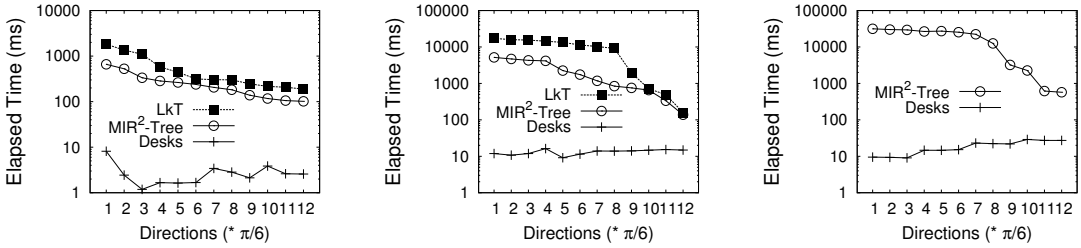
C. Comparison with Existing Methods

We compared our algorithm DESKS (DESKS+RD) with state-of-the-art methods MIR^2-tree and LkT. We first compared the index sizes and time as shown in Table III. LkT was very expensive to build indexes as it needed to cluster keywords in POIs. On the CN dataset, it took more than 2 days to index 1 million POIs, and it will take 1 month to index 16 million POIs. Thus we did not show the results on the CN dataset. MIR^2-tree used R-tree and keyword signatures to build indexes. Although DESKS had larger index sizes than MIR^2-tree (as DESKS built indexes for $O_{bl}, O_{br}, O_{tr}, O_{tl}$), DESKS still had acceptable index sizes. LkT had much larger index sizes as it built inverted lists for each R-tree node.

TABLE III
INDEXING TIME AND SIZES.

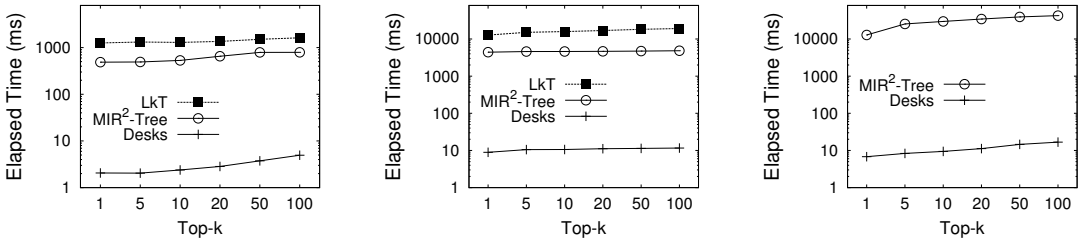
Data	Sizes(MB)	Index Sizes (MB)			Index Time (Minutes)		
		MIR^2-tree	LkT	DESKS	MIR^2-tree	LkT	DESKS
CA	72.2	72	1430	265	1.3	780	1.8
VA	54.8	76	920	149	0.8	690	1.2
CN	805	1304	–	3552	25	–	33

Varying directions: We first compared different methods by varying directions on 5000 queries and $k = 10$. Figure 17 shows the results. Although LkT and MIR^2-tree achieved high performance for large directions, they were very slow for small directions. This is because they needed to enumerate many MBRs and POIs, which was very expensive. For example, on the CA dataset, they took 200 ms for direction 2π, but took more than 5 seconds for direction $\frac{\pi}{3}$. DESKS only took 20 ms for any direction, since DESKS can use the index to do effective direction pruning. Even for the direction with 2π, DESKS still outperformed existing methods. There are three reasons. Firstly, our region structure is very effective and can be in memory. Secondly, our region inverted lists can prune many unnecessary POIs. Thirdly, existing methods usually



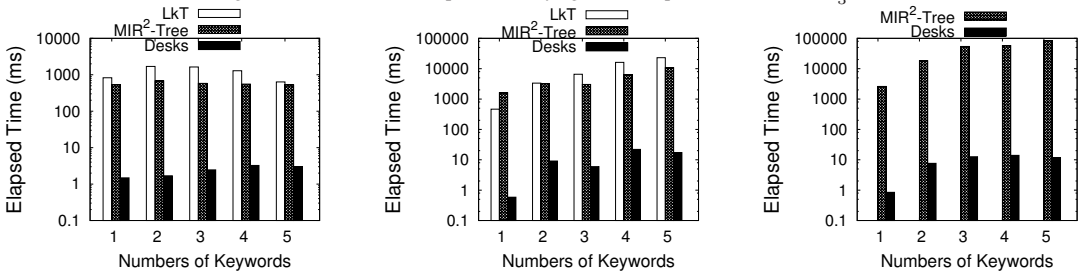
(a) VA (b) CA (c) CN

Fig. 17. Performance comparison: Varying directions $\beta - \alpha$ from $\frac{\pi}{6}$ to 2π (5000 queries, $k = 10$)



