# Reachable Region Query and Its Applications

Mengdie Nie[a,c], Zhi-Jie Wang[a,c,d], Jian Yin[a,c,d], Bin Yao[b,c,e]

[a] *School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China*
[b] *Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China*
[c] *Guangdong Key Laboratory of Big Data Analysis and Processing, Guangzhou, China*
[d] *National Engineering Laboratory for Big Data Analysis and Applications, Beijing, China*
[e] *Guangdong Province Key Laboratory of Popular High Performance Computers of Shenzhen University, Shenzhen, China*

**Abstract**

In this paper we study the reachable region queries for objects moving in *obstacle space*. Loosely speaking, given any one of moving objects and many other information, we are asked to return the whole region in which each location has the non-zero probability to be reachable for the moving object. To solve this problem, this paper presents an efficient algorithm that runs in $O(n \log n)$ time and consumes $O(n \log n)$ space, where $n$ is the number of vertices of obstacles. Moreover, this paper establishes the lower bound for the proposed problem, showing that our solution essentially is optimal in terms of *random access memory* (RAM) model. Empirical study based on both real and synthetic datasets demonstrates the effectiveness and efficiency of our solution.

*Keywords:* Reachable region query, uncertain data, location-based service, moving target search, algorithms

## 1. Introduction

Consider a set $\mathcal{O}$ of *disjoint* obstacles and a set $\mathcal{M}$ of moving objects in a two-dimensional space $\mathbb{R}^2$. For any moving object $m \in \mathcal{M}$, it has a latest known location $s$, and it can move freely (within its maximum speed $V_{max}$) in $\mathbb{R}^2$ except that it cannot directly pass through any obstacle $o \in \mathcal{O}$. See Figure 1(a). Given any object $m \in \mathcal{M}$, its elapsed time $T_e$, and a point $p$ in $\mathbb{R}^2$, we say $p$ has the non-zero probability to be reachable for $m$ such that $\pi(s, p) \leq V_{max} \times T_e$, where $\pi(\cdot)$ denotes the Euclidean shortest path distance (i.e., obstructed distance). The *reachable region query* (RRQ) refers to that, given any object $m \in \mathcal{M}$, it is asked to return the reachable region $\mathcal{R}$ (of $m$), where $\mathcal{R}$ denotes the whole region in which each location has the non-zero probability to be reachable for the object $m$.

Often the latest known location $s$ might be not always constant, namely, $s$ might be updated later. See Figure 1(b). In this paper, we are *mainly* interested in algorithms without precomputation; notice that precomputing all (or some) reachable regions (or other entities) and updating them efficiently could be an independent and more interesting topic, we leave it as the future work. For simplicity of discussion, in the following we make an assumption that the obstacles are line-segments. This assumption is only for ease of exposition, and our algorithm can be easily extended to the case of polygonal obstacles, as shown in Section 5.

### *1.1. Applications*

Our study is motivated by the increasing popularity of *location-based service* [21] and *probabilistic processing on uncertain data* [20]. Nowadays, many mobile devices, such as cell phones, PDAs, taxis are equipped with GPS. It is common for the service provider to keep track of the user' s location, and provide many location-based services, for instance finding the taxies being located in a query range, finding the nearest supermarkets or ATMs, etc. The database server, however, is often impossible to contain the total status of an entity, due to the limited network bandwidth and limited battery power of the mobile devices [5]. This implies that the current location of an entity is uncertain before

*Email addresses:* niemengd@mail2@sysu.edu.cn (Mengdie Nie), wangzhij5@mail.sysu.edu.cn (Zhi-Jie Wang), issjyin@mail.sysu.edu.cn (Jian Yin), yaobin@cs.sjtu.edu.cn (Bin Yao)

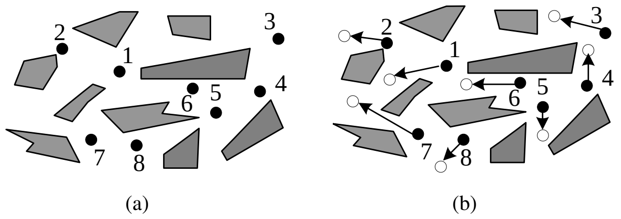


Figure 1: The settings of our problem. (a) Polygons denote obstacles, and black dots denote the latest known locations of moving objects. (b) The latest known locations might be updated later.

obtaining the next specific location. In this case, it is meaningful to use a closed region to represent the possible location of an entity [5, 22, 25]. In database community, most of results usually assumed this region is available beforehand, which is impractical. Although the recent work [28] considered the evaluation of such a region, it mainly assumed that the distance-based location update policy is adopted, in which the obstructed distance is not considered. A more justified manner to obtain such a region could be generally based on the following information: geographical information around the entity, say $G_I$, the (maximum) speed of the entity, and the elapsed time. By substituting $G_I$ with obstacles $\mathcal{O}$, it corresponds to our problem. We remark that, our reachable region is different from the uncertain region in [29], and the obstructed distance is different from other distance metrics such as *Mahalanobis distance* [19], *energy distance* [6] and *semantic distance* [30].

The RRQ problem can also find applications in the so-called *moving target search* [16, 24]. Traditionally, moving target search is the problem where a hunter has to catch a moving target, and they assumed the hunter always knows the current position of the moving target [24]. In many scenarios (e.g., when the power of GPS - equipped with the moving target- is used up, or in a sensor network environment, when the moving target walked out of the scope of being monitored), it is possible that the current position of the moving target cannot be obtained. In this case, we can infer the reachable region based on some available knowledge, such as the geographical information and the previous location. The reachable region here can contribute to the reduction of search range.

### *1.2. Related Work*

The RRQ problem is similar to many problems in computational geometry, e.g., *visibility polygon* problem [4], *visibility graph* problem [11], *art gallery* problem [3], *watchman route* problem [7], *shortest path* problem [26], and so forth. We refer the interested readers to recent works and textbooks [2, 4, 10, 26] for entry points into the literature. Among these problems, the ones most related to ours could be (1) the visibility polygon problem for a point in a polygon with holes, and (2) the shortest path problem in the presence of obstacles. For (1), optimal algorithms with $\Theta(n + h \log h)$ time can be found in [13], where $h$ is the number of holes. Correspondingly, for (2) the optimal algorithm with $O(n \log n)$ time is presented in [14].

On the other hand, our problem is also similar to many problems in spatial databases, e.g., *obstructed nearest neighbor queries* [9], *reasoning moving objects in symbolic indoor spaces* [15], *visible nearest neighbour queries* [18, 27], *group nearest neighbor queries with obstacles* [23], *range queries with obstacles* [31], and so forth. To solve these problems, various heuristic algorithms were developed. It is not hard to see that these problems and algorithms proposed in the literature are more or less different from ours.

Although our problem is easily stated and can find many applications, to the best of our knowledge, few attention has focused on this problem. To understand the challenges in-depth, it would be helpful to examine the traps into which a number of readers will likely fall. At the first glance, our problem seemingly can be solved using a surprisingly simple method — draw circles at the endpoints of obstacles using appropriate radii, and then merge these circles with a region that is reachable without the need of making a turn. Our analysis, however, breaks this delusion. See Figure 2 for some illustrations. Another intuitive method is, for each point located in a circle with the center $s$ and the radius $V_{max} \times T_e$, to check if it has the non-zero probability to be reachable; by collecting all the qualified points and merging them, it seemingly can obtain the result. This straightforward solution, however, is prohibitively expensive, as the number of points located in a closed region can be infinity.

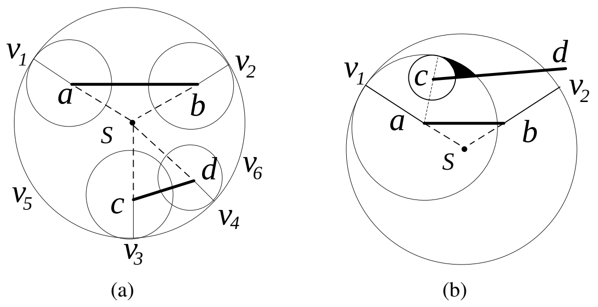


Figure 2: Illustrations of a simple but incorrect method. The radius of the biggest circle denotes $V_{max} \times T_e$ Segments $\overline{ab}$ and $\overline{cd}$ denote obstacles, other solid or dashed lines are the auxiliary lines. One can naturally get the reachable region for the left example, using this surprisingly simple method. However, such a method cannot get the correct result for the right example. See the black region, $m$ (with the starting point $s$) obviously cannot reach any point located in this region.

### *1.3. Our Contributions*

To solve the RRQ problem, we present a solution in which several novel notions, e.g., *control point*, *circular kernel region* (CKR) and *circular ordinary region* (COR), are employed. Our solution favourably relies on some insights into the nature of shortest path map (SPM) [14, 17], which was previously used to compute the Euclidean shortest path among polygonal obstacles. The key idea of our solution is to reduce the RRQ problem to computing the union of the CKR and a set of CORs. The fact above heavily relies on an insight formulated in Theorem 1 (see Section 3.2). It is challenging to prove Theorem 1 *directly*, our scheme is to reduce it to proving another intuitive version. In our proof, we utilize various classical tools, such as contradiction, and also several observations, e.g., any point in the ordinary region is visible to its control point. Besides, we establish the lower bound of the RRQ problem by reduction from the problem of sorting $n$ integers. In summary, our main contributions are as follows.

- We show the RRQ problem can be reduced to computing the union of the CKR and a set of CORs.
- We show the RRQ problem is solvable in $O(n \log n)$ time, using $O(n \log n)$ space.
- We show the lower bound of the RRQ problem is $\Omega(n \log n)$.
- We conduct experiments to demonstrate the effectiveness and efficiency of our solution.

In the remainder of the paper, Section 2 introduces the preliminaries. Sections 3 and 4 present our solution and prove the lower bound of the RRQ problem, respectively. Section 5 discusses the case of polygonal obstacles and other issues. Section 6 presents our experimental results and analyses. Finally, we conclude this paper in Section 7.

## 2. Preliminaries

Given two points $p$ and $p'$ in $\mathbb{R}^2$, we denote by $\triangleleft(p, p')$ they are visible to each other, by $\neg(\triangleleft(p, p')$ the opposite case, by $\pi(p, p')$ the shortest path distance between them, by $dist(p, p')$ the Euclidean distance between them when $\triangleleft(p, p')$, and by $\overline{pp'}$ the straight line segment joining the two points. Given any set, we use the notation "$|\cdot|$" to denote the cardinality of the set. $|\mathcal{O}|$ denotes the number of obstacles, for example. In addition, since $V_{max} \times T_e$ can be computed in $O(1)$ time, in the rest of the paper we use $l$ to denote $V_{max} \times T_e$ for short, and refer to it as the *maximum travel distance*.

**Definition 1.** (Shortest path map) *The shortest path map of a source point $s'$ with respect to a set $\mathcal{O}$ of obstacles is a decomposition of the free space $\mathbb{R}^2 \backslash \mathcal{O}$ ($\mathbb{R}^2$ minus the space occupied by $\mathcal{O}$) into regions, such that the shortest paths in the free space from $s'$ to any two points in the same region pass through the same sequence of obstacle vertices. See Figure 3.*

The shortest path map can store the shortest path distance $\pi(s', v)$ for all the vertices $v$ [17], and the map itself is usually stored using the *quad-edge data structure* [12]. Denote by $SPM(s')$ the shortest path map of the source point $s'$. It has the following properties.

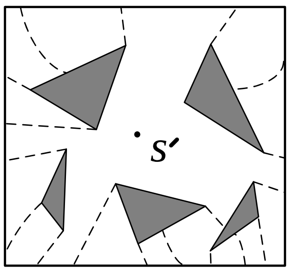


Figure 3: The $SPM(s')$ with respect to five polygonal obstacles.

**Lemma 1.** *Once the $SPM(s')$ is obtained, the map can be used to answer the single-source Euclidean shortest path query in $O(\log n)$ time [17].*

**Lemma 2.** *The map $SPM(s')$ has complexity $O(n)$, it consists of $O(n)$ vertices, edges, and faces. Each edge is a segment of a line or a hyperbola [17].*

**Lemma 3.** *Given a subdivision of a plane into more regions, to determine which of regions contains a query point $p$ can be finished in $O(\log n)$ time [8].*

An elegant algorithm to compute $SPM(s')$ among polygonal obstacles can be found in [14]. It has the following property.

**Lemma 4.** *Given a source point $s'$, and a set $\mathcal{O}$ of polygonal obstacles with a total number $n$ of vertices in $\mathbb{R}^2$, the $SPM(s')$ can be computed in $O(n \log n)$ time, consuming $O(n \log n)$ space.*

Remark that, from Lemma 4 one can have the immediate corollary below (since the line-segment obstacle can be viewed as the special or degenerate case of the polygonal obstacle — it has only 2 sides and no area)

**Lemma 5.** *Given the latest known location $s$, and a set $\mathcal{O}$ of $n$ line-segment obstacles in the plane (i.e., $\mathbb{R}^2$), the map $SPM(s)$ can be correctly constructed in $O(n \log n)$ time, using $O(n \log n)$ space.*

## 3. Our Solution

First, we describe our solution at a high level (Section 3.1), and then examine the central observation (Section 3.2), which serves as the cornerstone of our solution. Moreover, we detail the evaluation of major elements such as circular ordinary and kernel regions (Section 3.3). Finally, we analyze the time/space complexity of our solution (Section 3.4).