(a) VA (b) CA (c) CN

Fig. 18. Performance comparison: Varying k (5000 queries, $\alpha = 0, \beta = \frac{\pi}{3}$)



(a) VA (b) CA (c) CN

Fig. 19. Performance comparison: Varying numbers of keywords (1000 queries in each query set, $k = 10$, $\alpha = 0, \beta = \frac{\pi}{3}$)

achieved high performance for POIs with many keywords (documents) [5]. However real POIs have no many keywords.

Varying k: Then we compared different methods by varying k on 5000 queries and $\alpha = 0, \beta = \frac{\pi}{3}$. Figure 18 shows the results. We can see that DESKS significantly outperformed MIR^2-tree and LkT, even in 2-3 orders of magnitude. On the VA dataset, MIR^2-tree and LkT took about 500 ms, and DESKS improved the time to 2-5 ms. The main reason is that existing methods cannot use the index to do effective direction pruning. DESKS used the novel direction-aware index which can prune large numbers of unnecessary regions and POIs.

Varying the number of keywords: Next we compared different methods by varying keyword numbers and setting $k = 10$ and $\alpha = 0, \beta = \frac{\pi}{3}$. Figure 19 shows the results. We can see that for different numbers of keywords, DESKS was still much better than MIR^2-tree and LkT. For different numbers of keywords, DESKS only took about 10-20 ms.

D. Evaluation on Incremental Search

In this section, we test our incremental search method. We first initialized queries with $\beta - \alpha = \frac{\pi}{3}$ and then increased directions by $\frac{\pi}{36}, \cdots, \frac{12\pi}{36}$. Figure 20(a) shows the results. We can see that our incremental method DESKS-INCRE outperformed DESKS. This is because DESKS-INCRE can incrementally answer a query using the previously issued queries. We also evaluated DESKS-INCRE by moving directions. Figure 20(b) shows the results. We still initialized queries with $\beta - \alpha = \frac{\pi}{3}$ and then moved the directions by $\frac{-6\pi}{36}, \cdots, \frac{6\pi}{36}$. We can see that for a small direction, DESKS-INCRE was much better than DESKS, as DESKS-INCRE can use a tighter bound to answer new queries. For a large direction, the improvement was not high as DESKS-INCRE needed to answer queries from scratch.

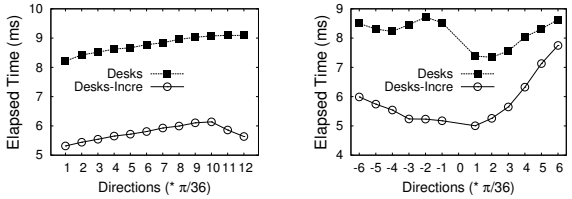
E. Scalability

In this section, we evaluate the scalability on the CN dataset by varying numbers of POIs. Figure 21 shows the results with different k values and directions. We can see that our method scaled very well. This is contributed to our effective direction-aware index structures and effective pruning techniques.

VII. Related Work

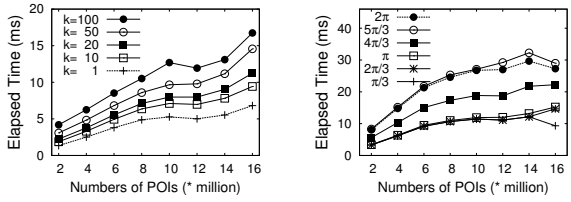
Many studies on spatial keyword search have been proposed recently [25], [3], [9], [6], [23], [5], [24], [22], [1], [21], [2], [19], [13]. The most related work to our problem is the study by Felipe et al. [6], which proposed the index structures by integrating signature files and R-tree to enable top-k spatial keyword queries. Another similar study [5] is provided by Cong et al., which combined inverted files and R-tree to answer the location-aware top-k text retrieval (LkT) query. Our direction-aware spatial keyword query is different from their methods as we have a direction constraint.