### *3.1. Solution Overview*

Our solution favourably relies on some new insights into the nature of *shortest path map*, which was previously used to compute the Euclidean shortest path among polygonal obstacles. To understand our solution, we need some preliminaries; so we start by introducing several notions.

Recall that, in Definition 1 we mentioned that for two different points $p$ and $p'$ in the same region of a shortest path map, their shortest paths pass through the same sequence of obstacle vertices. With this concept in mind, we define

**Definition 2.** (Control point) *Given a region of the shortest path map $SPM(s)$, and the sequence of obstacle vertices with regard to this region, we refer to the final obstacle vertex (among the sequence of obstacle vertices) as this region's control point, denoted by $e_c$.*

**Definition 3.** (Valid/invalid control point) *Given a control point $e_c$, we refer to it as a valid control point if $\pi(s, e_c) < l$; otherwise, we say it is an invalid control point.*

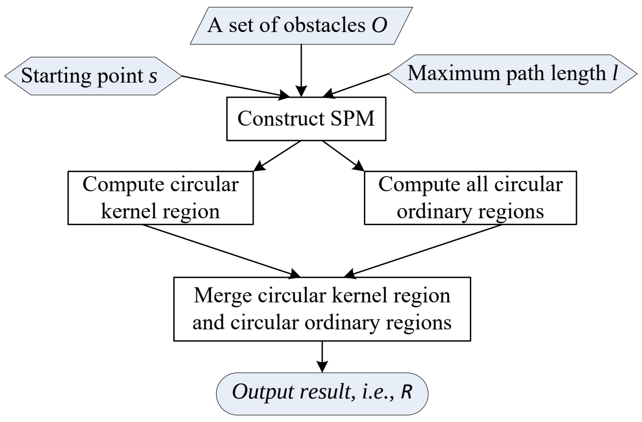

A set of obstacles $O$
Starting point $s$
Maximum path length $l$
Construct SPM
Compute circular kernel region
Compute all circular ordinary regions
Merge circular kernel region and circular ordinary regions
Output result, i.e., $R$

Figure 4: The work-flow of our solution.

**Definition 4.** (KERNEL-REGION, ORDINARY REGION) *Given the shortest path map $SPM(s)$, we refer to the region containing the latest known location $s$ as the kernel-region denoted by $\mathcal{R}_{map}(k)$, and any other region as the ordinary region denoted by $\mathcal{R}_{map}(o)$.*

**Definition 5.** (CIRCULAR KERNEL-REGION) *Given the kernel-region $\mathcal{R}_{map}(k)$ and the circle $C(s, l)$, we refer to their intersection set as the circular kernel-region denoted by $\mathcal{R}^*_{map}(k)$.*

**Definition 6.** (CIRCULAR ORDINARY REGION) *Given an ordinary region $\mathcal{R}_{map}(o)$, and assume that the control point $e_c$ of this region is an effective control point, we refer to the intersection set of this ordinary region and the circle $C(s, l - \pi(s, c_e))$ as the circular ordinary region denoted by $\mathcal{R}^*_{map}(o)$.*

For intuition, Figure 4 illustrates the workflow of our solution, while the pseudocodes of our solution are shown in Algorithm 1. In brief, it first assigns an empty set to $\mathcal{R}$, followed by building a shortest path map with regard to the latest known location $s$ and a set $\mathcal{O}$ of obstacles. Then, it attempts to obtain the circular kernel region $\mathcal{R}^*_{map}(k)$ and a set of circular ordinary regions (Lines 3-7). It finally gets the reachable region $\mathcal{R}$ by merging all these regions.

**Remark 1.** *The reasons we use the SPM technique is that, the reachable region query involves with a lot of computations related to obtruded distances and Euclidean shortest paths among points, while the SPM technique can efficiently manage obstructed distances and obtain the Euclidean shortest path distances among points easily.*

**Algorithm 1** RRQ

```
Input: $\mathcal{O}$, $s$, $l$
Output: $\mathcal{R}$
1: Set $\mathcal{R} = \emptyset$
2: Construct $SPM(s)$
3: Obtain $\mathcal{R}^*_{map}(k)$
4: for each ordinary region $\mathcal{R}_{map}(o)$ do
5:     Obtain the control point of $\mathcal{R}_{map}(o)$
6:     if it is a valid control point then
7:         Obtain $\mathcal{R}^*_{map}(o)$
8: Let $\mathcal{R} = \mathcal{R}^*_{map}(k) \bigcup_{\mathcal{R}^*_{map}(o) \in \Psi^*} \mathcal{R}^*_{map}(o)$ // i.e., merge all the regions obtained before
9: return $\mathcal{R}$
```

## 3.2. Reduction

The basic idea of our solution is to reduce our problem to computing the union of the circular kernel-region and a set of circular ordinary regions. Theorem 1 below exposes this fact.

**Theorem 1.** *Given $SPM(s)$ and the maximum travel distance $l$, assume, without loss of generality, that there are a set $\Psi$ of ordinary regions (among all the ordinary regions) such that each of these regions has the valid control point $e_c$ (i.e., $l - \pi(s, e_c) > 0$), implying that there are a number $|\Psi|$ of circular ordinary regions. Let $\Psi^*$ be the set of circular ordinary regions. Then, the reachable region $\mathcal{R}$ can be computed as $\mathcal{R} = \mathcal{R}^*_{map}(k) \bigcup_{\mathcal{R}^*_{map}(o) \in \Psi^*} \mathcal{R}^*_{map}(o)$.*

It is challenging to prove Theorem 1 directly, our scheme is reducing it to proving another claim (i.e., Lemma 6 below) holds. The intuition behind the reduction is that, the reachable region $\mathcal{R}$ strictly assures the following condition: "given any point $p$, if $\pi(s, p) > l$, then $p \notin \mathcal{R}$; otherwise, $p \in \mathcal{R}$". Clearly, if one can show that another region can also assure this condition, then it is equal to $\mathcal{R}$. In the following, we focus on the proof of Lemma 6.

**Lemma 6.** *Given $s$ and $l$, for any point $p$,*

- *if $\pi(s, p) > l$, then*
$$p \notin \mathcal{R}^*_{map}(k) \bigcup_{\mathcal{R}^*_{map}(o) \in \Psi^*} \mathcal{R}^*_{map}(o).$$
- *otherwise,*
$$p \in \mathcal{R}^*_{map}(k) \bigcup_{\mathcal{R}^*_{map}(o) \in \Psi^*} \mathcal{R}^*_{map}(o).$$

*Proof.* This lemma has two statements ("if" and "otherwise"), we prove each of them by *contradiction*.

• Case 1: assume that $\pi(s, p) > l$ but
$$p \in \mathcal{R}^*_{map}(k) \bigcup_{\mathcal{R}^*_{map}(o) \in \Psi^*} \mathcal{R}^*_{map}(o).$$
.

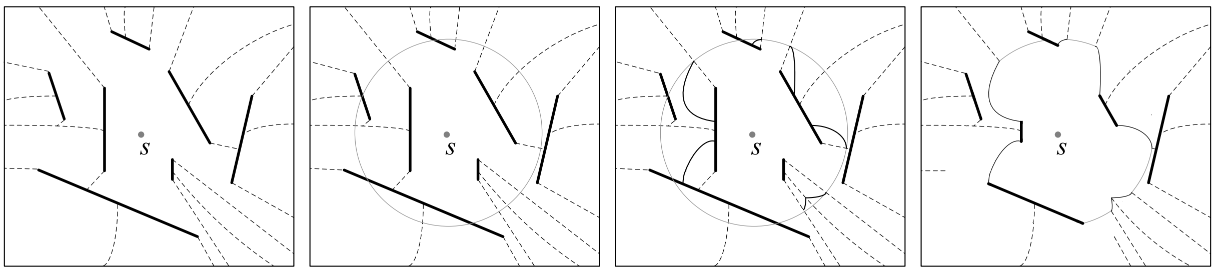
We first show it is contrary if $p \in \mathcal{R}^*_{map}(k)$, and then explain it is also contrary if $p \in \mathcal{R}^*_{map}(o)$. By Definition 5, we can easily know that
$$\mathcal{R}^*_{map}(k) = C(s, l) \cap \mathcal{R}_{map}(k).$$
See Figure 5(b) for an illustration. To show there does not exist a point $p \in \mathcal{R}^*_{map}(k)$ that satisfies $\pi(s, p) > l$, our key insight is that the kernel region $\mathcal{R}_{map}(k)$ has the following property:

**Proposition 1.** *The kernel region is a star-shaped polygon with respect to $s$, and any point in the polygon is visible to $s$.*

The claim above follows from Definition 1 and Lemma 2. See e.g., Figure 5(a) for an illustration. Since $\mathcal{R}^*_{map}(k)$ is a subset of $\mathcal{R}_{map}(k)$, naturally, we have
$$\forall_p (p \in \mathcal{R}^*_{map}(k) \rightarrow \lhd(s, p)).$$
Meanwhile, the radius of $C(s, l)$ assures that
$$\forall_p (p \in \mathcal{R}^*_{map}(k) \rightarrow dist(s, p) \leq l),$$
where $dist(s, p) = \pi(s, p)$, since $s$ and $p$ are visible to each other. Thus, in this case the assumption does not hold.

We proceed to explain it is also contrary if $p \in \mathcal{R}^*_{map}(o)$. By Definition 6, we can see that
$$\mathcal{R}^*_{map}(o) = \mathcal{R}_{map}(o) \cap C(e_c, l - \pi(s, e_c)),$$
where $e_c$ is an effective control point and $l - \pi(s, e_c) > 0$. To show any circular ordinary region does not contain such a point $p$ that satisfies $\pi(s, p) > l$, our key insight is as follows.

**Proposition 2.** *Any ordinary region is either a concave polygon or a convex polygon, but any point in the polygon is visible to its control point $e_c$.*



(a) (b) (c) (d)

Figure 5: Illustration of $SPM(s)$. (a) The kernel-region. (b) The circular kernel-region $\mathcal{R}^*_{map}(k)$. (c) The intersections among segments from $\mathcal{R}^*_{map}(k)$ and $\mathcal{R}^*_{map}(o)$ are less. (d) The merged result.

Similar to Proposition 1, the claim above follows from Definition 1 and Lemma 2. Again, see Figure 5(a) for an illustration. At the same time, the radius of $C(e_c, l - \pi(s, e_c))$ assures that

$$\forall_p (p \in \mathcal{R}^*_{map}(o) \rightarrow dist(e_c, p) \leq l - \pi(s, e_c)),$$

it implies that

$$\forall_p (p \in \mathcal{R}^*_{map}(o) \rightarrow \pi(s, p) \leq l),$$

since $\pi(s, p) = \pi(s, e_c) + dist(e_c, p)$. Thus, in this case the assumption does not hold, either. Combining the two results obtained just now, it completes the proof of Case 1.

• Case 2: Here we assume that $\pi(s, p) \leq l$ but

$$p \notin \mathcal{R}^*_{map}(k) \bigcup_{\mathcal{R}^*_{map}(o) \in \Psi^*} \mathcal{R}^*_{map}(o).$$

.

By Definition 4, we can see that for any point located in $SPM(s)$, it must be located in either the kernel region or an ordinary region, regardless of the distance from $s$ to this point. In the sequel, we first show it is contrary if $p \in \mathcal{R}_{map}(k)$ but $p \notin \mathcal{R}^*_{map}(k)$, and then expose it is also contrary if $p \in \mathcal{R}_{map}(o)$ but $p \notin \mathcal{R}^*_{map}(o)$.

Consider a point $p \in \mathcal{R}_{map}(k) \wedge p \notin \mathcal{R}^*_{map}(k)$. Intuitively, by Definition 5 and Proposition 1, $p$ shall be outside of the circle $C(s, l)$. By *analytic geometry*, we have that

$$dist(s, p) > l,$$

where $dist(s, p) = \pi(s, p)$, since $s$ and $p$ are visible to each other. Thus, in this case the assumption does not hold.