Zhou et al. [25] proposed to find web documents relevant to user input keywords within a pre-specified region. They developed several methods by combining R-tree and inverted indexes. Chen et al. [3] extended this problem by supporting



(a) Increasing directions (b) Moving directions

Fig. 20. Incremental Search on the CN dataset ($k = 10$)



(a) Varying k ($\beta - \alpha = \frac{\pi}{3}$) (b) Varying directions ($k = 10$)

Fig. 21. Scalability on the CN dataset

large numbers of "footprint representations." Hariharan et al. [9] focused on finding objects containing a set of keywords within a specific region. They proposed a hybrid index structures by integrating R-tree and inverted lists. Zhang et al. [23], [24] introduced the m-closest keyword query (mCK query) which aims at finding the closest objects that match keywords. Cong et al. [1] studied how to find top-k prestige-based relevant spatial web objects. Yao et al. [22] tackled the problem of answering approximate string match queries in spatial databases. Wu et al. [21] studied spatial keyword search for moving objects. Lu et al. [13] extended reversed knn techniques to support reverse spatial and textual k nearest neighbor search. Roy and Chakrabarti [19] studied type-ahead search in spatial databases using materialization techniques. Cao et al. [2] studied collective keyword search by considering multiple points. Leung et al. [12] proposed to use locations for personalized search. Obviously the above queries substantially differ from our direction-aware spatial keyword query.

There are many studies on knn [18], [16], [10], [11], [20], [17]. Ferhatosmanoglu et al. [7] studied constrained nearest neighbor search using polygon as a constraint. Cheng et al. [4] studied constrained knn queries over uncertain data. Gao et al. [8] and Nutanong et al. [14] proposed to answer visible knn queries. Patroumpas et al. [15] studied the problem of monitoring object orientations. However their methods cannot support our problem as we support keyword-based search. We consider direction constraint which is different from theirs.

Although we can build two separate indexes, one for keywords and another for locations, this method is expensive, as it cannot simultaneously apply textual and spatial pruning.

VIII. Conclusion

In this paper we have studied the problem of direction-aware spatial keyword search. We find the k nearest neighbors to the query that contain all input keywords and satisfy the direction constraint. To efficiently answer a direction-aware spatial keyword query, we proposed novel indexing structures, which can prune large number of unnecessary POIs. We developed effective region-based pruning and direction-based pruning techniques to increase the search performance. We devised efficient algorithms to answer direction-aware spatial keyword queries. We also studied how to incrementally answer a query. We have implemented our algorithms, and experimental results show that our method achieves high performance and outperforms existing methods significantly.

IX. Acknowledgement

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions. This work was partly supported by the National Natural Science Foundation of China under Grant No. 61003004 and 60873065, National Grand Fundamental Research 973 Program of China under Grant No. 2011CB302206, National S&T Major Project of China under Grant No. 2011ZX01042-001-002, and "NExT Research Center" funded by MDA, Singapore, under Grant No. WBS:R-252-300-001-490.

References

[1] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *PVLDB*, 3(1):373–384, 2010.

[2] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD Conference*, pages 373–384, 2011.

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

[4] R. Cheng, J. Chen, M. F. Mokbel, and C.-Y. Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, pages 973–982, 2008.

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

[6] I. D. Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, 2008.

[7] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. E. Abbadi. Constrained nearest neighbor queries. In *SSTD*, pages 257–278, 2001.

[8] Y. Gao, B. Zheng, W.-C. Lee, and G. Chen. Continuous visible nearest neighbor queries. In *EDBT*, pages 144–155, 2009.

[9] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, 2007.

[10] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Trans. Database Syst.*, 1999.

[11] M. R. Kolahdouzan and C. Shahabi. Voronoi-based k nearest neighbor search for spatial network databases. In *VLDB*, pages 840–851, 2004.

[12] K. W.-T. Leung, D. L. Lee, and W.-C. Lee. Personalized web search with location preferences. In *ICDE*, pages 701–712, 2010.

[13] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD Conference*, pages 349–360, 2011.

[14] S. Nutanong, E. Tanin, and R. Zhang. Visible nearest neighbor queries. In *DASFAA*, pages 876–883, 2007.

[15] K. Patroumpas and T. K. Sellis. Monitoring orientation of moving objects around focal points. In *SSTD*, pages 228–246, 2009.

[16] S. Pramanik and J. Li. Fast approximate search algorithm for nearest neighbor queries in high dimensions. In *ICDE*, page 251, 1999.

[17] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2):93–104, 2010.

[18] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD Conference*, 1995.

[19] S. B. Roy and K. Chakrabarti. Location-aware type ahead search on spatial databases: semantics and efficiency. In *SIGMOD Conference*, pages 361–372, 2011.

[20] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[21] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552, 2011.

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

[23] 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*, 2009.

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

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