Similarly, consider a point $p \in \mathcal{R}_{map}(o) \wedge p \notin \mathcal{R}^*_{map}(o)$. By Definition 6 and Proposition 2, $p$ shall be outside of the circle $C(e_c, l - \pi(s, e_c))$, implying that

$$dist(e_c, p) > l - \pi(s, e_c).$$

That is,

$$dist(e_c, p) + \pi(s, e_c) > l,$$

where $dist(e_c, p) + \pi(s, e_c) = \pi(s, p)$, since $dist(e_c, p) = \pi(e_c, p)$ (by Proposition 2). Thus, in this case the assumption does not hold, either. Combining the two results obtained just now, it completes the proof of Case 2. To summarize, Lemma 6 follows. □

### 3.3. Evaluation of Circular Kernel (resp., Ordinary) Region

**Lemma 7.** *Given $SPM(s)$, $s$ and $l$, we have that (1) the circle kernel-region $\mathcal{R}^*_{map}(k)$ can be obtained in linear time, (2) the control point $e_c$ of any circular ordinary region can be obtained in logarithmic time, (3) to determine if $e_c$ is an effective control point can be done in constant time, and (4) to obtain the circular ordinary region $\mathcal{R}^*_{map}(o)$ can be done in constant time, provided that $e_c$ is an effective control point of $\mathcal{R}_{map}(o)$.*

*Proof.* We first prove the 1st claim. To obtain $\mathcal{R}^*_{map}(k)$, we can execute a *point-location query* on $SPM(s)$, using the point $s$ as the input data point; this can be done essentially in $O(\log n)$ time (by Lemma 3). In this way, we can immediately identify the region containing the point $s$, i.e., the so-called kernel region $\mathcal{R}_{map}(k)$. The edges of the kernel-region can be efficiently obtained in time *linear to* the number of edges of this region. This is attributed to the quad-edge data structure, recall Section 2. It is worth noting that $\mathcal{R}_{map}(k)$ can have $\Omega(n)$ edges, see e.g., Figure 5(a). With the boundaries of $\mathcal{R}_{map}(k)$, we can immediate get $\mathcal{R}^*_{map}(k)$ by computing the intersection set between $\mathcal{R}_{map}(k)$ and the circle $C(s, l)$, which takes $O(n)$ time. It stems directly from the fact that $\mathcal{R}_{map}(k)$ can have $\Omega(n)$ edges, and computing their intersections needs linear time. In summary, $\mathcal{R}^*_{map}(k)$ can be obtained in $O(n)$ time.

We are now ready to prove the 2nd claim. For any ordinary region $\mathcal{R}_{map}(o)$, if we want to find its control point $e_c$, we just need to randomly choose a point in the region, and execute a *single-source Euclidean shortest path query*, which typically can be done in $O(\log n)$ time (by Lemma 1). The final obstacle vertex on the path is just the so-called control point, justifying the 2nd claim.

To determine whether $e_c$ is an effective control point, it is equivalent to checking whether $\pi(s, e_c) < l$. Since $e_c \in \mathcal{E}$, and $SPM(s)$ can store the shortest path distances $\pi(s, e)$ for all the endpoints $e \in \mathcal{E}$ (recall Section 2), typically, we can obtain $\pi(s, e_c)$ in constant time (notice: the distances $\pi(s, e)$ for all the endpoints $e \in \mathcal{E}$ can be managed using an array). Once getting $\pi(s, e_c)$, we can finish, in $O(1)$ time, the comparison between $\pi(s, e_c)$ and $l$. Thus the 3rd claim holds.

The essence of obtaining the circular ordinary region $\mathcal{R}^*_{map}(o)$ is to compute the intersection set between the circle $C(e_c, l - \pi(s, e_c))$ and the ordinary region $\mathcal{R}_{map}(o)$. First of all, the circle is immediately available in $O(1)$ time, and the edges of $\mathcal{R}_{map}(o)$ can be also available in $O(1)$ time, since we used quad-edge data structure to store $SPM(s)$ and the number of edges of this region has the constant descriptive complexity (by Lemma 2). With $C(e_c, l - \pi(s, e_c))$ and the edges of $\mathcal{R}_{map}(o)$, we can immediately obtain $\mathcal{R}^*_{map}(o)$ in constant time. It stems from the fact that computing their intersections needs constant time. To summarize, Lemma 7 follows. □

### *3.4. Time/Space Complexity*

**Theorem 2.** *The running time of Algorithm 1 is $O(n \log n)$, and it consumes $O(n \log n)$ space.*

*Proof.* First, by Corollary 5 and Lemma 7, we can easily see that Lines 2 and 3 take $O(n \log n)$ and linear time, respectively. Within the **for** loop, by Lemma 7, we can see that Line 5 takes logarithmic time, both Lines 6 and 7 take constant time. In addition, by Lemma 2, the number of ordinary regions has $O(n)$ complexity. Thus, the overall execution time of the **for** loop is $O(n \log n)$. Finally, Line 8 is used to merge all these regions obtained before, which can be achieved by executing a simple boolean aggregated operation. Since the number of edges of all these regions has complexity $O(n)$, arranging all the segments of these regions and appealing to the algorithm in [1] can immediately produce their union in $O((n + i) \log n)$ time, where $i$ is the number of intersections among all these segments. Note that in the context of our concern, $i$ has the linear complexity, see e.g., Figure 5(c). Thus, merging them can be finished in $O(n \log n)$ time essentially; the merged result shall be a closed region bounded by line segments and circular arcs, see e.g., Figure 5(d). To summarize, the upper bound of Algorithm 1 is $O(n \log n)$. On the other hand, one can see that, obtaining the circular kernel-region $\mathcal{R}^*_{map}(k)$ and circular ordinary region $\mathcal{R}^*_{map}(o)$ requires $O(n)$ space, since $SPM(s)$ has complexity $O(n)$, i.e., it consists $O(n)$ vertices, edges, and faces (by Lemma 2). Similarly, merging all these regions obtained before takes also linear space, since we need $O(n)$ space to store polygons. In addition, by Corollary 5, we can see that constructing $SPM(s)$ requires $O(n \log n)$ space. Overall, this algorithm consumes asymptotic space $O(n \log n)$. In addition, its correctness follows directly from Theorem 1, Corollary 5 and Lemma 7, this completes the proof. □

## 4. Lower Bound

This section establishes the lower bound of our problem. We prove the lower bound by reduction from the problem of sorting $n$ integers $\{x_1, x_2, \ldots, x_n\}$. Let $x_{min}$ and $x_{max}$ be the smallest and largest numbers among $x_1$, $x_2$, ..., $x_n$, and let $\tau = |x_{max} - x_{min}|$, denoting the "range" of these $n$ numbers. We create an instance of our RRQ problem as follows.

We choose some number $\alpha$ in the interval $(0, 2\pi)$, and define $\theta_i = \frac{\alpha(x_i - x_{min})}{\tau}$, where $i \in \{1, 2, \ldots, n\}$. Note that, all $\theta_i$'s lie in the semi-open interval $[0, 2\pi)$. Also, let $\epsilon$ be an arbitrary number in the interval (0,1), we define $\theta'_i = \frac{\alpha(x_i + \epsilon - x_{min})}{\tau}$, where $i \in \{1, 2, \ldots, n\}$. With each number $x_i$, we now associate a line-segment, denoted by $s_i$, whose two endpoints are on the unit circle having polar coordinates $(1, \theta_i)$ and $(1, \theta'_i)$ respectively (known as the *first* and *second* endpoints of $s_i$). Figure 6 illustrates the construction for $n = 4$.

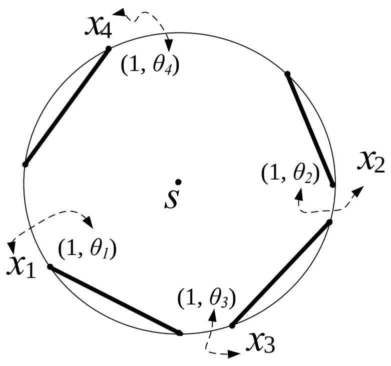


Figure 6: Illustration of lower bound proof for the RRQ problem.

The starting point $s$ is set to the center of the unit circle, and the maximum travel distance $l$ (i.e., $V_{max} \times T_e$ ) is set to the radius of the unit circle. It can be easily shown that, the first endpoint of each line-segment $s_i$ appears regularly on the boundaries of the reachable region in order of increasing values of $\theta_i$'s. It is therefore easy to extract, in $O(n)$ time, the number $n$ of "sorted $\theta_i$'s" from the boundaries of the reachable region. Moreover, in $O(n)$ time these sorted $\theta_i$'s can be mapped backed to the corresponding sorted $x_i$'s. Let $RRQ(n)$ be the time required for the reachable region query, and $S(n)$ be the time required for sorting $x_1, x_2, \ldots, x_n$. Based on the reduction above, we have $S(n) \leq RRQ(n) + O(n) + O(n)$. That is, $RRQ(n) \geq S(n) - O(2n)$. Furthermore, it is well-known that sorting $n$ integers in the general algebraic decision tree requires $\Omega(n \log n)$ time. Putting all together, it yields

**Theorem 3.** *The worst case lower bound of the RRQ problem is* $\Omega(n \log n)$.

## 5. Discussion

For the case of polygonal obstacles, the RRQ problem can be easily achieved. In brief, we also construct a shortest path map $SPM(s)$. Note that here we can construct $SPM(s)$ by directly appealing to the method in [14], since we are processing the case of polygonal obstacles. Clearly, by Lemma 4, this step can be done in $O(n \log n)$ time using $O(n \log n)$ space. Then we obtain the circular kernel-region $\mathcal{R}^*_{map}(k)$ and the circular ordinary regions $\mathcal{R}^*_{map}(o)$. The methods to compute these regions are same to the ones discussed in Section 3.3, except that the control points here are from the vertices of polygonal obstacles. So, by Lemma 7, obtaining all these regions can be done in $O(n \log n)$ time, consuming $O(n)$ space. We finally merge all these regions using the method discussed in Section 3.4, getting the reachable region $\mathcal{R}$. This step still takes $O(n \log n)$ time and requires $O(n)$ space, since the number of all edges of these regions has also $O(n)$ complexity. Thus, one can have the following immediate corollary.

**Corollary 1.** *Given a set $\mathcal{O}$ of disjoint polygonal obstacles with n vertices, we can answer the reachable region query in $O(n \log n)$ time using $O(n \log n)$ space.*

Note that, the lower bound of the reachable region query for polygonal obstacles with $n$ vertices is also $\Omega(n \log n)$. The proof is immediate by a simple adaptation to the proof of Theorem 3. For example, let $\epsilon$ be an arbitrary number in the interval (0,0.5); $\theta_i = \frac{\alpha(x_i - x_{min})}{\tau}$, $\theta'_i = \frac{\alpha(x_i + \epsilon - x_{min})}{\tau}$, $\theta^*_i = \frac{\alpha(x_i - \epsilon - x_{min})}{\tau}$ (notice: if $\theta^*_i < 0$, we can replace it by $\theta^*_i + 2\pi$); with each $x_i$, we associate an isosceles triangle whose vertices are $(\theta_i, 0.5)$, $(\theta'_i, 1)$, $(\theta^*_i, 1)$; and we also extract the sorted $\theta_i$'s from the boundaries of the reachable region, and map them back to the sorted $x_i$'s.

**Remark 2.** *Notice that the object may locate at different positions with different probabilities. In the usual case (especially when no much prior information is available), one may assume the possible locations follow random distribution in the reachable region. On the contrary, when there are much more prior information, the probability distribution could be calculated (or inferred) off-line. Notice that, computing the probability distribution is another interesting yet challenging problem. Such a topic may belong to the branch of statistics; it is beyond the topic of this paper.*

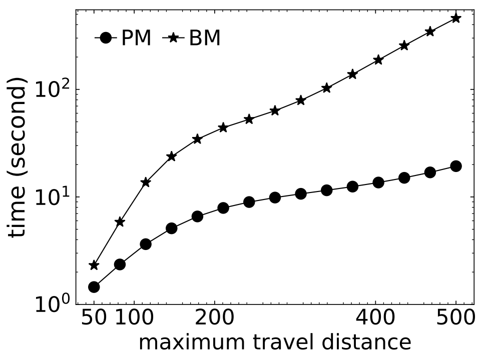


Figure 7: Runtime on the synthetic dataset.

## 6. Performance Evaluation

### *6.1. Experiment Setup*

Our experiments are based on both real and synthetic datasets, and the size of 2D space is fixed to 10000×10000. Two real datasets, called CA and LB, are deployed. The CA is available in site: http://www.usgs.gov/, and the LB is available in site: http://www.rtreeportal.org/. The CA contains 104770 2D points, the LB contains 53145 2D rectangles. We let the CA denote the latest known locations of moving objects, and the LB denote the obstacles in 2D space. All datasets are normalized in order to fit the 10000×10000 2D space. Synthetic datasets also consist of two types of data. We generate a set of polygons to denote the obstacles, and place them in this space uniformly. We generate a set of points to denote the latest known locations of moving objects, and let them randomly distributed in this space (note: there is a constraint that these points cannot be located in the interior of any polygonal obstacles). We remark that, as for the real dataset, once this constraint is employed, the number of effective 2D points in the CA is essentially 101871. Furthermore, since some rectangles in the LB are degenerated to line segments, or they are not disjoint, the number of effective rectangles in the LB is essentially 12765 after we remove those unqualified rectangles. For brevity, we use the CL and RU to denote the real (California points together with Long Beach rectangles) and synthetic (Random distributed points together with Uniform distributed polygons) datasets, respectively.

Since there is no available algorithms "directly" working for the reachable region query problem, we implement a baseline method that is based on the *visibility graph* [11] technique. This method generally works as follows. It constructs a visibility graph $\mathcal{G}$ for obstacle vertices and stores the shortest path distances from $s$ to all endpoints in $\mathcal{G}$. For each endpoint $e$ whose shortest path distance $\pi(s, e)$ is less than $l$, it computes a "circular" visibility region, which is the intersection set between the *visible region* [4] of $e$ and a circle whose radius and center are $l - \pi(s, e)$ and $e$, respectively. Meanwhile, it also computes the circular visibility region for $s$, using the radius $l$. It finally merges all these "circular" visibility regions, getting the reachable region. This baseline method takes $O(n^2 \log n)$ time, since the dominant step is to construct the "circular" visibility region, which takes $O(n \log n)$ time for a single endpoint. The space complexity is $O(n^2)$, since the dominant space is in the merging phase, in which the space is mainly for storing $O(n)$ polygons with linear size.

We evaluate the performance of our solution based on two metrics: (i) the execute time for reachable region query; and (ii) the memory cost used for computing the reachable region query. To study the time and space cost, in our experiments we randomly choose 100 moving objects, and run 10 times for each object, and then compute the average query and space cost. In our experiments, the number of edges in the polygonal obstacle is set to 4, the size is $40 \times 40$. The maximum path lengths of moving objects are set to $[50, 100, 200, 400, 500]$ for different groups of tests. In the implementation of our algorithm and of the baseline method, we use the maximum path length to remove obstacles that are obviously unqualified. That is, we remove those obstacles that are outside of the circle with the radius $l$ and the center $s$. In our experiments, both the proposed algorithm and the compared algorithm are written in C++. All of our experiments are conducted on a computer with Intel i7-8700 CPU@3.2GHz and 16GB of memory.

### *6.2. Experimental Results for RU Dataset*

Figure 7 shows the runtime of our method (i.e, PM) and the baseline method (i.e., BM) on the synthetic dataset (i.e., RU). From this figure, it can be seen that, with the increase of the maximum travel distance $l$, the runtime for both our method and the baseline grows. Nevertheless, at all cases (*l* from 50 to 500) our method is superior to the baseline, demonstrating the effectiveness of our method. On the other hand, we can see from these two curves that the growth speed of the runtime of our method is obviously slower than that of the baseline. This essentially demonstrates that our method has a better scalability. Notice that, when one increases the parameter *l*, it essentially increases the number of obstacles to be processed. Naturally, a visibility graph (resp., shortest path map) with more edges needs to be constructed, and so more circular visibility regions (resp., circular ordinary regions) need to be computed. This is why the runtime increases for both methods when *l* increases.

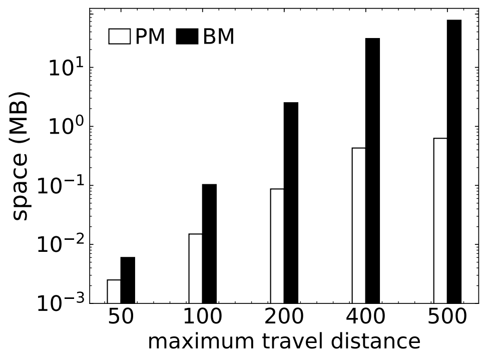


Figure 8: Space cost on the synthetic dataset.

Correspondingly, Figure 8 shows the space cost when computing the reachable region. We can easily see that, the larger *l* is, the more the space is used. This is mainly because there are more segments needing to be stored in the process of computation, when *l* is larger. Notice that, as for the BM, the segments may be from the visibility graph, circular visibility regions, some immediate results when merging these regions, and so on. Similarly, as for the PM, the segments may be from the shortest path map, circular ordinary region, and so on. An interesting finding is that, when *l* is small, the space cost for these two methods is close, while the gap turns obvious when *l* is relatively large. This is mainly because the number of obstacles to be considered in the computation is very small when *l* is small (in this case, both methods need to store less segments, and the gap is bounded by the cost of the baseline). Essentially, this phenomena also exists for the runtime results (see Figure 7 again), and the underlying reasons are basically similar.

## *6.3. Experimental Results for CL Dataset*

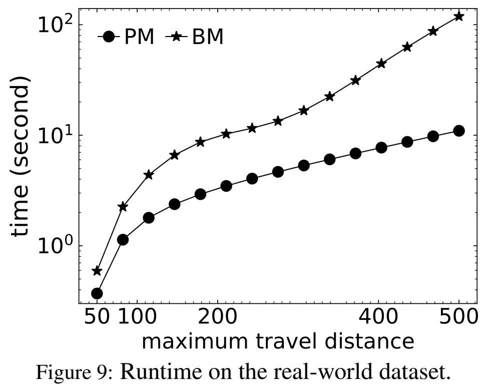


Figure 9: Runtime on the real-world dataset.

As mentioned in Section 6.1, besides the synthetic dataset (i.e., RU), we also test our method and the baseline based on the real-world dataset (i.e., CL). Figure 9 shows the runtime of these two methods. Generally, as for the real world dataset, our method (i.e, PM) is also more efficient than the baseline (i.e., BM), and the performance gap is more obvious when *l* is large. For example, when *l* = 500, our method outperforms the baseline by an order of magnitude. The reasons are basically the same to our previous analysis. Moreover, by comparing Figures 7 and 9, one can easily find that, given the same *l*, the runtime on the RU dataset is longer that on the CL dataset. This is mainly because the size of the real world dataset CL is smaller than the synthetic dataset RU. More specifically, the number of obstacles in CL is less than that in RU. Naturally, for an query object with the maximum path length *l*, the computation shall take a much longer time for the RU dataset (since more qualified obstacles may need to be considered in the computation).

Figure 10 shows the space cost of our method and the baseline on the real world dataset (i.e., CL). Overall, this set of experimental results are similar to that in Figure 8. On one hand, the space cost of our method is no more than that of the baseline. On the other hand, the performance gap is much more obvious when *l* is large. For example, when $l = 500$, the space cost of our method outperforms the baseline by 1-2 orders of magnitude. In addition, by comparing Figures 8 and 10, we can also find that, for the same *l*, the space cost of both methods on the CL dataset is obviously less than that on the RU dataset. The main reason is the same to our previous analysis.

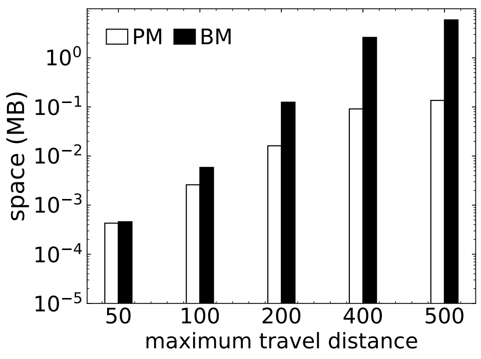


Figure 10: Space cost on the real-world dataset.

## 7. Conclusion

In this paper, we investigated the problem of reachable region query, and presented an algorithm that runs $O(n \log n)$ time and uses $O(n \log n)$ space. Experimental results based on both real and synthetic datasets demonstrate the efficiency and effectiveness of our method. As mentioned in Section 1, precomputing all (or some) reachable regions (or other entities) and updating them efficiently could be an independent and more interesting topic. In the future, we would like to study this open problem.

### Acknowledgments

This work was supported by the National Key R&D Program of China (2018YFB1004400), the NSFC (61472453, U1401256, U1501252, U1611264, U1711261,U1711262). Bin Yao was supported by the NSFC (61872235, 61729202, 61832017 and U1636210), Guangdong Key Laboratory of Big Data Analysis and Processing(201808), and Guangdong Province Key Laboratory of Popular High Performance Computers of Shenzhen University (SZU-GDPHPCL2017).

## References

[1] E. Berberich, A. Eigenwillig, M. Hemmer, S. Hert, K. Mehlhorn, E. Schmer, A computational basis for conic arcs and boolean operations on conic polygons, in: European Symposium on Algorithm (ESA), 2002, pp. 174–186.

[2] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational geometry: algorithms and applications, Third Edition, Springer, Berlin, 2008.

[3] Édouard Bonnet, T. Miltzow, An approximation algorithm for the art gallery problem, in: Symposium on Computational Geometry (SoCG), 2017, pp. 1–20.

[4] D.Z. Chen, H. Wang, Computing the visibility polygon of an island in a polygonal domain, Algorithmica 77 (2017) 40–64.

[5] R. Cheng, D.V. Kalashnikov, S. Prabhakar, Querying imprecise data in moving object environments, IEEE Transactions on Knowledge and Data Engineering (TKDE) 16 (2004) 1112–1127.

[6] W. Chu, J. Liu, Subband energy distance measure applied in multi-pass speech/non-speech discrimination, in: ISSPA, pp. 1–4.

[7] A. Dumitrescu, J.S.B. Mitchell, P. Zylinski, Watchman routes for lines and line segments, Computational Geometry: Theory and Applications 47 (2014) 527–538.

[8] H. Edelsbrunner, L.J. Guibas, J. Stolfi, Optimal point location in a monotone subdivision, SIAM Journal on Computing (SIAMCOMP) 15 (1986) 317–340.

[9] Y. Gao, B. Zheng, G. Chen, C. Chen, Q. Li, Continuous nearest-neighbor search in the presence of obstacles, ACM Transactions on Database Systems (TODS) 36 (2011) 9.

[10] S.K. Ghosh, Visibility algorithms in the plane, Cambridge University Press, New York, 2007.

[11] S.K. Ghosh, B. Roy, Some results on point visibility graphs, Theoretical Computer Science 575 (2015) 17–32.

[12] L.J. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and computation of voronoi diagrams, ACM Transactions on Graphics (TOG) 4 (1985) 74–123.

[13] P.J. Heffernan, J.S.B. Mitchell, An optimal algorithm for computing visibility in the plane, SIAM Journal on Computing (SIAMCOMP) 24 (1995) 184–201.

[14] J. Hershberger, S. Suri, An optimal algorithm for euclidean shortest paths in the plane, SIAM Journal on Computing (SIAMCOMP) 28 (1999) 2215–2256.

[15] S.H. Hussein, H. Lu, T.B. Pedersen, Reasoning about rfid-tracked moving objects in symbolic indoor spaces, in: International Conference on Scientific and Statistical Database Management (SSDBM), 2013, pp. 1–9.

[16] A. Kolling, A. Kleiner, M. Lewis, K.P. Sycara, Computing and executing strategies for moving target search, in: IEEE International Conference on Robotics and Automation (ICRA), 2011, pp. 4246–4253.

[17] J.S.B. Mitchell, Shortest paths among obstacles in the plane, in: ACM Symposium on Computational Geometry (SoCG), 1993, pp. 308–317.

[18] S. Nutanong, E. Tanin, R. Zhang, Incremental evaluation of visible nearest neighbor queries, IEEE Transactions on Knowledge and Data Engineering (TKDE) 22 (2010) 665–681.

[19] A. Pathak, A. Ramesh, A. Mitra, K. Majumdar, Automatic seizure detection by modified line length and mahalanobis distance function, Biomed. Signal Proc. and Control 44 (2018) 279–287.

[20] J. Pei, M. Hua, Y. Tao, X. Lin, Query answering techniques on uncertain and probabilistic data: tutorial summary, in: ACM International Conference on Management of Data (SIGMOD), 2008, pp. 1357–1364.

[21] J. Schiller, A. Voisard, Location-Based Services, Morgan Kaufmann Publishers, San Francisco, 2004.

[22] A.P. Sistla, O. Wolfson, S. Chamberlain, S. Dao, Modeling and querying moving objects, in: IEEE International Conference on Data Engineering (ICDE), 1997, pp. 422–432.

[23] N. Sultana, T. Hashem, L. Kulik, Group nearest neighbor queries in the presence of obstacles, in: ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL/GIS), 2014, pp. 481–484.

[24] X. Sun, W. Yeoh, S. Koenig, Efficient incremental search for moving target search, in: International Joint Conference on Artificial Intelligence(IJCAI), 2009, pp. 615–620.

[25] Y. Tao, X. Xiao, R. Cheng, Range search on multidimensional uncertain data, ACM Transactions on Database Systems (TODS) 32 (2007) 15.

[26] H. Wang, Bicriteria rectilinear shortest paths among rectilinear obstacles in the plane, in: Symposium on Computational Geometry (SoCG), 2017, pp. 1–16.

[27] Y. Wang, R. Zhang, C. Xu, J. Qi, Y. Gu, G. Yu, Continuous visible k nearest neighbor query on moving objects, Information Systems (IS) 44 (2014) 1–21.

[28] Z.J. Wang, D.H. Wang, B. Yao, M. Guo, Probabilistic range query over uncertain moving objects in constrained two-dimensional space, IEEE Transactions on Knowledge and Data Engineering (TKDE) 27 (2015) 866–879.

[29] H. Wu, Z. Miao, Y. Wang, J. Chen, C. Ma, T. Zhou, Image completion with multi-image based on entropy reduction, Neurocomputing 159 (2015) 157–171.

[30] H. Wu, Z. Miao, Y. Wang, M. Lin, Optimized recognition with few instances based on semantic distance, The Visual Computer 31 (2015) 367–375.

[31] J. Zhang, D. Papadias, K. Mouratidis, M. Zhu, Spatial queries in the presence of obstacles, in: International Conference on Extending Database Technology (EDBT), 2004, pp. 366–384.